Computers & Industrial Engineering 147 (2020) 106645

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses

Patrick Kübler[a,*,1], Christoph H. Glock[b,2], Thomas Bauernhansl[c,3]

[a] Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA, Universität Stuttgart, Nobelstraße 12, 70569 Stuttgart, Germany
[b] Fachgebiet Produktion und Supply Chain Management, Technische Universität Darmstadt, Hochschulstraße 1, 64289 Darmstadt, Germany
[c] Institut für Industrielle Fertigung und Fabrikbetrieb IFF, Universität Stuttgart, Allmandring 35, 70569 Stuttgart, Germany

ARTICLE INFO

Keywords:
Order picking
Dynamic order pattern
Picker routing
Order batching
Storage location assignment
Particle swarm optimization

ABSTRACT

Short and reliable delivery lead times are crucial for many buying decisions, making efficiently operated order picking systems a critical contributor to a company's competitiveness. To pick orders fast and with minimal effort, three planning problems need to be solved, namely the assignment of items to storage locations, the consolidation of orders in batches and the routing of the order pickers through the warehouse. Even though the problems are strongly interdependent, they have so far largely been solved separately, leading to losses in efficiency. Recent research has shown that a joint solution can lead to significant performance improvements as compared to individual solutions. Up to now, no method is available that solves all three problems jointly. This work contributes to closing this research gap by proposing an iterative heuristic method that solves the problems jointly and that takes account of future dynamics in customer demand and their influence on the three planning problems. The performance of the method is illustrated in numerical experiments. The results of our studies indicate that the method may lead to significant savings in travel distance.

1. Introduction

Order picking is the most costly warehousing activity (Marchet, Melacini, & Perotti, 2015). A well-organized order picking system that ensures short item retrieval and, therewith, short order lead times, allows a high responsiveness to last-minute orders and short delivery times to customers (De Koster, Le-Duc, & Roodbergen, 2007). Underperformance may result in high labor costs and unsatisfied customer demand (Wruck, Vis, & Boter, 2017). Improving order picking efficiency may therefore lead to a higher service level and to lower labor costs (Cergibozan & Tasan, 2019).

Due to rapidly changing customer preferences, warehouses face high product varieties, small order sizes, a fast-changing product mix, and a highly volatile customer demand (Boysen, de Koster, & Weidinger, 2018). Order picking has become even more important in this environment (Menéndez, Bustillo, Pardo, & Duarte, 2017).

In light of these developments, this paper studies the management of order picking operations in manual picker-to-parts-systems, which account for the majority of order picking systems in warehouses worldwide (De Koster et al., 2007). According to Henn, Koch, and Wäscher (2012), three planning problems occur in such warehouses: the storage location assignment problem, the order batching problem, and the picker routing problem. Even though these problems are strongly interdependent, they are traditionally solved separately, leading to suboptimal solutions. This paper therefore develops an iterative solution method for solving the three problems jointly because recent research has shown that a joint solution leads to significant performance improvements as compared to separate solutions (Van Gils, Ramaekers, Caris, & de Koster, 2018).

A few authors have solved two planning problems jointly (cf. Section 2.2 for a review of related research). However, to the best of our knowledge, no method that solves all three planning problems jointly has been proposed so far. This paper contributes to the literature on order picking as follows: First, it explicitly takes account of the strong

---

* Corresponding author.
E-mail addresses: patrick.kuebler@ipa.fraunhofer.de (P. Kübler), glock@pscm.tu-darmstadt.de (C.H. Glock), thomas.bauernhansl@iff.uni-stuttgart.de (T. Bauernhansl).
[1] ORCID ID: 0000-0002-3898-0230.
[2] ORCID ID: 0000-0001-6006-0070.
[3] ORCID ID: 0000-0001-5768-2055.

https://doi.org/10.1016/j.cie.2020.106645
Received 6 August 2019; Received in revised form 6 April 2020; Accepted 2 July 2020

interdependence of the three planning problems in an iterative solution approach. Secondly, previous solution approaches for the dynamic storage location assignment problem do not consider future customer demand or the efforts associated with relocating items in low-level picker-to-parts systems. In our solution approach, the expected future benefits of a relocation of an item are calculated with the help of a forecasting method and a newly developed estimation procedure. Our procedure only relocates items if the expected future benefits exceed the relocation efforts. Thirdly, we propose a novel heuristic algorithm based on an improved version of particle swarm optimization (PSO) for solving the joint order batching and picker routing problem. We choose PSO as it computes good results with short computation times, uses only primitive mathematical operators, is conceptually very simple (Das, Abraham, & Konar, 2008) and thus easy to implement in practice. This algorithm is also used to evaluate the benefit of an eventual relocation of items triggered by changes in customer demand. Fourthly, most of the existing solution approaches are theoretically complex and validated only with the help of small problem instances. Both may make practitioners reluctant to implement such methods. We validate our method with the help of large-scale problems and choose heuristics with an acceptable mathematical complexity to ensure the applicability of our method.

The three planning problems are investigated in a low-level picker-to-parts order picking systems (see Assumption 11 in Section 4.1) with parallel aisles of equal length connected by an arbitrary number of cross aisles (often referred to as a multi-block layout), where order sizes are small and customer demand is highly volatile.

The remainder of this paper is organized as follows: Section 2 reviews the related literature. Section 3 describes the problem investigated in this paper. Section 4 formulates a mixed-integer programming model for the dynamic storage location assignment problem. Section 5 develops a two-stage hybrid algorithm integrating PSO and a 2-opt-algorithm to solve the joint order batching and picker routing problem. This algorithm is integrated into a dynamic storage location assignment algorithm. Section 6 presents the results of a numerical experiment. Finally, Section 7 concludes the paper and presents some suggestions for future research.

2. Literature review

This section gives a brief overview of the related literature. Comprehensive literature reviews on warehouse planning are those of Rouwenhorst et al. (2000), De Koster et al. (2007), Gong and De Koster (2011), Davarzani and Norrman (2015), De Koster, Johnson, and Roy (2017) and Grosse, Glock, and Neumann (2017), for example.

2.1. Planning problems

To ensure efficient order picking operations, several planning problems need to be solved. We give an overview of these planning problems in the following. The basics introduced in the following are needed for developing the proposed solution method in later sections of this paper. **Storage location assignment** policies allocate items to storage locations in the warehouse (Grosse, Glock, & Jaber, 2013; Pan, Shih, & Wu, 2012). Common policies are random storage, dedicated storage and class-based storage (Hausman, Schwarz, & Graves, 1976). In random storage, incoming items are randomly assigned to empty storage locations, whereas in dedicated storage, each item has a fixed storage location that is left empty if the item is out of stock. In class-based storage, items are assigned to classes based on item characteristics, and each class is assigned a fixed set of storage locations. Storage within the classes is random, though (Muppani & Adil, 2008). Random storage leads to the longest expected travel distance but requires less space, while dedicated storage has the shortest expected travel distance at the expense of low space utilization. If the demand for a product fluctuates, then the optimal storage location of this item changes as well, which makes it necessary to relocate the item. In this case, dedicated storage assignment policies lead to a high maintenance effort. Class-based storage results in average travel distances that are between those of random and dedicated storage (Rao & Adil, 2013). It is widely used in practice because it can easily handle assortment changes or changes in pick frequencies (Le-Duc & De Koster, 2005). Most publications focus on the static storage location assignment problem. Thus, the impact of fluctuating order patterns, the effort for implementing a new assignment or relocations are not considered (Kofler, Beham, Wagner, & Affenzeller, 2015). For this reason, we concentrate on dynamic class-based storage in this paper.

Given a set of released orders, **order batching** partitions the set of orders into sub-sets, so-called batches (Gu, Goetschalckx, & McGinnis, 2007). Items belonging to multiple orders can then be picked in a single pick tour (see Assumption 6 in Section 4.1) to reduce the average travel time per order (Van Nieuwenhuyse & de Koster, 2009). Order batching is a bin packaging problem (Gu et al., 2007), and for more than two orders, it is in the class of NP-hard problems (Gademann & Velde, 2005). The literature discusses two methods for order batching: proximity batching and time-window batching. Proximity batching assigns orders to batches based on the proximity of storage locations that need to be visited. In time-window batching, all orders that arrive during the same time interval are grouped together in one batch (De Koster et al., 2007). In this paper, we focus on proximity batching and assume that all customer orders that need to be completed in the time period of interest are known in advance (see Assumption 8 in Section 4.1). When multiple orders are picked together, an additional sorting process is necessary. Sorting can be done either during the picking process (sort-while-pick) or after the picking process (sort-after-pick) (Gu et al., 2007). Sort-while-pick is common in many warehouses to avoid that the items need to be handled more than once. We assume sort-while-pick and that customer orders may not be split over different batches (see Assumption 3 in Section 4.1).

Picker routing determines the sequence in which the item locations contained in an order or a batch should be visited such that the total travel distance is minimized (Wäscher, 2004). This problem is a special case of the Travelling Salesman Problem (TSP), and it is referred to as the Steiner TSP. The aim is to find a minimum-length Steiner tour, where each non-Steiner node is visited at least once (Theys, Bräysy, Dullaert, & Raa, 2010). In a recent paper, Cambazard and Catusse (2018) developed a dynamic programming approach that solves the Steiner TSP for any rectangular warehouse with h cross-aisles, but that is exponential in h. In another paper, Pansart, Catusse, and Cambazard (2018) presented a sparse formulation in mixed-integer programming strengthened by preprocessing that can easily accommodate side constraints. In addition, they extended a dynamic programming approach to the case of an arbitrary number of cross aisles. For a detailed review of the order picker routing problem, the reader is referred to Masae, Glock, and Grosse (2020).

2.2. Joint optimization of order batching and picker routing

In the following, we discuss works with a specific focus on the joint optimization of order batching and picker routing.

Won and Olafsson (2005) formulated the batching and picker routing problem jointly as a combinatorial optimization problem. The picker routing problem investigated by the authors differs from the problem studied in this paper as they assumed that pickers (retrieval units) have a dedicated aisle and travel along the aisle vertically and horizontally at the same time. Tsai, Liou, and Huang (2008) proposed a batch-picking model that considers travel costs and an earliness and tardiness penalty. The batching problem was solved by a genetic algorithm that aims on minimizing total cost, and by a genetic algorithm that tries to construct a travel path with minimum length. In contrast to our paper, the authors permitted the splitting of orders over batches. This simplifies the order batching problem (and may lead to better optimization results), but leads to additional sorting efforts since items need to be assigned to customer orders at the end of the picking tour. The algorithms were developed for a one-block layout. Similarly, Ene and Öztürk (2012) developed a genetic algorithm for jointly solving the order batching and picker routing problems. In addition, they proposed a class-based storage location assignment policy. In contrast to the approach presented in this paper, only incoming items were considered. The authors assumed a dedicated storage location assignment without relocating items over time. Even though order batching and picker routing were solved jointly, the storage location assignment problem was solved separately. Kulak, Sahin, and Taner (2012) formulated a mixed-integer optimization model of the joint order batching and picker routing problem. The objective was to minimize the total travel distance over all batches and tours. Unlike most common approaches, the routing problem was modeled as a classical TSP such that any heuristic for the classical TSP can be used to solve the problem. To solve the picker routing problem, the authors used a nearest neighbor and an or-opt as well as a savings and a 2-opt heuristic. To solve the order batching problem, they developed a seed algorithm to create the initial solution for a tabu search algorithm. Azadnia, Taheri, Ghadimi, Mat Saman, and Wong (2013) proposed a solution approach for the joint order batching and picker routing problem in a single-block layout that aims on minimizing the average tardiness of all orders. Weighted associated rule mining was used to calculate associations between orders, where the latter is based on the similarity of the items with respect to their due dates. Genetic algorithms were used to solve the TSP and to sequence the batches. Grosse, Glock, and Ballester-Ripoll (2014) analyzed an order batching problem where orders are permitted to be split over batches. They solved the joint order batching and picker routing problem sequentially by first splitting orders into batches that can be picked in a single tour. The solution was improved by using a simulated annealing algorithm. Matusiak, de Koster, Kroon, and Saarinen (2014) considered a warehouse with precedence constraints of the products including potential multiple drop-off points in a route. The solution procedure consists of two approaches: a simulated annealing algorithm that estimates the savings gained from batching more than two customer orders, and an A*-algorithm that computes the travel tours. Chen, Cheng, Chen, and Chan (2015) studied the integrated order batching, sequencing and routing problem with the objective to minimize the total tardiness of customer orders. A genetic algorithm was used to solve the order batching and batch sequencing problem, and an ant colony optimization algorithm was used to solve the picker routing problem. Cheng, Chen, Chen, and Yoo (2015) used the model of Kulak et al. (2012) and first employed a PSO algorithm to determine the pick sequence of batches. Subsequently, ant colony optimization was used to calculate the shortest picking tour. The authors compared the solution quality obtained by their algorithm against the method developed by Tsai et al. (2008) and showed that PSO outperforms the genetic algorithm. Lin, Kang, Hou, and Cheng (2016) presented an improved version of PSO that is used for simultaneously solving the order batching and picker routing problem. Menéndez et al. (2017) solved the order batching and sequencing problem in a single-block warehouse using general variable neighborhood search for the case where each order has a certain due date. Scholz and Wäscher (2017) integrated different routing algorithms into an iterated local search approach to solve the joint order batching and picker routing problem in a multi-block layout. Scholz, Schubert, and Wäscher (2017) dealt with the joint order batching, batch sequencing and picker routing problem and assumed in addition that batches need to be assigned to pickers. Valle, Beasley, and da Cunha (2017) presented an integer programming formulation for the joint order batching and picker routing problem based on an exponential number of connectivity constraints. Van Gils, Caris, Ramaekers, and Braekers (2019) solved the order batching, picker routing and picker scheduling problems with an iterated local search algorithm. A real-life case showed significant performance benefits gained from integrating the three planning problems. Briant et al. (2020) proposed an exponential linear programming formulation of the joint order batching and picker routing problem where variables are related to single picking routes. The solution approach was designed to provide accurate lower and upper bounds.

2.3. Dynamic storage location assignment

The second stream of research that is relevant to this paper studies the dynamic storage location assignment problem. Since we focus on manual picker-to-part systems, approaches developed for automated systems or reserve storage areas such as those of Muralidharan, Linn, and Pandit (1995), Moon and Kim (2001) or Ang, Lim, and Sim (2012) are not discussed in the following.

A related paper is the one of Grosse et al. (2013), who investigated when to change a storage location assignment in situations where warehouse workers improve their performance over time as a result of learning, but where changes in customer demand may make adjustments in the storage assignment necessary. A change in the storage assignment would make it necessary for the worker to learn the locations of items anew. For the special case of a carton flow rack, Sadiq, Landers, and Don Taylor (1996) designed a dynamic stock location assignment algorithm that aims on minimizing the sum of order picking time and order picking system re-warehousing time. The storage configuration considered was an in-the-aisle order picking system where items can occupy flexibly-sized storage locations. Splitting stock among zones was permitted. The algorithm is run periodically and attempts to revise the assignments of storage locations when the stock mix changes. To assign stock to slots, the algorithm considers demand forecasts. Pierre, Vannieuwenhuyse, Dominanta, and Van Dessel (2004) described a dynamic variant of the traditional ABC storage policy, which can be used in manual order picking warehouses where items experience a rather unstable demand. This approach first classifies items into three classes according to their contribution to the total number of orderlines. In contrast to static variants of the ABC storage policy, the classification is reviewed on a short-term basis based on the past evolution of the number of orderlines per item and per day. Every item that is currently stored in the wrong class receives a priority. The priority is used to identify items that may lead to the highest efficiency gains if relocated. Kofler et al. (2015) extended this approach by permitting alternative storage paradigms beyond the ABC classification. Manzini, Accorsi, Gamberi, and Penazzi (2015) formulated two cost-based mixed integer linear programming models. The first model addresses the technology selection and the storage capacity of each class. The second model selects optional subsystems. In both models, the items are dynamically assigned to the classes. The authors did not propose a solution method and did not use forecast data. Li, Moghaddam, and Nof (2016) investigated a dynamic storage location assignment problem where incoming items need to be assigned to storage locations. We added a figure to Appendix A to highlight the contribution of our paper by comparing it to the discussed papers.

3. Problem statement

According to Tompkins, White, Bozer, and Tanchoco (2010), the time the order picker spends on travelling through the warehouse accounts for around 50% of the total picking time. Thus, as the majority of studies (Van Gils et al., 2018), we focus on reducing the total travel distance for a given number of orders (see Assumption 1 in Section 4.1). We concentrate on manual low-level, picker-to-parts systems, as these systems are most frequent in practice. In these systems, order pickers drive or walk along the aisles of the warehouse to pick items from storage locations (De Koster et al., 2007). If there are only two cross aisles located at the front and at the rear of the warehouse, the warehouse is usually referred to as a one-block warehouse. Introducing additional cross aisles generates a multi-block layout (Wäscher, 2004). Today, most companies use multi-block layouts, as these make warehouse operations more efficient (Kulak et al., 2012). Therefore, this paper also investigates a multi-block layout (see Assumption 2 in Section 4.1). We assume that at the start of the picking process, a set of customer orders is available. Each order consists of a number of orderlines, with each line representing an item and the corresponding quantity requested by the customer (Wäscher, 2004). We note that other factors, such as precedence constraints, batch sequences, order due times, worker learning or picker blocking (see Assumption 5 in Section 4.1) can also be important for a particular picking scenario. We point out in Section 7 that these aspects could be considered in future research.

A recent study of Van Gils et al. (2018) emphasized the strong interdependence between the three planning problems. Petersen and Aase (2004), Ho and Tseng (2006) and Ho, Su, and Shi (2008) showed that a statistically significant interdependence between the storage location assignment and the order batching problems exists. The interdependence between the storage location assignment and the picker routing problem was highlighted, among others, by Petersen and Schmenner (1999), Manzini, Gamberi, Persona, and Regattieri (2007), Theys et al. (2010), and Shqair, Altarazi, and Al-Shihabi (2014). A joint solution of the order batching and picker routing problem can lead to significant performance improvements as compared to separate solutions (Van Gils et al., 2018). Despite the strong interdependence between both planning problems, they are traditionally solved separately (De Koster et al., 2007), even though this usually leads to losses in efficiency (Hsieh & Huang, 2011). In fact, only a simultaneous solution of the three problems could result in an optimal solution for the system. However, for practical problems, this is not a realistic approach (Wäscher, 2004), first because of the mathematical complexity of the resulting optimization problem, and secondly because of the different planning horizons of the three problems (the batching and routing problems usually need to be solved on a daily basis, while the storage location assignment problem is usually not changed on a daily, but instead on a weekly or monthly level). In this paper, we develop an iterative approach for solving the joint dynamic storage location assignment, order batching and picker routing problem.

4. Model development

This section formulates a mathematical model for the dynamic storage location assignment problem. For a formulation of the joint order batching and picker routing problem, we refer to the model developed by Kulak et al. (2012).

4.1. Assumptions

We make the following assumptions:

1. The objective is to minimize the total travel distance for the planning period.
2. Since it is the most common order picking system in practice, we consider a manual picker-to-parts system in a warehouse with parallel aisles and multiple blocks.
3. To avoid additional sorting effort, orders may not be split up into multiple batches.
4. All locations have the same storage capacity.
5. Order pickers can pass through the aisles in both directions. We therefore do not consider picker blocking.
6. Since a batch should be picked in a single tour, the weight of an order is not allowed to exceed the capacity of the picker.
7. Each item type is stored only in one location, and each location contains only a single type of item.
8. We assume that all customer orders of the current planning period are known and that proximity batching is used. The future customer demand is not known in advance.
9. The order picker can retrieve items from the left and right sides in an aisle without significantly changing position. Movements from left to right or vice versa can therefore be neglected.
10. Since we consider a low-level system, vertical movements can be neglected.

Most of these assumptions have already been motivated in the earlier sections. Assumption 9 is commonly made in papers that deal with the picker routing problem (see, for example, Theys et al. (2010), Chen et al. (2015) or Scholz et al. (2017)). Assumptions 4 and 7 are commonly made in papers that deal with the storage location assignment problem (see, for example, Pierre et al. (2004), Kofler et al. (2015) or Li et al. (2016)).

4.2. Storage location assignment and the joint batching and picker routing problem

The static storage location assignment problem can be modeled as a quadratic assignment problem. We extend the model of Reschke (2013) to take account of multiple periods and to account for a relocation effort that depends on the distance between the items' current and future storage locations, an administrative time for updating the information in the IT-system, and the time required for retrieving and restoring the item. The model considers a planning horizon from period (e.g., weeks) t to period u and is solved in a rolling fashion at the end of each new period. When solving the model at the end of period $t-1$, all customer orders have already been picked for that period, and it needs to be decided which items should be relocated before the start of period t. The following nomenclature is used throughout the paper:

Indices and sets:	
$m, n \in A$	Items, including a dummy item assigned to the depot and indexed with "0"
$i, j \in V$	Storage locations, where the depot is represented by "0" and $\lvert V\rvert \geq \lvert A\rvert$
$r \in U$	Time index, where $t, \ldots, u$ are the periods included in the planning horizon
Parameters:	
$d_{i,j}$	Distance between storage locations i and j
$f_{m,n,t}$	Frequency of consecutive picks of items m and n in period t
v^{pick}	Walking speed of an order picker
z^{adm}	Administrative time for relocating one item
z^{phy}	Time for (physically) withdrawing and restoring one item
i_m^{cur}	Current storage location of item m, $i_m^{cur} \in V$
Decision variables:	
$W_{m,i} = 1$	if item m is assigned to storage location i ($=0$, otherwise)

We obtain the following optimization model:

$$\begin{aligned} minZ \\ &= \sum_{m\in A}\sum_{i\in V} W_{m,i}\cdot sign(\lvert i_m^{cur} - \mathrm{i}\rvert)\cdot(d_{i_m^{cur},\mathrm{i}} + v^{pick}\cdot(z^{adm} + z^{phy})) + \\ &\quad \sum_{r\in U}\sum_{m\in A}\sum_{\substack{n\in A\\ n\neq m}}\sum_{i\in V}\sum_{\substack{j\in V\\ j\neq i}} f_{m,n,r}\cdot d_{i,j}\cdot W_{m,i}\cdot W_{n,j} \end{aligned} \tag{1}$$

s.t.

$$\sum_{m\in A} W_{m,i} \leq 1 \quad \forall\, i \in V \tag{2}$$

$$\sum_{i\in V} W_{m,i} = 1 \quad \forall\, m \in A \tag{3}$$

$$W_{m,i} \in 0, 1 \quad \forall\, i \in V, m \in A \tag{4}$$

The objective function (1) minimizes the relocation effort consisting of the travel distance between the current and the future storage locations associated with the relocation of both items, an equivalent for administrative and physical withdrawal efforts, as well as the total travel distance of the order picking process in the future periods $t, \ldots, u$.

The signum function (sign) adopts the value 1 if the future storage location differs from the current one. If $W_{m,i} = 1$, the relocation effort is calculated. If the item is not relocated, the function becomes zero. The second part of Eq. (1) calculates the travel distances for all $t, \ldots, u$ periods under study for the case where item m is assigned to location i and item n is assigned to location j. Constraint (2) ensures that each storage location contains at most one item, while constraint (3) ensures that each item has exactly one storage location. Finally, (4) defines the domains of the variables.

Solving the storage location assignment problem at the end of period $t-1$ is associated with two main problems:

1) The future customer orders for periods $t, \ldots, u$ are usually not available before the start of the respective periods. Therefore, to evaluate a storage assignment at the end of period $t-1$, the future demand needs to be forecasted.
2) The frequencies ($f_{m,n,t}$) with which two items are consecutively picked in periods $t, \ldots, u$ are not known as long as the joint order batching and picker routing problem has not been solved. Hence, we have to solve these problems for periods $t, \ldots, u$ first.

Since the original model of Reschke (2013) is already NP-hard, we develop a heuristic solution method for the problem in Section 5.3.

For the joint order batching and picker routing problem, we refer the reader to Kulak et al. (2012). The picker routing problem in a multi-block warehouse is a Steiner TSP. Travel distances are determined by calculating the length of the shortest path between any two storage locations. The distance between two locations situated in two different blocks or in the same aisle can be calculated with the help of the Manhattan metric. In case two locations are in different aisles of the same block, the length of both possible paths that involve the adjacent cross aisles are calculated, and the shortest one is kept. All distances are saved in a symmetric distance matrix. With the help of this matrix, any of the heuristics for the classical TSP can be used to solve the picker routing problem (Theys et al., 2010). Our contribution to the literature is the development of a novel solution procedure for this problem, which is presented in Section 5.2.

5. Proposed solution approach

5.1. Structure of the proposed solution approach

To solve the three problems jointly, we propose an iterative solution procedure illustrated in Fig. 1. This section first outlines the general structure of the proposed solution approach. The subsequent section then explains the different steps of the solution procedure in detail.

As input data, orders, the weights of the items, the distance matrix, the current storage location assignment, and the capacity of the picker are needed. The left side of Fig. 1 illustrates the joint order batching and picker routing method that can also be applied separately without the storage location assignment algorithm. In the first step of this method, orders are grouped into batches. For each batch, picking tours are constructed and the travel distances are calculated. The distances of the tours are used to evaluate the quality of the corresponding batches. The method then continues with computing batches and routes until a stopping criterion is reached. The picking orders are then released. The right side of Fig. 1 illustrates the dynamic storage location assignment algorithm that starts with forecasting the future orderlines per item. The method then evaluates if items should be relocated and selects new storage locations if required. Using the joint order batching and picker routing algorithm and the customer orders of the current period, the method calculates the travel distance before and after relocating an item. The potential future reduction in travel distance is estimated by a newly developed procedure that will be outlined in detail below. If the future travel distance reduction exceeds the relocation effort, the item in question is relocated, and otherwise it is kept in its current storage location. Relocation options are tested until a stopping criterion is reached, and then the relocation orders are released.

5.2. Discrete evolutionary particle swarm optimization

In the following, we use a PSO procedure to group orders into batches and a 2-opt-algorithm combined with a nearest neighbor algorithm to construct the picker routes. PSO uses only primitive mathematical operators and is conceptually very simple (Das et al., 2008) and thus easy to implement in practice. We choose PSO as earlier research has shown that it works well for mixed-integer programming problems and especially for bin packing problems such as the order batching problem. Önüt, Tuzkaya, and Doğaç (2008) showed that PSO

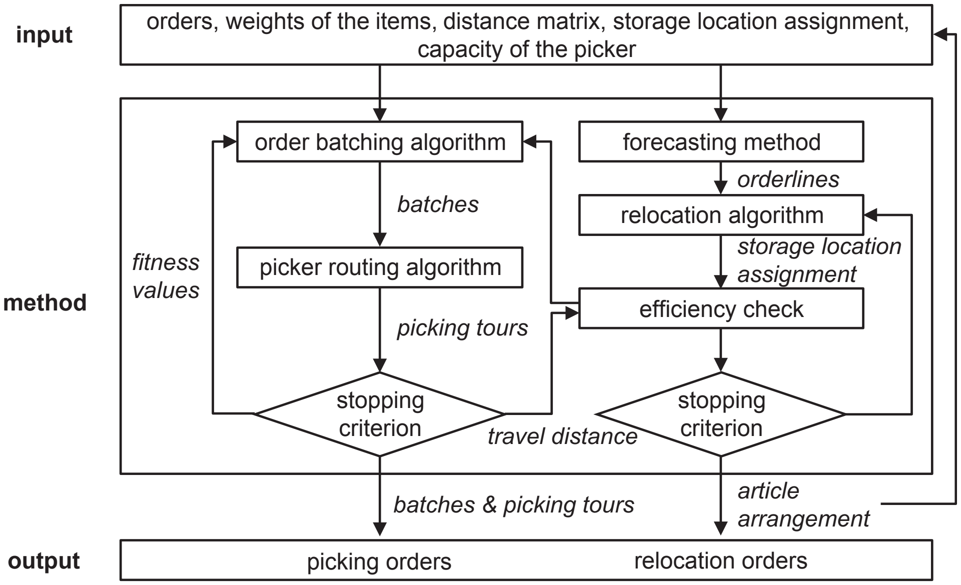


Fig. 1. Structure of the proposed iterative solution approach.

computes very good results in a short time. Hembecker, Lopes, and Godoy (2007) confirmed these results and showed that PSO works well even for large instances. Kashan, Kashan, and Karimiyan (2013) showed that PSO outperforms genetic algorithms in various instances of the bin packing problem. Cheng et al. (2015) developed a PSO algorithm for the order batching problem in manual picker-to-parts warehouses and showed that it provides better results than a genetic algorithm developed by Tsai et al. (2008). Lin et al. (2016) also achieved good results by solving the order batching problem with the help of a PSO. Both, Cheng et al. (2015) and Lin et al. (2016) tested their algorithms with the help of small test instances.

PSO was introduced by Kennedy and Eberhart (1995) for nonlinear functions. In PSO, particles are stochastically generated in the search space. Each particle is a candidate solution to the problem, and it is represented by a location in the search space and a velocity. Each particle has a memory that enables the particle to remember its own and the swarm's previous best position. During the optimization phase, particles move around in the search space. In each iteration, a particle adjusts its position and velocity, based on the best position it has visited so far and the best position found by any other particle. When the algorithm terminates, the best position the swarm reached so far is the solution to the optimization problem (Boussaïd, Lepagnot, & Siarry, 2013).

The initial PSO was designed to operate in continuous search space, but was adapted by Kennedy and Eberhart (1997) to operate on discrete binary variables. One drawback of the standard PSO is a possible premature convergence and the associated trapping in local optima (Boussaïd et al., 2013). To overcome these shortcomings, standard PSO can be combined with elements of evolutionary computation. A comprehensive literature review on PSO methods is the one of Sedighizadeh and Masehian (2009). Tian, Liu, Yuan, and Wang (2013) redefined the particle's velocity and moved the particle by considering the differences between the particle's individual best and swarm best. They also introduced a mutation strategy to prevent premature convergence. In addition, they adopted a randomized exchange neighborhood search to enhance the local search ability.

The work at hand extends the algorithm of Tian et al. (2013) and combines it with the savings algorithm developed by Clarke and Wright (1964), a 2-opt-algorithm and the nearest-neighbor heuristic to solve the joint order batching and picker routing problem. The 2-opt algorithm is probably the most basic local search heuristic for the TSP, but achieves good results for real world instances both with respect to runtime and approximation ratio (Englert, Röglin, & Vöcking, 2014). In case of the picker routing problem in manual picker-to-parts warehouses, Kulak et al. (2012) showed that 2-opt leads to good results within a very short time. Since the routing heuristic is embedded into PSO, and PSO itself is embedded into the dynamic storage location algorithm, it is essential that the picker routing problem is solved within a tight time window. Therefore, we choose the 2-opt algorithm to solve the picker routing problem. We choose nearest-neighbor to calculate a first feasible solution for the picker routing problem, which is later improved by the 2-opt algorithm. The latter has the advantage that the tours incorporate only a few serious mistakes. The calculated tours have long segments where nodes are connected by short edges. Such tours are good starting tours for a subsequent application of improvement methods (Tsai, Tsai, & Tseng, 2004).

The framework of the discrete evolutionary particle swarm optimization (DEPSO) is shown in Fig. 2.

5.2.1. Picker routing

The exact algorithm proposed by Cambazard and Catusse (2018) has a runtime complexity of $O(hv5^h)$, where n is the number of cities located on h horizontal and v vertical lines. Transferred to order picking, the algorithm has an exponential runtime in the number of cross-aisles. Since the picker routing problem needs to be solved many times during the optimization process of DEPSO, such exact algorithms cannot be used in real-life environments. For this reason, we use a nearest neighbor heuristic and a 2-opt-algorithm, which are both time-efficient. Nearest neighbor is a constructive heuristic, which builds a tour by adding the node that is closest to the current node. The 2-opt heuristic starts with the tour constructed by the nearest neighbor algorithm and tries to reduce its length by exchanging two edges that are part of the current tour with two new edges that are not part of the tour (Kulak et al., 2012). Both heuristics are summarized in Appendices A and B.

5.2.2. Initialization

We use the following encoding scheme for the algorithm: A particle consists of a permutation of all K costumer orders, where each order occurs exactly once. To assign orders to batches, we use the first fit batch selection rule. This rule assigns orders into fewer batches than other rules (as compared, for example, to the next fit rule) because several batches are open at the same time (Koch, 2014). We assign orders to batches in the sequence of the particle's permutation and open a new batch in case an order does not fit into the current batch anymore. We then continue in the permutation and assign the remaining orders to the batch with the smallest number into which they fit (Wäscher, 2004). This procedure is illustrated in an example in the Online Supplement, Part 1. The predefined number of particles $A^{particle}$ is randomly generated upon initialization of the algorithm.

To shorten computation time and to guide the swarm early to a promising region of the search space, a problem-specific heuristic can be used (see, e.g., Lam, Nicolaevna, & Quan, 2007). Therefore, we integrate the savings algorithm into the swarm to determine the permutation of one particle. Savings algorithms are based on the savings in travel distance, $Sav_{k_1k_2}$, that can be obtained by combining two orders k_1 and k_2 in a single route, as compared to the situation where both orders are picked separately (De Koster, Van der Poort, & Wolters, 1999). Routes are constructed with the heuristics outlined in Section 5.2.1 and the travel distances are calculated with the help of the distance matrix. The savings algorithm is described in Appendix D.

When all particles have been generated, each particle's travel distance is calculated with the help of the picker routing heuristics and the distance matrix. Each particle p has a memory and saves its previous best position P_p^{best} and the previous best position of the whole swarm, G_{best}. Besides the particle's position in the search space, each particle p has a specific velocity $V_{p,t} = \{v_{p,1,t}, v_{p,2,t}, ..., v_{p,k,t}\}$ at time t, where $v_{p,k,t} \in \{-1, 0, 1\}$ is assigned to order k. The velocity vectors are generated randomly. Also, a stagnation factor $S^{StagG_{best}}$ is defined and set to zero. This factor counts the number of iterations where G_{best} has not improved. If G_{best} is updated, $S^{StagG_{best}}$ is set to zero, and otherwise it is increased by one.

5.2.3. Movement

In each iteration, particles move through the search space to new positions. The movement is controlled by a movement probability and a velocity vector:

- If $v_{p,k,t} = 1$, particle p in position k is attracted by G_{best} and moves towards it.
- If $v_{p,k,t} = -1$, particle p in position k is attracted by P_p^{best} and moves towards it.
- If $v_{p,k,t} = 0$, there is no movement of particle p in position k.

Each particle has two movement probabilities, one in the direction of its P_p^{best} and one in the direction of G_{best}. The movement probabilities

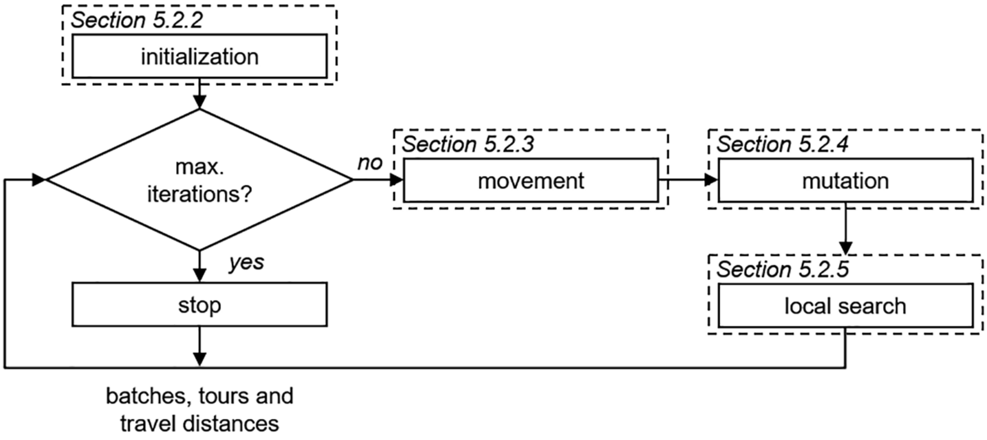


Fig. 2. The framework of DEPSO.

can be calculated by the difference of the permutations P_{p_1} and P_{p_2} of particles $p_1$and p_2:

$$Df_{p_1,p_2} = \frac{\sum_{j=1}^{K} sign(|P_{p_1}(j) - P_{p_2}(j)|)}{K},$$

where K is the total number of orders. The signum function *sign* is 0 if $P_{p_1}(j) = P_{p_2}(j)$, else 1. This procedure is illustrated in an example in the Online Supplement, Part 2. The movement probability of particle p to P_p^{best} is $B_{p,P_{best}} = Df_{p,P_p^{best}}$, and the movement probability to G_{best} is $B_{p,G_{best}} = Df_{p,G_{best}}$. Unlike Tian et al. (2013), we do not multiply the difference by a random number because we do not want to decelerate the particle's movement. The movement of a particle p is described in Appendix E, and an example is provided in the Online Supplement, Part 3.

After the particle has moved, orders are assigned to batches with the first fit rule, tours are constructed with the picker routing heuristics and the travel distances are calculated. If P_p^{best} or G_{best} has improved, they are updated and the next particle is moved. After all particles have been moved, we evaluate if G_{best} has improved. In this case, $S^{StagG_{best}}$ is set to zero, otherwise it is increased by one.

5.2.4. Mutation

In the case where a particle's current position, P_p^{best}, and G_{best} are the same, the particle will not move. To prevent stagnation, a mutation operator is introduced that modifies the position of some particles. The mutation operator is controlled by a mutation probability, which is based on the particle's intensity:

$$Int_p = \frac{1}{3}\cdot\left(Df_{p,P_p^{best}} + Df_{p,G_{best}} + Df_{P_p^{best},G_{best}}\right)$$

The intensity indicates how close P_p, P_p^{best} and G_{best} are to each other. The closer they are, the higher the intensity and vice versa. After calculating every particle's intensity, the maximum intensity of all particles Int^{max} and the minimum intensity Int^{min} can be determined. A particle's mutation probability is:

$$M_p = \frac{Int^{max} - Int_p}{Int^{max} - Int^{min}}, \quad if Int^{max} \neq Int^{min} (=1, else)$$

As Tian et al. (2013), we assume that a particle is closer to the global optimum if it has a better fitness. In such cases, only minor mutations should be allowed to prevent the particle from moving too far away from the global optimum and vice versa. Therefore, a particle's relative closeness Cl_p to the global optimum is estimated as:

$$Cl_p = \frac{Td_p - Td_{G_{best}}}{Td^{max} - Td_{G_{best}}}$$

where Td_p is the particle's travel distance, $Td_{G_{best}}$ is the travel distance of G_{best}, and Td^{max} is the travel distance of the particle with the lowest fitness value. All travel distances can be calculated with the help of the picker routing algorithms. We use three mutation operators:

- *Swap*: Swap the orders of two randomly chosen positions.
- *Shift*: A randomly chosen order is placed in front of a random position.
- *Inverse*: Two random positions are chosen and their orders are swapped. The orders in-between are inverted.

While the swap operator only applies minor changes to the order sequence, the changes are extensive if the inverse operator is applied. The mutation operation is summarized in Appendix D, and an example is provided in the Online Supplement, Part 4.

5.2.5. Local search

The PSO might stagnate and remain in a local optimum when G_{best} does not improve over several iterations. To avoid a premature termination of the algorithm, a local search procedure is integrated into DEPSO. Since the local search will cost some computational time, it is not performed in every iteration. The use of the local search is instead controlled by an adaptive stagnation threshold, which decreases over time such that the local search is performed rarely at the beginning of the optimization and more frequently towards its end. The stagnation threshold is calculated as follows:

$$S^{Stag} = round\left(1 + S^{maxStag}.\frac{It^{max} - It^{cur}}{It^{max}}\right)$$

$S^{maxStag}$ is the predefined maximum stagnation bound, It^{max} is the predefined maximum number of iterations of DEPSO, and It^{cur} is the current iteration. The symbol *round* represents the rounding function. The local search is described in Appendix G. Unlike Tian et al. (2013), we place one particle on the updated G_{best}. This particle will most likely be mutated in the next iteration and the area around G_{best} can be explored.

5.2.6. Overall procedure of DEPSO

The core parts of DEPSO were introduced in detail above. The overall procedure of DEPSO can now be described stepwise in algorithmic notation as follows:

```
DEPSO
Initialization
Step 1:   Randomly generate ($A^{particle}-1$) particles and their corresponding velocity vectors.
Step 2:   Generate one particle with the help of the savings algorithm.
Step 3:   Assign the orders of each particle to batches with the help of the first fit rule.
Step 4:   Construct the tours for each batch with the picker routing heuristics and calculate the travel distance.
Step 5:   Set the particle's current permutation as $P_p^{best}$ and choose the particle with the shortest distance as $G_{best}$.
Optimization
Step 6:   Choose particle $p = 1$.
Step 7:   Move the currently selected particle.
Step 8:   Assign orders of the currently selected particle with the help of the first fit rule to batches.
Step 9:   Construct the picking tours of the batches with the help of the picker routing heuristics and calculate the travel distance.
Step 10:  Update $P_p^{best}$ and $G_{best}$ if the solution has improved. If there are particles that have not been checked yet, increase $p$ by one and proceed with step 7, else with step 11.
Step 11:  Check if $G_{best}$ has been updated. If an update has occurred, set $S^{StagG_{best}}$ to zero, else increase $S^{StagG_{best}}$ by one.
Step 12:  Apply the mutation operation.
Step 13:  Apply the local search procedure.
Step 14:  If $It^{cur} < It^{max}$, increase $It^{cur}$ by one and go back to step 6, else proceed with step 15.
Output
Step 15:  Terminate and return the best batch assignment, the picking tours and the travel distance.
```

Before starting DEPSO, the following parameter values need to be defined:

- Number of particles: $A^{particle}$
- Number of iterations: It^{max}
- Threshold value of G_{best}: $S^{G_{best}}$
- Maximum number of iterations for the local search: It^{maxLS}
- Maximum stagnation bound: $S^{maxStag}$

An appropriate parameter combination is crucial for the results of the procedure, and they can be found during pretests. This is illustrated in Section 6.2.

5.3. Dynamic storage location assignment algorithm

To solve the dynamic storage location assignment problem, we propose an algorithm based on the dynamic ABC storage policy of Pierre et al. (2004). We extend it by a forecasting procedure to estimate the future demand, a relocation algorithm to identify candidates for a relocation and corresponding storage locations, and a procedure for estimating the future travel distance reduction of an item relocation. Moreover, we integrate the joint order batching and picker routing method and take the relocation effort into account. Fig. 3 illustrates the general structure of the algorithm.

We assume a class-based storage location assignment with three classes A, B and C, where A items are fast movers and C items slow movers. The algorithm is run at the end of every planning period, where the length of the planning period can be specified by the user. In the first step, the algorithm checks if items are stored in the wrong class. Each item stored in the wrong class then gets a priority coefficient based on its estimated future demand. We then evaluate if all items with a priority coefficient can be relocated in light of the available storage locations in the different classes and generate a relocation suggestion. The algorithm then evaluates the suggested relocation with respect to its potential to reduce future travel distances. If the reduction in future travel distance exceeds the relocation effort, the item is relocated, otherwise not. Relocation suggestions are tested until a termination criterion is reached.

5.3.1. Item classification

At the end of period $t-1$, items are assigned to classes based on their number of orderlines with the help of an ABC analysis. For periods $t, ..., t+U^{for}$, a demand forecast is made, and the items are again classified. The relative size of a class can be defined by the user. In the following, forecasted values are denoted by the symbol "^". For $t-1$ and the periods $r = t, ..., U$, the last items of class A and class B are identified, and their numbers of orderlines per period are the class limits AB ($A_{1,t-1}^{Poscl}$ resp. $\widehat{A}_{1,r}^{Poscl}$) and BC ($A_{2,t-1}^{Poscl}$ resp. $\widehat{A}_{2,r}^{Poscl}$). We refer to the class limit AB with the order number 1 and to the class limit BC with the order number 2. Fig. 4 shows the number of orderlines of item m and the class limits over time in an example. Note that the figure does not show the number of items of a class (the "size" of the classes), but instead the number of orderlines an item should have at least to be assigned to a certain class.

The class limits vary since the total number of orderlines may vary as well from period to period due to changes in customer demand. At the end of period $t-1$ (period 10 in the example in Fig. 5), an item's number of orderlines for period $t-1$ can be calculated from the customer orders that were received in that period. In a given period $t-1$, a forecast for the future number of orderlines in the next U^{for} periods can be made based on the time series of past orderlines. In the numerical experiments in Section 6, we use Holt-Winter's exponential smoothing method for this purpose, since the demand of the items follows a linear trend with a seasonal component. Several studies (e.g., Makridakis et al. (1982), Makridakis et al. (1993) or Makridakis and Hibon (2000)) have shown that Holt-Winter's exponential smoothing method gives results comparable to more complex forecasting methods. According to Schuhr (2012), Holt-Winter's exponential smoothing is a well-established method for applications in the field of warehousing or production

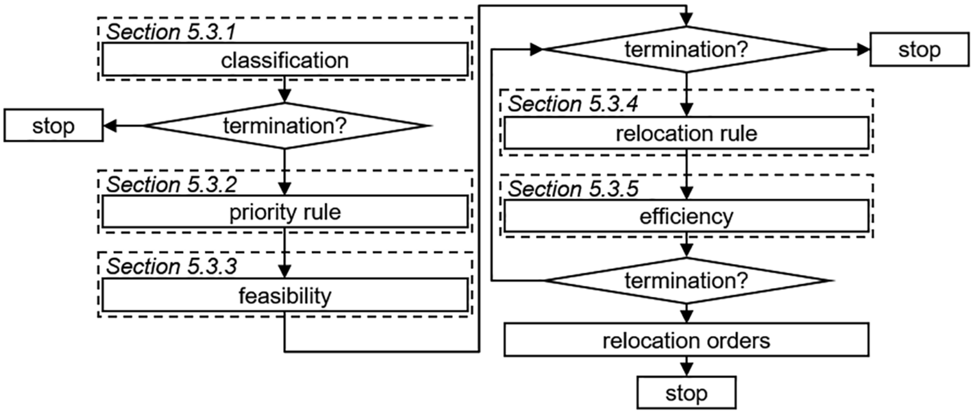


Fig. 3. Structure of the dynamic storage location assignment algorithm.

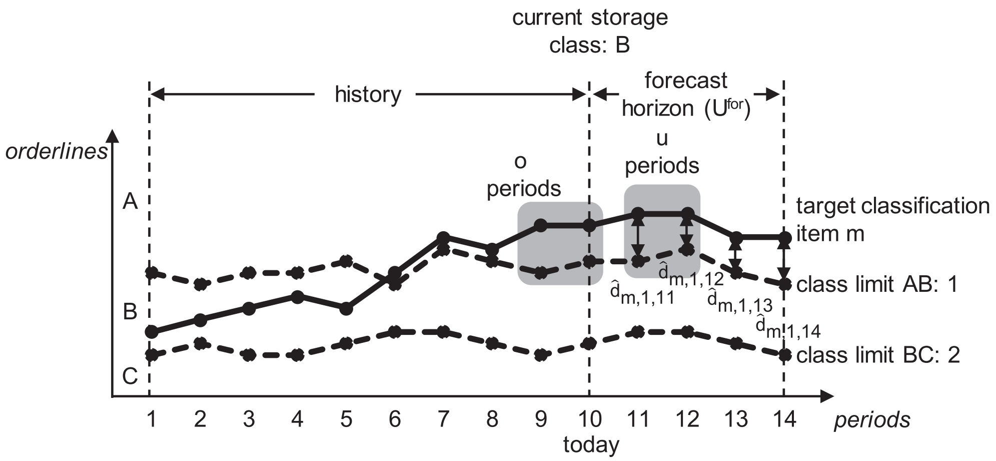


Fig. 4. Item time series and class limits.

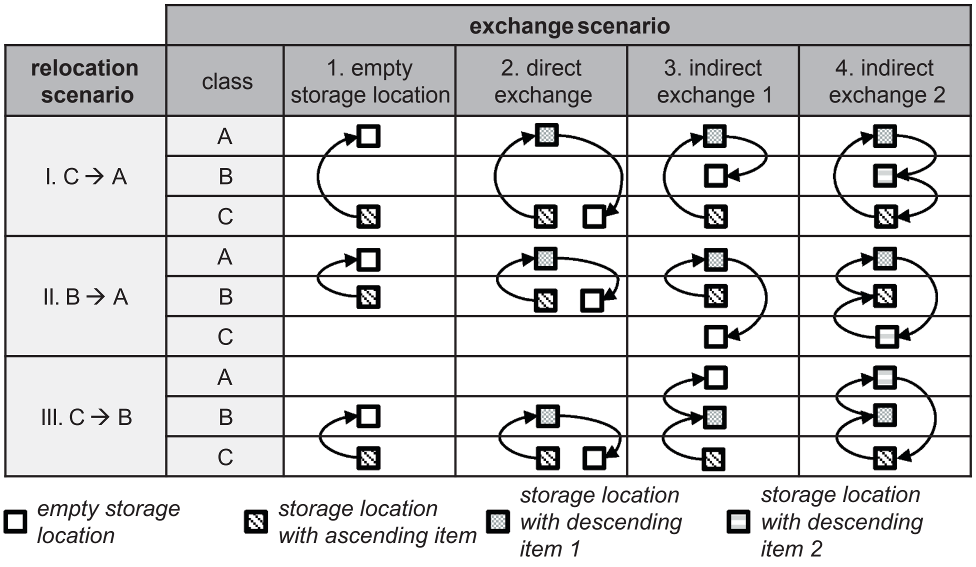


Fig. 5. Relocation scenarios for three storage classes.

planning that exhibit demand following a trend with a seasonal component. It should be noted that other forecasting methods could be integrated into the proposed procedure if required. In the example shown in Fig. 4, item m is stored in class B in period 10 (cf. the solid line, which indicates the class where item m should be stored). The solid line in Fig. 4 indicates that since period 6, item m should have been assigned to class A, and that it is expected to stay in this class for the next four periods. Item m is obviously a candidate for a relocation.

We now define two thresholds for a reassignment of an item to another class: First, the item should be stored in the wrong class for already o periods, and secondly we should expect that it will stay in the target class for at least u periods. Both thresholds are introduced to prevent that items are frequently assigned to a new class and then immediately assigned back in the next iteration, which would increase computational time.

5.3.2. Priority rule

For all items we identified as candidates for an eventual relocation in Section 5.3.1, we calculate a priority coefficient to find the most promising candidates. Let U_m^{tar} with $U_m^{tar} \leq U^{for}$ be the number of subsequent periods in which item m is assumed to stay in the target class. The priority coefficient then considers the forecasted number of orderlines per item, $\widehat{A}_{m,r}^{Pos}$ for $= t, ...,t + U_m^{tar}$. We denote the corresponding class limit, which is between this item's current class and its target class, as the 'relevant class' $relCL$ and the number of orderlines as $\widehat{A}_{relCL,r}^{Poscl}$. The difference between an item's number of orderlines ($\widehat{A}_{m,r}^{Pos}$) and the relevant class limit ($\widehat{A}_{relCL,r}^{Poscl}$) is called distance $\hat{d}_{m,relCL,r}$, and it can be calculated as follows:

$$\hat{d}_{m,relCL,r} = |\widehat{A}_{m,r}^{Pos} - \widehat{A}_{relCL,r}^{Poscl}| \quad \forall\ r$$

An item's priority value $PV_m$is calculated as the summed distances of the future periods in which the item will be in the target class:

$$PV_m = \sum_{r=t}^{t+U_m^{tar}} \hat{d}_{m,relCL,r}$$

All priority values are stored in a priority list, which is sorted in descending order.

5.3.3. Feasibility

Before relocating items, we need to make sure that the items in the priority list can be relocated and that no class $cl \in \{A, B, C\}$ contains more items than storage locations. Therefore, the number of storage locations that are currently empty, $NSL_{cl}^{alempty}$, the number of storage locations that may become empty during the relocation of items, NSL_{cl}^{bempty}, and the number of storage locations that may become occupied during the relocation of items, NSL_{cl}^{boccup}, are determined. The feasibility value is calculated as follows:

$$FV_{cl} = NSL_{cl}^{alempty} + NSL_{cl}^{bempty} - NSL_{cl}^{boccup}$$

If $FV_{cl} \geq 0$, all relocations can be executed for this class. $FV_{cl} < 0$, in turn, indicates that there is not sufficient space in a class. If more than one class has a negative value, the class with the lowest FV_{cl}-value is chosen, and $|FV_{cl}|$ items that were supposed to be relocated into this class are eliminated from the priority list. We thereby first eliminate the items with the lowest priority value and repeat this procedure until no negative feasibility values exist anymore.

5.3.4. Relocation rule

For relocating items, we chose the first item in the priority list that should be moved to a higher class ("ascending item"). Fig. 5 illustrates that there are three different relocation scenarios: The item can move from class C or B to class A or from class C to class B.

Each relocation scenario could have up to four different exchange scenarios: Item m can be relocated to an empty storage location in the target class (exchange scenario 1 in Fig. 5) or be exchanged with an item in the priority list ("descending item") that should be moved from the target to the current class of item m (exchange scenario 2). An indirect exchange occurs when item m is moved to the target class, and when an item from the target class is moved to an empty storage location in the third class (exchange scenario 3). In exchange scenario 4, three items are involved and change their storage location, leading to the highest relocation effort. To identify the descending item in exchange scenarios 2 to 4, we implemented the following procedure: In the class where an item needs to be removed, the item with the highest priority value is selected as this item is expected to have the lowest demand in its current class in the future. If multiple items have the same priority value, we choose the item that is closest to the depot. This way, we ensure that the future travel distance of the ascending item can be minimized. If there are still multiple items left as candidates for a removal from the class, we minimize the relocation effort and give priority to exchange scenario 1 before scenarios 2, 3 and 4 and to scenarios 2 and 3 before scenario 4. Decisions between scenarios 2 and 3 are made randomly.

5.3.5. Efficiency of the relocation

After selecting an ascending and eventual descending items (we refer to this set as a "relocation suggestion" in the following), the relocation effort of all involved items is calculated. Subsequently, the reduction in future travel distance that would result from relocating the ascending and the descending items is estimated. Both procedures are described in more detail below. If the estimated reduction in travel distance exceeds the relocation effort, this relocation suggestion is implemented, otherwise discarded. If the termination criterion (e.g., the number of evaluated relocation suggestions) has not been reached yet, then the next items from the priority list are evaluated. If a relocation suggestion was accepted, the next relocation suggestion is evaluated with the new storage location assignment.

The relocation effort of an item m consists of three components: The travel distance E_m^{dis} from the current to the target storage location, the physical effort E_m^{phy} associated with retrieving and storing the item, and the administrative effort E_m^{adm} to update the IT-system. Both E_m^{phy} and E_m^{adm} are expressed as travel distance equivalents:

$$E_m^{phy} = t^{phy} \cdot v^{pick}$$

$$E_m^{adm} = t^{adm} \cdot v^{pick}$$

The total relocation effort E_m^{total} of item m is:

$$E_m^{total} = E_m^{dis} + E_m^{phy} + E_m^{adm}$$

The total relocation effort E_{rel}^{total} is calculated for all involved (ascending and descending) items.

Before evaluating the first relocation suggestion, the travel distance (Td^{act}) of the current period without any relocations is calculated with the orders of the current period using DEPSO. Td^{act} is then saved as the comparison travel distance Td^{com}. We then execute the relocation suggestion and calculate the distance after the relocation Td^{rel} with the help of DEPSO. We continue with calculating the reduction in travel distance $Tdr_{rel,t-1}$ for the relocation suggestion in the current period $t-1$:

$$Tdr_{rel,t-1} = Td^{com} - Td^{rel}$$

If $Tdr_{rel,t-1} \leq 0$, the involved items are not relocated and removed from the priority list, and the next items are tested. If $Tdr_{rel,t-1} > 0$, the relocation would lead to a reduction in travel distance in the current period. We then estimate the future travel distance reductions using the procedure illustrated in Fig. 6.

The above procedure assumes that a high pick frequency in the future leads to a higher travel distance reduction for a particular relocation than the calculated reduction in the current period, and vice versa. An estimate of the pick frequency is the distance between an item's number of orderlines and the relevant class limit, $d_{m,relCL,t-1}$ (see Section 5.3.2). To simplify computation, we only consider the time series of the ascending item, which is the item driving the efficiency gains. We use the relative distance of item m to the relevant class limit, $d_{m,relCL,t-1}^{rel}$, to acknowledge that the class limits are fluctuating:

$$d_{m,relCL,t-1}^{rel} = \frac{d_{m,relCL,t-1}}{A_{relCL,t-1}^{Poscl}} \cdot 100\%$$

We then calculate a factor $\xi_{m,t-1}$ that describes the reduction in travel distance that can be achieved when deviating by one percent from the relevant class limit in the current period:

$$\xi_{m,t-1} = \frac{Tdr_{m,t-1}}{d_{m,relCL,t-1}^{rel}}$$

This factor can be used as an estimator for future periods:

$$\hat{\xi}_{m,r} = \xi_{m,t-1}$$

The future travel distance reductions can be estimated as follows:

$$\widehat{Tdr}_{m,r} = \hat{\xi}_{m,r} \cdot \hat{d}_{m,relCL,r}^{rel} \quad \forall\ r$$

The relocation is accepted if:

$$\sum_{r=t}^{t+U_m^{tar}} \widehat{Tdr}_{m,r} > E_m^{total}$$

Otherwise, the relocation is discarded. In both cases, the items are removed from the priority list. If the stopping criterion has not been reached yet, the next relocation is tested. If the relocation is accepted, the travel distance after the relocation, Td^{rel}, becomes the new comparison travel distance Td^{com}. The next relocation is then tested against the updated Td^{com}-value and the updated storage location assignment. Possible stopping criteria are the maximum number of tested relocations, the minimum number of accepted relocations, or a maximum computation time.

6. Numerical analysis

This section evaluates the performance of the proposed algorithms in numerical experiments. First, we compare DEPSO with the picking of individual orders, a first-come-first-served heuristic and the savings algorithm of Clarke and Wright (1964). The latter three heuristics are combined with the S-Shape heuristic to solve the picker routing problem. We use the S-Shape heuristic as it is the most popular heuristic in the literature (Masae et al., 2020) and often served as a benchmark to evaluate other solution approaches (Chen, Wang, Qi, & Xie, 2013). We further compare the dynamic storage location assignment algorithm to the case where items are not relocated.

For both experiments, we consider a warehouse with three blocks as

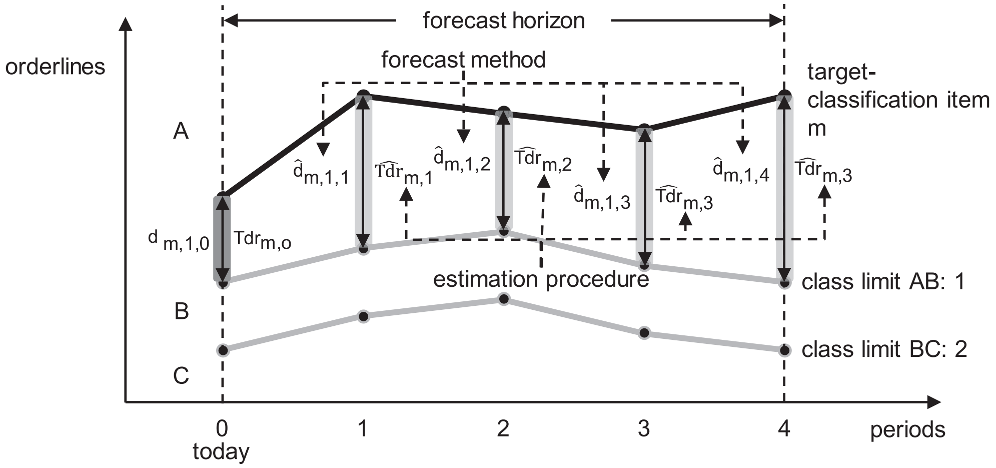


Fig. 6. Estimation procedure for the future travel distance reduction.

illustrated in Fig. 7.

The warehouse consists of 10 picking aisles and 4 cross aisles. On each side of a picking aisle, there are 90 rack elements with 4 storage locations per element. In total, the warehouse has 7,200 storage locations. One rack element is one length unit (LU) wide and 1 LU long. Cross and picking aisles are 1 LU wide. The depot is located in the right cross aisle next to the lower picking aisle. We use a conventional layout for our experiments since Masae et al. (2020) have shown that this is the most frequently used layout in the literature. According to Roodbergen and de Koster (2001) performance of heuristics in warehouses with one-block layout have been studied extensively. Additional cross aisles provide the opportunity to realize more efficient picking tours (Vaughan, 1999). Therefore, several studies focused on two-block layouts with three cross aisles (e.g. Ene and Öztürk (2012), Kulak et al. (2012), Chen et al. (2015) or Scholz et al. (2017)). Recent studies, such as the one of Cano, Correa-Espinal, Gómez-Montoya, and Cortés (2019), used a three-block layout with four cross aisles for their analysis. However, they tested their algorithms with a maximum of 15 positions per aisle side only. To increase the practical transferability, we also use a three-block layout for our analysis but enlarge the warehouse to 360 positions per aisle side.

Items are assigned to locations with the help of a class-based storage location assignment, where the top 5% items in terms of turnover are assigned to class A, the next 15% to class B, and the bottom 80% to class C. Item weights are randomly selected from [0.1;1.0] weight units (WU). There are 6,000 different items to be stored in the warehouse, and the order picking vehicle has a capacity of 100 WU. Storage locations are assigned to classes according to their distance to the depot.

6.1. Problem scenarios for the DEPSO algorithm

We analyze several problem scenarios. Each scenario is characterized by the number of orders (N^{Ord}), the maximum number of

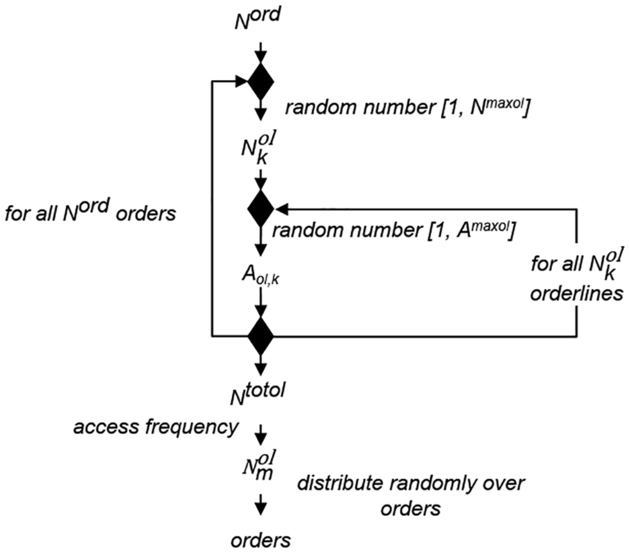

N^{ord}
random number [1, N^{maxol}]
N_k^{ol}
for all N^{ord} orders
random number [1, A^{maxol}]
$A_{ol,k}$
for all N_k^{ol} orderlines
N^{totol}
access frequency
N_m^{ol}
distribute randomly over orders
orders

Fig. 8. Generation of test data for DEPSO.

orderlines per order (N^{maxOl}), and the maximum number of parts per orderline (A^{maxOl}). The turnover rates per item can be simulated with the help of an access function. Different types of functions can be generated by varying the access frequency AF. AF is the percentage of accesses of the 20% most picked items. This can be modeled using the following function (Reschke, 2013):

$$Z = y^c$$

The parameter c is determined by:

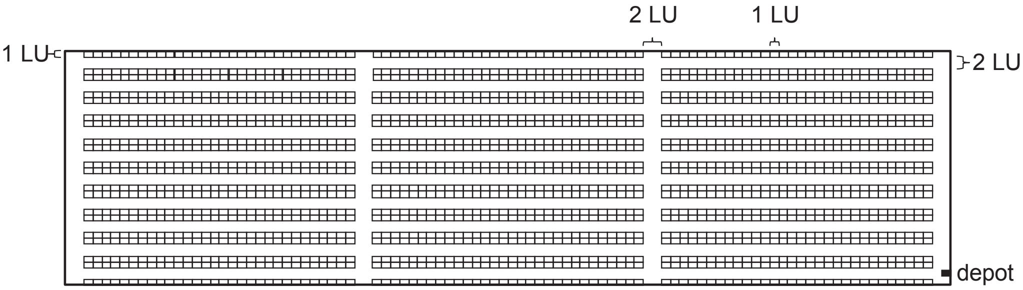


Fig. 7. Warehouse layout considered in the numerical experiments.

Table 1
Parameter values and problem scenarios (preliminary tests).

Name		Symbol	Values
Number of particles		$A^{particle}$	{5, 10, 15}
Number of iterations		It^{max}	{200, 350, 500}
Threshold value G_{best}		$S^{G_{best}}$	{0.25, 0.5, 0.75}
Maximum number of iterations for local search		It^{maxLS}	{50, 75, 100}
Maximum stagnation bound		$S^{maxStag}$	{20, 40, 60}
Problem number	Number of orders	Maximum number of orderlines	Maximum number of parts
1	50	10	10
2	100	2	6
3	150	6	2
4	200	2	2

$$c = \frac{\log_{10}AF}{\log_{10}0.2}$$

As shown in Fig. 8, for each problem scenario, we first generate the number of orders N^{ord}.

For each order $k \in N^{ord}$, a random number of orderlines $N_k^{ol} \in [1, N^{maxol}]$ is generated, where N^{maxol} is the problem-specific maximum number of orderlines. In Sections 6.1 and 6.3, all random numbers are drawn from the respective intervals according to a uniform distribution. Each orderline in order k has a random number of parts $A_{ol,k} \in [1, A^{maxol}]$. Based on the total number of orderlines N^{totol} of all orders, the number of orderlines of item m, N_m^{ol}, can be determined using AF. Then, the item's orderlines are randomly distributed over all orders N^{Ord}; note that each item can be in an order only once.

We analyze the following problem parameters:

- Number of orders (N^{Ord}): {50, 100, 150, 200}
- Maximum number of orderlines per order (N^{maxOl}): {2, 6, 10}
- Maximum number of parts per orderline (A^{maxOl}): {2, 6, 10}

We set AF to 0.6. The problem scenarios are referred to using a name tag structured as follows: $N^{Ord}\_N^{maxOl}\_A^{maxOl}$.

6.2. Evaluation of the DEPSO algorithm

In this section, we evaluate the performance of DEPSO for the case without dynamic storage location assignment. To determine a suitable parameter set for the algorithm, we performed comprehensive preliminary tests with the procedure described in Koch (2014). Table 1 shows the tested parameter values and the analyzed problem scenarios, leading to a total of 243 parameter sets.

For each problem scenario, we randomly generated five instances separate from the main numerical experiments. Every instance was solved for the 243 parameter sets of the DEPSO algorithm. The algorithm was implemented and tested in MATLAB on a computer with a 2.5 GHz processor and 16 GB RAM. We then selected the parameter set for the main numerical experiments that had the lowest average deviation from the best found solution for each problem instance:

- Number of particles: 5
- Number of iterations: 500
- Threshold value G_{best}: 0.5
- Maximum number of iterations for local search: 100
- Maximum stagnation bound: 20

Appendix H shows the results of the main numerical experiments in detail.

We analyzed 35 different problem scenarios with 40 randomly generated instances for each scenario. We did not analyze problem scenario 50_2_2, because in most cases all orders fitted into one batch, which significantly simplified the problem at hand. Compared to SOP, DEPSO reduced average travel distances by 83.78% on average. Using DEPSO led to travel distances that were, on average, 40.80% shorter than those obtained by FCFS. Compared to the savings algorithm, DEPSO reduced travel distances by 31.64% on average. As shown in Fig. 9, the average reductions in travel distance are higher when orders have a lower number of orderlines and number of parts per orderline.

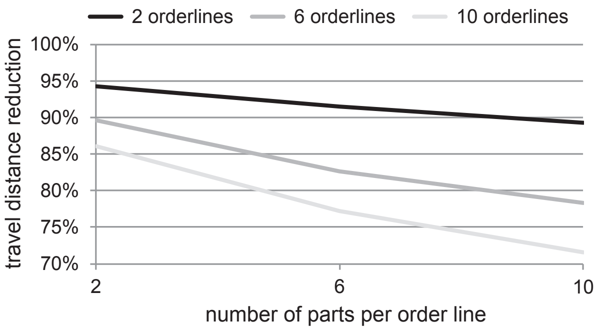


Fig. 9. Effect of the number of orderlines and the number of parts per orderline on the average reduction in travel distance.

Our results imply that the developed method leads to the highest performance improvements especially for companies with small orders. Given the computation times obtained for solving the test instances (all test instances could be solved in less than 4 min), the DEPSO algorithm seems to be a method well suited for solving large problems in practice. The results indicate that the improvements that can be obtained by using DEPSO may lead to a significant increase in the efficiency of order picking processes.

6.3. Problem scenarios for the dynamic storage location assignment algorithm

This section evaluates the performance of the dynamic storage location assignment algorithm including DEPSO. We first present a general procedure for generating the required problem data and then describe the two data sets we analyzed. The test data for the dynamic storage location assignment algorithm were generated as shown in Fig. 10.

The problem consists of several periods (e.g., months) divided into several sub-periods (e.g., weeks or days). Orders in period 1 are generated in the same way as for DEPSO and distributed uniformly over the sub-periods. Since we would like to test the performance of the dynamic storage location assignment algorithm in a dynamic situation with highly fluctuating demand, we modeled the customer demand with the help of a time series with a linear trend, an additive seasonality component and an irregular factor (see also Schröder, 2012).

The demand of each item (number of orderlines) per period for the U periods included in the planning horizon was modeled as follows:

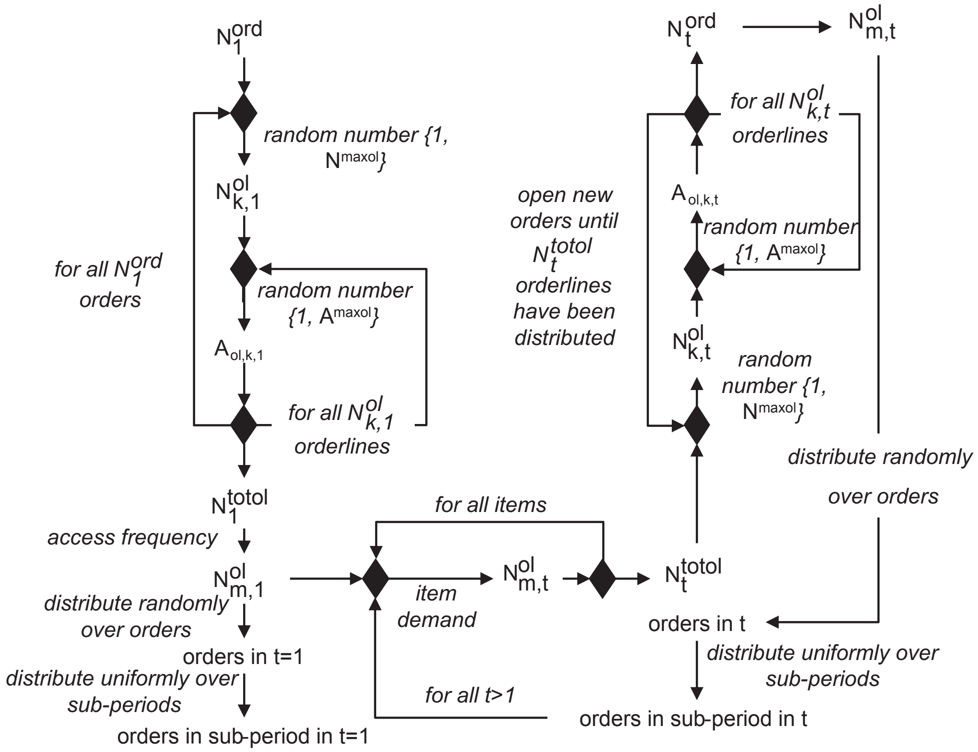
$$N_{m,t}^{ol} = (round)\left(N_{m,1}^{ol} + T_m^{trend}\cdot(t-1) + S_m^{season}\cdot cos\left(\frac{2\cdot\pi\cdot(t-1)}{L}\right) + Ir_{m,t}\right)$$
$$\forall\, t \in U$$

The trend T_m^{trend} and seasonality S_m^{season} components for each item m are modeled with the help of the number of orderlines in period 1, a problem-specific factor (trend or seasonality factor) and a random number:

$$T_m^{trend} = Tf\cdot N_{m,1}^{ol}\cdot rand$$

$$S_m^{season} = Sf\cdot N_{m,1}^{ol}\cdot rand$$

An item's number of orderlines per period without irregular component $N_{m,t}^{olwoIr}$ is:



N_1^{ord}
random number {1, N^{maxol}}
$N_{k,1}^{ol}$
for all N_1^{ord} orders
random number {1, A^{maxol}}
$A_{ol,k,1}$
for all $N_{k,1}^{ol}$ orderlines
N_1^{totol}
access frequency
$N_{m,1}^{ol}$
distribute randomly over orders
orders in t=1
distribute uniformly over sub-periods
orders in sub-period in t=1
for all items
item demand
$N_{m,t}^{ol}$
for all t>1
N_t^{totol}
orders in t
distribute uniformly over sub-periods
orders in sub-period in t
N_t^{ord}
$N_{m,t}^{ol}$
for all $N_{k,t}^{ol}$ orderlines
open new orders until N_t^{totol} orderlines have been distributed
$A_{ol,k,t}$
random number {1, A^{maxol}}
$N_{k,t}^{ol}$
random number {1, N^{maxol}}
distribute randomly over orders

Fig. 10. Generation of test data for dynamic storage location assignment.

$$N_{m,t}^{olwoIr} = N_{m,1}^{ol} + T_m^{trend}\cdot(t-1) + S_m^{season}\cdot cos\left(\frac{2\cdot\pi\cdot(t-1)}{L}\right) \quad \forall\, t \in U$$

The irregular component $Ir_{m,t}$ can be positive or negative, and it depends on the item's number of orderlines per period without the irregular component:

$$|Ir_{m,t}| = Irf\cdot N_{m,t}^{olwoIr}\cdot rand \quad \forall\, t \in U$$

$$Ir_{m,t}{=}|Ir_{m,t}|\cdot(-1),\ \ if\ \ \ rand\ \ \ \leq 0.5(=|Ir_{m,t}|,\ else) \quad \forall\, t \in U$$

The problem parameters *Tf* (trend factor), *Sf* (seasonality factor), *Irf* (irregular factor) and *L* (length of the seasonal cycles) influence how strongly the customer demand changes over time. To avoid unrealistic scenarios with an extreme demand fluctuation over time, we introduce a condition that limits demand variations. The total number of orderlines over all items in the period with the highest demand should not exceed *M* times the lowest demand that occurs in the planning horizon:

$$max\{N_t^{totol}\} \leq M\cdot min\{N_t^{totol}\}\ \forall\, t \in U$$

Modern warehouses often face situations where fast-moving products become slow movers over time and vice versa (cf. Kofler et al., 2015). Therefore, we also implemented a procedure simulating such dynamics. The basic idea is to randomly select a fast-moving product and to gradually turn this product into a slow mover over time. Similarly, we select a slow-moving item and turn it into a fast mover. Using a probability threshold (S^{prob}), we ensure that items with a high number of orderlines in the first period will more likely become a slow mover than items with fewer orderlines. Since we also want to permit that a fast mover grows over time, we implemented a random principle. We created the required data according to the following procedure:

Step 1: Create a list of all items and their number of orderlines in period 1. Sort the list in descending order of the number of orderlines.
Step 2: Choose the first item as basis item *BI*.
Step 3: Define a probability threshold S^{prob} and set it to 1.
Step 4: Calculate T_{BI}^{trend}, S_{BI}^{season}, $Ir_{BI,t}$ and $N_{BI,t}^{ol}$.
Step 5: If $rand < S^{prob}$, invert the algebraic sign $T_{BI}^{trend} = T_{BI}^{trend}\cdot(-1)$.
Step 6: Randomly choose an item from the list as counter item *CI*.
Step 7: Calculate T_{CI}^{trend}, S_{CI}^{season} and $Ir_{CI,t}$ as follows:

$$T_{CI}^{trend} = Tf\cdot N_{BI,1}^{ol}\cdot rand,\quad \text{if}\ \ T_{BI}^{trend} > 0:$$

$$T_{CI}^{trend} = Tf\cdot N_{BI,1}^{ol}\cdot rand\cdot(-1)$$

$$S_{CI}^{season} = Sf\cdot N_{BI,1}^{ol}\cdot rand$$

$$N_{CI,t}^{olwoIr} = N_{CI,1}^{ol} + T_{CI}^{trend}\cdot(t-1) + S_{CI}^{season}\cdot cos\left(\frac{2\cdot\pi\cdot(t-1)}{L}\right) \quad \forall\, t \in U$$

$$|Ir_{CI,t}| = Irf\cdot N_{CI,t}^{olwoIr}\cdot rand \quad \forall\, t \in U$$

$$Ir_{CI,t}{=}|Ir_{CI,t}|\cdot(-1),\ \ if\ \ \ rand\ \ \ \leq 0.5(=|Ir_{CI,t}|,\ else) \quad \forall\, t \in U$$

Calculate $N_{CI,t}^{ol}$.

Step 8: If $max\{N_t^{totol}\} \leq M\cdot min\{N_t^{totol}\}\ \forall\, t \in U$, keep $N_{BI,t}^{ol}$ and $N_{CI,t}^{ol}$, else set $N_{BI,t}^{ol} = N_{BI,1}^{ol}$, $N_{CI,t}^{ol} = N_{CI,1}^{ol}\ \forall\, t \in U$.
Step 9: $S^{prob} = S^{prob}\cdot 0.995$.
Step 10: Remove both items from the list and choose the first item as base item *BI*. If the list is empty, terminate the procedure.

In step 5, we induce that the most frequently picked items from period 1 will most likely become slow movers. To decrease the probability of those items becoming a slow mover, the probability threshold in step 3 can be set to a smaller value than 1. Step 9 lowers the probability threshold in each iteration, which reduces the probability that fast movers will be converted into slow movers. Step 7, in turn, effects that a randomly selected item gets an increasing (or decreasing) demand over time. The demand pattern of this item is based on the demand of the item *BI* that is turned into a slow-moving item (fast-moving item), multiplied by a random number. The intention of this step is to generate a new fast-moving item (slow-moving item) with a demand similar to the demand of the former fast-moving item (slow-moving

Table 2
Constants and parameters of the analyzed scenarios.

Parameters (constant)		Symbol	Value
Number of orders in t = 1		N_1^{ord}	5,000
Maximum number of orderlines per order		N^{maxol}	2
Maximum number of parts per orderline		A^{maxol}	6
Seasonal cycle		L	12 periods
Access frequency		Af	0.6
Number of items		A^{item}	6,000
Number of sub-periods		A^{subp}	20
Analyzed time		U	9 (+12) periods
Maximum fluctuation factor		M	2
Irregular factor		If	0.025
Physical effort		t^{phy}	3 min
Administrative effort		t^{adm}	1 min
Average walking speed		v^{pick}	1 length units/sec
Parameters (varied in experiments)	Symbol	Scenario 1	Scenario 2
Trend factor	Tf	0.300	0.150
Seasonality factor	Sf	0.150	0.075

item). If the condition in step 8 does not hold, both time series are discarded, and the next items are chosen from the list. The above procedure enables us to model a highly dynamic demand if required, where fast-moving items can (frequently) become slow movers, and vice versa. After the number of orderlines of the items per period have been calculated, orders are generated in a similar way as in period 1. Thus, an empty order is opened and the number of orderlines and number of items per orderline are generated randomly. This process is repeated until the number of orderlines of the orders equals the number of orderlines of all items in this period. Afterwards, the items are randomly assigned to orders. Finally, orders are distributed uniformly over the sub-periods. Table 2 shows the parameters used for this study.

We analyzed two test scenarios, one highly dynamic scenario and one with a less fluctuating demand. We considered 21 periods in total, but used the first 12 periods to parameterize the forecast coefficients and the last nine to test the dynamic storage location assignment algorithm. The current storage location assignment was determined based on the item demand of period 1, and items were assigned to classes with the help of an ABC analysis. Within the classes, items were assigned randomly to storage locations.

6.4. Results of the dynamic storage location assignment algorithm

As in Kofler et al. (2015), we evaluated 50 relocation suggestions in both test scenarios. Given that the demand of the items follows a linear trend and a seasonal component, we used Holt-Winters exponential smoothing method to forecast the item's orderlines per period (Schuhr, 2012):

$$\hat{x}_{T,\tau} = \hat{a}_t + \hat{b}_t \cdot \tau + \hat{c}_{T+\tau-\lceil \frac{\tau}{L} \rceil L}$$

$$\hat{a}_t = \alpha \cdot (x_t - \hat{c}_{t-L}) + (1-\alpha) \cdot (\hat{a}_{t-1} + \hat{b}_{t-1})$$

$$\hat{b}_t = \beta \cdot (\hat{a}_t - \hat{a}_{t-1}) + (1-\beta) \cdot \hat{b}_{t-1}$$

$$\hat{c}_t = \gamma \cdot (x_t - \hat{a}_t) + (1-\gamma) \cdot \hat{c}_{t-L}$$

where T is the period where the forecast is calculated, and τ is the period the forecast is made for. $\lceil \frac{\tau}{L} \rceil$ is the smallest non-negative integer not smaller than $\frac{\tau}{L}$. To set the smoothing parameters, we followed the recommendation of Silver, Pyke, and Thomas (2016) for situations without structural changes in the trend and seasonality pattern: $\alpha = 0.19$; $\beta = 0.053$; $\gamma = 0.1$. For this numerical experiment, we assumed that items are only classified as being stored in the wrong class if they were in the wrong class at least in the last period and in the current period (threshold $o = 2$) and if they were assumed to stay in the target class for at least one more period (threshold $u = 1$). Fig. 11 shows the results of scenario 1.

We compared the travel distances obtained with DEPSO for the case with no relocations to the results obtained using the dynamic storage location assignment algorithm. The relocation efforts are based on the travel distances resulting without relocations:

$$Relocation \quad effort_t\,[\%] = \frac{Relocation \quad effort_t\,[length \quad units]}{Travel \quad distance \quad without \quad relocation_t\,[length \quad units]} \cdot (-100\%)$$

In the nine periods studied here, relocation leads to travel distance reductions of 15.02% and relocation efforts of 2.79%. In total, the order picking effort can be reduced by 12.23% (travel distance reduction – relocation effort). As can be seen, the benefit of relocating items increases over time. On average, 0.28% of the items are relocated per period (Fig. 12).

Fig. 13 shows the results of scenario 2.

As in scenario 1, the performance improvement that results from relocating items improves over time. In total, travel distances can be reduced by 7.45%. The relocation efforts are 2.08%. In total, the order picking effort can be reduced by 5.37%. On average, 0.20% of the items are relocated per period (Fig. 14) in this scenario. The detailed results of the numerical experiments can be found in the Online Supplement, Part 5.

In summary, the dynamic storage location assignment algorithm reduces travel distances in both scenarios. With a relatively small number of relocated items, a significant reduction in travel distance could be obtained. As the method provides even better results for the more dynamic scenario 1, we conclude that it works well for companies facing highly fluctuating order patterns.

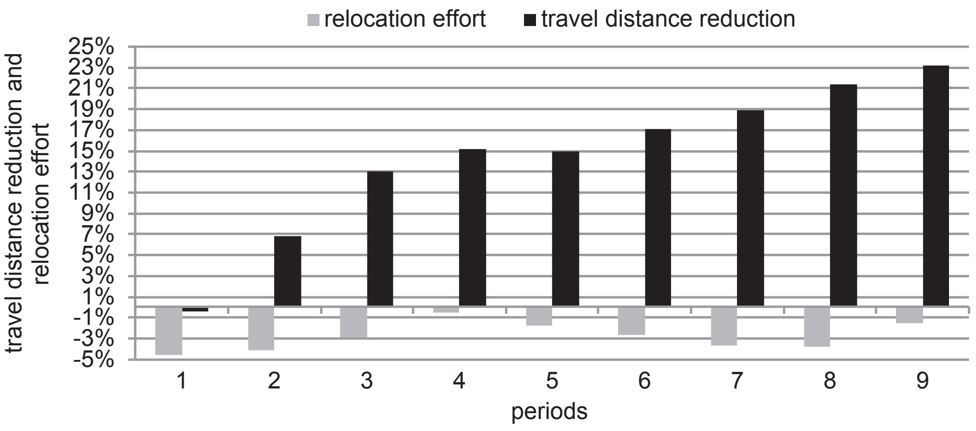


Fig. 11. Travel distance reduction and relocation effort in scenario 1.

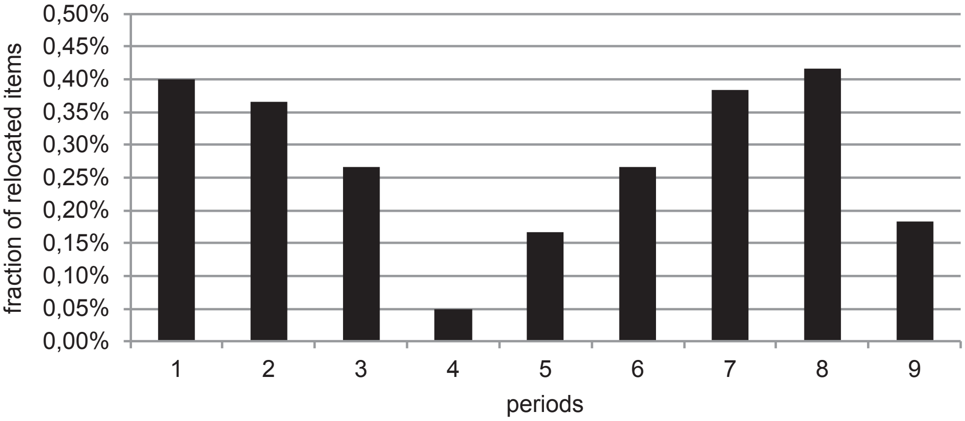


Fig. 12. Fraction of relocated items in scenario 1.

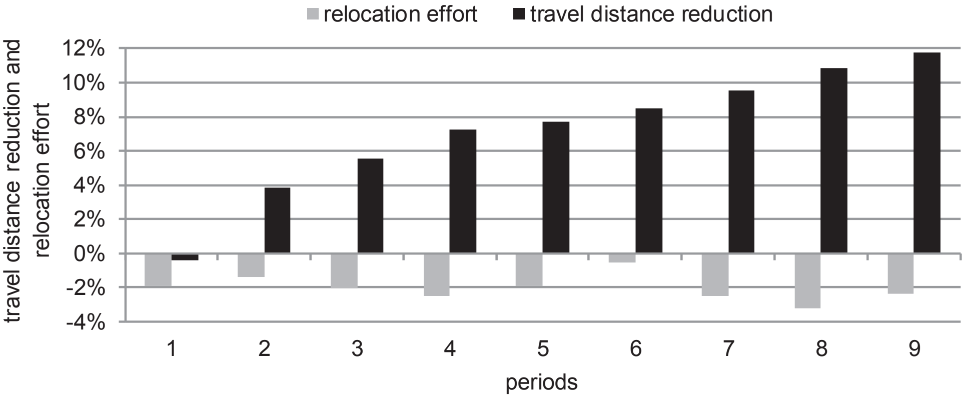


Fig. 13. Travel distance reduction and relocation effort in scenario 2.

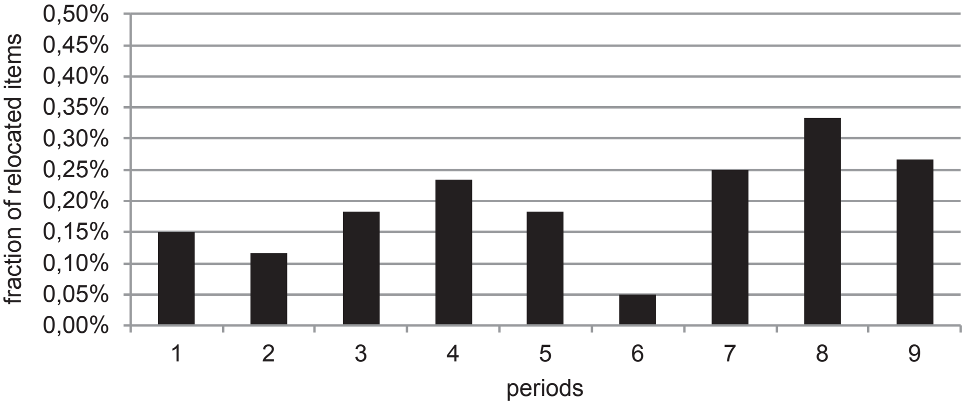


Fig. 14. Fraction of relocated items in scenario 2.

7. Conclusion

This paper investigated the joint storage location, order batching and picker routing problem in a dynamic environment. This problem is of considerable practical interest since it is a common problem companies face in a large number of industrial applications. Up to now, the literature did not propose an integrated method that facilitates solving this problem. In addition, previous work did not consider future customer demand and relocation effort when identifying promising relocations. The three planning problems are strongly interconnected and, in fact, only a simultaneous solution could result in a global optimum. Since this is not a realistic approach for large-scale problems encountered in practical applications, we developed a sequential iterative method in this paper. First, we proposed the DEPSO algorithm, which combines PSO with evolutionary computation, a savings algorithm and a 2-opt algorithm to solve the joint order batching and picker routing problem for a given storage location assignment. This hybrid approach helps to avoid some of the major drawbacks of the traditional PSO, which was reported to frequently arrive at a locally optimal solution too quickly. To further improve the optimization results and the computation times, we calculated an initial solution with the help of a savings algorithm. The routes were constructed by a 2-opt algorithm with short runtimes. In extensive numerical experiments, we showed that DEPSO outperforms algorithms that are frequently used in practice. Even for large-scale problems, the results could be computed in less than four minutes. This illustrates the practical applicability of the developed method.

This paper also developed a novel dynamic storage location assignment algorithm, which uses DEPSO to construct order batches and picker routes, and which assigns items into three classes based on their forecasted number of orderlines. With the help of an estimation procedure and DEPSO, travel distance reductions that may result from relocating items were calculated and compared to the required relocation effort. Items were relocated if the estimated reduction in travel distance exceeds the relocation effort. It could be shown that the method provides especially promising results for companies facing highly fluctuating demands.

Companies can benefit from the method proposed in this paper as follows: First, travel distances can be significantly reduced by solving the three problems in a single solution approach. This leads to high cost savings or the possibility to pick more items within the same time period. Secondly, further travel distance reductions can be obtained using the improved PSO, which computes results with short runtimes. Thirdly, the method allows aligning the storage assignment to forecasted customer demand. Quite large efficiency improvements can often be realized by just a few relocations. Fourthly, to guarantee that the proposed method is applicable in practice, we designed a method that is rather simple in terms of mathematical complexity and that is able to solve large-scale problems. Future research could analyze how the findings need to be adjusted to optimize other order picking systems, for example parts-to-picker systems, AS/RS, automated systems or systems with picking robots. Another interesting question is how different layout variants affect the results of the optimization. Moreover, the effect of picker blocking could be analyzed. Furthermore, future research could consider the due date of each order and solve the joint problem with the help of a multi-objective approach. Optimization results may be further improved by incorporating relocation orders resulting from the dynamic storage location assignment algorithm into the order batching and picker routing problem. The combination of regular picking and relocation processes may lead to further reductions in travel distance. The dynamic storage location assignment algorithm could be extended to facilitate the application of further storage location assignment strategies (as for example dedicated storage or family grouping). Another promising extension could be the consideration of different item or load carrier geometries. The integration of other forecasting methods, of ergonomic criteria or worker learning into the dynamic storage location assignment algorithm could also be examined. Finally, also the effect of storing an item in several storage locations could be analyzed.

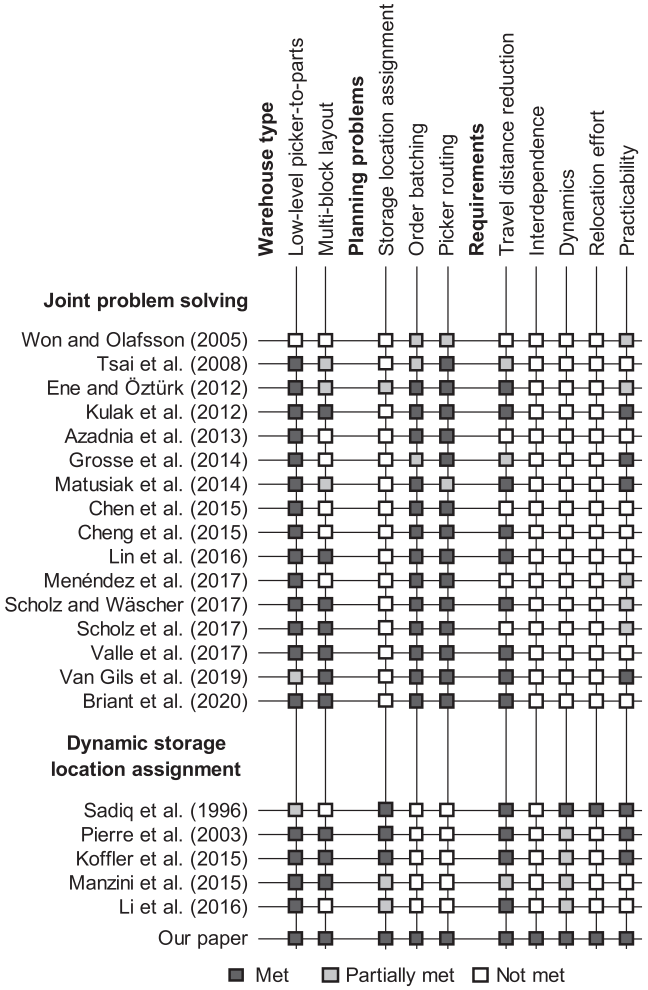
Funding sources

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

CRediT authorship contribution statement

Patrick Kübler: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Visualization. **Christoph H. Glock:** Conceptualization, Writing - original draft, Writing - review & editing, Supervision. **Thomas Bauernhansl:** Conceptualization, Supervision.

Appendix A. Contribution of this paper



	Warehouse type	Low-level picker-to-parts	Multi-block layout	**Planning problems**	Storage location assignment	Order batching	Picker routing	**Requirements**	Travel distance reduction	Interdependence	Dynamics	Relocation effort	Practicability
Joint problem solving													
Won and Olafsson (2005)		□	□		□	▨	▨		□	□	□	□	▨
Tsai et al. (2008)		■	▨		□	▨	■		▨	□	□	□	□
Ene and Öztürk (2012)		■	▨		▨	■	■		■	□	□	□	▨
Kulak et al. (2012)		■	■		□	■	■		■	□	□	□	■
Azadnia et al. (2013)		■	□		□	■	■		□	□	□	□	□
Grosse et al. (2014)		■	□		□	▨	■		▨	□	□	□	■
Matusiak et al. (2014)		■	▨		□	■	▨		■	□	□	□	■
Chen et al. (2015)		■	□		□	■	■		□	□	□	□	□
Cheng et al. (2015)		■	□		□	■	■		■	□	□	□	□
Lin et al. (2016)		■	■		□	■	■		■	□	□	□	□
Menéndez et al. (2017)		■	□		□	■	■		□	□	□	□	▨
Scholz and Wäscher (2017)		■	■		□	■	■		■	□	□	□	▨
Scholz et al. (2017)		■	■		□	■	■		□	□	□	□	▨
Valle et al. (2017)		■	■		□	■	■		■	□	□	□	□
Van Gils et al. (2019)		▨	■		□	■	■		■	□	□	□	■
Briant et al. (2020)		■	■		□	■	■		■	□	□	□	□
Dynamic storage location assignment													
Sadiq et al. (1996)		▨	□		■	□	□		■	□	■	■	■
Pierre et al. (2003)		■	■		■	□	□		■	□	▨	□	■
Koffler et al. (2015)		■	■		■	□	□		■	□	▨	□	■
Manzini et al. (2015)		■	■		▨	□	□		▨	□	▨	□	□
Li et al. (2016)		■	□		▨	□	□		■	□	▨	□	□
Our paper		■	■		■	■	■		■	■	■	■	■

■ Met ▨ Partially met □ Not met

Appendix B. Nearest neighbor heuristic

Step 1: Choose the depot as the starting node of the picking tour Pt and save it as the current node.

Step 2: Use the distance matrix to determine the distance between the current node and all other nodes that need to be visited, but that are not yet in the tour Pt. Choose the node which is closest to the current node and add it to the tour Pt. Save the new node as the new current node and repeat step 2 until no nodes are left that have not yet been visited.

Step 3: Choose the depot as the last node of the tour Pt.

Appendix C. 2-opt heuristic

Step 1: Calculate the length of the picking tour Pt that was constructed by the nearest neighbor heuristic and save it as the current travel distance Td.

Step 2: Choose position 1 of the current tour Pt as position q. Choose position 3 as r.

Step 3: Calculate the distance before ($d^{before} = d_{Pt(q),Pt(q+1)} + d_{Pt(r),Pt(r+1)}$) and after changing edges ($d^{after} = d_{Pt(q),Pt(r)} + d_{Pt(q+1),Pt(r+1)}$), where $d_{Pt(q),Pt(r)}$ is the distance between the nodes $Pt(q)$ and $Pt(r)$.

Step 4: Calculate $\Delta = d^{after} - d^{before}$. If $\Delta \geq 0$, discard the change and proceed as follows:

a) If r is not the second last position in the tour, keep q. Increase r by one and proceed with step 3.
b) If r is the second last position and q is not the fourth-to-last position, increase q by one and set $r = q + 2$. Proceed with step 3.
c) If q is the fourth-to-last position and r is the second last position, terminate and return Pt and Td.

If $\Delta < 0$, accept the change and set $Pt = (Pt(1{:}\ q), Pt(r), Pt((r-1){:}\ (-1){:}\ (q+1)), Pt((r+1){:}\ end))$.

Step 5: Calculate the length of the current tour, Td, and proceed with step 2.

Appendix D. Savings algorithm

Step 1: Calculate the savings $Sav_{k_1k_2} = Td_{k_1} + Td_{k_2} - Td_{k_1k_2}$ for every pair (k_1, k_2) of customer orders that satisfy the capacity restriction of the picker. Store the savings value in a savings list.

Step 2: Sort the savings list in decreasing order of the savings value.

Step 3: Choose the first order pair in the savings list that has not been examined yet. Three cases can be distinguished:

(1) Neither of the orders has been assigned to a batch. Open a new batch and assign both orders to it.
(2) One order has already been assigned to a batch. If there is sufficient capacity left, assign the second order to this batch. If not, choose the next order pair in the savings list.
(3) Both orders have already been assigned to a batch. Choose the next order pair in the savings list.

Continue with step 3 until all order pairs in the list have been evaluated.

Step 4: If there are still orders left that have not been assigned to a batch yet, open separate batches for each order and assign the orders to those batches. When all orders have been assigned to batches, terminate the algorithm.

Appendix E. Particle movement

Step 1: Start the procedure at position $h = 1$ in the permutation P_p of particle p.

Step 2: If $v_{p,h,t} = 1$ and $B_{p,G_{best}} > S^{G_{best}} \cdot rand$, choose G_{best} as comparative permutation P^{comp}. The variable *rand* represents a random variable uniformly distributed over [0,1]. In contrast to Tian et al. (2013), $S^{G_{best}}$ is introduced to allow the swarm to favor movements in the direction of G_{best} or P_p^{best}. If $v_{p,h,t} = -1$ and $B_{p,P_{best}} > rand$, choose P_p^{best} as comparative permutation P^{comp}. If none of the conditions is satisfied, proceed to step 5.

Step 3: Find position r in P_p which contains order $P^{comp}(h)$.

Step 4: If $v_{p,r,t} \neq 0$ or $v_{p,r,t} = 0$ and $rand < 0,5$, change orders $P_p(r)$ and $P_p(h)$ and recalculate the movement probabilities.

Step 5: Update the velocity $v_{p,h,t}$ by randomly selecting an element from the set $\{-1, 0, 1\}$. If the last position in P_p has not been reached yet, increase h by one and proceed with step 2, else terminate the procedure.

Appendix F. Mutation operation

Step 1: Calculate the particle's intensity Int_p.

Step 2: Choose particle $p = 1$ as the current particle.

Step 3: Calculate the particle's mutation probability M_p.

Step 4: If $M_p > rand$, calculate Cl_p, else increase p by one and proceed to step 3. Terminate the procedure if p was the last particle.

Step 5: If $Cl_p < 0,5$, apply the swap operator, if $0,5 \leq Cl_p < 0,8$, apply the shift operator, else apply the inverse operator.

Step 6: Randomly choose two positions in the velocity vector $V_{p,t}$ and change both values. Increase p by one and proceed with step 3. Terminate the procedure if p was the last particle.

Appendix G. Local search

Step 1: If $S^{StagG_{best}} > S^{Stag} \cdot rand$, set the iteration counter of the local search It^{LS} to zero and proceed with step 2, else terminate the procedure.

Step 2: Apply the swap operator on G_{best}.

Step 3: Assign orders to batches with the help of the first fit rule.

Step 4: Construct the tours with the help of the picking heuristics and calculate the travel distance. If the travel distance is shorter than before, set $S^{StagG_{best}}$ to zero, update G_{best}, choose a random particle, and place it on the position of G_{best} and terminate the procedure. If the travel distance is not shorter, increase It^{LS} by one and proceed with step 2. Terminate the procedure if the maximum number of iterations of the local search, It^{maxLS}, has been reached.

Appendix H. Detailed results of the numerical experiments for DEPSO

	DEPSO vs. SOP			*DEPSO vs. FCFS*			*DEPSO vs. Savings*			*runtime DEPSO [sec]*
Problem scenario	min.	Ø	max.	min.	Ø	max.	min.	Ø	max.	Ø
50_2_6	−87.50%	−88.52%	−89.37%	−32.66%	−38.98%	−48.61%	−17.12%	−30.03%	−38.54%	23.96
50_2_10	−86.04%	−87.13%	−88.25%	−36.73%	−44.00%	−48.69%	−26.54%	−33.36%	−41.96%	21.28
50_6_2	−86.23%	−87.36%	−88.48%	−24.98%	−30.74%	−36.97%	−23.22%	−28.35%	−36.82%	74.15
50_6_6	−80.18%	−81.54%	−84.61%	−36.91%	−41.09%	−48.75%	−30.71%	−35.92%	−41.48%	42.79
50_6_10	−75.49%	−77.49%	−80.20%	−38.04%	−44.20%	−50.11%	−29.99%	−36.31%	−42.50%	38.25
50_10_2	−83.83%	−84.83%	−86.56%	−18.74%	−28.61%	−34.26%	−15.85%	−25.90%	−32.07%	108.77
50_10_6	−74.69%	−76.61%	−78.83%	−34.44%	−38.75%	−43.04%	−26.33%	−32.60%	−37.59%	63.93
50_10_10	−68.30%	−70.60%	−73.90%	−38.03%	−42.37%	−46.18%	−26.95%	−33.87%	−37.96%	57.21
100_2_2	−92.29%	−92.74%	−93.43%	−31.29%	−36.46%	−43.34%	−18.53%	−26.29%	−33.78%	59.56
100_2_6	−89.76%	−90.79%	−91.41%	−40.31%	−47.77%	−53.42%	−26.00%	−34.77%	−39.34%	35.69
100_2_10	−87.43%	−88.63%	−89.56%	−45.77%	−51.33%	−54.92%	−31.86%	−38.24%	−44.05%	30.50
100_6_2	−87.92%	−89.20%	−90.26%	−24.20%	−30.78%	−35.11%	−21.29%	−26.55%	−32.89%	107.80
100_6_6	−81.34%	−82.77%	−84.61%	−38.24%	−42.66%	−47.14%	−31.32%	−35.36%	−39.40%	63.73
100_6_10	−76.62%	−78.51%	−80.30%	−41.30%	−45.74%	−49.09%	−29.87%	−36.43%	−40.70%	58.80
100_10_2	−84.70%	−86.23%	−87.76%	−24.14%	−29.22%	−35.04%	−22.49%	−26.40%	−30.49%	146.50
100_10_6	−75.48%	−77.36%	−79.54%	−38.08%	−41.29%	−45.44%	−29.73%	−33.56%	−36.36%	91.49
100_10_10	−69.20%	−71.25%	−74.83%	−41.58%	−44.12%	−46.84%	−29.62%	−33.65%	−36.71%	88.89
150_2_2	−93.74%	−94.22%	−94.64%	−29.00%	−34.15%	−39.97%	−18.52%	−24.76%	−30.73%	78.85
150_2_6	−90.30%	−91.23%	−92.65%	−46.81%	−50.26%	−55.19%	−28.10%	−34.75%	−40.20%	44.98
150_2_10	−88.50%	−89.25%	−90.16%	−45.77%	−51.97%	−55.94%	−32.30%	−35.80%	−39.95%	40.96
150_6_2	−88.74%	−89.34%	−90.06%	−28.85%	−32.23%	−36.94%	−23.05%	−28.04%	−31.86%	117.91
150_6_6	−81.32%	−82.89%	−84.07%	−38.98%	−43.43%	−46.62%	−30.10%	−35.08%	−38.98%	79.13
150_6_10	−76.54%	−78.45%	−79.67%	−43.67%	−46.16%	−48.21%	−32.03%	−35.84%	−38.94%	71.21
150_10_2	−85.47%	−86.22%	−87.04%	−24.56%	−29.21%	−35.75%	−20.73%	−24.83%	−28.74%	185.54
150_10_6	−76.02%	−77.32%	−78.65%	−37.19%	−40.74%	−43.23%	−29.21%	−32.25%	−35.59%	114.83
150_10_10	−70.28%	−71.41%	−72.88%	−42.42%	−44.38%	−47.21%	−29.71%	−32.15%	−35.38%	117.10
200_2_2	−93.83%	−94.26%	−94.65%	−36.54%	−41.36%	−45.51%	−20.65%	−26.89%	−32.32%	82.87
200_2_6	−90.69%	−91.54%	−92.10%	−47.30%	−51.42%	−55.07%	−27.60%	−33.72%	−38.08%	52.11
200_2_10	−88.43%	−89.26%	−89.95%	−50.27%	−53.33%	−57.27%	−30.25%	−35.29%	−40.23%	50.53
200_6_2	−88.69%	−89.61%	−90.15%	−27.62%	−31.43%	−35.35%	−22.90%	−26.41%	−31.27%	138.78
200_6_6	−81.17%	−82.58%	−83.94%	−38.89%	−41.96%	−44.15%	−29.02%	−33.17%	−36.30%	97.02
200_6_10	−77.19%	−78.35%	−80.04%	−43.56%	−45.24%	−48.21%	−29.40%	−34.01%	−36.77%	91.46
200_10_2	−85.42%	−86.11%	−87.13%	−25.49%	−28.26%	−31.57%	−20.97%	−24.02%	−28.02%	226.18
200_10_6	−75.60%	−77.21%	−78.16%	−37.99%	−40.34%	−42.47%	−28.11%	−31.02%	−33.45%	138.90
200_10_10	−70.31%	−71.57%	−72.62%	−41.01%	−43.94%	−46.19%	−29.88%	−31.79%	−34.19%	138.01

Appendix I. Supplementary material

Supplementary data to this article can be found online at https://doi.org/10.1016/j.cie.2020.106645.

References

Ang, M., Lim, Y. F., & Sim, M. (2012). Robust storage assignment in unit-load warehouses. *Management Science, 58*(11), 2114–2130.

Azadnia, A. H., Taheri, S., Ghadimi, P., Mat Saman, M. Z., & Wong, K. Y. (2013). Order Batching in warehouses by minimizing total tardiness: A hybrid approach of weighted association rule mining and genetic algorithms. *The Scientific World Journal, 2013*.

Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences, 237*, 82–117.

Boysen, N., de Koster, R., & Weidinger, F. (2018). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research.*.

Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A. L., & Ogier, M. (2020). An efficient and general approach for the joint order batching and picker routing problem. *European Journal of Operational Research, 285*(2), 497–512.

Cambazard, H., & Catusse, N. (2018). Fixed-parameter algorithms for rectilinear Steiner tree and rectilinear traveling salesman problem in the plane. *European Journal of Operational Research, 270*(2), 419–429.

Cano, J. A., Correa-Espinal, A. A., Gómez-Montoya, R. A., & Cortés, P. (2019). Genetic algorithms for the picker routing problem in multi-block warehouses. *International Conference on Business Information Systems* (pp. 313–322). Cham: Springer.

Cergibozan, Ç., & Tasan, A. S. (2019). Order batching operations: an overview of classification, solution techniques, and future research. *Journal of Intelligent Manufacturing, 30*(1), 335–349.

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

Chen, T. L., Cheng, C. Y., Chen, Y. Y., & Chan, L. K. (2015). An efficient hybrid algorithm for integrated order batching, sequencing and routing problem. *International Journal of Production Economics, 159*, 158–167.

Cheng, C. Y., Chen, Y. Y., Chen, T. L., & Yoo, J. J. W. (2015). Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem. *International Journal of Production Economics, 170*, 805–814.

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research, 12*(4), 568–581.

Das, S., Abraham, A., & Konar, A. (2008). Swarm intelligence algorithms in bioinformatics. *Computational Intelligence in Bioinformatics* (pp. 113–147). Berlin, Heidelberg: Springer.

Davarzani, H., & Norrman, A. (2015). Toward a relevant agenda for warehousing research: literature review and practitioners' input. *Logistics Research, 8*(1), 1.

De Koster, R., Johnson, A. L., & Roy, D. (2017). Warehouse design and management. *International Journal of Production Research, 55*(21), 6327–6330.

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

De Koster, M. B. M., Van der Poort, E. S., & Wolters, M. (1999). Efficient order batching methods in warehouses. *International Journal of Production Research, 37*(7), 1479–1504.

Ene, S., & Öztürk, N. (2012). Storage location assignment and order picking optimization in the automotive industry. *The International Journal of Advanced Manufacturing Technology, 60*(5–8), 787–797.

Englert, M., Röglin, H., & Vöcking, B. (2014). Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. *Algorithmica, 68*(1), 190–264.

Gademann, N., & Velde, S. (2005). Order batching to minimize total travel time in a parallel-aisle warehouse. *IIE transactions, 37*(1), 63–75.

Gong, Y., & De Koster, R. B. (2011). A review on stochastic models and analysis of warehouse operations. *Logistics Research, 3*(4), 191–205.

Grosse, E. H., Glock, C. H., & Ballester-Ripoll, R. (2014). A simulated annealing approach for the joint order batching and order picker routing problem with weight restrictions. *International Journal of Operations and Quantitative Management, 20*(2), 65–83.

Grosse, E. H., Glock, C. H., & Jaber, M. Y. (2013). The effect of worker learning and forgetting on storage reassignment decisions in order picking systems. *Computers & Industrial Engineering, 66*(4), 653–662.

Grosse, E. H., Glock, C. H., & Neumann, W. P. (2017). Human factors in order picking: a content analysis of the literature. *International Journal of Production Research, 55*(5), 1260–1276.

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

Hausman, W. H., Schwarz, L. B., & Graves, S. C. (1976). Optimal storage assignment in automatic warehousing systems. *Management Science, 22*(6), 629–638.

Hembecker, F., Lopes, H. S., & Godoy, W. (2007). Particle swarm optimization for the multidimensional knapsack problem. *International Conference on Adaptive and Natural Computing Algorithms* (pp. 358–365). Berlin, Heidelberg: Springer.

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

Ho, Y. C., & Tseng, Y. Y. (2006). A study on order-batching methods of order-picking in a distribution centre with two cross-aisles. *International Journal of Production Research, 44*(17), 3391–3417.

Ho, Y. C., Su, T. S., & Shi, Z. B. (2008). Order-batching methods for an order-picking warehouse with two cross aisles. *Computers & Industrial Engineering, 55*(2), 321–347.

Hsieh, L. F., & Huang, Y. C. (2011). New batch construction heuristics to optimise the performance of order picking systems. *International Journal of Production Economics, 131*(2), 618–630.

Kashan, A. H., Kashan, M. H., & Karimiyan, S. (2013). A particle swarm optimizer for grouping problems. *Information Sciences, 252*, 81–95.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Neural Networks, 1995 Proceedings, IEEE International Conference on 4* (pp. 1942–1948). .

Kennedy, J., & Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation, 1997 IEEE International Conference on 5* (pp. 4104–4108). .

Koch, S. (2014). *Genetische Algorithmen für das Order Batching-Problem in manuellen Kommissioniersystemen.* Springer-Verlag.

Kofler, M., Beham, A., Wagner, S., & Affenzeller, M. (2015). Robust storage assignment in warehouses with correlated demand. *Computational Intelligence and Efficiency in Engineering Systems* (pp. 415–428). Cham: Springer.

Kulak, O., Sahin, Y., & Taner, M. E. (2012). Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. *Flexible services and manufacturing journal, 24*(1), 52–80.

Lam, H. T., Nicolaevna, P. N., & Quan, N. T. M. (2007). A heuristic particle swarm optimization. *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (pp. 174). .

Le-Duc, T., & De Koster, R. M. B. (2005). Travel distance estimation and storage zone optimization in a 2-block class-based storage strategy warehouse. *International Journal of Production Research, 43*(17), 3561–3581.

Li, J., Moghaddam, M., & Nof, S. Y. (2016). Dynamic storage assignment with product affinity and ABC classification—a case study. *The International Journal of Advanced Manufacturing Technology, 84*(9–12), 2179–2194.

Lin, C. C., Kang, J. R., Hou, C. C., & Cheng, C. Y. (2016). Joint order batching and picker Manhattan routing problem. *Computers & Industrial Engineering, 95*, 164–174.

Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., & Winkler, R. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of forecasting, 1*(2), 111–153.

Makridakis, S., Chatfield, C., Hibon, M., Lawrence, M., Mills, T., Ord, K., & Simmons, L. F. (1993). The M2-competition: A real-time judgmentally based forecasting study. *International Journal of Forecasting, 9*(1), 5–22.

Makridakis, S., & Hibon, M. (2000). The M3-Competition: results, conclusions and implications. *International journal of forecasting, 16*(4), 451–476.

Manzini, R., Accorsi, R., Gamberi, M., & Penazzi, S. (2015). Modeling class-based storage assignment over life cycle picking patterns. *International Journal of Production Economics, 170*, 790–800.

Manzini, R., Gamberi, M., Persona, A., & Regattieri, A. (2007). Design of a class based storage picker to product order picking system. *The International Journal of Advanced Manufacturing Technology, 32*(7–8), 811–821.

Marchet, G., Melacini, M., & Perotti, S. (2015). Investigating order picking system adoption: A case-study-based approach. *International Journal of Logistics Research and Applications, 18*(1), 82–98.

Masae, M., Glock, C. H., & Grosse, E. H. (2020). Order picker routing in warehouses: A systematic literature review. *International Journal of Production Economics, 224*.

Matusiak, M., de Koster, R., Kroon, L., & Saarinen, J. (2014). A fast simulated annealing method for batching precedence-constrained customer orders in a warehouse. *European Journal of Operational Research, 236*(3), 968–977.

Menéndez, B., Bustillo, M., Pardo, E. G., & Duarte, A. (2017). General Variable Neighborhood Search for the Order Batching and Sequencing Problem. *European Journal of Operational Research, 263*(1), 82–93.

Moon, G., & Kim, G. P. (2001). Effects of relocation to AS/RS storage location policy with production quantity variation. *Computers & Industrial Engineering, 40*(1–2), 1–13.

Muppani, V. R., & Adil, G. K. (2008). A branch and bound algorithm for class based storage location assignment. *European Journal of Operational Research, 189*(2), 492–507.

Muralidharan, B., Linn, R. J., & Pandit, R. (1995). Shuffling heuristics for the storage location assignment in an AS/RS. *The International Journal of Production Research, 33*(6), 1661–1672.

Önüt, S., Tuzkaya, U. R., & Doğaç, B. (2008). A particle swarm optimization algorithm for the multiple-level warehouse layout design problem. *Computers & Industrial Engineering, 54*(4), 783–799.

Pan, J. C. H., Shih, P. H., & Wu, M. H. (2012). Storage assignment problem with travel distance and blocking considerations for a picker-to-part order picking system. *Computers & Industrial Engineering, 62*(2), 527–535.

Pansart, L., Catusse, N., & Cambazard, H. (2018). Exact algorithms for the order picking problem. *Computers & Operations Research, 100*, 117–127.

Petersen, C. G., & Aase, G. (2004). A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics, 92*(1), 11–19.

Petersen, C. G., & Schmenner, R. W. (1999). An evaluation of routing and volume-based storage policies in an order picking operation. *Decision Sciences, 30*(2), 481–501.

Pierre, B., Vannieuwenhuyse, B., Dominanta, D., & Van Dessel, H. (2004). Dynamic ABC storage policy in erratic demand environments. *Jurnal Teknik Industri, 5*(1), 1–12.

Rao, S. S., & Adil, G. K. (2013). Optimal class boundaries, number of aisles, and pick list size for low-level order picking systems. *IIE Transactions, 45*(12), 1309–1321.

Reschke, V. (2013). Lagerplatzvergabe in Person-zur-Ware-Kommissioniersystemen. Shaker.

Roodbergen, K. J., & de Koster, R. (2001). Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research, 39*(9), 1865–1883.

Rouwenhorst, B., Reuter, B., Stockrahm, V., van Houtum, G. J., Mantel, R. J., & Zijm, W. H. (2000). Warehouse design and control: Framework and literature review. *European Journal of Operational Research, 122*(3), 515–533.

Sadiq, M., Landers, T. L., & Don Taylor, G. (1996). An assignment algorithm for dynamic picking systems. *IIE Transactions, 28*(8), 607–616.

Scholz, A., Schubert, D., & Wäscher, G. (2017). Order picking with multiple pickers and due dates–Simultaneous solution of Order Batching, Batch Assignment and Sequencing, and Picker Routing Problems. *European Journal of Operational Research, 263*(2), 461–478.

Scholz, A., & Wäscher, G. (2017). Order Batching and Picker Routing in manual order picking systems: the benefits of integrated routing. *Central European Journal of Operations Research, 25*(2), 491–520.

Schröder, M. (2012). Einführung in die kurzfristige Zeitreihenprognose und Vergleich der einzelnen Verfahren. In Prognoserechnung (pp. 11–45). Physica, Heidelberg.

Schuhr, R. (2012). Einführung in die Prognose saisonaler Zeitreihen mithilfe exponentieller Glättungstechniken und Vergleich der Verfahren von Holt/Winters und Harrison. In Prognoserechnung (pp. 47–73). Physica, Heidelberg.

Sedighizadeh, D., & Masehian, E. (2009). Particle swarm optimization methods, taxonomy and applications. *International Journal of Computer Theory and Engineering, 1*(5), 486.

Shqair, M., Altarazi, S., & Al-Shihabi, S. (2014). A statistical study employing agent-based modeling to estimate the effects of different warehouse parameters on the distance traveled in warehouses. *Simulation Modelling Practice and Theory, 49*, 122–135.

Silver, E. A., Pyke, D. F., & Thomas, D. J. (2016). *Inventory and production management in supply chains.* CRC Press.

Theys, C., Bräysy, O., Dullaert, W., & Raa, B. (2010). Using a TSP heuristic for routing order pickers in warehouses. *European Journal of Operational Research, 200*(3), 755–763.

Tian, Y., Liu, D., Yuan, D., & Wang, K. (2013). A discrete PSO for two-stage assembly scheduling problem. *The International Journal of Advanced Manufacturing Technology, 66*(1–4), 481–499.

Tompkins, J. A., White, J. A., Bozer, Y. A., & Tanchoco, J. M. A. (2010). *Facilities planning.* John Wiley & Sons.

Tsai, C. F., Tsai, C. W., & Tseng, C. C. (2004). A new hybrid heuristic approach for solving large traveling salesman problem. *Information Sciences, 166*(1–4), 67–81.

Tsai, C. Y., Liou, J. J., & Huang, T. M. (2008). Using a multiple-GA method to solve the batch picking problem: considering travel distance and order due time. *International Journal of Production Research, 46*(22), 6533–6555.

Valle, C. A., Beasley, J. E., & da Cunha, A. S. (2017). Optimally solving the joint order batching and picker routing problem. *European Journal of Operational Research, 262*(3), 817–834.

Van Gils, T., Caris, A., Ramaekers, K., & Braekers, K. (2019). Formulating and solving the integrated batching, routing, and picker scheduling problem in a real-life spare parts warehouse. *European Journal of Operational Research, 277*(3), 814–830.

Van Gils, T., Ramaekers, K., Caris, A., & de Koster, R. B. (2018). Designing Efficient Order Picking Systems by Combining Planning Problems: State-of-the-art Classification and Review. *European Journal of Operational Research, 267*(1), 1–15.

Van Nieuwenhuyse, I., & de Koster, R. B. (2009). Evaluating order throughput time in 2-block warehouses with time window batching. *International Journal of Production Economics, 121*(2), 654–664.

Vaughan, T. S. (1999). The effect of warehouse cross aisles on order picking efficiency. *International Journal of Production Research, 37*(4), 881–897.

Wäscher, G. (2004). Order picking: A survey of planning problems and methods. *Supply Chain Management and Reverse Logistics* (pp. 323–347). .

Won, J., & Olafsson, S. (2005). Joint order batching and order picking in warehouse operations. *International Journal of Production Research, 43*(7), 1427–1442.

Wruck, S., Vis, I. F., & Boter, J. (2017). Risk control for staff planning in e-commerce warehouses. *International Journal of Production Research, 55*(21), 6453–6469.