University of Groningen

Shared Warehouse Layout Problem with Order Picking

Design of Combinatorial Algorithms: Project

Authors:	*Student number:*
Ivar Schilstra	S3446743
Pietro Mietto	S5095409

April 5, 2022

Abstract

This paper concerns the shared warehouse layout problem with unequal variable picking area surfaces (SWLP). The SWLP is solved by minimizing the amount of time spent in collecting the correct packages from the warehouse and delivering them to the customer. We create a meta-heuristic to solve the SWLP which combines two procedures: adaptive large neighborhood search (ALNS) and simulated annealing. With the meta-heuristic can to interchange picking area positions and adjust individual picking area layouts. The used data structure follows the flexible bay structure (FBS). Usually, FBS consists of a horizontal bay structure, we extend this feature with a vertical bay structure. The results show that the vertical FBS have lower objective values. Operator effectiveness indicates that one should pay attention to which operators one should use. Turning off operators could save computational time while the objective value does not increase substantially.

Keywords: Shared Warehouse Layout Problem (SWLP), FBS, ALNS, Simulated Annealing.

1 Introduction

In the last few years, the e-commerce sector has been growing exponentially. Consumers appreciate the convenience of purchasing products online which avoids trips to physical stores. In addition, consumers can also enjoy a greater variety of products and services both in terms of quality and price. For an e-commerce company, it is important to have an efficient strategy to satisfy customers' demands as quickly as possible. Companies in e-commerce are constantly trying to identify new strategies that can make their production and distribution more efficient in terms of time and overall customer experience.

One way in which these companies are trying to improve their distribution chains is by outsourcing their warehousing activities to logistics service providers. Logistics service providers are expected to support e-commerce companies in the process of managing their resources. By more optimally managing such resources, e-commerce companies should be able to significantly lower manufacturing costs, work in process inventory, lead time, and increase productivity (Drira, Pierreval, and Hajri-Gabouj 2006). A practical example of how better use of a company's resources can help lower its overall costs is to optimally assign workers to different warehouse areas to minimize time wasted in unnecessary movements. Another cost-minimizing trend is having multiple companies under the same warehouse. The process of optimizing Shared Warehouses layout configurations is commonly referred to as a *Shared Warehouse Layout Problem* (SWLP).

Order picking is the most critical warehousing function since it is the one that must directly match customer expectations. According to warehousing professionals, order picking has the highest priority in the warehouse when it comes to productivity improvements (Tompkins, White, Bozer, and Tanchoco 2003). The efficiency of the order-picking process may, under certain conditions, be improved by enlarging the floor space assigned to an order-picking area. This is because increasing the amount of open floor space may allow for more direct movement between locations, thus lowering travel distances (Roodbergen and De Koster 2001).

We use the travel distances of the workers inside and outside the order-picking areas as our key performance indicator. Even though the travel distance is an operational measure, there are several reasons for its usefulness. First of all, travel time is the main non-productive activity of the workers, which makes it different from other important activities such as product identification, counting products, and the actual handling of products. For this reason, reduction of travel distances is the most obvious approach for achieving efficiency improvement in the process. Furthermore, Hackman, Frazelle, Griffin, Griffin, and Vlasta (2001) observe that warehouses that achieved an overall high operating efficiency focused on minimizing travel time.

The literature suggests that two macro-methods can be used to solve a *Shared Warehouse Layout Problem*. The first macro-method is usually referred to as "exact method" and comprises methods such as the branch and bound algorithm by Kouvelis, Kurawarwala, and Gutierrez (1992), and the dynamic programming method by Rosenblatt (1986). Since exact approaches are often found not to be suited for large-size problems, numerous researchers have developed heuristic and metaheuristic methods, which belong to the second class of macro-methods. Heuristic and metaheuristic approaches aim at either building a solution or at improving an existing one.

Among the metaheuristic methods, one can distinguish local search methods (Tabu search and simulated annealing) and evolutionary approaches (genetic and ant colony algorithms). Chiang and Kouvelis (1996) developed a Tabu search algorithm to solve the facility layout problem that incorporates all basic features of the tabu search framework. Chwif, Barretto, and Moscato (1998) used a simulated annealing algorithm to solve the layout problem with aspect ratio facilities sizes. Genetic algorithms have become quite popular in solving facility layout problems (Pierreval, Caux, Paris, and Viguier 2003). In fact, a large number of studies using such approaches have been published.

In this paper, we create a meta-heuristic to solve the Shared Warehouse Layout Problem which combines two procedures: adaptive large neighborhood search (ALNS) and simulated annealing. For the data structure we follow the flexible bay structure. We built on the pre-existing methods used by Santini, Ropke, and Hvattum (2018) and Sacramento Lechado, Pisinger, and Ropke (2019). The objective is to identify the warehouse layout that minimizes the amount of time spent in collecting and delivering the correct packages. Throughout the paper, we interchange the definition of departments and picking areas. The produced algorithm is used twice, the algorithm differs in the flexible bay data structure. We distinguish between a horizontal and vertical flexible bay structure. The results indicate that the vertical bay structure is more appropriate since the objective values for the vertical final warehouse layouts are lower and that the horizontal flexible bay structure could not find a feasible solution for all 450 instances. Extra insights into the results can be found in Section 6.

In Section 2 we present a description of the problem and introduce parameters and variables. Section 3 presents a novel method for estimating travel distance. In Section 4, the model with a formal definition of the data structure is given. Section 5 explains the metaheuristic used, Adaptive Large Neighborhood Search. Section 6 reports and gives insights into the computational experiments. A conclusion is given in Section 7.

2 Problem Definition

We now present the problem mathematically. See Figure 1 for an illustration. We have a warehouse with a given width W and depth D. The warehouse has dock doors on one side (at the bottom of the figure). We have to place Z picking areas, one for each of Z companies (five such areas are depicted in Figure 1). The objective is to minimize total (average) travel distances. Total travel is composed of (1) travel within picking areas, where workers move between locations to retrieve products that were ordered by customers, (2) travel to replenish locations, and (3) travel from the picking areas to the dock doors, and vice versa.

For each picking area, a layout must be chosen. Essentially, this consists of selecting the width and depth of each area. A picking area for company i ($i = 1, \ldots, Z$) is composed of n_i aisles ($n_i = 1, \ldots, N$), with k_i cross aisles ($k_i = 2, \ldots, K$) perpendicular to the aisles. Both n_i and k_i are variables. See Figure 2, which originates from Roodbergen and De Koster (2001), in which paper you can also find more explanations of terminology. Note that in that paper the word 'warehouse' is used for what we call a 'picking area' here. And here we use the word 'warehouse' to encompass several 'picking areas'. The required size of the picking area for company i is given and expressed by the parameter S_i, the total required length of the aisles. The number of aisles n_i ($n_i = 1, \ldots, N$) and the length of the aisles l_i ($l_i \geq 0$) must be chosen such that $n_i l_i = S_i$. The resulting width of picking area i is $w_i n_i$, where w_i is the width of an aisle, and the depth of picking area i is $v_i k_i + l_i$, where v_i is the width of a cross-aisle.

Each picking area must be positioned in the warehouse. The position of picking area i is denoted by (x_i, y_i) (see also Figure 3). Here x_i gives the distance from the left wall of the warehouse to the left side of picking area i, and y_i gives the distance from the front of the warehouse to the front of picking area i. That is, we use the 'lower-left corner' of the warehouse of the picking areas as reference points. Note that picking areas must be positioned such that they are entirely within the building, and that they do not overlap each other. The mathematical model presented below contains constraints to achieve this. Each picking area has a 'depot' where all routes in the area start and end. It is positioned on the left of the front cross-aisle. This is the 'lower-left corner', as in Figure 2. Picking areas will not be rotated, and Figure 2 is drawn consistently with the layout in Figures 1 and 3, so the depot of a picking area is on that side of the picking area which is closest to the dock doors. And aisles in any picking area run always from the 'front' to the 'back' of the warehouse, as in Figure 2.

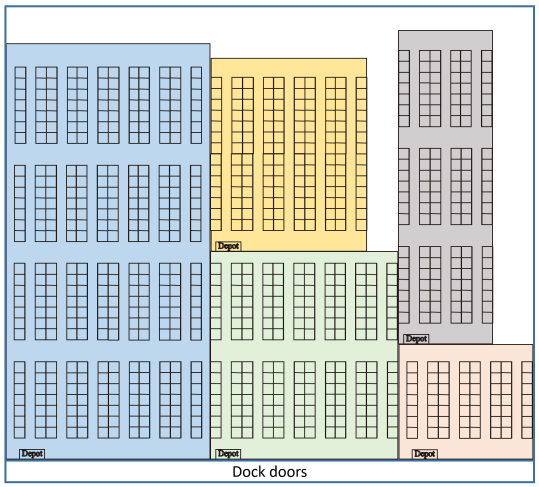


Figure 1: Top view of a warehouse with five picking areas.

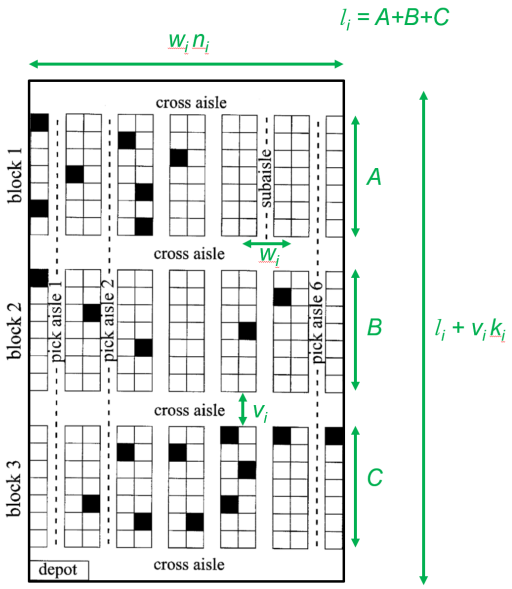


Figure 2: Top view of a picking area with $n_i = 6$ and $k_i = 4$.

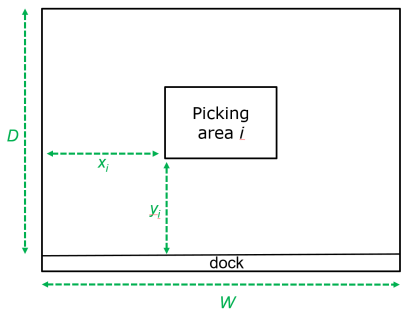


Figure 3: Visualization of x_i and y_i.

3 Travel Distances

This section presents a method for determining travel distances. Total travel is composed of (1) travel within picking areas, where workers move between locations to retrieve products that were ordered by customers, (2) travel to replenish locations, and (3) travel from the picking areas to the dock doors, and vice versa. Each of the components is explained separately below.

3.1 Travel for Order Picking

We start with a "standardized" picking area with k cross aisles, n aisles of length 1, an aisle width of 1, and a cross aisle width of 1. Let g_{nkm} be the average travel distance within the aisles of this standardized warehouse if m items must be picked (marked blue in Figure 4). And let h_{nkm} be the average left-to-right travel distance within the cross aisles of this standardized warehouse (marked yellow in Figure 4). Finally, let f_{nkm} be the average front-to-back travel distance within the cross aisles of the standardized warehouse (marked purple in Figure 4). For picking area i with v_i as cross aisle width, w_i as aisle width, k_i cross aisles, and n_i aisles of length $l_i = S_i/n_i$, we can now estimate travel time for picking a single order consisting of m items by:

$$T_{im} = l_i g_{n_i k_i m} + w_i h_{n_i k_i m} + v_i f_{n_i k_i m}. \tag{1}$$

This estimate is quite accurate as long as travel distances in the aisles and cross aisles are independent, which is true for many heuristics. Values for g_{nkm}, h_{nkm}, and f_{nkm} are determined by means of simulation. Note that the above formula would be very straightforward to use in a heuristic when values for n_i and k_i are available, and l_i can easily be determined as S_i/n_i. To obtain an estimate for average travel time for picking an order, taking different order sizes and their probabilities into account, we use:

$$T_i = \sum_{m=1}^{M} u_{im} T_{im}, \tag{2}$$

where u_{im} gives the probability that an order in picking area i has m items.

3.2 Travel for Replenishing Locations

A replenishment route serves to restock the pick locations in the warehouse. Usually, restocking occurs with large quantities of the same product, so a route that serves to replenish a location will visit only one location. Hence from a travel distance perspective, it has the same length as a picking route with just one item on the picking list. The length for this is $T_{i1}(n_i, k_i)$. We only replenish when a product is no longer available in the storage location, i.e., when it is out of stock. We assume that the average number of replenishment needed per order for picking area i equals α_i. Then the contribution of an order to travel distance for replenishment can be calculated as:

$$R_i = \alpha_i T_{i1}. \tag{3}$$

3.3 Travel Between Picking Areas and the Dock Doors

The third component is the distance from the picking areas' depots to the dock doors, and vice versa. This distance is traveled (1) to transport picked orders to the dock so that they can subsequently be loaded into trucks, and (2) to transport received goods from the dock doors to the picking areas, so that they can subsequently be stored in locations. And We calculate this distance only 'front-to-back', not 'left-to-right', because we do not know at which dock door the goods are needed. So we need to measure the distance from the 'lower-left' corner of the picking area to the docks. This distance is y_i. This distance needs to be traveled twice for each order, and additionally $2\alpha_i$ times for replenishment of locations due to the order. In total this gives:

$$U_i = 2y_i(1 + \alpha_i) \tag{4}$$

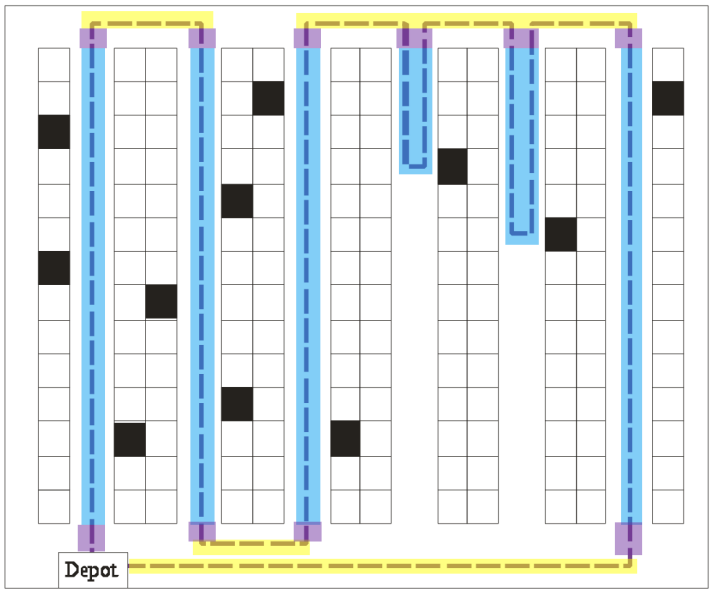


Figure 4: Components of travel distance.

4 The Model

The model can be formulated as follows.

$$\min \sum_{i=1}^{Z} (T_i + R_i + U_i) \tag{5}$$

subject to

$$x_i \leq W - n_i w_i \qquad i \in \{1, \ldots, Z\} \tag{6}$$

$$y_i \leq D - l_i - v_i k_i \qquad i \in \{1, \ldots, Z\} \tag{7}$$

$$l_i = \frac{S_i}{n_i} \qquad i \in \{1, \ldots, Z\} \tag{8}$$

$$T_i = \sum_{m=1}^{M} u_{im} \left(\frac{S_i}{n} g_{nkm} + w_i h_{nkm} + v_i f_{nkm} \right) \qquad i \in \{1, \ldots, Z\} \tag{9}$$

$$R_i = \alpha_i \left(\frac{S_i}{n} g_{nk1} + w_i h_{nk1} + v_i f_{nk1} \right) \qquad i \in \{1, \ldots, Z\} \tag{10}$$

$$U_i = 2y_i(1 + \alpha_i) \qquad i \in \{1, \ldots, Z\} \tag{11}$$

$$l_i, x_i, y_i, T_i, R_i, U_i \geq 0 \qquad i \in \{1, \ldots, Z\} \tag{12}$$

$$n_i \in \{1, \ldots, N\} \qquad i \in \{1, \ldots, Z\} \tag{13}$$

$$k_i \in \{2, \ldots, K\} \qquad i \in \{1, \ldots, Z\} \tag{14}$$

Constraints (6)–(7) ensure each picking area fits entirely inside the warehouse. Constraints (8) determine aisle length of each picking area based on the number of aisles. Constraints (9)–(11) determine travel distances. Constraints (12)–(14) define the nature and domain of the variables. An overview of all parameters and variables of the model are given in Table 1 and 2, respectively.

W	width of the entire warehouse
D	depth of the entire warehouse
w_i	width of an aisle in picking area i
v_i	width of cross aisle in picking area i
S_i	The total required length of all aisles in picking area i
N	maximum number of aisles that is allowed in any company's picking area
K	maximum number of cross aisles that is allowed in any company's picking area
Z	number of companies that each need a picking area in the warehouse
M	maximum number of items on any customer order
u_{im}	the probability that an order in picking area i has m items
α_i	the average number of replenishments needed per order for picking area i.

Table 1: The parameters of the model.

n_i	number of aisles in picking area i, $n_i \in \{1, \dots, N\}$
k_i	number of cross aisles in picking area i, $k_i \in \{2, \dots, K\}$
T_{im}	Average travel distance for picking an order with m items in picking area i
T_i	Average travel distance for picking an order in picking area i
R_i	The contribution to travel distance for replenishments in picking area i due to one order.
U_i	The distance traveled between picking area i and the dock doors.
l_i	length of each aisle in picking area i
x_i	x-coordinate (lower-left corner) of picking area i
y_i	y-coordinate (lower-left corner) of picking area i

Table 2: The variables of the model.

4.1 Data Structure

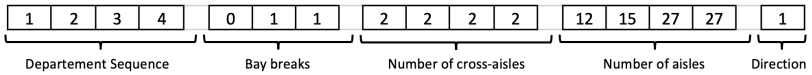
The main data structure follows the Flexible Bay Structure (FBS). We follow partly the FBS structure from Wong and Komarudin (2010), which consists of a department and bay break sequence. However, as described in Section 2 the surface of departments is not constant. Each department consists of several aisles and cross-aisles. Changes in the department size could reduce the objective function. Since more aisles imply longer cross aisles, the surface of picking areas can be made larger or smaller, and the shape of picking areas directly influences the objective value. For this reason, we have added sequences for the number of cross-aisles and aisles are added to the data structure. The complete data structure consists solely of integer numbers. The traditional FBS cuts the warehouse completely horizontally with each bay, this restricts the solution space for the metaheuristic which results in smaller neighborhood searches. To enlarge the solution space we include the possibility to have a vertical bay structure, which doubles the solution space for neighborhood search. The last number of the data structure indicates the bay break direction.

4.2 Formal Definition Data Structure

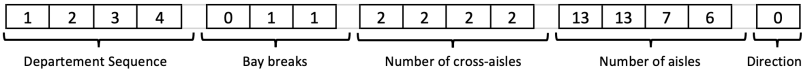
As given in Section 2 we have $Z \in \mathbb{N}$ picking areas, one for each of Z companies. Hence, the first part of the data structure consists of Z numbers. The second part of the data structure corresponds to the bay break structure. There will be in total $N = Z - 1$ bay break elements, where $N_i \in \{0, 1\}\ \forall i$. For each picking are we store the number of cross-aisles and aisles. Since we would like to start with a feasible solution we fix the number of cross aisles in the initial solution to a minimum of 2. We leave the number for aisles open since these have to be determined based on whether we use a horizontal or vertical bay structure. Hence, we have for both the cross-aisles and aisles sequence Z elements. The last part of the data structure indicates which direction the bay structure is. This data structure part only consists of one number K, where $K \in \{0, 1\}$. A 1 corresponds to the horizontal flexible bay structure and a 0 corresponds to the vertical flexible bay structure. Hence, our data structure consists of $(4 \cdot Z)$ elements.

4.3 Visual Data Structure Representation

A visual representation of an example data structure is given in Figure 5.



(a) Horizontal Flexible Bay Structure

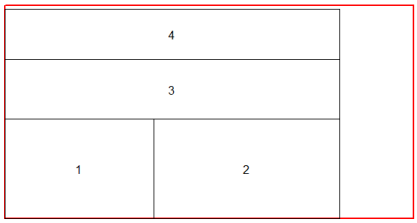


(b) Vertical Flexible Bay Structure

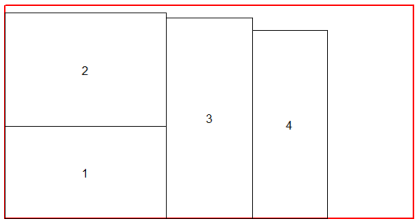
Figure 5: An example of solution representation in the proposed algorithm.

The corresponding graphical warehouse layout is given in 6. The left (right) graph reflects the data structure for the horizontal (vertical) flexible bay structure. For both bay-direction structures, the department sequences follow $(1, 2, 3, 4)$. The bay-break and cross-aisles are (0,1,1), (2,2,2,2), respectively. The sequence for aisles differs due to the bay-break direction difference. For the horizontal and vertical direction the aisle sequence is (12,15,27,27), and (13,13,7,6), respectively. The bay-break sequences indicate which picking areas correspond to which bay.

In the example representation, we find a zero and two ones, which indicates that the breaks are generated after the 2nd and 3rd place. So there are three bays, the first bay contains departments $\{1, 2\}$, the second and third bay only contains one department, department $\{3\}$ and $\{4\}$, respectively. The data structure transformation to a graphical layout starts from bottom to top and left to right. In Figure 6 the red rectangle represents the bounds of the warehouse. From Figure 6 we conclude that there is empty space that could be used to improve the objective function by the metaheuristic. The exact procedure for improving the objective function is explained in Section 5.



(a) Horizontal Flexible Bay Structure



(b) Vertical Flexible Bay Structure

Figure 6: Layout generated from the solution representations in Figure 5

5 Adaptive Large Neighborhood Search (ALNS)

Adaptive large neighborhood search (ALNS) is used as the metaheuristic. ALNS is an extension of large neighborhood search (LNS). LNS was introduced by Shaw (1998) and is used to improve an initial solution by destroying and repairing the current solution multiple times. The ALNS method steers towards a better solution, it is a self-learning metaheuristic that assigns high probabilities to heuristics which have led to a reduction. The ALNS method will search for heuristics that lead to cost reductions and it will exclude heuristics which does not improve the solution. An overview of the ALNS framework is given in Figure 7.

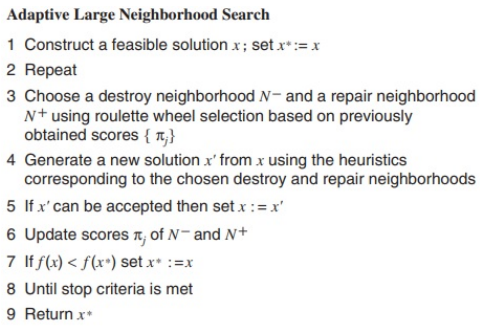


Figure 7: Framework Adaptive large neighborhood search (Pisinger and Ropke 2007)

We follow the ALNS framework description from Sacramento Lechado et al. (2019). First, a feasible solution x is constructed. This solution is set to be the current best solution x^*. Next, we start with the destroy and repair heuristics. Destroy methods eliminate part of the current solution, while repair methods rebuild the partial solution. The heuristics are statistically chosen according to obtained scores which depend on the performance of that particular heuristic. Pisinger and Ropke (2007) state that when using an ALNS framework one should be careful with the degree of destruction. They say that if a too small part of the solution is destroyed, it can be difficult for the ALNS method to escape local minima. On the other hand, if too much of the solution is destroyed, repair heuristics can have difficulties reconstructing a good solution. To overcome this problem, certain randomness is included in the destroy heuristics.

Different destroy and repair methods are used, denoted as Ω^- and Ω^+, respectively. In each iteration we randomly select based on obtained scores a destroy method $d \in \Omega^-$ and a repair method $r \in \Omega^+$. The selection of a heuristic is based on weights, consisting of previously obtained scores $\omega_{i,j}$, assigned to the different methods using the roulette wheel selection. In the beginning, the weights for the roulette wheel selection are all equal and they are updated each iteration concerning a reaction factor $\rho \in [0, 1]$ and the score $\{\pi_j\}$. The scores are defined in Table 3. These two parameters, ρ, and $\pi_{i,j}$, usually are fixed a-priory.

Table 3: Scores of the model.

Parameter π_j	Description
π_1	The new solution resulted in a new global best solution
π_2	The new solution resulted in a new solution which was accepted with a cost better than the cost of the current solution
π_3	The new solution resulted in a new solution which was accepted with a cost worse than the cost of the current solution
π_4	The new solution is rejected

Let w_{ij} be the weight of the method i at iteration j. Therefore, after each iteration, the weights are updated as follows:

$$w_{i,j+1} = \rho w_{ij} + \pi_j(1-\rho). \tag{15}$$

Furthermore, both the destroy and repair methods are assigned different weights. It is necessary to use different distinguished weights because one repair method may adjust more than one destroy method. For instance, there are different repair heuristics for the department sequence and the number of cross-aisles. These repair heuristics can not be used for both sequences, wherefore the probability distributions should be separated between the possible destroy and repair heuristic combinations. Formula (15) is used in order to calculate the weight for the destroy method, while to calculate the weight for the repair methods it is necessary to normalize the scores for each cluster of the repair heuristics. When the sum of the probabilities is greater than one, it is necessary to normalize them using Formula 16.

$$w_{i,j+1} = \frac{w_{i,j+1}}{\sum_{i=1}^{I} w_{i,j+1}}, \tag{16}$$

where I is the number of the destroy or the repair methods. The probabilities are reset to all equal probabilities after some predefined threshold such that the ALNS framework does not steer too much in one direction.

Another critical point mentioned by Pisinger and Ropke (2007) is that we should avoid the algorithm moving randomly through the solution space. The algorithm should be controlled how it should accept solutions that are created in each heuristic. The ALNS framework is therefore extended with the acceptance criteria from Simulated Annealing. The Simulated Annealing method starts with a high-temperature $T_{ST} > 0$ and it is linearly decreased towards zero, following Santini et al. (2018) and Sacramento Lechado et al. (2019). This temperature controls the acceptance probability of a new solution. The initial temperature T_{ST} is calculated as T^*_{ST} times the value of the initial solution. This will allow the algorithm to adjust the temperature according to the size of the instance. To avoid too small temperatures for small instances, the initial temperature is increased by 10% for these instances. If $T = 0$ we return the current best solution. If the new solution x^t obtained with the chosen heuristic has a better objective value than the current solution x, we always accept the new solution x^t. If the new solution x^t is not better than the current solution we accept the solution with probability:

$$e^{\frac{f(x)-f(x^t)}{T}}, \tag{17}$$

where $f(x)$ and $f(x^t)$ denote the objective values of the current and new solution x and x^t, respectively. We randomly generate a number between zero and one, if the generated number is below (17) we accept the solution. A maximum number of iterations (Φ) is used as the stopping criteria for the metaheuristic. Hence, the temperature is linearly decreasing with the number of iterations compared to the maximum number of iterations. We update the temperature using the formula

$$T = T_{st}\left(1 - \frac{t^{cur\_iterations}}{t^{\Phi}}\right).$$

Following Pisinger and Ropke (2007), we also use a feature that returns to the best solution so fair if a certain number of iterations has passed without any improvement. If we get stuck in an local optimum after resetting to the best solution so fair, we restart the whole process, starting with a new initial solution. This restarting process is maximized with a pre-defined threshold. The complete algorithm is executed twice, once with the horizontal and once with the vertical bay structure. The pseudo-code for the ALNS algorithm is given in Algorithm 1. After the ALNS algorithm has run for the horizontal and vertical bay structure we compare the results and store the best one as our final solution. A final feasibility check is executed on the final solution.

Algorithm 1: Pseudo-Code for the ALNS Algorithm

INPUT: Warehouse variables
Max iterations without improvement: Γ
Max iterations: Φ
Number of iterations without improvements: γ_i

$x \leftarrow InitialSolution()$;
$T_{ST} \leftarrow T^*_{ST} \cdot x_{obj}$;
$x^* \leftarrow x$;
$\Gamma \leftarrow 0$;
$\gamma_i \leftarrow 0$;
$current\_iteration \leftarrow 0$;

while $current\_iterations < \Phi$ **do**
  Choose a destroy method $d()$ and a repair method r() from Ω^- and Ω^+;
  $x^t \leftarrow r(d(x))$;
  $T \leftarrow T_{ST}(1 - \frac{current\_iterations}{\Phi})$;

  if $Random(0,1) < \exp(\frac{f(x)-f(x^t)}{T})$ **then**
    $x \leftarrow x^t$;
  end if
  if $f(x) < f(x^*)$ **then**
    $x^* \leftarrow x$;
    $\gamma_i \leftarrow 0$;
  else
    $\gamma_i \leftarrow \gamma_i + 1$;
    if $\gamma_i > \Gamma$ **then**
      $x \leftarrow x^*$;
      $\gamma_i \leftarrow 0$;
    end if
  end if
  Update scores of Ω^- and Ω^+ based on acceptance criteria
  $current\_iterations \leftarrow current\_iterations + 1$
end while
RETURN: x^*;

5.1 Initial Solution

The procedure to find an initial solution is divided into multiple steps. First, generate a random solution for the department and bay break sequence. Set the number of cross-aisles to a minimum of two. Then, store the direction for the bay structure in the fifth part of the data structure. Third, calculate the total area needed for each bay. Using the bay break structure, which tells us which department is placed in each bay, and the needed area for each picking area from the starting information we calculate the total area needed for each bay. Fourth, calculate the number of aisles. We do so by maximizing the depth of the warehouse. We calculate the maximum possible depth with a minimal number of aisles. The maximum depth is divided proportionally with the needed department surface to each picking area. This information is used to calculate the minimum number of aisles needed such that the departments fit within the depth of the warehouse.

Fifth, calculate the dimensions for each department. The dimension can be calculated using the length of the picking aisles and the number of aisles and cross aisles. Sixth, check feasibility, using the dimensions of each department with the fact that the first department starts at the bottom-left corner of the warehouse we calculate the total length and width of all departments to check whether this does not exceed the warehouse dimensions. If the result is not feasible, return to step 1. If the result is feasible, calculate the department's starting points. We know that the starting point of the first department is always $(x = 0, y = 0)$. Using the department dimensions we recursively find the other department starting points. Finally, calculate the objective function and store the initial solution as our current and best solution. The pseudo-code for the initial solution is given in Algorithm 2.

The solution consists of different parts. The first part is the complete data structure. The second part is an output matrix, consisting of the department sequence solution, the total aisles length l_i, the number of aisles n_i, the starting points of the departments, the number of cross-aisles k_i, and the department dimensions. The third part is a matrix with separated objective values for each department, to get insights into how the objective value is spread over the departments. The last part of the solution is the total objective value.

For relatively tight instances we steer the initial solution in a certain direction. Instances are tight whenever the total needed area for all picking areas is close to the total available warehouse surface. After some experiments, we concluded that most instances have solutions where all picking areas fit next to each other. Hence, for the horizontal flexible bay structure, we steer the initial bay structure to a solution where almost all picking areas fit in a single bay. For the vertical flexible bay structure, we try to fit almost every picking area in a single bay. However, to not push the ALNS method in a fixed direction, which could lead to a local optimum, some degree of freedom is added to the initial solution generator. In the normal initial solution generator, the number of bays is completely random, and in the initial solution generator for tight instances, we limit the solution space such that almost all picking areas are placed next to each other.

Algorithm 2: Pseudo-Code for the initial solution

INPUT: Warehouse variables

while *solution* is not feasible **do**
  solution[departments sequence] ← generate a random sequence with size Z;
  solution[bay breaks sequence] ← generate random sequence of 0 and 1 with size $Z-1$;
  solution[cross-aisles sequence] ← set all cross-aisles to 2;
  solution[direction] ← 0: vertical bay structure, 1: horizontal bay structure;
  for each bay **do**
    calculate area of the bay
  end for
  for each department **do**
    solution[aises sequence] ← the number of aisles;
    l_i ← aisle length;
    (x_i, y_i) ← x and y axis of the lower-left corner;
    $(x_{dimension}, y_{dimension})$ ← length of the sides
  end for
  CHECK THE FEASIBILITY
  for each bays **do**
    x_{bay} ← max(x_i in the bay)
    y_{bay} ← max(y_i in the bay)
  end for
  x_{total} ← sum all x_{bay}
  y_{total} ← sum all y_{bay}
  if x_{total} < width & y_{total} < dept **then**
    Solution is feasible
  elseSolution is not feasible
  end if
end while

RETURN
(1) complete data structure
(2) matrix with l_i, n_i, k_i, and starting points
(3) matrix with separated objective values for each departments
(4) total objective value

5.2 Destroy methods

In each iteration, the ALNS algorithm destroys a part of the current solution. Nine methods are defined and briefly described below. It is possible to divide these methods into four groups: destroy departments (four methods), destroy bay (three methods), destroy cross-aisles, and destroy aisles. In each iteration one of nine is selected considering the weights described in Section 5.

5.2.1 Destroy Departments

The first destroy method deletes randomly two departments from the current data structure. The deleted departments are stored and used again in the corresponding repair method. We note that for large instances with for example 50 picking areas removing only two departments could not be efficient. Therefore, the second destroy method generalizes the first one. This method deletes randomly at least (not only) two departments from the current solution and reports the positions of the removed departments.

The third method, instead, removes a random neighborhood of departments. More specifically, it first selects a random position in the department sequence and generates a random number that depicts how many departments before and after the randomly selected position are removed. The numbers of the final selected positions can range from 1 to Z, therefore it is important to control whether all the selected positions belong to this range. The algorithm returns, as in the previous methods, the removed departments.

The last method for the departments' sequence is named "Worst Removal Departments". This method consists of first calculating the objective value for each department and then storing the $X = \min\{Z, 10\}$ most costly departments, where Z is the total number of departments. The heuristic randomly selects n-departments from X and their respective positions are deleted from the current solution. As before, this algorithm returns the positions of the removed departments.

5.2.2 Destroy Bays

The first two heuristic methods for the Bays Breaks sequence are similar to the first two heuristic methods used for the departments sequence. The first method deletes randomly only two positions from the current solution, while the second one deletes randomly at least two positions. Both algorithms return the deleted positions.

The third heuristic method for the destroy bays sequence deletes a neighbor of Bay Breaks. This algorithm works similarly to the third heuristic method for the department sequence. Hence, first, the algorithm selects a random position in the Bay sequence and it decides a random size range. Then, it deletes those positions from the current solution. Also, in this case, the algorithm returns the deleted positions.

5.2.3 Destroy Cross-Aisles

The method for destroying the Cross-Aisles works in two steps. In the first step, a cross-aisle is added to or removed from each department and the objective function is calculated for each case. During this operation, the algorithm controls also whether the obtained solution is feasible or not, saving only the feasible new solutions. In the second step, the $X = min\{Z, 5\}$ solutions that report the lowest value of the objective function are selected. Then randomly the algorithm selects one solution from X. The number of cross-aisles can range from 2 to 10, therefore it is important to control whether it is possible to add or remove cross-aisles remaining in such a range. The algorithm returns the new solution and the position of the changed cross-aisle. The pseudo-code in Algorithm 3 reports this procedure.

Algorithm 3: Destroy Heuristic for Cross-Aisles sequence

INPUT: Solution, num. departments

for num. departments **do**
  if it is possible to add or remove **then**
    Add or remove a cross aisles;
    if the solution is feasible **then**
      New_Sol_Matrix ← (position changed, new Obj., new Solution);
    end if
  end if
end for
Select from New_Sol_Matrix the $X = min\{Z, 5\}$ new solutions that report the lowest objection function;
New Solution ← random choice from the $X = min\{Z, 5\}$ new solutions;
RETURN(New Solution, position changed)

5.2.4 Destroy Aisles

The last method destroys the aisles sequence from the current solution. This procedure is similar to the destroy method for the cross-aisles sequence. Hence, the algorithm adds and removes an aisle to each department. This method also controls if the obtained solution is feasible before selecting a new solution. Then, a changed solution is randomly selected from the best $X = min\{Z, 5\}$ new solutions in terms of the objective function. Similar to the heuristic method for the cross-aisles sequence, it is necessary to control the number of aisles. The possible range of aisles goes from 1 to 30. The algorithm returns the new solution and the changed position.

5.3 Repair Methods

In the repair phase, the ALNS algorithm repairs the solution previously destroyed. As each repair method is matched with its corresponding destroy method, it is possible to divide the repair methods into the same groups of the destroy methods: repair departments (two methods), repair bay breaks (two methods), repair cross-aisles, and repair aisles. As usual, the probabilities defined in Section 5 are assigned for each repair method and in each iteration, these probabilities are updated. Considering that a group of destroy methods can be repaired by only one repair method chosen from a subgroup of repair methods, the probability of a given repair method being chosen matters only when the subgroup is formed by more than one repair method.

For example, if the solution is destroyed by the destroy cross-aisles method, the algorithm can select just one repair heuristic among the possible six methods. However, if the destroy method belongs to the departments' group, the probability of choosing a repair method from the same group plays an important role because it is possible to choose from two repair methods. Therefore, it is necessary to calculate the probabilities for each cluster of repair methods.

5.3.1 Repair Departments

This group of repair heuristics is composed of two methods. The first heuristic method of this group is named Heuristic Random insertion. This algorithm repairs the deleted departments by reinserting them randomly. The algorithm also uses as inputs the deleted positions and the current solution returning a new solution with a new departments sequence. The second heuristic method, instead, repairs the destroyed solution considering the objective value. The most costly are reinserted first, the first position to be filled is by definition the closest to the depot, wherefore the costs for traveling between picking areas and the dock doors are minimized. In this repair method, there is no randomness involved.

5.3.2 Repair Bays

Similarly to the repair departments group, the repair bays group entails the use of two methods that can repair the three destroy methods described in Section 5.2.2. The first one reinserts randomly the deleted bay breaks generating a new solution with a different bay sequence composition. Differently, the second repair method for the bay breaks sequence changes the values of the deleted positions. Due to the definition of the data structure, the allowed values in this sequence can be either 0 or 1. The algorithm checks the values in the deleted position and if such value is 1, the algorithm assigns 0, otherwise, it assigns 1. In this case, the new solution changes the composition of the bay structure.

5.3.3 Repair Cross-Aisles

The repair cross-aisles method adjusts the solutions destroyed by the destroy cross-aisles heuristic. This method can adjust randomly the destroyed position selected by the destroy cross-aisles method through three options: (1) the algorithm adds one cross-aisle, (2) the algorithm removes one cross-aisle, or (3) the algorithm keeps the destroyed position and so does for the current solution. In this way, the repair method can add or remove at least two cross aisles to the position of the current solution. As the destroy method for the cross-aisles sequence, the number of cross-aisles can range from 2 to 10. Therefore, the algorithm checks also if the changed value belongs to such a range. Eventually, it is also important to check whether the obtained solution is feasible or not. If such a solution is not feasible, the previous solution is kept and the algorithm returns the repaired solution. The pseudo-code in the Algorithm 4 describes the steps of this procedure.

Algorithm 4: Repair Heuristic for Cross-Aisles sequence

INPUT: Destroyed Solution, position change

a ← random number (1, 2, 3):
if a = 1 **then**
    New Solution ← Destroyed Solution[*changed_position*] + 1 ;
end if
if a = 2 **then**
    New Solution ← Destroyed Solution[*changed_position*] - 1 ;
end if
if a = 3 **then**
    New Solution ← Destroyed Solution[*changed_position*];
end if
Check feasibility;
if the solution is feasible **then**
    RETURN: (New Solution)
else
    RETURN(Previous Solution)

5.3.4 Repair Aisles

The repair aisles method works similarly to the repair cross-aisles method. This method adjusts the destroyed solution by choosing among three options described in Section 5.3.3. Hence, this algorithm can add or remove at least two aisles to the selected position. Similar to the heuristic method for the cross-aisles sequence, the algorithm checks the number of aisles which must always range from 1 to 30. Furthermore, the algorithm controls whether the solution is feasible or not. If such a number is not feasible, the algorithm returns the previous feasible solution; otherwise, it returns the new solution.

6 Computational Experiments

The ALNS metaheuristic was implemented in R, and run on the "Peregrine" computer from MobaXterm. We have tested our method on a standard benchmark set of 450 instances, the results of which can be found in a separate file. The first 50 instances have 10 picking areas and in every step of 50, we add 5 picking areas to its instances. Hence, instances 51-100 have 15 picking areas, and instances 401-450 have 50 picking areas. Within each group of 50 instances, there are 10 levels of increasing tightness. So in instances 1-5, 51-55, 101-105, etc. the warehouse is quite large compared to the required space for the picking areas. While for instance 46-50, 96-100, 146-150, etc. the total required surface for the picking areas is fairly close to the total surface of the warehouse.

We analyze a small batch of those 450 instances to get insights into the obtained solutions. We pick three instances of each picking area size and we pick those three instances from the top level of tightness. The reason for this choice is that it is most realistic to have tight warehouses since extra space is costly. For example, we pick from the first 50 instances, instances 40, 45, and 50. First, we explain how the parameters used are set up. Next, we compare from these 27 instances the horizontal and vertical solutions and the calculation times of the algorithm. To analyze the different operators, we perform an effectiveness check of a few operators. In the last part of this section we compare different initial solution and final solution to see whether there is a clear relationship.

6.1 Parameters

As explained before, the ALNS algorithm is an iterative procedure. Hence, it is necessary to define some useful parameters which are reported in Table 4.

Υ	maximum number of iterations without improvement before going back to the current best solution within the algorithmic run
γ_i	number of iterations without improvements
Γ	maximum number of iterations without improvements before restarting the whole algorithm
Λ	maximum number of restarts allowed
Φ	maximum number of iterations
Ψ	restart threshold for heuristic probabilities
T^*_{ST}	initial temperature factor

Table 4: Parameters used in the procedure.

These parameters control each iteration and stop the ALNS algorithm at a certain point. During the ALNS iterations, the γ_i, Γ, and Υ play an important role. The first parameter defines the number of iterations without improvements in the objective function and the algorithm upgrades the name of the first parameter for each iteration. The second and third parameter are used as thresholds for the maximum number of permitted iterations that do not report any improvements in the objective function, which is fixed a priori.

If the γ_i reaches the value of Γ, the algorithm recalculates a new initial solution and performs all steps of the procedure again. These parameters are important because they do not allow the algorithm to go always in the same direction. The ALNS method stops when the number of iterations gets to the maximum (Φ) or when the procedure restart for the nth time (Λ). The maximum number of iterations needs to be quite big to let the ALNS algorithm explore as many solutions as possible. The parameters are fixed a priori and we use values computed after testing the ALNS procedure several times for a trial. We concluded from the experiments that for small instances it is not necessary to have many iterations while for large instances it takes more iterations to reach a stabilized result. Therefore, we set $\Phi = Z^2 \cdot 100$, where Z is the number of picking areas in the instance. For calculation time reasons we do not restart large instances as many times as small instances since large instance take longer time to find a final solution. However, for the large instances we have checked, we found that an optimal solution is found within the first few starts. Hence, we set $\Lambda = 50/\sqrt{Z}$, rounded down.

In the experiments we also found that whenever an solution is stuck for a certain amount of iterations is does not find a better solution. We found that this threshold comes close to the total number of departments of an instance. Therefore, we set the non-improvement parameter for going back to the current best solution within the run $\Gamma = Z$. If the solution get stuck after it has already set back to the best solution within that run we restart the whole algorithm after a while. This restart threshold is set to $\Upsilon = Z^2$. The heuristic probabilities are reset to equal weights after a while in a single algorithm run, to overcome the problem that the ALNS algorithm steers to much into a certain direction. Therefore, we set $\Psi = 3 \cdot Z$.

The parameter mentioned for the temperature has been selected based on previous results obtained by Sacramento Lechado et al. (2019) and Shaw (1998), where the initial temperature factor is set to $T^*_{ST} = 0.01$. Furthermore, the remaining algorithm parameters concerning the adaptive part of the metaheuristic were set to the values as documented in Pisinger and Ropke (2007). Therefore, the reaction factor is set to $\rho = 0.9$ and the scores of the methods to $\pi_i = 33, \pi_2 = 9, \pi_3 = 13$ and $\pi_4 = 0$.

6.2 Experiments and Results

In the upcoming section, the performance of the algorithm and the results obtained will be studied. To get insights into the objective value over time Figures 8 and 9 graphically depict the objective value. In Figure 8 and Figure 9 the dynamics of the objective value of instance 40 are reported considering respectively the horizontal bay structure and the vertical bay structure. The horizontal red line represents the obtained minimum value for the objective value. Both dynamics present a decreasing trend from the starting solution and each restart point. Comparing these two dynamics, we see that the vertical bay structure finds a lower objective value than the horizontal bay structure. In the horizontal bay structure, we see that the solution is quite often stuck in a local optimum. The optimal solution for the horizontal bay structure is found after 7 restarts. The solutions for the vertical bay structure are almost always lower than the horizontal bay structure. From Figure 9 we see that the best solution is already found using the first initial solutions. After a while, the vertical bay structure seems to get stuck in local optima.

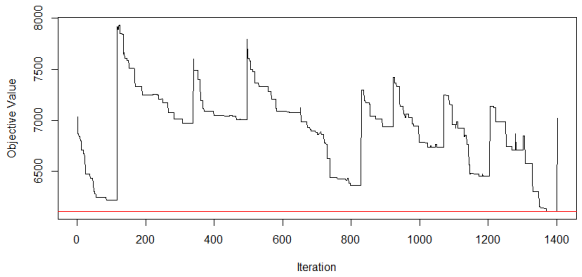


Figure 8: Horizontal bay structure

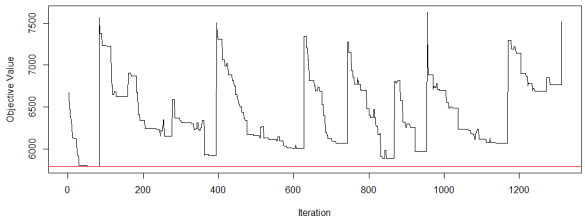


Figure 9: Vertical bay structure

Table 5 reports the final objective value and the calculation times for both the horizontal and vertical bay structures. We conclude that warehouses with a high number of picking areas require more time to explore different solutions. The reason for the increasing calculation time is that some operators require multiple comparisons and multiple computations for the objective value. The calculation time for the vertical bay structure is on average larger than the horizontal bay structure. This is due to the fact that with the vertical direction, the ALNS explores more solutions than with the horizontal bay structure. However, most of the lowest objective values are obtained considering the vertical bay structure. Only instances 295 and 395 report a lower objective value using the horizontal bay structure. In addition, the ALNS algorithm does not find any solution for instances 400 and 440 in the horizontal bay structure. Differently, the ALNS procedure managed to find an objective value for all 27 instances in the vertical bay structure. For the overall 450 instances, the vertical bay structure found for all instances a feasible solution. The horizontal bay structure could not find a feasible solution for 48 instances. For a complete overview of all the objective values and calculation time, one could contact the authors.

Table 5: Experimental results from a small batch of instances, calculation times are in minutes.

	Horizontal		Vertical	
Instance	Objective value	Calculation time	Objective value	Calculation time
40	6187.68	1.64	**5681.93**	3.12
45	2341.86	2.37	**2174.61**	2.72
50	3643.25	1.02	**3489.05**	1.00
90	3790.67	17.81	**3595.22**	23.28
95	6679.89	14.84	**6175.89**	16.71
100	6969.53	4.88	**6607.00**	9.04
140	15542.00	41.90	**15042.93**	33.50
145	9446.03	42.97	**8848.53**	39.68
150	14676.97	14.12	**14262.59**	15.21
190	12417.21	113.02	**11825.56**	118.49
195	14157.78	46.47	**13071.47**	56.32
200	17139.75	34.49	**16040.34**	61.03
240	23315.89	115.74	**20512.38**	180.54
245	**10148.86**	66.84	10282.75	69.86
250	12283.72	56.25	**12210.16**	47.87
290	24062.98	161.06	**23063.09**	280.13
295	24801.86	114.07	**21297.26**	193.74
300	18189.45	94.65	**16546.78**	153.64
340	17457.17	241.99	**17182.83**	274.98
345	28049.73	173.03	**26010.76**	282.96
350	22008.41	133.27	**20370.22**	233.14
390	19116.34	289.51	**18579.48**	400.06
395	**16164.63**	259.14	16312.77	278.76
400	-	-	**27641.95**	320.47
440	-	-	**40223.14**	948.36
445	42775.64	208.92	**39053.09**	431.32
450	28069.84	422.12	**26125.93**	456.96

6.3 Operator Effectiveness

To get insights into the operators we run a small batch of instances again by turning off and on some operators. Since most destroy and repair methods are relatively standard in the literature we only check the effectiveness of the new operators such as the cross-aisle and aisle operator. We concluded in Section 6.2 that almost all the best solutions are obtained from the vertical bay structure. Therefore, we only proceed with the operator effectiveness analysis on the vertical bay structure. The results of this effectiveness check can be found in Table 8, Appendix C. The first two columns correspond to the standard algorithm where all operators are functioning. Columns 4 and 5 represent the results where we have left out the operator that changes the number of aisles. Next, we have left out the operator that changes the number of cross-aisles, results can be found in columns 6 and 7. The last analysis is such that both operators, for aisles and cross-aisles, are turned off, columns 8 and 9.

From Table 8 in Appendix C. it can be noticed that generally, ALNS algorithms without these operators report higher objective values. However, leaving out both the operators for aisles and cross-aisles decreases the calculation times substantially. For example, the calculation time for instance 440 reduced from 948.36 to 25.02 minutes while turning off both operators. From Table 8 we can infer the effect of adding the aisle operator by comparing columns 8 and 9 with columns 6 and 7. From this comparison, we conclude that the objective value does not differ significantly. However, the calculation times are substantially higher when including the aisle operator. Therefore, we conclude that the aisle operator is not very useful. If we compare the results when adding the cross-aisle operator we compare columns 4 and 5 with columns 8 and 9. Here we can see a substantially lower objective value. However, again the calculation times a significantly higher than without the cross-aisle operator. We conclude, using the objective values and calculation times that the cross-aisle operator adds value to the algorithm while the aisle operator does not. The logic behind this result is that without the cross-aisle operator we do not use the empty warehouse surface optimally and therefore it should be included in the algorithm.

6.4 Initial & Final Solution

To see how effective the metaheuristic is we compare several initial solution with their final solution for small instances and give graphical representation of the best final solution. In Table 6 we give the results for instances 40, 45, and 50. We restart the process ten times, in for each of the instances the left column corresponds to the initial solution and the right column to the corresponding found solution. We order the rows on the initial objective value. In each first column of all instances we find the objective value of a randomly generated solution. The solutions in bold indicate the best two solutions out of these ten. From Table 6 we conclude that the metaheuristic does decrease the objective value substantially. However, for some runs the algorithms seems stuck in a local optima wherefore the reduction percentage is only three percent. We do not see a clear pattern of initial solution influence on the final solution. For instance 40, the middle initial solution objective values perform best while for the other instances this varies from the highest and lowest starting objective values.

Table 6: Initial solution comparison to the final found solution

	Instance 40			Instance 45			Instance 50		
#	Initial Sol.	Final Sol.	% Red.	Initial Sol.	Final Sol.	% Red.	Initial Sol.	Final Sol.	% Red.
1	7510.31	6240.14	16.9%	**3447.32**	**2239.95**	**35%**	4712.28	3908.71	21%
2	7306.68	6429.78	12.0%	3447.32	2505.45	27%	4694.50	4479.68	5%
3	7183.94	6272.26	12.7%	3447.32	2540.01	26%	**4632.50**	**3631.83**	**28%**
4	**7175.00**	**5744.59**	**19.9%**	3405.09	2327.72	32%	4536.01	3936.01	15%
5	7174.79	6265.86	12.7%	3386.48	2398.68	29%	4437.30	3797.83	17%
6	**7124.67**	**5740.85**	**19.4%**	3363.27	2494.24	26%	4423.22	3698.024	20%
7	7016.89	6383.89	9.0%	3264.041	2588.045	21%	4395.20	3847.30	14%
8	6807.72	6444.68	5.3%	**3261.16**	**2174.61**	**33%**	4387.19	4083.65	7%
9	6771.08	5939.82	12.3%	3170.97	2464.85	22%	4363.75	4227.00	3%
10	6755.47	5992.56	11.3%	2997.37	2634.02	12%	**4308.48**	**3489.04**	**23%**

The corresponding warehouse layouts with picking area information are given in Figures 10-12 in Appendix B. and Table 7 of the Appendix A. We see from the final warehouse layout that there is still some empty space left. However, it might be the case that adding an extra cross-aisle is not cost-reducing but leads to extra costs. From Table 7 we conclude that for all three instances it is beneficial to use extra cross-aisles different from the standard of two cross-aisles. Therefore, we conclude that the usage of extra cross-aisles leads to a reduction in the objective value up to a certain point since we still have empty space left.

7 Conclusion

The facility warehouse layout problem is solved by minimizing the amount of time spent in collecting the correct packages from the warehouse and delivering them to the customer. As specified before, a warehouse should be organized following an optimal arrangement where different picking areas are allocated in order according to the activities to be performed in them.

We developed a method that combines the ALNS framework with the simulated annealing method in order to explore a wide solution space. Introducing acceptance criteria and obtained scores for the heuristic methods gives the ALNS framework the advantage that it can quickly learn from each iteration which operators are more significant for minimizing the objective value without always exploring the same direction.

We conclude from the results that the algorithm with the horizontal bay structure has on average higher objective values and was not able to find a feasible solution for all instances. The algorithm with the vertical bay structure reports better results in terms of the objective function and in terms of the feasibility of the solution.

Regarding the effectiveness of the heuristic methods, using the destroy and the repair methods for the cross-aisles decreases substantially the objective value. However, it is worth noting that the algorithm takes significantly more time to perform the computation with this operator. The other heuristic method to destroy and repair the number of aisles does not report significant improvements in terms of the objective value. Hence, we may conclude that it might be better to exclude the aisle operator to decrease the calculation times. Concluding from the initial solution sensitivity check we do not see a clear pattern of initial solution influence. We see that the ALNS framework could decrease the objective value with more than 30%. However, the ALNS algorithm could get stuck wherefore it only decreases the objective value with 3%. We find that the usage of extra cross-aisles leads to a reduction in the objective value up to a certain point. Hence, empty space could be used for cost-reduction purposes.

For further research one should only consider the vertical bay structure since this has on average always lower objective values and it saves calculation times. Other further research that could be considered is optimizing the current operators, especially the cross-aisle operator, such that the algorithm becomes more efficient which reduces calculation times. To achieve better results one could think about other destroy and repair methods. It would also be interesting to compare the results of this analysis with different data structures and heuristic operators, for example with the metaheuristic as those reported in the introduction.

References

W-C Chiang and P Kouvelis. An improved tabu search heuristic for solving facility layout design problems. *International Journal of Production Research*, 34(9):2565–2585, 1996.

Leonardo Chwif, Marcos R Pereira Barretto, and Lucas Antonio Moscato. A solution to the facility layout problem using simulated annealing. *Computers in industry*, 36(1-2):125–132, 1998.

Amine Drira, Henri Pierreval, and Sonia Hajri-Gabouj. Facility layout problems: A literature analysis. *IFAC Proceedings Volumes*, 39(3):389–400, 2006.

S.T. Hackman, E.H. Frazelle, P.M. Griffin, S.O. Griffin, and D.A. Vlasta. Benchmarking warehousing and distribution operations: an input-output approach. *Journal of Productivity Analysis*, 16(1):79–100, 2001.

Panagiotis Kouvelis, Abbas A Kurawarwala, and Genaro J Gutierrez. Algorithms for robust single and multiple period layout planning for manufacturing systems. *European journal of operational research*, 63(2):287–303, 1992.

Henri Pierreval, Christophe Caux, JL Paris, and F Viguier. Evolutionary approaches to the design and organization of manufacturing systems. *Computers & Industrial Engineering*, 44(3):339–364, 2003.

David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers Operations Research*, 34: 2403–2435, 01 2007.

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

Meir J Rosenblatt. The dynamics of plant layout. *Management science*, 32(1):76–86, 1986.

David Sacramento Lechado, David Pisinger, and Stefan Ropke. An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones. *Transportation Research Part C Emerging Technologies*, 102:289–315, 03 2019.

Alberto Santini, Stefan Ropke, and Lars Magnus Hvattum. A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. *Journal of Heuristics*, 24, 10 2018.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems*, pages 417–431, 01 1998.

J.A. Tompkins, J.A. White, Y.A. Bozer, and J.M.A. Tanchoco. *Facilities Planning*. John Wiley and Sons, New York, 2003.

Kuan Wong and Komarudin Komarudin. Solving facility layout problems using flexible bay structure representation and ant system algorithm. *Expert Syst. Appl.*, 37:5523–5527, 07 2010.

Appendix

A.

Table 7: Final solution information for instances 40, 45, and 50. x_i and y_i are the starting coordinates of the corresponding picking area. n_i is the number of aisles and k_i is the number of cross-aisles.

	Instance 40				Instance 45				Instance 50			
	Obj. value = 5740.86				Obj. value = 2174.61				Obj. value = 3489.05			
#	x_i	y_i	n_i	k_i	x_i	y_i	n_i	k_i	x_i	y_i	n_i	k_i
1	84	0	3	4	54	0	7	3	30	0	5	4
2	102	0	4	6	68	0	4	4	0	86	10	3
3	72	0	2	2	22	0	7	7	78	73.15	7	4
4	18	101.5	4	4	86	0	15	2	30	68.01	5	2
5	42	101.5	4	4	116	0	7	6	54	84.58	7	4
6	18	0	4	3	54	40.29	7	6	45	0	3	2
7	0	0	3	9	0	0	11	8	0	0	10	2
8	42	0	5	2	76	0	5	6	75	0	1	2
9	84	175.34	3	5	86	25.34	15	4	54	0	7	4
10	42	170.01	5	4	36	0	9	6	78	0	7	4

B.

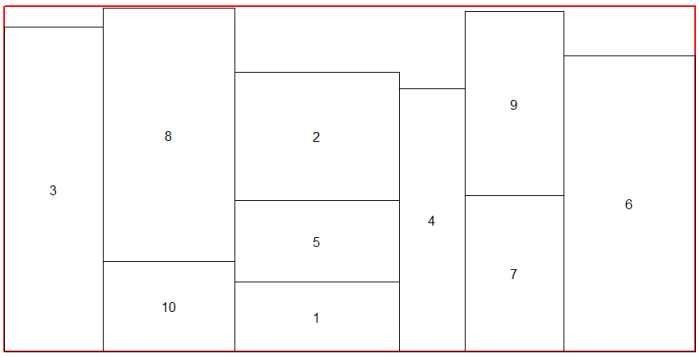


Figure 10: Final solution instance 40

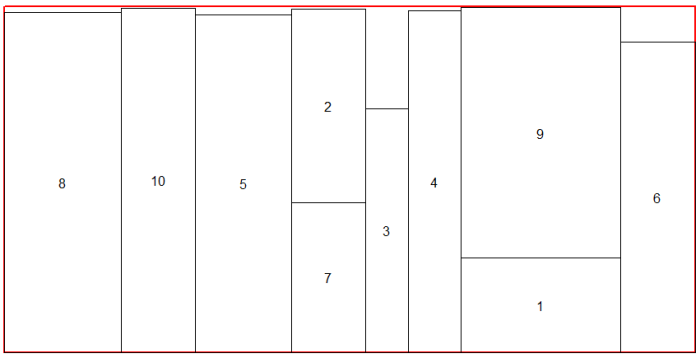


Figure 11: Final solution instance 45

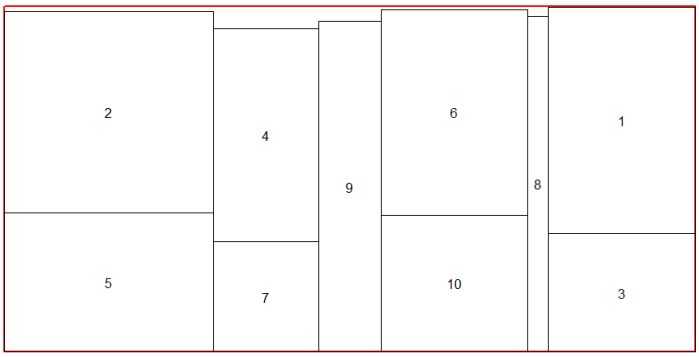


Figure 12: Final solution instance 50

C.

Table 8: Effectiveness check of different operators. Calculation times are in minutes. However, some instances were done in less than a minute, those calculation times are given in seconds, denoted with the s from seconds.

	Best Standard		Without Aisle Operator		Without Cross-Aisle Operator		Without Aisle & Cross-Aisle Operator	
Instance	Objective value	Calculation time	Objective value	Calculation time	Objective value	Calculation time	Objective value	Calculation time
40	5681.93	3.12	6358.07	38s	6640.66	46s	6578.28	13s
45	2283.84	2.72	2595.52	49s	2812.99	1.27	2923.77	10s
50	3643.25	1.02	3705.38	46s	4182.78	24.3s	4284.38	10s
90	3595.22	23.28	3996.93	4.14	4665.17	5.87	5019.24	15s
95	6175.89	16.71	6415.34	6.93	7387.23	3.51	7470.77	31s
100	6607.00	9.04	7181.91	3.18	8007.46	1.47	7969.23	14s
140	15042.93	33.50	15312.17	13.74	16227.07	16.61	17003.65	1.72
145	8848.53	39.68	9544.39	19.79	10978.35	8.76	10919.16	1.12
150	14262.59	15.21	14668.13	7.53	15196.11	5.22	15005.21	1.32
190	11825.56	118.49	12019.74	61.02	14577.87	22.79	14145.24	1.83
195	13071.47	56.32	13800.98	21.53	13898.26	25.49	14569.27	1.99
200	16040.34	61.03	16041.09	37.02	17708.21	30.64	17818.92	2.67
240	20512.38	180.54	22685.97	46.27	23697.51	80.86	24731.22	3.79
245	10148.86	69.86	10899.74	35.97	12344.10	27.76	12095.36	1.29
250	11845.38	47.87	12012.89	30.88	13755.46	21.91	13884.94	1.33
290	22915.72	280.13	23226.72	189.61	25033.93	130.59	25215.21	6.14
295	21297.26	193.74	21568.92	134.16	22629.84	111.28	23445.12	6.29
300	16546.78	153.64	17085.69	87.04	18350.24	101.08	17640.78	5.99
340	16341.41	274.98	17210.01	227.72	19187.65	190.19	20125.66	6.96
345	26010.76	282.96	26455.78	114.62	28121.76	152.41	28625.48	7.29
350	20370.22	233.14	21877.22	146.58	22987.14	151.74	23104.01	7.17
390	18579.48	400.06	19961.88	183.13	21735.72	157.99	22433.77	5.64
395	16164.63	278.76	16536.87	235.18	20077.04	229.08	21264.85	10.25
400	27347.24	320.47	28347.19	166.91	29666.54	71.28	29997.89	4.80
440	40223.14	948.36	40330.06	740.76	42448.33	735.12	43302.14	25.02
445	39053.09	431.32	40143.10	309.65	42225.53	329.59	41822.36	24.87
450	26125.93	456.96	27819.34	390.62	29506.38	384.80	32936.01	21.81