CIRRELT

CIRRELT-2019-50

Integrating Storage Location and Order Picking Problems in Warehouse Planning

Allyson Silva
Leandro C. Coelho
Maryam Darvish
Jacques Renaud

November 2019

Document de travail également publié par la Faculté des sciences de l'administration de l'Université Laval, sous le numéro FSA-2019-015.

Bureau de Montréal
Université de Montréal
C.P. 6128, succ. Centre-Ville
Montréal (Québec) H3C 3J7
Tél. : 1-514-343-7575
Télécopie : 1-514-343-7121

Bureau de Québec
Université Laval,
2325, rue de la Terrasse
Pavillon Palasis-Prince, local 2415
Québec (Québec) G1V 0A6
Tél. : 1-418-656-2073
Télécopie : 1-418-656-2624

Integrating Storage Location and Order Picking Problems in Warehouse Planning

Allyson Silva*, Leandro C. Coelho, Maryam Darvish, Jacques Renaud

Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Department of Operations and Decision Systems, 2325 de la Terrasse, Université Laval, Québec, Canada G1V 0A6

Abstract: Storage location and order picking are two interdependent problems arising in warehouse planning traditionally solved independently. We introduce the joint storage location and order picking problem in which routes are created using classical policies. Five variants are analyzed and modeled, some for the first time. These models are proven to be difficult to solve, even for small warehouses and few orders. Therefore, we present a General Variable Neighborhood Search metaheuristic, which is proved to be very efficient. The solutions for the integrated approach generated using our metaheuristic indicate that better storage leads to significant savings compared to common storage practices.

Keywords: Storage location, order picking, warehouse, General Variable Neighborhood Search.

Acknowledgements: This work was partly supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grants 2019-00094 and 2018-03712. We thank Compute Canada for providing high-performance parallel computing facilities. This support is gratefully acknowledged.

---

* Corresponding author: Allyson.Silva@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2019

1. Introduction

The most resource-intensive process performed at a warehouse is the order picking [37], which is highly dependent on the storage location policy used. The storage location may be performed immediately after the reception of a product or periodically. Its purpose is to place products in convenient locations where they can be easily picked during the picking process. Order picking refers to the activities performed to retrieve products from their storage locations to satisfy demands specified by customer orders. Due to labor intensity in such systems, the order picking process alone concentrates 50 to 75% of the total operating costs for a typical warehouse [12]. Thus, warehouses are often optimized for cost-efficient order picking.

The problem of deciding where to locate products in a warehouse is known as storage location assignment problem (SLAP). It consists of determining the most efficient assignment of products to locations in order to minimize the total handling effort. The pickings are performed by a routing strategy to be followed by the pickers to collect the required products, which is called the order picking problem (OPP). The SLAP solution is an input of the OPP, since routes can be created only after product locations are known. At the same time, a SLAP solution can only be evaluated when the strategy to solve the OPP is known.

The SLAP and OPP as individual problems have been extensively studied [7, 14]. When considered together, they are usually solved sequentially for different combinations of storage location and picking policies. These policies are usually very simple heuristics used to create a solution for each problem. They present advantages such as being easily memorized, reducing costs caused by pick errors, and they are easily accepted by the pickers [29, 30]. Little attention had yet been given to integrating problems in warehouse planning [42].

In this paper, we integrate the SLAP and OPP considering the most common warehouse layout found in practice, i.e., the rectangular single-block layout with multiple aisles. We name this integrated approach as the storage location and order picking problem (SLAP/OPP). We present a cubic mixed integer programming (MIP) model to solve the SLAP/OPP. We cannot ignore that heuristic routing policies are still widely used, so we also present MIP models to solve the SLAP considering four routing policies known as return (SLAP/Re), S-shape (SLAP/Ss), midpoint (SLAP/Mp) and largest gap (SLAP/Lg). We also propose a General Variable Neighborhood Search (GVNS) metaheuristic framework to approximate optimal solutions for all five models. Computational experiments attest to

the effectiveness of the GVNS in solving all problems for the small instances. We also test it for regular size instances, as used in literature, to analyze the advantage of solving the integrated version of the problems. We show that the integrated approach can yield large gains in the picking performance.

The remainder of this paper is organized as follows. In Section 2, we review the literature of the SLAP and the OPP. The MIP models for the SLAP/OPP and the other four variations of the SLAP, and their linearizations, are presented in Section 3. The proposed GVNS is presented in Section 4. Section 5 provides the computational experiments results. Finally, our conclusions follow in Section 7.

2. Background on the storage location assignment and order picking problems

We present a brief review on the storage location assignment and the order picking problems. For complementary information, we refer to the literature reviews in De Koster et al. [7], Gu et al. [14, 15], Reyes et al. [34] and Van Gils et al. [42].

2.1. Storage location assignment problem

The assignment of products to storage locations in the warehouse is known as the SLAP and aims at minimizing material handling effort. This problem considers the storage area layout, the set of orders to be fulfilled and assumes a picking policy. The SLAP was introduced by Hausman et al. [17] for an automated warehouse, and it has been widely studied since then (e.g., Carlo and Giraldo [2], Kofler et al. [21]).

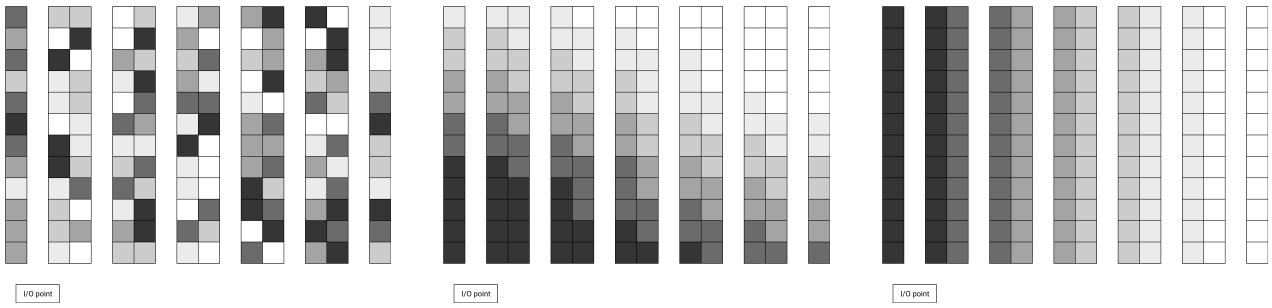
The SLAP is related to the quadratic assignment problem (QAP), which is considered to be one of the hardest combinatorial optimization problems to solve optimally due to its nonlinear objective function [10]. In their review, Reyes et al. [34] report that although exact methods exist to solve SLAP variations, they usually are not used due to the complexity of the problem.

Since SLAPs are very hard problems to solve using exact methods, typically simple heuristics are used to generate the product layout. These heuristics are typically divided into the random, dedicated, and class-based categories. This taxonomy was presented by Hausman et al. [17] and it is still widely in use. The choice of the most appropriate policy depends on the available product information [2, 14].

The *random storage* consists of the random assignment of products to locations in the storage area. Its main advantage is its simplicity, but the downside is obviously the longer travel times for pickers to retrieve products [6, 21, 29]. Tappia et al. [39] also point that when fast moving products change

rapidly over time, like in e-commerce warehouses, it is hard to implement other storage policies than the random one. The *dedicated storage* policy fixes the location of high-turnover products to "best" locations. Products are sorted using a demand based rule and locations sorted based on the distance to the I/O point and, then, the best products are assigned to the best locations. Several rules used to sort products and locations are discussed in literature [21, 28, 41]. The *class-based storage* policy is an alternative for when demand forecasting is possible but not very precise for each individual item. The class-based policy divides products into classes and storage locations into zones, then assigns each class to a zone. The final position of a product within its zone is usually determined randomly. A different way to classify items is by their affinity. Items that are frequently ordered together are said to be affine. Some studies [22, 44] use the QAP formulation to allocate products based on affinity, although not creating explicit classes. References for studies that use affinity-based rules in the classification of products are found in Kofler et al. [21].

Figure 1 provides an example of random and class-based policies (diagonal and within-aisle metrics). Squares with the same background color represent products classified in the same class. We note that the random and the dedicated storage policies are special cases of the class-based one for when the number of classes is equal to one and to the number of products to be located, respectively.



(a) Random **(b)** Diagonal **(c)** Within-aisle

Figure 1: Storage policies

2.2. Order picking problem

The objective of the OPP is to optimize a performance measure, e.g., minimize time or distance traveled by the picker. For pick lists with only one product, the OPP optimal solution is easily generated by solving a shortest path problem for the round trip between the I/O point and the product location. For a pick list with products in multiple locations, an optimal OPP solution is obtained by solving a special case of the Traveling Salesman Problem (TSP) [1], given that there is

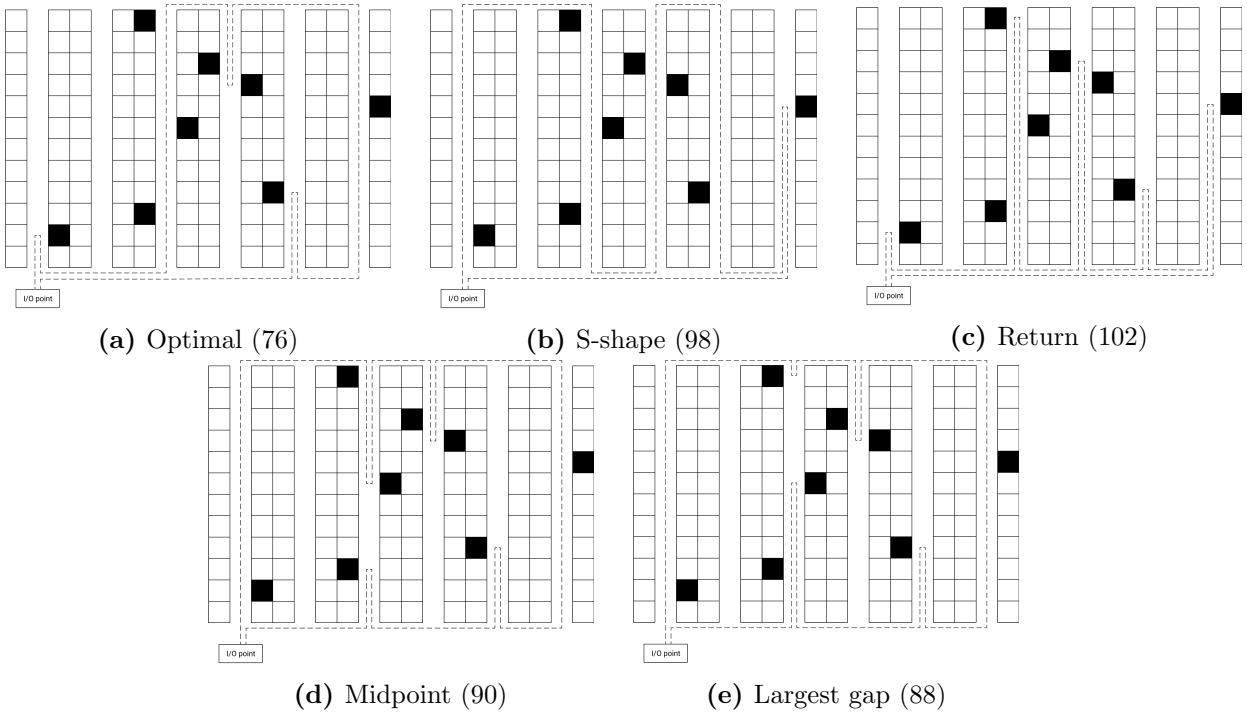
enough capacity for the picker to retrieve all products in only one route. In the OPP, each item in the pick list, as well as the I/O point, corresponds to a vertex and the distance between two vertices is set to be the shortest distance between their corresponding locations in the warehouse. Formulations for the OPP adapted from the TSP literature are found in Pansart et al. [25] and Scholz et al. [38].

Any warehouse layout can be represented using a TSP formulation. Even when special conditions apply, like for narrow aisles [3] or scattered warehouses [43]. While the TSP is a hard problem to solve, the OPP can be solved efficiently under certain conditions. Ratliff and Rosenthal [33] introduce an optimal algorithm for a rectangular single-block warehouse which is solvable in polynomial time using dynamic programming. The two-block case is described in Scholz and Wäscher [37]. Heuristics used for the TSP may be used to solve the OPP as well. The most successful one is the Lin-Kernighan-Helsgaun (LKH) [18]. Theys et al. [40] used LKH to approximate optimal OPP routes reporting an average optimality gap of 0.1% for different warehouse settings.

While LKH searches for near-optimal OPP solutions, certain routing policies provide simple solutions that can be easily memorized and executed by the pickers. These routing policies are chosen according to the problem characteristics, such as warehouse shape, number of aisles and cross aisles, pick list size, the storage and batching policies, etc. Van Gils et al. [42] provided an extensive list of studies that analyze the interaction of these factors on the performance of the chosen routing policy.

Several routing policies are found in the OPP literature. The *return policy* considers that each picker enters and leaves through the same end of aisles containing pick items. In the *S-shape policy*, also known as traversal policy, a picker enters an aisle that contains a product to be picked and traverses it until its end. A return is allowed in the furthest location containing a pick of the last aisle visited, if it is advantageous [35]. In the *midpoint policy*, pickers are able to go as far as the middle of the aisle before returning and leaving the aisle from where they entered. Exceptions are applied only for the first and last aisles containing items so that the picker can access the opposite cross aisle and return to the I/O point after picking all items. In the *largest gap policy*, pickers enter aisles as far as to the point where the item that has the largest gap to another item, or the closest cross aisle, is located. Gaps are calculated as the distance between two adjacent items in the aisle or, if the item is the first or last item in the aisle, the distance between this item and the closest cross aisle. Then, the pickers return and leave at the same end that they entered. As in the midpoint policy, a picker never crosses the entire aisle except for the first and last ones containing items. Other policies include the combined and the aisle-by-aisle [27, 35]. Examples of each of these routing policies are presented in Figure 2,

along with the optimal route and its cost in parentheses. Although most policies presented here are for a single-block layout, they are easily adaptable to multi-block cases [35].



(a) Optimal (76) **(b)** S-shape (98) **(c)** Return (102)

(d) Midpoint (90) **(e)** Largest gap (88)

Figure 2: Routing policies

2.3. Discussions on the interactions between SLAP and OPP

When analyzing the SLAP and OPP simultaneously, the main question is which OPP policy performs the best for a given SLAP policy, and vice-versa?

When deciding the route in the OPP, the product locations must be known prior to solving the problem. Nevertheless, the storage locations may be determined even if a routing strategy is not assumed *a priori*. Studies that analyze the interactions between the two problems consider that the SLAP and the OPP are solved in sequence. First, products are allocated to storage locations using a storage policy. Then, different routing policies are tested and their performances are evaluated to determine the overall performance of the system.

Table 1 presents an overview of the studies that considered multiple policies for the SLAP and the OPP to analyze well-performing combinations under different circumstances.

All cited studies so far consider the decisions on both problems taken sequentially. However, if the routing policy is given as an input for the SLAP, the storage area setup can be determined optimally

Table 1: Overview of the papers that investigated multiple storage and routing policies

Reference	*Storage policies*					*Routing policies*	
	Random	*Class-based*		*Dedicated*		*Optimal*	*Heuristic*
		Demand	*Distance*	*Demand*	*Distance*		
Chan and Chan [4]	✓	COI	Wa/Ot				Ss/Re/Ot
Chen et al. [5]	✓	Fr	Wa/Ot				Ss/Re/Lg/Ot
Dekker et al. [8]		Fr	Wa/Ot			✓	Ss/Mp/Lg/Ot
Dijkstra and Roodbergen [9]		Fr	Di/Wa/Ot				Ss/Re/Mp/Lg
Hsieh and Huang [19]	✓	Fr	Wa			✓	Ss/Ot
Hsieh and Tsai [20]		Fr/Af	Di				SS/Re
Pan and Wu [24]	✓			Fr	Di/Wa/Aa		Ss/Re/Ot
Petersen [28]	✓			Fr	Di/Wa	✓	Ss/Lg/Ot
Petersen and Aase [29]	✓	Fr	Wa	Fr	Wa	✓	Ss/Ot
Petersen et al. [30]				Fr	Di/Wa/Ot	✓	Ss
Quader and Castillo-Villar [32]	✓	Fr	Wa/Ot				Ss/Re/Mp/Lg
Roodbergen et al. [36]	✓	Fr	Di/Wa/Ot				Ss/Re/Lg/Ot
Theys et al. [40]	✓			Fr	Wa	✓	Ss/Lg/Ot
Van Gils et al. [41]	✓	Fr	Di/Wa/Ot			✓	Ss/Re/Lg/Ot

*Fr: Frequency-based, Af: Affinity-based, COI: Cube per order index, Wa: Within-aisle, Di: Diagonal, Ss: S-shape, Re: Return, Mp: Midpoint, Lg: Largest gap, Ot: Other

for that policy. An investigation on optimal storage locations for the most common routing policies is presented in Mantel et al. [22] and Dijkstra and Roodbergen [9]. The former presents a MIP for the SLAP for an S-shape routing where a return is not allowed in the last aisle. The latter presents a mathematical model to generate near-optimal assignments using dynamic programming for the four routing policies presented here. The route lengths are calculated from picking probabilities for product retrieval instead of static pick lists as considered here. Their findings confirm the best combinations of storage and routing policies, e.g., return combines with diagonal, S-shape with within-aisle, and largest gap and midpoint both with within-aisle, among other storage policies. However, Dijkstra and Roodbergen [9] did not evaluate the SLAP performance for the optimal routing policy, which is one of the objectives of this study.

3. Mathematical models

In this section, we present MIP models to solve the SLAP for the optimal routing policy (SLAP/OPP), as well as for the four routing policies considered (SLAP/Re, SLAP/Ss, SLAP/Mp and SLAP/Lg). The SLAP/OPP model is derived from a TSP one. It is designed in such a way to accept any warehouse structure, so it is not required to know specific layout parameters *a priori*, such as the number of aisles, number of slots, etc. We introduce it after the special case for a rectangular single-block warehouse layout, which is used as a reference to model the other four SLAPs. Finally, the linearization technique used for all nonlinear models is briefly described.

3.1. General SLAP/OPP

A general formulation for the SLAP/OPP is described next. We consider a set $\mathcal{L} = \{1, \ldots, L\}$ of locations representing the slots available, and $\mathcal{L}^* = \mathcal{L} \cup \{0\}$ the set including location 0 representing the I/O point. A set $\mathcal{P} = \{1, \ldots, P\}$ of products to be located is defined, with $\mathcal{P}^* = \mathcal{P} \cup \{0\}$ being the set that includes a dummy product 0 that is assigned to the I/O point. Each product in $\mathcal{P}$ must be assigned to exactly one location in $\mathcal{L}$, thus $P \leq L$. Products not demanded by any order may be ignored from $\mathcal{P}$ without loss of generality of the model. A set $\mathcal{O} = \{1, \ldots, O\}$ of orders is known, and each order o contains a pick list $\mathcal{Q}_o \subset \mathcal{P}^*$, so that the products to be picked in each route are known in advance. Since all routes start and end in the I/O point, the product 0 is contained in all pick lists. The matrix $\mathbf{D} = \{d_{ij}\}$ represents the cost (distance, time, etc.) for a picker to move from location i to location j. The binary routing variable x_{ij}^o is equal to one if the route to pick the items in the o-th order contains the path going from location i to location j, or zero otherwise. We assume that there is enough capacity for the picker to retrieve all products from o in only one route. The assignment variable y_{ki} is also binary and equals to one if product k is assigned to location i. Finally, we define auxiliary variables u_k^o, which are used to create a valid route for order o, representing in which position product $k \in \mathcal{Q}_o \setminus \{0\}$ is retrieved in the route. The SLAP/OPP is formulated as follows:

$$\text{Min} \sum_{o \in \mathcal{O}} \sum_{k \in \mathcal{Q}_o} \sum_{l \in \mathcal{Q}_o \setminus \{k\}} \sum_{i \in \mathcal{L}^*} \sum_{j \in \mathcal{L}^* \setminus \{i\}} d_{ij} x_{ij}^o y_{ki} y_{lj}, \tag{1}$$

subject to

$$y_{00} = 1, \tag{2}$$

$$\sum_{i \in \mathcal{L}} y_{ki} = 1, \quad \forall k \in \mathcal{P}, \tag{3}$$

$$\sum_{k \in \mathcal{P}} y_{ki} \leq 1, \quad \forall i \in \mathcal{L}, \tag{4}$$

$$\sum_{k \in \mathcal{Q}_o} y_{ki} = \sum_{j \in \mathcal{L}^* \setminus \{i\}} x_{ij}^o, \quad \forall o \in \mathcal{O}, i \in \mathcal{L}^*, \tag{5}$$

$$\sum_{k \in \mathcal{Q}_o} y_{kj} = \sum_{i \in \mathcal{L}^* \setminus \{j\}} x_{ij}^o, \quad \forall o \in \mathcal{O}, j \in \mathcal{L}^*, \tag{6}$$

$$u_k^o - u_l^o + |\mathcal{Q}_o| x_{ij}^o y_{ki} y_{lj} \leq |\mathcal{Q}_o| - 1, \quad \forall o \in \mathcal{O}, k, l \in \mathcal{Q}_o \setminus \{0\}, k \neq l, i, j \in \mathcal{L}^*, i \neq j, \tag{7}$$

$$0 \leq u_k^o \leq |\mathcal{Q}_o| - 1, \quad \forall k \in \mathcal{Q}_o \setminus \{0\}, o \in \mathcal{O}, \tag{8}$$

$$x_{ij}^o \in \{0, 1\}, \quad \forall o \in \mathcal{O}, i, j \in \mathcal{L}^*, i \neq j, \tag{9}$$

$$y_{ki} \in \{0,1\}, \quad \forall k \in \mathcal{P}, i \in \mathcal{L}^*. \tag{10}$$

The objective function (1) minimizes the total routing cost. Constraint (2) is used to define the I/O point at location 0. Constraints (3) assign each product to exactly one location. Constraints (4) define that each location can have no more than one product assigned to it. Constraints (5) and (6) link the assignment and routing variables. They ensure that when there is a product k in order o located in a certain slot, then one path must be chosen that leaves and enters this slot. Constraints (7) eliminate subtours in the routes by guaranteeing that if product k is located in i, product l is located in j, and slot j is visited after slot i in the route, then $u_l^o > u_k^o$, where l and k are products in the pick list. These constraints are derived from the Miller-Tucker-Zemlin (MTZ) formulation for the TSP as presented in Scholz et al. [38] for the OPP. Constraints (8) define the bound of the auxiliary variables, while constraints (9) and (10) define the domain of the routing and assignment variables, respectively.

We now describe the specific warehouse layout case examined in this paper. We consider a picker-to-product system for a rectangular warehouse with a single block. This is a common layout studied in literature (e.g., Dijkstra and Roodbergen [9], Petersen and Aase [29], Roodbergen and De Koster [35]). The storage area has a set $\mathcal{A} = \{1, \ldots, A\}$ of parallel aisles with a set $\mathcal{B} = \{1, \ldots, B\}$ of rows of slots. Each row, including those in the first and last aisles, contains two slots, i.e., one in the left rack and another in the right rack. We ignore the distance between the two slots in the same row (two-sided picking). We denote (a, b) as the slot located in aisle a and row b. Due to the two-sided picking, we can ignore if it is on the left or right side of the row. A cross aisle exists both in the front and the back of the aisles. The distance a picker has to move from the entrance of an aisle to the entrance of an adjacent aisle is equal to M. Each slot has the capacity for a product with a sufficient number of single items meaning that no replenishment is needed during the picking. All products have the same physical requirements (space, temperature, etc.). The distance between two neighbor rows within the same aisle is N, which is also the distance a picker has to travel to move from the head of the aisle in any cross aisle to the first row in the aisle. Pickers may perform a return in the aisles at no cost. The effects of congestion is negligible, i.e., time to remove items is small enough to avoid pick-column blocking, and aisles are wide enough to avoid in-the-aisle blocking. Finally, an I/O point is located at the head of the first aisle. We illustrate this setting in Figure 3. It represents the floor plan of a warehouse with $A = 5$ aisles and $B = 5$ rows in each aisle.

In order to solve the SLAP/OPP for the single-block warehouse, the cost matrix $\mathbf{D}$ is determined by

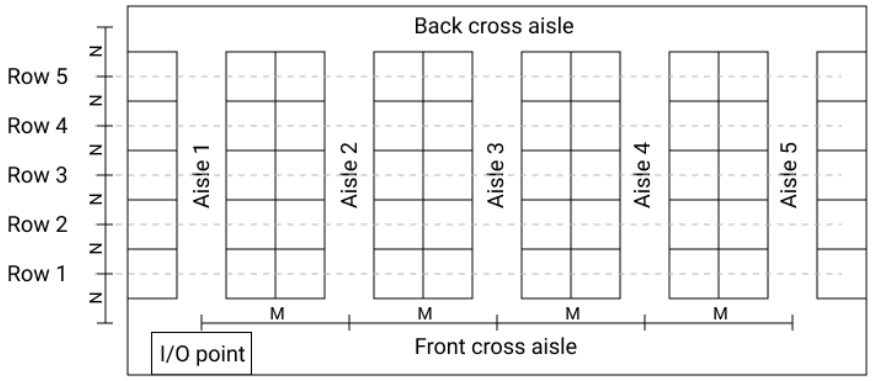


Figure 3: Representation of a warehouse with two-sided parallel aisles in a single block

solving a special case of the shortest path problem between every pair of slots. The minimum distance d_{ij} between two slots $i = (a_i, b_i)$ and $j = (a_j, b_j)$ is calculated as:

$$d_{ij} = M(|a_i - a_j|) + \begin{cases} N|b_i - b_j|, & \text{if } a_i = a_j, \\ N \min\{(b_i + b_j), (2B + 2 - b_i - b_j)\}, & \text{otherwise.} \end{cases} \tag{11}$$

The first term gives the distance traveled in the cross aisles, while the second term gives the distance within-aisle either when both slots are in the same aisle or when they are in different aisles. The *min* function represents the decision of using the cross aisle in the front or in the back of the storage area to change aisles. The SLAP/OPP may now be solved using the distance matrix as defined.

3.2. SLAP for different routing policies

It is also possible to model the SLAP when considering any other routing policy. A base model for the SLAP is defined next.

Consider D_o^{CA} as being the distance traveled by the picker in the cross aisles and D_o^{WA} as the distance traveled within the aisles to pick products from order o. For any routing strategy, the objective of the SLAP is to minimize the total distance traveled to fulfill all required orders given as

$$\min \sum_{o \in \mathcal{O}} \left(D_o^{CA} + D_o^{WA} \right). \tag{12}$$

In order to calculate D_o^{CA} and D_o^{WA}, we should define the locations of the products to be picked in each route. Considering the set of products ($\mathcal{P}$) to be located, and the single-block layout as described

in Section 3.1, we define the binary location variable y_{ab}^k as equal to one if product $k \in \mathcal{P}$ is assigned to slot (a, b), and zero otherwise. Thus, the following constraints can be used to assign all products in $\mathcal{P}$ to locations (constraints (13)) such that no location has more than two products (14), one on each side. Thus, D_o^{CA} and D_o^{WA} are functions dependent of y.

$$\sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} y_{ab}^k = 1, \quad \forall k \in \mathcal{P}, \tag{13}$$

$$\sum_{k \in \mathcal{P}} y_{ab}^k \leq 2, \quad \forall a \in \mathcal{A}, b \in \mathcal{B}, \tag{14}$$

$$y_{ab}^k \in \{0, 1\}, \quad \forall k \in \mathcal{P}, a \in \mathcal{A}, b \in \mathcal{B}. \tag{15}$$

Although the distances may be represented directly by a function of y, for the sake of easiness to understand the models, we define new auxiliary variables to compute D_o^{CA} and D_o^{WA} for different routing policies. We summarize in Table 2 the new auxiliary variables used in the four models we designed to solve the SLAP with return, S-shape, midpoint, and largest gap routing policies, also indicating in which models they are used.

Table 2: Notation of the models

Variable	Description	Re	Ss	Mp	Lg
f_{oa}	Furthest row containing a pick in aisle a	✓	✓	✓	✓
z_{oa}	If there is a pick in aisle a	✓	✓	✓	✓
v_o	Last aisle containing a pick	✓		✓	✓
v_{oa}	If aisle a is the last aisle containing a pick		✓	✓	✓
k_o	Auxiliary variable used to calculate s_o^{SS}		✓		
s_o^{SS}	(S-shape special case) If the number of aisles with picks is odd		✓		
f_{oa}^1	Furthest row from lower cross aisle below midpoint containing a pick in aisle a			✓	
f_{oa}^2	Furthest row from upper cross aisle above midpoint containing a pick in aisle a			✓	
z_{oa}^1	If there is a pick in a row below midpoint in aisle a			✓	
z_{oa}^2	If there is a pick in a row above midpoint in aisle a			✓	
u_o^2	First aisle containing a pick above midpoint			✓	
u_{oa}^2	If aisle a is the first aisle containing a pick above midpoint			✓	
s_o^{MP}	(Midpoint special case) If $u_o^2 \neq 0$ and $u_o^2 \neq v_o$			✓	
w_{oa}^{MP}	If aisle a is neither the first aisle above midpoint nor the last aisle with a pick			✓	
G_{oa}	Largest gap between two neighbor picks or a pick and a neighbor cross aisle in aisle a				✓
g_{oab}	Gap between a pick in row $b \in B \cup \{0\}$ to nearest pick in another row or cross aisle in aisle a. When $b = 0$ we have the largest gap from the lower cross aisle to the first pick				✓
h_{oab}	Auxiliary variable used to calculate the largest gap				✓
u_o	First aisle containing a pick				✓
u_{oa}	If aisle a is the first one containing a pick				✓
s_o^{LG}	(Largest gap special case) If $u_o \neq v_o$				✓
w_{oa}^{LG}	If aisle a is between the first and last aisles with a pick				✓

We now present the mathematical models for the SLAP under the return, S-shape, midpoint, and largest gap routing policies for the warehouse layout considered before.

3.2.1. SLAP with return policy (SLAP/Re)

Considering the sets, variables, and parameters as previously defined, we represent the distances traveled in the SLAP with the return policy as follows. For any route, the distance traveled in the cross aisle is $D_o^{CA} = 2M(v_o - 1)$, i.e., the distance to travel back and forth to the last aisle containing a product to be picked. Since the I/O point is located in the head of the first aisle, the subtraction $(v_o - 1)$ is required. Furthermore, the distance traveled within all aisles is $D_o^{WA} = 2N \sum_{a \in A} f_{oa}$, which represents the distance to go back and forth from the cross aisle to the furthest location that contains a product in each aisle. In order to calculate v_o and f_{oa}, we added the following constraints to the base model (12)–(15) to create a model for the SLAP/Re:

$$f_{oa} \geq \sum_{b \in \mathcal{B}} b y_{ab}^k, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, k \in \mathcal{Q}_o \setminus \{0\}, \tag{16}$$

$$f_{oa} \leq B z_{oa}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{17}$$

$$\textbf{if } z_{oa} = 1 \textbf{ and } \sum_{c=(a+1)}^{A} z_{oc} = 0 \rightarrow v_o = a, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{18}$$

$$z_{oa} \in \{0, 1\}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{19}$$

$$0 \leq f_{oa} \leq B, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{20}$$

$$1 \leq v_o \leq A, \quad \forall o \in \mathcal{O}. \tag{21}$$

Constraints (16) define a lower bound for f_{oa}. Since we minimize f_{oa} in the objective function, there is no need to limit its upper bound. Constraints (17) set z_{oa} to one if f_{oa} assumes a value greater than zero. z_{oa} is used to determine if a is the last aisle containing a product from order o, then $v_o = a$. These constraints can be written using indicator (or logical) constraints available in modern MIP solvers as presented in constraints (18). The remaining constraints (19)–(21) define the domain of the decision variables.

3.2.2. SLAP with S-shape policy (SLAP/Ss)

For the SLAP with S-shape routing policy, the distances traveled are calculated as follows. The distance in the cross aisles is $D_o^{CA} = \sum_{a \in \mathcal{A}} 2M v_{oa}(a - 1)$, which is similar to the one for the return policy, but using the equivalent binary decision variables v. The use of a binary v is more adequate to calculate the distance within-aisles given as $D_o^{WA} = \sum_{a \in \mathcal{A}} (N(B+1)z_{oa} + 2N s_o^{SS} f_{oa} v_{oa}) - N(B+1)s_o^{SS}$.

The expression $N(B+1)$ represents the distance traveled when the entire aisle is crossed by the picker. So, according to the first term of the equation, when there are products to be picked in an aisle, the picker crosses it entirely. Since this term also considers the last aisle independently of the value of s_o^{SS}, we have to add the last two terms to correct the distance traveled in the last aisle when the picker performs a return in it. Thus, the SLAP/Ss is modeled using the base model (12)–(15), constraints (16), (17), (19) and (20) to calculate z_{oa} and f_{oa}, and the following constraints to determine v_{oa} and s_o^{SS}:

$$\textbf{if } z_{oa} = 1 \textbf{ and } \sum_{c=(a+1)}^{A} z_{oc} = 0 \rightarrow v_{oa} = 1, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{22}$$

$$\sum_{a \in \mathcal{A}} z_{oa} = 2k_o + s_o^{SS}, \quad \forall o \in \mathcal{O} \tag{23}$$

$$v_{oa} \in \{0,1\}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{24}$$

$$s_o^{SS} \in \{0,1\}, \quad \forall o \in \mathcal{O}, \tag{25}$$

$$0 \leq k_o \leq A/2, \quad \forall o \in \mathcal{O}. \tag{26}$$

Constraints (22) are the equivalent indicator constraints to set the appropriate value for binary variable v. The variable s_o^{SS} is calculated as the remainder of $\frac{\sum_{a \in \mathcal{A}} z_{oa}}{2}$ using constraints (23). Constraints (24)–(26) define the domain of the variables v_{oa}, s_o^{SS} and k_o, respectively. We note that our model is rather different than that of Mantel et al. [22] because we consider the return option at the last aisle, which makes the problem significantly harder to solve.

3.2.3. SLAP with midpoint policy (SLAP/Mp)

To the formulation of the SLAP with midpoint policy, we have divided the storage area into two zones. Midpoint is defined as the row $p = \lceil B/2 \rceil$. Zone 1 ($\mathcal{B}^1 = \{1, \ldots, p\}$) contains all locations below p, including row p itself. Meanwhile, zone 2 ($\mathcal{B}^2 = \{p+1, \ldots, B\}$) contains all locations above p. This definition is used to create variables z^1 and z^2, which are the z equivalent of each zone, as well as f^1 and f^2, which are their f equivalents. These, along with other exclusive auxiliary variables for this policy, are used to calculate the distances traveled as follows. As in the return policy, $D_o^{CA} = 2M(v_o - 1)$. Furthermore, $D_o^{WA} = 2N((B+1)s_o^{MP} + \sum_{a \in A}(f_{oa}^1 w_{oa}^{MP} + f_{oa}^2 w_{oa}^{MP} + f_{oa} v_{oa} - f_{oa} v_{oa} s_o^{MP}))$. This long equation can be explained by dividing it into three parts. The first part ($2N(B+1)s_o^{MP}$) is the distance traveled to cross the first and last aisles entirely when s_o^{MP} is activated, i.e., there is at least one aisle

with a product to be picked above the midpoint and it is different than the last aisle with a product to be picked. The second part ($\sum_{a\in A}(f^1_{oa}w^{MP}_{oa} + f^2_{oa}w^{MP}_{oa})$) is the distance traveled to the rows closest to the midpoint in all aisles except for the first and the last. The third part ($\sum_{a\in A}(f_{oa}v_{oa} - f_{oa}v_{oa}s^{MP}_o)$) models the special case when there is only one aisle with products to be picked above midpoint and it is the last aisle. In this case, we ignore the zones and calculate the furthest row containing a product to be picked in the last aisle using variable f to determine the return point within the aisle. The SLAP/Mp is modeled using the base model (12)–(15), constraints (16) and (18) to calculate f_{oa} and v_o, and the following constraints to determine f^1_{oa}, f^2_{oa}, z^1_{oa}, z^2_{oa}, z_{oa}, u^2_o, u^2_{oa}, v_{oa}, s^{MP}_o, and w^{MP}_{oa}:

$$\textbf{if } \sum_{b\in\mathcal{B}^r}\sum_{k\in\mathcal{Q}_o} y^k_{ab} = 0 \rightarrow f^r_{oa} = 0, \quad o\in\mathcal{O}, a\in\mathcal{A}, r = \{1,2\}, \tag{27}$$

$$\textbf{if } \sum_{k\in\mathcal{Q}_o} y^k_{ab} \geq 1 \textbf{ and } \sum_{d=(b+1)}^{p}\sum_{k\in\mathcal{Q}_o} y^k_{ad} = 0 \rightarrow f^1_{oa} = b, \quad o\in\mathcal{O}, a\in\mathcal{A}, b\in\mathcal{B}^1 \tag{28}$$

$$\textbf{if } \sum_{k\in\mathcal{Q}_o} y^k_{ab} \geq 1 \textbf{ and } \sum_{d=p+1}^{b-1}\sum_{k\in\mathcal{Q}_o} y^k_{ad} = 0 \rightarrow f^2_{oa} = (B+1) - b, \quad o\in\mathcal{O}, a\in\mathcal{A}, b\in\mathcal{B}^2 \tag{29}$$

$$\textbf{if } f^r_{oa} = 0 \rightarrow z^r_{oa} = 0, \quad o\in\mathcal{O}, a\in\mathcal{A}, r = \{1,2\}, \tag{30}$$

$$f^r_{oa} \leq Bz^r_{oa}, \quad o\in\mathcal{O}, a\in\mathcal{A}, r = \{1,2\}, \tag{31}$$

$$z_{oa} \geq z^r_{oa}, \quad o\in\mathcal{O}, a\in\mathcal{A}, r = \{1,2\}, \tag{32}$$

$$z_{oa} \leq \sum_{r=1}^{2} z^r_{oa}, \quad o\in\mathcal{O}, a\in\mathcal{A}, \tag{33}$$

$$\textbf{if } \sum_{a\in\mathcal{A}} z^2_{oa} = 0 \rightarrow u^2_o = 0, \quad o\in\mathcal{O}, \tag{34}$$

$$\textbf{if } z^2_{oa} = 1 \textbf{ and } \sum_{c=1}^{a-1} z^2_{oc} = 0 \rightarrow u^2_o = a, \quad o\in\mathcal{O}, a\in\mathcal{A}, \tag{35}$$

$$\textbf{if } u^2_o \neq 0 \textbf{ and } u^2_o \neq v_o \rightarrow s^{MP}_o = 1, \quad o\in\mathcal{O}, \tag{36}$$

$$\textbf{if } u^2_o = 0 \textbf{ or } u^2_o = v_o \rightarrow s^{MP}_o = 0, \quad o\in\mathcal{O}, \tag{37}$$

$$u^2_o = \sum_{a\in\mathcal{A}} au^2_{oa}, \quad o\in\mathcal{O}, \tag{38}$$

$$\sum_{a\in\mathcal{A}} u^2_{oa} \leq 1, \quad o\in\mathcal{O}, \tag{39}$$

$$v_o = \sum_{a\in\mathcal{A}} av_{oa}, \quad o\in\mathcal{O}, \tag{40}$$

$$\sum_{a\in\mathcal{A}} v_{oa} \leq 1, \quad o\in\mathcal{O}, \tag{41}$$

$$w_{oa}^{MP} \leq 1 - u_{oa}^2, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{42}$$

$$w_{oa}^{MP} \leq 1 - v_{oa}, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{43}$$

$$w_{oa}^{MP} \geq 1 - u_{oa}^2 - v_{oa}, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{44}$$

$$0 \leq f_{oa}^1, f_{oa}^2 \leq p, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{45}$$

$$z_{oa}^1, z_{oa}^2, z_{oa}, u_{oa}^2, v_{oa}, s_o^{MP}, w_{oa}^{MP} \in \{0,1\}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{46}$$

$$0 \leq u_o^2 \leq A, \quad \forall o \in \mathcal{O}. \tag{47}$$

Constraints (27)–(29) determine the furthest row to be traveled in each zone of each aisle. Constraints (30) and (31) set the proper values to variables z^1 and z^2. Constraints (32) and (33) set the proper value to variables z. Constraints (34) and (35) determine the first aisle with products to be picked above the midpoint. Constraints (36) and (37) set s^{MP} to its appropriate value. Constraints (38) set proper values to the integer and binary versions of u^2 and constraints (39) certify that no more than one binary is activated. Constraints (40) and (41) do the same for v. Constraints (42)–(44) set w^{MP} to its correct value. Finally, constraints (45)–(47) define the domain of the variables.

3.2.4. SLAP with largest gap policy (SLAP/Lg)

The SLAP with largest gap policy is modeled as follows. The distance traveled in the cross aisles is similar to the return and midpoint, given as $D_o^{CA} = 2M(v_o - 1)$. To calculate the distance traveled within aisles, like in the SLAP/Mp, we have to consider the different cases when only the lower cross aisle or both cross aisles are used. The first case is obeserved when all picked items are located in a single aisle. If more than two aisles should be entered, then the first and last are entirely crossed, while in the intermediary ones returns are performed to avoid the distance from the largest gap. Thus, the distance within-aisle is given as $D_o^{WA} = 2N((B+1)s_o^{LG} + \sum_{a \in A}(s_o^{LG} w_{oa}^{LG}(B + 1 - G_{oa}) + (1 - s_o^{LG}) f_{oa} v_{oa}))$. The first term in the formula $((B+1)s_o^{LG})$ represents the first and last aisles crossed entirely when s^{LG} is active. The second term $(\sum_{a \in A}(s_o^{LG} w_{oa}^{LG}(B + 1 - G_{oa})))$ is the distance traveled in the intermediary aisles given by the total number of rows minus the largest gap in that aisle. The third term $(\sum_{a \in A}((1 - s_o^{LG}) f_{oa} v_{oa}))$ is the special case when all picks are in the same aisle, so that a simple return at the furthest row containing a pick is performed. Now, we model the SLAP/Lg using the base model (12)–(15), constraints (16)–(18), (40), (41) to calculate f_{oa}, z_{oa}, v_o, v_{oa}, and the

following constraints to determine u_o, u_{oa}, g_{oab}, G_{oa}, w_{oa}^{LG}, and h_{oab}:

$$\textbf{if } z_{oa} = 1 \textbf{ and } \sum_{c=1}^{a-1} z_{oc} = 0 \rightarrow u_o = a, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{48}$$

$$u_o = \sum_{a \in \mathcal{A}} a u_{oa}, \quad o \in \mathcal{O}, \tag{49}$$

$$\sum_{a \in \mathcal{A}} u_{oa} = 1, \quad o \in \mathcal{O}, \tag{50}$$

$$\textbf{if } u_o \leq (a-1) \textbf{ and } v_o \geq (a+1) \rightarrow w_{oa}^{LG} = 1, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{51}$$

$$\textbf{if } u_o \geq a \textbf{ or } v_o \leq a \rightarrow w_{oa}^{LG} = 0, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{52}$$

$$\textbf{if } u_o = v_o \rightarrow s_o^{LG} = 0, \quad o \in \mathcal{O}, \tag{53}$$

$$\textbf{if } u_o \neq v_o \rightarrow s_o^{LG} = 1, \quad o \in \mathcal{O}, \tag{54}$$

$$\textbf{if } z_{oa} = 0 \rightarrow g_{oa0} = 0, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{55}$$

$$\textbf{if } \sum_{k \in \mathcal{Q}_o} y_{oab}^k \geq 1 \textbf{ and } \sum_{d=1}^{b-1} \sum_{k \in \mathcal{Q}_o} y_{oad}^k = 0 \rightarrow g_{oa0} = b, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}, \tag{56}$$

$$\textbf{if } \sum_{k \in \mathcal{Q}_o} y_{oab}^k = 0 \rightarrow g_{oab} = 0, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}, \tag{57}$$

$$\textbf{if } \sum_{k \in \mathcal{Q}_o} y_{oab_1}^k \geq 1 \textbf{ and } \sum_{k \in \mathcal{Q}_o} y_{oab_2}^k \geq 1$$
$$\textbf{and } \sum_{b_3=(b_1+1)}^{b_2-1} \sum_{k \in \mathcal{Q}_o} y_{oab_3}^k = 0 \rightarrow g_{oab_1} = b_2 - b_1, \quad o \in \mathcal{O}, a \in \mathcal{A}, b_1, b_2 \in \mathcal{B}, \tag{58}$$

$$\textbf{if } \sum_{k \in \mathcal{Q}_o} y_{oab_1}^k \geq 1 \textbf{ and } \sum_{b_2=(b_1+1)}^{B} \sum_{k \in \mathcal{Q}_o} y_{oab_2}^k = 0 \rightarrow g_{oab_1} = B + 1 - b_1, \quad o \in \mathcal{O}, a \in \mathcal{A}, b_1 \in \mathcal{B}, \tag{59}$$

$$G_{oa} \geq g_{oab}, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}, \tag{60}$$

$$G_{oa} \leq g_{oab} + \sum_{b_2 \in B \cup \{0\}} (B+1) h_{oab_2}, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}, \tag{61}$$

$$\sum_{b \in \mathcal{B} \cup \{0\}} h_{oab} = 1, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{62}$$

$$u_o, g_{oab}, G_{oa} \geq 0, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{63}$$

$$u_{oa}, w_{oa}^{LG}, h_{oab} \in \{0, 1\}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}. \tag{64}$$

Constraints (48) determine the first aisle with a pick. Constraints (49) and (50) set proper values for u_o and u_{oa}. Constraints (51) and (52) use indicator constraints to ensure that $w_{oa}^{LG} = 1$ when

$u_o < a < v_o$, and zero otherwise. Constraints (53) and (54) guarantee that s^{LG} is equal to one only if the first and last aisles are different. If there is no pick at all in the aisle, then the gap is set to zero in constraints (55). Constraints (56) make sure that if there is a product in row b and there is no product between the lower cross aisle and row $b-1$, then the gap from the lower cross aisle to the first pick is b. Constraints (57) set the gap in row b to zero when there is no pick in b. Constraints (58) state that if there is a pick in b_1 and another in b_2 and no other pick between the two, then the gap in row b_1 is equal to $b_2 - b_1$. Constraints (59) handle the special case when the pick in b_1 is the last pick in the aisle. So, the gap in b_1 is the distance from it to the upper cross aisle. Constraints (60)–(62) are linear forms to compute the largest value between all gaps, i.e., $G_{oa} = \max(g_{oab} | b \in \mathcal{B} \cup \{0\})$. Finally, constraints (63) and (64) define the domain of the new variables.

3.3. Linearizations

The introduced models, except for the SLAP/Re, are nonlinear MIP formulations due to the product of variables either in the objective function or in constraints, as in constraints (7). Nonlinear MIP models can be reformulated to an equivalent linear representation using a linearization technique. In order to solve the problems, we reformulated the models using the standard linearization technique presented in Glover and Woolsey [13]. The technique is used to replace the product of variables as follows.

- product of two binary variables a and b is treated by replacing ab by a new variable c and adding the constraints $c \leq a$, $c \leq b$ and $c \geq a + b - 1$;
- product of three binary variables a, b, and c, abc is replaced by d and the constraints $d \leq a$, $d \leq b$, $d \leq c$ and $d \geq a + b + c - 2$;
- product of binary variable a and non-binary variable b, ab is replaced by c by adding the constraints $c \leq \overline{b}a$, where $\overline{b}$ is the upper bound value b can assume, $c \leq b$, and $c \geq b - \overline{b}(1 - a)$.

The advantage of linearizing a problem is that it can be solved by any integer linear programming solver. While the standard linearization benefits from its simplicity, it has the disadvantage of requiring a large number of auxiliary variables and constraints to replace the nonlinear terms.

4. A Variable Neighborhood Search for the SLAP

Introduced in Mladenović and Hansen [23], the Variable Neighborhood Search (VNS) is a metaheuristic that uses a systematic change of neighborhood within a descent phase to find a local optimum starting from different points generated by a perturbation phase. Among its several variants, the one with some of the most successful applications is the General VNS (GVNS) [16]. In the GVNS, the descent phase consists in the application of several local searches (LS) performed in a deterministic way by a method called Variable Neighborhood Descent (VND). The reasoning of VND is that a local optimum in a neighborhood is not necessarily a local optimum in another, such that the use of several LSs are more likely to reach global optimum [16]. The implemented VND is shown in Algorithm 1.

We first define an initial solution, as described in Section 4.1. Then, a set $\mathcal{N}_l$ of neighborhoods is defined with $l = \{1, 2, 3\}$ representing the three LSs, presented in Section 4.2, used for the exploration, and their exploration order is known *a priori* (line 1). The first improving solution found in a neighborhood replaces the current one (*first improvement* strategy) until there is no possibility of improvement in that neighborhood (lines 4–6). Preliminary experiments showed that the first improvement strategy performed better than searching the whole neighborhood for the best improving solution which is too time consuming. The search returns to the first neighborhood when the solution is improved in the current neighborhood. Otherwise, it continues in the next neighborhood (lines 7–11). The solution returned by the VND (line 13) should be a local minimum with respect to all three neighborhoods.

Algorithm 1 Function VND(x)

```
1: Neighborhoods N_l(x), l = {1,2,3};
2: repeat
3:     x_l ← x;
4:     while there is a x' ∈ N_l(x)|f(x') < f(x) do
5:         x ← x';
6:     end while
7:     if f(x) < f(x_l) then
8:         l = 1;
9:     else
10:        l = l + 1;
11:    end if
12: until l < 4
13: return x.
```

The perturbation phase of the GVNS consists in applying a function (*Shake*) as presented in Algorithm 2. This function is used to move from solution x to a random solution in one of its neighborhoods. Preliminary experiments showed that the *Slot Exchange* neighborhood performs particularly well. In summary, each shake consists in selecting two random and different products and swap their locations

in the storage area (more details about this neighborhood is presented in Section 4.2). A total of S swaps is performed before *Shake* ends. Note that small values of S tend to lead to the same local optimum when VND is applied. Otherwise, if S is too high, the new solution may lose the good characteristics of the initial solution, thus behaving like a randomly generated solution.

Algorithm 2 Function Shake(x, S)

```
1: for s = 1 to S do
2:     Randomly select i, j ∈ P|i ≠ j;
3:     x ← SlotExchange(x, i, j);
4: end for
5: return x.
```

The steps of our GVNS are presented in Algorithm 3. Besides an initial solution x, it receives three parameters: the maximum number of cycles K, the maximum number of shakes in each cycle S, and the time limit T. We start the GVNS by applying the VND to x (line 2). The stopping criteria are either when the number K of cycles is reached or the run time reaches T. In each cycle, we iterate a counter s until the maximum number of shakes (S) is reached. For each s, a shake followed by a VND is performed (lines 5–6). If the objective function of the new local optimum improves the previous best known solution (BKS), s is reset, and the BKS is updated. Otherwise, s is incremented (lines 7–12). The shake counter is reset at the end of each cycle (line 15).

Algorithm 3 Function GVNS (x, K, S, T)

```
1: k = 1, s = 1, t = 0;
2: x' ← VND(x);
3: while k ≤ K and t ≤ T do
4:     while s ≤ S do
5:         x' ← Shake(x', s);
6:         x' ← VND(x');
7:         if f(x') < f(x) then
8:             x ← x';
9:             s = 1;
10:        else
11:            s = s + 1;
12:        end if
13:    end while
14:    k = k + 1;
15:    s = 1;
16:    t ← Run time;
17: end while
18: return x.
```

4.1. Generating an initial solution for the SLAP

Regardless of the routing policy, a feasible solution for the SLAP may be generated using any storage policy, such as those presented in Section 2.1. The policies are used to assign products in $\mathcal{P}$ to locations

in $\mathcal{L}$ (i.e., to an aisle a and row b in the single-block layout).

The storage policies implemented are random, dedicated with frequency-based and diagonal policy, and dedicated with frequency-based and within-aisle policy. The product frequency is calculated by the number of times it appears in the orders $\mathcal{O}$. In the diagonal policy, the product with the highest frequency is assigned to the closest slot to the I/O point, and so forth. In case of ties, the decisions are made randomly. For the within-aisle policy, the distance from the closest slot of each aisle to the I/O point is measured and the aisles are sorted accordingly. The closest aisle is then filled with the products with the highest frequencies, and so on.

Given the SLAP solution generated, the picking route for each order is created to determine the objective function value for the problem. The routes are created according to the routing policy used. We implemented the five policies as for the models previously introduced, i.e., SLAP/OPP, SLAP/Re, SLAP/Ss, SLAP/Mp and SLAP/Lg. When the SLAP solution is known, we note that the SLAP/OPP reduces to a TSP for each order. While for the other policies the computation of the routes is straightforward, we highlight that the largest gap in each aisle can be calculated by solving the maximum gap problem [31], which can be solved in $O(n)$ using the pigeonhole principle for n points of interest in the aisle, i.e., the items to be retrieved in it and the two end aisles.

4.2. Local Searches

We defined three neighborhoods to search for local improvements during VND. They are all based on the swap of products locations in the warehouse. Whenever the storage setup is changed, a new call for the routing policy is made to reevaluate the picking routes. A picking route is reevaluated only if a product contained in its pick list is involved in the change.

The first neighborhood is defined as the swap of products assigned to different locations. The *slot exchange* LS finds two slots with an ordered product located in at least one of them and swaps their assignments. All combinations of pairs of slots are tested. Since only slots containing products are swapped, this neighborhood size is $O(|\mathcal{P}||\mathcal{L}|)$ in the worst case. A neighbor is an improving solution either if its objective function is better or, specially for this LS, when it has equal value, but the swap moves a product with higher frequency closer to the I/O point. This second condition proved to be very effective in avoiding bad local optimum.

The second neighborhood is defined as the swap of products contained in different rows. The *row exchange* LS finds two rows with at least one ordered product in at least one of them, even if they

are in different aisles, and swaps their products. The row exchange neighborhood is equal to the slot exchange neighborhood if the capacity of a row is equal to one. However, due to the two-sided picking in the single-block warehouse layout considered, the row exchange allows the exchange of up to four products simultaneously instead of two as in the slot exchange. The row exchange is applied for all pairs of rows with products. This neighborhood size is $O(|\mathcal{L}|^2)$ in the worst case, i.e., when there are products located in all existing rows and no improving solution is found.

The third neighborhood is defined as the swap of all products contained in different aisles. The *aisle exchange* LS finds two aisles with at least one ordered product in at least one of them and swaps their products. The position of the products in the original aisle is respected in their new aisle. This LS requires that the number of slots in the two swapped aisles be the same. For a warehouse with A identical aisles, this neighborhood size is $O(A^2)$ in the worst case.

5. Computational Experiments

The SLAP and OPP literature lack a well-established set of benchmark instances to represent real warehouses so that different methods could be compared on the same set. Instead, researchers usually create their own test instances based on the problem and the circumstances explored. We designed our instances using the parameters presented in Table 3 and share them on `https://www.leandro-coelho.com/slot-assignment-and-order-picking/`. All instances consider the distances as $M = N = 1$. The small set is used to evaluate the MIP models and to adjust the GVNS parameters, while the regular set, consisting of instances similar in size to those used in the literature [25, 37, 38], is solved only by the GVNS. Column *Aisles* represents the number of identical two-sided picking aisles in the storage area, *Rows per aisle* is the number of rows in each aisle, each one containing two slots, *Orders* corresponds to the number of orders to be fulfilled, and *Products per order* is the number of unique products contained in each order. The orders were generated using three different distributions, as shown in the *Demand* column. The *Random* distribution selects the products using a uniform distribution, while the *Skewed* distribution uses the Pareto principle to divide products into three classes. The *Skewed A/B/C* notation is used to state that items from the smaller group appear A% of the times, the medium group appears B%, and the bigger group appears C%. We used $A/B/C$ as either 50/30/20 or 80/15/5. Due to the low number of items in small instances, only the random distribution is used. Three different instances are created for each combination of these five parameters. The result is 108 small and 486 regular size instances. We highlight that the largest instances

in the small set have the same parameters as the smallest instances in the regular set. However, they are generated with different seeds, resulting in different instances.

Table 3: Warehouse layout and demand parameters used to generate the experimental instances

Set	*Aisles (A)*	*Rows per aisle (B)*	*Orders ($\|\mathcal{O}\|$)*	*Products per order ($\|\mathcal{Q}_o\|$)*	*Demand*
Small	1 aisle	5 rows	1 order	3 products	Random
	3 aisles	10 rows	5 orders	5 products	
	5 aisles		10 orders		
Regular	5 aisles	10 rows	10 lists	5 products	Random
	10 aisles	50 rows	30 lists	20 products	Skewed 50/30/20
	20 aisles		50 lists	50 products	Skewed 80/15/5

Computational experiments are performed in the Beluga cluster of Compute Canada national system using single cores of an Intel Gold 6148 Skylake CPU with 2.4 GHz and with a memory limit of 16 GB of RAM. The linear MIPs are solved using the CPLEX C++ API (version 12.5). The routes for the SLAP/OPP, when solved exactly, are generated using the Concorde TSP solver. As reported by Theys et al. [40], the LKH heuristic approximates very well the optimal OPP solution. Concorde provides an implementation of the LKH heuristic, which is used to solve the SLAP/OPP with the GVNS. Based on preliminary experiments, we fixed the number of 4-swap kicks in the LKH heuristic to 10 kicks each time it is used. The other routing policies are implemented in C++.

5.1. Solving the Linear MIP Model

The first set of experiments consists in solving the instances in the small set using the linear MIP models for all five SLAPs considered here. The problems are solved using CPLEX with standard parameters and a time limit of two hours. The objective is to evaluate the maximum instance size for which an optimal solution can be proved using these models. For that, we group the instances from the small set according to the number of slots in the warehouse ($2AB$), and the number of items picked in a route ($|\mathcal{O}||\mathcal{Q}_o|$). Thus, each combination of Slots $\times$ Picks contains three instances.

Table 4 shows the average time for each instance group solved to optimality. Each cell is colored with a shade of grey representing the number of instances solved in that group, with darker cells representing more instances solved.

The results show that the linear MIP model for the SLAP/OPP is of no practical use even for small instances with only three or five picks, which have optimal solutions that could be found intuitively. We also highlight how fast the average time to prove optimality increases by adding more slots to the warehouse, even with only three picks. The SLAP/OPP model is the only one to run out of

Table 4: Average time to prove optimality of small instances using the linear MIP models for all five SLAPs

Problem	*Slots ($A \times B$)*	*Picks ($\|\mathcal{O}\| \times \|\mathcal{Q}_o\|$)*					
		3 (1 × 3)	*5 (1 × 5)*	*15 (5 × 3)*	*25 (5 × 5)*	*30 (10 × 3)*	*50 (10 × 5)*
SLAP/OPP	*10 (1 × 5)*	0.3s	5.1s	-	-	-	-
	20 (1 × 10)	3.3s	-	-	-	-	-
	30 (3 × 5)	146.3s	-	-	-	-	-
	50 (5 × 5)	3435.6s	-	-	-	-	-
	60 (3 × 10)	7099.9s	-	-	-	-	-
	100 (5 × 10)	-	-	-	-	-	-
SLAP/Re	*10 (1 × 5)*	<0.1s	<0.1s	<0.1s	0.1s	3 0.1s	0.1s
	20 (1 × 10)	<0.1s	<0.1s	0.1s	0.9s	1.7s	22.8s
	30 (3 × 5)	<0.1s	<0.1s	0.8s	20.5s	83.1s	-
	50 (5 × 5)	<0.1s	<0.1s	4.6s	173.0s	181.9s	-
	60 (3 × 10)	<0.1s	<0.1s	0.7s	36.2s	216.5s	-
	100 (5 × 10)	<0.1s	0.1s	42.5s	294.3s	-	-
SLAP/Ss	*10 (1 × 5)*	<0.1s	<0.1s	<0.1s	0.1s	<0.1s	0.1s
	20 (1 × 10)	<0.1s	<0.1s	0.1s	0.7s	1.5s	21.7s
	30 (3 × 5)	<0.1s	<0.1s	1.0s	5.3s	296.3s	420.7s
	50 (5 × 5)	<0.1s	<0.1s	3.1s	21.6s	940.9s	-
	60 (3 × 10)	<0.1s	<0.1s	0.9s	15.1s	306.8s	-
	100 (5 × 10)	<0.1s	<0.1s	15.7s	57.0s	409.1s	-
SLAP/Mp	*10 (1 × 5)*	<0.1s	<0.1s	0.1s	0.2s	0.9s	1.4s
	20 (1 × 10)	<0.1s	<0.1s	1.2s	4.2s	10.7s	210.8s
	30 (3 × 5)	0.1s	0.1s	136.4s	1199.1s	-	-
	50 (5 × 5)	0.1s	0.2s	862.6s	4321.4s	-	-
	60 (3 × 10)	0.1s	0.2s	447.0s	3174.8s	-	-
	100 (5 × 10)	0.3s	0.9s	2984.6s	-	-	-
SLAP/Lg	*10 (1 × 5)*	<0.1s	<0.1s	0.4s	0.8s	2.5s	4.8s
	20 (1 × 10)	0.1s	0.1s	2.0s	25.4s	54.6s	6453.6s
	30 (3 × 5)	0.1s	0.2s	28.1s	71.6s	896.6s	-
	50 (5 × 5)	0.2s	0.3s	148.2s	387.2s	-	-
	60 (3 × 10)	0.4s	0.9s	155.1s	3018.8s	-	-
	100 (5 × 10)	1.5s	2.0s	656.0s	5044.5s	-	-

*each cell contains the average over three instances. Dark gray shows that all three instances are solved. Medium gray indicates that only two instances are solved to optimality, and light gray indicates that only one instance is solved to optimality

memory, which is in line with Forrester [11] about the standard linearization technique requiring a huge number of variables and constraints. The linearization of the cubic term in the objective function in the SLAP/OPP requires the introduction of $O(|\mathcal{O}||\mathcal{L}|^2|\mathcal{P}^2|)$ new variables and constraints, which for large instances of the small set represents hundreds of millions of new variables.

When solving the SLAP using the alternative routing policy models, the solver was able to find optimality in most of the small instances. The SLAP/Re and SLAP/Ss, which have simpler models in terms of number of variables and constraints, can solve instances slightly bigger than the SLAP/Mp and SLAP/Lg. We note that although there are cubic terms in some of the objective functions, they require the addition of only $O(|\mathcal{O}|A)$ new variables and constraints, significantly fewer than in the SLAP/OPP.

5.2. GVNS setting

The difficulty of solving bigger instances from the small set indicates that an alternative to deal with them is required. We now show how the GVNS was set for the experiments to solve the regular size set.

5.2.1. Setting the search strategy

In the neighborhood search strategy setting, we demonstrate through experiments the reason why the application of the three neighborhoods proposed in our VND is more effective than the application of only one or two of them. The experiment consists in creating initial solutions for the SLAP using the three storage policies implemented and by applying VND for each of them using different combinations of LSs. On the first experimentation round, we tested an improvement strategy of using a single LS exhaustively, i.e., until no improving solution can be found. In the second round, we tested the VND using all pairs of LSs in all possible orders. We also performed a third round of experiments by adding the third LS to all the combinations of the second round. We then tested each sequence on all instances of the small instance set for which comparison is possible with the MIP, as presented in the previous section. The results indicated that the average gap significantly decreases when more neighborhoods are explored, and when all three of them are performed, the order is not important (unlike when only two of them are used). For this reason, and to increase diversity, for the remaining experiments, we set the neighborhood exploration order to be randomly decided at the beginning of every VND. This leads to a reduction in the search bias during GVNS cycles, which helps avoid local minima. We also note that the whole exploration of the three neighborhoods is performed within one second, even for the largest instances of the set.

5.2.2. Analysis of the initial solution development

Having demonstrated that a single call to VND may significantly improve a solution generated by the simple storage policies for the SLAP with any routing policy, we use the proposed GVNS to solve all small instances. From preliminary experiments, we set the maximum number of shakes $S = 5$. Each instance of the small set is solved 10 times, starting from each of the three storage policies solutions for each one of the four SLAPs with heuristic routing policy. We do not consider the SLAP/OPP since the optimal was already found by only using a single VND call. In order to observe solution convergence, each run is performed up to $K = 20,000$ cycles. We calculate gaps to the BKS, either a

proven optimal or the best feasible solution known, at different parts of the GVNS. The first solution observed is the initial solution, represented by when the four SLAPs with routing policies are solved using the three common storage policies considered here random, diagonal, and within-aisle. The next solution observed is that after VND is applied to improve the initial solution (Algorithm 3, line 2). Then, we observe the current best solution for several values of K. We highlight that although solutions are very different in the first step when only the routing policies are considered, a very fast convergence is obtained after the application of the first VND and even more when GVNS begins to the point that at $K = 20$ the difference between solutions is no more than 0.01%. An important remark is that a convergence to better solutions is obtained using any of the three storage policies. This is evidence of the robustness of this metaheuristic framework when improving solutions for the SLAPs. The results of these tests indicate that solutions are improved up to $K = 10,000$, within a run time of less than two minutes. For these reasons, we choose this setting for the remaining experiments.

5.3. Solving the regular size set

We now perform detailed experiments to evaluate our algorithm on the regular size instance set. The GVNS is run with the previous determined parameters ($S = 5$ and $K = 10000$). Initial solutions are generated using the random storage policy. Each regular size instance is solved five times. The results are presented for each problem setting, so that it is possible to analyze their impacts on the policies used for each problem.

5.3.1. Solution improvement over the change of the storage policy

We analyze the potential improvement on route length by changing the storage policy from a heuristic to the optimal one considering that the routing policy is maintained. For each instance of the four SLAPs, we consider two solutions: (i) the best initial solution among the three generated using a storage policy (random, diagonal and within-aisle); and (ii) the average solution of the five runs of GVNS for the SLAP with the respective heuristic routing policy with a two-hour time limit. In Table 5, we present the improvement from (i) to (ii) in column *Improv. over initial* calculated as $improv(x) = \dfrac{f(x) - f(x')}{f(x)}$, where x is the solution (i) and x' is the solution (ii). Column *Time* reports the average time of each run to generate (ii). In each line of the table, average results are presented for the parameters shown on the left.

We can draw three main conclusions from these results. First, regardless of the storage assignment

Table 5: Comparison of the SLAP solutions solved by GVNS for the regular size data set with the initial solution generated by the storage policies

		SLAP/Re		*SLAP/Ss*		*SLAP/Mp*		*SLAP/Lg*	
		Improv. over initial (%)	*Time (s)*	*Improv. over initial (%)*	*Time (s)*	*Improv. over initial (%)*	*Time (s)*	*Improv. over initial (%)*	*Time (s)*
A	*5*	40.67	5560.1	28.60	5607.6	42.05	5964.8	33.25	6630.6
	10	51.03	6326.5	40.58	6531.6	52.58	6697.0	45.46	6948.6
	20	**59.02**	6978.7	**50.28**	6968.0	**58.40**	7034.6	**53.29**	7129.0
B	*10*	38.08	5473.6	32.87	5653.2	40.90	5997.9	35.47	6633.1
	50	**62.40**	7103.2	**46.77**	7085.0	**61.12**	7133.0	**52.52**	7172.4
$\mathcal{O}$	*10*	**56.48**	5504.5	40.88	5548.7	**55.86**	5966.0	44.83	6447.6
	30	49.61	6559.8	**41.03**	6651.0	50.60	6754.5	**45.40**	7059.1
	50	44.64	6800.9	37.55	6907.6	46.57	6975.9	41.76	7201.5
$\mathcal{Q}_o$	*5*	**53.63**	5157.8	**55.03**	5287.6	**53.33**	5569.4	**53.42**	6278.3
	20	53.17	6628.1	36.78	6750.6	52.74	6923.0	37.83	7204.0
	50	43.92	7079.4	27.65	7069.1	46.96	7204.0	40.75	7226.0
D	*Random*	**55.37**	6306.5	**45.88**	6386.8	**55.27**	6587.0	**49.65**	6902.2
	Skew 50	51.89	6286.6	41.01	6408.6	52.73	6573.5	45.98	6905.6
	Skew 80	43.46	6272.1	32.57	6311.9	45.03	6535.9	36.36	6900.4

strategy and routing policy used, the search for optimal storage location assignments significantly improves the solutions generated by the heuristic storage policies. Improvements obtained average from 27% (SLAP/Ss and $\mathcal{Q}_o = 50$) to 62% (SLAP/Re and $B = 50$).

Second, the average running time of GVNS is close to the imposed two-hour time limit in all cases. This is because almost all runs were stopped by the time limit rather than by the maximum number of cycles. As showed in Section 5.2.2, by stopping the search before 10,000 cycles, solutions may not yet have converged.

Finally, more savings in the picking process are obtained when optimal storage locations are explored compared to heuristic storage policies, when warehouses are bigger, when there are fewer orders and with fewer items, and when demands are less skewed. This pattern is observed in almost all scenarios tested as boldened in the table.

5.3.2. Comparison of SLAP solutions for different routing policies

The solutions from the last experiment for the SLAP with different routing policies are now detailed and compared between them. Table 6 shows the average route length given by GVNS for all five SLAPs. We have grouped all instances with different demand distributions in each row of the table, presenting the results for combinations of the remaining instance attributes. So, it is possible to identify the routing policy that GVNS finds better solutions under each combination of warehouse size and orders' characteristics.

Table 6: Average route length found by GVNS for different routing policies and instance settings

A	B	$\mathcal{O}$	$\mathcal{Q}_o$	*SLAP/Re*	*SLAP/Ss*	*SLAP/Mp*	*SLAP/Lg*	*SLAP/OPP*
5	10	10	5	**108.9**	144.7	**108.9**	144.7	109.1
			20	350.2	317.8	**292.4**	302.8	299.0
			50	717.0	542.5	549.5	559.4	**532.4**
		30	5	501.9	612.6	**496.6**	609.2	502.5
			20	1467.1	1249.1	**1141.8**	1176.4	1153.4
			50	2533.5	1874.4	1978.7	1941.2	**1756.8**
		50	5	999.2	1158.7	**966.8**	1146.3	969.6
			20	2696.3	2260.1	2081.4	2121.1	**2066.5**
			50	4413.7	3230.1	3469.4	3361.5	**3032.6**
	50	10	5	113.4	134.3	**113.2**	134.7	115.5
			20	374.7	820.5	**374.6**	830.3	584.0
			50	**924.2**	1161.4	949.7	1134.8	1398.7
		30	5	582.5	974.8	**582.4**	995.7	604.2
			20	**2224.1**	3108.0	2389.1	3183.9	3293.8
			50	4935.2	5273.2	**4016.8**	4411.5	5850.1
		50	5	1303.1	2262.2	**1296.7**	2278.1	1355.8
			20	4997.4	5982.8	**4899.9**	5745.6	6041.0
			50	10430.4	9706.9	**7854.7**	8490.0	11120.2
10	10	10	5	**110.0**	134.9	**110.0**	134.9	111.4
			20	364.9	360.2	**320.3**	324.4	340.7
			50	798.0	710.0	**668.9**	689.3	681.2
		30	5	515.1	606.6	**512.1**	604.2	523.8
			20	1686.4	1621.9	**1448.0**	1473.7	1483.9
			50	3266.1	2705.8	**2631.2**	2632.2	2709.0
		50	5	1058.9	1235.7	**1048.8**	1214.0	1068.7
			20	3302.0	3100.4	**2733.5**	2779.7	2790.7
			50	6051.6	4886.1	4834.8	4766.7	**4704.6**
	50	10	5	**114.8**	117.6	**114.8**	117.6	116.5
			20	**325.1**	804.8	325.4	839.9	549.2
			50	**787.6**	1188.8	803.0	1179.0	2330.0
		30	5	**541.2**	733.0	542.0	746.5	560.2
			20	1895.2	3235.4	**1876.8**	3271.6	4056.1
			50	4788.4	6661.4	**4491.9**	4785.3	8342.4
		50	5	1152.2	1725.1	**1150.5**	1772.4	1192.9
			20	**4312.1**	6523.3	4940.0	6190.5	6639.0
			50	10895.7	13129.8	**8992**	9662.8	17321.8
20	10	10	5	**112.8**	125.1	**112.8**	125.1	114.6
			20	361.0	378.0	**336.8**	337.1	383.6
			50	854.4	816.3	**726.8**	780.0	959.6
		30	5	524.8	584.6	**520.9**	582.5	538.1
			20	1814.3	1916.6	**1626.4**	1700.9	1892.3
			50	3848.6	3617.7	**3329.0**	3416.8	4339.4
		50	5	1100.4	1234.9	**1085.3**	1218.6	1123.3
			20	3696.4	3862.2	**3239.4**	3368.1	3673.7
			50	7581.6	6919.7	**6358.9**	6423.8	7896.8
	50	10	5	**116.7**	116.9	**116.7**	116.9	119.2
			20	321.5	723.1	**320.1**	737.8	840.3
			50	790.2	1231.8	**790.1**	1083.7	2431.7
		30	5	556.0	580.4	**555.6**	587.6	579.2
			20	1636.1	3233.5	**1626.3**	3347.2	4393.7
			50	**4206.9**	7403.5	4899.2	5115.9	9792.6
		50	5	**1161.5**	1319.6	1161.9	1347.1	1233.5
			20	**3703.3**	6734.6	3940.4	6662.5	7253.5
			50	9855.0	15818.5	**9531.5**	10100.2	22586.6
Average				2294.1	2720.6	**2062.7**	2385.3	3082.6

The solutions using the midpoint policy SLAP/Mp consistently outperform the others. They have the lowest average route length and the largest number of best average solution for combinations of instance attributes. The second best combination is SLAP/Re, which presents good performance, especially in warehouses with long aisles ($B = 50$). SLAP/OPP has some of the best solutions in smaller warehouses ($A = 5$ and $B = 10$, or 5×10), but significantly worse solutions in larger ones. This is due to the limitations in running time since it is the most time-consuming problem for the GVNS to solve. SLAP/Ss and SLAP/Lg did not have the best average solution in any case.

We also observe that solutions for instances with few picks and large warehouses (e.g., 20×50, $\mathcal{O} = 10$, $\mathcal{Q}_o = 5$) tend to be similar for no matter which routing policy is used. In these cases, it is common that the optimal route is the "go to the aisle, pick and return" special case since picks from a single order are concentrated close to each other in only one aisle across the front cross aisle.

The best situation to use SLAP/Ss is when several products have to be picked in each route ($\mathcal{Q}_o = 50$) in a small warehouse (5×10). In these cases, SLAP/Ss is outperformed only by SLAP/OPP. This is also the situation that SLAP/Re performs worst. In these instances, picks are more distributed along the storage area. Under return routing, there is a higher chance that the return is performed further inside the aisle, which increases the route length. In this case, an S-shape route is usually shorter, as shown by several studies [9, 27, 41].

In a special case with horizontal layouts, i.e., with a large number of short aisles, and for a large number of picks ($\mathcal{Q}_o = \{20, 50\}$), SLAP/Lg performs well, being outperformed only by SLAP/Mp. In vertical layouts, SLAP/Lg and SLAP/Ss solutions are considerably bad. When products to be picked are located in multiple aisles, these policies require a complete traversal of the long aisles, significantly increasing the route lengths. Meanwhile, in the SLAP/Re – and SLAP/Mp when all products are in the lower zone – products to be picked in a single route may be assigned to locations in multiple aisles across the front cross aisle, resulting in significantly shorter routes due to the return performed, even if multiple aisles are visited.

Finally, when comparing warehouses with a similar layout proportion (5×10 against 20×50) for the same number of picks, solutions are usually better in the smaller warehouse. This is because demands in our instances were generated considering that all storage locations contain a different product with the potential to be ordered. Thus, the smaller warehouse has a lesser variety of ordered products and, consequently, a higher average demand per product, meaning a higher concentration of picks near the I/O point.

6. Managerial insights

The results of the experiments show the outcomes of changing storage and routing policies in order to improve picking efficiency and which methods to use in different cases. We now discuss the practical implications of this research for warehouse managers.

Although it is interesting to see that mathematical models can be used to solve all considered versions of the SLAP, the experiments show that they are capable of solving only very small instances, which do not require sophisticated methods to be solved. We show that the time spent to solve the problem increases very fast when warehouses and orders increase in size. For the same instances, the GVNS finds better solutions in less than a second, as discussed in Section 5.2.1. For larger instances, the mathematical models cannot even provide feasible solutions to the problems. In all cases investigated, the heuristic approach is more effective.

When one changes from heuristic policies to optimal algorithms, it is expected to achieve better solutions. For the case considered – single-block layout with two-sided picking and orders known *a priori* – the results show that an optimal storage layout results in an average reduction of between 27% to 62% in total route length traveled by the pickers. Larger warehouses with fewer routes and smaller orders are especially fit for this change. This indicates that the common storage policies in use result in solutions very far from optimal, even when a good combination of storage and routing heuristic policies is adopted. In practice, managers should consider that the rearrangement of product locations has a cost that reduces the real savings. Warehouse reshuffling policies and costs are discussed in Pazour and Carlo [26]. SLAP solutions obtained by our GVNS may be used to compare with the reshuffling costs from the current storage layout to the optimal one, helping managers to decide the best time to rearrange it.

Solutions can be improved even further if we consider that the routing policies can be changed as well. The performances of the routing methods for an optimal storage policy are compared in Table 6. The results show a dominance of midpoint routing in most situations. Although it is expected that largest gap routing should dominate midpoint, since it is a less restricted version of it [9], it is the differences in the special cases that make SLAP/Lg solutions worse than SLAP/Mp. While the only special case for the former is when all products are in a single route, the latter also has the case where all products are below the midpoint. This allows products to be located in several aisles without the need to fully traverse any aisle to use the back cross aisle, which in the SLAP/Lg would lead to bad solutions as

described in Section 5.3.2. This is a rather different approach for the midpoint than in Dijkstra and Roodbergen [9] where they consider traversing even if no products are located above the midpoint.

All experiments performed here consider a tight run time limit. Since the warehouse rearrangement is a tactical level problem, it is common that managers have more time to make their decisions then the two-hour limit considered. For this limit, most experiments finished before the predetermined 10,000 cycles of our GVNS. This indicates that better solutions are still within reach if we could let our GVNS run for a longer time. Experiments for the SLAP/OPP running during eight hours improved solutions by 13% on average when compared to the two-hour limit, with higher improvements being found in larger instances. We should also expect improvements in the solutions for the SLAP with other routing policies, although in lower levels.

Finally, solutions found for the other SLAPs are upper bounds of the optimal solution for the SLAP/OPP, since a route found by any heuristic routing policy is also a feasible route for the OPP. In the cases that SLAP/OPP does not find the best results, warehouse managers may still opt for changing routing policies from a heuristic to the optimal one. Instead of using LKH during the search, managers can use another routing policy to speed up solutions convergence. Then, for the storage layout found, they can run LKH to search for optimal routes. The new solution should be at least as good as the one found using the heuristic routing policy.

7. Conclusions

In this paper we have introduced new optimization tools to solve storage location assignment problems. The SLAP consists in arranging the products in a warehouse in order to optimize material handling during the picking process. A SLAP solution is evaluated according to the performance of the order picking, measured as the distance traveled by pickers. When decisions regarding product assignment and routing are integrated, we have the storage location and order picking problem (SLAP/OPP) as introduced here. The SLAP/OPP is modeled as a cubic mixed integer programming problem. We also model four variants of the SLAP/OPP, i.e., when the routes are created using a heuristic routing policy, which are straightforward methods to create picking routes. Many of these policies are modeled mathematically for the first time in this paper.

Extensive computational experiments show that the models can deal with very limited size instances. As an alternative, we have proposed a General Variable Neighborhood Search (GVNS) metaheuristic

framework with a descent phase consisting of the exploration of three different neighborhoods adapted for the problem. The GVNS algorithm is capable of providing optimal solutions for all instances solved by the models within less than a second. We also show that, in instances of size comparable to those of real scenarios, the search for optimal storage location assignments leads to savings ranging from 27% to 62% compared to the solution using commonly used policies. Savings can be even higher when considering that routing policies can also be changed, for example, from a heuristic policy to an optimal one, and that GVNS has the potential to keep improving solutions the longer it runs. This demonstrates the potential benefits of searching for an optimal assignment of products to storage locations.

References

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, 2007.

[2] H. J. Carlo and G. E. Giraldo. Toward perpetually organized unit-load warehouses. *Computers & Industrial Engineering*, 63(4):1003–1012, 2012.

[3] T. Chabot, L. C. Coelho, J. Renaud, and J.-F. Côté. Mathematical model, heuristics and exact method for order picking in narrow aisles. *Journal of the Operational Research Society*, 69(8): 1242–1253, 2018.

[4] F. T. S. Chan and H. K. Chan. Improving the productivity of order picking of a manual-pick and multi-level rack distribution warehouse through the implementation of class-based storage. *Expert Systems with Applications*, 38(3):2686–2700, 2011.

[5] C.-M. Chen, Y. Gong, R. B. M. De Koster, and J. A. E. E. Van Nunen. A flexible evaluative framework for order picking systems. *Production and Operations Management*, 19(1):70–82, 2010.

[6] L. Chen, A. Langevin, and D. Riopel. A tabu search algorithm for the relocation problem in a warehousing system. *International Journal of Production Economics*, 129(1):147–156, 2011.

[7] R. B. M. De Koster, T. Le-Duc, and K. J. Roodbergen. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501, 2007.

[8] R. Dekker, R. B. M. De Koster, K. J. Roodbergen, and H. Van Kalleveen. Improving order-picking response time at Ankor's warehouse. *Interfaces*, 34(4):303–313, 2004.

[9] A. S. Dijkstra and K. J. Roodbergen. Exact route-length formulas and a storage location assignment heuristic for picker-to-parts warehouses. *Transportation Research Part E: Logistics and Transportation Review*, 102:38–59, 2017.

[10] Z. Drezner. Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Computers & Operations Research*, 35(3):717–736, 2008.

[11] R. J. Forrester. Tightening concise linear reformulations of 0-1 cubic programs. *Optimization*, 65 (4):877–903, 2016.

[12] E. Frazelle. *World-Class Warehousing and Material Handling*. McGraw-Hill, New York, 2nd edition, 2016.

[13] F. Glover and E. Woolsey. Converting the 0–1 polynomial programming problem to a 0–1 linear program. *Operations Research*, 22(1):180–182, 1974.

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

[15] J. Gu, M. Goetschalckx, and L. F. McGinnis. Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operational Research*, 203(3):539–549, 2010.

[16] P. Hansen, N. Mladenović, J. Brimberg, and J. A. M. Pérez. Variable neighborhood search. In *Handbook of Metaheuristics*, pages 57–97. Springer, New York, 2019.

[17] W. H. Hausman, L. B. Schwarz, and S. C. Graves. Optimal storage assignment in automatic warehousing systems. *Management Science*, 22(6):629–638, 1976.

[18] K. Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[19] L.-F. Hsieh and Y.-C. Huang. New batch construction heuristics to optimise the performance of order picking systems. *International Journal of Production Economics*, 131(2):618–630, 2011.

[20] L.-F. Hsieh and L. Tsai. The optimum design of a warehouse system on order picking efficiency. *The International Journal of Advanced Manufacturing Technology*, 28(5-6):626–637, 2006.

[21] M. Kofler, A. Beham, S. Wagner, and M. Affenzeller. Affinity based slotting in warehouses with dynamic order patterns. In *Advanced Methods and Applications in Computational Intelligence*, pages 123–143. Springer, 2014.

[22] R. J. Mantel, P. C. Schuur, and S. S. Heragu. Order oriented slotting: A new assignment strategy for warehouses. *European Journal of Industrial Engineering*, 1(3):301–316, 2007.

[23] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[24] J. C.-H. Pan and M.-H. Wu. Throughput analysis for order picking system with multiple pickers and aisle congestion considerations. *Computers & Operations Research*, 39(7):1661–1672, 2012.

[25] L. Pansart, N. Catusse, and H. Cambazard. Exact algorithms for the order picking problem. *Computers & Operations Research*, 100:117–127, 2018.

[26] J. A Pazour and H. J. Carlo. Warehouse reshuffling: Insights and optimization. *Transportation Research Part E: Logistics and Transportation Review*, 73:207–226, 2015.

[27] C. G. Petersen. An evaluation of order picking routeing policies. *International Journal of Operations & Production Management*, 17(11):1098–1111, 1997.

[28] C. G. Petersen. The impact of routing and storage policies on warehouse efficiency. *International Journal of Operations & Production Management*, 19(10):1053–1064, 1999.

[29] C. G. Petersen and G. R. Aase. A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics*, 92(1):11–19, 2004.

[30] C. G. Petersen, G. R. Aase, and D. R. Heiser. Improving order-picking performance through the implementation of class-based storage. *International Journal of Physical Distribution & Logistics Management*, 34(7):534–544, 2004.

[31] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Science & Business Media, New York, 2012.

[32] S. Quader and K. K. Castillo-Villar. Design of an enhanced multi-aisle order-picking system considering storage assignments and routing heuristics. *Robotics and Computer-Integrated Manufacturing*, 50:13–29, 2018.

[33] H. D. Ratliff and A. S. Rosenthal. Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem. *Operations Research*, 31(3):507–521, 1983.

[34] J. Reyes, E. Solano-Charris, and J. Montoya-Torres. The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 10(2):199–224, 2019.

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

[36] K. J. Roodbergen, I. F. A. Vis, and G. D. Taylor Jr. Simultaneous determination of warehouse layout and control policies. *International Journal of Production Research*, 53(11):3306–3326, 2015.

[37] A. Scholz and G. Wäscher. Order batching and picker routing in manual order picking systems: The benefits of integrated routing. *Central European Journal of Operations Research*, 25(2): 491–520, 2017.

[38] A. Scholz, S. Henn, M. Stuhlmann, and G. Wäscher. A new mathematical programming formulation for the single-picker routing problem. *European Journal of Operational Research*, 253(1): 68–84, 2016.

[39] E. Tappia, D. Roy, M. Melacini, and R. B. M. De Koster. Integrated storage-order picking systems: Technology, performance models, and design insights. *European Journal of Operational Research*, 274(3):947–965, 2019.

[40] C. Theys, O. Bräysy, W. Dullaert, and B. Raa. Using a TSP heuristic for routing order pickers in warehouses. *European Journal of Operational Research*, 200(3):755–763, 2010.

[41] T. Van Gils, K. Ramaekers, K. Braekers, B. Depaire, and A. Caris. Increasing order picking efficiency by integrating storage, batching, zone picking, and routing policy decisions. *International Journal of Production Economics*, 197:243–261, 2018.

[42] T. Van Gils, K. Ramaekers, A. Caris, and R. B. M. De Koster. Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, 267(1):1–15, 2018.

[43] F. Weidinger. Picker routing in rectangular mixed shelves warehouses. *Computers & Operations Research*, 95:139–150, 2018.

[44] F. Yener and H. R. Yazgan. Optimal warehouse design: Literature review and case study application. *Computers & Industrial Engineering*, 129:1–13, 2019.