The order picking problem under a scattered storage policy

Maastricht University

Farzaneh Rajabighamchi,
Stan van Hoesel, Christof Defryn

The order picking problem under a scattered storage policy

RM/23/006

ISSN: 2666-8807

GSBE

Maastricht University School of Business and Economics
Graduate School of Business and Economics

P.O Box 616
NL- 6200 MD Maastricht
The Netherlands

The order picking problem under a scattered storage policy

Farzaneh Rajabighamchi, Stan van Hoesel, and Christof Defryn

Maastricht University, School of Business and Economics, Maastricht, The Netherlands

{*f.rajabighamchi; s.vanhoesel; c.defryn*}*@ maastrichtuniversity.nl*

May 1, 2023

Abstract

When warehouses are operated according to a scattered storage policy, each Stock Keeping Unit(SKU) is stored at multiple locations inside the warehouse. Such a configuration allows for improved picking efficiency, as now an SKU can be picked from the location that is most compatible with the other SKU's in the picking batch. Seizing these benefits, however, comes at the cost of additional decisions to be made while planning the picking operations. Next to determining the sequence in which SKU's will be retrieved from the warehouse, the location at which each SKU needs to be extracted has to be chosen by the planner. In this paper, we model the order picking problem under a scattered storage policy as a Generalized Travelling Salesperson Problem (GTSP). In this problem, the vertices of the underlying graph are partitioned into clusters from which exactly one vertex should be visited in each cluster. In our order picking application, each cluster contains all product locations of a single SKU on the order list. The aim is to design a pick tour that visits all product locations of the SKU's on the pick list (i.e., visit each cluster exactly once) and minimizes the total travel distance. We present an ILP formulation of the problem and a variable neighbourhood heuristic, embedded in a guided local search framework. The performance of both methods is tested extensively by means of computational experiments on benchmark instances from the literature.

Keywords: Generalized Travelling Salesperson Problem, Order Picking, Variable Neighbourhood Search, Guided Local Search

1 Introduction and research context

Warehouses are crucial to the effectiveness of physical product supply chains. Even despite real-time data availability and exchange in today's supply chains — which could facilitate a just-in-time paradigm — warehouses play a key role in assuring high customer service levels by balancing supply and demand, and providing a buffer to absorb disruptions and uncertainty. [28]. In the last few years, the steep growth in e-commerce led to a significant increase in warehouse capacity [24, 15, 52]. The performance of these warehouses depends on the efficiency at which all warehouse operations are organized. Out of all warehouse operations, order picking — defined as the process of retrieving a given set of products (in the remainder of the paper referred to as SKUs) from storage locations or buffer areas in response to incoming customer orders [22, 85] — is the most time and labour consuming [86, 67, 58, 91]. Order picking activities include the setup of the pick tours for the individual pickers (either manual pickers or automated systems), travelling across the warehouse to visit all corresponding SKU locations, searching for the correct SKUs at these locations, and extract the correct amount of the requested SKUs.

The travelling between SKU locations — essentially not a value-adding process — accounts for around 50% of all the time spent in the order picking process [83]. Consequently, the reduction of this travel time (eventually in combination with an optimal warehouse layout and SKU allocation) has been the main focus of many research contributions (see e.g., Kapou et al. [40], Lee, Chung, and Yoon [49], De Koster, Le-Duc, and Roodbergen [22], and Accorsi, Manzini, and Bortolini [1]).

The problem of finding the optimal pick tour (i.e., a minimum cost / length tour that starts and ends at the depot and visits all SKU locations from the current pick list) is closely related to a variant of the Travelling Salesperson Problem (TSP). Often, a Steiner TSP formulation is used, in which aisle intersections are added as intermediary nodes that could (but should not) be part of the tour.

In this paper, we study the order picking problem in combination with a scattered storage policy [35]. As nowadays warehouse operations are commonly supported by warehouse management software, there is no real need to group all SKUs of a certain type at the same storage location. Incoming SKUs can be put away at any available storage location from which they can be retrieved once ordered. Consequently, SKUs are *scattered around the different storage locations* and each SKU becomes available at different locations within the warehouse.

Under such scattered storage policy, the construction of the pick tours combines the

decision on the storage location from which the required SKUs will be picked with the generation of optimal pick tours in which these selected locations are visited. As out of all available storage locations for each SKU the decision maker can select the location that is *most compatible* with that of all other SKUs to be picked within the same tour, more efficient pick tours can be constructed.

This paper contributes to the academic literature on warehouse operations in the following ways:

- Denoting all storage locations from which an SKU can be picked as a cluster from which exactly one location should be visited to pick the corresponding SKU, we formulate the order picking problem under a scattered storage policy as a Generalized Travelling Salesperson Problem (GTSP).
- We present an improved ILP formulation for the GTSP and prove its performance against state-of-the-art formulations by means of extensive computational experiments.
- To allow for scalability and solve larger problem instances fast, we present a guided local search heuristic for which we obtain very competitive results.

The remainder of the paper is organized as follows: we discuss the relevant literature in section 2. In section 3 we formally define the generalized traveling salesman problem for the order picking problem under a scattered storage policy. Section 4 details our Guided Local Search algorithm for solving the problem after which we test the performance of our formulations and benchmark our heuristic solution approach against the current state-of-the-art in Section 5. Finally, we summarize our main conclusions in Section 6.

2 Literature Review

2.1 The order picking problem

Due to its importance in the domain of warehouse operations, the order picking problem has received significant attention in the past decades [58, 66, 2, 87]. The main body of research on the modelling of order picking problems relies on the mathematical formulation of the Travelling Salesperson Problem (TSP) or the or the capacitated vehicle routing problem. We divide the existing literature in contributions that focus on exact models and those that present heuristics.

Almost all of the exact algorithms addressed in the literature for the order picker routing problem, are applied in single-block warehouses. Ratliff and Rosenthal [69] is one of the primitive studies on the optimal order picker routing problem. In this study, the authors introduce a dynamic programming approach for a warehouse layout with only one block (i.e., no cross aisle), which is polynomial in the number of items and aisles. This approach is extended for a two-block warehouse (i.e., one cross aisle) in Roodbergen and De Koster [70] and to a multi-block warehouse (i.e., more than one cross aisle) in Cambazard and Catusse [12]. There exist several extensions to the algorithm proposed by [69], each accounting for slight variations in the problem definition. One of these extensions is the study by Žulj et al. [95]. Here, the authors account for precedence constraint with respect to the picking sequence of the SKUs, based on, e.g., their weight, category, etc. Celik and Süral [16] add additional turn penalties to model that account for the loss of time (speed reduction) each time a picker changes direction within a picking aisle.

The studies by Letchford, Nasiri, and Theis [50], Scholz et al. [74], and Pansart, Catusse, and Cambazard [67] rely on the definition of the Steiner TSP to model the order picker routing problem. In these models, the warehouse is represented by a graph in which the nodes are the union of all SKU storage locations, the depot and all aisle intersections (as these are the main decision points on the route of the picker).

Chabot et al. [17], Irnich, Toth, and Vigo [39], Scholz et al. [74], and Glock and Grosse [27] model the order picking problem as a vehicle routing problem which they solve using branch-and-cut.

For the online order picking problem (i.e., new customer orders arrive over time) Lu et al. [57], Cambazard and Catusse [12], Matusiak et al. [61], and Masae, Glock, and Vichitkunakorn [59] present a dynamic programming formulation.

Apart from these exact methods — which quickly become intractable for increasing instance sizes — a wide range of heuristics and metaheuristics have been developed to solve the order picking problem. According to Masae, Glock, and Grosse [58], (meta)heuristic algorithms account for around 85% of all contributions on the order picker problem.

The best known heuristic algorithms are the *S-shape heuristic*, the *largest gap heuristic* and the *midpoint routing method* [31, 14]. Hybrid extensions of these algorithms have been suggested by Chabot et al. [17], Menéndez et al. [62], Matusiak, De Koster, and Saarinen [60], and Chen, Xu, and Wei [18].

Scholz and Wäscher [73], Theys et al. [82], and Hsieh and Tsai [37] rely on a TSP formulation of the order picking problem which they solve using the well-known *Lin-Kernighan*

(LKH) heuristic [56, 32].

Pferschy and Schauer [68] consider different starting and ending points for eacch pick tour and proposed three heuristics based on different insertion methods combined with a 3-opt local search.

In the majority of the cases, (meta)heuristics have been employed to solve combined problems that integrate order picking with order batching (i.e., group multiple orders together to be picked in a single pick tour, especially useful in e-commerce environments where order sizes are rather small) and batch sequencing (i.e., in which order should the constructed batches be picked such that average or worst response times are minimized). Examples include *particle swarm optimization* [54, 5], *memetic algorithms* [10, 10], *ant colony optimization* [20, 51, 23, 19, 18], *genetic algorithms* [94, 4] and *tabu search* [21].

2.2 The Generalized Travelling Salesperson Problem

The Generalized Travelling Salesperson Problem (GTSP) is defined as an extension of the Travelling Salesperson Problem in which the nodes are partitioned into clusters. To serve a cluster it is sufficient that exactly one node is the cluster is visited. The aim is to find a minimum cost / length tour that visits each cluster exactly once.

The GTSP is first introduced by [34]. Since then, the problem has been used to model a wide range of real-life problems [44], such as scheduling problems [7], vehicle routing problems [90], manufacturing problems [80, 36, 45] and telecommunication network design [8].

To solve the GTSP exactly, the problem can be converted into a TSP using dynamic programming, after which it can be solved using dedicated TSP solvers [65]. In the transformed graph, each arc represents the shortest path between each pair of product locations from the original graph, changing the problem into a clustered TSP with a fully connected directed graph, which is afterwards altered into a standard TSP. Such a transformation, however, drastically increases the problem's dimension (in some cases by a factor three or more) [48, 53].

Baniasadi et al. [6] develop a transformation method that is able to convert the Clustered GTSP to a TSP. A Clustered GTSP is defined as a GTSP where in each cluster is further subdivided in multiple sub-clusters that have to be visited consecutively. As the Clustered GTSP reduces to the GTSP with only one sub-cluster per cluster, the transformation method can be applied to turn the GTSP into a TSP instance.

[47, 46] propose a branch-and-bound algorithm to solve the GTSP. In [47] the authors

present a first integer programming formulation for the symmetrical GTSP. In Laporte, Mercure, and Nobert [46], the model is extended to the asymmetrical GTSP. In Salman, Ekstedt, and Damaschke [72] a branch-and-bound algorithm is combined with dynamic programming to solve the GTSP with precedence constraints (i.e., some locations have to visited before others) after which they present a comparison between different bounding methods for this problem.

A branch-and-cut strategy is applied by Fischetti, Salazar González, and Toth [26] for the symmetric GTSP. The authors solve a series of LP relaxations while adding valid inequalities to tighten the lower bound using a heuristic algorithm.

[64] develop a *Lagrangian based approach* to transfer the asymmetric GTSP into an asymmetric TSP. Relying on the principles of the Lagrangian relaxation — which removes the flow balancing constraints and adds the corresponding terms to the objective function, making sure that the optimality conditions of the original problem remains — a lower bound to the problem is computed. To find an upper bound, a heuristic algorithm is employed that removes arcs and nodes. This is done by computing the optimal dual solution, the reduced arc costs and their effect on the objective value.

A broad range of heuristic algorithms have been proposed for the GTSP. Similar to what we see for the exact methods, a first group of heuristics relies on a transformation of the GTSP into an asymmetric TSP, e.g. by means of a *Noon-Bean transformation* [9]. Amongst others, the Noon-Bean transformation is used in Helsgaun [33] after which the obtained TSP is solved using the Lin–Kernighan heuristic [56].

Karapetyan and Gutin [42] propose an adaptation of the Lin–Kernighan heuristic for solving the GTSP with non-overlapping clusters directly, by rearranging the path, breaking the path and improving the tour.

Multiple researchers rely on a local-search based algorithm for solving the GTSP by applying the 2-Opt, 3-Opt, and k-Opt operators [55, 11, 92, 43, 30].

Hu and Raidl [38] present a variable neighbourhood search including a generalized 2-opt neighborhood to speed up the search and node exchanges based on the Lin-Kernighan heuristic.

[78] propose a large neighbourhood search based on repeated worst removal and the cheapest insertion of the vertices from and into the tour. In each iteration, one would look for the removal operation (i.e., take out one node in the tour) that reduces the tour length the most. The node is then inserted at the position where it increases the tour length the least.

Other heuristics that have been developed for the GTSP include genetic algorithms [93, 79, 77], memetic algorithms [29, 11], Particle Swarm Optimization [76] and Ant Colony Optimization [92].

Existing models and algorithms developed for the GTSP mainly rely on the implicit assumption that clusters are defined geographically, i.e., clusters do not overlap in the geographical space [25]. Consequently, inter-cluster distances (defined as the distance between two sets of points) can be used as a proxy for the distance between two arbitrary vertices from different clusters and a tour can be constructed first at the cluster level.

Grouping different SKU locations within a warehouse in the same cluster, however, violates this principle and will give rise to an *overlapping graph*. El Krari et al. [25] define overlapping clusters as those that share a geographical space (i.e., when drawing their borders, they find themselves intertwined). To the best of knowledge, only Nalivajevs and Karapetyan [63] consider the order picking problem within a scattered storage policy. The authors developed an instance generator and propose a *conditional Markov chain search* that combines different algorithmic algorithms to solve the problem at hand. However, the scope of this algorithm is still limited with only limited operators that are presented mainly as a black box. Also, the instances are limited to single-block warehouse layouts making them not very representative for today's warehouses.

3 A Generalized TSP formulation for order picking under a scattered storage policy

In this section, we formally define the order picking problem under a scattered storage policy. We first introduce the mathematical notation associated with the warehouse layout. Then, we model the problem as a generalized TSP. Finally, we present an improved mathematical formulation for the GTSP.

3.1 Representation of the warehouse layout

We assume a traditional multi-parallel aisle (rack and shelf) warehouse that contains a predefined number of pick aisles where SKUs are stored. Additionally, the warehouse contains a given number of intersecting cross aisles that do not contain any SKUs but can be used by the picker to travel efficiently between different pick aisles. As such, the warehouse is divided in a number of blocks, defined as a row of pick aisles between two

cross aisles. Moreover, there is a depot located in the front of the warehouse where the picker starts and ends each pick tour (the depot could be seen as the packing area where all picked items are collected and prepared for shipment).

Adopting the common assumptions proposed by Pansart, Catusse, and Cambazard [67], all aisles have equal lengths and are sufficiently narrow such that pickers can pick from both sides while traversing the aisle (i.e., there is no additional delay for changing the pick side). Additionally, pickers can travel through the aisles in any direction and are able to change direction within the aisles if preferred.

Let $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ be the graph representation of the warehouse, in which $\boldsymbol{V}$ denotes the set of all vertices (SKU locations and the depot) and $\boldsymbol{E}$ is the set of all edges. The depot, included in the set $\boldsymbol{V}$ is denoted by "0". For all $(i, j) \in \boldsymbol{E}$, d_{ij} denotes the distance between storage locations i and j.

Let $\boldsymbol{K}$ be the set of unique SKUs in the warehouse. To represent the storage locations that contain the same SKU, $\boldsymbol{V}$ is partitioned into $|\boldsymbol{K}| + 1$ disjoint subsets (one for each SKU and another one that contains solely the depot). These subsets are referred to as *clusters* and denoted by $\boldsymbol{C}_k$ with $k = \{0, 1, \ldots, |\boldsymbol{K}|\})$ — such that $\boldsymbol{V} = \bigcup_k \boldsymbol{C}_k$ and $\boldsymbol{C}_l \cap \boldsymbol{C}_k = \emptyset \mid l, k \in \{0, 1, \ldots, |\boldsymbol{K}|\}$, $l \neq k$. Also, given that each SKU can be picked from at least one storage location, $|\boldsymbol{C}_k| \geq 1$.

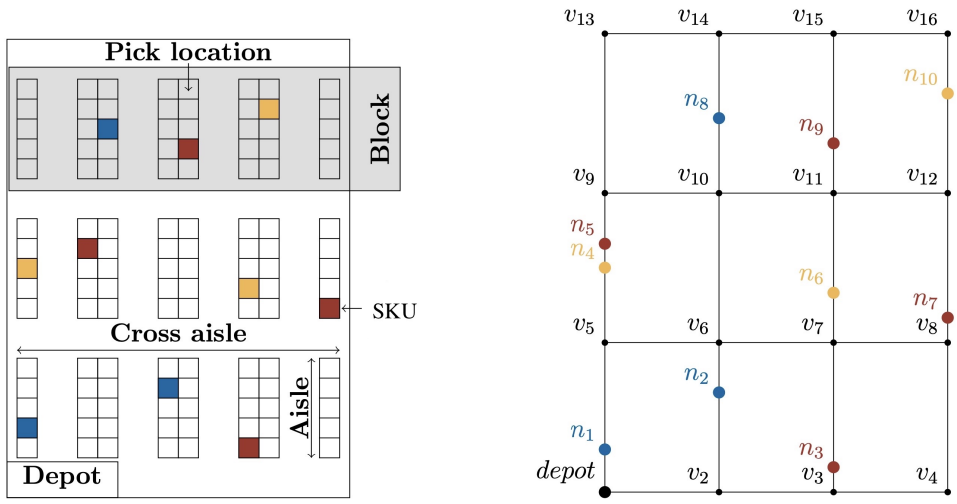
Based on the pick list containing all the SKUs that should be picked in the current tour, the warehouse representation is reduced by eliminating all SKU locations and the corresponding clusters that should not be visited by the picker. Let $\bar{\boldsymbol{G}} = (\bar{\boldsymbol{V}}, \bar{\boldsymbol{E}})$ be the reduced graph in which $\bar{\boldsymbol{V}}$ is the set of all storage locations of the SKUs that should be picked (including the depot) and $\bar{\boldsymbol{E}}$ the set of all edges connecting the vertices in $\bar{\boldsymbol{V}}$.

Figure 1 contains a visual representation of a typical warehouse layout next to its graph representation. Each colour represents an SKU that should be picked from the warehouse.

3.2 Definition of the path between two SKU locations

Let (x_i, y_i) be the coordinates of SKU location i, $(i = 1, \ldots, |\bar{\boldsymbol{V}}|)$. Without loss of generality and to simplify notation we assume the direction of x and y align with the cross aisle and pick aisle, respectively. To travel between two SKU locations i and j, the picker will always prefer the shortest path. We distinguish the following two scenarios:

1. If i and j are located in a different block, the shortest distance equals the Manhattan distance between both coordinates.



(a) Warehouse Layout (b) Graph Representation

Figure 1: *Warehouse layout and graph representation (adapted from Roodbergen [71]). Each colour represents an SKU that should be picked from the warehouse. As multiple locations have the same colour, it is sufficient to visit one location for each colour.*

$$d_{ij} = \mid x_i - x_j \mid + \mid y_i - y_j \mid$$

2. If i and j are located in the same block, the shortest path goes via the cross aisle adjacent to the block. The distance is then computed as follows:

$$d_{ij} = \mid x_i - x_j \mid + \min\{c_i^+ + c_j^+, c_i^- + c_j^-\}$$

where c_i^+ and c_i^- refer to the distance between SKU location i and its upper or lower corner point (i.e., intersection with the cross aisle), respectively. The notation is visualized in Figure 2

As we assume a picker will pick each SKU only from a single storage location (i.e., there is always sufficient inventory at each SKU location to satisfy the demand), all d_{ij} for which i and j contain the same SKU could be ignored.

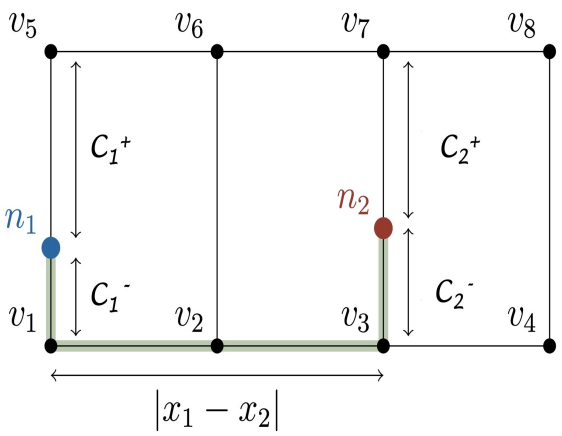


Figure 2: *Distance of vertices in the same block, but different aisles*

3.3 Existing mathematical model formulations

3.3.1 DFJ formulation of the GTSP

The first integer linear programming formulation of the GTSP was proposed by [47] using the Dantzig–Fulkerson–Johnson (DFJ) model. They consider the case where at least one node from each cluster must be visited. They proved that for the Euclidean distances case exactly one node from each cluster is visited, using the fact that the distances satisfy the triangle inequality. This inequality also holds in our case, thus we can conclude that also in our case exactly one of the locations of each cluster is visited. The Dantzig–Fulkerson–Johnson (DFJ) formulation of [47] is presented below.

In the order picking problem, the nodes refer to the product locations in the warehouse and the edges refer to the shortest path between two nodes. The order picker starts his cycle from the depot and after visiting each cluster exactly once and collecting the products present in the order list, by selecting the shortest route, he returns to the depot.

Given the warehouse representation and a list of storage locations to be visited by the picker, let x_{ij} be a binary decision variable denoting whether the edge $(i, j) \in \bar{\boldsymbol{E}}$ is traversed by the picker. Moreover, we will use y_i to model whether vertex $i \in \bar{\boldsymbol{V}}$ is visited by the picker. To ensure that all required SKUs are picked, exactly one y_i variable should be set to 1 in each cluster.

The set of variables and parameters used in mathematical formulations for the GTSP using integer linear programming is described as follow:

Table 1: *Mathematical notation for the DFJ and MTZ model.*

Sets and indices	
$\boldsymbol{V}$	Set of all the nodes (all storage locations of SKUs) in the graph, $V = \{0, 1, 2, ..., n\}$. The depot node is denoted by 0.
$\bar{\boldsymbol{V}} \subseteq \boldsymbol{V}$	The set of all storage locations of SKUs on the picking list, including the depot.
$\boldsymbol{K}$	The set of all SKUs in the warehouse.
$\bar{\boldsymbol{K}} \subseteq \boldsymbol{K}$	The set of all SKUs on the picking list.
$\boldsymbol{T} \subseteq \bar{\boldsymbol{V}}$	Non-empty subset of nodes in which not all SKUs are represented, i.e., there is a $k \in \{0, \ldots, \bar{\boldsymbol{K}}\} \mid T \cap C_k \mid = 0$.
$\boldsymbol{C}_T$	Set of the SKUs with at least one element in T in which $1 \leq \lvert C_T \rvert < \lvert \bar{\boldsymbol{K}} \rvert$.
$\boldsymbol{C}_k$	Set of all nodes in the cluster k.
Parameters	
d_{ij}	Length of edge (i, j).
Decision variables	
x_{ij}	Binary variable that equals 1 if edge (i, j) is traversed from i to j; 0 otherwise.
y_i	Binary variable that equals 1 if vertex i is visited and 0 otherwise.
u_p	order of visiting cluster p in the tour for MTZ subtour elimination constraint
w_{pq}	Binary variable 1, if the order-picker visits a node of SKU q immediately after visiting a node from SKU p and 0 otherwise

$$\min \sum_{(i,j)\in\bar{\boldsymbol{E}}} d_{ij} x_{ij} \tag{1}$$

$$\sum_{i\in\boldsymbol{C}_k} y_i = 1 \qquad \forall k \in \bar{\boldsymbol{K}} \tag{2}$$

$$\sum_{i\in\bar{\boldsymbol{V}} \mid i\neq j} x_{ij} = y_j \qquad \forall j \in \bar{\boldsymbol{V}} \tag{3}$$

$$\sum_{j\in\bar{\boldsymbol{V}} \mid j\neq i} x_{ij} = y_i \qquad \forall i \in \bar{\boldsymbol{V}} \tag{4}$$

$$\sum_{i\in\boldsymbol{C}_T \mid i\neq j} \sum_{j\in\boldsymbol{C}_T \mid j\neq i} x_{ij} \leq |\boldsymbol{C}_T| - 1 \qquad \forall \boldsymbol{C}_T \tag{5}$$

The objective function (1) minimizes the total distance travelled by the pickers. Constraints (2) ensure that every SKU must be visited exactly once. The flow constraints are indicated by constraints (3) and (4). These constraints ensure that if an item is picked in a given tour, one incoming and one outgoing edge to this node from other nodes should exist. Constraints (5) are the subtour elimination constraints.

3.3.2 MTZ formulation of GTSP

The second ILP formulation considered in this study is the GTSP formulation proposed by [41] and improved by [13] using Miller–Tucker–Zemlin (MTZ) subtour elimination constraints. The mathematical formulation is the following:

$$\min \sum_{(i,j)\in\bar{\boldsymbol{E}}} d_{ij}x_{ij} \tag{6}$$

$$w_{pq} = \sum_{i\in\boldsymbol{C}_p}\sum_{j\in\boldsymbol{C}_q} x_{ij} \qquad \forall \in \bar{\boldsymbol{K}} \mid p \neq q \tag{7}$$

$$\sum_{p|p\neq q} w_{pq} = 1 \qquad \forall q \in \bar{\boldsymbol{K}} \tag{8}$$

$$\sum_{q|q\neq p} w_{pq} = 1 \qquad \forall p \in \bar{\boldsymbol{K}} \tag{9}$$

$$\sum_{j\in\bar{\boldsymbol{V}}\setminus\{i\}} x_{ji} - \sum_{j\in\bar{\boldsymbol{V}}\setminus\{i\}} x_{ij} = 0 \qquad \forall i \in \bar{\boldsymbol{V}} \tag{10}$$

$$u_p - u_q + |\bar{\boldsymbol{K}}|w_{pq} \leq |\bar{\boldsymbol{K}}| - 1 \qquad \forall p,q \in \bar{\boldsymbol{K}} \mid p \neq q \tag{11}$$

$$2 \leq u_p \leq |\bar{\boldsymbol{K}}| + 1 \qquad \forall p \in \bar{\boldsymbol{K}} \mid p \neq 1; u_1 = 1 \tag{12}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in \bar{\boldsymbol{V}} \tag{13}$$

The objective function (6) is minimizing total distance traveled by the order picker. Constraints(7) are the definition of the auxiliary variables, w_{pq}, in terms of the defined decision variables x_{ij}. Constraints (8) and (9) ensure that every SKU must be visited exactly once. This is done by ensuring that the degree of the incoming and outgoing arcs from each SKU is equal to 1, respectively. Constraints (10) state that entering flow to every node should be equal to exiting flow (flow balance constraints). Constraints (11) are the subtour elimination constraints, which represent the visiting order for all SKUs.

3.4 Reflection on existing formulations

In the DFJ formulation, the computational time goes up easily as the number of constraints grows exponentially in the number of nodes in the graph. More specifically, the number of subtour elimination constraints equals $(2^{\bar{K}} - \bar{K} - 1)$, where $\bar{K}$ is the number of SKUs. This implies that solving this model must be done with a separation routine on the subtour elimination constraints, which is time-consuming.

[41] show that the MTZ formulation contains O(n^2) binary variables and O(n^2) constraints. As such, the number of variables and constraints are polynomial in the size of the problem. This formulation, however, makes use of three types of variables: one based on the nodes (all SKU locations), one based on the clusters (subsets of SKU locations that contain the same SKU), and one based on the arcs.

3.5 A new mathematical formulation for the GTSP

In this section, we present a new mathematical formulation for the generalized travelling salesman problem that combines the beneficial characteristics of the above-mentioned DFJ and MTZ formulations. This new formulation is based only on decision variables at the SKU level. For the subtour elimination constraints, we rely on the MTZ approach.

Table 2: *Summary of all mathematical notation.*

Sets	
$\boldsymbol{V}$	The set of all the vertices in the graph (complete warehouse).
$\bar{\boldsymbol{V}} \subseteq \boldsymbol{V}$	The set of all storage locations of SKUs on the picking list, including the depot.
$\boldsymbol{C}_k$	The set of all vertices in cluster k.
$\boldsymbol{K}$	The set of all SKUs in the warehouse.
$\bar{\boldsymbol{K}} \subseteq \boldsymbol{K}$	The set of all SKUs on the picking list.
Parameters	
d_{ij}	Length of edge (i, j).
Decision variables	
x_{ij}	Binary variable that equals 1 if edge (i, j) is traversed from i to j; 0 otherwise.
y_i	Binary variable that equals 1 if vertex i is visited and 0 otherwise.
u_i	The position of vertex i in the tour.

$$\min \sum_{(i,j)\in\bar{\boldsymbol{E}}} d_{ij}x_{ij} \tag{14}$$

$$\sum_{i\in\boldsymbol{C}_k} y_i = 1 \qquad \forall k \in \bar{\boldsymbol{K}} \tag{15}$$

$$\sum_{i\in\bar{\boldsymbol{V}}|i\neq j} x_{ij} = y_j \qquad \forall j \in \bar{\boldsymbol{V}} \tag{16}$$

$$\sum_{j\in\bar{\boldsymbol{V}}|j\neq i} x_{ij} = y_i \qquad \forall i \in \bar{\boldsymbol{V}} \tag{17}$$

$$u_0 = 1 \tag{18}$$

$$u_i - u_j + |\bar{\boldsymbol{K}}|x_{ij} \leq |\bar{\boldsymbol{K}}| - 1 \qquad \forall i,j \in \bar{\boldsymbol{V}}\backslash\{0\} \mid i \neq j \tag{19}$$

$$2 \leq u_i \leq |\bar{\boldsymbol{K}}| + 1 \qquad \forall i \in \bar{\boldsymbol{V}}\backslash\{0\} \tag{20}$$

$$u_i \in \mathbb{N} \qquad \forall i \in \bar{\boldsymbol{V}} \tag{21}$$

$$x_{ij}, y_j \in \{0,1\} \qquad \forall i,j \in \bar{\boldsymbol{V}} \tag{22}$$

The objective function (14) minimizes the total distance travelled by the picker. Constraints (15) ensure that each SKU on the pick list is picked exactly once. Constraints (16) and (17) set the incoming and outgoing edges for each visited cluster. In constraints (18) the depot is set as the first vertex in the pick tour. The positions of all other vertices in the tour are set via constraints (19), after which the domain of these positions is bounded by the number of clusters that will be visited in the complete pick tour in constraints (20). Finally, the domains of the decision variables are managed by constraints (21) and (22).

4 Guided Local Search algorithm

To solve the order picking problem under a scattered storage policy, we present a Guided Local Search (GLS) algorithm. In this variant of local search, problem-specific features of the solution are considered directly in the objective function, thereby 'guiding' the search to solutions that possess most features of high quality solutions.

4.1 Overview of our GLS procedure

In section we provide a brief overview of the different components within our GLS algorithm. In the following sections, we will elaborate on each of these in more details. The pseudo-code of our GLS algorithm and a flow diagram are provided in Algorithm 1 and Figure 3, respectively. In figure 3, r is the set of shaking phase operators, for $r = 1, ..., r_{max}$,

and l is the set of local search moves, for $l = 1, ..., l_{max}$.

Algorithm 1 Pseudo-code for our Guided Local Search algorithm.

```
1: k ← 0;                                                          ▷ iteration counter
2: x ← initial solution made by a Construction Method;
3: { set all penalties to 0}
4: for i ← 1 until M do
5:     p_i ← 0;
6: end for
7: {define the augmented objective function }
8: update the augmented objective value and set h(x) ← g(x) + λ_i Σ_{i∈M} p_i I_i c_i
9: while Stopping Criterion do
10:    x_{k+1} ← VNS (x_k, h);
11:    {compute the utility of features }
12:    {penalize features with maximum utility}
13:    for All i such that util_i is maximum do
14:        p_i ← p_i + 1;
15:    end for
16:    k ← k + 1;
17: end while
18: x* ← best solution found with respect to objective function g;
19: return x*
```

First, an initial solution is generated by means of a constructive heuristic. As we only accept feasible solutions, this initial solution will be a tour that starts and ends at the depot and visits exactly one storage location for each required SKU.

Then, we evaluate the initial solution based on its *augmented objective function*. The augmented objective function is a combination of the actual objective function (in our case the minimization of the total distance travelled) with a series of penalty terms. These penalty terms focus on problem-specific features that are likely not appearing in the optimal solution (e.g., very long travel distance between two consecutive picks). By penalizing these undesirable solution features, we help the algorithm to converge towards the more promising regions of the solution space faster. Moreover, the penalties are updated dynamically to help the local search heuristic to escape local optima (e.g., by suddenly allowing solutions that score less high on the included criteria) or to focus on high quality regions (e.g., by punishing certain criteria more) [88].

In the actual local search phase, we improve the current solution by exploring a series of local search neighbourhoods sequentially. As such, we make use of the principles of *Variable Neighbourhood Search*. If no improvements can be found (i.e., the algorithm is stuck in a local optimum), a shaking phase – often referred to as a *perturbation* – helps to reach different regions of the solution space, after which the algorithm continues its search.

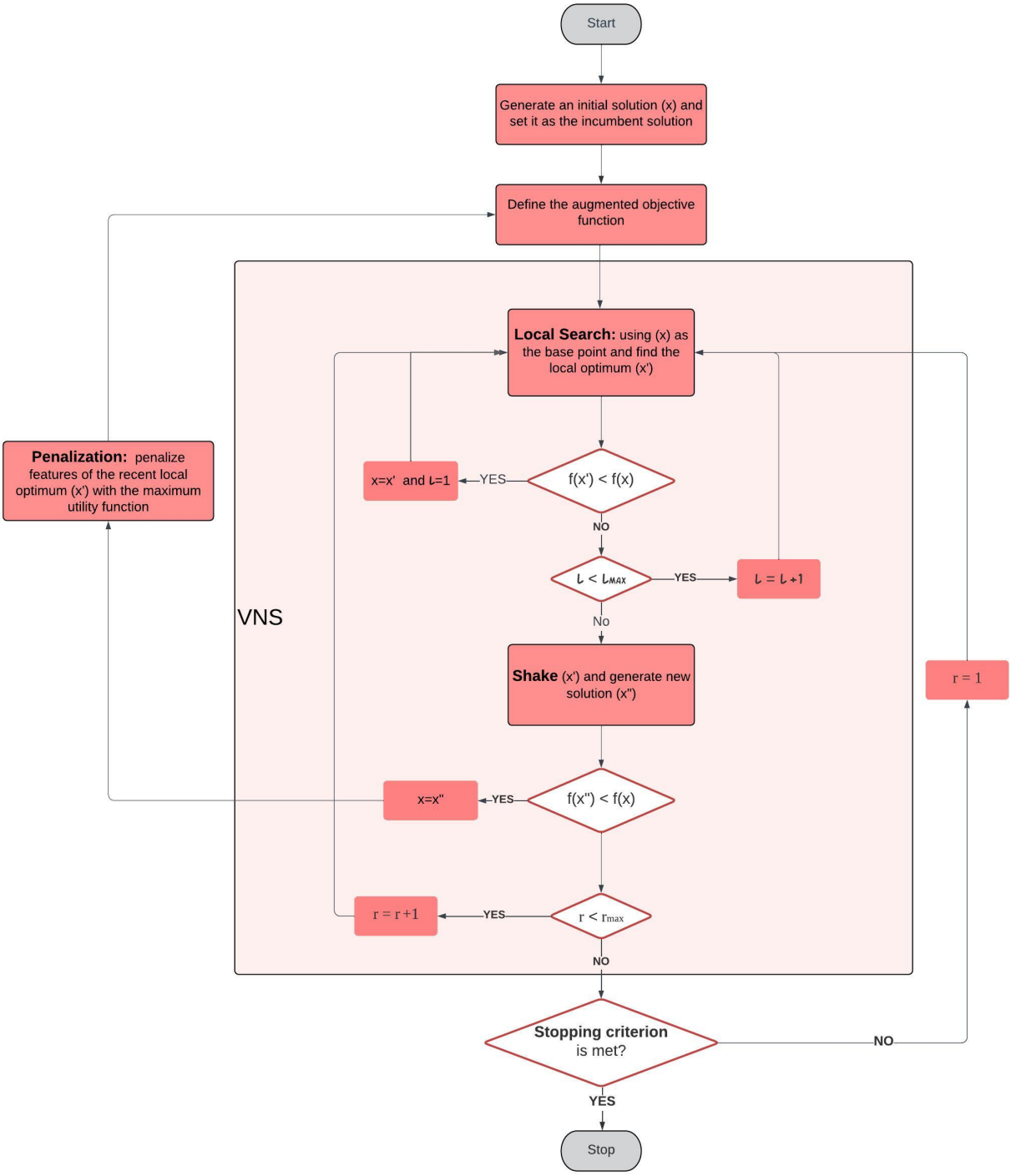


Figure 3: *Overview of the Guided Local Search algorithm.*

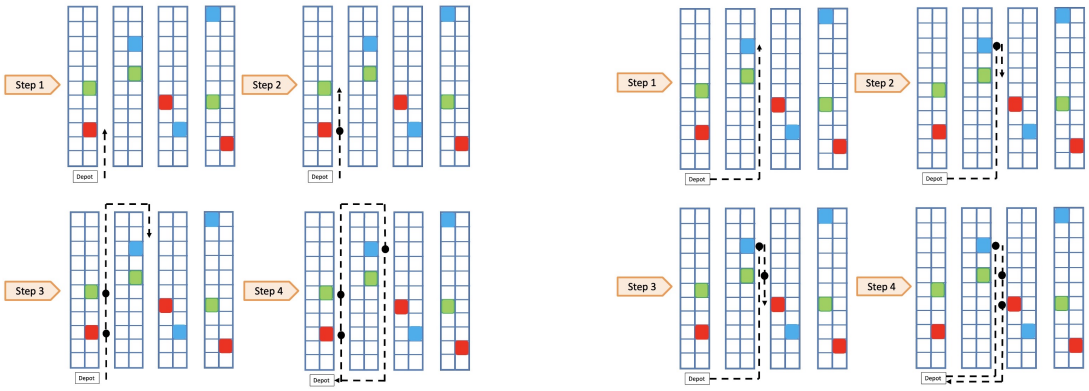
4.2 Step 1: Generate an initial solution

The aim of this step is to generate a feasible solution to the order picking problem under a scattered storage policy. To achieve this, we propose the following two approaches: a greedy nearest neighbour heuristic and a farthest insertion algorithm. The performance of both methods on the final solution is assessed (see Section 5) but no significant difference in final results (i.e., after the local search) could be found. Thus, since both of the constructive algorithms find the solution in less than one second, we run both algorithms, we select the best generated solution among them and use it as the initial solution for our GLS heuristic.

The first method, a *greedy nearest neighbour heuristic* is an adaptation of the nearest neighbour heuristic for the TSP. The picker starts at the depot after which the storage location that contains an SKU on the pick list and is closest to the current location is visited. Each time a storage location is added to the pick tour, all storage locations from the same cluster are removed from the graph. This procedure is repeated iteratively until all required SKUs have been picked. Then the picker moves from its current location back to the depot. As all distances in the graph are known, the procedure is very fast. However, due to the myopic decisions (i.e., we only consider a decision on the next location to visit), inefficient connections towards the end of the route are likely.

Second, we propose a *farthest insertion algorithm*, which works as follows: in the first step, we find for each SKU on the pick list the storage location that is closest to the depot. Then, among these locations we add the one that is farthest away from the depot to the pick tour. Knowing that we have to pick each SKU on the pick list, we know that for sure we need to travel up to this point into the warehouse (i.e., the nearest storage location of the SKU that is furthest away from the depot). For each next SKU, we perform a cheapest insertion (i.e., the storage location that increases the length of the pick tour the least is included into the partial tour. We continue until all required SKUs have been visited by the picker.

Figures 4a and 4b illustrate an example of the greedy and farthest insertion constructive solution respectively for the case where 3 clusters of products with 3 items (nodes) each exist.



(a) *Greedy nearest neighbour heuristic.* ***(b)*** *Farthest insertion algorithm.*

Figure 4: *Constructive algorithms to generate an initial solution.*

4.3 Step 2: Define the augmented objective function

Following the definitions from Alsheddy et al. [3], let $\boldsymbol{F}$ be set of solution features by which the quality of a solution can be assessed. For each feature $i \in \boldsymbol{F}$, let I_i be a binary variable indicating whether the feature is present in the current solution or not (also denoted a the *indicator function*). Let p_i record the number of times that feature i has been penalised (appeared in the local minima) and initially it is set to zero. Finally, we define λ_i as an overall weight factor for penalty dedicated to the feature i in the objective function. As such, λ_i balances the importance of the penalty factors over the main objective function (i.e., the minimization of the total distance travelled) and, thereby, controls the degree of guidance within the GLS. The value of λ_i is defined with parameter tuning (the value of λ depends on another parameter (α which will be explained later). Furthermore, let $c = \{c_1, c_2, \ldots, c_{|\boldsymbol{F}|}\}$ be a cost vector, in which c_i denotes the cost of feature i.

Denoting the main objective function (i.e., minimizing the total distance travelled by the order picker) by $g(s)$, the augmented cost function is given by

$$h(s) = g(s) + \sum_{i \in \boldsymbol{F}} \lambda_i p_i I_i(s) c_i \tag{23}$$

The objective within the GLS algorithm is to minimize the augmented cost function $h(s)$.

Each time a local minimum is found by the algorithm (and thus the augemented cost function cannot be optimised further), we will evaluate the current solution based on its solution features. This evaluation is done based on a so-called utility function $util(s^*, f_i)$

that scores the solution s^* on each feature i as follows:

$$util(s^*, i) = I_i(s^*)\frac{c_i}{(1+p_i)} \tag{24}$$

If a feature is not present in s^* (denoted by the indicator function $I_i(s^*) = 0$), then the utility of penalizing it equals to 0. For a feature that is present ($I_i(s^*) = 1$), its cost c_i will be computed (i.e., to what extend is the feature present in the solution). If a feature is penalized multiple iterations in a row, its penalty parameter p_i (which works as a counter) increases which will dampen the importance of the feature's utility which, at its turn, inserts more diversification on the search (i.e., other features will become more important in the augmented cost function).

The guide our local search to the more promising solutions, we define the following features:

1. **The maximum number of times that each picker-aisle is visited.**
   This feature tries to force the algorithm to pick all the items needed in the same aisle in one visit and avoids visiting an aisle more than twice (we know that in optimal solution, each aisle should be visited at most once in each direction). Let T^a be the number of times that aisle $a \in A$ is visited (A is the set of aisles) in our current solution s^*, then the utility of this aisle is given by:

   $$util(s^*, 1) = \max_a \left[I_1^a(s^*)\frac{T^a}{p_1^a + 1} \right] \tag{25}$$

   As part of our augmented objective function, we will punish the aisle with the largest utility. p_1^a is the penalty counter for aisle a. Moreover, λ_1 is the penalty dedicated to this feature.

2. **The maximum number of times that each cross-aisle is visited.**
   Similar to feature one, here we try to penalize the number of times that each cross-aisle is visited, so that we reduce the extra movements of an order picker in the warehouse. Let T^c be the number of times that cross-aisle $c \in C$ is visited (C is the set of cross-aisles) in our current solution s^*, then the utility of this cross-aisle is given by:

   $$util(s^*, 2) = \max_c \left[I_2^c(s^*)\frac{T^c}{p_2^c + 1} \right] \tag{26}$$

   As part of our augmented objective function, we will punish the cross-aisle with the

largest utility. p_2^c is the penalty counter for cross-aisle c. Moreover, λ_2 is the penalty dedicated to this feature.

3. **The maximum number of SKUs picked from each aisle.**

   The idea is that we would like to pick all items, while having to visit as few aisles as possible. Unlike the first two penalties, the cost associated with this feature is not as clear. To visit the least amount of aisles, we want to visit those that contain items from a large variety of clusters. Hence, the aim is to traverse those aisles which contain a large set of items from different clusters. Therefore, we wish to penalize the aisles which have low utilization. Considering that in the optimal case, we expect that in an aisle visit, an order-picker collects as many products as possible, in order to reduce the number of times other aisles are visited. To guide the search mechanism based on this feature, let N_a be the number of SKUs picked in aisle a in our current solution s^*, then the cost of this feature is defined as $\left(\frac{1}{N_a}\right)$. As a result, the utility of this feature is given by:

$$util(s^*, 3) = \max_a \left[I_3^a(s^*) \frac{1}{N_a(p_3^a + 1)} \right] \tag{27}$$

   As part of our augmented objective function, we will punish the aisle with the largest utility. p_3^a is the penalty counter for aisle a. Moreover, λ_3 is the penalty dedicated to this feature.

4. **The longest edges.** Our last important feature to be penalized is the longest edge travelled in the warehouse. If the length of an edge ij which is the shortest path connecting node i to node j in our solution is too high, it can be a sign of non-optimality since it may be the case that the order-picker has visited some aisles without picking any item on his way, or maybe the destination node of the edge could be visited by another closer node. Intuitively, we do not want long edges but cannot exclude them at the beginning of the search procedure, as they may be part of the optimal solution. Thus, we penalize them during the search if they appear in local optima instead of disregarding them.

   The utility function of this feature is as follows:

$$util(s^*, 4) = \max_{ij} \left[I_4^{ij}(s^*) \frac{d_{ij} x_{ij}}{p_4^{ij} + 1} \right] \tag{28}$$

where x_{ij} is a binary variable taking value 1 if order picker visits node j immediately after visiting node i and d_{ij} denotes the length of the edge. A natural choice for the cost associated with these solution features is the length of the analogous edge. Moreover, λ_4 is the penalty dedicated to this feature.

4.4 Step 3: Improvement by means of local search

4.4.1 Variable Neighbourhood Search

To improve the quality of the initial solution, we make use of variable neighbourhood search. Having defined multiple local search operators, the algorithm changes the operator as soon as no further improvement can be found in the current neighbourhood (i.e., the search is stuck in a local optimum). The following neighbourhoods are checked consecutively within our algorithm:

- 2-OPT. For a given set of two arcs in a single route that construct a crisscross, this move substitutes them with two new arcs by reversing the sequence of the nodes visited in between.
- 3-OPT. Remove three arcs and interchange their position in the itinerary.
- INTRA-SWAP. This move selects two random nodes (clusters) in our current solution (route) and swaps the nodes of these positions in the current route.
- INSERT. This move selects two random positions of nodes in our current solution (route) and inserts a randomly chosen element in front of another randomly chosen element.
- REVERSE. This move selects two random positions of nodes in our current solution (route) and reverses the direction between randomly chosen elements.

4.4.2 Shaking phase

Once a local optimum has been reached, the algorithm makes use of a shaking procedure to escape from it. The shaking procedure, inspired by the work of Sengupta, Mariescu-Istodor, and Fränti [75] and Tuononen [84], executes random moves from either of the two following operators.

- INTER-SWAP. This move selects one random node (cluster) in our current solution (route) and replaces another node of the same cluster which currently is not part of the solution with the existing node of that cluster. (see Figure 5)
- DOUBLE-BRIDGE. This move consists of a sequence of two disconnected 2-exchange moves. In the first exchange, the algorithm removes two random edges from the current tour and links their endpoints by adding new edges, resulting in two sub-tours. The second exchange removes two other edges, one from each sub-tour and reconnects the two parts by creating a bridge and making a feasible tour. (see Figure 6)

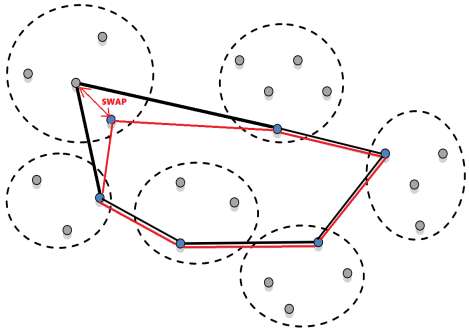


Figure 5: *Visualisation of the* INTER-SWAP *move.*

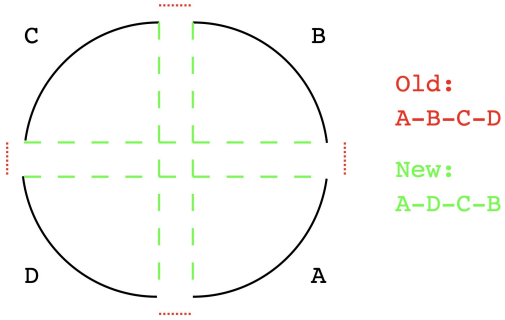


Figure 6: *Visualisation of the* DOUBLE-BRIDGE *move.*

4.5 Stopping criterion

The algorithm stops as soon as a maximum number of iterations of the VNS without improvement has been performed (which we fix to 1000) or if the running time exceeds 60 minutes.

5 Model implementation and numerical results

5.1 The instances

We perform computational experiments by generating a set of random instances, based on the instance generation procedure described in Theys et al. [81]. [1]

The benchmark used in our study is the one from [67] and [81] consisting of 9 different scenarios: three different number of aisles (5,15,60), three different number of cross aisles (3,6,11) and three different number of products in the order (15,60,240). As previously mentioned, there are no papers in the literature considering the GTSP in multi parallel aisle warehouses and the order picking problem.

5.2 Some details on implementation

All algorithms presented in this paper are implemented in Java, and the ILP formulations are solved with IBM CPLEX 22.1.0 with default parameters. The time limit for the exact algorithms has been set to 3600 seconds. Testing has been carried out on a Macbook Air with an Apple Silicon M1 chip and 16GB of RAM. It will be shown that high-quality solutions can be generated for real size instances of the order picker routing problem within a reasonable time. Please note that our instances are not exactly the same than those solved by [73] but were generated with the same parameters. It is worth mentioning that for each instance, we run the algorithms (all MILP, VNS and GLS) 10 times and report the mean values in the tables.

5.3 Parameter tuning

Another important part of our sensitivity analysis is the parameter tuning in order to find the best values for the penalty coefficients (λ) in our GLS algorithm. Recall that the λ parameters control the degree to which the respective penalties influence the search procedure. As it has been previously stated, these parameters present a trade-off between exploration and exploitation of the search space. Hence, the choice of the values associated with these parameters will impact the efficiency of the search procedure. Since the effectiveness of exploration and exploitation is highly dependent on the landscape of the search space, the choice of the λ parameters will, in general, be instance specific. Based on

[1] More information about the instances and instance generator can be found via `https://homepages.dcc.ufmg.br/~arbex/orderpicking.html`.

computational experiments for several problems, [89] observed that the basis of obtaining good values for these parameters can be found by dividing the objective value within a local optimum by the number of solution features present in this solution which makes sure that these parameters are instance independent. In another words, λ is computed dynamically after finding the first local optimum and before penalizing the features for first time. Having an α parameter which is calculated by sensitivity analysis, λ is calculated by

$$\lambda = \frac{\alpha g(s^*)}{|F_{s^*}|} \tag{29}$$

in which s^* and F_{s^*} denote the local optimum and the features present in the solution. The value of the α parameter should be between 0 and 1, using the information from [88] since we are using the same 2-opt and 3-opt operators in our VNS heusristic. To optimize performance on a built model, parameter tuning is necessary for any algorithm. To do so, we add each penalising feature to the objective function separately and then solve the model for several values of α and find the best point where our total cost is minimized. Figure 7 illustrates the optimal value for our α parameter.

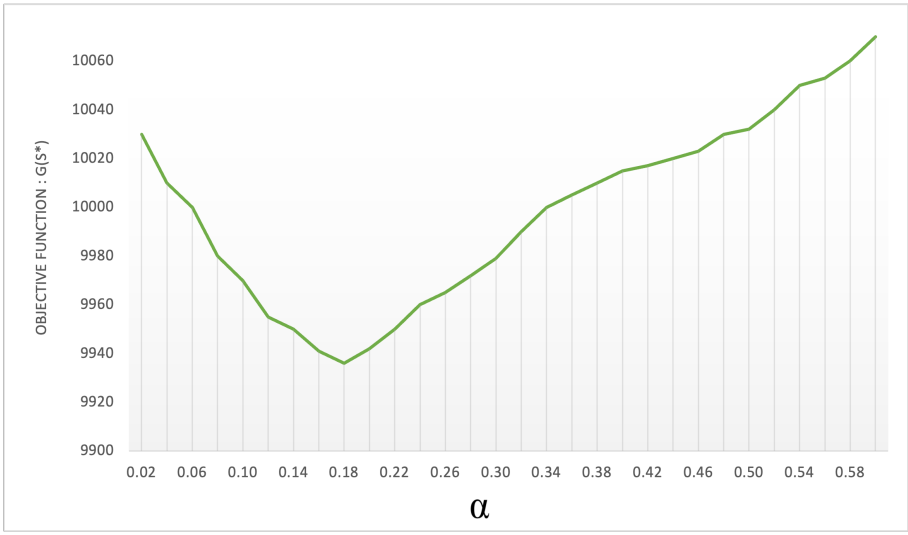


Figure 7: *Sensitivity analysis of GLS algorithm for optimal tuning of penalty confidentα*

5.4 Comparison between state-of-the-art formulations on traditional problem

We first assume that in our GTSP, each cluster consists of only one node, changing the problem to normal TSP, and then we solved our exact and heuristic algorithms to have a

better comparison and to examine the efficiency of our proposed algorithm. In figure 8, we solve the instances on the multi-block warehouse and each cluster containing only one item (TSP). As shown in this figure, for all the instances of different size, our exact model has the optimal solution in less than one minute. The calculation times are based on CPLEX solver time.

It is illustrated in this figure that our proposed MILP(MILP proposed) gives better solutions in comparison with the MILP in the literature using MTZ formulation (given in column 'MILP old') in a shorter running time. In this table 'O' means Optimal solution, 'F' is used to show Feasible solution and NS means no solution. In this table, MILP best cost refers to the the best objective value among the first two columns. Both of our constructive algorithms (greedy and Farthest-Insertion) give us the initial solution in less than one second. Furthermore, their solution quality is not comparable since non of them is outperforming the other one in the solution quality. For some instances, the greedy algorithm gives better solutions and for the other ones, the Farthest-Insertion. No pattern for their solution quality based on different instance sizes has been found. Thus we run both algorithm and since they are both very fast, we select the best generated solution among them and use it as the initial solution for our GLS heuristic. To have an analysis over the efficiency of the GLS and how much it improves the VNS without penalties, we separate these two parts and solve the instances with both of them (VNS and VNS+GLS). Since in this table, we consider only one item inside each cluster (TSP), the instance sizes are smaller than the other sets and our MILP model is giving optimal solution for most of the instances. Therefore, the improvements of the exact model by VNS is on average 6.5%. Also the average improvement of VNS by GLS is 2%. This number gets higher if the instance sizes (items in each cluster) increase. It is worth mentioning, that our GLS heuristic is capable of finding the optimal solutions for most of the instances along with our proposed MILP, but in a much shorter run time (less than 3.5 seconds for all instances). The other non-optimal instances, are having a huge negative gap with our exact solution, which shows the improvement in our heuristic solution. For bigger instances, the heuristic stops by the termination criteria regarding no improvements for 1000 iterations.

5.5 Performance for increasing nodes per cluster

In this section, we examine the performance of our exact and heuristic models on 4 different cluster sizes (2,5,10,20) and solve our instances based on these cluster sizes. In figures 9-12, the numerical results for the cluster sizes 2,5,10 and 20 are reported respectively. As

Test Problem			MILP Model (Proposed) (Using CPLEX in GMAS)		MILP Model (Old) (Using CPLEX in GMAS)		MILP (Best Cost)		Constructive Greedy Search		Constructive Max-Min Search		VNS (Using best constructive)		Guided Local Search (GLS)			
Aisles	Cross Aisles	Products	Run Time* (s)	O.F. (Cost)	Run Time (s)	O.F. (Cost)	O.F. (Cost)	Optimality	Run Time (s)	O.F. (Cost)	Run Time (s)	O.F. (Cost)	Run Time (s)	O.F. (Cost)	Run Time (s)	O.F. (Cost)	Gap w.r.t. CPLEX (%)	Improve w.r.t. VNS (%)
5	3	15	4.96	2090	7.07	2090	2090	O	<1	2091	<1	2091	0.55	2090	0.15	2090	0,00	0,00
5	3	60	6.28	2549	7.53	2549	2549	O	<1	2554	<1	2552	1.48	2549	0.48	2549	0,00	0,00
5	3	240	1792.63	5066	2030.86	5066	5066	O	<1	5296	<1	5178	3.52	5074	0.72	5066	0,00	-0,16
5	6	15	10.57	2687	12.53	2687	2687	O	<1	2745	<1	2726	0.89	2687	0.39	2687	0,00	0,00
5	6	60	36.16	3400	46.16	3400	3400	O	<1	3681	<1	3741	2.61	3410	1.01	3400	0,00	-0,29
5	6	240	3600*	6796	3600	7042	6796	F	<1	5912	<1	5892	3.91	5580	1.34	5513	-18,88	-1,20
5	11	15	21.39	2757	27.23	2757	2757	O	<1	2759	<1	2757	1.77	2759	0.64	2757	0,07	0,00
5	11	60	40.73	4075	43.49	4075	4075	O	<1	4171	<1	4156	3.32	4078	1.11	4075	0,00	-0,07
5	11	240	3600	5333	3600	5585	5333	F	<1	5500	<1	5436	4.58	4328	1.20	4067	-23,74	-6,03
15	3	15	6.82	2465	7.47	2465	2465	O	<1	2471	<1	2477	2.09	2465	0.87	2465	0,00	0,00
15	3	60	50.79	3944	81.59	3944	3944	O	<1	4010	<1	3990	2.58	3944	0.98	3944	0,00	0,00
15	3	240	3600	7856	3600	8132	7856	F	<1	7584	<1	7492	3.97	6890	1.07	6598	-16,01	-4,24
15	6	15	16.79	3497	20.17	3497	3497	O	<1	3523	<1	3550	3.11	3497	1.14	3497	0,00	0,00
15	6	60	67.43	6315	84.19	6315	6315	O	<1	6345	<1	6345	4.80	6315	1.22	6315	0,00	0,00
15	6	240	3600	17561	3600	18120	17561	F	<1	18800	<1	18650	5.94	13580	2.36	13465	-23,32	-0,85
15	11	15	18.71	5684	26.83	5684	5684	O	<1	5708	<1	5699	2.35	5684	1.03	5684	0,00	0,00
15	11	60	80.11	9642	91.98	9642	9642	O	<1	9679	<1	9700	4.93	9642	2.15	9642	0,00	0,00
15	11	240	3600	18365	3600	18956	19365	F	<1	17329	<1	17023	6.31	15471	2.67	13674	-25,54	-11,62
60	3	15	4.96	2293	6.45	2293	2293	O	<1	2365	<1	2320	3.59	2293	1.16	2293	0,00	0,00
60	3	60	112.80	3650	132.23	3650	3650	O	<1	3676	<1	3687	4.17	3661	1.71	3650	0,00	-0,30
60	3	240	3600	13211	3600	13750	13211	F	<1	12432	<1	12560	5.08	11045	2.32	10621	-19,60	-3,84
60	6	15	43.18	3645	51.55	3645	3645	O	<1	3689	<1	3674	4.55	3645	1.86	3645	0,00	0,00
60	6	60	156.25	4024	180.77	4024	4024	O	<1	4140	<1	4156	5.56	4024	2.09	4024	0,00	0,00
60	6	240	3600	19660	3600	19792	19660	F	<1	17328	<1	17506	6.26	15590	2.37	15065	-23,37	-3,37
60	11	15	123.39	4950	148.40	4950	4950	O	<1	5031	<1	5015	4.01	4955	1.91	4950	0,00	-0,10
60	11	60	163.79	6705	198.28	6705	6705	O	<1	6983	<1	6950	6.15	6745	2.95	6705	0,04	-0,55
60	11	240	3600	21260	3600	21361	21260	F	<1	18806	<1	18651	9.57	16517	3.49	15871	-25,35	-3,91

* Maximum run time has been set 3600 seconds for CPLEX solver in GAMS, and Heuristics in MATLAB

Figure 8: *Result for problem instances in case of each cluster containing only one product*

it is shown in these tables, with the increase in cluster sizes and accordingly the instance sizes, our proposed MILP is not able to give us a feasible solution within the time limit of one hour. However, our heuristic algorithm is able to solve the problem and give us a good solution in a very short time (for our biggest instance with 4800 nodes, the run time is 52.5 seconds which makes it remarkably fast). For the cluster size 1,2 and 5, our proposed MILP can provide optimal or feasible solutions, but for the larger instances, the model is not able to give any solutions within 60 minutes. The other notable point in these tables is the comparison between our proposed MILP and the existing MILP in the literature. On average, the run time of our proposed MILP is 20% less than the other MILP in the literature, and the quality of the solutions generated by our MILP is on average 15% better than the existing MILP.
Comparing the solutions of GLS with VNS, we can see that implementing GLS and penalizing the features of the solutions has an improvement of 15% on average; and this percentage is higher for bigger instances than for small instances with 2 products in each cluster, for which the average improvement is 4%.

Furthermore, the percentage of improvements of GLS implemented on a VNS algorithm for cluster sizes 2,10 and 20 is illustrated in figures 13-15. For the smaller cluster size (2), the average improvement achieved by GLS is 4% which is much lower than the percentage of improvements in larger instances (cluster size 20) for which this value is 15%. The reason for this is that in the smaller instances, our VNS algorithm is also able to find optimal or close to optimal solutions. Therefore, a GLS algorithm does not add to much to the VNS algorithm. However, in bigger instances, the solutions found by GLS have much better quality and the running times of the GLS is much lower than the VNS or MILP.

Test Problem			MILP Model (Proposed)		MILP Model (Old)		CPLEX (Best Cost)		Constructive Greedy Search		Constructive Max-min Search		VNS (Using best constructive)		Guided Local Search (GLS)			
Aisles	Cross Aisles	Products	Run Time (s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	O.F. (Cost)	optimality	Run Time(s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	Run Time (s)	O.F. (Cost)	Gap w.r.t. CPLEX (%)	Improve w.r.t. VNS (%)
5	3	15	103.96	1437	156.45	1437	1437	O	<1	1706	<1	1685	1.55	1437	0.61	1437	0,00	0,00
5	3	60	214.73	2195	300.95	2195	2195	O	<1	2500	<1	2459	4.48	2195	0.78	2195	0,00	0,00
5	3	240	3600	4991	3600	5080	4991	F	<1	5063	<1	5111	8.48	4524	1.29	4487	-10,10	-0,82
5	6	15	190.09	1789	263.15	1789	1789	O	<1	1823	<1	1860	3.43	1789	1.38	1789	0,00	0,00
5	6	60	267.98	4078	366.43	4078	4078	O	<1	4300	<1	4311	5.61	4140	1.59	4078	0,00	-1,50
5	6	240	3600	13840	3600	14280	13840	F	<1	14158	<1	14601	10.91	12115	2.71	11784	-14,86	-2,73
5	11	15	266.98	2671	311.17	2671	2671	O	<1	2773	<1	2791	9.77	2680	2.86	2671	0,00	-0,11
5	11	60	365.74	6807	460.97	6807	6807	O	<1	7215	<1	7165	12.32	6855	3.03	6807	0,00	-0,19
5	11	240	3600	17656	3600	18320	17656	F	<1	15154	<1	15630	18.58	13254	3.17	12704	-28,05	-4,15
15	3	15	137.95	2060	175.05	2060	2060	O	<1	2412	<1	2374	10.09	2060	1.28	2060	0,00	0,00
15	3	60	3600	4331	3600	4411	4331	F	<1	4650	<1	4790	16.58	4019	1.99	3985	-7,99	-0,85
15	3	240	3600	6403	3600	6640	6403	F	<1	6415	<1	6320	14.47	5961	2.63	5874	-8,26	-1,46
15	6	15	646.39	3841	810.47	3841	3841	O	<1	4260	<1	4178	12.11	3932	2.07	3841	0,00	-0,72
15	6	60	3600	8412	3600	8650	8412	F	<1	7890	<1	7546	11.80	6180	2.74	6054	-28,03	-2,04
15	6	240	3600	9991	3600	10219	9991	F	<1	9927	<1	9711	18.94	8361	3.98	8361	-16,31	0,00
15	11	15	1321.07	7016	1570.87	7016	7016	O	<1	8605	<1	8410	7.35	7218	1.21	7121	0,07	-0,89
15	11	60	3600	13247	3600	13600	13247	F	<1	10865	<1	11149	20.93	10691	3.46	10021	-24,35	-6,27
15	11	240	3600	15265	3600	16073	15265	F	<1	14170	<1	13905	26.31	11594	3.95	11461	-24,92	-1,15
60	3	15	1826.10	3751	2100.46	3751	3751	O	<1	4032	<1	4096	11.59	3794	2.13	3761	0,09	-1,13
60	3	60	3600	5641	3600	5960	5641	F	<1	5311	<1	5390	17.17	5050	2.82	5050	-10,48	0,00
60	3	240	3600	9410	3600	9740	9410	F	<1	9140	<1	8834	23.08	8165	3.01	8031	-14,65	-1,64
60	6	15	2873.40	4584	3018.71	4584	4584	O	<1	5760	<1	5814	13.55	4881	1.66	4591	0,08	-2,25
60	6	60	3600	9174	3600	9460	9174	F	<1	7978	<1	7674	19.56	6984	2.96	6984	-23,87	0,00
60	6	240	3600	14970	3600	15640	14970	F	<1	13791	<1	14268	28.26	12417	4.72	11941	-20,23	-3,83
60	11	15	3600	7941	3600	8234	7941	F	<1	8315	<1	7842	22.01	7102	3.14	7102	-10,57	0,00
60	11	60	3600	17410	3600	18764	17410	F	<1	15607	<1	15251	31.15	14107	3.96	13943	-19,91	-1,16
60	11	240	3600	36450	3600	38410	36450	F	<1	33687	<1	32964	39.57	29723	4.91	28641	-21,42	-3,64

* Maximum run time has been set 3600 seconds for CPLEX solver in GAMS, and Heuristics in MATLAB

Figure 9: *Result for problem instances in case of each cluster containing 2 products*

Test Problem			MILP Model (Proposed)		MILP Model (Old)		CPLEX (Best Cost)		Constructive Greedy Search		Constructive Max-min Search		VNS (Using best constructive)		Guided Local Search (GLS)			
Aisles	Cross Aisles	Products	Run Time (s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	O.F. (Cost)	optimality	Run Time(s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	Run Time (s)	O.F. (Cost)	Gap w.r.t. CPLEX (%)	Improve w.r.t. VNS (%)
5	3	15	589.54	1346	654.44	1346	1346	O	<1	1650	<1	1696	6.05	1346	0.74	1346	0,00	0,00
5	3	60	2497.54	2565	2658.54	2565	2565	O	<1	3320	<1	3396	7.08	3132	2.48	2565	0,00	-18,10
5	3	240	3600	5331	3600	5631	5331	F	<1	5931	<1	5976	25.08	5714	3.48	4832	-9,36	-15,44
5	6	15	1125.57	2065	1345.52	2065	2065	O	<1	2179	<1	2236	9.33	2065	1.43	2065	0,00	0,00
5	6	60	3241.65	2723	3380.54	2723	2723	O	<1	3871	<1	3954	13.67	3761	1.61	2723	0,00	-27,60
5	6	240	3600	NS	3600	NS	-	NS	<1	57946	<1	58388	29.81	54891	3.91	50328		-8,31
5	11	15	1948.54	3060	2248.54	3060	3060	O	<1	3410	<1	3320	13.98	3083	2.77	3083	0,75	0,00
5	11	60	3600	6551	3600	6743	6551	F	<1	6671	<1	6590	17.61	6504	3.32	6252	-4,56	-3,87
5	11	240	3600	NS	3600	NS	-	NS	<1	15743	<1	17679	34.64	15047	4.58	12134		-19,36
15	3	15	1195.43	1267	1305.54	1267	1267	O	<1	1532	<1	1696	10.63	1498	2.09	1287	1,58	-14,09
15	3	60	3600	3123	3600	3405	3123	F	<1	4106	<1	4210	13.33	3875	2.58	3087	-1,15	-20,34
15	3	240	3600	6733	3600	6974	6733	F	<1	12425	<1	13158	36.38	12425	3.47	6543	-2,82	-47,34
15	6	15	884.54	4125	1001.54	4125	4125	O	<1	4889	<1	4968	10.92	4889	2.11	4125	0,00	-15,63
15	6	60	3600	5691	3600	5793	5691	F	<1	5671	<1	5741	15.55	5664	1.80	5603	-1,55	-1,08
15	6	240	3600	NS	3600	NS	-	NS	<1	28919	<1	29828	40.93	25861	2.94	22851		-11,64
15	11	15	764.54	6239	805.54	6239	6239	O	<1	8015	<1	7371	29.56	7431	1.35	6365	2,02	-14,35
15	11	60	2554.54	11236	2740.54	11236	11236	O	<1	14658	<1	15432	32.21	12651	3.93	11242	0,05	-11,14
15	11	240	3600	14651	3600	15651	15651	F	<1	25715	<1	24375	43.39	20651	4.31	13542	-7,57	-34,42
60	3	15	1435.54	1895	1676.12	1895	1895	O	<1	2563	<1	2658	12.15	2321	2.59	1976	4,27	-14,86
60	3	60	3600	4563	3600	4695	4563	F	<1	4630	<1	4480	23.09	4400	3.17	3185	-30,20	-27,61
60	3	240	3600	NS	3600	NS	-	NS	<1	13476	<1	12876	45.63	11851	5.08	9954		-16,01
60	6	15	664.43	7827	984.21	7827	7827	O	<1	9958	<1	10719	15.70	9958	2.55	8051	2,86	-19,15
60	6	60	3120.54	10631	3600	10658	10631	O	<1	12868	<1	13314	21.92	12868	4.56	10631	0,00	-17,38
60	6	240	3600	NS	3600	NS	-	NS	<1	31891	<1	33149	51.12	30058	5.26	27412		-8,80
60	11	15	1841.43	16253	2254.54	16253	16253	O	<1	22281	<1	22734	25.00	19850	4.01	16464	1,30	-17,06
60	11	60	3600	NS	3600	NS	-	NS	<1	23461	<1	24187	35.98	22451	6.15	18540		-17,42
60	11	240	3600	NS	3600	NS	-	NS	<1	69460	<1	70481	59.53	56986	8.57	43541		-23,59

Figure 10: *Result for problem instances in case of each cluster containing 5 products*

Test Problem			MILP Model (Proposed)		MILP Model (Old)		CPLEX (Best Cost)		Constructive Greedy Search		Constructive Max-min Search		VNS (Using best constructive)		Guided Local Search (GLS)			
Aisles	Cross Aisles	Products	Run Time (s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	O.F. (Cost)	optimality	Run Time(s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	Run Time (s)	O.F. (Cost)	Gap w.r.t. CPLEX (%)	Improve w.r.t. VNS (%)
5	3	15	1874.54	2310	2349.54	2310	2310	O	<1	3174	<1	2894	7.38	2509	1.22	2310	0,00	-7,93
5	3	60	3600	7517	3600	7936	7517	F	<1	7105	<1	7320	13.08	6578	1.98	6231	-17,11	-5,28
5	3	240	3600	NS	3600	NS	-	NS	<1	27565	<1	26173	24.49	23914	2.37	20431		-14,56
5	6	15	2334.65	4890	2895.31	4890	4890	O	<1	5730	<1	5978	13.35	5276	2.56	5004	2,33	-5,16
5	6	60	3600	10357	3600	10451	10357	F	<1	9972	<1	9650	18.52	8991	3.91	8934	-13,74	-0,63
5	6	240	3600	NS	3600	NS	-	NS	<1	35125	<1	36960	29.08	33671	6.32	30754		-8,66
5	11	15	3421.43	3841	3600	4007	3841	O	<1	5107	<1	4973	14.97	4621	3.72	3841	0,00	-16,88
5	11	60	3600	12749	3600	13651	12749	F	<1	11117	<1	10500	27.52	9244	5.09	8964	-29,69	-3,03
5	11	240	3600	NS	3600	NS	-	NS	<1	44734	<1	44926	39.11	40361	8.20	34651		-14,15
15	3	15	3490.54	3931	3567.42	3931	3931	O	<1	5502	<1	5438	11.77	4911	2.55	4259	8,34	-13,28
15	3	60	3600	10967	3600	11683	10967	F	<1	10863	<1	11950	16.01	9931	3.09	9143	-16,63	-7,93
15	3	240	3600	NS	3600	NS	-	NS	<1	36985	<1	35169	31.36	33596	4.11	27541		-18,02
15	6	15	3531.43	10583	3600	11127	10583	O	<1	13349	<1	15041	15.11	12460	3.65	11184	5,68	-10,24
15	6	60	3600	23194	3600	25076	23194	F	<1	20176	<1	19136	23.83	17490	6.99	15055	-35,09	-13,92
15	6	240	3600	NS	3600	NS	-	NS	<1	62324	<1	60645	36.14	56596	10.42	50654		-10,50
15	11	15	3600	17604	3600	18750	17604	F	<1	17312	<1	16681	20.78	13490	7.41	13490	-23,37	0,00
15	11	60	3600	39640	3600	41341	39640	F	<1	36431	<1	37173	32.84	33145	12.33	31884	-19,57	-3,80
15	11	240	3600	NS	3600	NS	-	NS	<1	77020	<1	74686	45.49	65199	18.17	60438		-7,30
60	3	15	3600	3701	3600	3970	3701	F	<1	3353	<1	4042	21.51	2318	4.30	2134	-42,34	-7,94
60	3	60	3600	NS	3600	NS	-	NS	<1	13480	<1	13100	36.87	11891	12.78	11650		-2,03
60	3	240	3600	NS	3600	NS	-	NS	<1	36914	<1	36187	45.35	33306	18.22	27651		-16,98
60	6	15	3600	9706	3600	10368	9706	F	<1	9271	<1	8045	24.28	7360	5.96	7164	-26,19	-2,66
60	6	60	3600	NS	3600	NS	-	NS	<1	22741	<1	21720	43.40	17831	13.58	16541		-7,23
60	6	240	3600	NS	3600	NS	-	NS	<1	48117	<1	49930	49.36	44247	21.37	40541		-8,38
60	11	15	3600	16580	3600	17803	16580	-	<1	17260	<1	15894	28.84	12760	10.31	12760	-23,04	0,00
60	11	60	3600	NS	3600	NS	-	NS	<1	35015	<1	33291	46.78	29990	16.43	27143		-9,49
60	11	240	3600	NS	3600	NS	-	NS	<1	88382	<1	85151	79.38	78365	29.08	71581		-8,66

Figure 11: *Result for problem instances in case of each cluster containing 10 products*

Test Problem			MILP Model (Proposed)		MILP Model (Old)		CPLEX (Best Cost)		Constructive Greedy Search		Constructive Max-min Search		VNS (Using best constructive)		Guided Local Search (GLS)			
Aisles	Cross Aisles	Products	Run Time (s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	O.F. (Cost)	optimality	Run Time(s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	Run Time(s)	O.F. (Cost)	Run Time (s)	O.F. (Cost)	Gap w.r.t. CPLEX (%)	Improve w.r.t. VNS (%)
5	3	15	3600*	676	3600	783	676	F	<1	1013	<1	954	10.79	741	3.54	521	-22,93	-29,69
5	3	60	3600	4300	3600	4631	4300	F	<1	4023	<1	3917	16.19	3742	6.41	3370	-21,63	-9,94
5	3	240	3600	NS	3600	NS	-	NS	<1	13951	<1	13171	39.85	10637	10.34	7843		-26,27
5	6	15	3600	3634	3600	3810	3634	F	<1	3170	<1	3368	15.42	2657	5.20	2657	-26,88	0,00
5	6	60	3600	NS	3600	NS	-	NS	<1	24510	<1	26105	27.54	22695	10.43	18541		-18,30
5	6	240	3600	NS	3600	NS	-	NS	<1	69212	<1	72077	43.16	63971	15.54	59450		-7,07
5	11	15	3600	9684	3600	10848	9684	F	<1	8049	<1	7916	22.80	6049	7.54	5344	-44,82	-11,65
5	11	60	3600	NS	3600	NS	-	NS	<1	36971	<1	35200	35.61	32793	12.22	32793		0,00
5	11	240	3600	NS	3600	NS	-	NS	<1	66596	<1	65103	40.20	58250	24.65	52583		-9,73
15	3	15	3600	2341	3600	2694	2341	F	<1	2570	<1	2306	13.79	2022	4.25	2022	-13,63	0,00
15	3	60	3600	NS	3600	NS	-	NS	<1	14852	<1	16500	24.17	12851	7.54	10654		-17,10
15	3	240	3600	NS	3600	NS	-	NS	<1	59870	<1	57693	41.38	52176	18.32	47853		-8,29
15	6	15	3600	12960	3600	13912	12960	F	<1	12574	<1	12695	19.84	12167	9.31	10654	-17,79	-12,44
15	6	60	3600	NS	3600	NS	-	NS	<1	48054	<1	47220	36.58	43214	14.54	39544		-8,49
15	6	240	3600	NS	3600	NS	-	NS	<1	92914	<1	94045	18.37	88503	20.43	80541		-9,00
15	11	15	3600	17839	3600	18962	17839	F	<1	16420	<1	15970	26.41	14271	14.54	11563	-35,18	-18,98
15	11	60	3600	NS	3600	NS	-	NS	<1	65333	<1	67608	43.37	60419	17.59	60419		0,00
15	11	240	3600	NS	3600	NS	-	NS	<1	111978	<1	107248	79.96	100810	35.65	97546		-3,24
60	3	15	3600	3301	3600	3619	3301	F	<1	3029	<1	2980	16.68	2117	8.52	1947	-41,02	-8,03
60	3	60	3600	NS	3600	NS	-	NS	<1	14381	<1	15973	35.59	11170	19.65	10575		-5,33
60	3	240	3600	NS	3600	NS	-	NS	<1	126454	<1	120871	97.50	112691	35.43	107541		-4,57
60	6	15	3600	19730	3600	21035	19730	F	<1	19010	<1	19671	25.50	17493	21.54	16345	-17,16	-6,56
60	6	60	3600	NS	3600	NS	-	NS	<1	30087	<1	30643	48.12	26642	36.06	26642		0,00
60	6	240	3600	NS	3600	NS	-	NS	<1	144763	<1	142807	116.26	134196	44.65	121549		-9,42
60	11	15	3600	NS	3600	NS	-	NS	<1	36840	<1	39412	38.43	32540	23.48	32540		0,00
60	11	60	3600	NS	3600	NS	-	NS	<1	111539	<1	108941	72.78	99035	43.94	79553		-19,67
60	11	240	3600	NS	3600	NS	-	NS	<1	150767	<1	153491	130.92	142391	52.54	117584		-17,42

Figure 12: *Result for problem instances in case of each cluster containing 20 products*

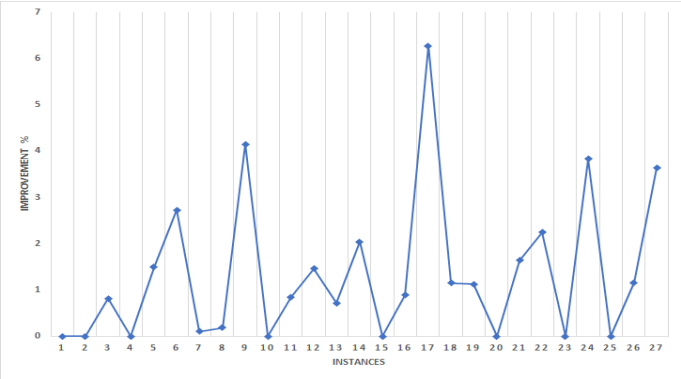


Figure 13: *%improvement of VNS by applying GLS in case of cluster size=2*

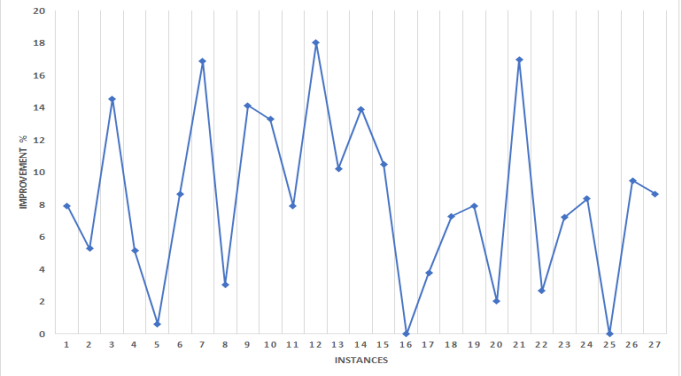


Figure 14: *%improvement of VNS by applying GLS in case of cluster size=10*

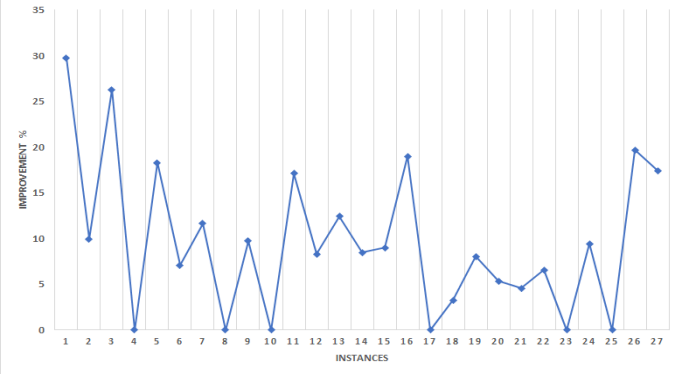


Figure 15: *%improvement of VNS by applying GLS in case of cluster size=20*

5.5.1 Warehouse Layout Comparison

In this section, we examine the effects of different warehouse sizes on the proposed algorithm. The computational run times for all problem instances are plotted in figure 16. The figure is separated into sub-figures for the instances related to the different cluster sizes. Furthermore, the nine different warehouse configurations are sorted along the x-axis according to the increasing number of total aisles contained in the warehouse. For example, the label k5h3 refers to the configuration with five aisles and three cross aisles, which is the smallest configuration considered in this paper. In this graph, n is considered as the number of products in the instance.

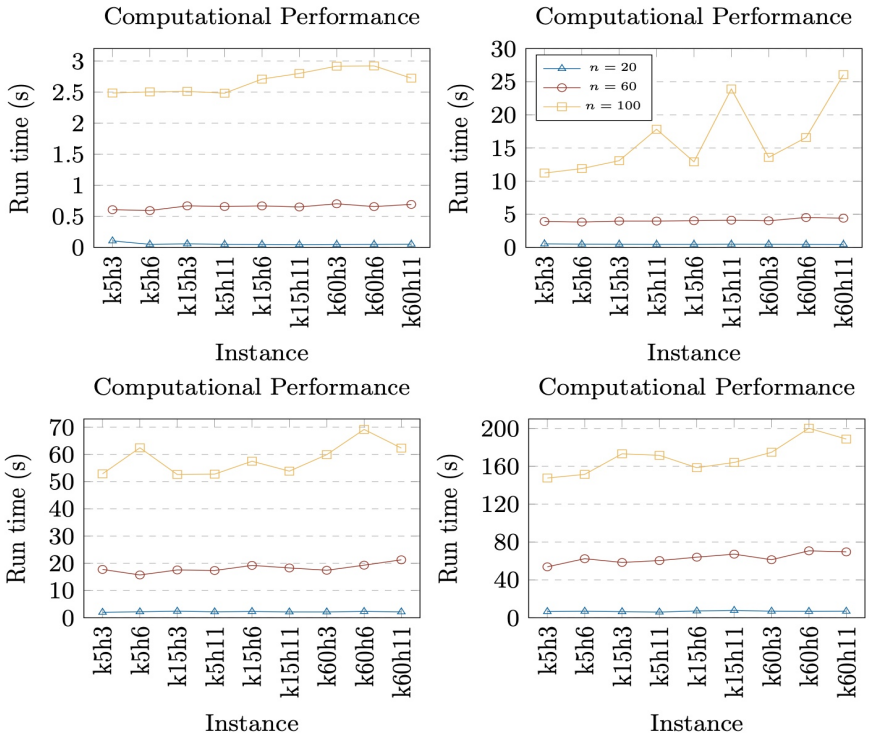


Figure 16: *Sensitivity analysis of Warehouse Layout*

In this figure, it is shown that the computational performance across all configurations with twenty items is very stable. A slightly increasing trend, especially for twenty clusters, can be identified when moving to sixty items. This implies that the computational performance worsens as the size of the warehouse grows. This trend becomes more evident when considering the instances containing one hundred items. However, it can also be seen that the warehouse configuration is not the driving factor affecting the run

times, but rather the number of items and clusters. This also conforms to the design of the algorithm since the number of iterations performed is solely based on these two factors.

6 Conclusion

in this paper, we proposed a novel exact model and a guided local search heuristic for the Generalized Travelling Salesman Problem (GTSP) with geographical overlap between clusters for order picking in a warehouse with scattered storage policy. This MILP model and proposed GLS algorithm can be applied for any warehouse layout (general graph), making it more practical. To the best of our knowledge, no exact models exist in the literature for warehouses with more than two blocks for this problem. Our exact model can be implemented in warehouses with many more blocks. This study is motivated by the possible warehouse efficiency gains and the fact that scattered storage policies where clusters overlap has received little attention.

The proposed algorithm uses problem-specific information to guide local search operators to promising search spaces. The algorithm performs very well based on its computational results. For cases with small clusters, the algorithm performs marginally better (although the computation time is still much lower than the exact model). In the majority of instances, when a greater number of clusters are considered, the algorithm discovers the optimal solution, as guaranteed by exact methods. The algorithm often outperforms the best known in the remaining cases. Although larger warehouses affected the algorithm's computational performance. Hence, the proposed GLS performs with many warehouse layouts and generates high solutions in seconds or minutes. Our efficient GLS heuristic solves large scale instances with up to 4800 nodes (with computation time less than one minute). Medium-to-large situations have superior solutions than our MILP. Our algorithm can solve all instances and provide a very good solution in a very brief amount of time.

Implementation of the two exact methods used to acquire the results for the instances used to evaluate the quality of the proposed heuristic algorithm is a significant limitation of this paper. The restriction is a direct result of the implementation of subtour elimination constraints, which ensure that solutions do not contain illegal subtours. The number of subtour elimination constraints increases exponentially as the cluster size increases. Thus, the computational performance of the commercial solver depends on how these constraints are

implemented. Resultantly, many instances with 15–20 clusters were not optimally solved with the exact model and retained a large GAP after 3600 seconds. This not only limited the ability to analyze the solution quality and computational performance of heuristic and exact methods, but also the size of the instances that could be considered. Due to the inability to assess solution quality, performance analysis across warehouse layouts was also affected.

Future research could examine order list due dates, dynamic customer orders, zoning, and multiple pickers joint problems, integration of batching/order picking decisions and inclusion of uncertain expected orders . Future research could also examine graph reduction (pre-processing) in scattered storage policies where clusters overlap.

References

[1] Riccardo Accorsi, Riccardo Manzini, and Marco Bortolini. "A hierarchical procedure for storage allocation and assignment within an order-picking system. A case study". In: *International Journal of Logistics Research and Applications* 15.6 (2012), pp. 351–364.

[2] Amir Reza Ahmadi Keshavarz et al. "A survey of the literature on order-picking systems by combining planning problems". In: *Applied Sciences* 11.22 (2021), p. 10641.

[3] Abdullah Alsheddy et al. *Guided Local Search.* 2018.

[4] Ehsan Ardjmand, Omid Sanei Bajgiran, and Eyad Youssef. "Using list-based simulated annealing and genetic algorithm for order batching and picker routing in put wall based picking systems". In: *Applied Soft Computing* 75 (2019), pp. 106–119.

[5] Ehsan Ardjmand et al. "Minimizing order picking makespan with multiple pickers in a wave picking warehouse". In: *International Journal of Production Economics* 206 (2018), pp. 169–183.

[6] Pouya Baniasadi et al. "A transformation technique for the clustered generalized traveling salesman problem with applications to logistics". In: *European Journal of Operational Research* 285.2 (2020), pp. 444–457.

[7] Ramin Bazrafshan, Sarfaraz Zolfani, and S.M.J. Mirzapour Al-e-hashem. "Comparison of the Sub-Tour Elimination Methods for the Asymmetric Traveling Salesman Problem Applying the SECA Method". In: *Axioms* 10 (Feb. 2021), p. 19. DOI: 10.3390/axioms10010019.

[8] Tolga Bektaş, Güneş Erdoğan, and Stefan Røpke. "Formulations and branch-and-cut algorithms for the generalized vehicle routing problem". In: *Transportation Science* 45.3 (2011), pp. 299–316.

[9] David Ben-Arieh et al. "Transformations of generalized ATSP into ATSP". In: *Operations Research Letters* 31.5 (2003), pp. 357–365.

[10] Tamás Bódis and János Botzheim. "Bacterial memetic algorithms for order picking routing problem with loading constraints". In: *Expert Systems with Applications* 105 (2018), pp. 196–220.

[11] Boris Bontoux, Christian Artigues, and Dominique Feillet. "A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem". In: *Computers & Operations Research* 37.11 (2010), pp. 1844–1852.

[12] Hadrien Cambazard and Nicolas Catusse. "Fixed-parameter algorithms for rectilinear Steiner tree and rectilinear traveling salesman problem in the plane". In: *European Journal of Operational Research* 270.2 (2018), pp. 419–429.

[13] Giovanni Campuzano, Carlos Obreque, and Maichel M Aguayo. "Accelerating the Miller–Tucker–Zemlin model for the asymmetric traveling salesman problem". In: *Expert Systems with Applications* 148 (2020), p. 113229.

[14] Jose Alejandro Cano, Alexander Alberto Correa-Espinal, and Rodrigo Andrés Gómez-Montoya. "An evaluation of picking routing policies to improve warehouse efficiency". In: *International Journal of Industrial Engineering and Management* 8.4 (2017), p. 229.

[15] Angelo Castelda. *Understanding The Impacts of eCommerce On Warehouse Operations*. `https://www.floship.com/blog/_ecommerce-warehouse-operations/`. Accessed: 2020-05-27.

[16] Melih Celik and HALDUN Süral. "Order picking in a parallel-aisle warehouse with turn penalties". In: *International Journal of Production Research* 54.14 (2016), pp. 4340–4355.

[17] Thomas Chabot et al. "Order picking problems under weight, fragility and category constraints". In: *International Journal of Production Research* 55.21 (2017), pp. 6361–6379.

[18] Fangyu Chen, Gangyan Xu, and Yongchang Wei. "Heuristic routing methods in multiple-block warehouses with ultra-narrow aisles and access restriction". In: *International Journal of Production Research* 57.1 (2019), pp. 228–249.

[19] Fangyu Chen et al. "An ACO-based online routing method for multiple order pickers with congestion consideration in warehouse". In: *Journal of Intelligent Manufacturing* 27 (2016), pp. 389–408.

[20] Tzu-Li Chen et al. "An efficient hybrid algorithm for integrated order batching, sequencing and routing problem". In: *International Journal of Production Economics* 159 (2015), pp. 158–167.

[21] Pablo Cortés et al. “A tabu search approach to solving the picking routing problem for large-and medium-size distribution centres considering the availability of inventory and K heterogeneous material handling equipment”. In: *Applied Soft Computing* 53 (2017), pp. 61–73.

[22] René De Koster, Tho Le-Duc, and Kees Jan Roodbergen. “Design and control of warehouse order picking: A literature review”. In: *European journal of operational research* 182.2 (2007), pp. 481–501.

[23] Roberta De Santis et al. “An adapted ant colony optimization algorithm for the minimization of the travel distance of pickers in manual warehouses”. In: *European Journal of Operational Research* 267.1 (2018), pp. 120–137.

[24] M Devaraj. *Impact of eCommerce Growth on the Logistics Sector*. `https://www.linkedin.com/pulse/impact-ecommerce-growth-logistics-sector-devaraj-m/?trk=pulse-article_more-articles_related-content-card`. Accessed: 2022-06-23.

[25] Mehdi El Krari et al. “A pre-processing reduction method for the generalized travelling salesman problem”. In: *Operational Research* 21.4 (2021), pp. 2543–2591.

[26] Matteo Fischetti, Juan José Salazar González, and Paolo Toth. “A branch-and-cut algorithm for the symmetric generalized traveling salesman problem”. In: *Operations Research* 45.3 (1997), pp. 378–394.

[27] Christoph H Glock and Eric H Grosse. “Storage policies and order picking strategies in U-shaped order-picking systems with a movable base”. In: *International Journal of Production Research* 50.16 (2012), pp. 4344–4357.

[28] David B Grant, Chee Yew Wong, and Alexander Trautrims. *Sustainable logistics and supply chain management: principles and practices for sustainable operations and management*. Kogan Page Publishers, 2017.

[29] Gregory Gutin and Daniel Karapetyan. “A memetic algorithm for the generalized traveling salesman problem”. In: *Natural Computing* 9.1 (2010), pp. 47–60.

[30] Gregory Gutin and Daniel Karapetyan. “Generalized traveling salesman problem reduction algorithms”. In: *Algorithmic Operations Research* 4.2 (2009), pp. 144–154.

[31] Randolph W Hall. “Distance approximations for routing manual pickers in a warehouse”. In: *IIE transactions* 25.4 (1993), pp. 76–87.

[32] Keld Helsgaun. “General k-opt submoves for the Lin–Kernighan TSP heuristic”. In: *Mathematical Programming Computation* 1.2 (2009), pp. 119–163.

[33] Keld Helsgaun. “Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun algorithm”. In: *Mathematical Programming Computation* 7.3 (2015), pp. 269–287.

[34] AL Henry-Labordere. “The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem”. In: *Revue Francaise D Informatique DeRecherche Operationnelle* 3.2 (1969), pp. 43–49.

[35] SS Ho and S Sarma. “The fragmented warehouse: Location assignment for unit-load picking”. In: *2008 IEEE International Conference on Industrial Engineering and Engineering Management*. IEEE. 2008, pp. 1159–1163.

[36] Jeffrey Hoeft and Udatta S Palekar. “Heuristics for the plate-cutting traveling salesman problem”. In: *IIE transactions* 29.9 (1997), pp. 719–731.

[37] Ling-feng Hsieh and Lihui Tsai. “The optimum design of a warehouse system on order picking efficiency”. In: *The International Journal of Advanced Manufacturing Technology* 28 (2006), pp. 626–637.

[38] Bin Hu and Günther R Raidl. “Effective neighborhood structures for the generalized traveling salesman problem”. In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2008, pp. 36–47.

[39] Stefan Irnich, Paolo Toth, and Daniele Vigo. “Chapter 1: The family of vehicle routing problems”. In: *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. SIAM, 2014, pp. 1–33.

[40] Vasiliki Kapou et al. “An Innovative Layout Design and Storage Assignment Method for Manual Order Picking with Respect to Ergonomic Criteria”. In: *Logistics* 6.4 (2022), p. 83.

[41] Imdat Kara, Huseyin Guden, and Ozge N Koc. “New formulations for the generalized traveling salesman problem”. In: *Proceedings of the 6th international conference on applied mathematics, simulation, modelling, ASM*. Vol. 12. 2012, pp. 60–65.

[42] Daniel Karapetyan and Gregory Gutin. “Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem”. In: *European Journal of Operational Research* 208.3 (2011), pp. 221–232.

[43] Natalio Krasnogor et al. *Nature inspired cooperative strategies for optimization (nicso 2007)*. Vol. 129. Springer, 2008.

[44] Gilbert Laporte, Ardavan Asef-Vaziri, and Chelliah Sriskandarajah. "Some applications of the generalized travelling salesman problem". In: *Journal of the Operational Research Society* 47.12 (1996), pp. 1461–1467.

[45] Gilbert Laporte, Luís Lopes, and François Soumis. "Optimal sequencing rules for some large-scale flexible manufacturing problems under the Manhattan and Chebychev metrics". In: *International journal of flexible manufacturing systems* 10 (1998), pp. 27–42.

[46] Gilbert Laporte, Hélène Mercure, and Yves Nobert. "Generalized travelling salesman problem through n sets of nodes: the asymmetrical case". In: *Discrete Applied Mathematics* 18.2 (1987), pp. 185–197.

[47] Gilbert Laporte and Yves Nobert. "Generalized travelling salesman problem through n sets of nodes: an integer programming approach". In: *INFOR: Information Systems and Operational Research* 21.1 (1983), pp. 61–75.

[48] Gilbert Laporte and Frédéric Semet. "Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem". In: *INFOR: Information Systems and Operational Research* 37.2 (1999), pp. 114–120.

[49] In Gyu Lee, Sung Hoon Chung, and Sang Won Yoon. "Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations". In: *Computers & industrial engineering* 139 (2020), p. 106129.

[50] Adam N Letchford, Saeideh D Nasiri, and Dirk Oliver Theis. "Compact formulations of the Steiner traveling salesman problem and related problems". In: *European Journal of Operational Research* 228.1 (2013), pp. 83–92.

[51] Jianbin Li, Rihuan Huang, and James B Dai. "Joint optimisation of order batching and picker routing in the online retailer's warehouse in China". In: *International Journal of Production Research* 55.2 (2017), pp. 447–461.

[52] Yi Li, Ruining Zhang, and Dandan Jiang. "Order-Picking Efficiency in E-Commerce Warehouses: A Literature Review". In: *Journal of Theoretical and Applied Electronic Commerce Research* 17.4 (2022), pp. 1812–1830.

[53] Yao-Nan Lien, Eva Ma, and Benjamin W-S Wah. "Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem". In: *Information Sciences* 74.1-2 (1993), pp. 177–189.

[54] Chun-Cheng Lin et al. "Joint order batching and picker Manhattan routing problem". In: *Computers & Industrial Engineering* 95 (2016), pp. 164–174.

[55] Shen Lin. "Computer solutions of the traveling salesman problem". In: *Bell System Technical Journal* 44.10 (1965), pp. 2245–2269.

[56] Shen Lin and Brian W Kernighan. "An effective heuristic algorithm for the traveling-salesman problem". In: *Operations research* 21.2 (1973), pp. 498–516.

[57] Wenrong Lu et al. "An algorithm for dynamic order-picking in warehouse operations". In: *European Journal of Operational Research* 248.1 (2016), pp. 107–122.

[58] Makusee Masae, Christoph H Glock, and Eric H Grosse. "Order picker routing in warehouses: A systematic literature review". In: *International Journal of Production Economics* 224 (2020), p. 107564.

[59] Makusee Masae, Christoph H Glock, and Panupong Vichitkunakorn. "Optimal order picker routing in the chevron warehouse". In: *IISE Transactions* 52.6 (2020), pp. 665–687.

[60] Marek Matusiak, René De Koster, and Jari Saarinen. "Utilizing individual picker skills to improve order batching in a warehouse". In: *European Journal of Operational Research* 263.3 (2017), pp. 888–899.

[61] Marek Matusiak et al. "A fast simulated annealing method for batching precedence-constrained customer orders in a warehouse". In: *European Journal of Operational Research* 236.3 (2014), pp. 968–977.

[62] Borja Menéndez et al. "Variable neighborhood search strategies for the order batching problem". In: *Computers & Operations Research* 78 (2017), pp. 500–512.

[63] Olegs Nalivajevs and Daniel Karapetyan. "Conditional Markov Chain Search for the Generalised Travelling Salesman Problem for Warehouse Order Picking". In: *2019 11th Computer Science and Electronic Engineering (CEEC)*. IEEE. 2019, pp. 75–78.

[64] Charles E Noon and James C Bean. "A Lagrangian based approach for the asymmetric generalized traveling salesman problem". In: *Operations Research* 39.4 (1991), pp. 623–632.

[65] Charles E Noon and James C Bean. "An efficient transformation of the generalized traveling salesman problem". In: *INFOR: Information Systems and Operational Research* 31.1 (1993), pp. 39–44.

[66] Johan Oscar Ong and Don Thomas Joseph. "A review of order picking improvement methods". In: *J@ ti Undip: Jurnal Teknik Industri* 9.3 (2014), pp. 135–138.

[67] Lucie Pansart, Nicolas Catusse, and Hadrien Cambazard. "Exact algorithms for the order picking problem". In: *Computers & Operations Research* 100 (2018), pp. 117–127.

[68] Ulrich Pferschy and Joachim Schauer. "Order batching and routing in a non-standard warehouse". In: *Electronic Notes in Discrete Mathematics* 69 (2018), pp. 125–132.

[69] H Donald Ratliff and Arnon S Rosenthal. "Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem". In: *Operations research* 31.3 (1983), pp. 507–521.

[70] Kees Jan Roodbergen and René De Koster. "Routing methods for warehouses with multiple cross aisles". In: *International Journal of Production Research* 39.9 (2001), pp. 1865–1883.

[71] Kees-Jan Roodbergen. *Layout and routing methods for warehouses*. EPS-2001-004-LIS. 2001.

[72] Raad Salman, Fredrik Ekstedt, and Peter Damaschke. "Branch-and-bound for the precedence constrained generalized traveling salesman problem". In: *Operations Research Letters* 48.2 (2020), pp. 163–166.

[73] André Scholz and Gerhard Wäscher. "Order batching and picker routing in manual order picking systems: the benefits of integrated routing". In: *Central European Journal of Operations Research* 25.2 (2017), pp. 491–520.

[74] André Scholz et al. "A new mathematical programming formulation for the single-picker routing problem". In: *European Journal of Operational Research* 253.1 (2016), pp. 68–84.

[75] Lahari Sengupta, Radu Mariescu-Istodor, and Pasi Fränti. "Which local search operator works best for the open-loop TSP?" In: *Applied Sciences* 9.19 (2019), p. 3985.

[76] Xiaohu H Shi et al. "Particle swarm optimization-based algorithms for TSP and generalized TSP". In: *Information processing letters* 103.5 (2007), pp. 169–176.

[77] John Silberholz and Bruce Golden. "The generalized traveling salesman problem: A new genetic algorithm approach". In: *Extending the horizons: advances in computing, optimization, and decision technologies*. Springer, 2007, pp. 165–181.

[78] Stephen L Smith and Frank Imeson. "GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem". In: *Computers & Operations Research* 87 (2017), pp. 1–19.

[79] Lawrence V Snyder and Mark S Daskin. "A random-key genetic algorithm for the generalized traveling salesman problem". In: *European journal of operational research* 174.1 (2006), pp. 38–53.

[80] Curtis L Stowers and Udatta S Palekar. "Lot sizing problems with strong set-up interactions". In: *IIE transactions* 29.2 (1997), pp. 167–179.

[81] Christophe Theys et al. "Towards a metaheuristic for routing order pickers in a warehouse". In: *Evolutionary methods for design, optimization and control* (2007), pp. 385–390.

[82] Christophe Theys et al. "Using a TSP heuristic for routing order pickers in warehouses". In: *European Journal of Operational Research* 200.3 (2010), pp. 755–763.

[83] James A Tompkins et al. *Facilities planning*. John Wiley & Sons, 2010.

[84] Jimi Tuononen. "Analysis of rebuild local search operator for TSP". MA thesis. Itä-Suomen yliopisto, 2022.

[85] Cristiano Arbex Valle, John E Beasley, and Alexandre Salles Da Cunha. "Optimally solving the joint order batching and picker routing problem". In: *European Journal of Operational Research* 262.3 (2017), pp. 817–834.

[86] Teun Van Gils et al. "Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review". In: *European Journal of Operational Research* 267.1 (2018), pp. 1–15.

[87] Sarah Vanheusden et al. "Practical factors in order picking planning: state-of-the-art classification and review". In: *International Journal of Production Research* (2022), pp. 1–25.

[88] Christos Voudouris and Edward Tsang. "Guided local search and its application to the traveling salesman problem". In: *European journal of operational research* 113.2 (1999), pp. 469–499.

[89] Christos Voudouris, Edward PK Tsang, and Abdullah Alsheddy. "Guided local search". In: *Handbook of metaheuristics*. Springer, 2010, pp. 321–361.

[90] Andres Weintraub et al. "An emergency vehicle dispatching system for an electric utility in Chile". In: *Journal of the Operational Research Society* 50.7 (1999), pp. 690–696.

[91] Sven Winkelhaus et al. "Hybrid order picking: A simulation model of a joint manual and autonomous order picking system". In: *Computers & Industrial Engineering* 167 (2022), p. 107981.

[92] Jinhui Yang et al. "An ant colony optimization method for generalized TSP problem". In: *Progress in Natural Science* 18.11 (2008), pp. 1417–1422.

[93] Xi Zhao and Xiao-Ping Zhu. "Innovative genetic algorithm for solving GTSP". In: *2010 Second International Conference on Modeling, Simulation and Visualization Methods*. IEEE. 2010, pp. 239–241.

[94] Li Zhou et al. "Performance analysis of three intelligent algorithms on route selection of fishbone layout". In: *Sustainability* 11.4 (2019), p. 1148.

[95] Ivan Žulj et al. "Picker routing and storage-assignment strategies for precedence-constrained order picking". In: *Computers & Industrial Engineering* 123 (2018), pp. 338–347.