Applied Intelligence manuscript No.
(will be inserted by the editor)

Optimizing Storage Assignment, Order Picking, and their Interaction in Mezzanine Warehouses

Veronika Lesch · Patrick B.M. Müller · Moritz Krämer · Marius Hadry · Samuel Kounev · Christian Krupitzer

Received: date / Accepted: date

Abstract In warehouses, order picking is known to be the most labor-intensive and costly task in which the employees account for a large part of the warehouse performance. Hence, many approaches exist, that optimize the order picking process based on diverse economic criteria. However, most of these approaches focus on a single economic objective at once and disregard ergonomic criteria in their

Funding: No funding was received to assist with the preparation of this manuscript.
Conflicts of Interest: The authors have no relevant financial or non-financial interests to disclose.
Availability of data and material: Not applicable.
Code availability: Not applicable.

V. Lesch (corresponding author)
University of Würzburg
Würzburg
Germany
E-mail: veronika.lesch@uni-wuerzburg.de

P.B.M Müller
University of Applied Sciences Würzburg-Schweinfurt
Würzburg
Germany

Moritz Krämer
io-consultants GmbH & Co. KG
Heidelberg
Germany

M. Hadry
University of Würzburg
Würzburg
Germany

S. Kounev
University of Würzburg
Würzburg
Germany

C. Krupitzer
University of Hohenheim
Stuttgart
Germany

optimization. Further, the influence of the placement of the items to be picked is underestimated and accordingly, too little attention is paid to the interdependence of these two problems. In this work, we aim at optimizing the storage assignment and the order picking problem within mezzanine warehouse with regards to their reciprocal influence. We propose a customized version of the Non-dominated Sorting Genetic Algorithm II (NSGA-II) for optimizing the storage assignment problem as well as an Ant Colony Optimization (ACO) algorithm for optimizing the order picking problem. Both algorithms incorporate multiple economic and ergonomic constraints simultaneously. Furthermore, the algorithms incorporate knowledge about the interdependence between both problems, aiming to improve the overall warehouse performance. Our evaluation results show that our proposed algorithms return better storage assignments and order pick routes compared to commonly used techniques for the following quality indicators for comparing Pareto fronts: Coverage, Generational Distance, Euclidian Distance, Pareto Front Size, and Inverted Generational Distance. Additionally, the evaluation regarding the interaction of both algorithms shows a better performance when combining both proposed algorithms.

Keywords Storage assignment, order picking, interaction, genetic algorithm, ant colony optimization, mezzanine warehouse

1 Introduction

Warehouses play a central role in the supply chain of a company and contribute to its logistical success. When employing humans, picker-to-parts and parts-to-picker methods are differentiated [12]. Experts estimate the picker-to-parts system to be the most common in Western Europe with a share of over 80% [13]. A well-known picker-to-parts system is the mezzanine warehouse which we address in this work.

Working within a mezzanine warehouse consists of two main tasks: (i) filling the storage with goods (storage assignment) and (ii) picking items out of the storage (order picking). The storage assignment problem defines the task of selecting storage locations to put a product into storage. The order picking problem defines the task of computing a pick route that collects the requested products of a customer order. Finding suitable storage allocations is important, as the allocation of products affects the travel distances during order picking. Due to the NP-hardness and, hence, the complexity of the storage assignment and the order picking problem, efficient optimization algorithms are required to find satisfying solutions within acceptable times. In the literature, many approaches exist for optimizing both warehouse problems. However, most approaches usually target either of the warehouse problems; some works target both problems, however miss to integrate the interrelation between them and view each problem separately [8]. However, as identified by [9], warehouse problems are strongly coupled. Thus, optimizing each warehouse problem individually may yield suboptimal solutions, harming the overall warehouse performance. Since the employees spend most time traveling in such a mezzanine warehouse [13], it is not surprising that most approaches focus on optimizing the travel distance. Additionally, ergonomic constraints are rarely considered, even though mezzanine warehouses represent labor-intensive working environments.

In this paper, we propose an integrated approach for combined storage assignment and order picking that simultaneously optimizes multiple economic and ergonomic constraints in mezzanine warehouses. Expert interviews have shown, that in practice the following set of economic criteria is important and, hence, supported by our approach: products should be spread equally among each floor, fast-moving products should be easily accessible, correlated products should be stored in proximity of each other, and the storage space should be used as efficiently as possible. Further, we integrate ergonomic constraints such as storing heavy products and fast-moving products at grip height or reducing the requirement to switch a mezzanine floor. In an evaluation using three simulated mezzanine warehouses of different sizes, we analyze the quality of the solutions returned by our algorithms compared to commonly used techniques. Finally, we assess the quality improvement when combining both of our algorithms compared to an isolated application. Hence, the contribution of this paper is threefold:

1. Design of storage allocation and order picking algorithms that incorporate the interdependence of both tasks.
2. Integration of diverse economic and ergonomic constraints.
3. Evaluation of the approach in a use case based on real-world data provided by our cooperation company.

The remainder of this paper is structured as follows. Section 2 presents related work and delineates our paper from existing approaches. Section 3 presents the meta-model and floor layout of considered mezzanine warehouses. Afterwards, Section 4 provides an overview of the goal and a 3-phase algorithm of our storage assignment approach, while Section 5 presents the details of the proposed Genetic Algorithm for storage assignment. Then, Section 6 shows our order picking approach based on an adapted Ant Colony Optimization (ACO) algorithm. Section 7 presents our evaluation methodology and discusses the results and threats to validity. Finally, Section 8 concludes the paper and summarizes future work.

2 Related Work

In the literature, diverse storage assignment policies exist such as the dedicated and the random storage policy [2], the closest open location storage policy [13], rank-based storage policies [24]. Further, class-based, golden zone, and family grouping storage policies are introduced in the literature [13,23]. Additionally, diverse approaches apply optimization techniques. [28] propose a Particle Swarm Optimization algorithm for warehouses that deploy the class-based storage policy. [14] presents a mixed integer programming model for optimizing the storage assignment problem for class-based assigned warehouses. [11] apply local search algorithms for reorganizing the products in the warehouse to keep it operating efficiently. [19] propose a multi-objective genetic algorithm for optimizing the storage assignment problem in automated storage/retrieval warehouses.

Similarly, heuristic policies exist for the order picking problem such as the S-Shape, Return, Mid-Point, Largest Gap, and Combined heuristic [22,26,29]. Besides, [25] presents an optimal algorithm using dynamic programming to find the shortest pick route in a single-block warehouse. Additionally, [5] propose a mathematical model in combination with construction heuristics and apply Tabu

Search to construct order picking routes. [7] present an integer programming model for optimizing the order picking problem. [32] propose an Max-Min Ant System (MMAS) algorithm for optimizing machine travel paths in automated storage/retrieval warehouses. [4] propose an ACO algorithm that detects congestion situations that arise when multiple order pickers traverse the same pick aisle simultaneously. [16] presents a comparison on different multi-objective evolutionary algorithms for order picking and storage assignment. Other approaches apply the information of orders and order picking to optimize the storage assignment [30, 21]. However, such approaches are not flexible enough in our setting which is not order-driven and, hence, dynamic interactions between storage assignment and order picking are required to be considered.

Finally, related work also assess the interaction of storage assignment and order picking approaches. [24] and [8] provide an overview of well-performing combinations of storage assignment strategies and routing heuristics. [20] analyze different parameters that affect the travel time in single-block warehouses that deploy the class-based storage policy. [27] study the effects of different parameters on the travel distance in multi-block warehouses.

Our work delineates from these existing approaches in diverse aspects. First of all, our work applies optimization techniques and does not rely on a policy on how to select fitting storage racks or shortest pick routes. Second, regarding existing optimization approaches, our work integrates multiple objectives at once considering economic as well as ergonomic constraints at once while most of the other approaches focus on a single economic goal. Similarly, the authors of [3] also integrate ergonomic considerations, but do not focus on optimizing the storage assignment but rather the order picking only. Finally, in contrast to existing work that address the influence of storage assignment and order picking tasks, we designed algorithms that optimize the targets of both tasks. Hence, they optimize storage assignment and order picking with regards to the interdependence of both algorithms, while other works only provide well-performing combinations of algorithms or perform parameter tuning. Similarly, [16] also integrates both activities, but do not focus on a multi-objective approach,especially neglecting the ergonomic constraints. Also the work of [17] integrates both activities, however, they apply a digital twin based approach rather than considering multi-objective optimization algorithms.

3 Meta-Model of Considered Mezzanine Warehouses and Overview on the Processes

The storage assignment and order picking algorithm require information on the warehouse layout, the product assortment, the products' storage locations, and the current state of the warehouse. Figure 1 illustrates our proposed meta-model. The blue box describes the floor layout defining the arrangement of racks within one floor of the mezzanine warehouse (`FloorLayout`). Each floor consists of the classes, `P/D-Point`, `WidePickAisle`, and `Rack`. A p/d-point is the pickup and delivery point where personal needs to collect items to be stored in the warehouse or deliver items of a customer order that were collected. Regard the class `WidePickAisle`, two types of pick aisles exist: wide and narrow pick aisles. In wide pick aisles, pickers can take along their pick cart to cross the aisle while it needs to be parked at the aisle

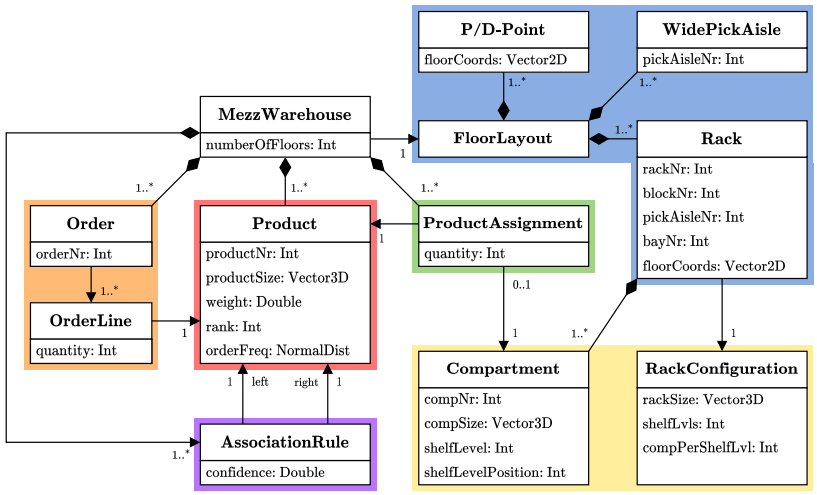


Fig. 1: The meta-model describes the structure and state of mezzanine warehouses.

entry for narrow pick aisles. A floor can be illustrated as a two-dimensional map as depicted in Figure 2: The racks with their unique identifiers r_3 and r_4 are assigned the floor coordinates $x = 1$ and $y = 2$ since their access points are both located at $(1|2)$. The vertical aisles located at $x = 0$ and $x = 4$, as well as the horizontal cross aisles at $y = 0$, $y = 4$, and $y = 7$, form the periphery of the floor. Periphery aisles usually contain the p/d-points (e.g. at $(2|0)$). A wide pick aisle is depicted at x-coordinate two and two narrow pick aisles are shown at x-coordinates one and three, where the picker needs to park his pick cart. Real-world mezzanine warehouses may apply different layouts on each floor; however, we assume that each floor in the mezzanine warehouse applies the same layout.

Since diagonal movements are not possible in this layout, the Manhattan distance function is applied to calculate the distance between two locations p and q (with n being the number of dimensions of the coordinates for p and q):

$$distance(p, q) = \sum_{i=1}^{n} |p_i - q_i| \tag{1}$$

The classes inside the yellow box (`Compartment` and `RackConfiguration`) define the configuration of a rack, referring to its size, the number of shelf levels, and the number of compartments per shelf level. The `Compartment` class includes an identifier and a three-dimensional vector specifying the compartment's dimensions. The shelf level and the shelf level position defines the compartment's location within the rack. The class `Product` defines the products using five properties: product number, size, weight, rank, and order frequency. The rank (≥ 1) allows identifying fast and slow-moving products by the frequency at which the product appears in recent customer orders. The product of rank 1 represents the most frequently ordered product. The order frequency describes the frequency to which a product is usually ordered using a gaussian distribution. Both properties are derived from recent customer orders and represent redundant information which prevents the algorithms from recalculating this information each time they need

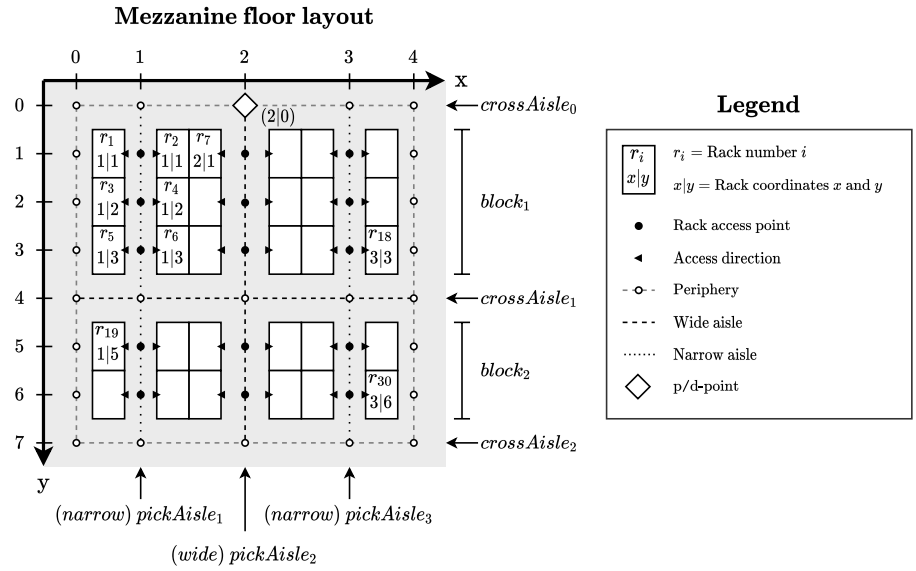
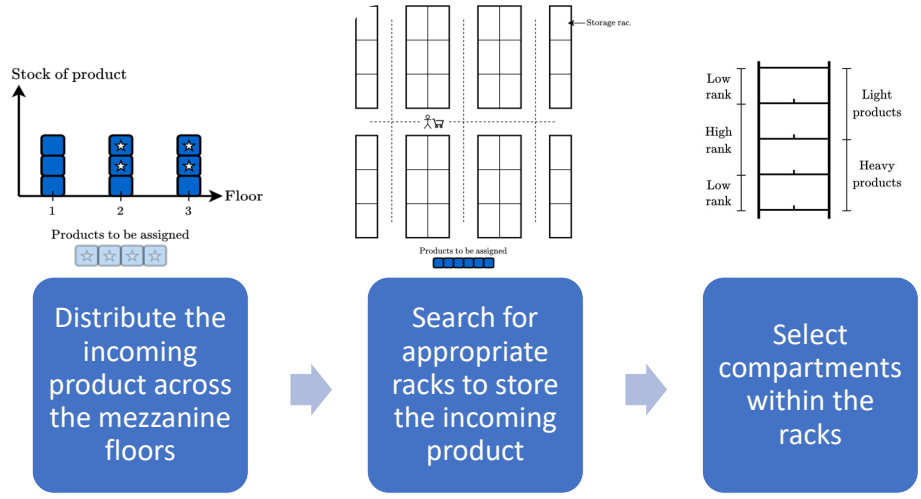


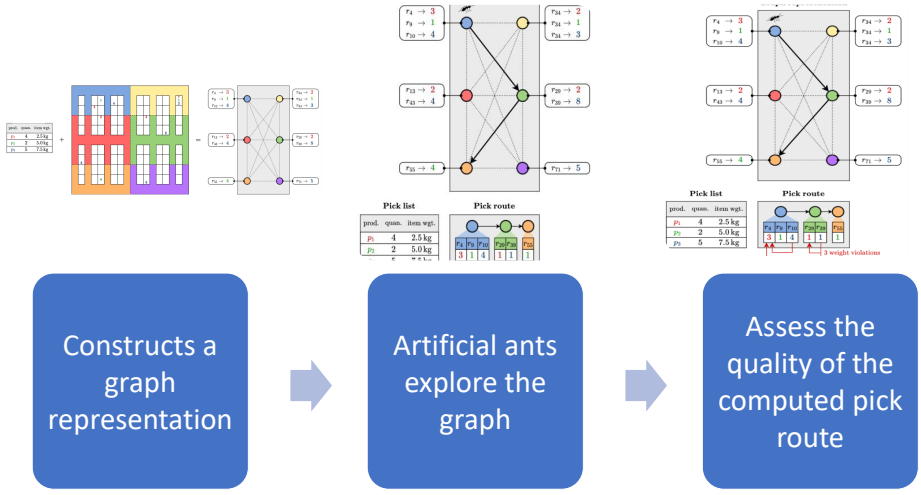
Fig. 2: Example mezzanine floor layout from top-down view.

it. Further, these properties are later used in the storage assignment optimization to find better racks regarding their frequency and usual ordered amount. The class `ProductAssignment` specifies the quantity of which a product is assigned to a specific compartment. The classes `Order` and `OrderLine` of the orange package define the structure of a customer order consisting of a unique order number and multiple order lines. An order line specifies the quantity to which a product is ordered. The class `AssociationRule` defines association rules derived by the Apriori algorithm [10]. The confidence ranges from 0 to 1 and expresses the strength of the correlation between the left-sided and the right-sided set of products. These rules are used in the storage assignment algorithm later on to store correlated products close to each other which may increase the order picking performance.

Figure 3 presents the two processes for the storage assignment as well as order picking. Storage assignment relies on the NSGA-II algorithm for optimal distribution of the items in the mezzanine warehouse (see Figure 3a). First, the algorithm distributes the incoming product across the mezzanine floors with the goal to reduce the need for changing floors during order picking. Second, on each floor, the algorithm searches for appropriate racks to store the incoming product based on the following four objectives: frequently requested quantity, spread items, distance to pick-up/delivery points, and correlated products. The NSGA-II algorithm shall optimize the four objectives. Third, the NSGA-II algorithm selects compartments for storing the incoming product based on two criteria. (1) Fast-moving products should be assigned to compartments at grip height. (2) Heavy products should be assigned to lower-level compartments. Order picking also follows a three-step approach (see Figure 3b). First, the algorithm constructs a graph representation based on the information of the pick list as well as the mezzanine floor layout. Second, the ACO algorithm applies artificial ants that explore the graph. Third,



(a) The figure shows the process for the storage assignment using the NSGA-II optimiaztion.



(b) The figure shows the process for the order picking using the ACO algorithm.

Fig. 3: Schematic overview on the storage assignment (above) and order picking (below) processes.

the quality of the computed pick route is assessed based on the travel distance (economic goal) and the weight violations of the items (ergonomic goal). In the following, we discuss those different algorithms.

4 Storage Assignment

The overall goal of the storage assignment algorithm is to select a set of compartments for storing an incoming product by considering multiple economic and ergonomic constraints simultaneously.

4.1 Constraints and Assumptions

In expert interviews, we identified multiple hard constraints that should be covered in our approaches. These hard constraints specify whether a storage allocation is considered feasible and a feasible solution never violates any of these constraints: Each incoming item must be assigned to a compartment (HC_1). The selected compartment must either be empty or partially occupied by items of the same product (HC_2). Each item has to fit in the remaining free space if its compartment (HC_3). Furthermore, we define multiple soft constraints that measure the extent to which a storage allocation fulfills economic criteria: The products should be evenly spread on each floor (SC_1). Fast-moving products should be assigned close to a p/d-point (SC_2). The mean ordered quantity of a product should be locally available (SC_3). Correlated products should be stored close to each other (SC_4). The storage space should be used as efficiently as possible (SC_5). Finally, we define two ergonomic soft constraints: Heavy products should be stored at grip height (SC_6). Fast-moving products should be assigned to compartments at grip height (SC_7).

Further, we state the following assumptions for our approach: The state of the warehouse does not change while the storage assignment algorithm is running. Thus, the products are not repositioned nor removed, and the racks' configurations do not change. Further, the algorithm allocates only one product at a time. The storage racks may apply different rack configurations and products may only be assigned to fitting compartments. A compartment is allowed to store multiple items of the same product but may not store two different products at the same time. Finally, product ranks and association rules are derived from recent customer orders.

4.2 3-Phase Storage Assignment Algorithm

Our storage assignment algorithm consists of three phases that intend to reduce the complexity of the optimization problem: (i) assignment of products to floors, (ii) assignment to racks w.r.t. economic criteria, and (iii) assignment to compartments w.r.t. ergonomic criteria.

In the first phase, the incoming product quantity is split among the mezzanine floors (SC_1) so that each floor provides the same quantity of the product. This way, we try to reduce the required floor changes during a pick route to a minimum. Thus, we first determine the total quantity of the incoming product that is already available in each floor, calculate the ideal quantity for each floor after storage assignment, and assign the missing quantity to each floor. Remaining items, due to rounded results, are allocated to a random floor.

The second phase addresses the economic soft constraints SC_2 to SC_5 and aims to reduce travel distances during order picking. This phase assigns the incoming products to racks on a specific floor. Since this phase requires optimizing a set of constraints, we apply a multi-objective optimization algorithm that is described in Section 5.

The third phase aims to satisfy the ergonomic soft constraints SC_6 and SC_7. We classify a product p into three weight classes: *light* (up to 3 kg), *medium* (between 3 kg and 7 kg), and *heavy* (over 7 kg). We set the grip height to be between 0.75 m to 1.25 m and refer to compartments below/above the grip height as low/high zone compartments. Additionally, we distinguish *fast-moving*, *moderately-moving*, and *slow-moving* products by their relative rank. The relative rank of a product p calculates as $rank_p/|P|$, where $rank_p$ denotes the rank of product p, and $|P|$ the size of the product assortment. In the first step, the incoming items are assigned to the rack's compartments that already provide items of the same product. In the second step, the remaining incoming items are assigned to the rack's unoccupied compartments based on predefined penalty values. The penalty values range from zero to three and the more a compartment *comp* is unsuited for storing the product p, the more penalty points are given (see Tables 1 and 2).

Table 1: Penalties for assigning a product to a specific compartment with regards to the product weight.

Zone	Weight	Penalty
high	light	0
high	medium	2
high	heavy	3
grip height	light	1
grip height	medium	0
grip height	heavy	0
low	light	0
low	medium	1
low	heavy	1

Table 2: Penalties for assigning a product to a specific compartment with regards to the product rank.

Zone	Rank	Penalty
high	slow	0
high	moderate	0
high	fast	2
grip height	slow	3
grip height	moderate	1
grip height	fast	0
low	slow	0
low	moderate	0
low	fast	2

5 Genetic Algorithm for Storage Assignment

This section presents our custom version of the NSGA-II algorithm that was proposed by [6]. The algorithm receives the current state of a floor and assigns the incoming items to a set of racks on this floor. Note that the NSGA-II is executed for each floor individually.

5.1 Chromosome Encoding

We propose the chromosome encoding depicted in Figure 4. The figure illustrates an example allocation task where ten items of product p_1 must be assigned to the racks on $floor_1$. The black numbers indicate the existing items of product p_1,

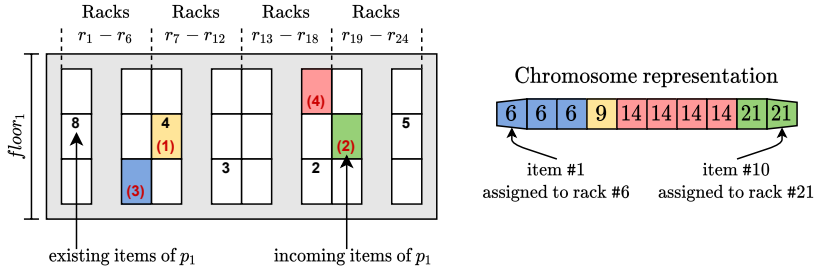


Fig. 4: A chromosome encodes the racks selected for storing the incoming items.

while the red numbers indicate the incoming items of product p_1. The right side shows the chromosome that encodes the storage allocation depicted on the left side by specifying the racks selected for storing each incoming item. Since ten items of product p_1 are assigned, the chromosome's length equals 10.

5.2 Objective Functions

A set of objective functions guide the NSGA-II algorithm to find good storage allocations. We propose four domain specific objective functions for our maximization problem: (i) spread score, (ii) distance score, (iii) quantity score, (iv) correlation score.

5.2.1 Spread Score

This score addresses constraint SC_1 and aims to equally spread the incoming quality of product p across the entire floor. Hence, we divide the floor f_j into multiple areas A of equal size. To calculate the spread score, we use the total ($totalQ$) and ideal quantity ($idealQ$) of a product in an area of a floor. The $totalQ$ is the sum of the existing and incoming items in an area, while the $idealQ$ is calculated by dividing the sum of the existing and incoming quantity of product p on the floor by the number of defined areas. The final spread score for chromosome C is calculated as the sum of differences between the total and the ideal quantity for all areas (see Equation 2).

$$spreadScore_{p,f_j,C} = (-1) \sum_{o=1}^{A} |idealQ_{p,f_j,j} - totalQ_{p,f_j,j}| \tag{2}$$

5.2.2 Distance Score

This score addresses constraint SC_2 and aims to allocate slow-moving products to racks further away from the p/d-points. Hence, the distance score quantifies the extent to which the walking distances ($dist_{r_i}$) of the selected racks match the ideal distance ($idealDist_{p,f_j}$). For calculating the $idealDist$, we perform the following steps: First, we determine the relative rank of the incoming product p

by dividing the rank of the product ($rank_p$) by the size of the product assortment P: $relRank_p = rank_p/|P|$. Then, the relative rank is mapped to a rack index: $rackIdx_{p,f_j} = relRank_p \cdot |R_{f_j}|$ with R_{f_j} being the list of racks of floor f_j sorted by the racks' walking distances to their closest p/d-point. The rack in R_{f_j} at index $rackIdx$ represents the best-suited rack for storing product p with regard to constraint SC_2. Finally, the $idealDist$ computes as: $idealDist_{p,f_j} = R_{f_j}[rackIdx_{p,f_j}] \cdot distance$. The overall distance score calculates as the sum over all racks in chromosome (C) of differences between the walking distances of the racks selected for storing product p and the $idealDist$ (see Equation 3).

$$distanceScore_{p,f_j,C} = (-1) \sum_{n \in C} |idealDist_{p,f_j} - dist_{r_i}| \tag{3}$$

Further, we provide an example of this calculation in Figure 5.

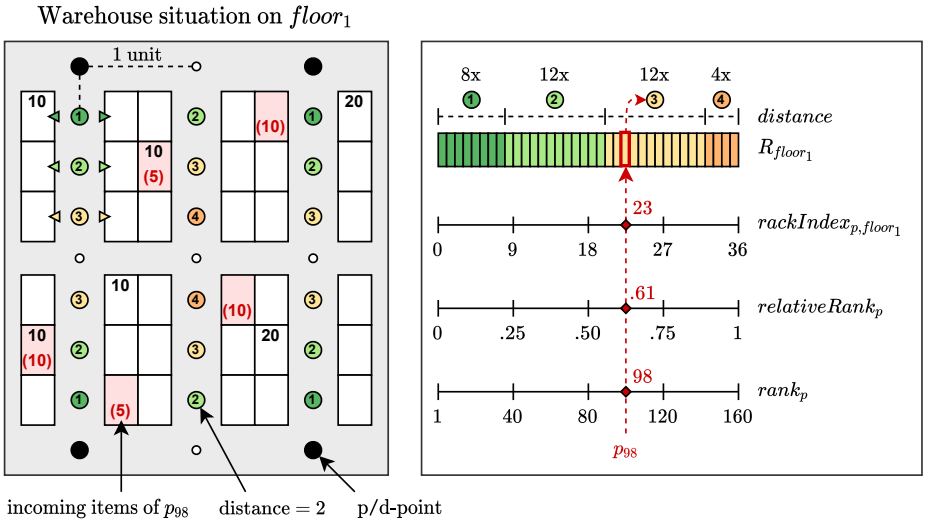


Fig. 5: Calculating the ideal distance for storing the incoming product p_{98}.

5.2.3 Quantity Score

This score assesses SC_3 and ensures that the mean ordered quantity of a product is locally available. Therefore, the target quantity defines the quantity to which the product p should be locally available based on a set of recent customer orders: $tq_p = \lceil \mu_p + 2\sigma_p \rceil$ with μ_p as expected value for the orders and σ_p for the standard deviation. Further, we define four masks and a modifier for each mask to measure the density to which the tq_p is locally available: M_1 equals the size of a rack ($maskMod = 1$), M_2 equals the size of two facing racks ($maskMod = 0.75$), M_3 is a sliding window with half the sub aisle's length ($maskMod = 0.5$), and M_4 covers an entire sub aisle ($maskMod = 0.25$). Using these masks, we calculate

a quantity factor for each sub aisle (sa) of a floor and each mask (M_k). Therefore, we select the quantity (q) of products inside a mask divided by the target quantity: $qFactor_{p,f_j,sa_l} = max(q_{p,f_j,sa_l}(M_k)/tq_p)$. This results in a value of 1 if the target quantity is met and a value of 0 if no products can be found within this mask. This quantity factor is then multiplied by the $maskMod$ to calculate the mask score: $maskScore_{p,f_j,sa_l}(M_k) = maskMod_{M_k} \cdot qFactor_{p,f_j,sa_l}(M_k)$. The highest possible mask score is 1, indicating that the target quantity is available in a single rack of the sub aisle. Based on these mask scores, the maximum value is selected to assign a score to each sub aisle: $subAisleScore_{p,f_j,sa_l} = max_{k=1}^{4} maskScore_{p,f_j,sa_l}(M_k)$. The final quantity score computes as the sum of all $subAisleScore$s (with $|SA|$ as the number of sub aisles):

$$quantityScore_{p,f_j,C} = \sum_{o=1}^{|SA|} subAisleScore_{p,f_j,sa_l} \tag{4}$$

Figure 6 illustrates the idea of using *masks* of different sizes to measure the density to which the target quantity tq_p of product p is locally available. The left side shows the storage locations of existing and incoming items of product p in a specific sub aisle sa. The center of the figure depicts the four masks M_k that iterate over the racks of the sub aisle. During this process, the masks count the existing and incoming quantities of product p that can be found in the covered regions. The right side shows the regions where the masks find the largest quantity of product p in the sub aisle sa.

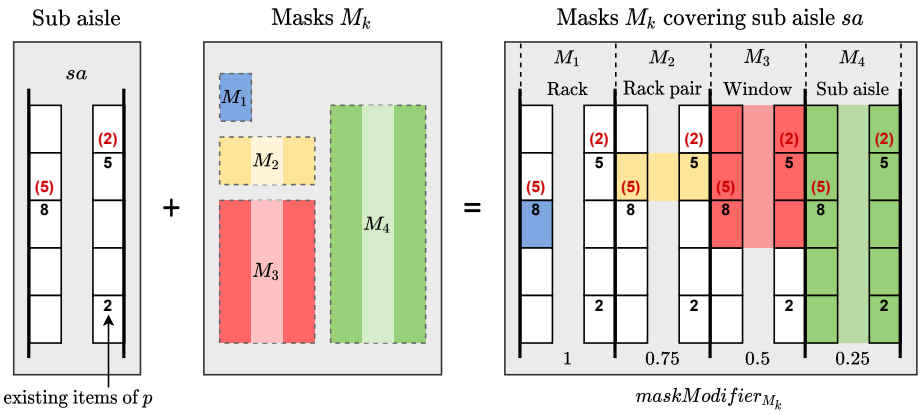


Fig. 6: The masks M_k count the quantities of product p in the covered regions.

5.2.4 Correlation Score

This score relates to SC_4 and describes the extent to which the incoming product is stored close to its correlated products. Association rules describe correlations between products and can be derived from recent customer orders. We consider

association rules of the form $rule = \{p\} \xrightarrow{conf} \{cp\}$, where p denotes the incoming product, cp the correlated product, and $conf$ a confidence value. We first calculate the number of possible clusters of target quantities of the incoming product: $qClusters_{p,f_j} = \lfloor totalQ_{p,f_j}/tq_p \rfloor$. We use this value to define the ideal quantity to which the correlated product should be available in the vicinity of the incoming product: $idealCorrQ_{rule,f_j} = \lceil qClusters_{p,f_j} \cdot tq_{cp} \cdot conf(rule) \rceil$. In the next step, we determine the quantity of cp that already is available in the vicinity of p. For this task, the previously introduced masks M_k are used and are placed directly on top of the racks containing cp. Again, the $qFactor$ is calculated to capture the extent to which the target quantity of p is available in the region covered by M_k placed on top of rack r: $qFactor_{p,r}(M_k) = q_{p,r}(M_k)/tq_p$. Then, we calculate the fraction to which the items of cp stored in r are considered to be in the vicinity of p: $corrQ_{rule,r}(M_k) = exQ_{cp,r} \cdot qFactor_{p,r}(M_k) \cdot maskMod_{M_k}$. $exQ_{cp,r}$ refers to the existing quantity of the correlated product cp in rack r. Afterward, we select the $corrQ$ with the highest value representing the mask with the largest amount of p in the vicinity of cp: $corrQ_{rule,r} = \max_{k=1}^{4} corrQ_{rule,r}(M_k)$. The sum of all $corrQ_{rule,r}$ over all racks on this floor denotes the quantity of the cp on this floor that is considered as being in the vicinity of p: $corrQ_{rule,f_j} = \sum_{rack \in R_{f_j,cp}} corrQ_{rule,r}$. Now, we calculated the quantity of the correlated product that is in the vicinity of the incoming product and the difference of this value to the ideal quantity. Based on this difference, the correlation score is calculated as (with A_p as the set of associations rules):

$$correlationScore_{p,f_j,C} = (-1) \sum_{rule \in A_p} idealCorrQ_{rule,f_j} - corrQ_{rule,f_j} \qquad (5)$$

5.3 Genetic Operators

The NSGA-II is a genetic algorithm and requires the definition of selection, crossover, and mutation operators.

5.3.1 Selection

We apply a binary tournament selection operator where two random parent individuals compete against each other [6]. The individual with the higher Pareto rank is declared the winner and is allowed to participate in the crossover procedure. In case both parents are of equal Pareto rank, the individual with the larger crowding distance, i.e. the higher diversity, wins the tournament.

5.3.2 Crossover

Since all chromosomes created during a run of the NSGA-II algorithm are of equal length, we use the traditional single-point crossover operator. It selects a random crossover point on both parents' chromosomes, splits them, and recombines them cross-wise to obtain two new children.

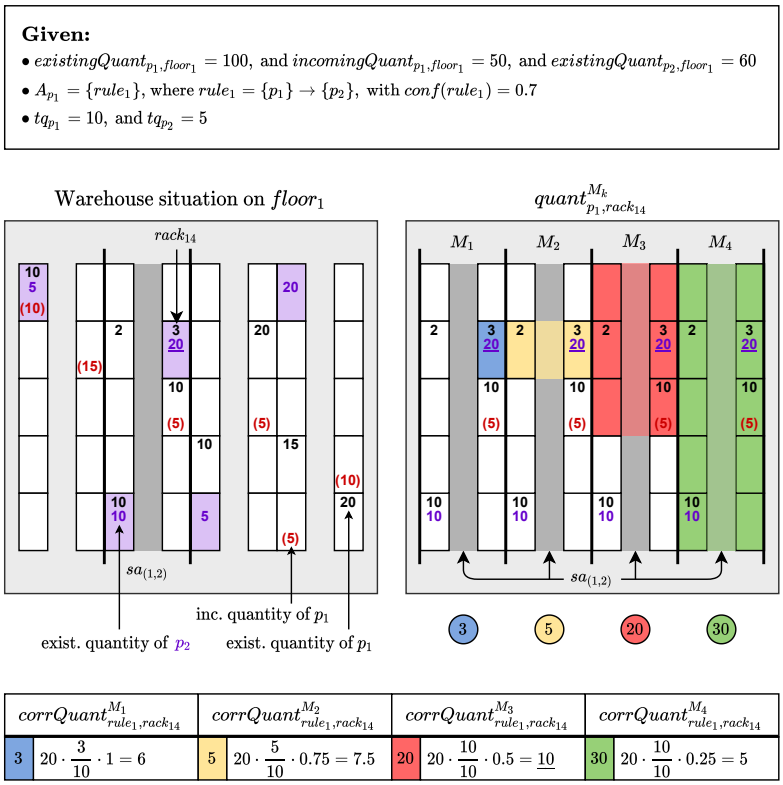


Fig. 7: Calculating the quantity of the correlated product p_2 that is available in the vicinity of the incoming product p_1.

5.3.3 Mutation

We define eight mutation operators that incorporate domain-specific knowledge to guide the search process: (1) The **FillRack** mutator selects a random rack and fills it with incoming items from the same sub aisle. (2) The **MoveRack** mutator selects a random rack containing at least one incoming item and moves them to a different rack within the same sub aisle. (3) The **FillSubAisle** mutator selects a random sub aisle and fills it with incoming items from other sub aisles until it provides the product's target quantity. (4) The **ClearSubAisle** mutator selects a random sub aisle and moves any incoming items to a different sub aisle. (5) The **RedistributeExceedingQuantities** mutator redistributes incoming items of racks that provide more items than the target quantity to racks that require only a few items to provide the target quantity. (6) The **ShiftRacks** mutator shifts all incoming items towards a randomly selected direction: left, right, up, or down. (7) The **SwapSubAisles** mutator first groups the sub aisles into pairs and swaps incoming items randomly within each pair. (8) The **SwapRacks** mutator is similar to (7) but swaps items within pairs of racks instead of sub aisles.

5.4 NSGA-II Algorithm

The overall procedure of our NSGA-II algorithm is summarized in Algorithm 1. The algorithm receives the `product` to be stored and its `quantity` as well as the

Algorithm 1: Proposed NSGA-II Algorithm.

Input: product, quantity, fittingRacks
Parameter: parentPopSize, mutProb, L, δ_{lim}, maxGen
Output: paretoFront

```
1  pop_parent = initParentPopulation(product, quantity, fittingRacks, parentPopSize)
2  gen = 0
3  historyOfMaxCD = new List()
4  while gen < maxGen && std(L) > δ_lim do
5      gen++
6      pop_children = createChildrenPopulation(pop_parent, furtherParametersOmitted)
7      pop_combined = pop_parent ∪ pop_children
8      pop_parent = createNextParentPopulation(pop_combined, parentPopSize)
9      paretoFront = calculateParetoFront(pop_parent)
10     maxCD = calculateMaxCD(paretoFront)
11     historyOfMaxCD.add(maxCD)
12 return calculateParetoFront(pop_parent)
```

list `fittingRacks`. Further, the `parentPopSize` defines the size of the parent population, the mutation probability is given by `mutProb`, the number of generations to be used when calculating the standard deviation of the maximum crowding distance $std(L)$ is called L, the threshold for the standard deviation of the crowding distance is δ_{lim}, and the maximum number of generations is called `maxGen`. In the end, the algorithm returns a `paretoFront` of the best storage assignments.

In the first step, the algorithm initializes the population by randomly creating the required amount of chromosomes. Therefore, the algorithm selects fitting racks for the product randomly which might produce invalid solutions due to exceeded rack spaces. Each invalid chromosome is then repaired by moving the amount of exceeding products to another available rack. Then, the generation counter `gen` and the history of observed maximum crowding distances are initialized. Then, the while loop starts and iterates using the two following stopping criterions: (i) the number of maximum generations (`maxGen`) is executed, or (ii) the standard deviation of observed crowding distances ($std(L)$) falls below the given threshold (δ_{lim}). Inside the while loop, the generations counter is incremented, and a complete new children population in the size of the parent population is bred using the proposed selection, crossover and mutation operators (`createChildrenPopulation`). This set is added to a combined population of existing parent individuals and select the best individuals to fill the new parent population (`createNextParentPopulation()`). Afterwards, a Pareto front is calculated from this parent population (`calculateParetoFront()`) and the maximum crowding distance of this front is calculated. This value is added to the history of maximum crowding distances. In case, the while loop stops, the current Pareto front is returned.

Since the NSGA-II algorithm returns a Pareto front, a user is usually required to identify the most valuable trade-off solution. However, we automate this step

by applying the following procedure. For each of the four objective functions (of_i), we select the solution (s_j) of the Pareto front with the highest value ($of_i(s_j)$) for this function. We then use these values as a 4-dimensional reference point ($p_{ref} = [e_1, e_2, e_3, e_4]$). Based on the Euclidean distance, the solution that is closest to the reference point is automatically selected as the most valuable trade-off solution.

6 Order Picking

This section introduces our order picking approach that is based on ACO. The overall goal of this algorithm is to construct a pick route for a given customer order. Since the travel distance is an essential economic goal, the pick route should be as short as possible. Additionally, the pick route should also be ergonomically favorable. The need for changing floors should be minimal to reduce the order picker's physical stress. Further, the product picking sequence is relevant as if light products are picked first, the order picker might need to rearrange the already picked products so that light products are placed on top of heavy products. Hence, the order picking algorithm aims to construct a short pick route that collects heavy products first and changes floors as little as possible to address economic and ergonomic criteria.

The main idea of this approach is to represent a mezzanine warehouse as a graph and let ants search for satisfactory order picking sequences. We make the following assumptions to better deal with the complexity of the order picking problem: (i) The state of the mezzanine warehouse does not change while the algorithm is running, that is no repositioning or removal of products is performed. (ii) The start and ending p/d points of a pick route may differ. (iii) Narrow sub aisles may only be traversed to the sub aisles' midpoint, as the picker always must go back to the cart in the wide pick aisle. (iv) The order pickers visit only one rack each time they enter a sub aisle. (v) Picking carts withstand infinite loads and can carry an unlimited amount of items.

6.1 Constraints

For the order picking algorithm, we define a set of hard and soft constraints. The hard constraints assess the feasibility of a solution, while the soft constraints measure the extent to which the solution fulfills economic and ergonomic goals. We define the following hard constraints: The pick route must start and end at a p/d point (HC_1). The pick route must collect the requested quantities of the products specified in the pick list (HC_2). After entering a narrow sub aisle, the route must always return to the sub aisle's entrance (HC_3). Further, we define one economic soft constraint: The travel distance should be minimal (SC_1); And two ergonomic soft constraints: The need for changing floors should be minimal (SC_2). Heavy products should be picked first, followed by lighter products (SC_3).

6.2 Graph Representation

We propose the following procedure for transferring a mezzanine warehouse into a graph representation. Figure 8 illustrates the procedure of dividing the warehouse into multiple zones called market zones.

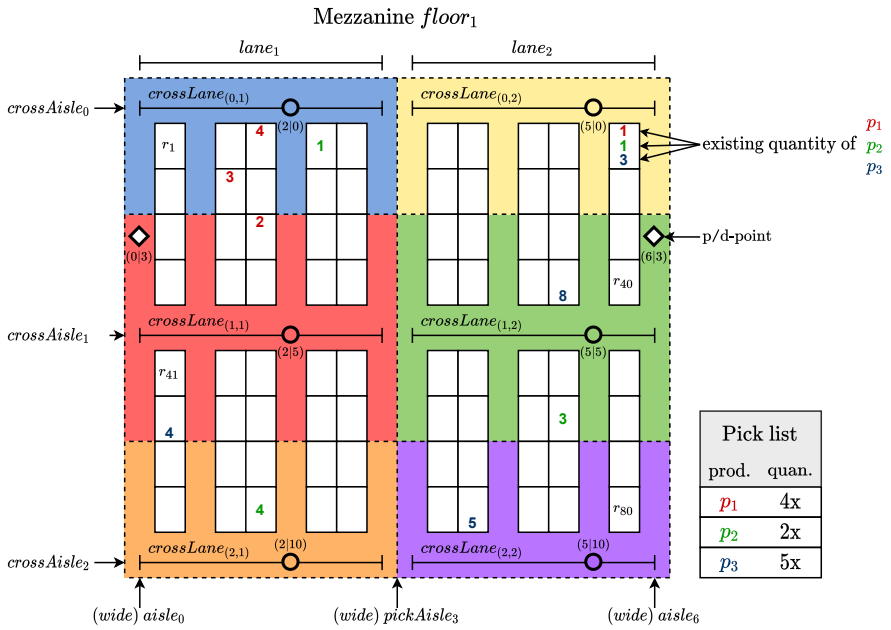


Fig. 8: The floor is divided into multiple market zones.

Each market zone is represented by a market, and thus, a node in the graph. The figure depicts the state of $floor_1$ that consists of three cross aisles, three wide (pick) aisles, and two p/d-points. We define six market zones obtained by dividing the floor along the wide (pick) aisles into multiple vertical lanes. In the depicted example, $lane_1$ refers to the area from $aisle_0$ to $pickAisle_3$, and $lane_2$ refers to the area from $pickAisle_3$ to $aisle_6$. A $crossLane_{(c,l)}$ refers to the part of the cross aisle c that lies within the lane l. For each cross lane, we define a market zone that comprises the storage racks that can be visited from the respective cross lane up to their midpoints. For example, the red market zone includes the racks that can be visited if the order picker is located at the $crossLane_{(1,1)}$. The market zones are limited to the midpoints of the corresponding sub aisles, which prevents the ants from constructing pick routes that entirely traverse the pick aisles. A market is referred to as $market_{(f,c,l)}$, where f denotes the floor, c the cross aisle, and l the lane. For each market, we define three attributes: (1) the market's *coordinates*, (2) the market's *closest p/d-point*, and (3) the market's *supply* that specifies which products are available at which quantity. After defining all markets, they are connected via edges to create a complete directed graph. The edges' weights represent the Manhattan distances between the markets. If the

warehouse consists of a second $floor_2$, the markets on $floor_1$ are also connected to the markets on $floor_2$ and vice versa, with an extra $floorPenalty$ added to the edges' weights.

6.3 Pick Route Construction

An ant colony explores the graph to construct a set of pick routes, i.e., a sequence of markets that provide the products, for a given pick list. A pick route consists of two layers: (i) representing markets, and (ii) rack sequences.

Figure 9 depicts an example pick route created by a single ant of the colony. The market sequence (layer one) of a pick route is computed by an ant that

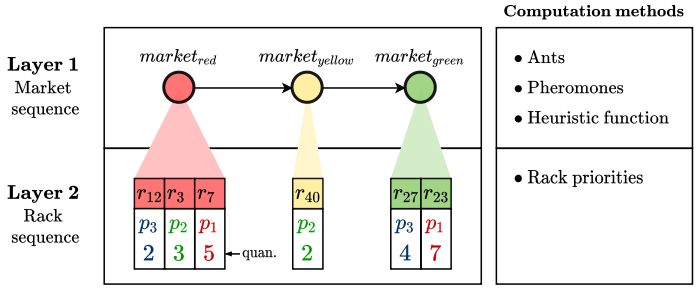


Fig. 9: A pick route consists of a market sequence and a rack sequence.

is placed on a market within the graph. Guided by the pheromone trails, the ant visits neighboring markets until it collected the requested product quantities specified in the pick list. The ant manages a purchasing list that specifies the missing items. The pick route is complete after the ant's purchasing list is empty. Further, each ant must decide whether it enters the market zone from the left or from the right side which depends on the position of the previously visited market. The left/right entrance is located at the position where the cross lane has its lowest/highest x coordinate value. The decision from which side the ant enters the market zone depends on the position of the previously visited market. While constructing pick routes, the ant applies a heuristic function to identify the markets within its vicinity that seem attractive to visit next.

The second layer represents the rack sequence, i.e., the racks the ant visited in each market. When calculating the rack sequence, the ants use the following priority rules: (1) Racks that provide heavy products should be visited first. (2) Racks located closer to the sub aisle's entrance should be visited second. (3) Racks that provide the largest quantities should be visited third.

6.4 Heuristic Function

To identify the most promising paths and assess the attractiveness of a market, the ants apply a heuristic function. The attractiveness of a market is based on two

factors: (i) the closeness of the market to the ant's current location, and (ii) the availability of required items. Thus, we define the heuristic function as follows:

$$\eta_{m,n}^{k} = \left(\frac{1}{d_{m,n}}\right)\left(I_{n}^{k}\right) \text{ , where } n \in U^{k} \tag{6}$$

where $\eta_{m,n}^{k}$ is the heuristic value that the ant k currently located at market m associates with the edge (m, n) leading to market n. U^{k} is the set of markets the ant has not visited yet and $d_{m,n} > 0$ refers to the Manhattan distance between the markets. $I_{n}^{k} \in [0; 1]$ denotes the percentage to which the required items of ant k are available at market n. The higher the heuristic value, the more attractive is the market for the ant.

6.5 Objective Functions

After retrieving possible pick routes from the algorithm, we use two objective functions to asses the quality of the route.

6.5.1 Travel Distance

This objective function calculates the travel distance of a pick route and measures the extent to which the soft constraint SC_1 and SC_2 are satisfied. We define a pick route P to be $P = (M, R)$ where $M = (m_1, ..., m_k)$ refers to the market sequence and $R = (r_1, ..., r_l)$ refers to the rack sequence. We then define the objective function as follows:

$$travelDistance(P) = d_{m_1}^{pd} + \sum_{i=1}^{l} d_{r_i}^{sub} + \sum_{i=1}^{k} d_{m_i}^{cross} + \sum_{i=1}^{k-1} d_{(m_i,m_{i+1})}^{market} + d_{m_k}^{pd} \tag{7}$$

where we sum up the distance from the start p/d-point to the first market, the sum of the distances within each entered sub aisle ($d_{r_i}^{sub}$), the sum of the distances within the cross lanes ($d_{m_i}^{cross}$), the distances between the visited markets ($d_{(m_i,m_{i+1})}^{market}$), and the distance from the last visited market to its closest p/d-point ($d_{m_k}^{pd}$).

6.5.2 Weight Violation

The second objective function measures the extent to which a pick route satisfies the soft constraint SC_3 and counts the number of weight violations in the product picking sequence. A weight violation occurs if a heavy product is collected after a much lighter product. In this case, the order picker must rearrange the lighter products already placed on the picking cart to prevent damage. The user-specified threshold `allowedWeightDifference` defines the acceptable weight difference between the heavier and the lighter products. Using this threshold, we count the number of weight violations in a product picking sequence.

6.6 ACO Algorithm Procedure

This section proposes our proposed ACO algorithm and shows the pseudo-code in Algorithm 2. First of all, the algorithm constructs the graph and initializes the pheromones. The pheromones are initialized with their maximum possible value determined by τ_{max}. Additionally, a minimum pheromone can be specified by using the value τ_{min} in the parametrization of the algorithm. Then, a while loop starts and uses the concept of cataclysms [4] and a maximum number of iterations as stopping criterion: The parameter `maxCataclysms` specifies the maximum number of cataclysms that may occur. The parameter `maxconsIterWoImpr` defines the time window in which the ACO algorithm must improve the current Pareto front to prevent the cataclysm operator from being applied. The parameter `maxIter` defines the maximum allowed number of iterations regardless of happened cataclysms.

Inside the loop the number of current iterations is incremented and pick routes are constructed. The general idea is to place one ant on each market of the graph from which the ant starts to create a pick route. The next market is selected based on the pheromone values and the heuristic function. We propose two different versions of the ACO to combine these values as explained later. For each found pick route, the reverse pick route is calculated by reversing the market sequence, toggling the sides from which the ant entered the markets, and recalculating the rack sequence. We store the pick routes the ants construct in each iteration in the variable `pickRoutes`. In the next step, the Pareto-optimal pick routes of this iteration are selected by calculating the objective function and the Pareto rank of all routes. Afterward, the iteration-best ($\texttt{pickRoutes}_{\texttt{ib}}$) and the global-best pick routes ($\texttt{nextPickRoutes}_{\texttt{gb}}$) are merged into a single set and the Pareto-optimal pick routes in this set represent the next set of global-best pick routes. The iteration-best pick routes and the global-best pick routes are used to perform the pheromone update, which is explained later. In the further course of the iteration, the ACO algorithm checks whether the cataclysm operator must be applied and compares the global best pick routes of the last and the current iteration. If the ACO algorithm succeeded in improving the Pareto front, the set $\texttt{pickRoutes}_{\texttt{gb}}$ is updated, and the counter variable `consIterWoImpr` is reset to 0. However, if no improvement was made, this counter variable is incremented. If multiple consecutive iterations fail to achieve an improvement, the search is considered stuck, and the cataclysm operator is applied. In case the cataclysm is applied, the global-best pick routes $\texttt{pickRoutes}_{\texttt{gb}}$ are included in the set $\texttt{pickRoutes}_{\texttt{cataclysm}}$, the pheromones on the edges representing the pick routes in $\texttt{pickRoutes}_{\texttt{gb}}$ are reset to the lowest possible value, and the set $\texttt{pickRoutes}_{\texttt{gb}}$ is emptied. Then, the number of cataclysms is incremented and the counter variable `consIterWoImpr` is reset to 0. After the main loop terminates, the global-best pick routes $\texttt{pickRoutes}_{\texttt{gb}}$ of the last iteration are included in the set $\texttt{pickRoutes}_{\texttt{cataclysm}}$ and the algorithm returns the Pareto-optimal pick routes in this set.

6.7 ACO_3 Variant

In the following, we introduce two variants of our algorithm that show a distinct pheromone handling. Both variants are inspired by [1] that propose four different variants to handle multi-objective problems with an ACO. We select the two

Algorithm 2: Proposed ACO Algorithm.

```
Input: warehouseState, pickList
Parameter: maxIterWoImpr, maxCataclysms, maxIter
Output: pickRoutes
graph = constructGraph()
pheromones = initializePheromones()
while cataclysms < maxCataclysms || iter < maxIter do
    iter++
    pickRoutes = constructPickRoutes()
    pickRoutes_ib = selectParetoPickRoutes(pickRoutes)
    pickRoutes_merged = pickRoutes_ib ∪ pickRoutes_gb
    nextPickRoutes_gb = selectParetoPickRoutes(pickRoutes_merged)
    updatePheromones()
    if isParetoFrontImproved() then
        pickRoutes_gb = nextPickRoutes_gb
        consIterWoImpr = 0
    else
        consIterWoImpr++
        if consIterWoImpr >= maxIterWoImpr then
            pickRoutes_cataclysm = pickRoutes_cataclysm ∪ pickRoutes_gb
            resetPheromonesOnGlobalBestRoutes()
            cataclysms++
            consIterWoImpr = 0
pickRoutes_cataclysm = pickRoutes_cataclysm ∪ pickRoutes_gb
return selectParetoOptimalPickRoutes(pickRoutes_cataclysm)
```

best performing variants (ACO_3 and ACO_4) and integrate them in our approach to compare which variant produces the best results in our problem domain. This section introduces the ACO_3 variant that applies one ant colony using a single pheromone matrix τ^1 for optimizing both objectives simultaneously. In each construction step, the probability of selecting an edge calculates as:

$$prob^k_{m,n} = \frac{(\tau^1_{m,n})^\alpha (\eta^k_{m,n})^\beta}{\sum_{u \in U^k} (\tau^1_{m,n})^\alpha (\eta^k_{m,n})^\beta}, \text{ where } n \in U^k \tag{8}$$

where $prob^k_{m,n}$ denotes the probability of ant k located at market m to select the edge (m, n) leading to market n. $\tau^1_{m,n}$ refers to the pheromone value of edge (m, n). $\eta^k_{m,n}$ denotes the heuristic value (see Formula 6) that the ant associates with the edge (m, n). The parameters α and β control the importance of the pheromone values and heuristic values. Lastly, U^k represents the set of markets that ant k has not visited yet.

When performing the pheromone update, the ACO_3 variant rewards in 90% of the time the iteration-best pick routes and in 10% of the time, the global-best pick routes (found since the last cataclysm) to update the pheromone matrix τ^1. The pheromone values are updated according to the following rule [1]:

$$\tau^1_{m,n} = (1 - \rho) \cdot \tau^1_{m,n} + \Delta\tau^1_{m,n} \tag{9}$$

$$\Delta\tau^1_{m,n} = \begin{cases} 1, & \text{if } (m, n) \text{ belongs to a pick route in } PF \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

where ρ refers to the evaporation factor and $\Delta\tau^1_{m,n}$ is the amount of pheromone that is added to the edge (m,n). PF refers to the Pareto front containing the solutions to be rewarded.

6.8 ACO_4 Variant

The ACO_4 variant also applies one ant colony but a pheromone matrix τ^1 for optimizing the first objective function, and another pheromone matrix τ^2 for optimizing the second objective function. When deciding which edge to explore next, an ant randomly chooses a pheromone matrix. In each construction step, the probability of selecting an edge calculates as [1]:

$$p^k_{m,n} = \frac{(\tau^i_{m,n})^\alpha (\eta^k_{m,n})^\beta}{\sum\limits_{u \in U^k} (\tau^i_{m,n})^\alpha (\eta^k_{m,n})^\beta}, \text{ where } n \in U^k \text{ and } i \in \{1,2\} \tag{11}$$

where $\tau^r_{m,n}$ refers to the pheromone value of edge (m,n) w.r.t. pheromone matrix τ^i. At the end of an iteration, the ACO_4 variant updates the pheromone matrix τ^i by rewarding the iteration-best pick route PR^i_{ib} that minimizes the objective function of_i [1]:

$$\tau^i_{m,n} = (1-\rho) \cdot \tau^i_{m,n} + \Delta\tau^i_{m,n} \tag{12}$$

$$\Delta\tau^i_{m,n} = \begin{cases} \frac{1}{1+of_i(PR^i_{ib})-of_i(PR^i_{gb})}, & \text{if } (m,n) \text{ belongs to the pick route } PR^i_{ib} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where ρ again refers to the evaporation factor and $\Delta\tau^i_{m,n}$ is the pheromone added to the edge (m,n) in pheromone matrix τ^i. PR^i_{gb} refers to the global-best pick route that minimizes the ith objective function of all pick routes constructed since the last cataclysm occurred.

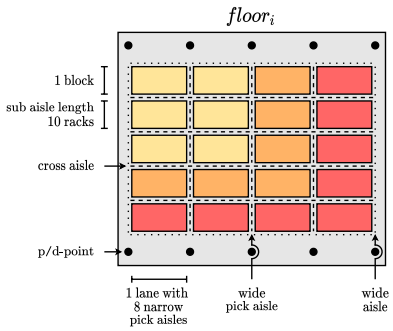
7 Evaluation

This section presents the evaluation of our approaches. It defines the used warehouse models for applying our algorithms, presents performance indicators, summarizes alternative policies to which we compare our algorithms, and provides the parameter settings of our algorithms. Afterwards, we first evaluate our storage assignment and order picking algorithms individually before we evaluate the interaction of both algorithms.

7.1 Mezzanine Warehouse Models

The NSGA-II and the ACO algorithm are evaluated in three artificial mezzanine warehouses of different sizes that are defined in cooperation with our cooperation company to build real-world test cases. The warehouses are shown in Figure 10: WH_{small} (yellow), WH_{medium} (orange), and WH_{large} (red). For the small, medium, and large warehouses, we define the size of the product assortment to be 500, 1000, and 1500, respectively. Since each product requires a weight, we

define three normal distributions and a probability to determine the weight using this distribution: 25% to use $\mathcal{N}(2, 1.0^2)$, 50% to use $\mathcal{N}(5, 2.0^2)$, and 25% to use $\mathcal{N}(8, 1.0^2)$. Using these distributions and probabilities, we aim at a representative set of product weights where most of the products have a medium weight and some products have low and some have heavy weights. The products might also have correlations to up to three other products: With a probability of 30%, 40%, 20%, and 10% a product has no, one, two, or three correlated products, respectively, with a randomly generated correlation confidence between 10% and 90%. For evaluating the order picking algorithm, we fill the storage up to 50% of the available storage space and randomly generate 100 customer orders based on the product assortment and given correlations between products. Each customer order comprises 20 items to pick that are selected as follows: We split the product assortment into four equally sized groups based on the product rank. With a probability of 40%, 30%, 20%, and 10% an order contains an item of the highest, second highest, third highest, and lowest rank class, respectively, which ensures that high-ranked products appear more often in customer orders.



WH_{size}	#Floors	#Racks		#Markets	
		per floor	total	per floor	total
WH_{small}	2	1020	2040	8	16
WH_{medium}	2	2080	4160	15	30
WH_{large}	2	3500	7000	24	48

Fig. 10: The warehouses WH_{small}, WH_{medium}, and WH_{large} use different floor layouts.

7.2 Performance Indicators for Assessing Pareto Fronts

Since we assess a multi-objective optimization problem, the algorithms compute a Pareto front. We use the following quality indicators for Pareto fronts introduced by [31]. The Coverage (Cov) quality indicator quantifies the extent to which a computed Pareto front covers the reference Pareto front. The quality indicators Generational Distance (GD) and Euclidean Distance (ED) measure the distance from a computed Pareto front to a reference Pareto front or a reference solution. The quality indicators Pareto Front Size (PFS) and Generated Spread (GS) measure the diversity of the solutions that exist in a computed Pareto front. Finally, the quality indicator Inverted Generational Distance (IGD) combines convergence and diversity aspects. Since most of these quality indicators require the calculation of a reference Pareto front, we use the Pareto fronts returned by all algorithms

as basis. From these Pareto fronts, we select the non-dominated solutions of the union of all computed Pareto fronts and use this front as reference Pareto front.

7.3 Alternative Policies

We use the following alternative policies for the storage assignment problem. The **random storage assignment policy** allocates the incoming items to random racks on the floors in clusters of target quantity size [2]. In the **closest open location storage assignment policy**, the warehouse employees select the storage locations for storing an incoming product, which are usually the racks closest to the p/d-points [13]. The **rank-based storage assignment policy** assigns fast-moving products close to the p/d-points, while slow-moving products are assigned to racks further away [24].

For the order picking problem, we apply a modified S-Shape heuristic for comparison that constructs s-shaped pick routes based on the graph representation [22]. This heuristic uses all markets as starting point iteratively as well as the reversed versions of each route to generate a Pareto front of possible solutions.

7.4 Algorithm Parameter Settings

Based on a preliminary parameter study, we parameterize our NSGA-II algorithm as follows: We set the mutation probability to 0.95 for all warehouse sizes so that the mutation operators are applied very frequently. Additionally, we set the crossover probability to 1.00 so that the crossover operator is applied for all generations. Further, we define parameters dependent on the warehouse size (small/medium/large): The parent population size is set to (50/60/70), and the maximum number of generations to (200/250/300). These values increase with the size of the warehouse since the number of possible solutions increases with the warehouse size and we provide the algorithm more exploration possibilities (population size) and more time (number of generations) for optimizing the solutions.

Further, we set the parameters for our ACO algorithm as follows: In line with the literature, we set the pheromone factor α to 1.0 and the heuristic factor β to 2.0. We set the evaporation factor ρ to 0.02 causing the pheromones to evaporate rather slowly which enables the algorithm to achieve a higher degree of exploration especially in the early stages. The min/max values for the pheromone matrices ($\tau_{min/max}$) are set to 1 and 25, respectively, add a floor change penalty of 50, and set the allowed weight difference to 3 kg. As stopping criterion, we set the maximum number of cataclysms to 3 and hence, the algorithm terminates after it became stuck for the third time. Using the results of a preliminary parameter study, we set the maximum consecutive iterations without improvements to 20, and the maximum iterations to 250 since these values yield the best results w.r.t the scores.

7.5 Evaluation of the NSGA-II Algorithm for Storage Assignment Tasks

We evaluate our NSGA-II algorithm against the random, closest open location, and rank-based storage assignment policies. We apply all approaches on the three

Table 3: Mean values of the six quality indicators achieved by the storage assignment algorithms in Setting 1.a, 1.b, and 1.c (best values are shown in bold).

Setting	Policy	Cov [μ]	GD [μ]	ED [μ]	PFS [μ]	GS [μ]	IGD [μ]
1.a	Random	0.01	1.59	25.33	24.80	0.73	2.26
	Closest	0.01	2.14	28.37	21.98	0.74	2.53
	Rank	0.09	0.95	21.88	28.20	0.78	2.53
	NSGA-2	**0.90**	**0.04**	**16.48**	**47.52**	**0.50**	**0.26**
1.b	Random	0.01	2.38	26.20	16.80	0.75	1.92
	Closest	0.00	3.32	31.28	14.82	**0.73**	2.18
	Rank	0.06	1.68	22.03	19.44	0.83	2.13
	NSGA-2	**0.93**	**0.02**	**14.01**	**52.84**	1.04	**0.11**
1.c	Random	0.00	2.78	29.47	9.92	**0.81**	1.86
	Closest	0.00	3.11	28.58	11.08	0.86	2.10
	Rank	0.01	1.40	25.23	14.24	0.94	1.92
	NSGA-2	**0.99**	**0.00**	**16.76**	**64.34**	1.37	**0.02**

warehouse sizes (Setting 1.a, 1.b, 1.c) and on five randomly generated storage assignment tasks, i.e., we select a random product from the product assortment and set the quantity to be assigned to the quantity already existing in the warehouse. We repeat the execution of the NSGA-II algorithm ten times to reduce random effects and present mean and standard deviation values. All generated solutions of all algorithms are then used to calculate the reference Pareto front required for the quality indicators. Table 3 summarizes the mean values and Table 4 shows the standard deviation values for this evaluation. The coverage (Cov) results for the small warehouse show that the NSGA-II Pareto front covers about 90% of the reference Pareto front while the other approaches cover only around 9% and 1% Since, the NSGA-II shows the lowest GD and ED values, this Pareto front is located closest to the reference front. The NSGA-II algorithm finds around 48 solutions per problem instance with a maximum possible value of 50 solutions for the small warehouse size. The other policies only construct 22 to 28 Pareto-optimal solutions while their maximum possible value is set to 500. Further, the NSGA-II achieves the lowest GS and IGD values which indicates, that the solutions converge well towards the reference Pareto front and offer diverse solutions.

In the medium warehouse, the results show similar behavior. The table shows that the Pareto front $PF_{nsga_2(c_m)}$ covers approximately 93% of the reference Pareto front PF_{ref}. Except for some outliers, the alternative policies struggle to cover the solutions in PF_{ref}. The observed GD and ED values are fairly similar to the values in Setting 1.a. However, the standard deviations of the ED metric increased noticeably, which may be related to the larger search space where the solutions tend to be more spread out. Nevertheless, the Pareto front $PF_{nsga_2(c_m)}$ still achieves the lowest GD and ED values, indicating that this Pareto front converges best towards PF_{ref}. Concerning the PFS metric, the NSGA-II algorithm finds about 53 solutions per problem instance, while the alternative policies find approximately less than 20 solutions per problem instance. The GS values of $PF_{nsga_2(c_m)}$ increased remarkably, which may be due to the larger parent population size and the larger search space that make it difficult for the NSGA-II algorithm to fill the gaps in the Pareto front so that all solutions are evenly distributed. Lastly, the

Table 4: Standard deviations of the six quality indicators achieved by the storage assignment algorithms in Setting 1.a, 1.b, and 1.c.

Setting	Policy	Cov [σ]	GD [σ]	ED [σ]	PFS [σ]	GS [σ]	IGD [σ]
1.a	Random	0.01	0.78	10.20	11.52	0.16	1.91
	Closest	0.01	1.26	12.29	9.49	0.19	1.71
	Rank	0.09	0.44	10.66	14.98	0.21	1.99
	NSGA-2	0.10	0.08	7.93	5.35	0.17	0.26
1.b	Random	0.02	1.21	15.20	13.29	0.13	0.96
	Closest	0.01	1.74	18.95	9.84	0.08	0.95
	Rank	0.12	1.14	13.53	19.53	0.18	1.13
	NSGA-2	0.14	0.05	9.38	9.46	0.54	0.20
1.c	Random	0.00	2.34	26.32	4.89	0.13	1.81
	Closest	0.00	3.55	26.45	7.49	0.16	2.17
	Rank	0.01	1.39	23.91	6.79	0.17	2.12
	NSGA-2	0.01	0.00	14.06	9.60	0.45	0.06

IGD values of $PF_{nsga_2(c_m)}$ are close to 0, indicating that $PF_{nsga_2(c_m)}$ represents the entire reference Pareto front PF_{ref} in most cases.

Similarly, the large warehouse shows comparable results. The Pareto front $PF_{nsga_2(c_l)}$ covers about 99% of the reference Pareto front PF_{ref}, while PF_{rank} covers only 1%. Thus, almost all solutions found by the rank-based policy are dominated by the solutions found by the NSGA-II algorithm. The mean GD value of $PF_{nsga_2(c_l)}$ equals 0, indicating that the entire Pareto front $PF_{nsga_2(c_l)}$ is part of PF_{ref} in almost all cases. Other than that, the observations made in the previous Setting 1.b also occur in this setting. Thus, the standard deviations of the ED metric further increase, the NSGA-II algorithm finds the most solutions per problem instance, the alternative policies find fewer solutions per problem instances, the GS values of $PF_{nsga_2(c_l)}$ further increase, and the IGD values $PF_{nsga_2(c_m)}$ are closer to 0.

In summary, the results show that the random and the closest open location policy struggle to cover even a single solution in the reference Pareto front. Additionally, the NSGA-II algorithm outperforms the alternative policies in smaller warehouses. Further, the NSGA-II finds on average the most solutions per problem instance and the solutions are less equally distributed.

In addition to the quality evaluation, we also measure the mean execution time of the approaches for solving 50 problem instances in each warehouse size[1]. The alternative policies achieve low execution times of about 0.17/0.30/0.50 seconds for small/medium/large which is due to their comparably simple operation. In the warehouse small/medium/large, the NSGA-II algorithm achieves execution times of about 2/6/15 seconds, which is due to the increase population and iteration count for larger warehouses. The execution times of the NSGA-II algorithm may be considered acceptable, as the algorithm requires only a few seconds to find storage allocations that are remarkably better than the ones found by the alternative policies.

[1] We run our experiments on a MacBook Pro using macOS Sierra 10.12.6, a 2.2GHz Intel Core i7 CPU and 16GB DDR3 RAM.

Table 5: Mean values of the six quality indicators achieved by the order picking algorithms in Setting 2.a, 2.b, and 2.c (best values are shown in bold).

Setting	Policy	Cov [μ]	GD [μ]	ED [μ]	PFS [μ]	GS [μ]	IGD [μ]
2.a	sShape	0.00	22.15	80.34	2.80	**0.84**	18.28
	ACO_3	0.73	**1.40**	32.03	**10.66**	0.99	2.74
	ACO_4	**0.74**	1.59	**32.07**	9.54	0.87	**1.91**
2.b	sShape	0.00	31.20	117.42	3.60	0.72	18.78
	ACO_3	**0.69**	**1.97**	**50.11**	**12.14**	0.81	**3.00**
	ACO_4	0.33	4.30	54.59	11.30	**0.66**	4.15
2.c	sShape	0.00	41.41	121.35	2.00	0.88	27.18
	ACO_3	**0.84**	**1.56**	**56.75**	**10.50**	0.74	**5.64**
	ACO_4	0.16	9.78	70.02	10.14	**0.68**	7.28

Table 6: Standard deviations of the six quality indicators achieved by the order picking algorithms in Setting 2.a, 2.b, and 2.c.

Setting	Policy	Cov [σ]	GD [σ]	ED [σ]	PFS [σ]	GS [σ]	IGD [σ]
2.a	sShape	0.00	14.20	28.26	1.60	0.16	6.55
	ACO_3	0.13	1.91	16.21	5.63	0.36	2.96
	ACO_4	0.18	2.64	16.14	3.97	0.35	2.52
2.b	sShape	0.00	6.92	34.51	1.02	0.12	5.03
	ACO_3	0.15	2.08	15.74	3.80	0.20	2.67
	ACO_4	0.16	3.62	14.43	3.23	0.17	2.68
2.c	sShape	0.00	16.42	35.71	0.89	0.14	6.95
	ACO_3	0.11	2.90	18.95	2.87	0.23	6.27
	ACO_4	0.11	5.83	21.76	3.28	0.21	3.91

7.6 Evaluation of the ACO Algorithm for Order Picking Tasks

We evaluate both versions of our ACO algorithm against the modified S-Shape heuristic. We apply all approaches on the three warehouse sizes (Settings 2.a, 2.b, 2.c) using five customer orders randomly selected from the set of generated customer orders as explained earlier and repeat the execution of the ACO algorithms ten times to reduce random effects and present mean and standard deviation values. Then, we use all generated solutions the algorithms to calculate the reference Pareto front required for the quality indicators. Table 5 summarizes the mean values and Table 6 shows the standard deviation values for this evaluation. For the small warehouse, the Pareto fronts of the ACO_3 and ACO_4 variants cover 74% of the reference Pareto front while the S-Shape heuristic fails to cover even a single solution. Both ACO algorithms achieve nearly the same GD and ED values and the close to zero GD values show that many solutions are part of the reference front. The ACO algorithms find around ten solutions per problem instance, while the S-Shape only finds three solutions per problem instance. The S-Shape achieves the lowest, hence, the best GS values, but it is not meaningful to compare these values to the ACO ones as it contains only three solutions that are considerably worse than solutions of the ACO algorithms. The ACO algorithms achieve low IGD values, indicating that both Pareto fronts converge well towards the reference front and provide diverse solutions.

The metrics of the medium warehouse show similar behavior as in the previous setting. Like in the previous setting, the Pareto front PF_{sShape} fails to cover even a single solution in the reference Pareto front PF_{ref}. The Pareto front PF_{aco_3} covers approximately 69% of PF_{ref}, while PF_{aco_4} covers only 33%. Thus, the ACO_3 variant tends to find better pick routes than the ACO_4 variant. The Pareto front PF_{aco_3} achieves the lowest GD and ED values among all computed Pareto fronts. Thus, PF_{aco_3} converges best towards PF_{ref}, which is not surprising, as PF_{aco_3} covers most of the solutions in PF_{ref}. The GD and ED values of PF_{aco_4} are slightly larger than the ones of PF_{aco_3}, indicating that PF_{aco_4} does not converge as well as PF_{aco_3} towards PF_{ref}. Compared to the previous setting, the GD and ED values of PF_{sShape} increased, which may be due to the larger search space. Concerning the PFS metric, the ACO_3 variant finds about 12 solutions per problem instance, followed closely by the ACO_4 variant that finds around 11 solutions per problem instance, while the S-Shape heuristic only finds about 4 solutions per problem instance. Regarding the GS metric, the solutions in PF_{aco_4} are slightly better distributed than the solutions in PF_{aco_3}. With respect to the IGD metric, PF_{aco_3} achieves the lowest IGD values, indicating that PF_{aco_3} converges well towards PF_{ref} and offers a high diversity of solutions.

In the large warehouse, the S-Shape heuristic is again unable to cover a solution in PF_{ref}. The Pareto front PF_{aco_3} covers 84% of the reference Pareto front PF_{ref}, while PF_{aco_4} covers only 16%. Thus, most of the solutions found by the ACO_4 variant are dominated by the solutions found by the ACO_3 variant. Accordingly, the ACO_4 variant has problems to compete with the ACO_3 variant in larger warehouses. Compared to the previous setting, the GD and ED values of PF_{aco_4} further increased, indicating that the distances between the solutions in PF_{aco_4} and the solutions in PF_{ref} became larger. The Pareto front PF_{aco_3} converges best towards PF_{ref}, as it achieves the lowest GD and ED values. Regarding the PFS metric, both ACO variants find about 10 solutions per problem instance. The GS metric indicates that the solutions in PF_{aco_4} are marginally better distributed than the solutions in PF_{aco_3}. Finally, the Pareto front PF_{aco_3} achieves the lowest, and thus, best IGD values, signalizing that PF_{aco_3} converges best towards PF_{ref} and offers diverse solutions.

In summary, the ACO algorithms outperform the S-Shape heuristic in all warehouse sizes, and ACO_3 and ACO_4 show similar performance in smaller warehouses. With increasing warehouse size, the solutions found by the ACO_3 variant dominate more and more solutions of the ACO_4 variant. Hence, the ACO_3 variant starts to find better pick routes than the ACO_4 variant, while the ACO_4 variant produces slightly better distributed solutions.

In addition to the quality evaluation, we also measure the mean execution time of the approaches for solving 50 problem instances in each warehouse size. The S-Shape heuristics takes around 0.15 seconds to compute routes. The ACO_3 and the ACO_4 variant achieve fairly the same execution times in all warehouse sizes of around 1/3/6 seconds for $WH_{small}/WH_{medium}/WH_{large}$. As the warehouse size increases, the graph consists of more markets causing more ants to be deployed in each iteration. Still, we consider the ACO execution times acceptable, as they require only a few seconds to find noticeably better pick routes.

7.7 Evaluation of Computational Costs

Besides evaluating the performance of the proposed algorithms in terms of solution quality, we also measured execution times and evaluated the computational costs. The experiments were conducted on a MacBook Pro with a 2.2GHz Intel Core i7 processor, 16GB of RAM, and macOS Sierra 10.12.6. Each time measurement was repeated 50 times. Table 7 and Table 8 report the mean and standard deviation for the initialization and execution phases in seconds.

Table 7: Mean initialization times of the storage assignment algorithms in seconds.

	WH_{small}		WH_{medium}		WH_{large}	
Algorithm	μ	σ	μ	σ	μ	σ
Order Picking	63.86	1.40	122.95	2.48	195.95	4.25
Storage Assignment	5.15	1.42	6.32	1.50	8.15	1.05

Table 7 shows the initialization times for the storage assignment and order picking algorithms. The initialization time increases with the warehouse size as more data must be requested from the database. However, the measured initialization times must be analyzed with caution, as the initialization time depends more on the size of the database tables than on the warehouse's size. Initialization times for the order picking algorithms are much higher due to the graph construction, which makes the initialization time proportional to the number of markets. This also indicates that tuning the SQL queries would only partly enhance the initialization time of the order picking but could have a more significant impact on the order picking. The graph construction could be parallelized further to increase the initialization time for order picking, as no data dependencies exist between the markets.

Table 8: Mean execution times of the order picking algorithms in seconds.

	WH_{small}		WH_{medium}		WH_{large}	
Algorithm	μ	σ	μ	σ	μ	σ
Storage Assignment: *sShape*	0.14	0.02	0.15	0.03	0.15	0.03
Storage Assignment: aco_3	1.11	0.34	2.75	0.33	5.98	0.82
Storage Assignment: aco_4	1.08	0.14	2.86	0.32	6.23	0.77
Order Picking: *random*	0.18	0.10	0.29	0.13	0.47	0.22
Order Picking: *closest*	0.17	0.07	0.33	0.13	0.55	0.25
Order Picking: *rank*	0.16	0.06	0.30	0.09	0.47	0.13
Order Picking: $nsga_2(c_s/c_m/c_l)$	2.23	0.40	6.22	0.75	15.42	1.10

Table 8 shows the execution times for the proposed algorithms as well as the execution time of the baselines. For storage assignment, alternative policies achieve low execution times of up to 0.5 seconds, which is not surprising, as they require only a few calculation steps to assign the incoming items to racks on the floor.

Table 9: Mean values of the six quality indicators achieved by the combination of storage assignment and order picking algorithm in Setting 3 to 5 (best values are shown in bold).

Setting	Policy	Cov [μ]	GD [μ]	ED [μ]	PFS [μ]	GS [μ]	IGD [μ]
3.a	Random, ACO_3	0.00	60.32	215.43	9.96	**0.97**	56.49
	NSGA-2, ACO_3	**1.00**	**0.00**	**22.62**	**12.20**	1.12	**0.00**
3.b	Random, ACO_3	0.00	38.37	168.05	**13.76**	**0.91**	38.85
	NSGA-2, ACO_3	**1.00**	**0.00**	**36.65**	12.12	0.96	**0.00**
3.c	Random, ACO_3	0.00	50.06	213.69	**13.96**	0.90	51.35
	NSGA-2, ACO_3	**1.00**	**0.00**	**32.78**	12.02	**0.87**	**0.00**
4.a	Random, ACO_4	0.00	54.03	165.75	8.14	**0.93**	46.16
	NSGA-2, ACO_4	**1.00**	**0.00**	**25.08**	**10.26**	0.96	**0.00**
4.b	Random, ACO_4	0.01	44.61	198.83	**11.82**	0.86	48.94
	NSGA-2, ACO_4	**0.99**	**0.12**	**35.65**	11.10	**0.71**	**0.21**
4.c	Random, ACO_4	0.00	51.19	207.95	**12.04**	0.85	52.91
	NSGA-2, ACO_4	**1.00**	**0.00**	**38.08**	10.04	**0.71**	**0.03**
5.a	NSGA-2, ACO_3	**0.80**	**1.43**	**25.30**	**10.26**	0.99	1.38
	NSGA-2, ACO_4	0.66	1.52	26.58	9.32	**0.88**	**0.98**
5.b	NSGA-2, ACO_3	**0.69**	**1.40**	**21.37**	**9.84**	0.94	3.53
	NSGA-2, ACO_4	0.41	3.83	22.85	8.14	**0.79**	**3.13**
5.c	NSGA-2, ACO_3	**0.79**	**1.92**	**33.80**	9.16	0.81	**3.11**
	NSGA-2, ACO_4	0.22	8.69	43.25	**9.18**	**0.66**	5.71

The NSGA-II algorithm needs more time to compute the storage allocations and requires up to 15 seconds for the large warehouse. However, the execution times of the NSGA-II algorithm may be considered acceptable, as the algorithm requires only a few seconds to find storage allocations that are remarkably better than the ones found by the alternative policies.

For order picking, the alternative S-Shape heuristic requires 0.15 regardless of warehouse size. The $AC0_3$ and the $AC0_4$ achieve fairly the same execution times across the different warehouse sizes, with approximately six seconds for the largest. The execution time increases with the warehouse size since the graph consists of more markets, causing more ants to be deployed in each iteration. Considering the noticeably better routes compared to the S-Shape heuristic, the execution time of a few seconds to find a picking route may be considered acceptable.

7.8 Evaluation of the Interaction between NSGA-II and ACO Algorithm

In the previous section, we have shown the applicability of our algorithms for storage assignment and order picking in dedicated analyses. The results indicate that both algorithms outperform state-of-the-art solutions for those tasks. In this section, we evaluate the interaction between our proposed algorithms by assessing them in three settings: Section 7.8.1 determines whether the ACO_3 performs better on the NSGA-II planned warehouse compared to the random warehouse; Section 7.8.2 performs a similar assessment for the ACO_4; Section 7.8.3 evaluates whether the ACO_3 or the ACO_4 perform better on the NSGA-II planned warehouse. Tables 9 and 10 summarize the results.

Table 10: Standard deviations of the six quality indicators achieved by the combination of storage assignment and order picking algorithm in Setting 3 to 5.

Setting	Policy	Cov [σ]	GD [σ]	ED [σ]	PFS [σ]	GS [σ]	IGD [σ]
3.a	Random, ACO_3	0.00	22.94	62.38	4.10	0.06	22.96
	NSGA-2, ACO_3	0.00	0.00	7.51	6.17	0.35	0.00
3.b	Random, ACO_3	0.00	13.66	35.09	3.88	0.07	14.41
	NSGA-2, ACO_3	0.00	0.00	16.50	4.18	0.36	0.00
3.c	Random, ACO_3	0.00	14.03	54.74	3.56	0.07	17.53
	NSGA-2, ACO_3	0.00	0.00	8.19	5.40	0.32	0.00
4.a	Random, ACO_4	0.00	18.52	73.60	3.69	0.10	26.01
	NSGA-2, ACO_4	0.00	0.00	8.71	3.65	0.31	0.00
4.b	Random, ACO_4	0.03	15.52	62.11	2.96	0.09	22.17
	NSGA-2, ACO_4	0.03	0.57	22.53	4.20	0.28	0.77
4.c	Random, ACO_4	0.01	21.77	68.40	3.55	0.08	26.20
	NSGA-2, ACO_4	0.01	0.00	12.91	3.82	0.28	0.22
5.a	NSGA-2, ACO_3	0.15	2.70	6.61	4.74	0.20	1.93
	NSGA-2, ACO_4	0.20	2.11	7.98	3.72	0.20	0.93
5.b	NSGA-2, ACO_3	0.16	1.74	4.58	3.43	0.25	3.46
	NSGA-2, ACO_4	0.16	5.59	4.58	1.90	0.22	3.48
5.c	NSGA-2, ACO_3	0.14	3.81	12.68	3.28	0.25	4.40
	NSGA-2, ACO_4	0.14	6.32	17.01	3.54	0.18	3.54

7.8.1 Comparison of NSGA-II and Random Planned Warehouses for ACO_3

In this setting, we apply the ACO_3 algorithm on all warehouse sizes (Setting 3.a, 3.b, 3.c) twice: once for the warehouse that used the NSGA-II algorithm for storage assignment and once for the randomly assigned warehouse. Again, we select five random items from the product assortment and set the amount to assign to the already existing amount inside the warehouse. In the small warehouse, the Pareto front of the NSGA-II planned warehouse covers the entire reference front while the random planned warehouse does not cover a single solution in the reference front, hence, the GD and IGD values of the NSGA-II planned warehouse are 0 and the ED values are minimal. The high GD and ED values of the random planned warehouse indicate that its Pareto front does not converge well towards the reference front. Thus, the solutions found in the random warehouse are considerably worse than the solutions found in the NSGA-II warehouse. In the medium warehouse, the Pareto fronts of random and NSGA-II planned warehouses achieve fairly the same quality indicator values as in the previous setting. However, the GD and ED metric indicate that the results for the random warehouse unexpectedly converge better towards the reference front than in Setting 3.a. This could be due to the limited amount of executed problem instances and needs to be further assessed with a higher number of problem instances. Nevertheless, the Pareto front of the random warehouse is still far from converging towards reference front. In the large warehouse, the same observations can be made as in the previous settings, underlining that the ACO_3 variant finds better pick routes in the NSGA-II warehouse than in the random warehouse. In summary, the evaluation results show that the NSGA-II algorithm and the ACO_3 variant interact well together and the ACO_3 variant profits from the NSGA-II algorithm that ensures our four economic constraints.

7.8.2 Comparison of NSGA-II and Random Planned Warehouses for ACO_4

This setting repeats the Settings 3.a to 3.c for the ACO_4 algorithm. We now discuss the results for the Settings 4.a to 4.c. In the small warehouse, the Pareto front of the NSGA-II warehouse covers the entire reference Pareto front and the random warehouse fails to cover a single solution. Thus, all solutions found in the random warehouse are dominated by the solutions of the NSGA-II warehouse. The GD and ED values for the random warehouse are higher than the ones of the NSGA-II warehouse which shows that the Pareto front of the random warehouse is further away from the reference front. Hence, the solutions of the random warehouse are noticeably worse than the solutions found in the NSGA-II warehouse. In the medium warehouse, the Pareto front of the NSGA-II warehouse does not always cover the entire reference front, while the random warehouse covers at least one solution in the reference front in 7 of 50 repetitions. Thus, the ACO_4 variant occasionally finds a few solutions in the random warehouse that are comparable with the solutions found in the NSGA-II warehouse. Despite these few outliers, the results show a similar behavior as in the previous setting. Similar to the previous settings, the evaluation in the large warehouse show comparable results. The NSGA-II warehouse covers all solutions in the reference front in 49 of 50 repetitions and the random warehouse manages to cover at least one solution in the reference front. In summary, the results show that the ACO_4 variant also finds better pick routes if the warehouse applies the NSGA-II storage strategy and both algorithms interact well with each other.

7.8.3 Comparison of ACO_3 and ACO_4 on NSGA-II Planned Warehouses

This section investigates which ACO variant performs better if the warehouse applies the NSGA-II storage strategy. Both variants are applied on five randomly selected customer orders on all warehouse sizes (Settings 5.a, 5.b, 5.c). In the small warehouse, the Pareto front of the ACO_3 algorithm covers approximately 80% of the reference front, while ACO_4 covers only 66%. Both ACO variants converge well towards the reference front as indicated by the low GD and ED values and find approximately ten solutions per problem instance, and IGD values of both fronts are almost the same. However, the GS values indicate, that the solutions of the ACO_4 variant have a better distribution than the ones of ACO_3. In the medium warehouse, the Pareto front of ACO_3 covers approximately 69% of the reference front, while the one from ACO_4 covers only 41%, and thus, the solutions found by ACO_4 tend to be dominated by the ones from ACO_3. The GD and ED metric indicate that the ACO_3 Pareto front converges better towards the reference front. Similar to the small warehouse, the GS indicate, that solutions of the ACO_4 Pareto front are better distributed. In the large warehouse, the ACO_3 dominates ACO_4 even more with regards to the coverage metric. Furthermore, the GD and ED values of ACO_4 increased, indicating that the distance between the solutions in ACO_4 Pareto front and the solutions in the reference front become larger. Again, the solutions in the ACO_4 Pareto front have a slightly better distribution than the solutions in the ACO_3 Pareto front. However, this time, ACO_3 achieves better IGD values, as ACO_3 covers large parts of the reference front. In summary, we can state that with increasing warehouse size, the ACO_3 variant finds better pick routes than the ACO_4 variant while both variants find approximately the

same number of solutions per problem instance. However, the solutions found by the ACO_4 variant are slightly better distributed than the ones from the ACO_3 variant.

7.9 Threats to Validity

We identified the following threats to validity of our evaluation. First, the NSGA-II and ACO algorithms are evaluated in three mezzanine warehouses of different sizes. However, real-world mezzanine warehouses may consist of more floors, blocks, pick aisles, and racks than specified in the warehouses used for evaluation. Nevertheless, we are convinced that our defined warehouses form a representative set for mezzanine warehouses and can easily be extended for further evaluation runs. Second, since the algorithms are evaluated in warehouses that apply either the random or the NSGA-II storage strategy, the evaluation results may not be transferable to warehouses that apply different storage strategies. Even though the product assortment, the product correlations, the customer orders, and the storage allocations are randomly generated reflecting specific characteristics of real-world mezzanine warehouses, the proposed algorithms are easily transferable to real application data. Third, we decided to compare the ACO algorithm only to one order picking policy. This decision was made in awareness of the limited expressiveness of our results but was necessary as the majority of policies in the literature violate assumptions made in this work. Finally, we only evaluate our NSGA-II and ACO algorithms against heuristic policies. Hence, they also should be evaluated against other optimization methods like other evolutionary optimization algorithms or graph-based optimization techniques. However, we decided to do this evaluations as future work. Additionally, we performed less than 30 runs for each setting due to the complexity and runtime of the runs. Obviously, 30 runs and more might result in more stable and reliable results. However, the report the standard deviations. As those are pretty low, we assume that the results are already stable.

8 Conclusion

Due to the complexity of the storage assignment and the order picking problem, efficient optimization algorithms are required to find satisfactory solutions within reasonable times. This paper proposes an NSGA-II algorithm for optimizing the storage assignment problem, and an ACO algorithm for optimizing the order picking problem in mezzanine warehouses. The algorithms incorporate knowledge about the interdependency between both problems to improve the overall warehouse performance. Besides optimizing economic constraints, the algorithms also optimize ergonomic criteria, as mezzanine warehouses represent labor-intensive working environments in which the employees account for a large part of the warehouse performance. We evaluate the NSGA-II algorithm against three storage assignment policies frequently applied in practice: the random, the closest open location, and the rank-based policy. The evaluation results show that the NSGA-II algorithm outperforms the alternatives already in smaller warehouses and the

larger the warehouse, the better the NSGA-II algorithm prevails against the alternative policies. We evaluate the ACO algorithm against the S-Shape heuristic that is frequently applied in practice. Our evaluation results show that the ACO outperforms the S-Shape heuristic in all tested warehouse sizes. Finally, we evaluate the interaction between the NSGA-II and the ACO algorithm. The evaluation results show that both ACO variants find better pick routes if the warehouse assigns its products by applying the NSGA-II algorithm instead of the random storage strategy, thus, the NSGA-II and the ACO algorithm interact well with each other.

In the future, we plan to integrate additional features to further increase the applicability of the storage assignment. First, we want to allow state changes of the mezzanine warehouses while the storage assignment is running which would make some of the solutions in the Pareto front infeasible. Further, we plan to parallelize the storage assignment algorithm so that the sequential assignment of products is replaced and the execution times will decrease. Regarding the order picking algorithm, we also aim at parallelizing the execution to reduce the required calculation times. Finally, we want to research on integrating forecasts (e.g., based on our previous work [33]) of future assignment and order picking tasks to proactively replace goods within the storage that will be ordered in the near future. Aditionally, a promising option can be to study further algorithms for optimization and different parameter setting. On the one hand, one option is to analyze additional optimization techniques. For example, some multi-objetive evolutionay algorithms with great performance have been reported recently, such as, Multi-strategy co-evolutionary differential evolution for mixed-variable optimization, firefly algorithms with courtship learning, or multi-objective artificial bee colony algorithms. However, those algorithms were applied in different domains. Still, it would be interesting as future work to analyze them in our target domain. On the other hand, we showed In previous work that choosing the best optimization algorithm is situation-aware [18] and, hence, requires to be modeled as a self-adaptive system [15]. Implementing such an approach could also help to identify the best fitting algorithms for a new warehouse design.

References

1. Alaya, I., Solnon, C., Ghedira, K.: Ant Colony Optimization for Multi-Objective Optimization Problems. In: 19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007), vol. 1, pp. 450–457 (2007)
2. Bartholdi III, J.J., Hackman, S.T.: Warehouse and Distribution Science. Supply Chain and Logistics Institute, School of Industrial and Systems Engineering, Georgia Institute of Technology (2019)
3. Calzavara, M., Glock, C.H., Grosse, E.H., Sgarbossa, F.: An integrated storage assignment method for manual order picking warehouses considering cost, workload and posture. International Journal of Production Research **57**(8), 2392–2408 (2019). DOI 10.1080/00207543.2018.1518609. URL `https://doi.org/10.1080/00207543.2018.1518609`
4. Chen, F., Wang, H., Qi, C., Xie, Y.: An ant colony optimization routing algorithm for two order pickers with congestion consideration. Computers & Industrial Engineering **66**(1), 77–85 (2013)
5. Daniels, R.L., Rummel, J.L., Schantz, R.: A model for warehouse order picking. European Journal of Operational Research **105**(1), 1–17 (1998)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002)

7. Ene, S., Öztürk, N.: Storage location assignment and order picking optimization in the automotive industry. The International Journal of Advanced Manufacturing Technology **60**(5-8), 787–797 (2012)
8. van Gils, T., Ramaekers, K., Caris, A., de Koster, R.B.M.: Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. European Journal of Operational Research **267**, 1–15 (2018)
9. Gu, J., Goetschalckx, M., McGinnis, L.F.: Research on warehouse design and performance evaluation: A comprehensive review. European Journal of Operational Research **203**(3), 539–549 (2010)
10. Izquierdo, J., Campbell, E., Montalvo, I., Pérez-García, R.: Injecting problem-dependent knowledge to improve evolutionary optimization search ability. Journal of Computational and Applied Mathematics **291**, 281–292 (2016)
11. Kofler, M., Beham, A., Wagner, S., Affenzeller, M., Reitinger, C.: Reassigning storage locations in a warehouse to optimize the order picking process. In: Proceedings of the 22th European Modeling and Simulation Symposium, pp. 77–82 (2010)
12. de Koster, M.B.M.: Warehouse assessment in a single tour. In: Facility Logistics, pp. 53–74. Auerbach Publications (2007)
13. de Koster, R., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: A literature review. European Journal of Operational Research **182**(2), 481–501 (2007)
14. Kovács, A.: Optimizing the storage assignment in a warehouse served by milkrun logistics. International Journal of Production Economics **133**(1), 312–318 (2011)
15. Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G., Becker, C.: A Survey on Engineering Approaches for Self-Adaptive Systems. Pervasive and Mobile Computing Journal **17**(Part B), 184–206 (2015)
16. Lee, I.G., Chung, S.H., Yoon, S.W.: Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations. Computers & Industrial Engineering **139**, 106129 (2020). DOI https://doi.org/10.1016/j.cie.2019.106129
17. Leng, J., Yan, D., Liu, Q., Zhang, H., Zhao, G., Wei, L., Zhang, D., Yu, A., Chen, X.: Digital twin-driven joint optimisation of packing and storage assignment in large-scale automated high-rise warehouse product-service system. International Journal of Computer Integrated Manufacturing **34**(7-8), 783–800 (2021)
18. Lesch, V., Noack, T., Hefter, J., Kounev, S., Krupitzer, C.: Towards situation-aware meta-optimization of adaptation planning strategies. In: 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), pp. 177–187 (2021). DOI 10.1109/ACSOS52086.2021.00042
19. Li, M., Chen, X., Liu, C.: Pareto and niche genetic algorithm for storage location assignment optimization problem. In: 3rd International Conference on Innovative Computing Information and Control, pp. 465–465. IEEE (2008)
20. Manzini, R., Gamberi, M., Persona, A., Regattieri, A.: Design of a class based storage picker to product order picking system. International Journal of Advanced Manufacturing Technology **32**, 811–821 (2007)
21. Mirzaei, M., Zaerpour, N., de Koster, R.B.: How to benefit from order data: correlated dispersed storage assignment in robotic warehouses. International Journal of Production Research **60**(2), 549–568 (2022). DOI 10.1080/00207543.2021.1971787. URL `https://doi.org/10.1080/00207543.2021.1971787`
22. Petersen, C.G.: An evaluation of order picking routeing policies. International Journal of Operations & Production Management **17**(11), 1098–1111 (1997)
23. Petersen, C.G., Siu, C., Heiser, D.R.: Improving order picking performance utilizing slotting and golden zone storage. International Journal of Operations & Production Management **25**(10), 997–1012 (2005)
24. Petersen II, C.G., Schmenner, R.W.: An Evaluation of Routing and Volume-based Storage Policies in an Order Picking Operation. Decision Sciences **30**(2), 481–501 (1999)
25. Ratliff, H.D., Rosenthal, A.S.: Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Operations Research **31**(3), 507–521 (1983)
26. Roodbergen, K.J., de Koster, R.: Routing methods for warehouses with multiple cross aisles. International Journal of Production Research **39**(9), 1865–1883 (2001)
27. Shqair, M., Altarazi, S., Al-Shihabi, S.: 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 (2014)

28. Sooksaksun, N., Kachitvichyanukul, V., Gong, D.C.: A class-based storage warehouse design using a particle swarm optimisation algorithm. International Journal of Operational Research **13**(2), 219–237 (2012)
29. Vaughan, T.: The effect of warehouse cross aisles on order picking efficiency. International Journal of Production Research **37**(4), 881–897 (1999)
30. Wang, M., Zhang, R.Q., Fan, K.: Improving order-picking operation through efficient storage location assignment: A new approach. Computers & Industrial Engineering **139**, 106186 (2020). DOI https://doi.org/10.1016/j.cie.2019.106186
31. Wang, S., Ali, S., Yue, T., Li, Y., Liaaen, M.: A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In: Proceedings of the 38th International Conference on Software Engineering, pp. 631–642 (2016)
32. Xing, B., Gao, W.J., Nelwamondo, F.V., Battle, K., Marwala, T.: Ant colony optimization for automated storage and retrieval system. In: IEEE Congress on Evolutionary Computation, pp. 1–7. IEEE (2010)
33. Zuefle, M., Bauer, A., Lesch, V., Krupitzer, C., Herbst, N., Kounev, S., Curtef, V.: Autonomic forecasting method selection: Examination and ways ahead. In: 2019 IEEE International Conference on Autonomic Computing (ICAC), pp. 167–176 (2019). DOI 10.1109/ICAC.2019.00028

A Variable Summary

Table 11: Overview on used variables in Section 3 and 4.

Variable	Explanation
r_i	Rack with number i
$x \mid y$	Rack coordinates x and y
p	Product p
$rank_p$	Rank of product p
$\|P\|$	Size of the product assortment

Table 12: Overview on used variables in Section 5.

Variable	Explanation
f_j	Floor with id j
A	Area A
$totalQ$	Total quantity of a product in an area
$idealQ$	Ideal quantity of a product in an area
C	Chromosome representation of an assignment solution
$dist_{r_i}$	Walking distance to rack i
$idealDist_{p,f_j}$	Ideal walking distance for product p
$relRank_p$	Relative rank of product p
$rackIdx_{p,f_j}$	Rack index of product p on floor f_j
R_{f_j}	List of rack on floor f_j
tq_p	Target quantity for product p
M_k	Mask with identifier k
q	Quantity of products inside a mask
sa_l	Sub aisle with identified l
$qFactor$	Factor of actual existing quantity of a product compared to the target quantity
$maskScore$	Score for a mask indicating how close the actual quantity of a product is to the target quantity
$subAisleScore$	Score for each sub aisle based on the $maskScore$
$quantityScore$	Score for a floor based on the $subAisleScore$
cp	Correlated product
$conf$	Confidence value of a correlation
$qClusters$	Possible clusters of target quantities
$idealCorrQ$	Ideal Quantity to which the correlated product should be available in the vicinity of the incoming product
$corrQ$	Fraction to which the items of cp stored in r are considered to be in vicinity of p
$exQ_{cp,r}$	Existing quantity of the correlated product cp in rack r
L	Number of generations to be used for calculating the standard deviation of the maximum crowding distance
δ_{lim}	Threshold for the standard deviation of the crowding distance

Table 13: Overview on used variables in Section 6.

Variable	Explanation
k	Ant with identifier k
m	Market with identifier m
$d_{m,n}$	Manhattan distance between two markets m and n
U^k	Set of markets ant k did not visit yet
I_n^k	Percentage to which the required items of ant k are available at market n
$d_{m_1}^{pd}$	Distance between p/d-point and the first market to be visited
$d_{r_i}^{sub}$	Walking distance within a sub-aisle until rack i
$d_{m_i}^{cross}$	Walking distance within a cross lane until market m_i
$d_{m_i,m_{i+1}}^{market}$	Walking distance between visited markets
$d_{m_j}^{pd}$	distance from last visited market to next p/d-point
$prob_{m,n}^k$	Probability that ant k moves from market m to market n
$\tau_{m,n}$	Pheromone value of edge (m,n)
$\eta_{m,n}^k$	Heuristic value for ant k on edge (m,n)
$\tau_{m,n}^i$	Pheromone value of edge (m,n) in matrix i
PR_{ib}^i	Iteration-best pick route that minimizes objective function of_i
PR_{gb}^i	Global-best pick route that minimizes objective function of_i