European Journal of Operational Research 260 (2017) 534–545

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing and Logistics

Order picking along a crane-supplied pick face: The SKU switching problem

Stefan Schwerdfeger[a], Nils Boysen[b,*]

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

ARTICLE INFO

Article history:
Received 23 May 2016
Accepted 21 December 2016
Available online 27 December 2016

Keywords:
Logistics
Warehousing
Order picking
Sequencing

ABSTRACT

This paper treats an order picking system where a crane continuously relocates stock keeping units (SKUs) in a high-bay rack subdivided into the bottommost picking level and the upper reserve area. The capacity of the pick face is not large enough to store all SKUs, so that the crane has to ensure that all SKUs demanded by a current picking order are timely provided and picker idle time is avoided. We aim at a processing sequence of picking orders and a SKU switching plan, i.e., an instruction when to exchange which SKUs in the picking level, such that an unobstructed order picking is enabled. Our problem is closely related to the tool switching problem of flexible manufacturing. Here, each job requires a subset of tools to be loaded into the tool magazine (with limited capacity) of a single flexible machine. We, however, show that an alternative objective function, i.e., minimizing the maximum number of switches between any successive job pair, is better suited in the warehouse context and even better results can be obtained by a multi-objective approach. Elementary complexity proofs as well as suited solution procedures are provided and we also address managerial aspects, such as the sizing of the pick face.

1. Introduction

In the e-commerce era, next-day or even same-day deliveries have become an elementary component of many business models and supply chains (e.g. Yaman, Karasan, & Kara, 2012). This increases the pressure on the warehousing processes, which have to ensure a fast and efficient retrieval of stock keeping units (SKUs) from the shelves. Traditional warehouses, which according to de Koster, Le-Duc, and Roodbergen (2007) still amount to 80% of all Western European warehouses, are organized according to the picker-to-parts paradigm and – as a main drawback – require a lot of unproductive picker travel. Consequently, it is not astounding that plenty of innovations have been generated that focus on the reduction of travel times. The manifold ideas range from automated parts-to-picker systems (e.g. Gu, Goetschalckx, & McGinnis, 2010) to organizational modifications like batching and zoning (e.g. de Koster et al., 2007). In this paper, we deal with an order picking system where picker travel is not completely eliminated, but considerably reduced by, at any one time, only loading a subset of SKUs into a pick face of reduced size.

1.1. A crane-supplied pick face

The basic element of a warehouse applying a crane-supplied pick face (dubbed CSPF) is a high-bay rack, which is subdivided into two parts: the pick face and the reserve area (see Fig. 1). The bottom row is the picking level where SKUs are directly accessible by human order pickers. Parallel to the pick face is a conveyor, e.g., an unpowered roll conveyor, on which bins or cardboard boxes associated with a picking order accompany an order picker. The picker and the current bin move along the pick face and collect all SKUs demanded by the current picking order. Once the end of the pick face is reached and the current picking order is completed, the conveyor moves the order onwards to the shipping area and the picker returns to the start where a new picking order is initiated. Note that often a pick-to-light system is applied to ease the orientation of pickers. Such a system indicates all SKUs requested by the current order with a light signal.

The upper rows of the high-bay rack constitute the reserve area where SKUs are stored, which are currently not available in the pick face. We call a SKU currently loaded into the pick face to be *active* and a SKU only available in the reserve area to be *passive*. To reduce the walking distance of the picker, the pick face does not

* Corresponding author.
E-mail addresses: stefan.schwerdfeger@uni-jena.de (S. Schwerdfeger), nils.boysen@uni-jena.de (N. Boysen).
URL: http://www.wiwi.uni-jena.de/Entscheidung/ (S. Schwerdfeger), http://www.om.uni-jena.de (N. Boysen)

http://dx.doi.org/10.1016/j.ejor.2016.12.037
0377-2217/© 2016 Elsevier B.V. All rights reserved.

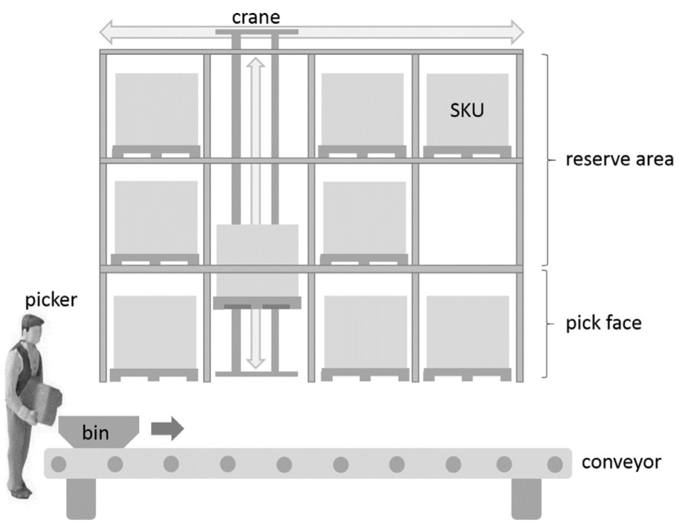

Fig. 1. Schematic layout of the CSPF system.

have enough capacity to store the complete assortment of SKUs. Only a subset of SKUs is active at a time and whenever the current picking order requires a passive SKU, an active SKU is to be removed from the pick face and replaced by the respective (formerly passive) SKU. We call such a combined loading and unloading event a (SKU) *switch*. A switch is executed by an automated storage/retrieval machine, which we also call the *crane*. Note that picker and crane do not interfere, because they work at opposite sides of the rack.

Kim, Heragu, Graves, and Onge (2003) report on a miniload CSPF in a distribution center for cosmetic products. We saw such a system based on unit loads in a warehouse of a book distributor. Here, the CSPF system is applied to assemble fast-moving items ordered by brick-and-mortar book stores. In general, the system seems especially suited if each picking order contains just a few order lines and the assortment of SKUs is not overly large as well. The former allows to reduce the length of the pick face, such that picker travel is reduced, and the latter reduces the capacity of the high-bay rack where excessive demand for space considerably increases investment costs.

The main operational problem in a CSPF system is the *blocking* of a picker, who has to halt order picking whenever the crane has not yet switched all SKUs demanded by the current picking order. Clearly, each blocking enlarges the picking time and deteriorates picking performance. It, thus, seems advantageous to reduce the number of SKU switches required between any pair of successive picking orders, so that the crane has enough time to execute all required switches prior to the new arrival of the picker with the subsequent order. This paper addresses the associated optimization problem, which we denote the SKU switching problem. We aim to derive processing sequences of picking orders and SKU switching plans, i.e., an instruction of which SKUs to switch when, such that picker blocking is minimized.

1.2. Literature review and relation to the tool switching problem

An efficient processing of picking orders is among the main topics of warehousing research ever since. Instead of trying to summarize the vast body of literature that has accumulated on the wide range of different warehouse settings we only refer to the most recent survey papers of de Koster et al. (2007) and Gu et al. (2010). A more recent survey paper specialized on crane operated storage systems is provided by Boysen and Stephan (2016). They provide a classification scheme for the resulting crane scheduling problems and come to the conclusion that papers explicitly dealing with CSPF systems are rare. This is also the finding of Ramtin and Pazour (2014) and our own literature search.

CSPF systems are also known as automated storage and retrieval systems with multiple-in-aisle pick positions (Ramtin & Pazour, 2015). Existing research rather treats long-term layout problems; e.g., analytical models are derived to determine expected travel times (including retrieval times when the crane accesses a SKU) for different layouts (Ramtin & Pazour, 2014) or product allocations (Ramtin & Pazour, 2015). Operational decision problems are (to the best of the authors' knowledge) only treated by Kim et al. (2003) and Boysen and Stephan (2016). They, however, only treat the crane movement and presuppose given switches, whereas we treat how these switches are generated. The former paper deals with a huge CSPF system in the cosmetics industry and focuses on an efficient crane scheduling. Different heuristic procedures are provided to reduce the makespan of the crane when switching bins. Crane scheduling is also addressed in Boysen and Stephan (2016). Here, a complexity proof is provided that proves NP-hardness for the special case of the travelling salesman problem where all crane moves start or end at the bottom line. However, the SKU switching problem in a CSPF system has not been treated in the warehousing literature yet.

Instead, a very similar optimization problem is vividly researched in the context of flexible manufacturing (for an overview, e.g., see Buzacott & Yao, 1986; Kouvelis, 1992). The tool switching problem, which was introduced by Tang and Denardo (1988a), can be briefly characterized as follows: given a set of jobs, a set of tools, and a single machine with a tool magazine of given capacity. Each tool requires a single slot in the tool magazine and a specific job can only be processed if the subset of tools required by this job is available in the tool magazine. The tool switching problem seeks a job sequence and a tool switching plan. There are two alternative objective functions for the tool switching problem that are discussed in the flexible manufacturing literature:

(a) The sum of tool switches is minimized, which minimizes the makespan, if during tool changing no job can be processed and each switch takes equal time (Crama, Oerlemans, & Spieksma, 1994). We denote this objective as the *min–sum* objective.
(b) The number of tool switching events is minimized, which is a suited performance criterion if the switching time is (roughly) constant and independent of the number of tool switches (Tang & Denardo, 1988b).

NP-hardness of the tool switching problem with objective (a) is proven by Crama et al. (1994) and many heuristic (e.g. Bard, 1988; Chaves, Lorena, Senne, & Resende, 2016; Hertz, Laporte, Mittaz, & Stecke, 1998; Tang & Denardo, 1988a) and exact (Laporte, Salazar-Gonzalez, & Semet, 2004) solutions procedures have been provided. With objective (b), NP-hardness is shown in Tang and Denardo (1988b), heuristics are provided, e.g., in Tang and Denardo (1988b), Konak, Kulturel-Konak, and Azizoğlu (2008), and a column generation approach to solve the linear relaxation is presented in Crama and Oerlemans (1994).

The tool switching problem can easily be transferred to the warehouse context. If we substitute jobs with picking orders, tools with SKUs, and the tool magazine with the pick face, then the mapping between both problems is readily available. In the warehousing context, however, an important characteristic of the tool switching problem is not fulfilled. Switching can not only be executed if order processing is stopped. As crane and picker operate on different sides of the rack, they can work in parallel. The crane can switch SKUs during order picking if either an active SKU is not required by the current picking order or the respective SKU has already been picked for the current order. Thus, an alternative performance criterion seems better suited in the warehousing context. We suggest to minimize the maximum number of switches between any pair of successive orders. If just a few switches are to be executed, then the probability is high that the crane has enough time to execute these switches prior to the successive arrival of the picker with the next picking order. Thus, picker blocking is avoided. To the best of the authors' knowledge, with this objective neither the tool nor the SKU switching problem has been treated in the literature yet.

1.3. Contribution and paper structure

This paper treats the order processing in a CSPF system. We aim at a processing sequence of picking orders and a SKU switching plan, such that the maximum number of switches between any pair of successive orders is minimized. We formulate the problem (Section 2), investigate computational complexity, and provide suited solution procedures (Section 3). The computational performance is investigated in Section 4. Managerial aspects are addressed in Section 5. With the help of a simulation study, we investigate whether our proposed objective is indeed well suited to avoid picker blocking. The results reveal that our min–max objective indeed outperforms the traditional min–sum objective in most settings, but even better results can be obtained by a multi-objective procedure, which is also introduced and tested. Furthermore, we systematically explore how differently sized pick faces impact picking performance. Finally, Section 6 concludes the paper.

2. Problem description and complexity

Suppose that a set $J = \{1, \ldots, n\}$ of picking orders has to be successively assembled, one at a time, from the pick face of a CSPF system. Each order $j \in J$ demands a subset $S_j \subset S$ of SKUs, whose unit loads (or bins) have to be loaded into the pick face prior to picking order j, i.e., S_j has to be active. The total set of SKUs demanded by the current order set is denoted by S and throughout the paper we assume the orders being labeled by their order size, i.e., $|S_1| \geq \cdots \geq |S_n|$. The unit loads of all SKUs require identical space, e.g., each SKU is stored on the same type of standardized pallet. The pick face has a limited capacity $C < |S|$, which limits the maximum number of active SKUs being concurrently loaded into the pick face. To ensure feasibility of the problem we presuppose that $|S_j| \leq C$, that is no order requires more SKUs than there is space in the pick face.

The crane switches active with passive SKUs stored in the reserve area of the high-bay rack. We call the set of active SKUs currently stored on the picking level a *loading* of the pick face. Initially, we have a given loading L_0 and loadings L_i denote the set of SKUs being active when picking the ith order starts.

The SKU switching problem seeks an order sequence π, i.e., a permutation of $\{1, \ldots, n\}$ with π_i denoting the order processed at sequence position i, and a SKU switching plan defining loadings $L_1, \ldots, L_n$. We call a solution, i.e., order sequence plus switching plan, feasible, if:

- $|L_i| \leq C$ for all $i = 1, \ldots, n$, that is a loading may not exceed the capacity of the pick face, and
- $S_{\pi_i} \subseteq L_i$ for all $i = 1, \ldots, n$, that is all SKUs demanded by an order are active at the start of its processing.

Among all feasible solutions we seek one, which minimizes

$$F(\pi, L_1, \ldots, L_n) = \max_{i=1,\ldots,n} |L_i \setminus L_{i-1}|, \tag{1}$$

that is we aim to minimize the maximum number of switches to be executed by the crane between any pair of successive orders. We denote this objective as the *min–max* objective. A mixed-integer model for the SKU switching problem under the min–max objective is presented in the Appendix A.

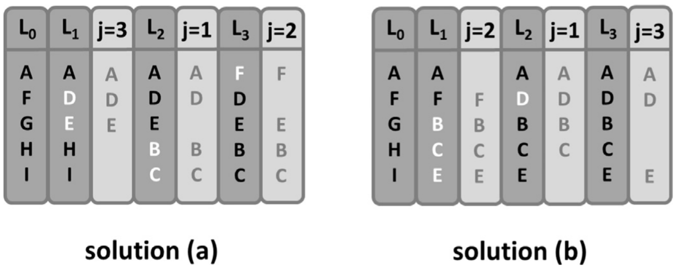

Fig. 2. Two alternative solutions for the example.

Example 1. Consider $n = 3$ picking orders, which require the following SKUs: $S_1 = \{A, B, C, D\}$, $S_2 = \{B, C, E, F\}$, and $S_3 = \{A, D, E\}$. The pick face has capacity $C = 5$ and the initial loading is $L_0 = \{A, F, G, H, I\}$. Fig. 2 depicts two alternative solutions where the white letters indicate SKU switches. Solution (a) requires a maximum number of $F = 2$ switches over all sequence positions, whereas alternative solution (b) leads to $F = 3$. Note that solution (a) is optimal with regard to our min–max objective and solution (b) with regard to the two traditional objectives of the tool switching problem. Specifically, solution (b) requires four switches and two switching events. This leads to the conclusion that, in this instance, optimality with regard to our novel objective can only be achieved accepting suboptimality with regard to both other objectives. Thus, there exist instances where the objectives are competitive. Nevertheless, there are clearly instances where some or even all of the objectives can be optimized simultaneously.

Let us now briefly discuss the most important assumptions (explicitly and implicitly) contained in the SKU switching problem:

- Each SKU is assumed to consume the same space in the pick face, so that capacity C can be measured in number of active SKUs. Fully automated switches by cranes, typically, require standardized load carriers, e.g., bins and pallets, so that this assumption should be met in most applications. However, removing this assumption would add considerable difficulties. For instance, the physical location of a SKU would become relevant, because the gap freed by a removed SKU may not be sufficient for another SKU with larger space requirement, so that further SKUs to the left or right need to be removed.
- No mixed-pallets containing more than a single SKU are available. Each SKU is assumed to be stored in a single homogeneous bin or unit-load. Thus, compared to the demanded number of items per order line a large number of items are in stock. We assume that there are enough items in stock to satisfy all picking orders, so that is only relevant whether or not an order requires a SKU but not the specific quantity.
- We presuppose $|S_j| \leq C$, that is no order requires more SKUs than there is capacity in the pick face. If, nonetheless, such orders exceptionally occur in the real-world, they can simply be subdivided into multiple smaller orders that have to be merged in the shipping area.
- The list of picking orders is known with certainty. Typically, the steady stream of incoming picking orders is broken down by successively releasing subsets of orders. Our given order set, then, equals the next wave of orders.
- We restrict our view on a single picker and a single crane. In larger CSPF systems multiple pickers and cranes may work in parallel, which is, for instance, described in Kim et al. (2003). We, however, aim at an analysis of the most basic problem setting and exclude the additional complexity arising from the

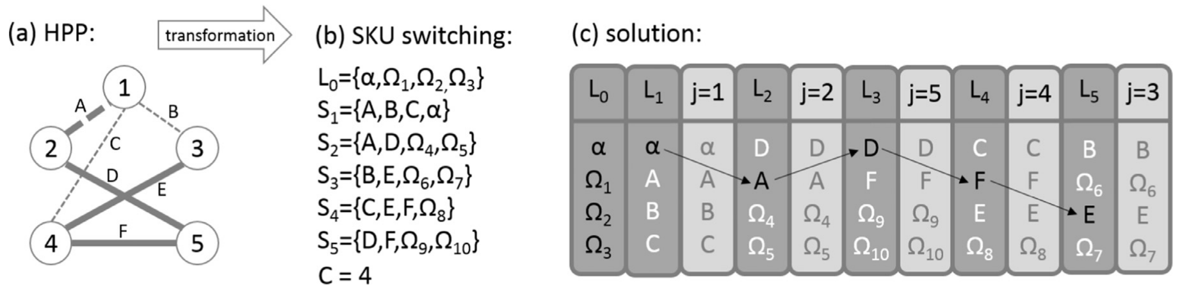

Fig. 3. Transformation of HPP to SKU switching and corresponding feasible solutions.

coordination of multiple resources. In the system we observed at a book distributor, parallel resources were indeed not relevant, but even for larger systems our research could lead to basic structural insights. Moreover, our solution procedures may be applicable for solving subproblems in a larger algorithmic framework.

- We do not explicitly coordinate the movement of picker and crane. This would add considerable complexity to the problem. To exactly quantify picker blocking, the exact positions of picker and crane would have to be tracked over the complete planning horizon, which leads to a very bulky and complicated optimization problem. We suggest a much simpler surrogate objective. Minimizing the maximum number of switches over sequence positions increases the probability that the crane has enough time to timely execute all required switches prior to the next arrival of the picker with the successive picking order. This way only the SKU switches have to be calculated and neither picker nor crane positions need to be considered. The simulation study provided in Section 5.2 is specifically dedicated to evaluating whether our objective is indeed a good proxy for the actual objective, i.e., minimizing picker blocking.

In spite of our simplified surrogate objective the SKU switching problem still constitutes a challenging optimization task, which directly follows from the subsequent complexity proof.

Theorem 1. *The SKU switching problem is strongly NP-hard.*

We prove this theorem by a reduction from the Hamiltonian Path problem (HPP), which is well-known to be strongly NP-complete (Garey & Johnson, 1979) and can be stated as follows. Given an undirected graph $G = (V, E)$ the HPP determines whether or not a path through the graph exists that visits all vertices $v \in V$ exactly once. Note that this problem remains NP-complete if a fixed start vertex for the path is given.

Proof. Consider an instance I of HPP given by graph $G = (V, E)$ and start vertex p. Let $d = \max_{v \in V}\{\deg(v)\}$ be the maximum degree (or valency) of all vertices $i \in V$. We construct an instance I' of SKU switching in polynomial time as follows. We introduce $n = |V|$ picking orders each requiring $|S_j| = d + 1$ SKUs. The capacity of the pick face also equals $C = d + 1$. For each edge $(i, j) \in E$ we introduce a SKU, which is exclusively demanded by the two orders i and j representing the incident vertices of the edge. The order associated with start vertex p additionally receives a SKU α that is exclusively contained in this order. All orders that by now not already contain $|S_j| = d + 1$ SKUs are filled up with individual SKUs that exclusively occur once in single order. Finally, we add SKU α to initial SKU loading L_0 and fill it up to $C = d + 1$ with individual SKUs that are contained in neither picking order. The question we ask is whether a solution with $F \leq d$ exists, that is the maximum number of SKU switches over all sequence positions does not exceed d.

An example of the transformation scheme is given in Fig. 3. Initial SKU loading L_0 only contains individual SKUs, which are indicated by Ω_i with $i = 1, \ldots, 10$ in Fig. 3, except for α. Thus, to not exceed d switches, the first order has to be the one representing start vertex p, which is vertex 1 in our example. From then on, given switching budget d can only be met, if exactly one SKU remains active in any sequence position, which is only possible if two successive orders have a SKU in common. As SKUs are only shared by two orders, if they have a connecting edge in HPP, a Hamiltonian path directly equals the succession of SKUs remaining in the pick face and vice versa. □

3. Solution procedures

This section is dedicated to exact and heuristic solution procedures for the SKU switching problem. At first, we introduce a heuristic decomposition approach, which separates the generation of order sequence and switching plan (Section 3.1). Then, Section 3.2 introduces three lower bounds, which are extended to local lower bounds and applied in an exact branch-and-bound procedure (Section 3.3). Eventually, Section 3.4 presents two benchmark procedures, i.e., one procedure solves SKU switching under the min–sum objective and the other considers both objectives in a lexicographic order. These procedures are applied in our computational study to explore the suitability of our objective function.

3.1. A heuristic decomposition procedure

Our decomposition procedure (dubbed D-MIN–MAX) follows an approach often applied to the tool switching problem under the min–sum objective. In the first stage, order sequences are generated, which are handed over to the second stage where the switching plan for the given sequence is determined. For the tool switching problem under the min–sum objective optimal switching plans for a given order sequence can be derived in polynomial time with the so-called “Keep Tool Needed Soonest” (KTNS) procedure (Tang & Denardo, 1988a). We, first, show how to alter this algorithm, so that for our problem, too, optimal switching plans for predetermined order sequences can be derived in polynomial time. Afterwards, we elaborate on the first stage of our decomposition approach and describe how order sequences are generated.

Our adapted version of KTNS minimizing the maximum tool switches for a given order sequence (dubbed “Keep SKU Needed Soonest” KSNS) is a two stage procedure. On the first stage, a binary search looks for the min–max number of switches and is initiated with the search interval between a lower and an upper bound, e.g., 0 and C. In each iteration of the binary search, stage two is called where algorithm ϱ checks whether or not there is a feasible switching plan with at most M switches. Depending on the result of ϱ the search interval of the binary search is adapted until the min–max number of tool switches is found.

The basic idea of algorithm ϱ is fairly simple. Starting in the first sequence position, we move onwards along the given order sequence. In each step, all passive SKUs required by the next order have to be switched anyway. They are exchanged with those active SKUs required again either not at all (first priority) or in the latest sequence positions. If these mandatory switches exceed M, no feasible switching plan can be found and this negative result is returned. If our "budget" of M switches is not yet depleted additional switches are executed where we switch those SKUs required again latest against those required soonest (provided that the passive SKUs about to be moved into the pick face are required earlier that the active SKUs to be removed). These additional switches are executed until M moves are reached. Then, we proceed with the next sequence position and once all sequence positions are processed the positive result is returned. A formal definition of algorithm ϱ is given in the following.

For a shorthand notation, we use $t(j_k^A)$ as the minimum number of sequence positions (or orders according to the given sequence) between the first order after the current one S_{π_i} at which SKU $j_k^A \in A,\ k \in 1,\dots,|A|$ is needed (default: $n+1$) for a given set $A \in \{L_i, S\backslash L_i\}$, e.g. $t(j_k^{L_i}) = 1 \Leftrightarrow j_k^{L_i} \in S_{\pi_{i+1}}$. Without loss of generality, we may assume the SKUs being ordered by $t(j_1^{L_i}) \le \dots \le t(j_C^{L_i})$ as well as $t(j_1^{S\backslash L_i}) \le \dots \le t(j_{n-C}^{S\backslash L_i})$.

Algorithm ϱ: Set $k := 1$. As long as $k \le M$ and $t(j_{C-k+1}^{L_i}) > t(j_k^{S\backslash L_i})$ remove the active SKU $j_{C-k+1}^{L_i}$ from the pick face and replace it by the passive SKU $j_k^{S\backslash L_i}$ from the reserve area and set $k := k+1$, that is replace the active SKU needed latest by the passive SKU needed soonest. Now, either the SKUs needed to fulfill the order $S_{\pi_{i+1}}$ are available in the pick face and the iterative procedure is repeated (set $i := i+1$ while there are orders left) or it is not possible to fulfill order $S_{\pi_{i+1}}$ by at most M SKU switches from L_i.

In the following we provide a proof of correctness for KSNS by proving the following theorem.

Theorem 2. *For a given sequence π KSNS minimizes the maximum number of switches between any successive job pair.*

Proof. Without loss of generality let $\pi = \langle 1, 2, \dots, n]$. In order to prove that algorithm KSNS yields the minimum number of maximum switches to fulfill the orders for a given sequence π it suffices, therefore, to prove that in case algorithm ϱ fails for a given number of maximum switches M there exists no other algorithm (or switching plan) to satisfy all orders with at most M switches between two adjacent orders of sequence π. In other words, applying ϱ there exists an iteration q with $S_{q+1} \setminus L_q > M$, i.e., it is not possible to fulfill the order S_{q+1} with at most M switches from L_q.

To prove this, let $t^\sigma(j_k^{L_i})$ be the minimum number of sequence positions between the first order after current order S_i, at which SKU $j_k^{L_i} \in L_i,\ k \in 1,\dots,C$, is needed (default: $n+1$) applying algorithm $\sigma \in \Sigma$, e.g., $t^\sigma(j_k^{L_i}) = 1 \Leftrightarrow j_k^{L_i} \in S_{i+1}$. Without loss of generality, we may assume the (active) SKUs to be ordered by $t^\sigma(j_1^{L_i}) \le \dots \le t^\sigma(j_C^{L_i})$.

By construction of the greedy algorithm ϱ

$$t^\varrho(j_k^{L_i}) \le t^\sigma(j_k^{L_i}) \quad k = 1,\dots,C;\ i = 1,\dots,q;\ \sigma \in \Sigma \tag{2}$$

holds. From condition (2) it immediately follows that

$$\left|\left\{k = 1,\dots,C : t^\varrho(j_k^{L_i}) = 1\right\}\right| \le \left|\left\{k = 1,\dots,C : t^\sigma(j_k^{L_i}) = 1\right\}\right| \quad i = 1,\dots,q;\ \sigma \in \Sigma. \tag{3}$$

As algorithm ϱ fails to find a feasible switching plan with at most M switches per iteration $\left|\left\{k = 1,\dots,C : t^\varrho(j_k^{L_q}) = 1\right\}\right| > M$ holds at iteration q. Eventually, (3) completes the proof. □

The binary search of KSNS requires the solution of $\mathcal{O}(\log C)$ decision problems. Algorithm ϱ itself iterates through all sequence positions where it has to sort all SKUs. Thus, KSNS runs in $\mathcal{O}((n \cdot |S| \log |S|) \log C)$. This result is summarized by the following theorem.

Theorem 3. *KSNS runs in polynomial time.*

The first stage of our decomposition procedure generates order sequences, which are handed over to stage two where KSNS determines the min–max number of SKU switches for the given sequences. Instead of only producing a single order sequence, our decomposition approach generates n sequences (each evaluated with KSNS) and returns the best among them. The general idea is to start each sequence with a new order, so that each order is the initial order in exactly one order sequence. Given an initial order all following sequence positions are successively fixed in a greedy manner by choosing the one order among those not yet fixed, which increases the objective value least. This is determined by solving KSNS for the resulting partial solution, if the respective order would be added to the already fixed sequence positions. A formal definition of stage one is given in the following.

Let $d_{0j}^l := |S_j \setminus L_0|$ be the minimum number of required switches in case $\pi_1 = j$ and π^i the partial sequence $\pi^i = \langle S_{\pi_1}, \dots, S_{\pi_i}]$ with objective function value $UB(\pi^i)$. First, we set $UB = C$ and sort the orders by d_0^l, i.e., $d_{0\sigma(1)}^l \le \dots \le d_{0\sigma(n)}^l$. For each $j = 1,\dots,n$ a partial sequence is initialized with the order $S_{\pi_1},\ \pi_1 := \sigma(j)$. At each iteration i, $i = 2,\dots,n$, all remaining orders are evaluated by temporarily appending them to the currently fixed partial sequence (one at a time) and evaluating the resulting partial sequence with KSNS. The one increases the current objective function value least of all (tie breaker: largest order), is append to the partial sequence to obtain π^i. In case $UB(\pi^i) \ge UB$ the partial solution π^i will not improve the current best solution, so that we continue with a new sequence $S_{\pi_1},\ \pi_1 := \sigma(j+1)$, while $d_{0\sigma(j+1)}^l < UB$. On contrary, whenever $UB(\pi^n) < UB$ we obtain a new best solution and set $UB = UB(\pi^n)$.

Example 2. Consider $n = 4$ picking orders, which require the following SKUs: $S_1 = \{E, I, L, M\}$, $S_2 = \{H, I, K, N\}$, $S_3 = \{F, G, O\}$, and $S_4 = \{D, J\}$. The pick face has capacity $C = 5$ and the initial loading is $L_0 = \{A, B, C, D, F\}$. Hence, $d_0^l = (4, 4, 2, 1)$, so that picking order S_4 requires fewest initial SKU switches and is evaluated first. Then, applying KSNS to determine the objective function value $UB(\pi^i)$ for the partial sequence π^i yields

π^2	$UB(\pi^2)$	π^3	$UB(\pi^3)$	π^4	$UB(\pi^4)$
$\langle S_4, S_1]$	3	$\langle S_4, S_3, S_1]$	3	$\langle S_4, S_3, S_1, S_2]$	3
$\langle S_4, S_2]$	3	$\langle S_4, S_3, S_2]$	3		
$\langle S_4, S_3]$	2				

and we obtain the sequence $\langle S_4, S_3, S_1, S_2]$ with objective function value $UB = 3$. As $d_{0\sigma(2)}^l = 2 < UB$ holds, the procedure continue as follows

π^2	$UB(\pi^2)$	π^3	$UB(\pi^3)$	π^4	$UB(\pi^4)$
$\langle S_3, S_1]$	3	$\langle S_3, S_4, S_1]$	3	–	
$\langle S_3, S_2]$	3	$\langle S_3, S_4, S_2]$	3		
$\langle S_3, S_4]$	2				

The execution stops, because of $UB(\pi^3) \ge UB$ and $d_{0\sigma(3)}^l = 4 > UB$.

3.2. Lower bounds

To evaluate the quality of solutions and, especially, to verify optimality, we derive some lower bounds in this section. They are later on extended to local lower bounds applied in a branch-and-bound algorithm (see Section 3.3).

```
0. Set $I = \{1, \dots, n\}$
1. for $h = 1, \dots, \lfloor \frac{n}{2} - 1 \rfloor$
2.     Set $i := \text{argmax}_{i \in I} \left\{ \left| \{ j \in I : d^l_{ij} = \min_{a,b \in I} \{d^l_{ab}\} \} \right| \right\}$, $I := I \backslash \{i\}$
3. end
4. Set $LB^2 = \min_{a,b \in I} \{d^l_{ab}\}$
```

Fig. 4. LB^2 procedure.

First and foremost, let us define two distance measures $d^l_{ij} := \max \left\{ 0, |S_i \cup S_j| - C \right\}$ and $d^u_{ij} := |S_j \setminus S_i|$. d^l_{ij} represents the minimum number of SKU switches if orders $i, j \in J$ are adjacent, whereas d^u_{ij} constitutes the maximum number of switches needed, if order j is a direct successor of order i. Recall that $d^l_{0j} := |S_j \setminus L_0|$ denotes the minimum number of required switches in case $\pi_1 = j$.

Then, applying Kruskal's algorithm (Kruskal, 1956) to determine the minimum 0-spanning tree (spanning tree on J with cheapest cost edge to L_0) with vertices $J \cup \{0\}$ and cost edges d^l_{ij} the highest cost edge among the used edges yields lower bound LB^1 (cf. Crama et al., 1994).

Based on the observation that each subset of $\lceil \frac{n}{2} + 1 \rceil$ jobs contains at least a pair of two adjacent jobs, we can derive the following straightforward procedure (see Fig. 4) to determine another lower bound.

As observed by Crama et al. (1994), $l := |\bigcup_{j \in J} S_j \cup L_0| - |L_0|$ constitutes a lower bound for the minimum total number of SKU switches. Thus, we can derive a lower bound for our min–max objective by $\lceil l/n \rceil$. Introducing l_h, which denotes a lower bound for the sum of SKU switches at iteration h defined by

$$l_h := \min_{I \subseteq J : |I| = h} \left| \bigcup_{j \in I} S_j \cup L_0 \right| - |L_0|, \quad h = 1, \dots, n,$$

we obtain the following bound $LB^3 := \max_{h=1,\dots,n} \left\lceil \frac{l_h}{h} \right\rceil$. As the computational effort of l_h grows exponentially with n it is advisable to compute l_h merely for $h \in J(k) := \{1, \dots, k, n-k, \dots, n\}$ and a small value k. Therefore, the computation of LB^3 is restricted to all i-element subsets ($i = 0, \dots, k$) and their corresponding complementary $(n-i)$-element subsets. As they can be computed simultaneously, $\sum_{i=0}^{k} \binom{n}{i}$ subsets are investigated when determining LB^3. Note that our computational study presented in Section 4.2 evaluates suited values for k.

Altogether, we define $LB := \max\{LB^1, LB^2, LB^3\}$ as the best lower bound.

Example 2 (cont.)**.** First our two distance measures d^l_{ij} and d^u_{ij}, which define the minimum and maximum number of switches if order j follows i, respectively, amount to

$$d^l = \begin{pmatrix} - & 2 & 2 & 1 \\ 2 & - & 2 & 1 \\ 2 & 2 & - & 0 \\ 1 & 1 & 0 & - \end{pmatrix}; \; d^u = \begin{pmatrix} - & 3 & 3 & 2 \\ 3 & - & 3 & 2 \\ 4 & 4 & - & 2 \\ 4 & 4 & 3 & - \end{pmatrix}.$$

The minimum 0-spanning tree consists of edges (0, 4), (3, 4), (1, 4), (2, 4), so that $LB^1 = 1$. To strengthen the obtained lower bound let us determine LB^2. As $\lfloor \frac{n}{2} - 1 \rfloor = 1$ we are allowed to remove at least one order $i := \text{argmax}_{i \in I} \left\{ \left| \left\{ j \in I : d^l_{ij} = 0 \right\} \right| \right\} = 4$. Consequently $I := \{1, 2, 3\}$ and $LB^2 = 2$. To compute LB^3 let us define $h_I := \left| \bigcup_{j \in I} S_j \cup L_0 \right| - |L_0|$. Then,

I	h_I	I	h_I	I	h_I	I	h_I
{1}	4	{1, 2}	7	{1, 2, 3}	9	{1, 2, 3, 4}	10
{2}	4	{1, 3}	6	{1, 2, 4}	8		
{3}	2	{1, 4}	5	{1, 3, 4}	7		
{4}	1	{2, 3}	6	{2, 3, 4}	7		
		{2, 4}	5				
		{3, 4}	3				
$l_1 = 1$		$l_2 = 3$		$l_3 = 7$		$l_4 = 10$	

$LB^3 = \max \left\{ \left\lceil \frac{1}{1} \right\rceil, \left\lceil \frac{3}{2} \right\rceil, \left\lceil \frac{7}{3} \right\rceil, \left\lceil \frac{10}{4} \right\rceil \right\} = 3$ and $LB = \max\{1, 2, 3\} = 3$.

3.3. An exact branch-and-bound procedure

In this section, an exact branch-and-bound procedure is developed, whose key ingredients, i.e., branching and bounding, are described in the following.

Abbreviatory, we define LB and UB as global lower and upper bound, respectively. π^i denotes the partial sequence with $\pi^i = \langle S_{\pi_1}, \dots, S_{\pi_i}]$ and $LB(\pi^i)$ ($UB(\pi^i)$) is the local lower (upper) bound associated with π^i.

The branching strategy is a depth-first search, where, for a given partial sequence $\pi^i = \langle S_{\pi_1}, \dots, S_{\pi_i}]$, the next picking order selected for branching is $\pi_{i+1} := \text{argmin} \left\{ h \in J \setminus \bigcup_{k=1}^{i} \pi_k : d^l_{\pi_i h} \le UB \right\}$, i.e., the unbranched picking order which requires the most SKUs. In case $d^u_{\pi_i \pi_{i+1}} \le UB(\pi^i)$, we set $UB(\pi^{i+1}) := UB(\pi^i)$, otherwise we apply KSNS to the partial sequence $\pi^{i+1} = \langle S_{\pi_1}, \dots, S_{\pi_{i+1}}]$ to obtain $UB(\pi^{i+1})$. In case $UB(\pi^{i+1}) \ge UB$, we prune the subtree.

Note that π^n represents a feasible solution and $UB(\pi^n)$ the corresponding objective function value. Therefore, whenever $UB(\pi^n) < UB$ holds, we set $UB = UB(\pi^n)$. Moreover, in case $UB(\pi^n) = LB$ (see Section 3.2) π^n yields an optimal sequence and thus, the procedure is aborted.

To derive local lower bounds we adapt the three lower bounds defined in Section 3.2:

- Given a partial sequence $\pi^i = \langle S_{\pi_1}, \dots, S_{\pi_i}]$ $LLB^1_{\pi^i}$ is equal to the highest cost edge among the used edges of minimum π_i-spanning tree with vertices $J_{\pi^i} \cup \{\pi_i\}$ and cost edges d^l_{ij} with $J_{\pi^i} := J \setminus \left\{ \bigcup_{j=1}^{i} \pi_j \right\}$.
- We obtain $LLB^2_{\pi^i}$ by removing $\lfloor \frac{n-i}{2} - 1 \rfloor$ jobs of $I := J_{\pi^i}$ (similar to Fig. 4). Thus, $LLB^2_{\pi^i} := \min_{a,b \in I} \{d^l_{ab}\}$.
- Finally, $LLB^3_{\pi^i} := \max_{h=1,\dots,n-i} \left\lceil \frac{l^i_h}{h} \right\rceil$ with $l^i_h := \min_{I \subseteq J_{\pi^i} : |I| = h} \left| \bigcup_{j \in I} S_j \cup S_{\pi_i} \right| - C$, $h = 1, \dots, n-i$. Analogously, we set $J_{\pi^i}(k) := \{1, \dots, k, n-i-k \dots, n-i\}$.

In case $LLB_{\pi^i} := \max \left\{ LLB^1_{\pi^i}, LLB^2_{\pi^i}, LLB^3_{\pi^i} \right\} \ge UB$ we prune the subtree.

Example 2 (cont.)**.** For our example, assume that the partial sequence $\pi^1 = \langle S_3]$ is given. The minimum 3-spanning tree consists of the edges (3, 4), (1, 4), (2, 4), so that $LLB^1_{\pi^1} = 1$. As $\lfloor \frac{n-1}{2} - 1 \rfloor = 0$ we are not allowed to remove any order. Hence, $I := \{1, 2, 4\}$ and $LLB^2_{\pi^1} = 1$. To compute $LLB^3_{\pi^1}$ let us define $h_I := \left| \bigcup_{j \in I} S_j \cup S_3 \right| - C$. Then,

I	h_I	I	h_I	I	h_I
{1}	2	{2, 4}	4	{1, 2, 4}	7
{2}	2	{1, 4}	4		
{4}	0	{1, 2}	5		
$l^1_1 = 0$		$l^1_2 = 4$		$l^1_3 = 7$	

$LLB^3_{\pi^1} = \max \left\{ \left\lceil \frac{0}{1} \right\rceil, \left\lceil \frac{4}{2} \right\rceil, \left\lceil \frac{7}{3} \right\rceil \right\} = 3 \ge UB.$

3.4. A multi-objective procedure

To show that our min–max objective function of minimizing the maximum number of switches between any successive job pair is indeed better suited in the warehouse context than minimizing the total number of switches (i.e., the min–sum objective), we will explore their impact on the makespan within Section 5.2. With this in mind, we want to introduce two further algorithms.

For solving the tool switching problem under the min–sum objective we apply an approach dubbed D-MIN–SUM. It generates order sequences in the first stage (in an identical manner as D-MIN–MAX) and only applies the KTNS procedure of Tang and Denardo (1988a) to minimize the sum of switches for the given order sequence instead.

Due to the possibility of cranes and pickers working in parallel we, generally, expect the min–max objective to be superior to the min–sum objective. However, it is still possible that in extreme situations the min–sum objective outperforms the min–max objective. If the picker moves very fast (compared to the crane) – in the very extreme he/she moves with infinite velocity – then the picker will have to wait for any switch and minimizing the sum of switches is the better objective choice. Furthermore, there may be many solutions being optimal with regard to the min–max objective, which, however, considerably vary in the total number of switches. Thus, we also introduce multi-objective procedure D-LEX, which considers both objectives in a lexicographic order.

Firstly, D-LEX targets the min–max objective and under all solutions with the lowest number of maximum switches it aims to find one causing the fewest number of switches. Specifically, we apply D-MIN–MAX to determine a sequence π with an objective function value $UB := UB(\pi)$, which is handed over to the second stage. Here, we successively swap two orders of π and apply algorithm ϱ to check whether or not there is still a feasible switching plan with at most UB switches per changeover. In case of a positive result, we use Gurobi to determine the minimum total number of switches for the given sequence with at most UB switches between any successive job pair. The mixed-integer model applied during this step is given in Appendix B. Whenever there is an improvement with respect to the min–sum objective we substitute sequence π. The procedure stops once all swaps have been evaluated.

4. Evaluation of algorithms

This section evaluates the computational performance of our algorithms. First, we elaborate on our test instances (Section 4.1). Then, we address the performance of the lower bounds (Section 4.2) and compare our heuristic decomposition procedure and our branch-and-bound with off-the-shelf solver Gurobi.

All of our methods have been implemented in C# (using Microsoft Visual Studio 2013) and the tests have been carried out on an Intel Core i7-3770, 3.40 gigahertz PC, with 16 gigabytes RAM. While we allowed Gurobi (version 6.0.0) to use four cores in parallel, our algorithms have not been parallelized.

4.1. Instance generation

As there is no established testbed for the SKU switching problem, we had to generate our own test instances. To do so, we adapt the generation scheme suggested by Crama et al. (1994) for the tool switching problem. Specifically, we analyze four parameter combinations $(n, |S|, \underline{S}, \overline{S})$, where $\underline{S}$ $(\overline{S})$ denotes the minimum (maximum) order length. For each combination, we investigate four different pick face capacities $C \in \{C_1, \dots, C_4\}$. All resulting 16 parameter settings and their specific values are listed in Table 1. For each setting, we repeat instance generation 10 times, so that, in total, we receive 160 instances. To obtain an instance, we apply the following generation scheme:

Table 1
Parameters for instance generation.

n	$\|S\|$	$\underline{S}$	$\overline{S}$	C_1	C_2	C_3	C_4
10	20	2	4	4	5	6	7
15	40	2	6	6	8	10	12
30	80	5	15	15	17	20	25
40	120	7	20	20	22	25	30

Table 2
Performance of the lower bounds.

	LB^1	LB^2	LB^3
best	63	15	155
best*	1	2	93
∅ abs	0.66	2.29	0.05

For each order S_j, $j = 1, \dots, n$, order length $|S_j|$ is determined by drawing a uniformly distributed integer from $\left\{\underline{S}, \dots, \overline{S}\right\}$. Next, $|S_j|$ distinct SKUs are drawn from an integer uniform distribution on $\{1, \dots, |S|\}$, which results in order S_j. After having generated all n orders, initial pick face loading L_0 is determined by drawing C distinct SKUs in the same manner.

Note that other than in Crama et al. (1994) we double the number $|S|$ of considered SKUs to generate instances more suitable in the context of warehousing. Therefore, the high-bay racks ranges from at least three to seven levels.

4.2. Lower bound performance

In our computational study we, first, concentrate on the evaluation of our lower bounds. As elaborated in Section 3.2, the effort for calculating lower bound LB^3 raises exponentially in n, so that the subsets of orders evaluated are restricted by parameter k, i.e., $LB^3 = \max_{h \in J(k)} \left\lceil \frac{l_h}{h} \right\rceil$ to $J(k) = \{1, \dots, , k, n-k, \dots, n\}$. Therefore, we want to investigate the influence of parameter k on bound quality and computational time. In this context, the left-hand side of Fig. 5 illustrates the improvement of LB^3 depending on the value of k. Here, $\Delta LB^3 = \frac{LB^* - LB^3}{LB^*}$, where LB^3 is the cumulative bound over all 160 instances and LB^* equals LB^3 in case $k = 6$. Analogously, the right-hand side of Fig. 5 depicts the total computation time of LB^3 over all 160 instances. Fig. 5 reveals that the positive impact of increasing k on the bound quality quickly diminishes. On the other hand, there is an exponential increase of computation time with increasing k. In light of these results, a value of $k = 4$ seems as a good compromise in this trade-off. Thus, we set $k = 4$ for LB^3 and, analogously, $k = 2$ for $LLB^3_{\pi^i}$ (for the local lower bound of our branch-and-bound procedure described in Section 3.3) throughout the computational study. Note that further improvement may be possible if varying k values for different problem sizes n are applied. Specifically, for smaller instances larger k values would be affordable. However, we abstain from such an approach.

Table 2 summarizes the results of all three lower bounds. Row "best" displays the number of instances satisfying $LB^i = LB,\ i = 1, 2, 3,$ whereas "best*" represents the number of exclusive best bounds, i.e., $LB^i < \min_{x, y \neq i}\{LB^x, LB^y\}$. Row "∅ abs" depicts the average absolute deviation from LB. Note that the gaps to the optimal solution values are reported in the next section.

The evaluation shows, that LB^3 performs best. For 155 out of the 160 instances, LB^3 is equal to the best bound obtained and moreover, in 93 cases it exclusively yields the best bound. However, in spite of their simplicity $\min\{LB^1, LB^2\}$ yields the best bound for

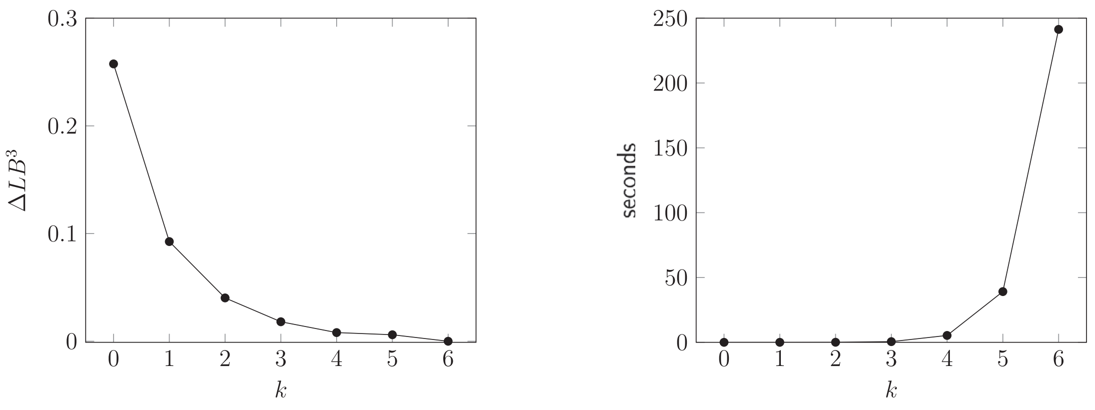

Fig. 5. Performance of lower bound LB^3 depending on k.

Table 3
Performance criteria.

Criteria	Description
opt	Number of optimal solutions
∅ abs	Average absolute deviation from the lower bound
∅ seconds	Average CPU-seconds

67 instances (42%). In addition, the average absolute gap indicates that LB^1 is tight.

4.3. Algorithmic performance

This section explores the solution quality of our tailor-made solution procedures, i.e., the heuristic decomposition approach (dubbed D-MIN–MAX) and our exact branch-and-bound procedure (dubbed BB) initialized with D-MIN–MAX. They are compared to standard solver Gurobi solving the mathematical model given in the Appendix A. Gurobi is executed with a time limit of 1800 CPU-seconds, while we abort BB after 60 seconds. Table 3 defines the performance criteria reported in our performance tests.

Table 4 summarizes the results of our performance tests. The following conclusions can be drawn from these tests:

- Our lower bound LB is tight. Over all 80 (small) instances with $n \leq 15$, which BB was able to solve to proven optimality, the average absolute gap is just 0.2 additional SKU switches. Note that we abstain from reporting relative gaps, because most objective values are rather small, so that only a single additional switch results in a large relative gap.
- Among the exact solution approaches BB clearly outperforms Gurobi. Both were able to solve all 40 instances in case $n = 10$ to proven optimality. In case $n = 15$, only BB was capable of verifying optimality for each of the 40 instances and, moreover, within considerably less computation time. Nevertheless, the solution quality of Gurobi is equal (see column "∅ abs"), but in four instances the standard solver was not able to prove optimality. For the large instances ($n \geq 30$) the gap widens. BB is considerably faster and produces better solutions. Specifically, in 49 out of 80 large instances BB determines a better solution than Gurobi (none the other way round). Recall that the gap to lower bound LB is reported, so that it can be conjectured that the actual gap to the (unfortunately unknown) optimal solutions is even lower.
- The most striking result is certainly the excellent performance of decomposition approach D-MIN–MAX. Even in the largest instances it consumes barely more than a second of computational time and produces solution values only slightly inferior to BB. 70 out of 80 small instances with $n \leq 15$, i.e., 87.5%, are solved to optimality.

It can be concluded that both of our solutions procedures seem well suited for solving the SKU switching problem. While decomposition approach D-MIN–MAX should be sufficient in most practical applications, BB can be applied to (slightly) improve solutions values, if additional computational time is available.

5. Managerial aspects

Beyond the pure computational performance, this section aims to address some managerial aspects. Specifically, we want to investigate whether our (surrogate) optimization objective is indeed a good proxy for the actual objective, which is to minimize the total makespan for picking a given order set. This question is answered in Section 5.2. Moreover, Section 5.3 explores the impact of the size of the pick face on picking performance. This way, warehouse managers receive some decision support on how to appropriately size the pick face of a CSPF system. For both topics, we have to calculate the makespan of order picking resulting from a solution of our SKU switching problem. To do so, we simulate the picker and crane movements along with the resulting blockings among them during executing a derived plan. The setup of this simulation is elaborated first in the following section.

5.1. Setup of simulation

Given a solution for the SKU switching problem, our simulation first generates an initial storage assignment, which locates the SKUs in specific rack positions. Then, we emulate the crane and picker movement while realizing the solution. The result is the makespan for picking the given set of orders.

Storage assignment: To store all $|S|$ SKUs plus having some extra empty slots for realizing switches, the high-bay rack receives $\left\lceil \frac{|S|}{C} \right\rceil + 2$ rows and C columns. Given initial loading L_0 of the pick face, these SKUs are placed in random sequence along the bottom-most row. Then, we randomly fill the upper reserve area with the remaining (passive) SKUs.

Crane movement: A typical crane's horizontal travel speed is $v^c_{hor} = 3$ meter per second, while its vertical speed is $v^c_{ver} = \frac{3}{4}$ meter per second (Tompkins, White, Bozer, & Tanchoco, 2003). Equipped with two independent engines a crane simultaneously moves in horizontal and vertical direction (Boysen & Stephan, 2016). Hence, applying the Chebyshev (or maximum) metric the travel time (in seconds) between two slots $X_1 = (x_1, y_1)$ and $X_2 =$

Table 4
Performance [measured according to the criteria defined in Table 3] of Gurobi, the proposed branch-and-bound (BB) algorithm as well as the our heuristic D-MIN–MAX.

					Gurobi			BB			D-MIN–MAX		
n	$\|S\|$	$\underline{S}$	$\bar{S}$	C	opt	∅ abs	∅ seconds	opt	∅ abs	∅ seconds	opt	∅ abs	∅ seconds
10	20	2	4	4	10	0.0	0.78	10	0.0	0.00	8	0.2	0.00
10	20	2	4	5	10	0.0	1.23	10	0.0	0.00	10	0.0	0.00
10	20	2	4	6	10	0.1	0.86	10	0.1	0.00	10	0.1	0.00
10	20	2	4	7	10	0.1	0.44	10	0.1	0.00	8	0.3	0.00
15	40	2	6	6	9	0.7	309.68	10	0.7	0.13	8	0.9	0.00
15	40	2	6	8	7	0.3	601.01	10	0.3	0.03	7	0.6	0.00
15	40	2	6	10	10	0.0	27.22	10	0.0	0.00	9	0.1	0.00
15	40	2	6	12	10	0.0	205.30	10	0.0	0.00	10	0.0	0.00
				Total	76	0.2	143.31	80	0.2	0.02	70	0.3	0.00
30	80	5	15	15	0	2.6	1800.05	0	2.4	60.19	0	2.4	0.20
30	80	5	15	17	0	2.0	1800.07	0	1.7	60.17	0	1.9	0.18
30	80	5	15	20	1	1.6	1643.82	2	1.2	48.21	1	1.3	0.17
30	80	5	15	25	2	1.1	1517.58	4	0.8	36.17	3	0.9	0.17
40	120	7	20	20	0	6.2	1800.35	0	2.9	61.06	0	2.9	1.10
40	120	7	20	22	0	5.6	1800.09	0	3.0	61.09	0	3.2	1.14
40	120	7	20	25	0	5.0	1800.10	0	2.7	61.12	0	3.0	1.15
40	120	7	20	30	0	2.9	1800.08	0	1.8	61.01	0	2.0	1.04
				Total	3	3.4	1745.27	6	2.1	56.13	4	2.2	0.64

(x_2, y_2) can be calculated by

$$d(X_1, X_2) = d((x_1, y_1), (x_2, y_2)) = \max\left\{\frac{w}{v^c_{hor}} \cdot |x_1 - x_2|, \frac{h}{v^c_{ver}} \cdot |y_1 - y_2|\right\}.$$

We assume that each storage slot has a width of $w = 1$meter and height of $h = 2$meter, so that there is enough space to store a standardized euro-pallet in each slot. Each time a SKU is moved, the crane requires time to pick up the unit load at its initial storage position and to release it at its target position. We assume both times being equal and (in line with typical crane configurations) set $t^c_{pick} = t^c_{put} = 10$ seconds. Switching an active SKU stored at position X with a passive SKU from position Z using the empty slot at Y, requires $d(X, Y) + d(Y, Z) + d(X, Z)$ seconds of travel time. Taking all together a switch concerning the slots X, Y, Z takes

$$d(X, Y, Z) = d(X, Y) + d(Y, Z) + d(X, Z) + 2 \cdot t^c_{put} + 2 \cdot t^c_{pick}$$

seconds. When switching an active SKU at position X with one of the passive SKUs, we apply the following greedy approach to determine the slots Y, Z: among all passive SKUs to be moved into the pick face, we choose the one closest to X, while Y is the empty slot, which minimizes the resulting distance $d(X, Y, Z)$. This way, we successively switch all SKUs to be removed from the pick face from left to right.

Picker movement: We apply $v^p = 3$kilometer per hour $= \frac{5}{6}$meter per second as the speed of the picker, which is a moderate working speed often agreed as an average target with German trade unions. We assume a constant pick time of $t^p_{load} = 20$ seconds per SKU, which includes the time to withdraw the requested number of items and to put them into the bin. Finally, it takes another $t^p_{prep} = 90$ seconds at the beginning of the pick face each time a new order is prepared. In case that there occurs no idle time, i.e., the picker never has to wait for the crane still switching required SKUs, the total time to fulfill the given order set is

$$T^{no\_idle} = \underbrace{n \cdot t^p_{prep}}_{(a)} + \underbrace{\sum_{j \in J} |S_j| \cdot t^p_{load}}_{(b)} + \underbrace{2 \cdot n \cdot C \cdot \frac{w}{v^p}}_{(c)} \qquad (4)$$

seconds. Specifically, the picker prepares each order (a), loads $\Sigma_{j \in J}|S_j|$ SKUs into the bins (b), and walks n times up and down the pick face (c). Unfortunately, switches may cause idle times whenever the crane has not yet switched all demanded SKUs. We denote the total idle time by T^{idle}. Hence, the total processing time (makespan) amounts to

$$T^{total} = T^{no\_idle} + T^{idle}.$$

5.2. Suitability of objective function

The simulation allows us to investigate whether our (surrogate) objective, i.e., minimizing the maximum number of switches per changeover (min–max objective), applied by the SKU switching problem is better suited than the traditional objective of flexible manufacturing, which minimizes the total number of switches (min–sum objective). Furthermore, we apply our multi-objective approach D-LEX (see Section 3.4), which optimizes both objectives in lexicographic order, and ask whether this approach leads to even better solutions. This way the best proxy for the actual objective, which is to reduce the makespan of order picking, can be identified. To do so, we generate solutions for the SKU switching problem with all three procedures, simulate these solutions, and compare the resulting picking performance.

Specifically, the setup of our test is as follows. For each of our 160 test instances, we determine the solutions of the introduced approaches D-MIN–MAX (Section 3.1), D-MIN–SUM, and D-LEX (Section 3.4) and record their objective function values due to the min–max and the min–sum objective. The solutions are handed over to the simulation stage, where for 100 different initial storage assignments the simulation is executed to compute the resulting average idle time T^{idle}. Recall that the other component of the makespan, i.e., execution time T^{no}_idle required to retrieve all items, is independent of the actual switching plan, so that we measure performance in (avoided) idle time.

Table 5 reports on the objective function values min–max and min–sum and the resulting idle times T^{idle} of the approaches D-MIN–MAX and D-MIN–SUM divided by the corresponding values obtained by D-LEX (labeled Δmin–max, Δmin–sum, and ΔT^{idle}), which allows a compact comparison of all three approaches. The following conclusions can be drawn from these results:

- When comparing D-MIN–MAX and D-MIN–SUM, it can be concluded that the expected results with regard to both objectives are confirmed. D-MIN–MAX leads to the lowest maximum number of switches, but requires a larger total number of switches compared to D-MIN–SUM and vice versa. With regard to the picking performance (i.e., measured by the total

Table 5
Performance of our decomposition approaches D-MIN–MAX and D-MIN–SUM, which only address the min–max and the min–sum objective, respectively, in relation to our D-LEX approach which considers both objectives in lexicographic order.

					D-MIN–MAX			D-MIN–SUM		
n	$\|S\|$	$\underline{S}$	$\overline{S}$	C	Δmin–max	Δmin–sum	ΔT^{idle}	Δmin–max	Δmin–sum	ΔT^{idle}
10	20	2	4	4	1.00	1.06	1.80	1.38	0.96	7.80
10	20	2	4	5	1.00	1.17	2.05	1.25	0.95	1.06
10	20	2	4	6	1.00	1.17	7.73	1.16	0.96	8.51
10	20	2	4	7	1.00	1.15	7.94	1.44	0.99	66.15
15	40	2	6	6	1.00	1.08	1.59	1.40	0.97	1.75
15	40	2	6	8	1.00	1.15	3.00	1.50	0.99	3.37
15	40	2	6	10	1.00	1.07	0.97	1.81	0.98	10.07
15	40	2	6	12	1.00	1.12	0.93	1.55	0.95	8.20
30	80	5	15	15	1.00	1.09	2.07	1.72	0.99	1.52
30	80	5	15	17	1.00	1.09	1.14	1.67	0.98	1.42
30	80	5	15	20	1.00	1.11	1.28	1.72	1.00	1.86
30	80	5	15	25	1.00	1.11	1.28	1.88	0.98	5.98
40	120	7	20	20	1.00	1.09	1.52	1.70	0.98	0.94
40	120	7	20	22	1.00	1.12	2.59	1.72	0.98	1.37
40	120	7	20	25	1.00	1.12	2.58	1.58	0.98	1.35
40	120	7	20	30	1.00	1.11	1.11	1.77	0.99	2.38

idle time obtained by the simulation run) D-MIN–MAX outperforms D-MIN–SUM in 11 out of 16 parameter constellations. Thus, in most cases our novel min–max objective indeed seems the better choice than the traditional min–sum objective. Only if there is a large diversity of SKUs, i.e., $|S| = 120$, the traditional min–sum objective shows superior. In these cases, plenty switches are required between any pair of successive orders. This, in turn, increases the chance that the picker arrives with the next picking order way before the crane has executed the maximum number of required switches. In these settings, the system is rather unbalanced, the picker consistently has to wait for a heavily occupied crane, and minimizing the sum of switches turns out as the better objective. Contrariwise, the min–max objective is the right choice in balanced systems where the optimal min–max objective function value is comparatively low. Then, the crane has a good chance to timely execute all switches prior to the arrival of the picker.

- As both objectives have shown their appropriateness for different parameter settings, it is not astounding that multi-objective approach D-LEX improves the solution quality and finds suited solution for all settings. By construction of D-LEX, there is no improvement in terms of Δmin–max compared to D-MIN–MAX, whereas there is a significant reduction of the total number of switches (Δmin–sum). On contrary, the switching plans due to D-LEX require considerably fewer maximum switches per changeover than D-MIN–SUM, while it typically just executes a few more switches in total. Altogether, D-LEX provides well suited solutions for both objectives. Concerning the idle times, it is to observe that D-LEX considerably improves the results of the other two approaches each considering one objectives in isolation. Only in two parameter settings D-LEX leads to a slightly lower performance than D-MIN–MAX, whereas D-MIN–SUM finds slightly better results in merely a single setting. In all other cases, D-LEX leads to much lower idle times and, in the very extreme, D-MIN–MAX and D-MIN–SUM result in idle times being 8 and 66 times higher, respectively.

It can be concluded that for CSPF systems our min–max objective is often better suited than the traditional min–sum objective of flexible manufacturing. However, considering both objectives in lexicographic order leads to even better results, so that applying solution approach D-LEX seems a good choice for real-world CSPF systems.

Additional advice is derived from Fig. 6 where we compare the results of multi-objective approach D-LEX with a typical real-world solution policy. That is, incoming orders are successively considered in their given arrival sequence and switches are performed in the KTNS manner. Specifically, we apply the basic parameter setting of our largest instances with $n = 40$, $|S|$=120, $\underline{S} = 7$, $\overline{S} = 20$ and vary the pick face capacity $C \in \{20, \dots, 30\}$. For each C, we generate 10 instances solved by D-LEX and for each instance, we simulate 10 different initial storage assignments and report on the makespan $T^{total,*}$. Furthermore we generate 100 random sequences solved by KTNS and determine the average makespan T^{total}. Thus, we obtain the time savings TS in percent by $\frac{T^{total} - T^{total,*}}{T^{total,*}} \cdot 100$.

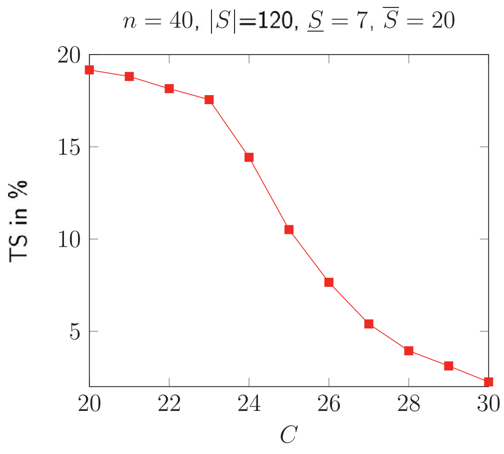

Fig. 6. Time savings (TS) of D-LEX in comparison with a real-world solution procedure depending on pick face capacity C.

Fig. 6 reveals that significant time savings up to 20% can be realized by our tailored-made optimization procedure compared to a simple real-world solution approach. With increasing pick face capacity C, however, fewer switches have to be executed between two successive orders, so that the probability of picker idle time decreases. Therefore, fewer and fewer picker blockings can be avoided by optimization and the fraction of idle times on the makespan diminishes. For our parameter setting a capacity of $C =$ 30 is large enough, so that no significant time savings can be realized anymore. Note, however, that a larger capacity C also increases the walking distances of the picker along the pick face, so in spite of fewer blockings the makespan still may increase. The impact of pick face capacity C is further analyzed in the following section.

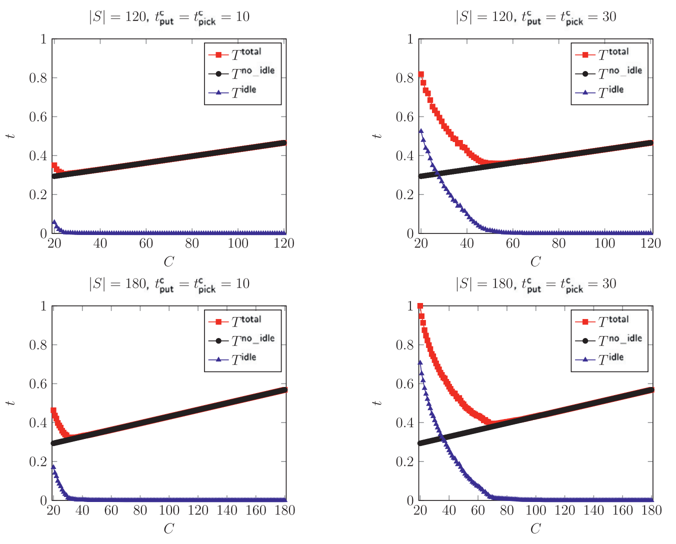

Fig. 7. Picking performance depending on pick face capacity C.

5.3. Sizing the pick face

When dimensioning pick face capacity C warehouse managers face the following trade-off. A smaller pick face reduces the walking distance for the picker, so that the makespan is positively influenced by reducing C. On the other hand, in a small pick face there is not much room for “stand-by SKUs”, which are required again every now and then. Thus, a smaller pick face tends to increase the number of switches and reduces the picker’s walking time, so that the time span for the crane to timely execute all required switches prior to the new arrival of the picker decreases. This, in turn, increases the probabilities of picker blockings, so that additional idle time negatively impacts the makespan. This section is dedicated to this trade-off, so that warehouse managers get some decision support when having to size a pick face.

For this purpose, we solve each problem instance with D-LEX when applying differently sized pick face capacities C, simulate each solution, and compare the resulting makespans. Specifically, we apply the basic parameter setting of our largest instances (see Section 4.1) with $n = 40$, $\underline{S} = 7$, and $\overline{S} = 20$. We vary parameter $|S| \in \{120, 180\}$ and generate 10 instances, i.e., orders S_j $(j = 1, \dots, n)$ and an initial pick face L_0^{C-1} $(|L_0^{C-1}| = C - 1)$. For each $C \in \{\overline{S}, \dots, |S|\}$, we add a randomly chosen SKU $r \in S \setminus L_0^{C-1}$ to the pick face $(L_0^C = L_0^{C-1} \cup \{r\})$ and solve this instance with D-LEX. The solutions are handed over to the simulation stage, where for 10 different initial storage assignments the simulation is executed (once with $t^c_{\text{pick}} = t^c_{\text{put}} = 10$ seconds and once with $t^c_{\text{pick}} = t^c_{\text{put}} = 30$ seconds). Fig. 7 illustrates the resulting idle time T^{idle}, processing time $T^{\text{no\_idle}}$, and makespan T^{total} depending on pick face capacity C. Four different settings are plotted: On the right-hand side, we increase the time it takes the crane to pick and retrieve a pallet, i.e., $t^c_{\text{pick}} = t^c_{\text{put}} = 30$ seconds a pallet, so that we have a slower crane. In the bottommost cases, we increase the total number $|S| = 180$ of SKUs, so that we have a larger variety of SKU demands among picking orders. Note that to facilitate comparability we have scaled the ordinates’ axes.

The following main findings can be derived from these plots:

- Within all four settings our basic trade-off becomes evident. The idle time of the picker waiting for the SKU switches of the crane reduces with increasing pick face capacity C, until it reaches zero. On the other hand, the pure processing time $(T^{\text{no\_idle}})$ increases linear with C, because the walking distance constantly increases with every additional slot. In the aggregate, the minimum makespan is reached once capacity C is large enough, so that (nearly) no idle time occurs.
- If a faster crane is applied (left hand side of Fig. 7), the pick face should be reduced in size. An identical amount of switches can be processed faster, so that picker idle time becomes less likely.
- If the picking orders demand a larger variety of SKUs (bottommost plots of Fig. 7), more switches need to be executed and the pick face has to be increased in size.

When dimensioning a real-world CSPF system such a plot should be derived for the respective crane configuration and picking order structure, so that an appropriate pick face capacity can be derived. However, at the point in time a CSPF system is erected it might be hard to anticipate representative sets of picking orders. In case of doubt, the plots of Fig. 7 suggest to rather dimension capacity C a bit larger than too small, because idle time takes a steep increase whereas the increase of the walking distance only moderately impacts picking performance.

6. Conclusion

This paper introduces the SKU switching problem, which occurs in warehouses applying a crane-supplied pick face (CSPF) system. An automated crane has to move SKUs from the reserve area of a high-bay rack into the bottommost pick face, so that human order pickers can assemble picking orders. We aim to derive order sequences and SKU switching plans, such that the maximum number of switches between any pair of successive orders is minimized. We suggest a heuristic decomposition approach and an exact branch-and-bound procedure for solving this problem. Furthermore, a simulation study is executed to explore the impact of SKU switching solutions on the picking performance. Our computational study reveals that our novel min–max objective outperforms the traditional min–sum objective in most cases. However, combining both objectives in a multi-objective approach leads to even better results and considerable improvements compared to simple real-world decision rules are gained. Finally, we show that a properly sized pick face significantly contributes to efficient picking processes.

Future research could challenge our solution procedures, so that even better solutions can be gained. Furthermore, there exist large CSPF systems where multiple cranes and pickers cooperate (see Kim et al., 2003). Adapting our SKU switching problem to these multi-resource settings seems a challenging task. Finally, the exact positions of the SKUs both in the pick face and the reserve area impact the picker’s waiting time, so that a holistic SKU switching problem also integrating the crane movement could be a valid contribution.

Acknowledgment

We are grateful to three anonymous referees for their valuable comments, which helped to improve the quality of the paper.

Appendix A. Mixed-integer model for the SKU switching problem

For the tool switching problem there exist the mixed-integer models of Tang and Denardo (1988a) and Laporte et al. (2004). As elaborated in Laporte et al. (2004), the problem of the former model is its weak LP-relaxation, so that the latter model takes advantage of the correspondence to the traveling salesman problem in order to design a stronger model. However, as discovered by the authors themselves even if the LP-relaxation is superior, it is only practicable for small instances due to the exponential number of

Table 6
Notation.

J	Set of picking orders (with $J = \{1, \ldots, n\}$)
S	Set of SKUs
J_s	Set of picking orders that require SKU s
C	Pick face capacity
$v_{0,s}$	Binary parameter: 1, if $s \in L_0$, that is SKU s is in the initial loading; 0, otherwise
$u_{j,i}$	Binary variable: 1, if order j is processed at sequence position i; 0, otherwise
$v_{i,s}$	Binary variable: 1, if SKU s is active during sequence position i; 0, otherwise
$w_{i,s}$	Binary variable: 1, if SKU s becomes active in sequence position i; 0, otherwise
z	Integer variable: maximum number of switches between any pair of successive orders

subtour elimination constraints. Hence, we decided to adapt the model of Tang and Denardo (1988a) for our tool switching problem. Applying the notation summarized in Table 6 the model consists of objective function (5) subject to constraints (6)–(13).

$$\text{Minimize } F(u, v, w) = z \tag{5}$$

$$\sum_{i \in J} u_{j,i} = 1 \qquad j \in J \tag{6}$$

$$\sum_{j \in J} u_{j,i} = 1 \qquad i \in J \tag{7}$$

$$\sum_{j \in J_s} u_{j,i} \le v_{i,s} \qquad i \in J;\ s \in S \tag{8}$$

$$\sum_{s \in S} v_{i,s} \le C \qquad i \in J \tag{9}$$

$$v_{i,s} - v_{i-1,s} \le w_{i,s} \qquad i \in J;\ s \in S \tag{10}$$

$$\sum_{s \in S} w_{i,s} \le z \qquad i \in J \tag{11}$$

$$u_{j,i} \in \{0, 1\} \qquad j, i \in J \tag{12}$$

$$v_{i,s}, w_{i,s} \in \{0, 1\} \qquad i \in J;\ s \in S \tag{13}$$

Objective function (5) minimizes the maximum number of switches, which is determined by inequalities (10) and (11). Constraints (6) and (7) ensure a one-to-one mapping between orders and sequence positions. While constraints (8) guarantee that an order can be processed at position i only if the required SKUs s are active, the pick face capacity is ensured by constraints (9). Finally, the domains of the variables are set by (12) and (13).

Appendix B. Mixed-integer model for the D-LEX

In this appendix we specify the mixed-integer program applied in multi-objective procedure D-LEX (see Section 3.4). For a given maximum number of switches between any successive job pair M and a sequence $\bar{u}$ where $\bar{u}_{j,i} = 1$, if order j is processed at sequence position i (0 otherwise) the following model minimizes the number of tool switches (14) subject to constraints (15)–(19) applying the notation summarized in Table 6.

$$\text{Minimize } F(v, w) = \sum_{i \in J} \sum_{s \in S} w_{i,s} \tag{14}$$

$$\sum_{s \in S} w_{i,s} \le M \qquad i \in J \tag{15}$$

$$\sum_{j \in J_s} \bar{u}_{j,i} \le v_{i,s} \qquad i \in J;\ s \in S \tag{16}$$

$$\sum_{s \in S} v_{i,s} \le C \qquad i \in J \tag{17}$$

$$v_{i,s} - v_{i-1,s} \le w_{i,s} \qquad i \in J;\ s \in S \tag{18}$$

$$v_{i,s}, w_{i,s} \in \{0, 1\} \qquad i \in J;\ s \in S \tag{19}$$

Constraints (16)–(19) directly correspond to constraints (8)–(10) of the previous MIP, while constraints (15) ensure the condition of at most M switches per changeover.

References

Bard, J. F. (1988). A heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions, 20*, 382–391.

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

Buzacott, J. A., & Yao, D. D. (1986). Flexible manufacturing systems: A review of analytical models. *Management Science, 32*, 890–905.

Chaves, A. A., Lorena, L. A. N., Senne, E. L. F., & Resende, M. G. C. (2016). Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Computers & Operations Research, 67*, 174–183.

Crama, Y., & Oerlemans, A. G. (1994). A column generation approach to job grouping for flexible manufacturing systems. *European Journal of Operational Research, 78*, 58–80.

Crama, Y., Oerlemans, A. G., & Spieksma, F. C. (1994). Minimizing the number of tool switches on a flexible machine. *The International Journal of Flexible Manufacturing Systems, 6*, 33–54.

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.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability*. New York: Freeman.

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.

Hertz, A., Laporte, G., Mittaz, M., & Stecke, K. E. (1998). Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE Transactions, 30*, 689–694.

Kim, B. I., Heragu, S. S., Graves, R. J., & Onge, A. S. (2003). Clustering-based order-picking sequence algorithm for an automated warehouse. *International Journal of Production Research, 41*, 3445–3460.

Konak, A., Kulturel-Konak, S., & Azizoğlu, M. (2008). Minimizing the number of tool switching instants in flexible manufacturing systems. *International Journal of Production Economics, 116*, 298–307.

Kouvelis, P. (1992). Design and planning problems in flexible manufacturing systems: A critical review. *Journal of Intelligent Manufacturing, 3*, 75–99.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society, 7*, 48–50.

Laporte, G., Salazar-Gonzalez, J. J., & Semet, F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions, 36*, 37–45.

Ramtin, F., & Pazour, J. A. (2014). Analytical models for an automated storage and retrieval system with multiple in-the-aisle pick positions. *IIE Transactions, 46*, 968–986.

Ramtin, F., & Pazour, J. A. (2015). Product allocation problem for an AS/RS with multiple in-the-aisle pick positions. *IIE Transactions, 47*, 1379–1396.

Tang, C. S., & Denardo, E. V. (1988a). Models arising from a flexible manufacturing machine, part I: Minimization of the number of tool switches. *Operations Research, 36*, 767–777.

Tang, C. S., & Denardo, E. V. (1988b). Models arising from a flexible manufacturing machine, part II: Minimization of the number of switching instants. *Operations Research, 36*, 778–784.

Tompkins, J., White, J., Bozer, Y., & Tanchoco, J. (2003). *Facilities planning* (3rd). John Wiley & Sons.

Yaman, H., Karasan, O. E., & Kara, B. Y. (2012). Release time scheduling and hub location for next-day delivery. *Operations Research, 60*, 906–917.