European Journal of Operational Research 262 (2017) 550–562

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing and Logistics

Parts-to-picker based order processing in a rack-moving mobile robots environment

Nils Boysen[a,1], Dirk Briskorn[b,2,*], Simon Emde[c,3]

[a] Friedrich-Schiller-Universität Jena, Lehrstuhl für Operations Management, Carl-Zeiß-Straße 3, Jena 07743, Germany
[b] Bergische Universität Wuppertal, Professur für BWL, insbesondere Produktion und Logistik, Rainer-Gruenter-Str. 21, 42119 Wuppertal, Germany
[c] Technische Universität Darmstadt, Fachgebiet Management Science / Operations Research, Hochschulstraße 1, Darmstadt 64289, Germany

ARTICLE INFO

Article history:
Received 3 June 2016
Accepted 20 March 2017
Available online 25 March 2017

Keywords:
Freight/material handling
Warehousing
Scheduling
Mobile robots

ABSTRACT

This paper treats a special parts-to-picker based order processing system, where mobile robots hoist racks and bring them directly to stationary pickers. This technological innovation – known as the Kiva system – heavily influences all traditional planning problems to be solved when operating a warehouse. We, specifically, tackle the order processing in a picking station, i.e., the batching and sequencing of picking orders and the interdependent sequencing of the racks brought to a station. We formalize the resulting decision problem and provide suited solution procedures. In a comprehensive computational study we show that an optimized order picking allows to more than halve the fleet of robots compared to simple decision rules often applied in real-world warehouses.

1. Introduction

In a traditional picker-to-parts based order picking environment, human pickers (with or without a supporting vehicle) successively visit the shelves where the parts defined by a specific pick list are stored in. Obviously, the travel of pickers is an unproductive part of the order picking process, so that it is not astounding that the parts-to-picker paradigm has attracted plenty of technical innovations such as carousel racks (e.g., (Van den Berg, 1996)), vertical lift modules (e.g., (Meller & Klote, 2004)), and automated storage/and retrieval systems (ASRS, see (Roodbergen & Vis, 2009)) over the past decades. The basic aim of these systems is to automatically move the stock keeping units (SKUs) to the pickers, so that they can concentrate on the productive part of their employment – the picking and packing of orders.

When choosing between both paradigms warehouse managers face the basic trade-off, which is schematically depicted by the radar plot of Fig. 1. For a more detailed discussion of both systems see, e.g., the in-depth survey paper of de Koster, Le-Duc, and Roodbergen (2007). On the one hand, unproductive travel times reduce productivity per picker, so that picker-to-parts systems require a larger workforce for realizing the same output compared to parts-to-picker systems. Note that in Fig. 1 this worse evaluation is represented by picker-to-parts systems being located closer to the radar plot's point of origin with regard to the criteria 'personnel costs' and 'picks per hour'. On the other hand, manual order picking can be accomplished without a large investment in hardware, because it relies on simple equipment such as standard racks, picking vehicles, and forklifts. This standardized equipment makes picker-to-parts systems easily adaptable to varying capacity situations, because it can simply be added to and removed from a distribution center. Within Fig. 1 this reverts the evaluation of picker-to-parts systems compared to parts-to-picker systems. The former reduce investment costs and are better scalable, so that they are located farther from the radar plot's origin. It can be concluded, that the advantages of picker-to-parts systems constitute the disadvantages of parts-to-picker systems and vice versa.

In this paper we treat a special parts-to-picker system, which aims to avoid the basic drawbacks of these two systems. As a parts-to-picker system it avoids unproductive picker travel and, thus, high personnel costs. At the same time it aims at reducing idle time of specialized and expensive equipment.

1.1. A rack-moving mobile robot based order picking system

The specific parts-to-picker system treated in this paper is based on quite simple mobile robots, which are able to lift a rack and transport them directly to a stationary picker. Alternative descriptions of this system, which is known as the Kiva system (e.g.,

* Corresponding author.
E-mail address: briskorn@uni-wuppertal.de (D. Briskorn).
[1] http://www.om.uni-jena.de/nils.boysen@uni-jena.de.
[2] http://www.prodlog.uni-wuppertal.de/briskorn@wiwi.uni-wuppertal.de.
[3] http://www.or.wi.tu-darmstadt.de/emde@bwl.tu-darmstadt.de.

http://dx.doi.org/10.1016/j.ejor.2017.03.053
0377-2217/

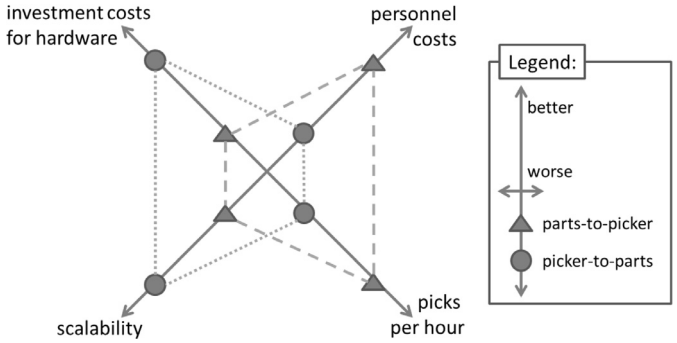


Fig. 1. Pros and cons for picker-to-parts and parts-to-picker order picking.

(Mountz, 2005)), are provided by Guizzo (2008) and WurWorking Paperman, D'Andrea, and Mountz (2008). Similar rack-moving mobile robot systems developed by competitors are introduced by Kirks, Stenzel, Kamagaew, and ten Hompel (2012); one of them is the so-called G-Com-System of the German Grenzebach group.

The mobile robot, which is schematically depicted in Fig. 2(a), consists of a driving and a lifting unit. The driving unit has a rotation mechanism and is electrically powered. For its orientation the shop floor is subdivided into a grid and each square of the grid is labeled with a bar code. The robot's integrated camera system continuously reads these bar codes to locate itself and the rotation mechanism allows the robot to move in the grid analogously to the rook on a chessboard. The lifting unit is able to lift more than 1,000 kilograms, so that it can drive directly under a rack, which are about man-high, lift it, and bring it directly to where the SKUs stored in it are required. Clearly, a distribution center requires many of these robots. For instance, office supply retailer Staples uses about 500 Kiva robots at its 30,000-square-meter distribution center in Chambersburg (USA) (e.g., (Guizzo, 2008)).

The mobile robot based workflow along the elementary system elements is schematically depicted in Fig. 2(b). First, SKUs arriving at the distribution center are received (step (1)). For this purpose, robots move racks with empty storage locations to the rack loading area where the SKUs are put inside the shelves. Then, the robots move the racks into the storage area where they wait for their employment. Once an SKU stored in a specific rack is required, a robot moves to the rack, hoists it, and brings it to a picking station, where a stationary picker assembles picking orders (step (2)). Here, the picker retrieves those SKUs defined on the current picking list, packs them it a cardboard box, and moves the completed cardboard box on a conveyor belt. The conveyor transports the shipment towards the outbound area (step (3)), where the box is packed onto an outbound truck, e.g., of a postal service provider, which moves the order onwards to the customer (step (4)).

This paper specifically addresses the order processing in a picking station, whose typical layout is schematically depicted in Fig. 2(c). Each station is operated by a stationary picker, who, first, receives a picking list from the information system, which defines the picking orders to be assembled. According to the processing sequence defined by the list, the picker folds cardboard boxes, identifies them with a bar code sticker, and puts them on the workbench. Typically, multiple boxes are filled in parallel. Then, robots bring racks to the station and queue in front of it. Some pick-to-light system, e.g., a laser pointer, may support the identification of the right SKUs, so that the picker can retrieve items from the current rack being first in the queue, scan the SKU, and put them into the right cardboard boxes. Once all items are retrieved from the current rack, robot and rack depart and the next rack in the queue moves forward. Whenever a picking order is complete its cardboard box is sealed and put on the conveyor belt, which moves the order towards the outbound area. Finally, the empty place on the workbench is filled with a new cardboard box for the next order.

Note that the overall workflow in the warehouse and in a specific picking station may, of course, slightly vary in different real-world applications. For instance, distribution centers can do without the conveyor belt and apply the mobile robots for supplying the outbound area instead (Guizzo, 2008).

1.2. Decision problems and literature review

The basic decisions to be made during the daily processes when operating a mobile robot based warehouse are of course basically the same as in traditional warehouses (e.g., (Gu, Goetschalckx, & McGinnis, 2010; de Koster et al., 2007)). However, the altered workflow requires modifications of these operational decision problems:

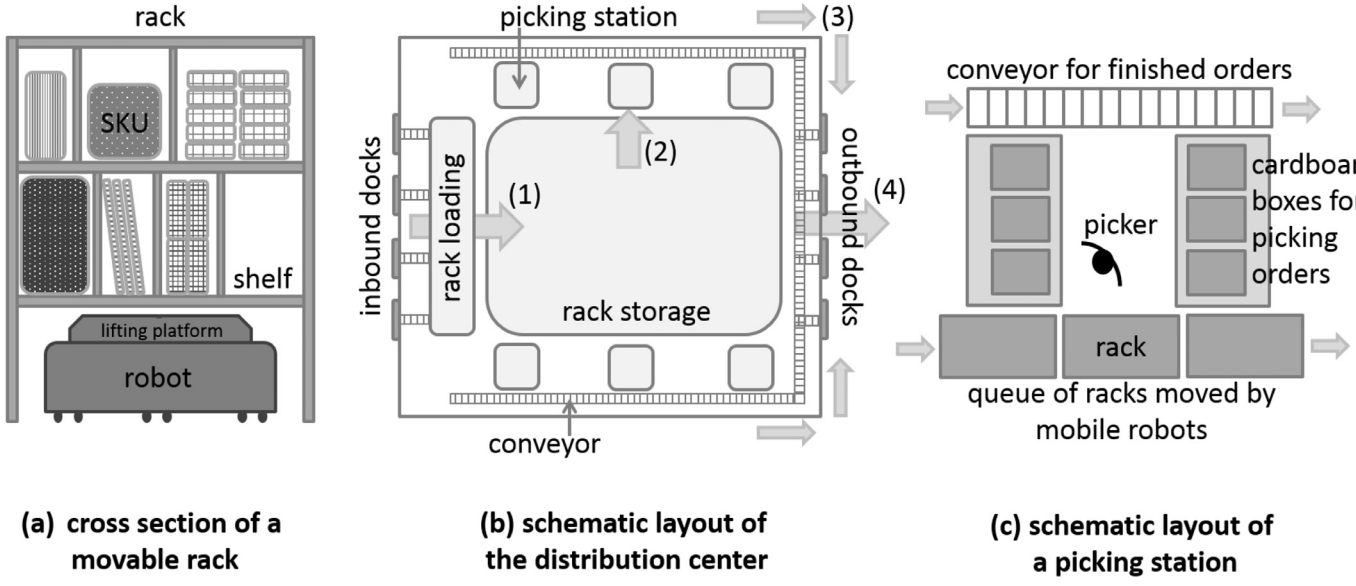


Fig. 2. The rack-moving mobile robots system.

- Directly after their arrival, SKUs need to be put away into racks. In doing so, it is not the specific rack that is crucial (because they do not differ and can alter their positions), but with what other SKUs they are stored together. Typically, a shared storage policy is applied (e.g., (Bartholdi & Hackman, 2016)), which means that SKUs of the same kind are not stored together in a unique rack, but are spread over multiple racks. This way a specific picking order can be satisfied by multiple racks, so that there is a greater probability to always have some rack close-by that carries a specific item. In such a setting, it seems advantageous to store those SKUs together in one rack, which are often ordered together by the same customer (see, e.g., (Frazelle & Sharp, 1989)).
- Then, the racks need to be assigned a location in the rack storage area. As in traditional warehouses, where this problem is known as the storage assignment problem (de Koster et al., 2007), it seems advantageous to classify products according to their turnover, e.g., by applying the famous cube-per-order index of Heskett (1963), so that fast-moving items can be stored closer to the picking stations. The great advantage of the mobile robot system is that racks can continuously be relocated once their demand frequency alters.
- The order processing in a picking station is the focus of this paper. At a picking station, the orders' arrivals are to be sequenced resulting in sets of orders being processed simultaneously. The respective cardboard boxes are jointly placed on the workbench, and the sequence of their processing needs to be determined. This decision is closely coupled to the arrival sequence of the racks, because the better the content of the racks fits to the SKUs demanded by the batch of current picking orders, the sooner the orders are readily assembled. The huge research effort on batching and order sequencing in traditional warehouses is, for instance, summarized by the surveys of de Koster et al. (2007) and Gu et al. (2010). However, the facultative arrival sequence of racks completely alters the problem, so that previous research seems not directly applicable.
- Finally, the traffic planning and management has to coordinate the mobile robots with all their different destinations. This coordination of multiple agents has attracted most research on rack-moving mobile robots (see, e.g., (D'Andrea & Wurman, 2008; Herrero-Pérez & Martínez-Barberá, 2011; Wurman et al., 2008; Yu, 2016)). The overall research effort on planning and managing automated guided vehicles is summarized by the survey paper of Vis (2006).

Beyond the operational decision problems, Lamballais, Roy, and De Koster (2017) approximate the order throughput of a mobile robot based warehouse with the help of queuing networks. This way, long term decision tasks, such as determining the layout of the rack storage area, can quickly be evaluated. However, our operational problem, i.e., synchronizing the processing of picking orders and racks in mobile robot based warehouses, has yet not be considered in scientific literature.

1.3. Contribution and paper structure

This paper treats the processing of picking orders in a picking station supplied with racks by rack-moving mobile robots. We formalize the resulting decision problem, i.e., we determine the batches of picking orders, their processing sequence, and the arrival sequence of the racks delivered by the mobile robots. In addition to the analysis of computational complexity we provide suited solution procedures. Furthermore, we investigate managerial aspects such as the impact of the shared storage policy. Specifically, we show that an optimized order picking allows to more than halve the fleet of robots compared to simple decision rules often applied in real-world warehouses. Also the diversity of SKUs stored in the racks is shown to greatly impact picking performance and the fleet size.

The remainder of the paper is structured as follows. Section 2 describes our problem in detail and Section 3 introduces heuristic decomposition procedures, which iteratively solve the order sequencing and rack sequencing subproblem, respectively. Our computational study, which tests the performance of our solution procedures and applies them in order to explore managerial aspects, is presented in Section 4. Finally, Section 5 concludes the paper.

2. Problem description

We treat the order picking process in a single picking station, where the set S of SKUs is handled and a given set of picking orders $O = \{o_1, \ldots, o_n\}$ is to be retrieved. Each order $o_i \in O$ is defined by a set $o_i \subseteq S$ of SKUs demanded by the order's customer. Furthermore, we have a given set of racks $R = \{r_1, \ldots, r_m\}$, where each rack r_j is defined by the set $r_j \subset S$ of SKUs it contains. Each picking station has a given capacity C, which defines the maximum number of customer orders that can be processed in parallel. Once an order is completed it is replaced by the next customer order. In this setting, order processing is mainly influenced by the sequence π in which the picking orders are processed and the sequence μ in which the mobile robots deliver the racks to the picking station. The impact of both sequences on order picking is shown by the following example.

Example: Consider the set $S = \{A, B, C, D\}$ of different SKUs, which are contained in $n = 4$ picking orders defined as follows: $o_1 = \{A, B, C\}$, $o_2 = \{A, B, C, D\}$, $o_3 = \{B, C, D\}$, and $o_4 = \{C, D\}$. Furthermore, we have $m = 3$ racks containing the following SKUs: $r_1 = \{A, C\}$, $r_2 = \{B, D\}$, $r_3 = \{C, D\}$. The capacity of the workbench is $C = 2$, so that two picking orders can be processed concurrently. Fig. 3 depicts two alternative solutions. Solution (a) is based on order sequence $\pi^a = \langle o_4, o_3, o_1, o_2 \rangle$ and rack sequence $\mu^a = \langle r_3, r_2, r_1 \rangle$, so that all orders are processed after three rack visits. Solution (b) requires four rack visits for order sequence $\pi^b = \langle o_2, o_1, o_4, o_3 \rangle$ and rack sequence $\mu^b = \langle r_3, r_2, r_1, r_2 \rangle$.

The example shows that the order sequence, which directly influences the batches of concurrently processed orders, and the rack sequence impact the number of rack visits necessary until all orders are completed. Minimizing the number of rack visits seems a well-suited objective for the order picking process, because of the following two considerations:

- With fewer rack visits required to complete an order set the number of robots required tends to be smaller. Seeing that a typical distribution center applies many picking stations to be supplied in parallel, this reduces the risk of bottleneck situations where no robot is available and a station is starving for SKUs. Over the long run, this objective may even allow to reduce the fleet of robots.
- Furthermore, the number of rack visits influences the makespan of the order picking schedule. Once all SKUs of the current rack are retrieved the picker releases the rack in the information system, so that he/she has to wait for the departure of the current rack and the arrival of the successive one. Thus, minimizing the number of rack visits reduces waiting time. All other time components, e.g., retrieving the items from their shelves, packing them, and folding new cardboard boxes, are fixed once the picking list is defined and not influenced by the order sequence. Note that during a rack change the picker can do alternative work, e.g., folding a new cardboard box. However, the more breaks due a rack change occur, the more likely it becomes that alternative work is not available. Therefore, there

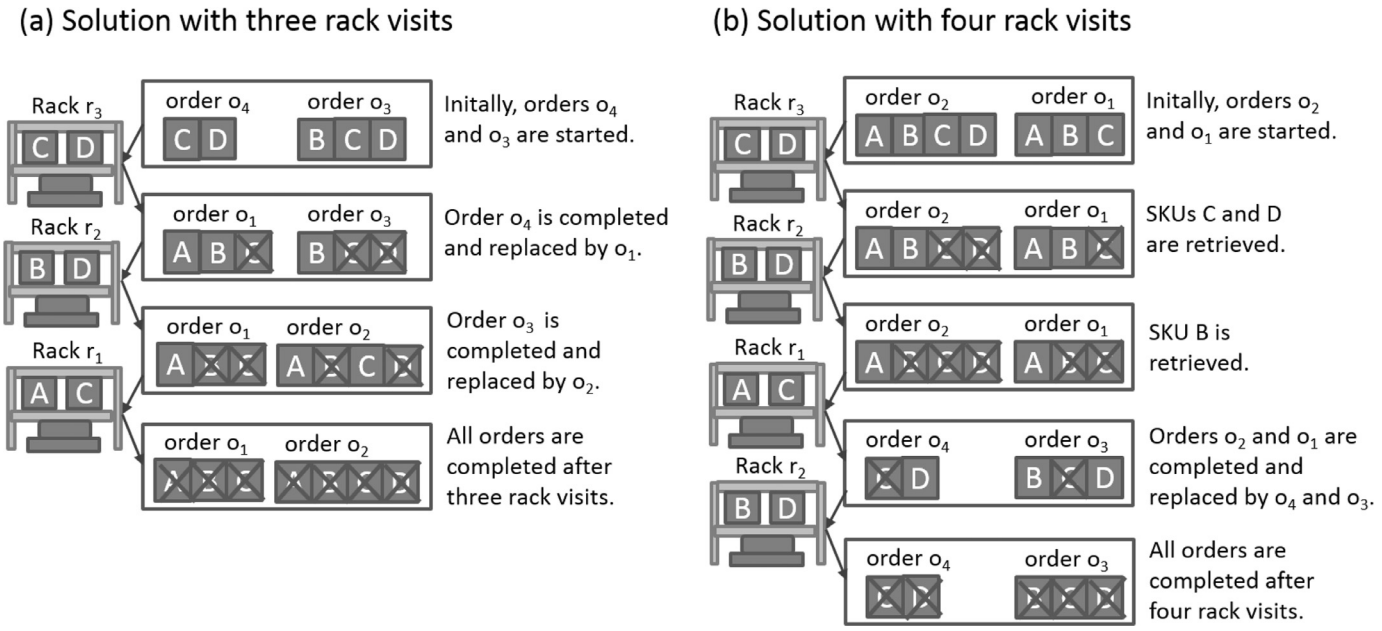


Fig. 3. Example for the picking process depending on the order and rack sequence.

should be a positive impact on the picking time when reducing the number of rack visits although it is not exactly quantifiable.

We will investigate the suitability of our objective in detail during the computational study (see Section 4). Our decision task characterized above is based on the following simplifying assumptions:

- We presuppose a shared storage policy (e.g., (Bartholdi & Hackman, 2016)), which means that a SKU is not stored in a single rack, but scattered over multiple racks. In a mobile robot environment with multiple parallel picking stations such a policy seems inevitable, because otherwise the stations would always have to wait for the one rack delivering a fast-moving item demanded by many customer orders. If fast-moving items are scattered over many racks, then there is a much better chance to always have one of these racks close to a picking station.
- Interdependencies with other picking stations are neglected. Although SKUs are scattered over multiple racks it may still occur that two or more picking stations request the same rack simultaneously. This would necessitate an additional assignment of racks to stations. We only consider an isolated station assigned a fixed set of racks. However, our station specific problem still seems valuable even if interdependencies among stations need to be considered. For instance, our problem allows an evaluation of alternative racks-to-stations assignments in an explorative search process or a larger algorithmic framework. If the racks cannot cleanly be partitioned among stations, because some rack contains a rare SKU which is required at multiple stations, then this rack can be considered at multiple stations and it only has to be avoided that this rack is scheduled during the same time at different stations. This can, for instance, be ensured during the iterative evaluating of multiple stations as follows. If the first station requiring some rack has scheduled it at some sequence position, we only have to block all those sequence positions at a later station also requiring this rack, which do not allow a timely transport of the rack from the former station to the latter. Integrating blocked sequence positions for racks is truly straightforward in our model. However, we leave the evaluation of this idea up to future research.
- If a rack contains a SKU, we assume that it carries enough units to satisfy any picking order. Thus, we deem the number of units demanded by an order and contained in a rack not relevant and only record whether order and rack contain a SKU or not. This simplifying assumption seems well suited, if the number of units per SKU and order is low, which often holds true if the orders are directly placed by final customers.
- The capacity C of the workbench is measured in number of orders, so that C defines the maximum number of picking orders processed concurrently. This presupposes that all cardboard boxes are (nearly) of equal size, so that it is not possible to substitute a larger cardboard box by multiple smaller ones.

Given this decision context our mobile robot based order picking problem (denoted as MROP) can be formally defined as follows. The overall schedule consists of two different pieces of information. First, the sequence μ defining the succession of rack visits at the picking station has to be determined. Here, $\mu_l \in R$ defines the rack at lth position of μ. Note that this sequence can be of varying length depending on how many racks are required to finally supply all orders (in their processing sequence). Further note that the same rack may be assigned to multiple positions, because it is possible – and may be even necessary – to reinsert a just processed rack (and its respective robot) into the queue. Of course, it is also possible to not select a rack at all, if its SKUs can also be retrieved from alternative racks. Furthermore, the batches of picking orders and their exact succession need to be defined, for which we apply a set Ω of quadruples where $(i, c, S_i, C_i) \in \Omega$ specifies the position $c \in \{1, \ldots, C\}$ on the workbench the picking order o_i is assigned to and the first rack visit $S_i > 0$ and last rack visit $C_i \geq S_i$ during its processing. Note that C_i and S_i refer to positions in the rack sequence. Note also that after finishing an order, the successive order replacing its predecessor at the same position of the workbench may still pick items from the current rack. Only afterwards the rack is removed. We say a schedule (μ, Ω) is feasible if

1. μ has at least $\max\{C_i|(i, c, S_i, C_i) \in \Omega\}$ positions,
2. for each $o_i \in O$ there is exactly one $(i, c, S_i, C_i) \in \Omega$, that is each picking order is scheduled exactly once,

Table 1
Notation.

$\mathcal{T}$	Number of slots ($t = 1, \ldots, \mathcal{T}$)
n	Number of picking orders ($i = 1, \ldots, n$)
m	Number of racks ($j = 1, \ldots, m$)
S	Set of SKUs ($s \in S$)
C	Capacity of picking station
R_s	Set of racks providing $s \in S$
$x_{j,t}$	Binary variables: 1, if rack j is visiting in slot t
$z_{s,i,t}$	Binary variables: 1, if SKU s is delivered for order o_i in slot t
$y_{i,t}$	Continuous variables: 1, if order o_i is processed in slot t
α_t	Continuous variables: 1, if the racks visiting in $t-1$ and t differ

3. for each pair of picking orders $o_i, o_{i'} \in O,\ i \neq i'$, with $(i, c, S_i, C_i) \in \Omega$ and $(i', c', S_{i'}, C_{i'}) \in \Omega$ we have $c \neq c'$, $C_{i'} \leq S_i$ or $C_i \leq S_{i'}$, that is orders are either processed at different positions of the workbench or their processing intervals do not overlap by more than one rack visit, i.e., overlap is only allowed in a single rack visit where the predecessor order is completed and its successor starts, and
4. for each $(i, c, S_i, C_i) \in \Omega$ we have $o_i \subseteq \bigcup_{l=S_i}^{C_i} r_{\mu_l}$, that is all SKUs of an order are provided by the racks visiting during the respective processing interval.

Among all feasible schedules the MROP seeks one schedule Ω that minimizes the number of rack visits $Z(\Omega) = \max_{(i,c,S_i,C_i)\in\Omega}\{C_i\}$.

Applying the notation summarized in Table 1 our mixed-integer programming (MIP) model for MROP consists of objective function (1) and constraints (2) to (11). We use time slots in the MIP model where a time slot comprises the time interval where a certain subset of orders is in processing and a certain rack is visiting. From one time slot to the following one or more of the aforementioned may change. While the model allows to keep these elements from one slot to the following it is easy to see that we can restrict ourselves to solution where two consecutive time slots differ in at least one order being processed or in the rack visiting. A trivial upper bound for the number of time slots necessary, therefore, is $\mathcal{T} = m \cdot n$.

$$\text{(MROP) Minimize } F = \sum_{t=2}^{\mathcal{T}} \alpha_t \tag{1}$$

subject to

$$\sum_{j=1}^{m} x_{j,t} \leq 1 \qquad \forall t = 1, \ldots, \mathcal{T} \tag{2}$$

$$\sum_{i=1}^{n} y_{i,t} \leq C \qquad \forall t = 1, \ldots, \mathcal{T} \tag{3}$$

$$y_{i,t} + y_{i,t'} \leq 1 + y_{i,t''} \qquad \forall i = 1, \ldots, n, 1 \leq t < t'' < t' \leq \mathcal{T} \tag{4}$$

$$\sum_{t=1}^{\mathcal{T}} z_{s,i,t} \geq 1 \qquad \forall i = 1, \ldots, n, s \in o_i \tag{5}$$

$$2z_{s,i,t} \leq y_{i,t} + \sum_{j \in R_s} x_{j,t} \qquad \forall i = 1, \ldots, m, s \in o_i, t = 1, \ldots, \mathcal{T} \tag{6}$$

$$\alpha_t \geq x_{j,t} - x_{j,t-1} \qquad \forall t = 2, \ldots, \mathcal{T}, j = 1, \ldots, m \tag{7}$$

$$x_{j,t} \in \{0, 1\} \qquad \forall j = 1, \ldots, m, t = 1, \ldots, \mathcal{T} \tag{8}$$

$$z_{s,i,t} \in \{0, 1\} \qquad \forall i = 1, \ldots, n, s \in o_i, t = 1, \ldots, \mathcal{T} \tag{9}$$

$$1 \geq y_{i,t} \geq 0 \qquad \forall i = 1, \ldots, n, t = 1, \ldots, \mathcal{T} \tag{10}$$

$$\alpha_t \geq 0 \qquad \forall t = 2, \ldots, \mathcal{T} \tag{11}$$

Objective (1) minimizes the number of rack changes and, therefore, rack visits. Inequalities (2) and (3) assure that in no slot more than one rack is visiting or more than C orders are processed. Constraint (4) ensures that an order is processed during a set of consecutive slots. Inequality (5) states that each SKUs required by order o_i must be delivered. This can happen only in a slot where both, o_i and a suitable rack are present due to (6). Finally, (7) tracks changes in the racks visiting. Note that due to (6) and (7) and the binary nature of $x_{j,t}$ and $z_{s,i,t}$ variables $y_{i,t}$ and α_t are either forced to take value 1 or allowed to take value 0. Hence, in optimum solutions $\alpha_t = 0$ unless $x_{j,t} - x_{j,t-1} = 1$ for a rack j. However, even in optimum solutions we may have $0 < y_{i,t} < 1$ for a slot t and an order o_i. Note that for each feasible solution rounding down the value of each $y_{i,t}$ to the next lowest integer yields a feasible solution with the same objective value.

Prior to solving the MROP, which is elaborated in the following section, we state the complexity status of MROP and present two simple reduction rules to eliminate superfluous input data from a problem instances.

Theorem 1. *MROP is strongly NP-hard even for $n = 1$.*

Proof. See Appendix A. □

Due to the first reduction rule, we only have to consider those SKUs, which are demanded by at least one picking order. Consequently, we can also delete all those racks, which do not contain any item from the reduced SKU set S, since these racks, obviously, cannot contribute to supplying some order. Furthermore, we can delete all those racks $j \in R$ for which another rack $j' \in R$, $j \neq j'$, exist with $r_j \subset r_{j'}$. This directly follows from allowing racks to revisit the station multiple times. Given the previous condition, any SKU supplied by j can also be supplied by j', so that j' can always substitute j without increasing the objective value.

3. Decomposing MROP

We, first, investigate two subproblems in Sections 3.1 and 3.2 and propose solution methods. In Section 3.3 we employ these methods as a toolbox and provide procedures combining a search mechanism and the solution methods for one or both subproblems in order to tackle MROP.

3.1. Rack sequencing for a given order sequence

In this section, we treat the subproblem of MROP where the sequence π of picking orders is given, so that only the optimal rack sequence μ is sought. A fixed order sequence defines the next picking order started once a picking order is completed and removed from the workbench. We call this subproblem MROP-RS. Unfortunately, the following complexity status is to be assigned to the problem.

Theorem 2. *MROP-RS is strongly NP-hard.*

Proof. Exactly the same transformation as in the proof of Theorem 1 can here be applied, as well. □

For solving MROP-RS, we now introduce an exact dynamic programming (DP) scheme, which is later on extended to a heuristic beam search procedure. Our DP is subdivided into (at most) $nm + 1$ stages where stage $l = 0, 1, \ldots, nm$ decides on the assignment of a rack to sequence position l of rack sequence μ. Note that we do not need more than nm rack visits. Each stage l, $l > 0$, contains states $(\delta_1, \ldots, \delta_C, \gamma, l)$, where δ_c is the set of SKUs not yet supplied for the picking order currently processed at position c on the workbench and pointer γ refers to the next order of given order sequence π to be processed. In stage 0 we have only $(o_{\pi_1}, \ldots, o_{\pi_C}, C + 1, 0)$ as an initial state.

Transitions between states exist only between two consecutive stages l and $l + 1$ and correspond to assigning a specific rack to

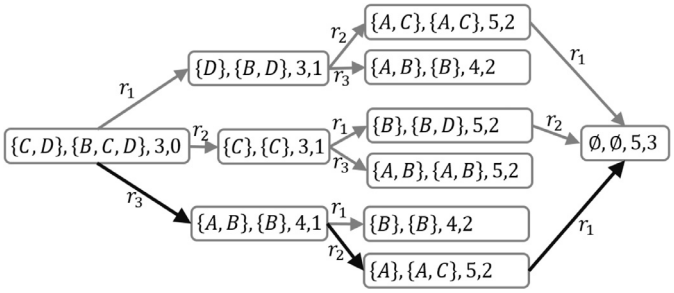


Fig. 4. DP graph for MORP-RS.

position $l+1$ of μ. Consequently, each state may be starting point for $O(m)$ transitions. It is easy to see that once a picking order is completed it may be replaced immediately, that is it does not make sense to leave a position on the workbench empty or keep a completed picking order on the workbench. Furthermore, once a picking order o_i is on the workbench a SKU in o_i is retrieved from the first available rack providing it. The transitions, then, can be specified as follows. Assigning rack j as next rack based on $(\delta_1, \ldots, \delta_C, \gamma, l)$ leads to $(\delta_1 \setminus r_{j'}, \ldots, \delta_C \setminus r_{j'}, \gamma, l+1)$ if $\delta_c \setminus r_j \neq \emptyset$ for each position c. If, however, there is a subset of C' positions on the workbench where picking orders can be completed with those SKUs provided by rack j, then the picking orders in positions $\gamma, \ldots, \gamma + C' - 1$ of π replace them (unless there are no further picking orders to be processed). The corresponding entries in the resulting state are $o_{\pi_\gamma} \setminus r_{j'}, \ldots, o_{\pi_{\gamma+C'-1}} \setminus r_{j'}$ and the corresponding pointer is $\gamma + C'$.

Analogously to the reduction rules presented above, which allow to remove some racks from the problem instance, not all racks may have to be considered for the next sequence position. A rack that does not contain any SKU demanded by the current batch of orders can be eliminated, that is for a state $(\delta_1, \ldots, \delta_C, \gamma, l)$ rack j is not relevant, if $\bigcup_{c=1}^{C} \delta_c \cap r_j = \emptyset$.

This way, successor states are developed in a stage-wise manner until, for the first time, a final state $(\delta_1 = \emptyset, \ldots, \delta_C = \emptyset, n+1, l)$ is reached, that is all picking orders are completed. Then, the DP terminates and the optimal objective value equals l. An optimal rack sequence can be simply obtained by a simple backward recursion along some path to the final state.

Example (cont.): Consider the example of Fig. 3 and a given order sequence $\pi = \langle o_4, o_3, o_1, o_2 \rangle$. The resulting DP graph is depicted in Fig. 4 and the optimal solution corresponding to the black-marked path to final node (∅, ∅, 5, 3) is the one depicted in Fig. 3(a).

The number of states is in $\mathcal{O}(mn|2^{|S|}|^C)$. We have $\mathcal{O}(m)$ transitions per state and determining the resulting state may take up to $\mathcal{O}(|S| \cdot C)$ time. Thus, the runtime of DP is in $\mathcal{O}(|S| \cdot Cmn|2^{|S|}|^C)$. Hence, in line with our complexity result, the state space of DP grows exponentially. However, the proposed DP scheme can also be used as the basis for a beam search heuristic, see (Lowerre, 1976). Applying a beam search heuristic requires the setting of a parameter BW (the beam width), which constrains the number of states that are further explored in each stage to the BW most promising ones. For selecting the BW states per stage we rank them according to increasing $n - \gamma$, i.e., the number of remaining picking orders, and as a tie-breaker we apply the minimum remaining SKUs of the current batch, i.e., $\left|\bigcup_{c=1}^{C} \delta_c\right|$.

3.2. Order scheduling for a given rack sequence

In this section, we mirror our view on the problem and consider subproblem MROP-OS where we have a given rack sequence and seek an order schedule. An order schedule can be specified by assigning two positions in the rack sequence to each order o_i marking the first rack visit and the last rack visit, respectively, during o_i's processing. We say that the former and the latter is o_i's start position and end position, respectively. Note that for an arbitrary feasible order schedule, where all orders can be completed, there is potential to drop positions from the given rack sequence without losing feasibility. We consider dropping the last visits from the rack sequence and define MROP-OS as the problem of finding an order schedule for the given rack sequence where the maximum end position among picking orders is minimum. Unfortunately, even the feasibility version of MROP-OS where we ask for a feasible order schedule turns out to be NP-complete.

Theorem 3. *The problem to decide whether there is a feasible solution to MROP-OS is strongly NP-complete even if* $C = 1$.

Proof. See Appendix B. □

For MROP-OS, too, we propose a DP procedure. In this DP approach orders are added one by one to a partial schedule constituting the first orders to be processed and their respective start positions and end positions. We restrict ourselves to a strategy where a job is added only such that its start position is not smaller than the maximum start position of the orders already scheduled. It is obvious that such a strategy does not prevent us from finding an optimum order schedule since for each order schedule such a sequence of orders exists. Note that following this line of thought we have a huge set of potential start positions and end positions, respectively, each time we add an order to a partial schedule. In the following we develop three ideas enabling us to decide both, the unit of capacity an order is processed on and the pair of start position and end position, for given partial schedule and order o_i to be added.

- In order to reduce the set of potential start positions and end positions, respectively, we predetermine the possible processing intervals of each order as follows. A possible processing interval v is determined by start position $s(v)$ and end position $e(v)$. A position p of the rack sequence is a candidate for o_i's start position only if the pth rack visiting provides a SKU for o_i. For given p we determine position q which is the smallest position such that all SKUs for o_i are provided in positions $p, \ldots, q$. Note that for given p it is not necessary to consider end positions larger than q since corresponding intervals are dominated by (p, q). The pair (p, q) then determines a feasible processing interval v with $s(v) = p$ and $e(v) = q$ to be considered in our DP approach. We may find two possible processing intervals v and v' with $e(v) = e(v')$ and $s(v) < s(v')$. If so, we can drop v since v' dominates v. The set of remaining processing intervals for each order o_i, then, determines the set of possible start positions (and respective end positions) to be considered in our DP approach.
- Given a partial schedule and an order o_i to be added, we have s, i.e., the maximum actual start position among orders already scheduled, and s', i.e., the smallest position in the rack sequence, such that $s' \geq s$ and at least one unit of capacity is available in s', readily available. Note that a unit of capacity becomes available in the last position where the last order on this unit is processed. A possible processing interval v determined for o_i as above is ruled out as the actual processing interval of o_i if $s' > s(v)$ due to our adding strategy. From the set of remaining possible processing intervals we can choose the one having the smallest start position. Note that this is a unique choice considering the set of possible processing intervals generated above. Further note that this choice can easily be seen to dominate choosing any other processing interval. This is again due to our adding strategy.

- Given a partial schedule and an order o_i to be added, we can assign o_i to the unit of capacity becoming finally available earliest. Obviously, this is a valid choice due to the actual processing interval determined as described above. Moreover, considering the adding strategy it is also optimum with regard to capacity availability after adding o_i.

While constructing a schedule by adding jobs we have to track the capacity occupation profile. Since we only add orders "at the end of the schedule" it is sufficient to keep the earliest position in the rack position when a certain unit of capacity becomes finally available.

We are now equipped to formulate the DP approach. We propose to employ states $(O', p_1, \ldots, p_C)$ with

- O' representing the orders already scheduled and
- p_c representing the rack position at which the cth unit of capacity becomes available.

Then, we consider a transition from $(O', p_1, \ldots, p_C)$ to $(O' \cup \{i\}, p'_1, \ldots, p'_C)$ if it represents choosing a processing interval v and unit of capacity for order o_i to be added to the partial schedule represented by $(O', p_1, \ldots, p_C)$, that is if

- $i \notin O'$,
- v is the possible processing interval of o_i with smallest start position not smaller than $\min\{p_c \mid c = 1, \ldots, C\}$, and
- $p'_1, \ldots, p'_C$ is the capacity occupation profile when interval v is chosen for order o_i and o_i is assigned to capacity unit $c^* = \arg\min\{p_c \mid c = 1, \ldots, C\}$.

Note that more than one capacity unit may be available at rack position $s(v)$. These units have to be considered available not before $s(v)$ for further orders due to our adding strategy. A capacity occupation profile $p'_1, \ldots, p'_C$, therefore, results from choosing interval v for o_i and assigning o_i to capacity unit c^* for given $(O', p_1, \ldots, p_C)$ if

- $p'_c = p_c$ if $p_c > s(v)$,
- $p'_{c^*} = e(v)$, and
- $p'_c = s(v)$ if $p_c \leq s(v)$ and $c \neq c^*$.

Solving MROP-OS, then, is accomplished by determining a state $(O, p_1, \ldots, p_C)$ with minimum $\max\{p_c \mid c = 1, \ldots, C\}$ which can be reached by transitions starting from an initial state $(\emptyset, 0, \ldots, 0)$.

The number of states used in the DP approach is bounded by $O(2^n T^C)$ where T denotes the maximum end time over all possible processing intervals. The number of transitions is $O(n2^n T^C)$, i.e., for each choice of the order to be added next the actual processing interval and the unit of capacity it is processed on is determined by the capacity occupation profile of the current state.

Note that in order to eliminate symmetry we can handle $p_1, \ldots, p_C$ such that $p_1 \leq \ldots \leq p_C$ since identical units of capacity need not be differentiated. Finally, if two distinct states $(O', p_1, \ldots, p_C)$ and $(O', p'_1, \ldots, p'_C)$ are reached with $p_c \leq p'_c$ for each $c = 1, \ldots, C$ we can drop $(O', p'_1, \ldots, p'_C)$ from further consideration.

Example (cont.): Consider the given rack sequence $\mu = \langle r_3, r_2, r_1, r_2 \rangle$ of Fig. 3(b) leading to the following processing intervals: o_1: [2, 3], [3, 4], o_2: [2, 3], [3, 4], o_3: [1, 2], [2, 3], [3, 4], and o_4: [1, 1], [2, 3], [3, 4]. Given these intervals the resulting DP graph and a bold-faced optimal solution are depicted in Fig. 5. When applying optimal order sequence $\pi = \langle o_4, o_3, o_1, o_2 \rangle$ order processing can already be completed during the third rack visit, so that the final rack visit of r_2 can be dropped from the given rack sequence.

Just as in Section 3.1 the DP approach presented here can very well serve as a basis for a beam search heuristic. For selecting the BW most promising states per stage we rank them according to their capacity occupation profiles, i.e., we order them by increasing values and consider $p_C, \ldots, p_1$ in lexicographic order.

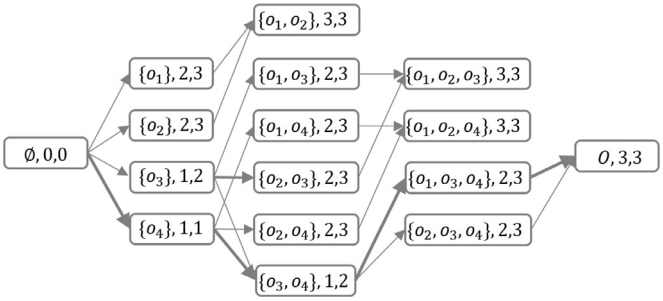


Fig. 5. DP graph for MORP-OS.

3.3. Decomposition methods for MROP

Provided either a "good" order sequence or rack sequence can be found, it is likely that using the methods proposed in Section 3.1 and 3.2, respectively, we can obtain a good solution to MROP. Hence, we provide search mechanisms here based on the well-known simulated annealing scheme, see (Kirkpatrick, Gelatt, & Vecchi, 1983), in order to find such good sequences. Each sequence considered in the course of the search process is evaluated by applying the beam search mechanism in Section 3.1 and 3.2, respectively. As a third alternative we propose a procedure iteratively solving MROP-RS and MROP-OS in an alternating manner.

SA for finding a good order sequence:. When searching for a good order sequence, each individual of the search space is simply encoded as a permutation π of $\langle o_1, o_2, \ldots, o_n \rangle$. The fitness value $fv(\pi)$ corresponds to the objective value of the solution determined by the beam search approach defined in Section 3.1. The initial solution is determined as a randomly generated order sequence and the temperature τ is initialized as proposed by Ben-Ameur (2004): (i) Generate 100 random initial solutions as well as one random neighborhood solution for each, where the neighborhood solution must have a strictly worse objective value than the corresponding initial solution. Let Λ^i be the set of initial solutions and Λ^n be the set of neighborhood solutions. (ii) Starting from $\tau_0 := m$ in iteration $i = 0$, iteratively calculate $\omega(\tau_i) = \frac{\sum_{\lambda \in \Lambda^i} \exp(-fv(\lambda)/\tau_i)}{\sum_{\lambda \in \Lambda^n} \exp(-fv(\lambda)/\tau_i)}$. If $|\omega(\tau_i) - 0.8| < 0.01$, stop and and set the initial temperature $\tau := \tau_i$. Otherwise, set $\tau_{i+1} := \tau_i \cdot \frac{\ln(\omega(\tau_i))}{\ln(0.8)}$, $i := i + 1$, and repeat. This ensures that the initial temperature is chosen such that about 80% of the transitions in the first iteration are accepted, which is often proposed as a reasonable value (see, e.g., (Aarts, Korst, & van Laarhoven, 1997)).

For a given temperature value τ, a phase of K iterations is started. The initial epoch length is $K := 10$. After each epoch K is modified similar to Cho, Paik, Yoon, and Kim (2005): $K := K + \lfloor K \cdot (1 - \exp((fv^{min} - fv^{max})/fv^{max})) \rfloor$, where fv^{min} (fv^{max}) is the lowest (highest) fitness value recorded in the past epoch. In each iteration, a neighborhood solution π' is reached by randomly swapping two orders in the current solution π. Then, the difference in fitness values $\Delta = fv(\pi') - fv(\pi)$ is calculated. If $rnd(0; 1) < exp(-\Delta/\tau)$, then π' replaces π, where $rnd(0; 1)$ is a random number drawn from a uniform distribution over [0, 1]. Once the phase is finished, τ is lowered to $0.99 \cdot \tau$ and the next epoch is started. Finally, the search process terminates when τ falls below 0.01.

SA for finding a good rack sequence:. Here, a solution is encoded as a sequence of racks μ, in which each rack may occur multiple times or not at all. While it is hard to check whether there is a feasible solution corresponding to a given rack sequence, see Theorem 3, it is not hard to come up with a rack sequence μ that

allows a feasible solution. However, in order to obtain a reasonably good initial solution we draw 20 random order sequences for each initial solution to be generated, apply the procedures proposed in Section 3.1 to each of them, and use the rack sequence in the best solution found as a starting point. The fitness value $fv(\mu)$ of each rack sequence μ corresponds to the objective value of the solution determined by the beam search approach in Section 3.2.

The overall scheme of the SA approach is the same as described above with the exact same parameters. In each iteration of the SA procedure, a neighborhood solution μ' is reached by randomly swapping two racks in the current solution μ or adding one more occurrence of a random rack in the sequence in a random position. Note that there is no guarantee that the ensuing neighborhood solution even allows for a feasible order sequence. Simply disallowing infeasible neighbors, however, would cut off a huge part of the (feasible as well as infeasible) search space. We therefore add a penalty term β to the fitness value, which is initialized to $\beta = 1$. If the past 5 accepted neighborhood solutions were all infeasible, the penalty term is increased by a factor of 2. If the past 5 accepted neighbors were all feasible, it is set to $\beta/2$ (self-adjusting penalties, e.g., (Hertz, 1992)).

3.3.0.1. Alternatingly solving MROP-RS and MROP-OS:. We start with a random order sequence π^1 and find a rack sequence μ^1 by employing the beam search approach proposed in Section 3.1. In turn, we now determine a new order sequence π^2 for given rack sequence μ^1 with the beam search procedure of Section 3.2 and so on until, finally, the procedure terminates if $\pi^k = \pi^{k+1}$.

4. Computational study

In order to assess the computational efficacy of the proposed algorithms, we implemented them in C# 5.0 and ran a series of tests on an x64 PC with an Intel Core i7-3770 3.4 gigahertz CPU and 8,192 megabyte of RAM. We also used CPLEX 12.5 to solve the MIP model from Section 2 to have a benchmark. In this section, we will first describe how the test data was obtained, which we then use to examine the performance of the heuristics. Finally, we will investigate the effect of optimal schedules on real-world Kiva systems in a simulation experiment, deriving managerial insight.

4.1. Instance generation

Kiva systems are typically used in distribution centers where many small SKUs are stored in a scattered manner on about man-high racks (e.g., (Guizzo, 2008)). Accordingly, we set the number of racks $|R|$ required to fulfill the orders of the planning horizon to either 5 or 100. Note that the warehouse as a whole may well contain more racks; however, for a given time frame, only a subset of these will actually be in use. The number of positions on the workbench C is drawn randomly (discrete uniform distribution) from the interval [2; 10].

During the planning horizon, a total of $|O| = 10$, 50, or 100 orders need to be fulfilled. In typical real-world distribution centers, all SKUs are not picked with the same frequency; some take a much larger share of the total flow than others (e.g., (Bartholdi & Hackman, 2008)). To take this into account, we generate the contents of each order $o_i \in O$ as follows: First, the number $|o_i|$ of items in each order is set to a random integer between 1 and $\lceil 0.1 \cdot |R| + 1 \rceil$ (discrete uniform distribution). Then, each item in o_i is randomly selected according to an exponential distribution with exponent 0.5, where each order contains at most one copy of each SKU. Each rack $r_j \in R$ contains a theoretical maximum random number of between 1 and $\lceil \xi \cdot |R| \cdot 50 \rceil$ SKUs (discrete uniform distribution), where each SKU is also selected according to an exponential distribution with exponent 0.5, meaning that the diversity of SKUs per rack depends on the parameter ξ, and that more frequently ordered SKUs are also more likely to be on a rack. Note that SKUs which are not actually required by any order are not considered, i.e., $S = \bigcup_{o_i \in O} o_i$. Seeing that SKUs are randomly selected according to the exponential distribution, this implies, effectively, that it is highly improbable that racks will ever contain more than a handful of different SKUs. Moreover, the input data is cleaned up as described in Section 2. Note that the way we generate rack contents, it may exceptionally occur that some SKU demanded by an order is not contained in any rack at all. To avoid these infeasible instances, we just add each leftover SKUs to a single randomly selected rack. For reference, the parameters used for instance generation are summarized in Table 2.

Table 2
Parameters for instance generation.

Symbol	Description	Values
$\|O\|$	Number of orders	10, 50, 100
$\|R\|$	Number of racks	5, 100
ξ	Fraction of SKUs per rack	0.005, 0.05, 0.2

4.2. Algorithmic performance

Table 3 shows the relative optimality gaps for the small instances with $n = 10$ orders and $m = 5$ racks. Note that CPLEX is unable to solve larger instances to optimality within the time limit of 60 minutes, therefore optimality gaps are only available for the small instances; even for these, the average CPU time of CPLEX is about 12 minutes. Column ξ is the "SKU density" per rack as explained above, *BW* is the beam width as used by the beam search (BS) heuristics from Sections 3.1 (for rack sequencing) and 3.2 (for order sequencing). *F (opt.)* is the optimal objective value as reported by CPLEX, averaged over the instances that CPLEX could actually solve. Gap (AH) (%), Gap (SA-RS) (%), and Gap (SA-OS) (%) denote the relative optimality gaps of the alternating heuristic (AH), the simulated annealing scheme for finding a good rack sequence (SA-RS), and the simulated annealing scheme for finding a good order sequence (SA-OS), respectively. The columns labeled seconds contain the average CPU time in seconds taken by the corresponding algorithms.

At first glance, both SA procedures may seem to be quite capable of solving the MROP to (near-)optimality, at least if the beam width is sufficiently large ($BW = 25$). However, upon closer scrutiny, it becomes clear that SA-OS decisively outperforms SA-RS. First, SA-RS displays an optimality gap in some instances when $BW = 10$, which, while not huge, nonetheless suggests that this algorithm has a harder time finding good solutions consistently. Second, SA-RS takes a lot more time to solve the instances than SA-OS, despite an almost identical number of iterations (compare 11.3 seconds of CPU time for SA-RS to only 0.3 seconds on average for SA-OS). These problems most probably stem from the fact that SA-RS has to cope with the added difficulty of a lot of infeasible solutions in its search space. For a given rack sequence there is no guarantee that a feasible order sequence even exists. Many neighborhood solutions encountered during the search process will therefore be nonsensical. SA-OS, on the other hand, does not have to deal with these issues because there is always a feasible rack sequence for a given order sequence, apparently making the search far more efficient.

The simple alternating heuristic (AH), consisting of alternatingly solving MROP-RS and MROP-OS, also delivers quite competitive results. Nonetheless, the optimality gap is greater than that of SA-OS in several cases. However, AH is very quick, solving the small instances in significantly less than a second of CPU time, and may therefore be attractive when time is of critical importance.

Table 3
Average optimality gap of the three heuristics (small instances, $n = 10,\ m = 5$).

ξ	F (opt.)	BW	Gap (AH)(%)	Gap (SA-RS)(%)	Gap (SA-OS)(%)	Seconds (AH)	Seconds (SA-RS)	Seconds (SA-OS)
0.005	2.65	10	2.50	8.33	0.00	0.00	7.76	0.20
		25	0.00	0.00	0.00	0.00	15.73	0.20
0.05	2.80	10	0.00	5.83	0.00	0.00	7.41	0.33
		25	0.00	0.00	0.00	0.00	15.68	0.44
0.2	2.65	10	1.67	9.17	0.00	0.00	7.04	0.24
		25	0.00	0.00	0.00	0.00	14.20	0.25

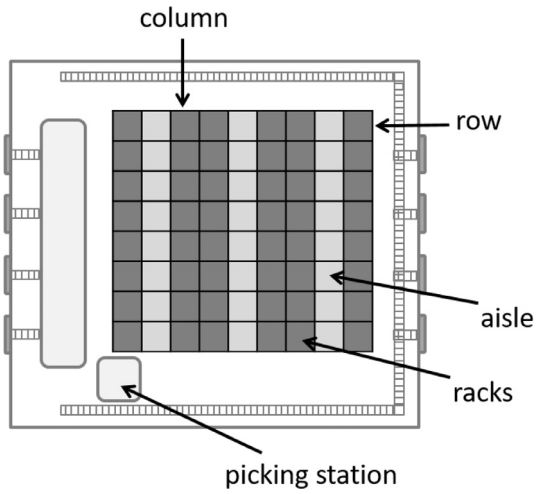


Fig. 6. Schematic layout of the simulated warehouse.

4.3. Simulation study

Objective function F measures the number of rack changes for a given schedule. This is only a surrogate objective, however: In reality, it is not the rack movements per se that drive costs, it is mainly the number of robots that perform these movements as well as the time and energy that these robots spend doing their jobs. Obviously, not every moved shelf will automatically translate to an additional required robot; during a day, one and the same robot can perform a lot of such operations. In order to investigate whether the number of rack changes is actually a good stand-in for the cost of the equipment and the time and energy expended during actual operation, we propose a simulation experiment.

4.3.1. Description of the simulation

The instances are generated as described above. In addition, we assign to each rack $r \in R$ a two-dimensional position (x^r, y^r) on the warehouse floor. We assume that all racks are located on a grid with 10 rows and 20 columns, where we assign each rack to one slot on the grid, i.e., $x^r \in \{1, \ldots, 10\}$ and $y^r \in \{1, \ldots, 20\}$. Each column is 4 meters wide and each row 6 meters long. There is an aisle every two columns of 3 meters width. The picking station (denoted as $r = 0$) is at the bottom left edge of the grid, at position (0, 0). The distance $d(r, r')$, $\forall r, r' \in \{0, 1, \ldots, m\}$, between two racks r and r' (or between a rack location and the picking station in case r or $r' = 0$) on the warehouse floor can then be calculated via the Manhattan metric. The layout is schematically depicted for a warehouse with 8 rows and 6 columns in Fig. 6. Note that this warehouse layout contains more space than is actually needed to house $m = 5$ or 100 racks. For the purposes of our simulation, the remaining space will simply be considered vacant; in the real world, we would expect the space to be used by racks for other picking stations, or simply by racks that are not needed to fulfill the orders in the current order set. Considering that Kiva robots can move underneath racks, it is immaterial for our simulation whether the space is actually occupied or not.

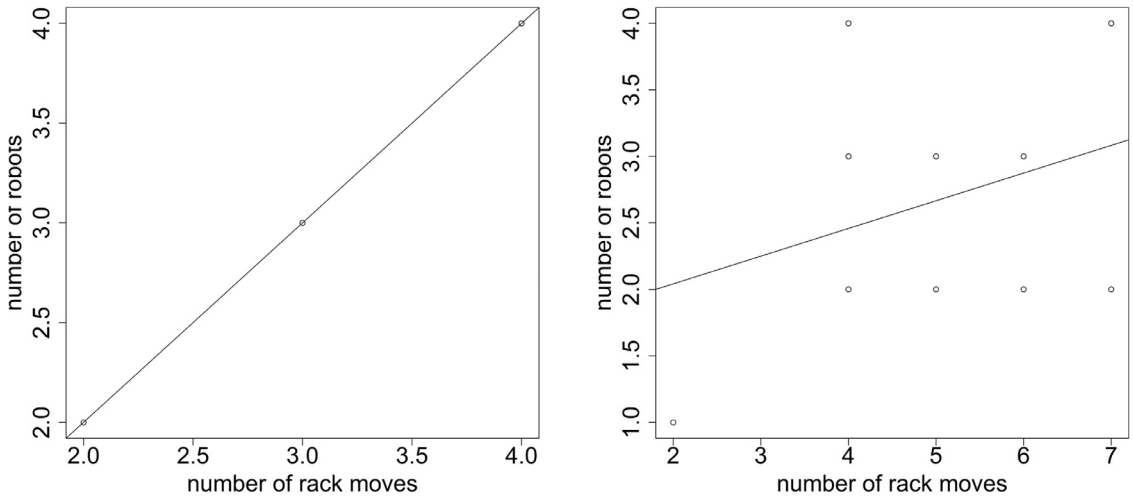
Given a rack sequence μ of length m and an order sequence π of length n, the pick times are calculated as follows. Let δ_i, $\forall i = 1, \ldots, C$, be the set of SKUs that still need to be picked in order to complete the order currently occupying position i of the workbench (initially, $\delta_i = o_{\pi_i}$, $\forall i = 1, \ldots, C$). The jth rack μ_j stays at the picking station for $d_j := \sum_{i=1}^{C} |\delta_i \cap r_{\mu_j}| \cdot T^{\text{pick}}$, where $T^{\text{pick}} = 10$ seconds is the time it takes to pick one SKU. The picked items are then removed from the picking list, i.e., $\delta_i := \delta_i \setminus r_{\mu_j}$. If any δ_i is subsequently $\emptyset$, it is replaced by the next open order from sequence π and the picking time d_j is updated if rack r_{μ_j} contains SKUs demanded by the new order(s). This may have to be done multiple times until no more SKUs can be picked from the current rack, at which time the new rack μ_{j+1} needs to be moved to the station.

Assuming that it is mandatory that the picking station never wait for new items to pick, the moment in time when the jth rack must be at the station is given by $s_j := s_{j-1} + d_{j-1}$, $\forall j = 2, \ldots, m$, where $s_1 := 0$. For the rack to actually be present at the station at time s_j, a robot must have carried it from its location on the floor to the picking station, i.e., one robot must be occupied during the interval $[s_j - 2 \cdot T^{\text{lift}} - d(\mu_j, 0) \cdot T^{\text{speed}}; s_j]$, where $T^{\text{lift}} = 15$ seconds is the time it takes to lift/set down a rack, and a robot moves at a speed of $1/T^{\text{speed}} = 2$ meters per second on average. Similarly, once all SKUs have been picked from rack j, it needs to be carried back to its position during interval $[s_j + d_j; s_j + d_j + 2 \cdot T^{\text{lift}} + d(0, \mu_j)]$. Collecting all of these intervals for every rack move yields the set of trips $V = \{v_1, \ldots, v_{2 \cdot m}\}$. We say that the ordered pair (v_i, v_j) of two trips is compatible if they can be performed by the same robot one after another, i.e., if their time intervals, including the moving time from the end position of trip i to the start position of trip j, do not overlap. The resulting problem of assigning the minimal number of robots to trips can be reduced to finding a maximum matching in a bipartite graph (Bertossi, Carraresi, & Gallo, 1987).

4.3.2. Correlation between rack moves and number of robots

Given a solution to the MROP, that is, an order schedule and a rack sequence, we can calculate both the objective value, i.e., the rack moves, as well as the minimal number of robots via the simulation procedure detailed above. Comparing these values to each other for all 20 instances per parameter constellation should show that whenever the objective value is high then so is the number of robots and vice versa. The Pearson product-moment correlation coefficient for all instance sets is listed in Table 4. Significant results are marked with an asterisk (95% confidence) or two asterisks (99% confidence). The schedules were found by SA-OS with BW = 25. The scatter plots for the cases with both the strongest (1.00) as well as the weakest (0.35) correlation coefficients are in Fig. 7.

The data indicate that there is indeed a very strong correlation between our surrogate objective value of counting the rack changes and the actual number of robots required to execute a schedule. In all cases except for three there is a significant correlation. The three parameter constellations where no significance



(a) $n = 10$, $m = 5$, $\xi = 0.005$. (b) $n = 50$, $m = 5$, $\xi = 0.05$.

Fig. 7. Correlation between objective value and actual number of robots.

Table 4
Correlation between objective value (rack moves) and actual number of robots; results significant at the 95% confidence level are marked with *, those at the 99% level with **.

n	m	ξ	correlation	CPU seconds (AH)	CPU seconds (OS)
10	5	0.005	1.00**	0.00	5.43
10	100	0.005	0.73**	0.01	12.26
10	5	0.05	0.75*	0.00	3.47
10	100	0.05	0.68**	0.01	7.24
10	5	0.2	0.79*	0.00	2.74
10	100	0.2	0.83**	0.00	4.00
50	5	0.005	0.59*	0.12	65.78
50	100	0.005	0.66**	0.33	156.72
50	5	0.05	0.35	0.13	43.26
50	100	0.05	0.80**	0.38	102.72
50	5	0.2	0.5	0.12	42.37
50	100	0.2	0.75**	0.31	90.31
100	5	0.005	0.70**	0.95	168.51
100	100	0.005	0.42	2.54	413.21
100	5	0.05	0.61*	0.84	123.02
100	100	0.05	0.66**	2.54	295.65
100	5	0.2	0.60**	0.91	116.06
100	100	0.2	0.67**	2.45	262.08

could be established require very few robots all around, making it harder to detect trends. We can thus conclude that the number of rack moves is indeed a good stand-in for the actual cost-driver, the size of the robot fleet.

Note that, for reference, Table 4 also contains the CPU times of AH and SA-OS with beam width 25. This piece of information reveals that both algorithms are somewhat more sensitive to the number of orders n than the number of racks m. AH, however, is always very fast, solving all instances in just a handful of seconds, whereas SA-OS may take several minutes.

4.3.3. Evaluating the performance for the large instances

We use our simulation to investigate the performance of the heuristics for the large instances with $m = 100$ racks. The results can be found in Table 5. The columns entitled *robots (AH)* and *robots (SA-OS)* list the average number of robots for the corresponding algorithms. For comparison, we also list the average number of robots for a simple rule-based order processing (*robots (FCFS)*), derived as follows: First an arbitrary order sequence π is created, which in the real-world occurs, for instance, whenever orders are sequenced according to first-come-first-served (FCFS). Then, racks are greedily selected by always choosing the one rack having most SKUs in common with the subset of orders currently processed. Note that ties are broken according to lower index. Finally, columns *driving time* contain the total net time (in seconds) the robots are on the move while performing their respective jobs according to the simulation as described above. Note that we dismissed SA-RS from these tests because it is already clear from the small instances that it is inferior to SA-OS in every regard. Recall that parameter BW defines the beam width of the beam search procedure, which is not relevant for the FCFS heuristic.

Concerning the algorithmic performance, the two heuristics exhibit a similar picture as for the small instances: AH is very quick, hardly ever exceeding 2 or 3 seconds of CPU time (see Table 4), but its results – measured here in terms of required robots – are significantly weaker than those of the more sophisticated SA-OS; the average gap over all large instances is about 21.9%. However, even AH is substantially better than assigning racks in a purely random way: On average the picking station can be supplied by about 40.4% fewer robots even when using the quick AH. When using SA-OS, the savings are even more striking, at about 54.6%. In other words, even a fast heuristic like AH can already almost halve the required equipment, potentially saving a lot of investment money. Similar conclusions also hold for the total driving time of the robots (and thus energy consumption), which can be dramatically reduced by using smart scheduling algorithms.

The tests also show that the beam width does not have a huge impact on the solution quality: For SA-OS, only about 3% of robots can be saved on average by increasing BW. On the other hand, the runtime increases almost linearly, i.e., using SA-OS with BW = 25 consumes about 2.5 times as much time as when BW = 10.

4.3.4. Effect of SKU diversity

Finally, we investigate the impact of the diversity of SKUs per rack, parameter ξ. For classic warehouses with fixed storage locations, we would expect this value to be low – each shelf contains only a few different SKUs, and no or few other shelves contain the same SKUs. This would correspond to our parameter setting $\xi = 0.005$: On each rack there are only at most 0.5% of the SKUs in the warehouse. Many modern distribution centers, on the other hand, assign random storage positions to individual items, meaning that two identical products need not necessarily be stored on the same shelf (denoted as mixed-shelves or scattered storage (Weidinger & Boysen, 2015)). This would correspond to $\xi = 0.05$ or

Table 5
Results of the simulation for the large instances ($n \geq 50$, $m = 100$).

n	ξ	BW	Robots			Driving time		
			FCFS	AH	SA-OS	FCFS	AH	SA-OS
50	0.005	10	21.35	7.95	5.80	9494.93	6720.18	4943.28
50	0.005	25		7.30	5.55		6674.90	5021.85
50	0.05	10	18.15	6.30	4.80	7764.58	5658.25	4274.30
50	0.05	25		6.20	4.65		5494.73	4002.18
50	0.2	10	5.05	3.10	2.40	2802.08	2331.30	1733.40
50	0.2	25		3.00	2.20		2278.58	1793.80
100	0.005	10	47.55	7.65	6.20	23420.98	15090.35	10790.28
100	0.005	25		8.15	6.25		14450.90	10454.98
100	0.05	10	40.00	7.40	5.35	19330.03	12454.40	8918.05
100	0.05	25		6.85	5.20		12419.38	9012.13
100	0.2	10	7.75	3.20	2.60	5781.83	4427.10	3210.80
100	0.2	25		3.15	2.45		4291.90	3159.75

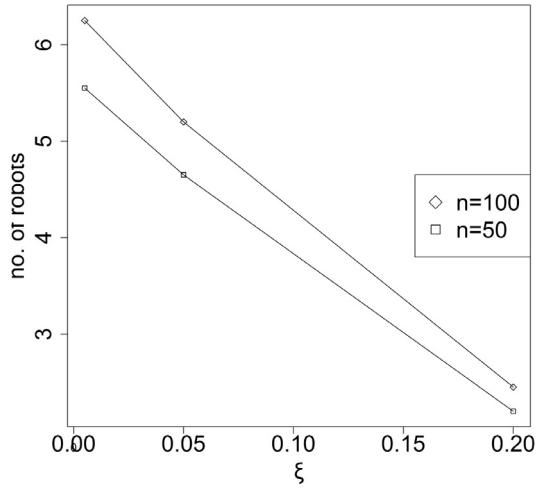


Fig. 8. Effect of SKU diversity ξ on required number of robots ($m = 100$).

even more extremely $\xi = 0.2$. In typical distribution centers that ship consumer products, individual orders are often rather small, and there is therefore little need for multiple copies of the same item to be on one shelf. The likelihood that multiple items can be picked from the same shelf is, of course, larger if many different SKUs are on it. We would thus expect a Kiva system to work more efficiently the more scattered the storage is.

Fig. 8 summarizes the results for the large instances with $m = 100$ racks: As expected, having many different SKUs on the same shelf greatly reduces the effort for the robots, the number of which is about linearly related to ξ. We can thus conclude that a Kiva-driven distribution center will indeed work much more efficiently using mixed-shelves storage. On the other hand, distributing SKUs among multiple shelves increases the effort during replenishment. Evaluating this trade-off in more detail seems a worthwhile task for future research. The graphs also reveal another interesting fact: The number of orders n plays no appreciable role in terms of required robots. Apparently, a certain number of robots – about three to six for our medium-size warehouse with 100 racks – is sufficient to always keep one picking station well supplied, regardless of how many orders need to be fulfilled. Of course, a real-world distribution center will consist of more than just one picking station, so more equipment will be necessary, but these numbers can be a good point of reference.

5. Conclusion

This paper investigates the order processing in picking stations of a Kiva warehouse, where stationary pickers are directly supplied with storage racks by rack moving mobile robots. We formulate the resulting order scheduling and rack sequencing problem and provide suited decomposition procedures. Our comprehensive computational study, which also includes a simulation study of the rack delivery processes of the mobile robots, reveals the following key findings:

- In our test instances, an optimized order processing allows to more than halve the fleet size of robots required to timely supply a picking station compared to a simple rule-based approach like it is often applied in real-world warehouses. The result in our stylized setting are a strong indicator that substantial reductions of the robot fleet should be possible when optimizing real-world Kiva warehouses too. Although a single standardized robot may be comparatively low priced, seeing robot fleets that vary between 75 (warehouse of German workware retailer Engelbert Strauss) and 500 (warehouse of office supply retailer Staples) the potential savings in investment, maintenance, and energy charging equipment are nonetheless substantial. Also total driving duration of robots is cut by a huge portion (at least 21%, often more than 40%).
- A further reduction of the robot fleet is enabled by the shared storage policy. The more scattered SKUs are among racks and the more different SKUs each rack contains the fewer rack movements are required during order processing. Thus, a Kiva system seems especially suited for warehouses storing mainly small sized items and processing orders with just a few order lines; both prerequisites are often fulfilled for online retailers selling directly to final customers.

Future research should focus the additional decision problems listed in Section 1.2 and answer the questions where racks should intermediately be “parked” in a dynamic environment and which SKUs should be stored together in the same rack. Such contributions could considerably facilitate the diffusion of rack moving mobile robots in warehouses all around the world. An other direction of research might generalize the problem setting treated in the paper at hand to account for multiple picking stations.

Appendix A. Complexity proof for MROP

In this appendix we prove that MROP is strongly NP-hard even for $n = 1$.

If we consider an order set consisting of only a single picking order, i.e., $n = 1$, then the transformation from the set covering problem, which is well known to be strongly NP-hard (e.g., (Garey & Johnson, 1979)) is readily available. Given a collection K of subsets of a finite set Γ the set cover problem aims at a subset $K' \subseteq K$ such that every element in Γ belongs to at least one member of K' and $|K'|$ is minimized. We simply have to set $S = o_1 = \Gamma$, $C = 1$, and have a unique rack j for each subset $k \in K$ such that $r_j = k$. Obviously, this results in a one-to-one mapping of both problems.

Table B.6
Pattern of SKUs for order o_i provided by racks.

Pos	1	2	3	4	5	6	7	8	9	10	11
SKUs	s_i^1	s_i^2	s_i^3		s_i^1	s_i^2	s_i^3		s_i^1	s_i^2	s_i^3
	s_i^4	s_i^5	s_i^6		s_i^6	s_i^4	s_i^5		s_i^5	s_i^6	s_i^4
	s_i^7	s_i^8	s_i^9		s_i^8	s_i^9	s_i^7		s_i^9	s_i^7	s_i^8

Appendix B. Complexity proof for MROP-OS

In this section we show that the problem to find a feasible solution to MROP-OS is strongly NP-complete. The proof is based on a transformation from the following interval scheduling problem (denoted as ISP), which was shown to be strongly NP-hard by Nakajima and Hakimi (1982).

ISP: Consider a set J of jobs to be processed on a single machine. Each job $j \in J$ is assigned a set of alternative processing intervals. If a specific processing interval of a job is selected, this means that during the complete time span from the fixed start to the end of the interval the machine is occupied by this job. The ISP maximizes the number of processed jobs, where each job is either not processed at all or assigned exactly one of its processing intervals. Since we have only a single machine the selected intervals are not allowed to overlap.

Strengthening the result above, Keil (1992) showed that it is strongly NP-complete to decide whether all jobs can be processed even if each job has no more than three processing intervals and all processing intervals have identical lengths $\tau = 3$. Spieksma and Crama (1992), finally, proved that the problem is strongly NP-complete even if $\tau = 2$ and any two possible starting times for a job differ by at least 3. The actual difference between two consecutive starting times is 7 in their proof but for our purpose it is sufficient that there is at least one time period between each pair of consecutive processing intervals of a job. We refer to this decision version of ISP where interval lengths equal 2 as ISP2. We now reduce ISP2 to MROP-OS.

Transformation of ISP2 to MROP-OS: Given an instance I of ISP2 with n jobs and possible start times t_i^1, t_i^2, and t_i^3 for each job i we construct an instance I' of MROP-OS as follows. We have n orders where order o_i corresponds to job i and each order o_i requires nine SKUs $s_i^1, \ldots, s_i^9$. Let T denote the maximum end time over all jobs and intervals in I. Then, we consider T racks and our rack sequence consists of T positions with each rack visiting exactly once. Position p of the sequence corresponds to time interval $[p-1, p[$ in I. The SKUs provided by racks are defined as follows.

- If the first processing interval of job i starts in t, then we let rack $t+1$, $t+2$, and $t+3$ provide $\{s_i^1, s_i^4, s_i^7\}$, $\{s_i^2, s_i^5, s_i^8\}$, and $\{s_i^3, s_i^6, s_i^9\}$, respectively.
- If the second processing interval of job i starts in t, then we let rack $t+1$, $t+2$, and $t+3$ provide $\{s_i^1, s_i^6, s_i^8\}$, $\{s_i^2, s_i^4, s_i^9\}$, and $\{s_i^3, s_i^5, s_i^7\}$, respectively.
- If the third processing interval of job i starts in t, then we let rack $t+1$, $t+2$, and $t+3$ provide $\{s_i^1, s_i^5, s_i^9\}$, $\{s_i^2, s_i^6, s_i^7\}$, and $\{s_i^3, s_i^4, s_i^8\}$, respectively.

Due to the specific properties of ISP2, for each order o_i we obtain a pattern of three distinct sets of three consecutive racks each providing the SKUs required by o_i. The pattern of provided SKUs (for $t_i^1 = 0$, $t_i^2 = 4$, and $t_i^3 = 8$) is depicted in Table B.6. The following property is easy to derive.

Property 1. *Any feasible processing interval for order o_i spans at least one of the intervals associated to processing intervals of job i completely.*

Furthermore, we can see that start times t_i and $t_{i'}$, $t_{i'} \geq t_i$, of two jobs i and i' are compatible in I if and only if orders o_i and $o_{i'}$ can be processed in the corresponding intervals. Let $t_{i'} - t_i - 2 = k$. The corresponding intervals in I' overlap by 1 if $k = 0$, overlap by more than one if $k < 0$, and do not overlap at all if $k > 0$. Two orders can be processed in intervals overlapping by at most one in I'.

Property 2. *If and only if two processing intervals can be used by orders o_i and $o_{i'}$ in a feasible solution to I' the corresponding processing intervals can be used by jobs i and i' in a feasible solution to I.*

Now, we claim that there is a feasible solution to I' if and only if there is a solution to I where all jobs are processed. Clearly, if there is a solution to I where all jobs are processed, then it directly corresponds to a feasible solution to I' due to Property 2. If there is a feasible solution to I', then there also is a feasible solution to I' where the actual processing interval of each order o_i coincides with one the processing intervals corresponding to processing intervals of job i due to Property 1. It follows obviously that there is a solution to I where all jobs are processed due to Property 2.

References

Aarts, E., Korst, J., & van Laarhoven, P. (1997). Simulated annealing. In E. Aarts, & J. Lenstra (Eds.), *Local search in combinatorial optimization* (pp. 91–120). John Wiley and Sons, NJ.

Bartholdi, J. J., III, & Hackman, S. T. (2008). Allocating space in a forward pick area of a distribution center for small parts. *IIE Transactions, 40*, 1046–1053.

Bartholdi, J. J., III, & Hackman, S. T. (2016). *Warehouse & distribution science*. Supply Chain and Logistics Institute. Release 0.97.

Ben-Ameur, W. (2004). Computing the initial temperature of simulated annealing. *Computational Optimization and Applications, 29*, 369–385.

Van den Berg, J. P. (1996). Multiple order pick sequencing in a carousel system: a solvable case of the rural postman problem. *Journal of the Operational Research Society, 47*, 1504–1515.

Bertossi, A. A., Carraresi, P., & Gallo, G. (1987). On some matching problems arising in vehicle scheduling models. *Networks, 17*, 271–281.

Cho, H.-S., Paik, C.-H., Yoon, H.-M., & Kim, H.-G. (2005). A robust design of simulated annealing approach for mixed-model sequencing. *Computers & Industrial Engineering, 48*, 753–764.

D'Andrea, R., & Wurman, P. (2008). Future challenges of coordinating hundreds of autonomous vehicles in distribution facilities. In *Proceedings of the IEEE international conference on technologies for practical robot applications.* (pp. 80–83).

Frazelle, E. A., & Sharp, G. P. (1989). Correlated assignment strategy can improve order-picking operation. *Industrial Engineering, 4*, 33–37.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman, New York.

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

Guizzo, E. (2008). Three engineers, hundreds of robots, one warehouse – kiva systems wants to revolutionize distribution centers by setting swarms of robots loose on the inventory. *IEEE Spectrum, 45*(7), 26–34.

Herrero-Pérez, D., & Martínez-Barberá, H. (2011). Decentralized traffic control for non-holonomic flexible automated guided vehicles in industrial environments. *Advanced Robotics, 25*, 739–763.

Hertz, A. (1992). Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics, 35*, 255–270.

Heskett, J. L. (1963). Cube-per-order index – a key to warehouse stock location. *Transportation and Distribution Management, 3*, 27–31.

Keil, J. M. (1992). On the complexity of scheduling tasks with discrete starting times. *Operations Research Letters, 12*, 293–295.

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

Kirks, T., Stenzel, J., Kamagaew, A., & ten Hompel, M. (2012). Zellulare transportfahrzeuge für flexible und wandelbare intralogistiksysteme. *Logistics Journal, 2192*(9084), 1–8.

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

Lamballais, T., Roy, D., & De Koster, M. B. M. (2017). Estimating performance in a robotic mobile fulfillment system. *European Journal of Operational Research, 256*, 976–990.

Lowerre, B. (1976). The harpy speech recognition system. Ph.d. thesis, Carnegie Mellon University.

Meller, R. D., & Klote, J. F. (2004). A throughput model for carousel/VLM pods. *IIE Transactions, 36*, 725–741.

Mountz, M. C. (2005). Material handling system and method using mobile autonomous inventory trays and peer-to-peer communications. *US Patent, 6*(950), 722.

Nakajima, K., & Hakimi, S. L. (1982). Complexity results for scheduling tasks with discrete starting times. *Journal of Algorithms, 3*, 344–361.
Roodbergen, K. J., & Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research, 194*, 343–362.
Spieksma, F. C., & Crama, Y. (1992). The complexity of scheduling short tasks with few starting times. In *Reports in Operations Research and Systems Theory m92-06*. Maastricht University.
Vis, I. F. (2006). Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research, 170*, 677–709.
Weidinger, F., & Boysen, N. (2015). Scattered storage: How to distribute stock keeping units all around a mixed-shelves warehouse. Working Paper Friedrich-Schiller-University Jena.
Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine, 29*(1), 9–19.
Yu, J. (2016). Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics and Automation Letters, 1*, 33–40.