mathematics

MDPI

Article

Enhancing Manual Order Picking through a New Metaheuristic, Based on Particle Swarm Optimization

Massimo Bertolini [1,*], Davide Mezzogori [1] and Francesco Zammori [2]

[1] "Enzo Ferrari" Engineering Department, University of Modena and Reggio Emilia, Via Università 4, 41121 Modena, Italy; davide.mezzogori@unimore.it
[2] Department of Engineering and Architecture, University of Parma, Parco Area delle Scienze 181/A, 43124 Parma, Italy; francesco.zammori@unipr.it
* Correspondence: massimo.bertolini@unimore.it

Abstract: This paper proposes a new metaheuristic algorithm called Particle Swarm-based picking time minimization (Pkt_PSO), ideated for picking time minimization in manual warehouses. As the name suggests, Pkt_PSO is inspired by Particle Swarm Optimization (PSO), and it is specifically designed to minimize the picking time in order case picking contexts. To assess the quality and the robustness of Pkt_PSO, it is compared to five alternative algorithms used as benchmarks. The comparisons are made in nine different scenarios obtained by changing the layout of the warehouse and the length of the picking list. The results of the analysis show that Pkt_PSO has a slower convergence rate and suffers less of early stagnation in local minima; this ensures a more extensive and accurate exploration of the solution space. In fact, the solutions provided by Pkt_PSO are always better (or at least comparable) to the ones found by the benchmarks, both in terms of quality (closeness to the overall best) and reliability (frequency with which the best solution is found). Clearly, as more solutions are explored, the computational time of Pkt_PSO is longer, but it remains compatible with the operational needs of most practical applications.

Keywords: order picking; picker-to-goods; Particle Swarm Optimization; metaheuristics; logistics

MSC: 90C27

check for updates

Citation: Bertolini, M.; Mezzogori, D.; Zammori, F. Enhancing Manual Order Picking through a New Metaheuristic, Based on Particle Swarm Optimization. *Mathematics* **2023**, *11*, 3077. https://doi.org/10.3390/math11143077

Academic Editor: Alfonso Mateos Caballero

Received: 24 March 2023
Revised: 29 June 2023
Accepted: 1 July 2023
Published: 12 July 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Warehousing is of paramount importance in supply chains and logistics, as a well-designed warehousing policy leads to benefits within the company and across the entire supply chain [1,2]. Most of the time, high operating performance is achieved using highly efficient Automated Storage and Retrieval Systems (AS/RS) [3,4]. However, in case of small-item storage and/or spare-parts handling, manual warehouses are still more efficient and represent the standard solution [5,6]. In this case, order picking (the problem of collecting a set of items in a warehouse) is an essential issue of order fulfilment, with an immediate impact on customer satisfaction, business reputation and profitability. Picking is generally a major bottleneck in logistics and one of the most expensive and labour-intensive logistic activities, which is why improving this activity is considered a top priority for many companies [7].

When the main operations concern pallet decomposition, short-period storage, pallet recomposition, and shrink wrapping of new pallets, a picker-to-goods strategy is generally adopted [8,9]. Accordingly, the picker moves in the warehouse to collect the required number of cases of each Stock Keeping Unit (SKU) included in the picking list. As the travelling time represents the largest part of the time required to complete an order, many picking strategies have been proposed to reduce the distance covered by the picker. Among them, batch picking, zone picking, wave picking, voice picking and vision picking are certainly the most studied [10,11]. However, all the above-mentioned strategies need

specific operating conditions and/or costly equipment. Batch and wave picking assume that items can be sorted after retrieval [12], zone picking has proven to be efficient only if a very high number of SKUs are stored in the warehouse, while vision and voice picking require a computerized control system and expensive augmented reality glasses or Bluetooth headphones [13].

Due to these reasons, the basic picker to goods strategy, is still the best choice in many industrial contexts [14] and it is the most widespread policy in the context of small and spare-parts handling [15]. According to this policy, the picker is in charge of completing an entire order and, to do so, he/she can move freely in each area of the warehouse to collect the cases (of the SKUs indicated in the picking list) needed to form a pallet. Its implementation is simple and cheap, as it does not require any expensive equipment or additional operations such as batch composition or sorting. Operationally, the only decision concerns the routing that the picker must follow, that is, the order in which storage locations must be visited to complete the picking list. This sequence, in fact, determines the distance covered by the picker and the total travelling time. Optimizing the routing assigned to the picker is, thus, of paramount importance to improve productivity and to cut costs. Not surprisingly, literature in this field is rather extensive, and many interesting and efficient metaheuristics have already been introduced to minimize the total travelling time.

To the authors' knowledge, the last and most efficient framework is the one proposed by De Santis et al. [9], which is also characterized by extraordinary speed of convergence. Although this feature is desirable, it also suggests that most of the provided solutions could be points of local minimum. Therefore, we believe that there is still room for improvement, and to test this hypothesis we propose a novel metaheuristic called Pkt_PSO. As the name suggests (where Pkt stands for picking time), Pkt_PSO is an evolutionary metaheuristic inspired by the well-known Particle Swarm Optimization (PSO). However, to exploit the peculiar structure of the considered problem, many features of the standard PSO have been readapted and modified. Making major modifications to the PSO is indeed a common practice, and it is generally recommended to exploit the potentialities of this approach (see for example [16,17]) This readaptation is, therefore, the main contribution of the paper. The paper is also novel because, to the best of our knowledge, PSO has never been applied to the order-picking domain, although some recent works have shown its effectiveness and flexibility in solving very difficult-to-tackle tasks, such as Bayesian network training [18,19] and different instances of the Traveling Salesman Problem (TSP) [20,21]. The latter is a classic operating research problem very similar to the one herein considered. Assessing the operational benefits that can be achieved using a PSO-based framework is the second main contribution of the paper.

The rest of the paper is organized as follows. Section 2 presents a brief introduction on the state-of-the-art approaches for travelling time minimization in manual warehouses. In the same section, literature gaps are highlighted and the research questions addressed in this paper are clearly defined. For the sake of clarity, Section 3 pinpoints the main features of the standard PSO and Section 4 formalizes the problem and its possible formulation as a TSP. The proposed algorithm (Pkt_PSO) is introduced in Section 5, where all the new procedures and operators developed to exploit the peculiar structure of the considered problem are clearly explained. In view of an experimental section used to validate Pkt_PSO, its operating parameters are fine tuned in Section 6. Next, in Section 7 its performance is tested in nine different scenarios (obtained changing the layout of the warehouse and the length of the picking list) and compared to that of five benchmarks. One of the benchmarks [9] is a top-of-the-class metaheuristic specifically designed for the picking problem in manual warehouses, two [22,23] are outstanding PSO-based approaches for the TSP, while the last two are the well-known Full Depth 2-Opt and the nearest neighbour constructive procedure. Finally, conclusions and directions for future works are drawn in Section 8.

For the sake of clarity, all the abbreviations used in the paper are listed at the end of the paper in the "Abbreviation Section".

2. Picker-to-Goods Strategy, a Brief Introduction and Literature Review

2.1. The Basic Picker-to-Goods Strategy

The functioning of the basic picker-to-goods strategy is very simple. Each picker receives one picking list at a time and he or she must collect all the SKUs specified therein. Specifically, the picking list includes all the SKUs needed to form an entire pallet (as required by the end customer) and indicates, for each one of them, the number of cases to be picked and their exact locations in the warehouse. The picker, generally onboard of an electric order picker, is free to move in all the aisles of the warehouse and collects cases stored at the ground level. Top racks, in fact, are used to store full pallets that will be lowered to the ground (with forklifts) whenever necessary.

The implementation of this strategy is straightforward: given a set of storage locations, the only decision concerns the order in which they must be visited. This sequence, in fact, determines the routing followed by the picker, which is the only element that can be optimized to increase productivity. Whereas the travelling time highly depends on the routing, all the other times (e.g., pick up time of a case, loading and unloading of the order picker, etc.) are constant and, therefore, irrelevant. It is, therefore, an NP-hard combinatorial problem belonging to the family of the Travelling Salesman Problem (TSP) [24,25]. Even in case of order picking with additional constraints, such as one-way aisles [26], limited handling capacity of the pickers [27], partial overlapping of picking and refilling operations, narrow aisles that can be crossed by a single picker at most [28] or allocation of the same items in multiple locations [29], the task can be reduced to a routing optimization problem with the objective of makespan minimization.

2.2. Solution Approaches Proposed in the Literature to Improve Productivity

Solutions proposed in the literature are manyfold and very differentiated but, as for many other operational research problems, the solution approaches can be classified into two broad categories, namely, exact and heuristic methods. Due to the NP-hard nature of the problem, exact methods exist only for specific or simple warehouse configurations. Just to give an idea, the exact model proposed by Ratliff et al. [30], probably the first one published on the subject, is tailored to a one-block warehouse with 50 aisles and a picking list of 10 locations. Although the problem had such as small size, some minutes of CPU time were needed to find the global optimal solution on a mainframe computer of that time. Another good example of exact models can be found in [31], where two models (based on mixed-integer programming and dynamic programming, respectively), are used to solve the order-picking problem for a two or three cross-aisles warehouse.

Anyhow, even using modern processors, for most real applications the computational time needed to execute an exhaustive approach is too high to be of any practical help. Hence, over the years, exact methods have gradually been ignored, to be replaced by heuristics and metaheuristics capable of providing very good and feasible solutions (or *pseudo-optimal solutions*) in a reasonably acceptable computational time. A clear discussion about the superiority of heuristic methods over exact ones can be found in [32], where it is shown that heuristics are more flexible and versatile and that can be easily readapted to different layouts and/or to incorporate additional objectives and constraints. In virtue of this, to give an idea of the different trends in heuristics and metaheuristics for the picking problem in manual warehouse, a selected set of interesting algorithms is presented next. For a more comprehensive review, the interested readers are referred to the work by Masae et al. [33].

Some authors studied the effect of different layouts on routing optimization, as in [34], where three heuristics for optimal routing in warehouses with more than two cross aisles are compared, and in [35], where a fishbone layout is considered. In addition to different layouts, in [36], economic and ergonomics issues are also considered as further objectives. More recently, heuristics have been integrated with discrete event simulations, as in [37], that introduce an algorithm for dynamic order-picking problems, in which the picker might receive instructions for new picking requests while other ones are still in progress. A con-

current simulation-based design of experiments is used in [38] to optimize picking activities and to assess the effect of a combination of operating parameters (layout, throughput, size, order pickers, etc.) on overall performance.

Most of these studies assume a random storage assignment policy, but some authors have also considered dedicated assignment policies. Among them, Caron et al. [39] evaluated two simple routing heuristics in a manual single cross-aisle warehouse, with an allocation strategy based on the Cube per Order Index. Petersen [40] carried out a detailed analysis of variance to examine possible interdependencies among routing policies, warehouse layout and depot locations under different operating conditions. The same author also addressed the issue of optimizing the routing strategy in a class-based storage system [41]. In [42], the correlation between the number of aisles, picking list size, and path length in a class-based storage environment is studied, and in [43], the optimal size of ABC zone is determined for different layouts.

In conclusion, the present work fits into this line of research, as it explores the application of metaheuristics to solve the order-picking problem. The reader who is interested in a complete review of the literature with gap analysis and future research directions can refer to over 70 papers published in Scopus; the last two [44,45] published during 2023 are recommended.

2.3. Gap Analysis and Research Questions

As discussed, the use of metaheuristics allows for the extension of non-exact applications that are operationally more attractive. Indeed, pushing the application to more realistic contexts, the TSP that models an order-picking problem becomes intractable in exact terms. As mentioned above, the existing literature on the subject is rather extensive and many interesting and efficient metaheuristics have already been introduced. To the authors' knowledge, the last and most efficient framework is the one proposed by De Santis et al. [9], which exploits:

- the famous Floyd–Warshall algorithm to solve the all-pair shortest path problem [46,47],
- the Ant Colony Optimization (ACO) to provide an effective solution in a very short time.

Obtained solutions have proved to be superior, or at least equal, to those provided by previous algorithms. However, its extraordinary speed of convergence suggests that most of the times the obtained solution could be a point of local minimum. Therefore, we believe that there is still room for improvement, and to test this hypothesis, we investigate the use of a new metaheuristics inspired by the classic PSO. The rational of this choice can be traced to the fact that, rather recently, some works have demonstrated their good applicability to the TSP (see among others [20,21]). Hence, our main goal is to investigate possible advantages and/or disadvantages of the PSO when applied to this specific instance of the TSP and, most importantly, to modify its underlying structure to exploit the specific structure of the problem at hand.

We make clear from the outset that high-performance improvements are unlikely to be achieved, mainly because of the very constrained structure of the TSP underlying a warehouse organized in blocks and divided by aisles, such as the one considered in this study. Because of this binding layout structure and considering that many SKUs could be found in the same aisle, it is reasonable to expect the existence of many solutions very close in terms of makespan (i.e., point of local minimum). Testing this hypothesis, and thus understanding how sensible it is to further explore optimization techniques, is the second objective of this work.

3. Some Background on Particle Swarm Optimization

PSO is a famous metaheuristic optimization technique inspired by flocks of birds. Its ideation dates to 1995 when Eberhart and Kennedy proposed PSO as an optimization technique for continuous functions [48]. Since then, as stated in the notable works by Banks et al. [49,50] and in the comprehensive literature review by Jain et al. [51], PSO has been widely applied in many fields of operational research both for continuous and

discrete problems and in closed-loop supply chain network optimisation [52]. PSO is an evolutionary and population-based metaheuristic that exploits a population of N candidate solutions, which are evolved at each step of the algorithm. For the sake of clarity, the basic notation used to describe PSO is shown in Table 1.

Table 1. PSO Parameters.

Symbol	Definition
i	A generic particle of the swarm.
C	A positive constant.
$gbest(t)$	The overall best solution found until iteration t.
n	The dimension of the solution space.
N	The number of particles of the swarm.
$p_i(t)$	A n-dimensional vector that codifies the position of particle i at iteration t, i.e., a feasible solution of the problem.
$pbest_i(t)$	The best solution found by particle i until iteration t.
t	The current iteration or epoch of the algorithm.
u	A generic random number uniformly distributed in the range [0, 1].
$v_i(t)$	A n-dimensional vector that codifies the velocity of particle i at iteration t.
ω	The inertia factor, a random number uniformly distributed in the range [0, 1].
χ	The constraining factor, a constant with a value between $(0,\ 1]$.
$f(p)$	The objective function of the problem.

The whole population is referred to as a *swarm*, while each solution ($i = 1, \ldots, N$) of the swarm is referred to as a *particle*. At each step (or iteration) t, each particle moves in the solution space to generate a new and potentially improved solution. Specifically, its movement is formally defined by Equation (1):

$$p_i(t+1) = p_i(t) + v_i(t) \tag{1}$$

where $v_i(t)$ and $p_i(t)$ are, respectively, the velocity and the position of particle i at iteration t. Both $v_i(t)$ and $p_i(t)$ are vectors with several components equal to the dimensions of the solution space: the position $p_i(t)$ corresponds to the current solution coded by particle i, and the velocity $v_i(t)$ defines how far it will move at the next step.

At each iteration, the speed of each particle is updated as in Equation (2).

$$v_i(t+1) = v_i(t) + C_1 u^t_{1,i}(p_i(t) - pbest_i(t)) + C_2 u^t_{2,i}(p_i(t) - gbest(t)) \tag{2}$$

where:

- $gbest(t)$, the best solution found up to iteration t by the whole swarm.
- $pbest_i(t)$, the best solution found up to iteration t by particle i.
- $u^t_{1,i}$ and $u^t_{2,i}$ are two random numbers uniformly distributed in the range [0, 1]. At each iteration t they are generated for each particle i.
- C_1 and C_2 are fixed parameters.

According to Equation (2), each particle tries to keep moving following the direction coded by $v_i(t)$ that, for this reason, is also referred to as the particle's *intention*. However, the trajectory of the particle is subjected to small deviations that are related to its experience and to the knowledge (or culture) of the whole storm. Experience and culture are measured as the distance from the current position $p_i(t)$ and to the position of $gbest(t)$ and $pbest_i(t)$, respectively. The entity of both deviations is random but limited by the two constants C. In this way, the algorithm balances the exploration of new solutions and the exploitation of the best solutions found so far. When $C_1 = C_2 = 0$, all particles fly keeping their current speed

and direction until they finally hit the boundaries of the search space. If $C_1 > 0$ and $C_2 = 0$, particles behave as if they were independent individuals, whereas if $C_1 = 0$ and $C_2 = 0$, they are all attracted by the same point. A c choice common choice is to set $C_1 = C_2 = 2$. In this way, all particles are attracted towards the midpoint between $gbest(t)$ and $pbest_i(t)$ and the strength of this attraction is not negligible (as it would be with $C_1 = C_2 \ll 1$).

To summarize, the standard PSO works as follows:

- At every iteration t, each particle i moves from $p_i(t)$ to $p_i(t+1)$ according to Equation (1).
- If the new solution $p_i(t+1)$ is better than $pbest_i(t)$, the current best solution found by i is updated accordingly.
- When all the swarm has moved, $gbest(t)$ is updated and the velocity of each particle i is updated using Equation (2).
- The process is repeated until a stopping criterion is met.

For the sake of clarity, the pseudocode is shown in Figure 1, where it is assumed that the PSO ends when the number of iterations without improvement exceeds a predefined threshold value. The pseudocode is written using a Python 3.9© style, and so, indentation is not only for readability but it also defines different blocks of code.

```
def PSO (f, P, n, C1 = 2, C2 = 2, Max_T = 100):
"""
f is the objective function to be minimized,
N is the number of particles of the swarm, n the dimension of the solution space R^n,
C1 and C2 are two constants with a default value of 2,
Max_T is maximum number of iterations, without a solution improvement, before
the algorithm stops. Default is 100
"""
""" PSO SET UP """
01    gbest = (None, math.inf) # a tuple containing gbest and its objective function, initially infi-
nite02    no_imp = 0 # an iteration counter (number of iterations without an improvement)
03    swarm = {} # a mapping associating to each particle i its current position, velocity and pers.
best
04    for i in range(N):
05        p = gen_random_position(n) # a random point in the solution space
06        v = gen_random_velocity(n) # a random velocity
07        of = f(p) # the objective function evaluated at p
08        swarm[i] = {'p': p, 'v': v, 'pbest': p, 'of_pbest': of)} # particle i
09        if of < gbest [1]: gbest = (p, of) # the overall best solution is updated, if needed
""" PSO EVOLUTION """
10    while True: # each iteration of the while loop is an epoch t of the PSO
11        tbest = (None, math.inf) # the best solution found at epoch t and its objective function
""" Each particle moves in the solution space. """
12        for part in swarm: # iterating on swarm, pr becomes a mapping like
# {'p': p, 'v': v, 'pbest': pbest, 'of_pbest': f(pbest)}
13            part['p'] = gen_new_pos(part['p'], part['v']) # new pos. generated with Eq.
(1)
14            of = f(part['p']) # objective function evaluated at point p
15            if of < part['of_pbest']:
16                part['pbest'] = part['p'] # updating personal best
17                part['of_pbest'] = of
18                if of < tbest [1]: tbest = (part['p'], of) # updating the best sol. of epoch t
""" All particles have changed position. """
19        if tbest [1] < gbest [1]:
20            gbest = tbest # if there is an improvement gbest is updated accordingly
21            no_imp = 0 # the counter is reset
22        else:
23            no_imp += 1
24            if no_imp > Max_T: return gbest
""" Velocity is updated before starting the next epoch. """
25        for part in swarm:
26            part['v'] = gen_new_vel(part['p'], part['v'],
part['pbest'], gbest [0], C1, C2) # new velocity
# computed as in Equation (2)
```

Figure 1. PSO pseudocode.

At present, the original PSO has been modified and extended in many ways. A significant contribution is that of [53], where a new parameter called *inertia* is introduced, and the updating procedure is modified as in Equation (3).

$$v_i(t+1) = \omega_i^t \cdot v_i(t) + C_1 r(p_i(t) - pbest_i(t)) + C_2 r(p_i(t) - gbest(t)) \quad (3)$$

where the inertia ω_i^t is a random number uniformly distributed on the interval [0, 1] generated for each particle i at each iteration t. Hence, the scope of ω is to mitigate the effect of particles' intention, and in many applications, its value is progressively reduced as the number of iterations increases [53]. The resulting effect is to force the exploration of a wider area of the solution space during the first runs of the algorithm and to exploit the local optima (sort of neighbour search) when the procedure is almost at the end.

As an alternative, [54] proposed updating the velocity, as in Equation (4).

$$v_i(t+1) = \chi[v_i(t) + C_1 r(p_i(t) - pbest_i(t)) + C_2 r(p_i(t) - gbest(t))] \quad (4)$$

where χ is a constant value (in the range [0, 1]) called *constriction factor* that should ensure better reliability and greater stability of the results. As for the inertia, to assure convergence, some authors have also proposed a time dependent constriction factor, as in [55] where, as the particle becomes closer to the global minimum, a lower value of the constriction factor is used which helps in stabilizing the algorithm with fast convergence. We anticipate that neither the inertia nor the constriction factor will be used in the metaheuristic here proposed.

All the above-mentioned approaches work well when the objective function is continuous, and it is possible to introduce a velocity vector v_i on the solution domain. When, as in most industrial settings, problems are binary and/or discrete, their industrial relevance becomes rather scarce. For this reason, many scholars have proposed discretized versions of the PSO. For instance, [56] proposed using a sigmoid function to convert continuous variables into binary ones, whereas [57] suggested working with continuous variables and rounding off the obtained solution to the nearest integer.

Relatively to the discrete domain, PSO has recently been proposed as an effective way to tackle NP-hard routing problems such as the TSP. Among the first contributions, we can cite the relevant works by [58,59]. However, the real burst of interest originated from [20,21], who showed the superiority of PSO in tackling these NP-hard problems. The point to note is that, when operating over a discrete domain, to outperform other metaheuristics, PSO must be subject to major changes that can make it distinctive from the standard and continuous version. Hence, the main question is how the concept of speed can be redefined so that the distance between two discrete solutions can be measured. In this regard, two remarkable contributions were proposed by [22,60]. Specifically, letting S_1 and S_2 be two sequences representing the order with which n nodes must be visited, both authors expressed speed as a function of the number of swaps needed to convert S_1 to S_2. Although the results are rather good, the performances cannot be compared with those obtainable with PSO implementations where the concept of speed is completely overturned, as in the works by [23,61]. For a more in-depth description of the latter approach, please see Appendix A, which describes the algorithm by [22,23] and other approaches used as benchmarks.

4. Problem Definition and Mathematical Formulation

4.1. Problem Assumptions

In general terms, the order-picking problem can be defined as "*the problem of collecting a set of SKUs in a manual warehouse in a minimum amount of time*" [31]. In the present paper, we contextualize the problem using the following assumptions, which hold in most industrial settings.

1. The set of SKUs to be collected is indicated in a picking list, which also reports the quantity, expressed in the number of cases, to be picked up.

2. Each SKU is stored at the ground level and in a dedicated location in the warehouse. Hence, the picking list univocally defines the locations that must be visited to fulfil an order.
3. The aisles of the warehouses are bidirectional and are large enough to avoid any traffic problems.
4. There is a single Input Output (I/O) point, from which each tour starts and ends.
5. The picker receives a picking list at a time and must collect all the cases indicated in it with a single tour (i.e., the handling capacity of the picker is assumed to be limitless).
6. When the picker moves from one location to the next one, he or she proceeds at a constant speed, moving approximately at the centre of an aisle.
7. When the picker reaches a picking location, he or she moves from the centre of the aisle to the proximity of the shelves where SKUs are stored. The time needed for this traversal movement is constant.
8. The time needed to pick up a case is constant, and it does not depend on the shape, dimension, or weight of the case.

4.2. Problem Formulation and Graph Representation

It is easy to see that the total travelled time is the only relevant quantity of the problem; all the other times are constant (see assumptions 6 and 7) and can be excluded from the optimization procedure. Also, since the travelling time depends on the sequence with which the picker collects the SKUs, the goal is to find the optimal sequence (or optimal picking tour). It is, therefore, a permutational problem that can be reduced to a standard TSP.

To this aim:

- The layout must be schematized as a bidirectional graph, with nodes representing the I/O point, the storage locations, and the access points to the aisles. Considering that each aisle serves two counterposed shelves, the total number of nodes is $\mathcal{N} = (0.5\mathcal{L} + 2\mathcal{A} + 1)$, where $\mathcal{L}$ is the number of storage locations and $\mathcal{A}$ is the number of aisles. Furthermore, due to the constrained layout of the warehouse, (based on horizontal and vertical aisles), each node is linked with two or at most with three other nodes. The only exception is the I/O point that is linked with only one node. For the sake of clarity, an example of graph representation is shown in Figure 2, which is relative to a warehouse with 24 locations (from A to X) and three horizontal aisles. The total number of nodes is therefore (24/2 + 2 × 3 + 1) = 19. Also, note that node zero represents the I/O point and that nodes inside the aisles serve two locations each (for instance node two serves both location S and location T).
- The shortest path connecting each pair of nodes (including the I/O point) must be computed and saved in a triangular distance matrix D. This can easily be carried out in a polynomial time, using the well-known Floyd–Warshall algorithm [47,48].
- Given a picking list with M SKUs, a bidirectional and fully connected network with $(M+1)$ nodes and $M \cdot (M+1)$ edges must be created. The nodes correspond to the I/O point and to the M locations that must be visited, where the edges codify the distances among them. These distances are read in the distance matrix D, i.e., the length of the edge connecting node i to node j equals $D[i,j] = d_{ij}$. For instance, if the picking list was {S, R, V}, the network of Figure 3 should be built, where it is supposed that the distance between two adjacent nodes equals 2 and that between node zero and node one equals 1.
- The optimal sequence is finally found in solving a TSP (as in Equation (5)) formulated for the network generated by the picking list.
- OF:

$$\min(z) = \sum_{i=1}^{M+1} \sum_{j=1}^{M+1} x_{ij} d_{ij} \tag{5a}$$

- ST:

$$\sum_{i=1}^{M+1} x_{ij} = 1 \quad \forall j \in \{1, 2, \ldots, (M+1)\} \tag{5b}$$

$$\sum_{j=1}^{M+1} x_{ij} = 1 \quad \forall i \in \{1, 2, \ldots, (M+1)\} \tag{5c}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \tag{5d}$$

$$x_{ij} \in \{0,1\} \; \forall i \in \{1, 2, \ldots, (M+1)\}, \; \forall j \in \{1, 2, \ldots, (M+1)\} \tag{5e}$$

where

- x_{ij} is a binary decision variable that equals one if the picker moves from location i to location j, and it is zero otherwise.
- d_{ij} is the distance between location i to location j.
- M is the number of SKUs in the picking list.
- $(M+1)$ is the number of nodes in the network (M locations for M SKUs and one I/O point).
- S is a generic set of nodes, and $|S|$ denotes its cardinality. Since the cardinality $|S|$ ranges from 2 to M there are 2^M possible set S.

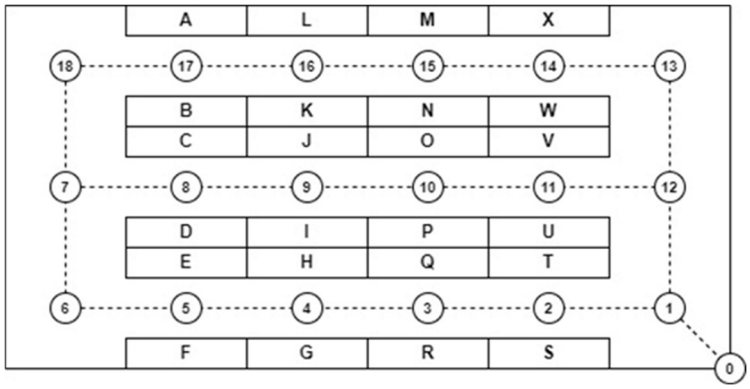

Figure 2. An example of graph representation.

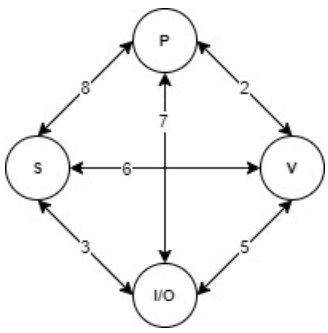

Figure 3. The subgraph generated by a picking list.

The objective (Equation (5a)) consists of finding the minimum travelling distance, and the constraints make sure that each location of the picking list is visited only once (i.e., constraints (5b,c)) and that subtours are avoided (i.e., constraint (5d)). For an additional insight of this integer linear programming model, the interested reader is referred to the extensive literature review on the TSP (see for example [24]).

Continuing with the example, solving the optimization problem, the optimal tour {I/O → P → V → S → I/O} is obtained, for a total distance of 18. We also note that the

actual path followed by the picker can be reconstructed, starting from the optimal tour, using the Floyd–Warhsall algorithm in reverse. In this way, the following path is finally obtained: {0, 1, 12, 11, 10, 11, 12, 1, 2, 1, 0}.

5. Pkt_PSO, the Proposed Metaheuristic

Since the TPS is a well-known NP-hard problem, a novel metaheuristic is herein proposed to solve the order-picking problem. The metaheuristic takes inspiration from the PSO and so, from here on, it will be referd to as the Pkt_PSO.

5.1. Notation

As discussed in Section 4, the problem input is a picking list $\mathcal{P}$. We denote the picking list as $\mathcal{P} = \{A, B, C, D, E, F\}$, where the uppercase letters are the M locations to be visited. Locations are not sorted or can be sorted in lexicographical order just for convenience.

Similarly, a feasible solution is a sorted array of $(M+2)$ elements, which defines a picking tour. Since a tour always starts and ends at the I/O point, the first and last elements can be removed, and the tour can be fully represented with an array of only M elements. We refer to this array as the picking tour and we denote it as $p = \langle h_1, h_2, \ldots, h_M \rangle$, where each element $p[j] = h_j$ identifies the j-th location that must be visited by the picker. Consider, for instance, a picking list with six codes $\mathcal{P} = \{A, B, C, D, E, F\}$, and a picking tour $p = \langle B, A, C, F, D, E \rangle$. In this case, the first stop is at location B, the second one is at location A, and so on until location E is reached and the picker completes the tour (i.e., $p[1] = h_1 = B,\ p[2] = h_2 = A,\ \ldots, p[6] = h_6 = E$).

The objective function evaluated in a specific tour is therefore computed as in Equation (6):

$$f(p) = d_{(IO,\ h_1)} + \sum_{j=1}^{M} d_{(h_j,\ h_{j+1})} + d_{(h_M,\ IO)} \tag{6}$$

where $d_{(x,y)}$ is the minimal distance between location x and location y and IO is the input–output point.

For the sake of clarity, before delving into the details of the Pkt_PSO, the notation of the problem is fully introduced in Tables 2 and 3. This notation integrates that of Table 1, valid for a generic PSO.

Table 2. Variables related to the picking problem and to the layout of the warehouse.

Symbol	Definition
$\mathcal{P}$	The picking list $\mathcal{P} = \{A, B, C, \ldots\}$ containing all the storage locations that must be visited by the picker to complete an order. Locations in $\mathcal{P}$ are indicated with an uppercase letter and they are not sorted; they may be sorted in lexicographical order just for convenience.
p	The picking tour $p = \langle h_1, h_2, \ldots, h_M \rangle$ containing all the locations of $\mathcal{P}$ sorted in the order that the picker must follow during the tour. Please note that locations in p are denoted with the letter h.
j	The positional index $j = 1, \ldots, M$ that specifies at which point of the tour a location is visited.
h_j	The j-th location to be visited, i.e., $p[j] = h_j$.
$d_{x,y}$	The minimum distance between location x and location y, with $x, y \in \mathcal{P}$. The minimum distance is always obtained using the Floyd–Warshall algorithm.
$\mathcal{S}_{x,y}$	The set of all the storage locations l located within the shortest path linking x and y. That is, to move from x to y, the picker also passes in front of all locations contained in $\mathcal{S}_{x,y}$.
s	The cardinality of the set $\mathcal{S}_{x,y}$, i.e., the number of locations l in $\mathcal{S}_{x,y}$

Table 3. Variables related to the parameters of the algorithm.

Symbol	Definition
$p_i(t)$	A completed picking tour (solution) generated for particle i at iteration t. With "complete" we mean that $p_i(t)$ is made of M locations.
$p_i^m(t)$	A partial picking tour generated for particle i at iteration t. The tour is partial because it is made of m locations with $m < M$.
T	The maximum number of iterations before the algorithm stops.
$greedy$	A greedy solution shared by all the N particles of the swarm.
$rnd_intention_i(t)$	A random solution generated for particle i at iteration t.
$\mathcal{H} \subseteq \mathcal{P}$	A subset of $\mathcal{P}$ containing four candidate locations that can be selected to generate a new tour.
$\mathscr{h}_k$	One of the four candidate locations contained in $\mathcal{H}$.
$d_{(h_j,\mathscr{h}_k)}$	The distance between the j-th and last location of $p_i^j(t)$ and the candidate location $\mathscr{h}_k$.
$w_i(t)$	A four-elements vector $w_i(t) = \langle w_1, w_2, w_3, w_4 \rangle$ that determines the tendency of particle i at iteration t to change its position, moving toward one of reference positions.
$\mathcal{D}(t)$	A rescaled distance vector generated at each iteration t, which is used to create the new solutions $p_i(t)\ \forall\ i$.
$\vec{\mathcal{H}}$	The array of the candidate solution sorted in ascending order of their rescaled distance.
g	The greediness factor used to quantify the propensity of a particle to exploit the greedy solution.
α	The shape factor of the quasi-geometric distribution used to select a candidate location $\mathscr{h}_k$ among the four included in $\vec{\mathcal{H}}$.
γ	The local search probability, i.e., the probability of performing a 2-Opt-based neighbour search on a starting solution.
β	The probability of exploiting the *aisle-based* generation mechanism; in a certain way, this probability codifies the propensity of a particle to follow a predetermined path inside an aisle of the warehouse.

5.2. Basic Framework of the Algorithm

The iterative procedure of the Pkt_PSO follows a standard framework, like the one described in Section 3. Specifically, at every iteration $t \in \{1, 2, \ldots, T\}$:

- Each particle i moves from its previous position $p_i(t-1)$ to the new position $p_i(t)$. This part is the core of the optimization procedure, which is explained in full detail in Sections 5.4 and 5.5.
- Next, with probability γ, a neighbour search is made on $p_i(t)$ using a 2-Opt-based approach, as explained in Section 5.6.
- If the solution has improved, the personal best is updated of particle i is updated, i.e., $pbest_i(t) \leftarrow p_i(t)$, if $f(p_i(t)) < f(pbest_i(t-1))$.
- When all particles have moved to a new position, the minimum personal best is computed and compared to the global best. In case of improvement, the global best is updated accordingly, i.e., $gbest(t) \leftarrow \min_i pbest_i(t)$, if $\min_i f(pbest_i(t)) < f(gbest(t-1))$.

Lastly, after T iterations (or when a termination criterion is met), the algorithm ends and the best solution $gbest(T)$ found so far is returned as the output. Since optimality cannot be guarantee, this solution is referred to as pseudo-optimal solution.

5.3. New Solution Generation

Anytime particle i moves, a new solution $p_i(t)$ is generated in an *element-by-element* way, by sequentially adding to an initially empty tour $p_i^0(t)$ one location at a time. Also, the locations used to form $p_i(t)$ are sequentially extracted from four reference solutions, that are: $pbest_i(t)$, $gbest(t)$, $greedy$ and $rnd_intention_i(t)$. As revealed by the notation, $greedy$ is a solution that depends neither on iteration t nor on particle i. As better described in Section 5.5, this solution is shared by all the particles of the swarm, and it is generated with a greedy algorithm at the very beginning of Pkt_PSO. From this point onward, it is no longer changed. Conversely, $rnd_intention_i(t)$ is a random solution that is regenerated at every iteration t for each particle i.

Concerning how locations are extracted from the four reference solutions, the following procedure is used. Suppose that a total of j locations have already been inserted in the partial tour $p_i^j(t)$, i.e., the last inserted location is h_j and $p_i^j(t) = h_1, \ldots, h_j$. Now, we need to define, among the remaining $(M - j)$ locations of the picking list, which one should be inserted at position $(j + 1)$. To this aim, a set of possible options $\mathcal{H} \subseteq \mathcal{P}$ is created by taking all locations that come immediately after h_j in the four reference solutions. More precisely, let k_{gb}, k_{pb}, k_{gr} and k_{rn} be the position occupied by location h_j in $gbest(t)$, $pbest_i(t)$, $greedy$ and $rnd_intention_i$, respectively. So, we have that $\mathcal{H} = \left\{\hbar_{gb}, \hbar_{pb}, \hbar_{gr}, \hbar_{rn}\right\}$, where:

- $\hbar_{gb} = gbest_i\left[k_{gb} + 1\right]$,
- $\hbar_{pb} = pbest_i\left[k_{pb} + 1\right]$,
- $\hbar_{gr} = greedy\left[k_{gr} + 1\right]$,
- $\hbar_{rn} = rnd_intention_i[k_{rn} + 1]$.

Note that, if the partial tour $p_i^0(t)$ is empty, the k-indexes cannot be defined and so they are conventionally set to zero, i.e., $k_{gb} = k_{pb} = k_{gr} = k_{rn} = 0$.

To better clarify this concept, the extraction procedure is exemplified in Figure 4, for a partial tour made of two locations $p_i^2(t) = \langle h_1 = E, h_2 = A \rangle$. Since location A is the last one inserted, to generate the set of the candidate locations $\mathcal{H}$, we need to consider all the locations that come after A in the reference solutions. For instance, in $gbest_i(t)$, location A is at position four (i.e., $k_{gb} = 4$) and so $\hbar_{\mathrm{gb}} = gbest_i\left[k_{gb} + 1\right] = gbest_i[5] = C$. Considering the other reference solutions, we finally have that: $\mathcal{H} = \left\{\hbar_{\mathrm{gb}} = C, \hbar_{\mathrm{pb}} = D, \hbar_{\mathrm{gr}} = \mathrm{B}, \hbar_{\mathrm{rn}} = F\right\}$. Therefore, the next and third location of $p_i^3(t)$ (indicated with three question marks in Figure 1) will be chosen among $\{C, D, B, F\}$.

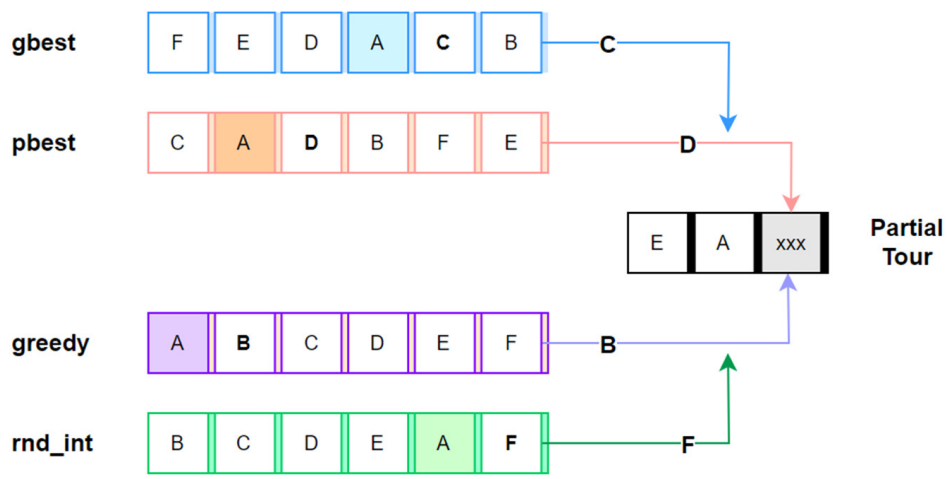

Figure 4. Creation of the set $\mathcal{H}$ of the candidate locations.

Once the four candidate solutions of $\mathcal{H}$ have been identified, one of them must be selected and appended to the partial tour. As the objective is to minimize the overall

travelling distance, a natural choice is to link the selection probability to the distance $d_{(h_j,\hbar_k)}$ between the last location h_j of the partial tour $p_i^j(t)$ and each location $\hbar_k$ of $\mathcal{H}$. In other words, the greater is $d_{(h_j,\hbar_k)}$, the lower is the probability of selecting $\hbar_k$ and vice versa. Also, to give a different importance to the reference solutions, the distance $d_{(h_j,\hbar_k)}$ should be rescaled, taking the Hadamard (or elementwise) product with the weighting vector $w_i(t) = w_1, w_2, w_3, w_4$.

This is shown in Equation (7), where $\mathcal{D}$ is the rescaled distance vector:

$$\mathcal{D}(t) = \left\langle d_{(h_j,\hbar_{gb})}, d_{(h_j,\hbar_{pb})}, d_{(h_j,\hbar_{gr})}, d_{(h_j,\hbar_{rn})} \right\rangle \circ \langle w_1, w_2, w_3, w_4 \rangle \tag{7}$$

In general terms, neither $d_{(h_j,\hbar_{gb})}$ nor $d_{(h_j,\hbar_{pb})}$ should be modified, whereas $d_{(h_j,\hbar_{gr})}$ and $d_{(h_j,\hbar_{rn})}$ should be increased. The first two distances correspond to $gbest_i(t)$ and $pbest_i(t)$, which are the most important reference solutions, whereas the latter two correspond to *greedy* and $rnd_intention_i(t)$ that are the less important ones. Therefore, a natural choice is to use the following weight vector $w_i(t) = \langle 1,\ 1,\ (1/g), 1/(g-1) \rangle$, for each particle i and for each iteration t, where $g \in [0,\ 1]$ is a *greedy factor*, that measures how much the greedy solution outweighs the random one. A neutral level is obtained for $g = 0.5$ and rescales the distances by doubling the original ones. We anticipate that, under some peculiar conditions, the weight vector $w_i(t)$ could be modified for a limited number of particles, to trigger an antistagnation mechanism, as described in Section 5.5.

To implement this logic, first the four candidate solutions $\hbar_k$ are sorted in ascending order of their rescaled distance, to form the sorted array $\vec{\mathcal{H}}$. Next, an element of $\vec{\mathcal{H}}$ is randomly extracted using the quasi-geometric distribution defined as in Equation (8) [62].

$$f(x) = (1-\alpha)^x \tag{8}$$

where α is a shape parameter in the interval [0, 1), and x is the position occupied by a candidate solution in the sorted array $\vec{\mathcal{H}}$. Therefore, the selection probability is maximum for the element at position one and minimal for the element at position four.

It is also important to stress that the use of a quasi-geometric distribution has a twofold purpose, as clearly explained by Juan et al. [63]. First, the adoption of a theoretical distribution provides implementation advantages concerning the use of an empirical distribution. Second, the quasi-geometric distribution has a single parameter, which greatly simplifies the fine-tuning step.

We conclude by noting that two issues could occur during the selection process. Indeed:

- If a candidate location $\hbar_k$ is already included in the partial tour $p_i^j(t)$, $\hbar_k$ is unusable and cannot be included in $\mathcal{H}$.
- If the location h_j (the last one of the partial tours $p_i^j(t)$) matches the last location of a reference solution, its candidate location $\hbar_k$ cannot be defined, since position $[M+1]$ does not exist.

When this happens, the cardinality of $\mathcal{H}$ is less than four; anyhow, provided there is at least one element in $\mathcal{H}$, the selection procedure remains unaltered. Should $\mathcal{H}$ be empty, the next location $h_{(j+1)}$ is randomly extracted among all locations that have not yet been included in $p_i^j(t)$.

5.4. Exploiting the Structure of the Problem: Aisle-Based Construction and Greedy Algorithm

To exploit the underlying structure of the picking problem, an additional solution-generation scheme is considered. As described in Section 4.2, the picking problem is characterized by a very constrained layout that makes the path connecting two locations virtually unique [64]. In other words, whenever the picker goes from location x to y, he or she necessarily passes in front of other locations. This is typical when x and y are at opposite sides of the same aisle, so that moving from x to y, all the intermediate locations

are necessarily encountered. For instance, referring to Figure 2, if the picker moves from location V to location D, he or she also passes in front of locations {O, J, C, U, P, I}.

Let $S_{(x,\,y)} \subseteq \mathcal{P}$ be the set of all the locations found on the minimal path connecting x and y. To take advantage of this fact, when a new candidate location $\hbar_k$ is selected, all locations $l \in S_{(h_j,\hbar_k)}$ will also be included in the partial tour, with probability β. Therefore, the following two partial tours may be generated:

- $p_i^j(t) = \left\langle h_1, \ldots, h_j, h_{(j+1)} = \hbar_k \right\rangle$, with probability $(1-\beta)$.
- $p_i^{j+s}(t) = \left\langle h_1, \ldots, h_j, h_{(j+1)} = l_1, \ldots, h_{(j+s)} = l_m, h_{(j+s+1)} = \hbar_k \right\rangle$, with probability β

  where s is the cardinality of $S_{(h_j,\hbar_k)}$. Please note that the first one is the standard tour, and the second is the one generated using the aisle-based construction algorithm.

Due to the very constrained graph of routes, we also suggest generating the greedy solution (shared by all the particles of the swarm) using the well-known *nearest neighbour algorithm*. According to this algorithm, the picker starts the tour from the I/O point and repeatedly visits the nearest location until all locations in $\mathcal{P}$ have been visited. Indeed, due to the constraints induced by the layout on the feasible paths, this solution is known to be good, and it is thus advantageous to include it within the four reference solutions used to build a new tour. Also note that this solution never changes and remains the same from iteration t = 1 to iteration $t = T$. This choice may seem rather odd, but it can be justified as follows. At the single-item level, the nearest neighbour move is certainly optimal; it is therefore useful to ensure that this move is always available when a new tour is generated in an element-by-element way. This is exactly why the greedy solution must be included in the reference solutions.

For clarity, Figure 5 provides a Python 3.9©-based pseudocode, showing the generation of a new solution $p_i(t)$.

5.5. Local Search via 2-Opt

Any time a new solution $p_i(t)$ is generated, with probability γ, a local search is also performed. The local search is based on the 2-Opt, a simple local search algorithm originally introduced by Croes in 1958 [65], which is characterized by a special swapping mechanism.

Specifically, the proposed local search can be executed in a *complete* and *full-depth* mode, which differ in the number of 2-Opt exchange steps that they perform. If the *complete 2-Opt* is used, a new sequence is generated by inverting the order in which the locations between h_j and h_k in $p_i(t)$ are visited. This procedure is called complete because the inversion is made for each possible couple of nodes (j,k). To make a simple numerical example, let us consider a four-item sequence $p_i(t) = \langle A, B, C, D \rangle$. In this case, a complete 2-Opt search generates the following six tours: $\langle B, A, C, D \rangle_{1,2}$, $\langle C,\ B, A, D \rangle_{1,3}$, $\langle D,\ C,\ B, A \rangle_{1,4}$, $\langle D,\ C,\ B, A \rangle_{1,4}$, $\langle A,\ C,\ B, D \rangle_{2,3}$ and $\langle A,\ C,\ B, D \rangle_{2,3}$, where the subscripts refer to node j and k, respectively. Instead, when the *full-depth 2-Opt* is used, the complete 2-Opt is recursively repeated on each improved solution that is found. The two approaches are performed with probability γ_c and γ_f, respectively, with $\left(\gamma_c + \gamma_f\right) = \gamma$. Also, to balance accuracy and computational time, we suggest using $\gamma_f = 0.5\gamma^2$ so that $\gamma_c = \left(\gamma - 0.5\gamma^2\right) \cong \gamma$.

For clarity, Figure 6 shows the Python 3.9 © pseudocode of the local search via 2-Opt.

```
def new_solution(𝒫, Gb, Pb, Gr, R_int, M, alpha, beta, g)
"""
𝒫 is the picking list,
Gb, Pb, Gr and R_int are the four reference solutions, i.e., global best, personal best,
greedy solutionand particle's intention,
alpha is the shape parameter of the quasi-geometric distribution,
beta is the probability to perform the aisle-based generation,
g is the greediness factor.
"""
p = [] # an initially empty tour
j = -1 # the starting index; a zero-based indexing is assumed
M = len(𝒫) # number of locations or SKUs
w = [1, 1, 1/g, 1/(g − 1)] # the weight vector
while len(p) != M: # Loop while the new solution is still uncomplete
 ℋ = [get_location after(j, Ref) for Ref in (Gb, Pb, Gr, Rin)]
# This is the list of the candidate solutions [ℎ_gb, ℎ_pb, ℎ_gr, ℎ_r_int],
# The function get_location_after(j, Ref) returns Ref[k_Ref + 1],
# None if j == M or gb[k_gb + 1] in p
    if all(ℎ_k == None for ℎ_k in ℋ): # if all candidate solutions are None
        h = random_choice_from(𝒫) # next position of p is randomly extracted
# from the locations in 𝒫
    else:
        if j == -1: dst = [distance_from(IO, ℎ_k) for ℎ_k in ℋ]
# distances from the IO point and the locations ℎ_k in ℋ
        else: dst = [distance_from(p[−1], ℎ_k) for ℎ_k in ℋ] ù
# distances from the last location of p and locations ℎ_k in ℋ
        res_dst = [dst[x]*w[x] for x in range(M)]
# rescaled distances, obtained as the Hadamard Product
        ℋ.sort(key = res_dist, reverse = False)
# candidate locations sorted in ascending order of their rescaled distances
        h = quasi_geometric_distr(ℋ, alpha) # next position of p is sampled from ℋ
""" Aisle based generation """
    r = random() # returns a pseudo-random number
    if r < beta:
        S = S_find(p[-1], h) # Returns the set of all locations in between
# the last locations of p and the new location h
        p.extend(S) # All locations in S are appended to the partial tour p
        𝒫 = [l for l in 𝒫 if l not in S] # All locations in S are removed
# from the picking list
    p.append(h) # h is appended to the partial tour, too
    𝒫.remove(h) # h is removed from 𝒫, too
 return p
```

Figure 5. Pseudocode concerning the generation of a new solution.

5.6. Antistagnation Mechanism

Let i^* be a particle for which the full-depth 2-Opt, executed at iteration t, has failed (i.e., $pbest_{i^*}(t)$ has not changed). To avoid stagnation, particle i^* activates a procedure that is meant to escape possible local minima. The proposed procedure acts on the weight vector $w_{i^*}(t)$, as explained next.

- Only for step $(t+1)$ and limited to particle i^*, the weight vector is changed to: $w_{i^*}(t+1) = \langle 1, \infty, (1/g), 1/(g-1) \rangle$.
- Additionally, if $pbest_{i^*}(t) = gbest(t)$, the weight vector becomes: $w_{i^*}(t+1) = \langle \infty, \infty, (1/g), 1/(g-1) \rangle$.
- At the end of the generating process:
  - The current personal best is replaced by the solution just created, regardless of its objective function, i.e., $pbest_{i^*}(t+1) = p_{i^*}(t)$.

- The weigh vector is restored to its initial values, i.e., $w_{i^*}(t+2) = \langle 1,\ 1,\ (1/g),\ 1/(g-1)\rangle$.

```
def Two_Opt_local_search (p, f_search):
"""
p is a copy of the current solution,
f_search is a boolean value; if it is true the full deep search must be made
"""
    best_sol = p # the best solution found so far
    """ By iterating on i and on j, all the possible cutting points are tried """
    for i in range(0, len(p) - 1):
        for  j in range(i + 1, len(p)):
            new_sol = p[:i] + list(reversed(p[i:j])) +p[j:] # initial and ending part
# of p are joined with the reversed sub path p[i:j]
            if f(new_sol) < f(best_sol): # if the objective function has improved,
# the best solution is updated
                best_sol = candidate_sol
                if f_search == True:
                    best_sol = Two_Opt_local_search(best_sol.copy())
# This is a recursive call, made on a copy of the best solution
    return best_sol
```

Figure 6. Pseudocode of the 2-Opt local search.

In this way, all locations $\hbar \in pbest_{i^*}(t+1)$ become the least likely to be selected and the particle tends to move away (or escape) from its current best. Metaphorically, limited to iteration $(t+1)$, particle i^* temporarily forgets its personal best (and eventually the global best) and explores the solution space without making use of this past information.

5.7. Time Complexity of Pkt_PSO

For the sake of clarity, the whole algorithm is summarized in the flow diagram of Figure 7. From the diagram, it should be clear that in each epoch t, each particle i generates at least a new solution using the *element-by-element* generation scheme, possibly enhanced by the *aisle-based* generation approach. Next, if a neighbour search is performed, an additional number of solutions is generated. This extra number can be easily determined if the *complete 2-Opt* is used. In this case $C(M,n)$ solutions will be considered, where $C(M,n)$ is the binomial coefficient. Conversely, determining an exact number of different sequences generated by the full-depth 2-Opt before convergence is challenging, as this number will vary depending on the problem size and characteristics. Anyhow, since the complexity of the full-depth 2-Opt is generally considered to be $O(M^2)$, we can confidently assume that the number of explored solutions is $<<M^3$, especially because, in relative terms, the number of iterations required by the 2-Opt to converge tends to decrease as the problem size increases. or a further discussion on this point see [66].

Because of this, considering the number of particles of the storm N and the probability to perform a neighbour search γ, the total number of generated solutions during T epochs can be estimated as in Equation (9).

$$N_Solutions = T\cdot\left(N\cdot\left(1+\left(\gamma-0.5\gamma^2\right)\cdot\binom{M}{2}+0.5\gamma^2\cdot\varphi(M)\right)\right) \tag{9}$$

where $\varphi(M) \ll M^3$ is the number of solutions generated by the full-depth 2-OPT. For instance, using problem instances as the ones discussed next in Section 7, we experimentally found that $\varphi(20) \cong 1400$, $\varphi(30) \cong 3750$ and $\varphi(30) \cong 8400$.

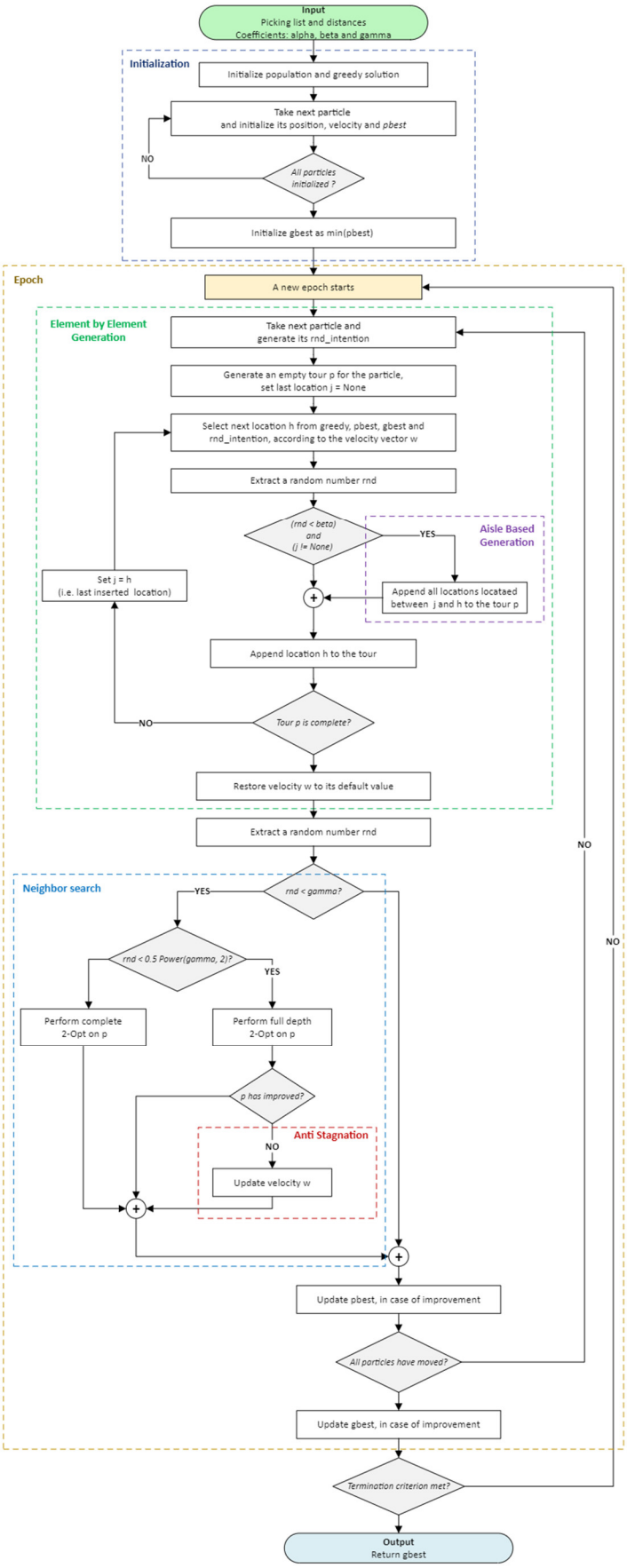

Figure 7. The flowchart of the proposed metaheuristic.

5.8. Concluding Remarks on the Proposed Algorithm

In a standard PSO, particles constantly move according to a velocity profile that codifies the knowledge of the good solutions visited in the past. Indeed, any time a particle moves, a new solution $p_i(t+1)$ is generated based on its current position $p_i(t)$ and current velocity $v_i(t)$. The latter, in turn, depends both on a particle's personal best and on the swarm's global best. Metaphorically, each particle represents a bird that explores an area looking for food and changes its trajectory based on its intuition (*cognitive behaviour*) and/or to imitate the paths followed by the other birds of the storm (*social behaviour*).

Akin to the standard PSO, we also construct new solutions exploiting $pbest_i(t)$ and $gbest(t)$; however, aside from this similarity, a careful reader could spot some evident divergences. Specifically:

- Due to the discrete nature of the problem, an analytic formula (as in Equation (1)) is not implemented to recombine the reference solutions; an element-by-element approach is used instead.
- Two additional solutions, namely $greedy$ and $rnd_intention_i(t)$, take part in the generation process.
- Conversely, the current solution does not explicitly appear in this process.
- The concept of speed does not seem to be explicitly implemented in the framework.

However, especially concerning the last two points, at a conceptual level, these differences are only apparent. Our algorithm can be reinterpreted in the following way. Each particle always moves to start from its personal best, which coincides with its current position. When the particle moves to a new point, it stops there if the solution has improved; if not, it returns to the starting point. Furthermore, the direction and modulus of the movement depend on four reference solutions and the weight vector $w_i(t)$. In this sense, the reference solutions can be seen as four cardinal points toward which the particles are attracted. Moreover, the weight vector can be seen as a velocity profile that determines how much a particle deviates from its precedent trajectory, heading toward one of the above-mentioned cardinal points.

Also note that the velocity vector is dynamic, and it is also specific for each particle of the swarm. Most of the time, the velocity vector is shared by all particles, but if a full-depth search is made on particle i at epoch t, and the full-depth search fails, for that particle, the velocity vector immediately changes, helping the particle to escape the local minimum, where it presumably was trapped.

To conclude, we note that in the case of discrete problems, the existing literature has two main strands of thought. A minority of authors propose readapting PSO with few marginal changes, while most of the scholars disrupt the original version, giving it a new shape. Given the description of the procedure, we claim that our approach stays in the middle. We reinterpreted the concept of speed, taking inspiration from the recent work by [23], trying at the same time to preserve most of the underlying features of the original PSO, such as the intention of the particles.

6. Parameter Tuning

The algorithm has four operating parameters: (i) the shape factor α of the quasi-geometric distribution, (ii) the probability β of activating the aisle-based construction procedure, (iii) the probability γ to carry out a local search via 2-Opt, and (iv) the greediness factor g, which defines how much the exploration is influenced by the greedy solution. The number of particles of the swarm is another operating parameter that should be fine tuned. However, we verified that the results do not appreciably change if the number of particles remains within the range of 30–40, which is usually used in similar works proposed in technical literature. Therefore, to assure a fair comparison with the other algorithms used as a benchmark, the number of particles was arbitrarily set to 30. Even the probability of triggering a full-depth search should be optimized. However, this event is so rare that performing a detailed optimization would be useless. Hence, also this parameter was arbitrarily fixed to γ^2, as mentioned in Section 5.

For proper fine-tuning, as in a two-level factorial experiment design, we defined reasonable low and high values for each parameter. Specifically:

- For the probability α, the values 0.7 and 0.2 were chosen, as in Juan et al. [63]. When $\alpha = 0.7$, the selection of a potential location $\hbar_k$ is strongly influenced by its rescaled distance. For α approaching one, the selection tends to be deterministic and the candidate solution with the lowest distance is almost always returned. Conversely, when $\alpha = 0.2$, the selection is almost random (with pure randomness that would be achieved with $\alpha \approx 0$).
- For the probability γ, the values 0.2 and 0.05 were chosen. Both values are rather low because the execution of a 2-Opt local search should be an extreme condition with a very low occurrence rate. For values higher than 0.2, the time complexity (see Equation (8)) would grow tremendously, making its practical implementation impossible in a real industrial context.
- For the greediness factor g, we are "pioneers", as this is a completely new parameter. Lacking scientific references (or experimental evidence), two extreme values 0.8 and 0.1 were tested. When $g = 0.8$ the greedy exploration prevails on the random behaviour, and vice versa.
- Also, probability β is a new parameter and so, in line with the previous point, we tested two opposite situations using an extreme value of 0.8 and a low value of 0.1.

All the resulting 2^4 combinations were tested using five different picking lists with an average length of 30 locations (i.e., $M = 30$). A common layout characterized by three blocks divided by two cross aisles and 10 racks per block was also considered. A more accurate description of this layout is presented in Section 7.2, where the numerical assessment of the algorithm is discussed. For each combination and each picking list, the algorithm was executed five times for a total of 400 repetitions. Table 4 reports the results, expressed as the sample mean and standard deviation of the total travelling time of each parameter's combination and picking list.

Table 4. Parameters' tuning obtained results (in red the experiments where Pkt_PSO did not find the global optimum, in green the best parameters combination).

Parameters	Picking List 1		Picking List 2		Picking List 3		Picking List 4		Picking List 5	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
$g = 0.8$, $\alpha = 0.70$, $\beta = 0.8$, $\gamma = 0.20$	533	0	619	0	502	0	432	0	389	0
$g = 0.8$, $\alpha = 0.70$, $\beta = 0.8$, $\gamma = 0.05$	533	0	619	0	502	0	433	0	389	0
$g = 0.8$, $\alpha = 0.70$, $\beta = 0.2$, $\gamma = 0.20$	512	0	619	0	508	3	432	0	389	0
$g = 0.8$, $\alpha = 0.70$, $\beta = 0.2$, $\gamma = 0.05$	512	0	619	0	502	0	432	0	389	0
$g = 0.8$, $\alpha = 0.20$, $\beta = 0.8$, $\gamma = 0.20$	534	0	619	0	508	2	432	0	389	0
$g = 0.8$, $\alpha = 0.20$, $\beta = 0.8$, $\gamma = 0.05$	534	0	619	0	508	3	432	0	389	0
$g = 0.8$, $\alpha = 0.20$, $\beta = 0.2$, $\gamma = 0.20$	513	0	619	0	502	0	432	0	389	0
$g = 0.8$, $\alpha = 0.20$, $\beta = 0.2$, $\gamma = 0.05$	513	0	619	0	502	0	432	0	389	0
$g = 0.1$, $\alpha = 0.70$, $\beta = 0.8$, $\gamma = 0.20$	512	0	619	0	502	0	432	0	389	0

Table 4. *Cont.*

Parameters	Picking List 1		Picking List 2		Picking List 3		Picking List 4		Picking List 5	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
$g = 0.1$, $\alpha = 0.70$, $\beta = 0.8$, $\gamma = 0.05$	512	0	619	0	503	0	432	0	389	0
$g = 0.1$, $\alpha = 0.70$, $\beta = 0.2$, $\gamma = 0.20$	512	0	619	0	502	0	432	0	389	0
$g = 0.1$, $\alpha = 0.70$, $\beta = 0.2$, $\gamma = 0.05$	512	0	619	0	502	0	432	0	389	0
$g = 0.1$, $\alpha = 0.20$, $\beta = 0.8$, $\gamma = 0.20$	512	0	619	0	502	0	438	2	389	0
$g = 0.1$, $\alpha = 0.20$, $\beta = 0.8$, $\gamma = 0.05$	512	0	619	0	503	0	432	0	389	0
$g = 0.1$, $\alpha = 0.20$, $\beta = 0.2$, $\gamma = 0.20$	512	0	619	0	502	0	432	0	389	0
$g = 0.1$, $\alpha = 0.20$, $\beta = 0.2$, $\gamma = 0.05$	520	3	619	0	502	0	432	0	389	0
Global Optimum	512	[-]	619	[-]	502	[-]	432	[-]	389	[-]

The values shown in Table 5 demonstrate that Pkt_PSO is very consistent and reliable. Indeed, for each parameter's combination, the standard deviation is always very low, and the mean value does not change significantly as the parameters change. Results also show that the only parameter that has a significant impact on the results is the greediness factor, which should be kept at its low level (i.e., $g = 0.1$).

Table 5. Comparison of the results, with Avg. Dev expressed in meters; best results are highlighted in bold.

Layout	Pick. List Length	NeNe		Pkt_PSO		PSO_1 [22]		PSO_2 [23]		ACO [9]		2-Opt	
		Avg. Dev.	N. Wins	Avg. Dev.	N Wins	Avg. Dev.	N. Wins	Avg. Dev.	N Wins	Avg. Dev.	N Wins	Avg. Dev.	N Wins
L1	20	75.2	0/10	**0**	**10/10**	537.1	0/10	0.3	8/10	8.0	3/10	34.9	0/10
	30	88.1	0/10	**0**	**10/10**	1095.2	0/10	0.3	8/10	19.1	0/10	59.7	0/10
	40	90.8	0/10	**0**	**10/10**	1711.3	0/10	0.6	6/10	27.2	0/10	67.8	0/10
L2	20	74.0	0/10	**0**	**10/10**	390.5	0/10	**0.0**	**10/10**	5.5	0/10	66.8	0/10
	30	108.0	0/10	**0**	**10/10**	797.2	0/10	**0.0**	**10/10**	19.9	0/10	76.6	0/10
	40	118.7	0/10	**0**	**10/10**	1160.9	0/10	0.2	9/10	28.9	0/10	113.9	0/10
L3	20	162.3	0/10	**0**	**10/10**	720.7	0/10	**0.0**	**10/10**	16.1	0/10	112.8	0/10
	30	129.4	0/10	**0**	**10/10**	1262.8	0/10	**0.0**	**10/10**	19.4	0/10	124.1	0/10
	40	200.2	0/10	**0**	**10/10**	2089.1	0/10	**0.0**	**10/10**	41.8	0/10	186.8	0/10

For clarity, we also highlighted in red the experiments where Pkt_PSO failed, at least one time, to find the global optimum shown in the last row of the Table 5. The global optimum was computed with the famous Concorde TSP solver (using the Python version available in the PyTSP library) which, according to Mulder and Wunsch [67], is one of the fastest TSP solvers. For instance, the tuple (533, 0), relative to the first parameters' combination and first picking list), means that the algorithm always converged to a value of 533 higher than the global optimum. Similarly, the tuple (508, 3), relative to the third parameters' combination and third picking list, means that the algorithm has not always converged to the global optimum. The mean is higher than the global optimum and the standard deviation is not. Similarly, we highlighted in green the combination of parameters

that never failed in finding the global optimum. This is the combination we opted for, also because, having a low value of the local search probability $\gamma = 0.05$ it is also the combination requiring the lower computational time. From Equation (8) and using $\varphi(30) \cong 3750$, the expected number of generated solutions for each epoch are approximately 950.

7. Case Study

7.1. The Benchmarks

To test Pkt_PSO, a set of competitive benchmarks was chosen. Since our algorithms are inspired by the PSO and since PSO often outperforms other metaheuristics when applied to the TSP (see for example [20]), the very recent and highly performant PSO by Zhou et al. [22] (PSO_1) and by Zhong et al. [23] (PSO_2) were considered. We also included in the benchmarks the ACO by De Santis et al. [9]. This is one of the most recent solutions specifically designed for the picking problem in manual warehouses, and it has been shown to perform better than other high-performance algorithms. Lastly, as simple benchmarks, we also considered the basic solution provided by the nearest neighbour constructive algorithm and by the full-depth 2-Opt, the same algorithms used by Pkt_PSO to generate the greedy solution and to perform the neighbour search. For a more in-depth description of the selected benchmarks, please see Appendix A.

7.2. Layouts and Picking Lists

Three different layouts that differ in terms of the disposition and number of blocks and rack were considered. For the sake of clarity, Figure 8 shows what we mean by blocks and racks.

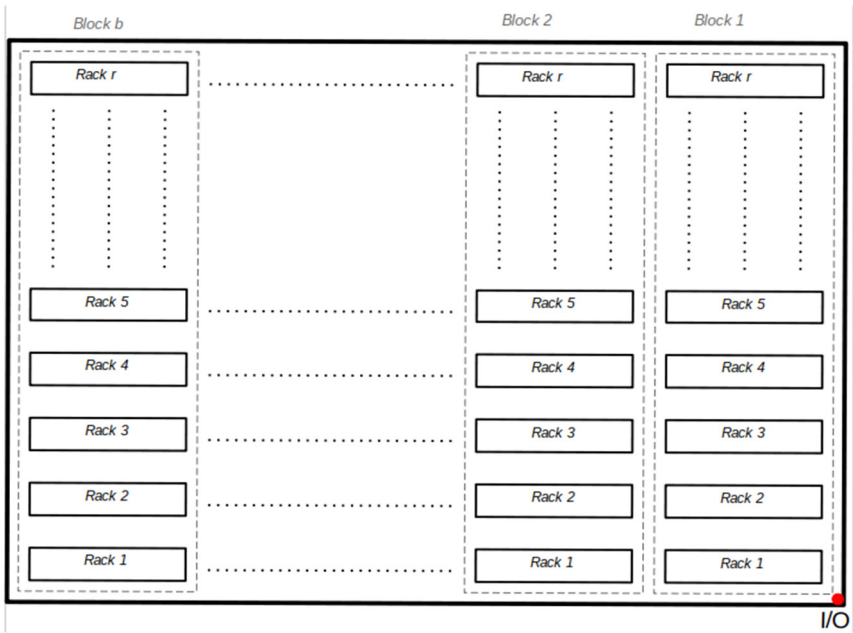

Figure 8. Template of the layouts adopted.

The first layout (L1) is a very long warehouse with no cross aisles, made of a single block with 30 racks divided by 32 aisles. The second one (L2) has the same storage capacity as the first, but the racks are arranged differently. Specifically, it is made of three blocks divided by two cross aisles, and, in each block, there are 10 racks divided by 12 aisles. The third one (L3) is a high-capacity warehouse made of three blocks divided by two cross aisles and 30 racks per block, each one divided by 32 aisles. For all considered configurations:

- Each aisle has 11 storage locations on both sides; the only exceptions are the first and the last aisle that, being flanked by the wall, have only 11 storage locations on a single side

- The I/O position is in the bottom-right corner
- The storage locations have a standard size of 2×2 m, while the aisles and cross aisles are, respectively, 2 and 4 m wide.

For each layout, we tested several problems of increasing complexity. To the best of our knowledge, and according to [60], given a routing problem, the number of locations to visit is a good proxy of its complexity. For this reason, on each layout, we tested 10 picking lists of length 20, 10 of length 30, and 10 of length 40. We also verified that with a few locations $M \leq 19$ all the algorithms always converged to the global optimum computed with the Concorde TSP solver. On the other one hand, a number $M > 40$ would be very unusual in a manual warehouse and, due to capacity constraints, certainly unrealistic with a single tour. Lastly, we note that for each generated picking list, we randomly selected the storage locations, to simulate a purely random allocation policy.

7.3. Results

All the algorithms were coded in Python 3.9© using the standard CPython interpreter (to increase speed) and run on a standard PC powered by Intel i7 with an 8-core processor 3.2 GHz and 16 Gb RAM. We also open-sourced the source code and the Jupyter-Notebook© used to run the tests at the following link: <GitHub-dmezzogori/op-pso>. To determine the reliability of the compared approaches, for each layout configuration and picking list, each algorithm was launched five times, for a total of $(3 \times 10 \times 3) \times 5 = 450$ tests performed for each algorithm. Clearly, since the nearest neighbour is a constructive and therefore deterministic algorithm, it was tested only one time for each layout configuration and picking list for a total of $(3 \times 10 \times 3) \times 1 = 90$ tests. For each algorithm and for each run, the total travelled distance of the generated picking tour (i.e., the objective function) and the computational time were recorded. We clarify that we did not calculate the global optimum of each scenario, as this would have taken too long (especially for the 40-locations configuration) and also because: (i) the capability to converge to the real global optimum was previously tested with values of $M \leq 19$ (and even with $M = 30$ for Pkt_PSO) and (ii) all the benchmarks had already been compared with exhaustive procedures and/or with optimization solvers in their original papers, to which we refer for more details. In view of this, comparisons will be made relatively to the overall best solution found by the five algorithms; from here on this solution will be referred to as the *pseudo-optimal* solution.

Obtained results are reported in Table 5 in terms of two performance indicators:

- Number of wins (N. Wins column), the number of times an algorithm found the pseudo-optimal solution (i.e., its solution was better or equal to that provided by the other algorithms).
- Average deviation (Avg. Dev. column) relative to the pseudo-optimal solution, computed as in Equation (10) for layout l, picking list length p and algorithm a
$$Avg_Dev_{(l,p,a)} = \frac{\sum_{i=1}^{10}(result_{(l,p,a,i)} - pseudo_optimal_{(l,p)})}{10}$$
where the index $i \in [1, 10]$ indicates the i-th instance of the 10 randomly generated picking lists of equal length p.

For the sake of clarity, and due to space constraints, the starting data from which the performance indicators were computed (i.e., the length of the picking tours generated by each algorithm) have been omitted from the main body of the article. However, they are included in Appendix B.

For clarity, in Table 6 we highlighted in bold the best result for a given layout and picking list combination, that is, the best result of each row.

A complementary analysis is provided in Table 6 that shows the average computation times in seconds. Note that the nearest neighbour algorithm has not been included in the table, as its running time is negligible, given its simplicity and determinism.

Table 6. Comparison of computational times in seconds.

Layout	Picking List Length	Pkt_PSO	PSO_1 [22]	PSO_2 [23]	ACO [5]	2-Opt
		Avg.	Avg.	Avg.	Avg.	Avg.
L1	20	32.57	0.49	22.56	20.55	0.02
	30	139.44	0.99	44.66	39.02	0.09
	40	410.84	2.03	88.44	53.76	0.36
L2	20	28.47	0.50	24.70	21.15	0.02
	30	113.20	1.17	42.75	34.46	0.11
	40	371.87	2.07	77.05	51.68	0.33
L3	20	32.35	0.53	22.51	22.05	0.02
	30	136.15	1.24	44.52	35.24	0.11
	40	425.26	2.39	74.59	49.94	0.38

7.4. Discussion

Results synthesized in Tables 6 and 7 clearly show that Pkt_PSO is more accurate and precise than the benchmarks. Indeed, as Table 6 reveals, Pkt_PSO always converges to the pseudo-optimal at every run and so the average deviation (of the solution provided by Pkt_PSO with respect to the pseudo-optimal) is always null. This fact incontrovertibly proves that Pkt_PSO is more stable and consistent than the benchmarks and that the added procedures (i.e., aisle-based generation, 2-Opt, and deep search), tailored to the task at hand, tackle the problem structure effectively. Moreover, a consistent convergence to a superior solution and a null average deviation is observed for all the investigated combinations of layout and picking lists. This behaviour is extremely important, especially from an operational viewpoint, as interested practitioners should not bother to fine tune the hyperparameters of the algorithm for the specific layout at hand.

Table 7. 3-way ANOVA to compare performance of ACO and Pkt_PSO, *** denotes a very high significant factor.

	Sum_Sq (Type III)	df	F	*p*-Value
Intercept ***	5.15×10^{7}	1	238,180.2	9.78×10^{-141}
ALG	7.84×10^{1}	1	0.3623	5.49×10^{-1}
LT ***	1.97×10^{6}	2	4542.17	4.36×10^{-83}
PL ***	1.01×10^{6}	2	2351.34	8.64×10^{-72}
(LO:PL) ***	5.32×10^{4}	4	61.54	1.18×10^{-23}
Residuals	1.73×10^{4}	80	NaN	NaN

The only algorithm that shows similar results, although it does not match them, is the PSO_2. This PSO-based algorithm was able to consistently track the optimum in five out of nine configurations. In all the remaining configurations, the deviation from the best solution (found by Pkt_PSO) remained very contained, although not null. Besides being less precise, this algorithm is also less accurate, as it consistently converged to the same (supposedly best) solution at each attempt, only for the third investigated layout (L3). Instead, in all other cases, it found different solutions at every run, as certified by a non-null value of the average deviation. As for the remaining methods, none of them can track the optimum, and the average deviation is always very high.

Concerning the computational times shown in Table 7, we can say that the best operational trade-off is achieved by the PSO_2. This algorithm, in fact, has a low computational time comparable to that of the ACO, which is the fastest, but it provides much better results in terms of total travelled distance. Pkt_PSO, instead, while showing the best results, needs considerably longer computational times, probably due to the application of the 2-Opt procedure with deep search. These computational times are operationally acceptable and could be easily improved by optimizing the code and/or using a low-level programming

language (such as C). Nonetheless, we checked the quality of the solution provided by our algorithm if it stopped after an amount of time equal to that required by the ACO to reach convergence. We performed this comparison for all scenarios described above, with five repetitions each. As shown in Figure 9, which displays the trend of the best solution found by the two algorithms (for problems of increasing complexity), the solutions are practically indistinguishable, with a maximum difference (or error) lower than 1%.

These results were also confirmed by the three-way ANOVA of Table 7, where "layout type" (LT), "picking list length" (PL), and "algorithm type" (ALG) are used as blocking factors. In the ANOVA, we also included the interaction between factor LT and PL; the other ones were excluded because they were non-significant in a preliminary analysis.

As the ANOVA shows, ALG is the only factor that is not significant (with a very high p-value), and so we can conclude that the performances of the two algorithms is indistinguishable. We also note that all the other factors are highly significant, and this is a further demonstration of the correctness of the choices we made to define the complexity of the problem.

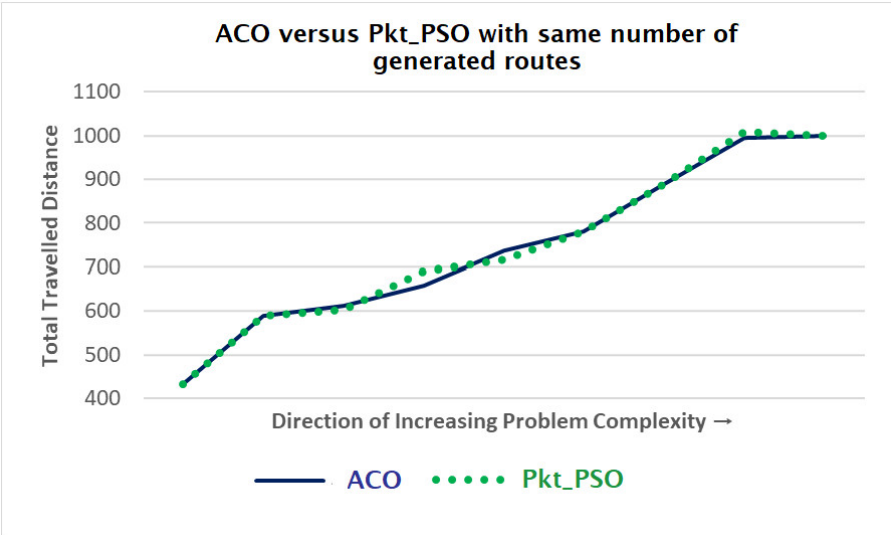

Figure 9. Comparison between the ACO and the proposed PSO, with the same number of generated routes.

8. Conclusions and Future Works

In this paper, we focused on the picking problem in manual warehouses, a very relevant problem with many managerial implications. Considering that a plethora of metaheuristics have been proposed to solve this problem, the paper aims to check if, and how much, the solution can be improved. Furthermore, since the problem is very constrained, it could be also characterized by many points of local minimum with a similar value. If so, at least operationally, a good local minimum could be enough, and it would be unnecessary to use a very sophisticated approach to find the absolute minimum.

To shed light on these research questions, we developed a novel metaheuristic (referred to as Pkt_PSO) and we compared it with a set of state-of-the-art benchmarks. Outcomes of the experimental campaign showed that Pkt_PSO provides the best results, in terms of both accuracy and precision. However, as we initially supposed, the obtained improvements are marginal, and to reach the optimum (or the pseudo-optimum), it is necessary to use an approach that favours exploration over convergence and that requires more computational time. If the execution of Pkt_PSO is interrupted as soon as the benchmarks reach convergence, the obtained results are almost indistinguishable. This finding shows that for this peculiar type of problem, research has practically come to an end, and the use of a general-purpose metaheuristic can be considered enough for many industrial applications. Nonetheless, the use of more accurate approaches could be useful when additional features

and/or constraints are considered, such as dedicated storage, multilevel shelves, one-way aisles, and transport capacity of the picker. The application of Pkt_PSO to these alternative scenarios will be a topic of future works.

An interesting and promising field of application for future research could be the development of a methodology, based on the PSO, to solve the three-dimensional case picking problem (3D-CPP) in an order-picking strategy. 3D-CPP combines two important issues in logistics: the problem of pallet loading and the routing of pickers in manual warehouses. The proposed method could find an extension in the combined optimization of pickers and 3D bin-packing problem.

Author Contributions: M.B.: conceptualization, methodology, supervision. D.M.: software, validation, formal analysis, data curation, writing—original draft preparation. F.Z.: conceptualization, methodology, software, validation, formal analysis, data curation, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the National Recovery and Resilience Plan (NRRP), Mission 04 Component 2 Investment 1.5—NextGenerationEU, Call for tender n. 3277 dated 30 December 2021. Award Number: 0001052 dated 23 June 2022.

Data Availability Statement: All the code and the obtained results are available at the following link GitHub-dmezzogori/op-pso.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

All the abbreviations used in the paper are listed below.

ACO	Ant Colony Optimization,
AS/RS	Automated Storage and Retrieval System,
BSS	Basic Swap Procedure,
CPU	Central Processing Unit,
I/O	Input Output Point,
NaN	Not a Number,
NeNe	Nearest Neighbour,
NP	Non-deterministic Polynomial-time hardness
OF	Objective Function
Pkt_PSO	The proposed metaheuristic, i.e., an evolutionary approach inspired by PSO and designed for picking time optimization.
PSO	Particle Swarm Optimization,
PyTSP	A Python library containing external solvers for the traveling salesman problem,
SA	Simulated Annealing,
SKU	Stock Keeping Unit,
ST	Subjected To (constraints),
TSP	Traveling Salesman Problem,
3D-CPP	Three-dimensional case picking problem.

Appendix A. Description of the Heuristics and Metaheuristics Used as Benchmarks

Appendix A.1. PSO_1 by Zhou et al. [22]

PSO_1 is a PSO specifically designed for the TSP. Its peculiarity lies in the way in which the difference or "*distance*" between two solutions (or tours) is calculated. Let $swap(x, y)$ be the operator that swaps the elements in positions x and y of an array; and let $p_i(t)$ be an array containing the storage locations in their order of visitation. Accordingly, the distance between $p_i(t)$ and a reference solution, say $p_*(t)$, is defined as the minimum number of swaps needed to turn $p_i(t)$ into $p_*(t)$. In this sense, the velocity $v_i(t)$ of particle i can be represented by the *Basic Swap Sequence* (BSS), that is the minimum sequence of $swap(i, j)$ needed to turn $p_i(t)$ into $p_*(t)$. To clarify these concepts a simple example follows, where a zero-based indexing is assumed:

- Solutions: $p_i(t)$ = (4, 5, 1, 3, 2, 6, 7, 8), $p_*(t)$ = (1, 2, 3, 4, 5, 6, 7, 8).
- BSS(($p_i(t) \rightarrow p_*(t)$): [swap(0, 2), swap(1, 4), swap(2, 3)]
- The BSS produces: (1, 5, 4, 3, 2, 6, 7, 8) → (1, 2, 4, 3, 5, 6, 7, 8) → (1, 2, 3, 4, 5, 6, 7, 8).

Overall, the algorithm proceeds as a standard PSO. In fact, at each iteration t, for each particle i, a new position is obtained by applying the velocity vector $v_i(t)$, or BBS, to the current position $p_i(t)$. Next, the velocity is updated as follows:

$$v_i(t+1) = \lceil w \cdot v_i(t) \rceil \odot \lceil \alpha \cdot u \cdot (p_i(t) - pbest_i(t)) \rceil \odot \lceil \beta \cdot u \cdot (p_i(t) - gbest(t)) \rceil \tag{A1}$$

where w, α and β are coefficients, u is a uniformly distributed random number and $\odot$ is the similarity operator.

Relative to Equation (A1), it is important to note that:

- The differences between the current and the best solution (either personal, or global) must be interpreted as distances. Therefore, the result of the difference is a minimal swap sequence.
- The product of a scalar for a swap sequence is interpreted as the *element-wise product* of the scalar with each element of the sequence, e.g., $2 \cdot [(1,\ 2), (3,\ 4)] = [(2,\ 4), (6,\ 8)]$.
- The similarity operator $\odot$ is the average of the elements of a two-swap sequence paired together, e.g., $[(1,\ 2), (3,\ 4)] \odot [(1,\ 5), (2,\ 4)] = [((1+1)/2, (2+5)/2), ((3+2)/2, (4+4)/2)] = [(1,\ 3.5), (2.5,\ 4)]$.
- The ceil function is used to eliminate the floating-point values.

Appendix A.2. PSO_2 by Zhong et al. [23]

PSO_2 makes use of a novel notation, and the classic concept of velocity is deeply modified. Each solution $p_i(t)$ is coded using an edge-based notation, that is a sequence of 2-tuples (A, B) where A and B are storage locations. Each 2-tuple represents an edge included in the solution, and so (A, B) means that storage location B is visited immediately after storage location A. For instance, the sequence $\{A, C, D, B, F, G, E, A\}$ is coded as $\{(A, C), (B, F), (C, D), (D, B), (E, A), (F, G),\ (G, E)\}$, where the 2-tuples are sorted in lexicographical order of their first element. Velocity is coded using the same approach but, in this case, a weight w is associated to each edge, e.g., $\{[(A,\ C),\ 0.5], [(B,\ F),\ 1.0], \ldots, [(X,\ Y),\ w]\}$. Also, the edges included in the velocity vector do not necessarily form a tour; they are just candidate edges that could be inserted in a new solution.

Concerning distances, the distance between two solutions $p_i(t) - p_j(t)$ is given by the sequence of edges of $p_i(t)$ without the edges that are also in $p_j(t)$. This distance is transformed in a velocity by associating to each edge a weight of 1. In other words, the difference of two solutions is interpreted as a speed sequence with all weights equal to one. Similarly, to sum speeds, their edges are compared one by one: if the edges are equal, the respective weights are added together; otherwise, the edge with the highest weight is taken. For instance, taking $v_i(t) = \{[(A,\ C),\ 0.5],\ [(B,\ F),\ 1],\ \ldots\}$ and $v_j(t) = \{[(A,\ C),\ 0.8],\ [(M,\ P),\ 1.2],\ \ldots\}$, their sum equals to $v_i(t) + v_j(t) = \{[(A,\ C),\ 1.3], [(M,\ P),\ 1.2],\ \ldots\}$.

Lastly, at each iteration t, the velocity of particle i is updated as in Equation (A2):

$$v_i(t+1) = W \cdot v_{gr}(t) + u \cdot (p_i(t) - rnd_pbest(t)) \tag{A2}$$

where $v_{gr}(t)$ is a velocity generated using a greedy algorithm, $rnd_pbest(t)$ is the personal best of a randomly selected particle of the swarm, W is a weight factor and u is a random number.

It is important to note that:

- When the velocity is multiplied by a scalar, say x, each edge-weight w is multiplied by x.
- The difference $\left(p_i(t) - rnd_{pbest(t)}\right)$ generates a speed sequence that can be added to $W \cdot v_{gr}(t)$.
- The velocity $v_{gr}(t)$ is iteratively constructed edge by edge. Specifically, for each storage location X, a location Y is randomly extracted (with probability proportional to the distance between X and Y) and a new edge (X, Y), with weight w = 1 is created.

Next, a new solution $p_i(t+1)$ is generated from $p_i(t)$ using one of the edges included in $v_i(t+1)$. Specifically, each edge of $v_i(t+1)$ is considered for possible insertion in $p_i(t)$ and the insertion that generated the best tour is finally selected. Also, any time one of the edges of $v_i(t+1)$ is selected, three different inserting strategies (namely *swap*, *insert* and *inverse*) are tested, and the best one is retained. Lastly, $p_i(t)$ is replaced by $p_i(t+1)$ using a *Metropolis acceptance criterion* akin to the one commonly implemented in Simulated Annealing. In this way, new solutions may be accepted even in the case of a small deterioration of the objective function.

Appendix A.3. ACO by De Santis et al. [9]

This metaheuristic is a classic Ant Colony Optimization, an algorithm that mimics the behaviour of ants looking for food. When an ant leaves the nest, it must decide which is the best way to find some food. To this aim, the ant looks at the quantity of pheromone left by other members of the colony who had previously walked a path. A higher level of pheromone means a better path and vice versa. In fact, a good path will be selected by many ants will receive much pheromone. In contrast, a poor one will be visited by few ants and the pheromone will tend to evaporate. Inspired by this behaviour, at each iteration of the ACO, a new path is built from scratch, adding one location at a time. Let j_1 be the last location included in the new path; the next one, say j_1, is randomly selected with probability $\pi(j_1 \rightarrow j_2)$ that is proportional to the amount of pheromone laid on the edge connecting j_1 to j_2, as shown by Equation (A3):

$$\pi(j_1 \rightarrow j_2) = \frac{\tau_{j_1,j_2}}{\sum_k \tau_{j_1,\mathrm{k}}} \tag{A3}$$

where τ_{j_1,j_2} is the pheromone on edge (j_1, j_2). Therefore, the probability to visit j_2 after j_1 equals the amount of pheromone on edge (j_1, j_2) divided by the sum of the amount of pheromone on all the edges leaving from j_1.

Once the new path has been completed, it is compared to the best solution found so far. In case of improvement, the best solution is updated and the amount of pheromone on each edge of the network is modified as in Equation (A4).

$$\begin{aligned} \tau_{j_1,j_2} &= \left(\tau_{j_1,j_2}\right)^{\alpha} \cdot \left(\frac{1}{d_{j_1,j_2}}\right)^{\beta} && if\ (j_1, j_2) \in best\ path \\ \tau_{j_1,j_2} &= \rho \cdot \tau_{j_1,j_2} && if\ (j_1, j_2) \notin best\ path \end{aligned} \tag{A4}$$

where α, β and ρ are parameters of the algorithm and d_{j_1,j_2} is the distance between two locations.

Appendix B. Experimental Results

The following tables show the results obtained in each evaluated scenario. Specifically, for each combination of layout and picking list, the following data are reported: (i) mean and standard deviation of the total travelled distance, (ii) the number of solutions explored, and (iii) the computational time.

Table A1. Layout L1 and picking lists with 20 locations.

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	644	604	0	1139	23	604	0	618	6	646	13
2	672	608	0	1087	55	608	0	630	4	650	7
3	624	572	0	1074	64	575	2	572	0	582	9
4	708	624	0	1197	50	624	0	643	7	690	6
5	780	600	0	1169	55	600	0	607	3	629	2
6	636	584	0	1143	60	584	0	589	4	634	19
7	732	656	0	1202	45	656	0	660	0	683	24
8	664	604	0	1143	68	604	0	604	0	631	10
9	760	656	0	1212	75	656	0	656	0	670	17
10	824	784	0	1297	47	784	0	793	4	628	2
	Number of Generated Solutions before the Pseudo-Optimum Was Found										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	1	23,945	11,523	7728	6712	16,538	2577	567	496	4617	320
2	1	201,331	88,161	4596	5191	22,321	2349	345	465	5198	587
3	1	59,533	33,890	5464	7367	12,281	1661	500	248	5643	242
4	1	17,008	7124	7296	7174	15,884	3659	722	302	5096	699
5	1	28,070	15,015	11,116	8195	14,009	2436	485	308	4549	624
6	1	102,395	139,718	11,224	8203	17,747	5141	645	546	4685	394
7	1	57,607	22,124	9104	4186	20,626	2326	286	352	4720	893
8	1	36,382	12,408	5896	6770	17,543	3864	113	46	4309	371
9	1	20,437	7274	10,156	6909	11,302	2630	362	405	4036	941
10	1	16,747	10,126	15,180	11,334	16,759	2855	446	277	4891	624
	Computational Time [s]										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	0	37	2	20	4	27	4	1	0	0	0
2	0	36	2	17	3	9	1	0	0	0	0
3	0	27	3	20	4	29	3	1	0	0	0
4	0	33	3	18	6	15	2	1	0	0	0
5	0	33	3	23	7	28	3	0	0	0	0
6	0	30	4	23	3	23	2	1	0	0	0
7	0	36	2	18	3	16	3	0	0	0	0
8	0	31	3	19	5	22	2	0	0	0	0
9	0	33	2	22	5	31	2	0	0	0	0
10	0	31	3	25	7	25	3	0	0	0	0

Table A2. Layout L1 and picking lists with 30 locations.

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	776	716	0	1738	94	716	0	754	7	794	16
2	824	748	0	1766	63	748	0	761	2	762	5
3	848	764	0	1822	109	764	0	787	7	795	18
4	744	672	0	1824	100	672	0	690	5	744	13
5	828	696	0	1778	79	696	0	706	5	843	36

Table A2. *Cont.*

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
6	796	722	5	2011	35	720	0	743	7	818	14
7	792	708	0	1773	93	708	0	724	12	735	23
8	820	740	0	1928	118	740	0	759	2	772	17
9	680	644	0	1645	109	647	4	668	0	694	17
10	804	616	0	1691	30	616	0	623	8	664	0
List	**Number of Generated Solutions before the Pseudo-Optimum Was Found**										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	1	679,224	563,344	14,556	8730	41,545	7362	459	433	20,219	880
2	1	325,733	273,041	15,388	12,760	40,482	2593	169	120	19,488	2798
3	1	494,748	323,066	6384	7468	38,944	8312	611	330	18,838	1303
4	1	436,762	331,037	11,792	13,176	38,669	3910	479	279	22,898	1059
5	1	544,264	357,273	16,060	14,527	36,769	5318	566	560	19,163	1475
6	1	800,584	647,063	11,368	9776	93,652	119,083	571	680	20,381	1328
7	1	537,062	377,383	24,052	13,396	36,747	6869	578	789	20,706	1572
8	1	1,489,469	1,943,847	9096	2781	46,986	13,081	416	320	23,467	2714
9	1	1,453,857	1,491,128	15,172	11,404	39,862	11,840	33	14	20,300	3360
10	1	60,388	25,161	5176	9852	30,578	1902	319	525	17,620	2412
List	**Computational Time [s]**										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	0	145	12	43	13	44	8	1	0	0	0
2	0	152	9	41	12	56	5	1	0	0	0
3	0	129	8	33	10	35	3	1	0	0	0
4	0	141	15	37	14	41	5	1	0	0	0
5	0	134	9	44	20	38	6	1	0	0	0
6	0	148	17	36	10	54	11	1	1	0	0
7	0	145	17	53	19	37	9	1	1	0	0
8	0	144	26	35	6	47	6	1	0	0	0
9	0	143	22	42	14	42	8	1	0	0	0
10	0	113	9	26	8	52	5	1	0	0	0

Table A3. Layout L1 and picking lists with 40 locations.

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	952	880	0	2700	90	881	2	888	3	923	8
2	864	824	0	2378	104	826	4	846	4	895	19
3	776	720	0	2439	74	720	0	720	0	758	20
4	888	796	0	2570	93	796	0	838	9	823	16
5	864	804	0	2659	139	804	0	838	5	922	36
6	896	832	0	2468	70	834	2	864	4	871	11
7	836	736	0	2331	71	736	0	756	8	810	24
8	924	784	0	2490	121	785	2	830	5	844	39
9	912	780	0	2410	89	780	0	817	7	884	24
10	916	764	0	2588	35	764	0	795	14	868	7

Table A3. *Cont.*

List	Number of Generated Solutions before the Pseudo-Optimum Was Found										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	1	5,983,813	7,970,098	6932	5004	72,769	18,190	675	526	59,280	2160
2	1	5,497,910	4,844,449	9576	9256	77,403	7769	1057	491	55,427	1606
3	1	328,813	189,128	9432	10,166	61,593	10,545	869	758	59,132	4550
4	1	2,177,531	2,156,725	3688	3195	74,534	10,015	870	633	58,094	3690
5	1	1,867,387	726,807	4136	3763	90,334	8611	732	458	58,687	5677
6	1	5,059,675	3,423,042	16,528	9372	127,268	102,554	151	92	53,945	3289
7	1	2,023,022	2,128,967	16,172	8641	78,578	10,722	597	768	52,611	6177
8	1	2,164,523	1,124,101	24,212	20,450	113,708	23,875	528	365	57,205	5379
9	1	1,671,421	1,476,398	16,172	13,441	70,229	10,748	511	501	59,576	5435
10	1	1,595,432	939,008	5480	4879	73,193	18,738	898	775	49,202	3420
	Computational Time [s]										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	0	510	109	46	9	88	2	2	1	0	0
2	0	441	77	52	17	78	14	3	1	0	0
3	0	299	31	51	19	75	8	2	1	0	0
4	0	375	55	38	5	66	10	2	1	0	0
5	0	397	22	41	8	85	10	2	1	0	0
6	0	487	35	63	17	91	8	1	0	0	0
7	0	341	35	61	13	66	10	2	1	0	0
8	0	415	24	78	36	84	4	2	0	0	0
9	0	381	34	64	24	90	7	2	1	0	0
10	0	460	46	44	8	81	6	2	1	0	0

Table A4. Layout L2 and picking lists with 20 locations.

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	498	432	0	820	46	432	0	433	3	474	31
2	558	466	0	838	31	466	0	466	0	526	8
3	520	456	0	868	47	456	0	469	7	564	29
4	558	486	0	930	30	486	0	489	4	562	35
5	548	472	0	799	34	472	0	475	4	503	21
6	506	444	0	820	28	444	0	451	6	469	22
7	626	522	0	938	40	522	0	529	7	590	24
8	582	434	0	819	25	434	0	448	5	546	12
9	510	498	0	950	21	498	0	505	5	532	12
10	452	408	0	741	38	408	0	408	0	520	17
	Number of Generated Solutions before the Pseudo-Optimum Was Found										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	1	52,310	58,687	9704	6905	14,610	1785	143	142	4617	419
2	1	24,113	18,633	6796	7234	15,383	1655	190	135	3796	867
3	1	96,898	77,515	19,256	13,976	17,408	2198	753	116	4309	797
4	1	114,025	87,207	9528	7511	17,738	2912	470	734	4480	475
5	1	13,144	6089	14,212	15,269	16,886	3073	435	453	4754	475

Table A4. *Cont.*

List	Number of Generated Solutions before the Pseudo-Optimum Was Found										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
6	1	38,375	43,511	4876	4912	19,273	5007	687	299	3762	342
7	1	23,206	11,193	9976	5606	16,968	3398	433	441	4754	585
8	1	26,716	18,291	10,628	8439	19,822	2067	516	244	4822	709
9	1	775,262	268,161	4688	6543	55,382	79,078	651	544	4549	647
10	1	17,163	20,641	5752	4997	13,858	1652	411	291	4891	732
	Computational Time [s]										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	0	26	3	20	3	19	3	0	0	0	0
2	0	25	2	21	4	26	4	0	0	0	0
3	0	32	4	28	8	26	3	1	0	0	0
4	0	30	3	21	4	23	5	1	0	0	0
5	0	26	2	25	9	29	4	0	0	0	0
6	0	28	0	18	4	19	7	1	0	0	0
7	0	26	1	22	4	25	2	0	0	0	0
8	0	27	1	22	7	28	3	1	0	0	0
9	0	40	5	16	2	25	3	1	0	0	0
10	0	24	2	19	4	27	2	0	0	0	0

Table A5. Layout L2 and picking lists with 30 locations.

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	650	590	3	1319	34	590	0	606	9	694	34
2	706	552	0	1267	31	552	0	555	6	630	22
3	746	594	0	1474	46	594	0	640	9	699	47
4	654	530	0	1400	35	530	0	531	1	624	31
5	608	558	0	1458	46	558	0	569	8	638	23
6	670	548	0	1270	29	548	0	582	12	594	53
7	788	576	0	1438	56	576	0	603	5	628	25
8	732	622	1	1357	38	622	0	652	5	676	31
9	642	568	0	1284	31	568	0	593	16	672	14
10	590	568	0	1411	73	568	0	574	2	617	37
	Number of Generated Solutions before the Pseudo-Optimum Was Found										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	1	626,456	292,699	13,220	15,661	57,943	17,432	362	263	19,488	3105
2	1	170,798	135,804	8072	8371	43,995	4995	737	603	20,544	2702
3	1	130,234	44,252	9020	5166	39,359	5478	919	626	19,894	1770
4	1	557,410	584,354	10,640	5704	41,506	9254	288	302	21,843	1583
5	1	294,690	163,301	18,532	18,042	44,706	9389	967	481	21,843	2231
6	1	405,151	249,705	10,388	12,004	39,117	5640	629	364	21,031	2590
7	1	2,174,014	2,251,594	7964	10,483	59,537	21,114	417	230	19,001	1051
8	1	1,694,219	1,688,462	14,104	9348	40,043	8557	1000	898	19,326	2198
9	1	862,602	633,729	9992	4774	41,338	7219	551	260	20,625	1891
10	1	984,257	843,098	6576	6327	83,557	48,769	612	889	19,732	2894

Table A5. *Cont.*

List	Computational Time [s]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	0	115	12	37	19	51	8	1	0	0	0
2	0	98	5	28	14	43	5	1	0	0	0
3	0	86	6	32	7	29	2	1	0	0	0
4	0	103	10	33	7	43	10	1	0	0	0
5	0	98	10	40	13	39	9	1	0	0	0
6	0	112	8	36	14	38	3	1	0	0	0
7	0	139	35	34	13	35	10	1	0	0	0
8	0	139	35	39	13	58	1	1	1	0	0
9	0	120	20	37	7	43	6	1	0	0	0
10	0	123	11	29	11	48	10	1	1	0	0

Table A6. Layout L2 and picking lists with 40 locations.

List	Distance Covered by the Picker [m]										
	Greedy	Proposed		PSO_1		PSO_2		ACO		2-Opt	
		Avg.	St. Dev.	Avg.	St. Dev.	Avg.	St. Dev.	Avg.	St. Dev.	Avg.	St. Dev.
1	848	640	0	1853	16	640	0	672	26	725	23
2	738	614	0	1833	78	614	0	637	3	741	26
3	802	646	4	1850	35	646	0	670	7	722	44
4	670	626	3	1750	34	626	0	650	4	738	35
5	784	645	3	1849	42	645	3	673	21	768	35
6	660	608	0	1886	18	608	0	635	1	690	18
7	782	650	0	1698	65	650	0	672	15	762	29
8	810	682	4	1781	60	682	0	720	12	814	29
9	680	592	0	1686	19	592	0	602	0	767	17
10	846	730	0	1856	43	732	3	791	29	845	20
	Number of Generated Solutions before the Pseudo-Optimum Was Found										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	1	2,633,156	1,939,483	13,108	14,624	82,471	14,208	722	820	51,722	6340
2	1	920,262	708,087	9032	8670	93,486	20,323	566	452	48,313	3569
3	1	4,597,871	4,668,581	5632	4868	110,048	29,942	556	403	47,869	6279
4	1	7,165,294	4,851,889	13,868	17,083	121,714	30,371	416	226	56,168	2641
5	1	7,555,943	5,344,932	17,652	11,119	122,606	32,045	1466	1017	49,202	3420
6	1	3,257,009	2,299,982	8496	8448	83,373	22,915	368	188	51,129	2723
7	1	3,231,814	3,723,196	12,796	8351	119,858	38,861	680	566	50,092	2320
8	1	6,252,100	4,754,099	7044	14,420	148,304	82,385	1528	868	54,389	4340
9	1	2,123,368	1,589,036	11,508	8632	85,400	23,327	473	458	50,240	5797
10	1	5,311,844	37,19,317	7476	11,650	174,341	160,655	613	598	49,202	1444
	Computational Time [s]										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	0	342	48	54	22	70	13	2	1	0	0
2	0	284	33	50	15	73	11	2	1	0	0
3	0	367	59	43	9	77	8	2	1	0	0
4	0	407	87	58	30	76	6	2	0	0	0
5	0	458	89	64	19	70	5	3	1	0	0

Table A6. *Cont.*

List	Computational Time [s]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
6	0	357	51	48	15	77	2	2	0	0	0
7	0	339	73	52	7	74	12	2	1	0	0
8	0	423	60	46	26	92	8	3	1	0	0
9	0	331	56	56	17	75	10	2	1	0	0
10	0	411	58	46	21	86	12	2	1	0	0

Table A7. Layout L3 and picking lists with 20 locations.

List	Number of Generated Solutions before the Pseudo-Optimum Was Found										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	890	780	0	1512	21	780	0	794	8	831	2
2	974	862	0	1628	74	862	0	902	17	1024	53
3	904	798	0	1620	95	798	0	798	0	920	29
4	1048	866	0	1596	81	866	0	879	8	1055	27
5	880	726	0	1485	67	726	0	737	12	780	8
6	1012	766	0	1482	88	766	0	766	0	878	25
7	846	672	0	1292	21	672	0	688	6	716	49
8	1138	859	3	1569	70	859	3	892	9	938	21
9	992	846	0	1475	71	846	0	880	12	1050	41
10	858	744	0	1467	47	744	0	744	0	855	31
	Solutions Explored before Finding the Best										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	1	85,647	71,148	12,416	11,738	17,900	1528	256	160	5233	902
2	1	306,429	340,821	7736	9082	62,058	85,374	841	1044	4856	1051
3	1	21,422	12,062	10,928	13,780	15,710	1979	332	199	5233	798
4	1	166,403	214,506	11,132	12,206	16,853	6761	853	485	5848	371
5	1	26,491	12,982	17,616	14,150	13,363	2209	266	125	4993	585
6	1	31,337	16,873	8308	7344	15,263	2626	828	441	5096	281
7	1	35,501	16,457	8028	6227	14,380	639	386	258	5301	897
8	1	180,454	219,406	8840	4422	19,466	3930	524	372	4891	877
9	1	150,524	82,364	19,260	9805	17,810	4694	592	394	5198	843
10	1	38,961	40,340	15,196	19,087	16,123	2224	360	49	5027	732
	Computational Time [s]										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	0	38	1	21	8	19	6	0	0	0	0
2	0	35	7	21	5	27	4	1	0	0	0
3	0	28	2	21	6	19	5	0	0	0	0
4	0	35	6	22	6	26	6	1	0	0	0
5	0	29	1	27	6	26	1	0	0	0	0
6	0	31	1	19	6	22	3	1	0	0	0
7	0	30	1	18	2	17	7	0	0	0	0
8	0	29	4	21	2	22	5	1	0	0	0
9	0	34	1	26	4	22	4	1	0	0	0
10	0	34	5	24	9	24	3	0	0	0	0

Table A8. Layout L3 and picking lists with 30 locations.

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	1120	974	0	2280	73	974	0	1008	14	1090	22
2	1024	894	0	2273	122	894	0	919	2	976	15
3	1040	988	0	2245	91	988	0	996	16	1182	14
4	1224	1046	0	2320	74	1046	0	1064	5	1133	32
5	1012	896	0	2084	55	896	0	942	11	998	43
6	1162	1020	0	2400	114	1020	0	1034	11	1204	38
7	918	860	0	2034	136	860	0	862	3	1018	72
8	1194	1046	0	2262	58	1046	0	1073	13	1176	48
9	992	866	0	2097	64	866	0	871	11	944	45
10	1114	916	0	2139	54	916	0	931	17	1026	19
	Number of Generated Solutions before the Pseudo-Optimum Was Found										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	1	590,671	481,109	11,928	10,894	38,020	4271	652	491	22,898	2395
2	1	435,755	519,099	12,400	15,467	36,674	2339	867	423	21,031	1503
3	1	657,692	524,307	9796	12,464	42,730	14,515	1204	545	19,813	4624
4	1	2,289,460	2,121,883	22,968	13,973	45,619	18,402	462	521	20,219	2933
5	1	655,740	450,274	15,368	9165	33,497	6594	398	584	22,330	1968
6	1	446,289	312,469	4376	4060	34,590	6511	695	480	20,056	3179
7	1	339,281	313,093	4520	4391	39,590	5455	881	481	17,133	2684
8	1	847,059	835,719	8068	7488	60,481	32,652	500	361	22,980	2777
9	1	1,458,344	659,167	7252	3667	40,673	7071	1064	940	20,138	3256
10	1	206,184	120,885	6472	7413	31,816	5125	489	345	19,732	2748
	Computational Time [s]										
List	**Greedy**	**Pkt_PSO**		**PSO_1**		**PSO_2**		**ACO**		**2-Opt**	
		Mean	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**	**Mean**	**St. Dev.**
1	0	140	10	38	16	49	5	1	0	0	0
2	0	124	18	34	17	46	8	1	0	0	0
3	0	119	5	32	8	48	7	2	0	0	0
4	0	183	28	51	17	42	9	1	0	0	0
5	0	129	18	42	9	39	3	1	0	0	0
6	0	116	9	28	4	43	3	1	0	0	0
7	0	125	9	26	6	55	4	1	0	0	0
8	0	155	16	35	10	47	3	1	0	0	0
9	0	141	4	34	5	39	9	2	1	0	0
10	0	130	4	33	10	39	9	1	0	0	0

Table A9. Layout L3 and picking lists with 40 locations.

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	1334	990	0	2971	97	990	0	1026	18	1091	63
2	1140	1030	0	3267	120	1030	0	1053	8	1298	62
3	1398	1093	1	3369	108	1092	0	1124	13	1299	32
4	1154	1112	0	3256	57	1112	0	1128	0	1240	13
5	1288	1098	0	3221	111	1098	0	1133	22	1340	34
6	1416	1212	0	3362	93	1212	0	1273	19	1396	34

Table A9. *Cont.*

List	Distance Covered by the Picker [m]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
7	1222	1008	0	2835	99	1008	0	1068	32	1177	21
8	1378	1090	0	3239	90	1090	0	1110	11	1224	18
9	1238	1122	0	3127	58	1122	0	1153	22	1308	39
10	1310	1122	0	3120	132	1122	0	1226	25	1371	24

List	Number of Generated Solutions before the Pseudo-Optimum Was Found										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	1	6,344,181	4,493,413	7204	9484	67,223	10,117	1097	695	59,132	7672
2	1	3,575,121	2,002,846	6732	5041	85,580	25,045	525	446	61,207	4244
3	1	4,764,772	3,897,361	5984	5582	93,930	21,680	338	244	57,650	5629
4	1	3,160,268	1,488,146	20,792	9577	91,401	12,016	608	192	59,132	5430
5	1	968,025	396,864	9884	15,886	68,002	9561	971	616	59,576	3460
6	1	7,802,870	4,103,116	12,012	14,495	125,633	23,729	2337	1369	50,240	3757
7	1	7,585,250	4,333,599	11,004	5338	82,655	17,005	755	691	53,945	2792
8	1	3,608,348	3,960,953	12,396	8536	69,860	11,998	1150	282	57,502	3690
9	1	1,085,203	541,226	2624	1791	62,120	15,526	983	947	54,834	3778
10	1	6,460,064	3,381,169	2972	2258	104,167	21,514	709	386	59,576	3942

List	Computational Time [s]										
	Greedy	Pkt_PSO		PSO_1		PSO_2		ACO		2-Opt	
		Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.	Mean	St. Dev.
1	0	475	92	47	17	84	9	3	1	0	0
2	0	411	56	47	10	78	11	2	1	0	0
3	0	405	60	44	11	68	11	2	0	0	0
4	0	399	40	71	17	72	11	2	0	0	0
5	0	350	57	52	30	65	4	2	1	0	0
6	0	527	81	55	27	69	14	4	2	0	0
7	0	431	73	55	10	70	3	2	1	0	0
8	0	420	49	53	14	71	10	3	0	0	0
9	0	364	37	38	4	77	11	2	1	0	0
10	0	470	61	38	5	92	2	2	1	0	0

References

1. Roodbergen, K.; Vis, I. A model for warehouse layout. *IIE Trans.* **2006**, *10*, 799–811. [CrossRef]
2. Chowdhury, N.A.; Ali, S.M.; Paul, S.K.; Mahtab, Z.; Kabir, G. A hierarchical model for critical success factors in apparel supply chain. *Bus. Process Manag. J.* **2020**, *26*, 1761–1788. [CrossRef]
3. Ma, Z.; Shang, X.; Fu, X.; Luo, F. The Architecture and Key Technologies of Internet of Things in Logistics. In Proceedings of the International Conference on Cyberspace Technology, Beijing, China, 23 November 2013.
4. Jaghbeer, Y.; Hanson, R.; Johansson, M.I. Automated order picking systems and the links between design and performance: A systematic literature review. *Int. J. Prod. Res.* **2020**, *58*, 4489–4505. [CrossRef]
5. Janssen, C.P.; Donker, S.F.; Brumby, D.P.; Kun, A.L. History and future of human-automation interaction. *Int. J. Hum. -Comput. Stud.* **2019**, *131*, 99–107. [CrossRef]
6. Roodbergen, K.J.; Sharp, G.P.; Vis, I.F.A. Designing the layout structure of manual order picking areas in warehouses. *IIE Trans.* **2008**, *40*, 1032–1045. [CrossRef]
7. Van Gils, T.; Ramaekers, K.; Caris, A.; De Koster, R.B.M. Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *Eur. J. Oper. Res.* **2018**, *267*, 1–15. [CrossRef]
8. Koster, R.D.; Le-Duc, T.; Roodbergen, K.J. Design and control of warehouse order picking: A literature review. *Eur. J. Oper. Res.* **2007**, *2*, 481–505. [CrossRef]
9. De Santis, R.; Montanari, R.; Vignali, G.; Bottani, E. An adapted ant colony optimization algorithm for the minimization of the travel distance of pickers in manual warehouses. *Eur. J. Oper. Res.* **2018**, *1*, 120–137. [CrossRef]
10. Van Gils, T.; Ramaekers, K.; Braekers, K.; Depair, B.; Caris, A. Increasing order picking efficiency by integrating storage, batching, zone picking, and routing policy decisions. *Int. J. Prod. Econ.* **2018**, *197*, 243–261. [CrossRef]

11. Bottani, E.; Volpi, A.; Montanari, R. Design and optimization of order picking systems: An integrated procedure and two case studies. *Comput. Ind. Eng.* **2019**, *137*, 1–17. [CrossRef]
12. Henn, S.; Koch, S.; Doerner, K.F.; Strauss, C.; Wäscher, G. Metaheuristics for the order batching problem in manual order picking systems. *Bus. Res.* **2010**, *1*, 82–105. [CrossRef]
13. Zhao, Z.; Zhang, M.; Yang, C.; Fang, J.; Huang, G.Q. Distributed and collaborative proactive tandem location tracking of vehicle products for warehouse operations. *Comput. Ind. Eng.* **2018**, *125*, 637–648. [CrossRef]
14. Casella, G.; Volpi, A.; Montanari, R.; Tebaldi, L.; Bottani, E. Trends in order picking: A 2007–2022 review of the literature. *Prod. Manuf. Res.* **2022**, *11*, 1–71. [CrossRef]
15. Kim, T.Y.; Dekker, R.; Heij, C. Spare part demand forecasting for consumer goods using installed base information. *Comput. Ind. Eng.* **2017**, *103*, 201–215. [CrossRef]
16. Valarmathi, B.; Santhi, K.; Ravi, C.; Peeyush, G.; Bhagyashree, B. Performance analysis of genetic algorithm, particle swarm optimization and ant colony optimization for solving the travelling salesman problem. *Int. J. Recent Technol. Eng.* **2019**, *2*, 2277–3878.
17. Islam, M.R.; Ali, S.M.; Fathollahi-Fard, A.M.; Kabir, G. A novel particle swarm optimization-based grey model for the prediction of warehouse performance. *J. Comput. Des. Eng.* **2021**, *8*, 705–727. [CrossRef]
18. Aouay, S.; Jamoussi, S.; Ayed, Y.B. Particle swarm optimization-based method for Bayesian Network structure learning. In Proceedings of the 5th International Conference on Modeling, Simulation and Applied Optimization, Hammamet, Tunisia, 28–30 April 2013.
19. Cowie, J.; Oteniya, L.; Coles, R. Particle swarm optimization for learning Bayesian networks. In Proceedings of the World Congress on Engineering, WCE, London, UK, 2–4 July 2007.
20. Gharib, A.; Benhra, J.; Chaouqi, M. A performance comparison of PSO and GA applied to TSP. *Int. J. Comput. Appl.* **2015**, *975*, 8887. [CrossRef]
21. Sumathi, M.; Rahamathunnisa, U.; Anitha, A.; Druheen, D.; Nallakaruppan, M.K. Comparison of particle swarm optimization and simulated annealing applied to travelling salesman problem. *Int. J. Innov. Technol. Explor. Eng.* **2019**, *8*, 2278–3075.
22. Zhou, H.; Song, M.; Pedrycz, W. A comparative study of improved GA and PSO in solving multiple traveling salesman problem. *Appl. Soft Comput.* **2018**, *64*, 564–580. [CrossRef]
23. Zhong, Y.; Lin, J.; Wang, L.; Zhang, H. Discrete comprehensive learning particle swarm optimization algorithm with Metropolis acceptance criterion for traveling salesman problem. *Swarm Evol. Comput.* **2018**, *42*, 77–88. [CrossRef]
24. Lawler, E.L.; Lenstra, J.K.; Kan, A.R.; Shmoys, D.B. *The Traveling Salesman Problem, A Guided Tour to Combinatorial Optimization*; John Wiley and Sons: Hoboken, NJ, USA, 1985.
25. Le Bouthillier, A.; Crainic, T.G.; Kropf, P. A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intell. Syst.* **2005**, *20*, 36–42. [CrossRef]
26. Gue, K.R.; Meller, R.D.; Skufca, J.D. The effects of pick density on order picking areas with narrow aisles. *IIE Trans.* **2006**, *38*, 859–868. [CrossRef]
27. Battini, D.; Glock, C.H.; Grosse, E.H.; Persona, A.; Sgarbossa, F. Human energy expenditure in order picking storage assignment: A bi-objective method. *Comput. Ind. Eng.* **2016**, *94*, 147–157. [CrossRef]
28. Chen, F.; Wang, H.; Qi, C.; Xie, Y. An ant colony optimization routing algorithm for two order pickers with congestion consideration. *Comput. Ind. Eng.* **2013**, *1*, 77–85. [CrossRef]
29. Önüt, S.; Tuzkaya, U.R.; Doğaç, B. A particle swarm optimization algorithm for the multiple-level warehouse layout design problem. *Comput. Ind. Eng.* **2008**, *4*, 783–799. [CrossRef]
30. Ratliff, H.; Rosenthal, A. Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem. *Oper. Res.* **1983**, *3*, 507–521. [CrossRef]
31. Pansart, L.; Catusse, N.; Cambazard, H. Exact algorithms for the order picking problem. *Comput. Oper. Res.* **2018**, *100*, 117–127. [CrossRef]
32. Koster, R.D.; Van der Poort, E. Routing order pickers in a warehouse: A comparison between optimal and heuristic solutions. *IIE Trans.* **1998**, *30*, 469–480. [CrossRef]
33. Masae, M.; Glock, C.H.; Gross, E.H. Order picking routing in warehouses: A systematic literature review. *Int. J. Prod. Econ.* **2020**, *224*, 107564. [CrossRef]
34. Roodbergen, K.J.; Koster, R. Routing methods for warehouses with multiple cross aisles. *Int. J. Prod. Res.* **2001**, *9*, 1865–1883. [CrossRef]
35. Çelk, M.; Süral, H. Order picking under random and turnover-based storage policies in fishbone aisle warehouses. *IIE Trans.* **2014**, *3*, 283–300. [CrossRef]
36. Calzavara, M.; Glock, C.H.; Gross, E.; Sgarbossa, F. An integrated storage assignment method for manual order picking warehouses considering cost, workload and posture. *Int. J. Prod. Res.* **2019**, *57*, 2392–2408. [CrossRef]
37. Lu, W.; McFarlane, D.; Giannikas, V.; Zhang, Q. An algorithm for dynamic order-picking in warehouse operations. *Eur. J. Oper. Res.* **2016**, *1*, 107–122. [CrossRef]
38. Altazari, S.A.; Ammouri, M.M. Concurrent manual-order-picking warehouse design: A simulation-based design of experiments approach. *Int. J. Prod. Res.* **2018**, *56*, 7103–7121. [CrossRef]

39. Caron, F.; Marchet, G.; Perego, A. Routing policies and COI-based storage policies in picker-to-part systems. *Int. J. Prod. Res.* **1998**, *36*, 713–732. [CrossRef]
40. Petersen, C. An evaluation of order picking routing policies. *Int. J. Oper. Prod. Manag.* **1997**, *17*, 1098–1111. [CrossRef]
41. Petersen, C. The impact of routing and storage policies on warehouse efficiency. *Int. J. Oper. Prod. Manag.* **1999**, *19*, 1053–1064. [CrossRef]
42. Rao, S.S.; Adil, G.K. Optimal class boundaries, number of aisles, and pick list size for low-level order picking systems. *IIE Trans.* **2013**, *45*, 1309–1321. [CrossRef]
43. Silva, A.; Roodbergen, K.J.; Coelho, L.C.; Darvish, M. Estimating optimal ABC zone sizes in manual warehouses. *Int. J. Prod. Econ.* **2022**, *252*, 108579. [CrossRef]
44. Pardo, E.; Gil-Borrás, S.; Alonso-Ayuso, A.; Duarte, A. Order batching problems: Taxonomy and literature review. *Eur. J. Oper. Res.* **2023**, *in press*. [CrossRef]
45. Vanheusden, S.; van Gils, T.; Ramaekers, K.; Cornelissens, T.; Caris, A. Practical factors in order picking planning: State-of-the-art classification and review. *Int. J. Prod. Res.* **2023**, *61*, 2032–2056. [CrossRef]
46. Floyd, R.W. Algorithm 97: Shortest path. *Commun. ACM* **1962**, *5*, 345–354. [CrossRef]
47. Warshall, S. A theorem on Boolean matrices. *J. ACM* **1962**, *9*, 11–12. [CrossRef]
48. Eberhart, R.; Kennedy, J. A New Optimizer Using Particle Swarm Theory. In Proceedings of the 6th International Symposium on Micro Machine and Human Science, Nagoya, Japan, 4–6 October 1995.
49. Banks, A.; Vincent, J.; Anyakoha, C. A review of particle swarm optimization. Part I: Background and development. *Nat. Comput.* **2007**, *6*, 467–484. [CrossRef]
50. Banks, A.; Vincent, J.; Anyakoha, C. A review of particle swarm optimization. Part II: Hybridization, combinatorial, multicriteria and constrained optimization, and indicative applications. *Nat. Comput.* **2008**, *7*, 109–124. [CrossRef]
51. Jain, N.K.; Nangia, U.; Jain, J. A review of particle swarm optimization. *J. Inst. Eng. Ser. B* **2018**, *99*, 407–411. [CrossRef]
52. Asghari, M.; Afshari, H.; Mirzapour, S.M.J.; Fathollahi-Fard, A.M.; Dulebenets, M. A Pricing and advertising decisions in a direct-sales closed-loop supply chain. *Comput. Ind. Eng.* **2022**, *171*, 108439. [CrossRef]
53. Shi, Y.; Eberhart, R. A modified particle swarm optimizer. In Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, Word Congress on Computational Intelligence, Anchorage, AK, USA, 4–9 May 1998; pp. 69–73.
54. Clerc, M.; Kennedy, J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput.* **2002**, *6*, 58–73. [CrossRef]
55. Al-Awami, A.T.; Zerguine, A.; Cheded, L.; Zidouri, A.; Saif, W. A new modified particle swarm optimization algorithm for adaptive equalization. *Digit. Signal Process.* **2011**, *21*, 195–207. [CrossRef]
56. Kennedy, J.; Eberhart, R.C. A discrete binary version of the particle swarm algorithm. In Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, Orlando, FL, USA, 12–15 October 1997; pp. 4104–4108.
57. Laskari, E.C.; Parsopoulos, K.E.; Vrahatis, M.N. Particle swarm optimization for integer programming. In Proceedings of the 2002 Congress on Evolutionary Computation, Honolulu, HI, USA, 12–17 May 2002; pp. 1582–1587.
58. Shi, X.H.; Liang, Y.C.; Lee, H.P.; Lu, C.; Wang, Q.X. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Inf. Process. Lett.* **2007**, *103*, 169–176. [CrossRef]
59. Chen, W.N.; Zhang, J.; Chung, H.S.; Zhong, W.L.; Wu, W.G.; Shi, Y.H. A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Trans. Evol. Comput.* **2009**, *14*, 278–300. [CrossRef]
60. Wang, K.P.; Huang, L.; Zhou, C.G.; Pang, W. Particle swarm optimization for traveling salesman problem. In Proceedings of the 2003 International Conference on Machine Learning and Cybernetics, Xi'an, China, 2–5 November 2003; pp. 1583–1585.
61. Zhong, W.H.; Zhang, J.; Chen, W.N. A novel discrete particle swarm optimization to solve traveling salesman problem. In Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007; pp. 3283–3287.
62. Grasas, A.; Juan, A.A.; Faulin, J.; de Armas, J.; Ramalhinho, H. Biased randomization of heuristics using skewed probability distributions: A survey and some applications. *Comput. Ind. Eng.* **2017**, *110*, 216–228. [CrossRef]
63. Juan, A.A.; Faulin, J.; Ruiz, R.; Barrios, B.; Caballé, S. The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem. *Appl. Soft Comput.* **2010**, *10*, 215–224. [CrossRef]
64. Theys, C.; Bräysy, O.; Dullaert, W.; Raa, B. Using a TSP heuristic for routing order pickers in warehouses. *Eur. J. Oper. Res.* **2010**, *200*, 755–763. [CrossRef]
65. Croes, G.A. A method for solving traveling salesman problems. *Oper. Res.* **1958**, *6*, 791–812. [CrossRef]
66. Englert, M.; Rglin, H. Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. *Algorithmica* **2014**, *68*, 190–264. [CrossRef]
67. Mulder, S.A.; Wunsch, D.C. Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. *Neural Netw.* **2003**, *16*, 827–832. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.