23rd International Conference on Harbor, Maritime and Multimodal Logistic Modeling & Simulation Symposium
18th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0339 ISBN 978-88-85741-59-1 © 2021 The Authors.
doi: 10.46354/i3m.2021.hms.008

Optimizing picking operations in a distribution center of the large-scale retail trade

Eleonora Bottani[1*], Beatrice Franchi[2]

[1]Department of Engineering and Architecture, University of Parma, Viale delle Scienze 181/A, 43124 Parma, Italy
[2] Department of Engineering and Architecture, University of Parma, Viale delle Scienze 181/A, 43124 Parma, Italy

* Corresponding author. Email addresses: eleonora.bottani@unipr.it; beatrice.franchi@unipr.it

Abstract

This paper proposes different scenarios for optimizing picking operations in the distribution center of a large retail chain business case, operating mainly in the fruit and vegetable sector. The chosen approach consists in modifying and combining the routing of pickers and the allocation rules of items, with the aim of decreasing the average distance traveled by the operator during the picking tasks. Four allocation policies have been implemented to this end, namely: 1) *random*, which is the one currently used by the Company (and therefore represents the benchmark scenario); 2) *dedicated*, based on the withdrawal frequency; 3) *dedicated*, based on the quantity of product; 4) a *categorization* of products into classes based on their demand. As regards the pickers' routing, two heuristic algorithms currently used by the Company and three meta-heuristics are compared: Ant colony optimization (ACO), Min-max ant system (MMAS) and Backtrack. The study reveals that the best choice is to use the Backtrack algorithm on orders up to 15 lines, because in these cases the algorithm is very fast and always finds the best possible path. Instead, MMAS is to be preferred in case of larger orders: although finding the optimal path is no longer guaranteed, MMAS has a computational time much shorter (approximately 15 times) compared to the remaining algorithms.

Keywords: large retail chain; picking; routing; item allocation; heuristic algorithms.

1. Introduction

Among the activities carried out in a warehouse, order picking, i.e. the process of items' retrieval from their storage locations, is the one that consumes the most in terms of time and costs: it can consume as much as 60% of all activities in the warehouse and, for a typical warehouse, the cost of order picking is estimated to be as much as 55% of the warehouse's total operational expense (Zhang et al., 2021).

The routing optimization of a picker (i.e. the warehouse operator responsible for picking up the goods) is a special case of the traveling salesman problem (TSP) (Christofides, 1975): the picker, indeed, must visit all the picking locations only once and return to the starting point covering a minimum path. Since there are no generally valid algorithms to find the exact solution of the TSP, heuristic routing policies are often used. Some of the best known heuristic policies focused on routing methods for a single order picker include the S-shape (or traversal), return, composite, midpoint and largest gap heuristics (for an accurate and thorough description see Hall, 1993, Petersen, 1997 and 1999). However, these algorithms do not take into account the congestion problem that occurs when there are several pickers simultaneously in the same region (Gu et al., 2007). In addition to the aforementioned heuristic algorithms, which are conceptually simple and with a low computational load, many companies are implementing picking logics based on meta-heuristic algorithms, which

© 2021 The Authors. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY-NC-ND) license (https://creativecommons.org/licenses/by-nc-nd/4.0/).

often have better results in terms of distance optimization than those analysed above at the cost of more processing power and higher complexity.

The proposed study refers to the context of the large retail chain, which is a centralized distribution logistic apparatus that directly supplies various points of sale of different sizes scattered throughout the territory in which various players act as distributors and wholesalers. Large-scale retail has the typical structure of goods management of a warehouse in which a series of activities are repeated in a recurring way: receipt, storage, picking, consolidation and shipment. The picking phase, as mentioned before, is often the most expensive task in terms of time and costs.

In the light of these considerations, this study aims to propose three different meta-heuristic algorithms (ACO, MMAS and Backtrack) and two heuristic algorithms currently used by the Company (called S-shape and S-shape$^+$) appropriately combined with four allocation policies (i.e. random, dedicated on the basis of the withdrawal frequency, dedicated on the basis of the product quantity, and categorization of products into classes based on their demand), with the ultimate purpose of speeding up the picking process, taking into account the advantages and disadvantages of each change to the current scenario. In particular, the techniques used to optimize the resulting configurations have been adapted to an atypical warehouse configuration, consisting in two adjacent sites connected to each other, each characterized by narrow aisles where the operator cannot change direction. This particular layout configuration reflects that of a distribution center of a large retail chain, operating in the fruit and vegetable sector, and taken as the case study for this paper. For the sake of privacy, the anonymity is respected and the company will be simply called *Company X*.

The optimization simulations have been performed using an original VBA code, developed *ad hoc* for this study and interfaced with Microsoft Excel™. This program was chosen because it is extremely widespread and used in the industrial world.

The remainder of this article is organized as follows. Section 2 presents the state of the art of heuristic and meta-heuristic routing policies adopted in this study. Section 3 presents the methodology adopted, in particular, the allocation policies adopted are shown and the implementation of the S-Shape and S-Shape +, Backtrack, ACO and MMAS algorithms is explained. Section 4 presents the results and discussions of the study. Finally, section 5 reports the conclusions and the possible future developments of the study.

2. State of the art

For the purpose of this study, we will focus on the following meta-heuristic algorithms: Backtrack, ACO and MMAS; they will be compared with two heuristic algorithms currently used by Company X (which are called S-Shape and S-Shape$^+$).

For the literature concerning the S-shape algorithm see Rao (2013), who has developed new analytical models to estimate travel distance for random, full turnover and class-based storage in which a single picker follows a traversal routing policy. Hong and Kim (2016), instead, have presented a route-selecting order batching model with the S-shape routing method in parallel-aisle order picking.

The Backtracking algorithm, instead, consists in the exploration of all the admissible solutions through a decision tree that is created as the choices are made. It is a recursive method, where each node of the search tree is a set of matches. Coming down from the tree, this whole expands. Therefore, the set of matches of the parent node is always a subset of the original set in any child node (Cao et al., 2009). Esteve et al. (2021) found that backtracking obtains a higher accuracy than heuristics, although at a very high computational cost. Yuan et al. (2020) have applied the backtracking algorithm to solve the truck and trailer routing problem in populated and intensive downtown.

The ACO algorithm was developed in 1992 by Dorigo. This algorithm follows the natural behavior of ants. The goal is the search for the optimal path between a starting node (the colony of ants) and a target node (the food), based on the use of probabilistic techniques and meta-heuristic optimization. For the use of the ACO algorithm applied with congestion problems see Chen et al. (2013), in which the authors propose a new routing algorithm based on ACO for two order pickers with congestion consideration. Instead, De Santis et al. (2018) have illustrated a new metaheuristic routing algorithm for the minimization of the travel distance of pickers in manual warehouses based on the combination of the ACO and the Floyd–Warshall (FW) algorithms.

MMAS is an important variant of the ACO algorithm, developed by Stützle and Hoos (1999). In this approach, only the best path of each ant generation is further taken into account; moreover, the pheromone deposit is limited between an upper limit (to avoid that a path becomes too preponderant) and a lower one (so as not to completely remove the possibility of exploring some solutions). This algorithm has proven to achieve better results than the original algorithm formulation, by, in particular, avoiding premature convergence (Stützle & Hoos, 1999). Tang et al. (2013) have applied the MMAS to solve the split-delivery weighted vehicle routing problem. Wan et al. (2005), instead, have used the algorithm for solving the vehicle routing problem with time windows.

3. Methodology

In order to optimize the picking activities at Company X, various scenarios have been reproduced using an original VBA code, developed *ad hoc* for this study and

interfaced with Microsoft Excel™. Each scenario stems from the combination of an allocation policy with a particular routing algorithm, starting from the simplest configuration (S-Shape and S-Shape+), currently used at Company X, up to the more complex ones such as Backtrack, ACO and MMAS. To select the best scenario, 500 orders were extracted as a representative sample from the company's database, with a length of the picking list ranging from a minimum of 5 to a maximum of 50 locations. In particular, the choice fell on lengths equal to 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 order lines. The rationale for this selection is the fact that for orders with less than 5 locations, the routing policies are almost irrelevant, while if the list were longer than 50 locations, from a practical point of view it would need to be processed in more than one trip, resulting in several picking tours. Hence, the results given from simulations would not be reliable because would do not take into account multiple missions. For each of them, all 20 possible combinations between routing algorithm and allocation policy were simulated through the VBA code.

The development environment consists of two main contiguous warehouses, configured with double-sided shelving, where more than 85% of the company's picking activities is performed. The two warehouses are called S2 and S6 (see Figure 1). The first one, which also contains the input/output point of each mission (top left), is made up of 10 aisles for a total of 20 shelves and is crossed transversely by two corridors, which divide the area into three blocks. The second warehouse is almost identical to the first one, with the only difference that it has shorter corridors and contains only 19 rows of shelves. The two storage areas are connected to each other through two openings at the end of aisles 3 and 8. The aisles of both warehouses are not particularly wide (2.5m width), so turning the trolley requires a delicate and slow operation; in fact this maneuver will always be avoided in all proposed solutions, meaning that conceptually, we assume the warehouse to operate with the constraint of narrow aisles.

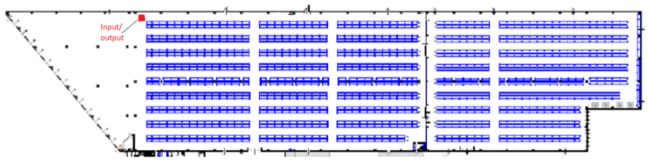

Figure 1. Warehouse S1 and S6 of Company X

3.1. Distance calculation functions and dataset description

Before illustrating the functions that implement the allocation policies and the routing algorithms, a little insight is made on the functions that calculate the travel distance within the program, since these will then be recalled within all the algorithms.

The first step that must be performed for correctly calculating the path is the transformation of the data retrieved from the database. This latter lists the orders as a set of records consisting of 5 fields: week (from 1 to 12), list (indicates the unique order number), warehouse (2 or 6), shelf and finally position (location). The operation to be carried out is given by the transposition of the shelf into the aisle, given that along the same corridor the operator will independently make withdrawals on the right and left, therefore on two different shelves. The only detail to pay attention to is that depending on the warehouse, the shelf could be associated to a different aisle; for example, in warehouse 2 the shelves 18 and 17 overlook corridor 9, while in warehouse 6 on corridor 9 there are shelves 17 and 16. Once the shelf has been associated with the aisle, the VBA functions that calculate the distance can be run. There are 6 main routines, listed below explaining their goal:

- Calculation_distance_warehouse_2_high: as the name suggests, this function has the capability to calculate the distance traveled by the picker if he/she moves between two points belonging to warehouse 2. It is therefore able to evaluate both horizontal movements, i.e. along the corridor, taking into account the possibility of changing the blocks, both vertical ones, so when the operator has to pick up SKUs from shelves distributed in height. The function allows vertical (transverse) movements at the level of cross aisles only. The word "high" indicates the direction in which the operator reaches the place of destination; in particular, in this case the travel direction is opposite to that in which the picker started from the place of departure.
- Calculation_distance_warehouse_6_high: the objective and operation are very similar to those of the previous function, apart from the different length of the longitudinal aisles and the different position of the transverse ones.
- Calculation_distance_warehouse_2_low: this function works similarly to the previous ones, but in the event of an aisle change this does not reverse the direction of travel because it simulates the operator's ability to perform an S-shaped maneuver between the aisles.
- Calculation_distance_warehouse_6_low: similar in purpose and operation to the previous one, but only applied to the size and layout of the warehouse 6.
- Door_passage_2_6_: this function, unlike the previous ones, calculates the distance between two points not belonging to the same warehouse, in particular the first must be found in the one called S2, while the second in S6. This function has the purpose of simulating the movement of the operator, calculating the distance traveled, from the current location to one of the two passages that connect the two warehouses (see Figure 1).

The routine automatically makes the choice of which door to use (the one that minimizes the distance to travel).
- Door_passage_6_2_: the code of this function is similar to that of the previous routine except that in this case the first point belongs to magazine 6, while the second to magazine 2.

3.2. Allocation policies

In this paragraph, the four allocation policies implemented will be illustrated, which are then combined with the various routing algorithms:

- Random: the policy currently in use by Company X.
- Dedicated on the basis of the withdrawal frequency: the products with the highest index are stored in the locations closest to the entry/exit point. This is the frequency index:

  Frequency index = $\frac{\sum_{i=1}^{n} k}{\sum^{n} \sum^{h}}$, with k=single reference, n=total number of orders, h=total number of different references for each order.
- Dedicated on the basis of the product quantity: in this case the order lines do not all have the same "weight", as in the case of Frequency, but each reference requested is evaluated depending on the number of secondary packaging that are picked up. This is the quantity index:

  Quantity index = $\frac{\sum_{i=1}^{n} k * q_k}{\sum^{n} \sum^{h}}$ with k=single reference, n=total number of orders, h=total number of different references for each order, q=quantity ordered for the single item.
- Categorization of products into classes based on demand: class A contains 1268 products, which cover 70% of sales, class B is made up of 1979 products which cover 15% of sales, while the remaining 5% is given by the least requested items and these are 3093.

Figure 2 shows the division into classes of Company X's warehouse.

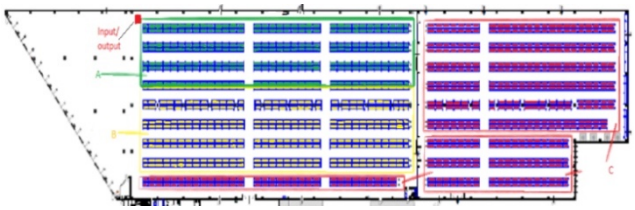

Figure 2. Warehouse scheme with allocation of the product classes.

3.3. Routing algorithms

In this paragraph, all the algorithms that have been implemented will be outlined with an explanation of how the VBA code works.

3.3.1. S-shape and S-shape+ algorithms

The algorithms take as input the positions in which the various items present within the order are stored, the shelf and the location (this type of input will be the same for all algorithms). At first, the algorithm determines which aisles the picker will need to cross (each aisle has two shelves associated); then it will order the items (*sequencing* operation) following the logic of the S-shape algorithm: the picker thus will cross the entire aisle, regardless of how many locations it has to visit, always exiting from the opposite side.

The algorithm always first carries out all the withdrawals inside warehouse 2 and then repeats the same thing for warehouse 6. The function calculates the distance travelled by the operator by counting the amount of different aisles that he/she has to travel and multiplying them by their length; then, it adds the width of the aisles and the shelves crossed in the vertical displacements. When necessary, the program simulates the passage of the employee in warehouse 6 from the nearest door and then its final picks. In this heuristic the various transverse aisles are not used.

The only difference between the S-Shape and S-Shape+ is that the "+" symbol indicates that this algorithm can also make use of cross aisles. Therefore, after reordering the sequence of objects to be picked up, S-Shape+ is able to simulate whether the picker must go all the way down the corridor each time, or if he has the opportunity to turn earlier.

3.3.2. Backtrack algorithm

The formulation of the algorithm is divided into two functions that are called recursively. These are called Tree_rec_Function and Distance_rec_Function. The Tree_rec_Function is the one that builds the path and takes 4 variables as inputs:

- i: indicates at what level of the decision tree we are. An order pick-up point is associated with each level (the lower it is, the closer you are to the root);
- n: the number of points that the picker must visit;
- S: the vector of the choices, at the beginning is empty and at the end it will contain the ordered sequence of the past points;
- V: the Boolean vector of length n (just like S) used to check if a particular point has already been visited.

The function operates through a simple choice: that is, if the leaf level (i=n) is reached, if the condition is true, the Distance_rec_Function is called, while if the criterion is not respected, the algorithm enters in a cycle that will check the first point still available (in which the picker has not already passed); this can be done thanks to the vector of states V. When a valid point is found, the state changes and is added to the

vector *S* in the position corresponding to the level being evaluated. Then, the function is called again by increasing *i* by 1; the procedure is repeated until the aforementioned condition becomes true. This sequence of operations leads to the proposal of a path. At this point the Distance_rec_Function is launched, with the aim of checking if the chosen path is the best and to save it in the event of a positive outcome, it also has the purpose of testing which combination of the two distance functions, linked to the path constructed by the Tree_rec_Function, returns the smallest value. The inputs of this function are the same as those of the previous one, with different values, plus the minimum distance found up to that moment. In this way it will be possible to compare the distance calculated at the current iteration and the best solution found so far.

To better understand the algorithm in Figure 3, an example is presented with three mandatory points plus the origin. The points are called A, B, C. The Tree_rec_Function generates the path starting from point 0 (fixed root) and then it chooses point A as the first level, then B and finally fixes point C as a leaf. At this point the second function (Distance_rec_Function) calculates the path with all possible combinations (2^n) of the two distance functions. At the end of this first path, it tries with the second one, that is, the algorithm takes two steps backwards (it deletes the last two levels and it releases the relating points from those still available) and it tries the lap A-C-B, then it tries again the possible combinations. Then it goes up even further in the tree and it tries B-A-C as the best possible route and so on.

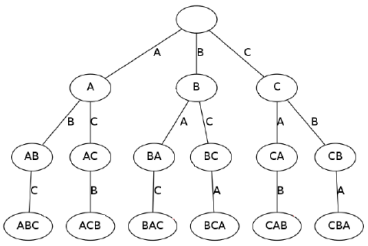

Figure 3. Example of Backtrack decision tree.

To reduce the processing time, pruning rules must be inserted which allow to lower the number of paths to try and consequently the number of calculations to be made. These rules are as follows:

- The first rule is the most classic and intuitive, that is, if at a certain step of a path the length of the previously found minimum complete path has already been exceeded, then it is clear that the solution being explored is not optimal, and therefore, it is discarded. To speed up this process and cut further upstream, a fictitious distance of 10,000 meters has been inserted in the event that the route proposal forces the operator to turn around and go back along the same aisle. This situation, given the constraint of narrow aisles, is always pejorative and therefore never leads to the most efficient path.
- The second pruning is adapted to the case study in question, i.e. if there is at least one pickup point in warehouse 2 (the one closest to the starting/ending point) then the first level of the tree will consist of the points of that warehouse only. This choice is reasonable, as it is always more efficient to go through those locations before changing shelves or eventually on the way back. In both cases the positions of warehouse 2 would be at the boundaries of the path and therefore if they are at the beginning the proposed rule is perfect, while if they are at the end just consider doing the path found in the opposite direction, consequently going along the same path. distance. So this rule avoids trying alternatives without risking to cut the optimal path.
- The third and last pruning is also adapted to the situation taken as an example, that is, if there is at least one point located in the first two aisles of warehouse 2 then that will certainly be the first point of collection. The choice of the first two aisles is given by the fact that they are the only ones that are located between the door closest to the input/output point of warehouse 6 and the point itself, so the operator must always walk through them even if he has to do all the withdrawals only in the warehouse 6.

In the case of orders with a fair length (from 20 onwards), the three heuristics are not effective in keeping the calculation time of the solution below an acceptable limit; hence, the only possibility is to set a timer which stops processing and records the best solution found up to that point. That timer is set at 5 minutes for each order, because this is the maximum time that can be left for each of them to be processed by the software, otherwise, on peak demand days (i.e., Mondays and days close to holidays) the computer system it would not be able to process all orders if they are all medium/large in size.

Since the termination criterion is in terms of time, the improvement of the calculation structure, that is the use of more powerful computers or servers, or a modification of the supporting software architecture - for example writing the same routine in C++ or Python language (processing faster than the current VBA), would allow to perform a greater number of operations in the same time span, thus allowing to try a wider number of paths and consequently finding better and better results. This is supported by the fact that the Backtrack algorithm is strongly influenced by the length and the calculation architecture.

3.3.3. ACO algorithm

The ACO algorithm is commonly applied to a data structure with undirected graphs, that is, the pairs of

nodes are connected by two arcs with opposite directions, easily schematized with a matrix. The three matrices underlying the method are that of cost (in our case the cost reflects the "distance"), that of the pheromone (the parameter "weights" of the various paths) and that of the Delta-pheromone, a supporting matrix that is needed programmatically to properly update the pheromone. Each of them, in the basic case, is squared with size *N x N*, being *N* the number of nodes that the agent (ant) is obliged to pass. In the proposed case study, however, the matrices have size $2N \times 2N$, because, working in a situation of narrow aisles, the direction in which a pickup point is reached affects the movement to the next location. The first step of the algorithm consists in assigning the following parameters:

- alpha (α): weight of the pheromone factor;
- beta (β): weight of the visibility factor ($1/distance$);
- rho (ρ): evaporation factor of the pheromone
- Q: initial amount of pheromone
- m (number of ants): number of independent agents generated at each step
- N (number of nodes): number of points (nodes) where the ant must pass
- iterations: number of iterations of the algorithm.

The second step of the algorithm is its implementation. The algorithm is based on 3 nested cycles: the outermost is the one that is repeated for the chosen number of iterations and it is the one that enhances the number of generations of ants that the user wants to simulate. The middle cycle is the one that explores the experiences of all agents, in fact it is repeated many times equal to the number of ants. In the innermost part we have a set of cycles in series that are used to build the path of the single agent.

Since it is a probabilistic algorithm, at each step of the cycle the probability of moving towards a node or towards another must be calculated. The formula is as follows:

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha}(t) * \varphi_{ij}^{\beta}}{\sum_{l \in J_i^h} \tau_{ij}^{\alpha}(t) * \varphi_{ij}^{\beta}} \quad (1)$$

if $j \in J_i^h$, otherwise 0 if $j \notin J_i^h$, being J_i^h is the list of nodes still to be visited. τ_{ij}^{α} is the value of the pheromone in the *i*-*j* position (i.e. the one deposited on the arc that connects those two particular nodes) with α which is one of the weight parameters initially set up. φ_{ij}^{β} is the inverse value of the distance matrix at position *i*-*j* with β being the weight of the visibility factor. Finally, the superscript *k* indicates the current iteration number.

For each ant of each iteration, the same procedure is followed: the starting point of the ant is inserted in the vector of the past nodes; then, using eq.1, the probability for an ant to choose the next node is determined; the selected node is removed from the vector of the available nodes and inserted in that of the past nodes. The allocation is repeated until all nodes have been visited. The last node will be the initial node again. In this way the path of the ant is created and recorded in the vector of the past nodes, from which the total distance travelled can be easily calculated. After each ant's tour, it is necessary to check whether the distance travelled is shorter than the current minimum distance: if this is the case, the route just calculated will become the new "best route". Besides updating the best route, the Delta-pheromone matrix is also updated applying this criterion:

$$\Delta\tau_{ij}^{k} = \frac{1}{L_h^k} \quad (2)$$

if *i*-*j* $\in T^h$, otherwise 0. T^h is the turn made by the *h*-th ant in the *k*-th iteration. The pheromone only tracks where the ant has passed. At the end of each iteration the Delta-pheromone matrix will be filled with all the deposits of the various ants; if more ants have passed on some arches, they will have a greater weight and higher potential to be chosen by new ants in the future. Before moving on to the next iteration, the "knowledge" must be transmitted to the next generation of solutions and this is done by updating the complete pheromone matrix with the formula in eq.3:

$$\tau_{ij}(t+1) = \rho * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \quad (3)$$

According to eq.3, for each arc of the graph a part of the pre-existing pheromone evaporates, while everything that the various ants (from 1 to *m*) have deposited on the arc in this iteration is added. At this point there will be new agents with improved knowledge compared to the previous ones, who will do the same procedure again.

As described earlier in the manuscript, the context of this study does not reflect a classic case; on the contrary, the tables are double in size, because ants can move from one node to another by means of two arcs with different lengths. This peculiarity leads to the implementation of a vector of directions within the algorithm; that vector at each step will record the direction from which direction the previous node has been left, thus allowing to keep track of the arc used and to calculate the probability correctly.

The ACO algorithm can be used with different configurations of its input parameters. After experimenting with some combinations, the choice

felt on the calculation of the distance and the comparison with the other algorithms with these settings:

- N-ants: 5;
- N-iterations: 1000 (the choice is mandatory due to the scarce computational power available);
- ρ: 0.5 (an intermediate evaporation rate is suitable for the application);
- α: 1 (the pheromone factor must have a greater weight than the distance factor);
- β: 3 (the number is greater because if the exponent of a power that has a base number between 0 and 1 is increased, the result is smaller and smaller);
- Q: 1 (this is an arbitrary value).

3.3.4. MMAS algorithm

The key difference between the classic ACO and the MMAS lies in the different updating procedure for the Delta pheromone matrix and consequently, of the complete pheromone amount. In addition, MMAS makes use of an additional parameter, which can be defined quite arbitrarily, called P_{dec}. This latter is used to calculate the maximum and minimum - exactly as the name of the algorithm itself suggests - of the pheromone that can be found on an arc. The new "release" system makes the algorithm more competitive; indeed, only the pheromone deposited by the ant that made the best path will be passed to the next generation and this allows for the new agents to be directed more easily towards the shortest tour. This feature also increases the probability that the global optimum will be found, but at the same time has the risk of converging quickly and finding a local minimum. To avoid this situation, the maximum and minimum parameters are entered:

$$\text{Maximum: } \tau_{max} = \left(\frac{1}{1-\rho}\right) * \frac{1}{L\_best} \quad (4)$$

where L_best is the shortest distance found up to that point

$$\text{Minimum: } \tau_{min} = \tau_{max} * \frac{1-Pdec}{\left(\frac{n}{2}-1\right)*Pdec} \quad (5)$$

The maximum pheromone amount does not allow to have a too high quantity of pheromone on a single path, while the minimum amount keeps more possibilities open so that the exploration of the solution space continues even after a certain number of iterations. All other parameters have been initialized to the same value as those of the traditional ACO (see the previous subsection). In most cases – as the algorithm is probabilistic in nature - MMAS leads to better or equal solutions to the ACO, but with a shorter computation time.

4. Results and Discussions

The main findings are represented by results grouped by routing criteria and results grouped by allocation policy.

The first type of results, that is the results grouped by routing criterion, analyzes the outcome of the simulations keeping the routing algorithm fixed and varying the allocation policy. The following pairs are compared:

- Random and Frequency: the dedicated allocation on the basis of the withdrawal frequency, that is the best in the case of picking on order in the aspect of optimizing the route, against the Random policy, the best for traffic management and for the saturation of the compartments.
- Random and Classes: the categorization of products into classes based on demand is the hybridization between Dedicated and Random, therefore rather similar to the latter and easily implemented starting from the same; for this reason it is very interesting to see how much a small change has an impact on the distance without particularly worsening the saturation.
- Classes and Frequency: very similar to the previous point, that is the comparison between hybrid management and the reference policy on the opposite side to Random.
- Quantity and Frequency: this comparison serves to highlight how large the individual orders are in terms of quantity, that is, if the difference were significant then it would also be necessary to consider the capacity of the picker's cart for the single journey, while if the difference is small then the hypothesis of infinite capacity in the range of orders selected by us can be considered acceptable.

The second type of results, that is the results grouped by allocation policy, analyses the outcome of the simulations keeping the allocation policy fixed and varying the routing algorithm. The following pairs are compared:

- S-shape vs S-shape$^+$: this comparison is necessary to give an idea of how the exploitation of all warehouse spaces is fundamental and how even a slightly more complex logic can already bring substantial improvements.
- S-shape$^+$ vs ACO: this comparison allows to appreciate the very different performances that heuristic algorithms and meta-heuristic algorithms can achieve. The only aspect that should be emphasized is the difference in

processing time: an order of any size with the S-Shape+ algorithm is handled in 1-1.5 seconds, while, depending on the length, with the ACO the time processing varies from 15 to 25 seconds.
- ACO vs MMAS: the results obtained are very similar, while the processing time, depending on the number of picking points, ranges from 12 to 20 seconds in the MMAS, therefore slightly faster than in the ACO.
- ACO vs Backtrack: most significant comparison among all those proposed. The processing time of the Backtrack is 5-6 seconds for small orders, then for those of 15 locations it is about 1 minute, while for those of even more the limiter must intervene at 5 minutes. This means that for small orders (less than 16 order lines), which are 49.7% of the total requests, it is very efficient and effective, while for large orders it is inefficient because, to have a result comparable with the 'ACO, it is necessary to wait at least 12 times longer.
- S-shape+ vs Backtrack: this comparison is interesting because the Backtrack uses the S-shape+ as its base path and then improves it, so the difference indicates exactly the optimization made.

The results obtained show that for small/medium-sized orders (up to 15 locations) the Backtrack algorithm is the best choice among all the algorithms at equal allocation policy.

This result was predictable because the proposed routing logic returns the optimal path, but as a limit, it has a timer set at 5 minutes, which is the maximum time allowed for processing one single order. Due to this constraint, if the size of the order is greater than 15 pickup points, the algorithm does not have enough time to try all the routes and consequently, it could not be able to find the global minimum, which instead is always detected in the case of a small number of references processed.

Table 1 and related Figure 4 illustrate the average distance traveled during a picking mission using the Backtrack routing algorithm in combination with the four allocation policies.

Table 1. Table of percentage differences between pairs of combinations of routing algorithms and allocation policies.

Lenght	Random	Frequency	Quantity	Class
5	435,3	354,98	340,3	375,28
10	602,64	433,8	439,14	514,24
15	764,42	465,16	518,6	596,14
20	811,16	615,54	605,9	704,36
25	946,9	696	710,92	826,94
30	1120,58	814,88	850,78	944,86
35	1221,38	913,06	978,1	1063,86
40	1295,82	965,44	1014,1	1097,2
45	1378,94	1025,52	1085,7	1152,9
50	1447,24	988,86	1050,24	1166,76

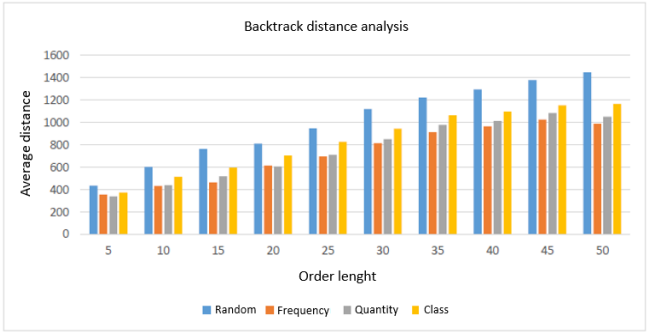

Figure 4. Simulated average distances with different allocation policies combined with the Backtrack algorithm

The graph shows a higher slope in its first part, especially for orders ranking from 15 to 20 items, where the columns relating to dedicated allocations undergo a sudden increase. The most plausible explanation is that when working with up to 15 lines per order the optimal path is actually found, while beyond that order size, the routine is interrupted by the timer without reaching the optimal solution nor exploring all possible picking tours.

The outcomes also indicate that, besides the threshold just mentioned, the differences between MMAS and Backtrack algorithms are almost null in terms of distance optimization; nonetheless, the processing time of the second algorithm is approximately 15 times higher than that of the first one. Therefore, backtrack is not recommended for a practical usage, as routing decisions need to be made in a very short time (hopefully even in real time if needed).

A further interesting point offered by the result of the simulations is the opportunity to evaluate and compare the impacts given by a modification of the allocation logic with respect to the application of a new routing system. The comparison highlights that the usage of an intelligent algorithm, such as the Backtrack, instead of a heuristic criterion can bring greater benefits, in the case of a few order lines, or equal benefits to a change in the goods positioning policy. This means that a relatively easy modification of the WMS, which does not impact on other warehouse activities to a particular extent and does not generate any significant disadvantage, has the same effect as a radical reorganization which, in the case of a dedicated logic, has as intrinsic criticalities a non-optimal saturation of the storage locations and a greater probability of congestion in the aisles close to the input/output point, which is an even more important problem if the aisles are narrow as in the business case analysed.

5. Conclusions

The paper highlights on the one hand, the possibility of adapting some mathematical algorithms, such as

ACO, MMAS and Backtrack, for searching the optimal path of pickers to an atypical real case, consisting of two connected warehouses. On the other hand, it also shows how these algorithms, combined with an appropriate allocation policy (possibly different from the random one), are able to improve the system performance up to 35%.

By analyzing the outcomes obtained from all the solutions, it is evident that the application of an intelligent algorithm is always a good choice, regardless of the storage assignment policy: no matter the size of the order, there is an average saving of 10% in the travel distance, with peaks up to 20%.

To maximize the effect of meta-heuristic algorithms, the best choice would be to use the Backtrack algorithm on orders of up to 15 lines, because in these cases the algorithm is fast and always able to find the best possible path. Instead, it is preferable to use the MMAS method with larger orders: although the optimal path is no longer guaranteed, the algorithm returns a solution quality on average similar to the recursive algorithm, with a resolution time 15 times shorter.

The study also highlights the quantitative improvements of a new type of organization of goods. Starting from the assumption that an advanced routing criterion is used, moving from the current random policy to a dedicated policy can lead to an improvement between 20% and 40% in terms of reduction of the path itself. If switching to a class-based policy, the positive difference can be between 13% and 20%.

The combination with the best performance is undoubtedly the frequency allocation coupled with MMAS and Backtrack algorithms. However, a company operating in the large scale retail trade typically has a wide range of products in stock and it is always possible that a certain item is not available for a time period (e.g. a seasonal item). Hence, it would be preferable to use a subdivision into classes similar to the one proposed in this paper, which allows maintaining an almost optimal and homogeneous saturation of the storage locations. The last aspect to take into account is the narrow configuration of the aisles, in which two operators can move in the same aisle, but "overtaking" maneuvers are not possible. Therefore, the congestion in the area near the input/output depot, typical of the dedicated allocation, becomes a disadvantage relevant in the proposed business case.

In conclusion, for the proposed case, the best scenario consists in a class-based allocation policy with the combination of Backtrack/MMAS routing algorithms; that combination allows to reduce the picking process time by an average of 15% compared to the random allocation policy, without incurring the problems of saturation and congestion typical of the management of dedicated storage locations.

A possible future improvement of the presented study could be the writing of the code in a faster computable programming language like C++ or Python. This rewriting would give the opportunity to increase the size of orders that can be handled with the Backtrack algorithm, maintaining the same time limiter, and consequently to find the optimal routes even with a greater number of pickup points.

A further possible development in the field of allocation policy is the creation of a calculation system that cyclically analyzes the orders processed and autonomously proposes the new subdivision of products into classes. In particular, this specification is important in the Food and Beverage sector because seasonality and anniversaries, such as Christmas, could lead to a very strong variation in both the volume of orders and the type of products requested.

References

Cao, Y. & Jiang, T. & Girke, T. (2008). A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics* (Oxford, England), 24. i366-74.

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

Christofides, N. (1975). Graph Theory: An Algorithmic Approach. *Academic Press*, London.

De Santis, R. & Montanari, R. & Vignali, G. & Bottani, E. (2018). An adapted ant colony optimization algorithm for the minimization of the travel distance of pickers in manual warehouses. *European Journal of Operational Research*, 267. 120-137.

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms. *PhD thesis, Department of Electronics, Politecnico di Milano*.

Esteve, M. & Rodriguez-Sala, J. & Lopez-Espin, J. & Aparicio, J. (2021). Heuristic and Backtracking Algorithms for Improving the Performance of Efficiency Analysis Trees. *IEEE Access*.

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

Hall, R.W. (1993). Distance approximations for routeing manual pickers in a warehouse. *IIE Transactions*, 25.76-87.

Hong, S. & Kim, Y. (2016). A route-selecting order batching model with the S-shape routes in a parallel-aisle order picking system. *European Journal of Operational Research*, 257.

Petersen, C. (1999). The impact of routing and storage policies on warehouse efficiency. *International*

Journal of Operations & Production Management, 19. 1053-1064.

Petersen, Charles. (1997). An evaluation of order picking routing policies. *International Journal of Operations & Production Management*, 17. 1098

Rao, S. (2013). Class based storage for the exact S-Shaped traversal policy in low-level picker-to-part systems. *International Journal of Production Research*, 51. 4976-4996.

Stützle, T. & Hoos, H. (1999). MAX-MIN ant system. 16.

Tang, J. & Ma, Y. & Guan, J. & Yan, C. (2013). A Max–Min Ant System for the split delivery weighted vehicle routing problem. *Expert Systems with Applications*, 40. 7468-7477.

Wan, X. & Lin, J.-L & Yang, X.-W. (2005). Improved MMAS for vehicle routing problem with time window. *Computer Integrated Manufacturing Systems*, CIMS, 11. 572-576.

Yuan, S. & Fu, J. & Cui, F. & Zhang, X. (2020). Truck and Trailer Routing Problem Solving by a Backtracking Search Algorithm. *Journal of Systems Science and Information*, 8, 253-272.

Zhang, J., Zhang, Y., & Zhang, X. (2021). The study of joint order batching and picker routing problem with food and nonfood category constraint in online-to-offline grocery store. *International Transactions in Operational Research*, 28(5), 2440-2463.