DEPARTMENT OF ENGINEERING MANAGEMENT

Scattered storage assignment: mathematical model and valid inequalities to optimize the intra-order item distances

Harol Mauricio Gámez Albán, Trijntje Cornelissens & Kenneth Sörensen

DEPARTMENT OF ENGINEERING MANAGEMENT

Scattered storage assignment: mathematical model and valid inequalities to optimize the intra-order item distances

Harol Mauricio Gámez Albán, Trijntje Cornelissens & Kenneth Sörensen

RESEARCH PAPER 2020-008
NOVEMBER 2020

D/2020/1169/008

Scattered storage assignment: mathematical model and valid inequalities to optimize the intra-order item distances

Harol Mauricio Gámez Albán*, Trijntje Cornelissens, Kenneth Sörensen

ANT/OR – Operations Research Group, Department of Engineering Management, University of Antwerp, Prinsstraat 13, 2000 Antwerp, Belgium

Abstract

Under a scattered storage policy, a single stock-keeping unit is stored in different locations throughout the warehouse. This policy aims to reduce travel times during order picking since it increases the probability of finding items belonging to the same order in nearby locations. Such type of storage policy is adequate for a typical e-commerce environment where a large variety of products are ordered in small quantities. The assignment of single items to different storage locations in the warehouse results in the scattered storage assignment problem. This paper proposes an exact algorithm for this problem that minimizes the sum of pairwise distances between the locations of the items belonging to the same order. We develop a mixed-integer model with different sets of valid inequalities (cuts) and test their performance on a set of data instances. Computational results show that the valid inequalities improve our model's objective function value by 32% compared to the initial formulation. Finally, we prove that the pairwise distance between items of the same order and the picker routing distances are better for our storage allocation policy than a more traditional storage allocation and random storage policy.

Keywords: Logistics, scattered storage, e-commerce, mathematical optimization, valid inequalities

*Corresponding author

Email addresses: `mauricio.gamezalban@uantwerpen.be` (Harol Mauricio Gámez Albán), `trijntje.cornelissens@uantwerpen.be` (Trijntje Cornelissens), `kenneth.sorensen@uantwerpen.be` (Kenneth Sörensen)

1. Introduction

E-commerce has been growing considerably in recent years. By 2021 online sales will have grown more than 40% compared to projected sales for 2019 (Beeketing (2019)). This forces companies to be attentive to the needs of this e-commerce market and to synchronize and optimize their logistics processes. Some of the most important challenges are the small order quantities, large assortment, scalability, and tight delivery schedule that characterize the e-commerce supply chain (Weidinger (2018b)). Consequently, the challenges in the warehouse management processes (receiving, storage, order picking, and shipping) are increasingly relevant. According to Bartholdi III and Hackman (2019), order picking is responsible for about 55% of warehouse operating costs.

Because order picking is the most expensive process in a warehouse, the storage allocation process must be considered (Bartholdi III and Hackman (2019)). Indeed, one of the most critical warehouse management activities is defining the products' positions (slotting). Historically, slotting aims to maximize the use of available space within a warehouse while reducing handling costs (any expenditure of moving the goods into or out of the warehouse) (Bureau (2011)). However, due to the rise of e-commerce, it is expected that new storage strategies may be more useful than conventional methods. To face these new challenges, the concept of scattered storage appears as a possible alternative solution. Scattered storage consists of unbundling unit loads in the receiving area and storing single stock-keeping units (SKU) at different positions within the warehouse to increase the probability of finding an item close by during order picking (Weidinger (2018b)).

The optimization problem that aims to assign a single item type to different storage locations in the warehouse is called the scattered storage assignment problem (SSA). The first formulation for the SSA is introduced by Weidinger and Boysen (2018) to reduce walking time. The authors propose a mathematical formulation that defines the positions of the products (single items) around the warehouse, such that the maximum distances of SKUs to some measuring point are minimized. The concept of measuring point (introduced by the authors) is a predetermined location in the warehouse, from which the distances to the closest item of each SKU are determined. They prove that their SSA algorithm outperforms random storage assignments (which are the most typical assignments in real-world situations) in terms of the picking effort.

Although there are not many references that study the concept of SSA itself, some study a similar concept, e.g., random storage, where the products are randomly allocated in the warehouse. In this case, the products can be scattered throughout the warehouse. Scholz et al. (2016) emphasize that storage allocation methods have a significant impact on the tour lengths during order picking. Theys et al. (2010) and Pansart et al. (2018) compare their picker-routing algorithms in two types of storage policies, "random" and "volume-based". Under a volume-based policy, twenty percent of the most demanded items are located near the first cross aisle. Both articles conclude that the volume storage policy improves the picking walking time. Concerning SSA, Weidinger (2018a) shows that scattered storage helps minimize travel distance when the pick list is more heterogeneous, and the quantities requested by each item are not too high. Weidinger et al. (2019) analyze how much the size of orders affects the travel times of the pickers in a scattered storage policy. The authors identify the turning points (percentage of items demanded by online orders) where scattered storage may be more convenient over dedicated storage or vice versa.

Notwithstanding the papers mentioned above, there is still a lack of research on the SSA problem. In their survey, Boysen et al. (2019) address the issue of mixed-shelves (different types of items are stored in the same position) as a scattered storage strategy. They find that scientific decision support for mixed-shelves warehouses, e.g., on how to position mixed-shelves, is non-existent. Van Gils et al. (2018), on the other hand, present an exhaustive literature review on the integration of the different processes in a warehouse. They identify that the interrelationships to define storage location assignments, batching, and routing policies have not received sufficient research attention.

This paper aims to introduce a new scattered storage assignment policy that seeks to minimize the sum of the pairwise distances (SPD) between the positions of items belonging to the same order. We will refer to this problem as the scattered storage assignment problem with pairwise distances (SSA-SPD).

The contributions of this paper are the following:

1. the SSA policy based on the SPD is introduced;
2. a mathematical programming formulation for the SSA-SPD problem is

developed;

3. different classes of valid inequalities are introduced to strengthen the initial formulation;
4. benchmark instances are generated to test the solution method's performance with different combinations of valid inequality classes;
5. the SSA-SPD policy is compared to a standard storage policy in which products of the same type are stored in close positions.

This paper is organized as follows: In Section 2, we describe the SSA-SPD problem and introduce the initial mathematical formulation. In Section 3, we present our valid inequalities. In Section 4, we describe the instances and compare the mixed integer linear programming solver results for the different models. Section 5 concludes the paper.

2. Problem description

2.1. Problem definition

In this paper, the SSA-SPD is defined as follows: Given a set of available storage positions and a set of orders containing multiple products in different quantities, what should be the storage location for each product of each order to minimize the SPD between the products of the same order? Additionally, the SSA-SPD takes into account the following characteristics:

- Only one type of product can be assigned to each storage position.
- Every single storage position has a predefined capacity for each type of product.
- The shortest pairwise distances between positions are known.
- For each product, the quantity requested by a single order must be retrieved from a single position.

Each item of each order has to be assigned to a position in the warehouse, taking into account the constraints mentioned above. We assume that the orders taken into account in the SSA-SPD problem are created based on historical analogy, and represent combinations of items (product type and quantity) frequently ordered. The SPD is expected to be larger than the walking picker distance since SPD compares the distances between all possible connections between positions of the same order. In contrast, in the walking picker distance, each position is only compared with one position instead of all positions of the same order. We can say the SPD is only an approximation of the pickers' actual walking distance.

2.2. Initial formulation

Let K be a set of orders, where $K = \{1, \dots, n\}$. The set I refers to the type of products, where $I = \{1, \dots, m\}$. Finally, let S be the set of storage positions, where $S = \{1, \dots, p\}$.

Three parameter sets are included in this formulation. The first one, u_{ik}, defines the quantity requested by order k for product i. The second is Q_{is}, which defines the capacity of position s when it is used to store product i. The third one is γ_{sr}, which represents the distance between storage positions s and r.

Decision variable x_{is} is equal to 1 if product i is stored in position s, and 0 otherwise. Variable y_{isk} is equal to 1, if an item i is assigned to storage position s to meet the demand of order k, and 0 otherwise. Finally, to identify if the storage positions s and r are assigned to order k, the binary variable v_{srk} is equal to 1 if both storage positions belong to the same order, and 0 otherwise. Using these definitions, the SSA-SPD problem can be defined as follows:

$$z = \quad \min \sum_{(s,r)\in S} \sum_{k\in K} v_{srk}\gamma_{sr} \tag{1}$$

$$\sum_{i\in I} x_{is} \leq 1 \qquad \forall s \in S \tag{2}$$

$$\sum_{s\in S} y_{isk} = 1 \qquad \forall i \in I, k \in K, u_{ik} > 0 \tag{3}$$

$$\sum_{k\in K} y_{isk}u_{ik} \leq x_{is}Q_{is} \qquad \forall i \in I, s \in S \tag{4}$$

$$\sum_{i\in I}(y_{isk} + y_{irk}) - 1 \leq v_{srk} \qquad \forall (s,r) \in S | s > r, k \in K \tag{5}$$

$$y_{isk}, x_{is}, v_{srk} \in \{0,1\} \qquad \forall i \in I, (s,r) \in S, k \in K \tag{6}$$

The objective function (1) minimizes the SPD. Constraints (2) allow storing only one type of product in each position. Restrictions (3) guarantee that all products demanded by each order are stored in a single position. Constraints (4) guarantee that the storage capacity of every position is never exceeded. Constraints (5) define whether the pair of positions s and r is used by order k. Finally, binary variables are defined in constraints (6).

3. Valid inequalities

In this Section, two groups of valid inequalities are proposed to enhance the previous formulation when relaxing some of the binary variables. The first group of valid inequalities concerns the possible values that each binary variable can take. The second group of inequalities is derived from the lifted cover inequalities developed for different 0-1 linear programs (Balas (1975)).

In addition to the inclusion of valid inequalities, the variables v_{srk} and x_{is} become unrestricted to take any value between 0 and 1. The fact of keeping the variable y_{isk} as an integer, guarantees that the variable v_{srk} is integer too, due to the restriction (5). However, variable x_{is}, although also restricted by the variable y_{isk} in the restriction (4), could still take non-integer values.

In general, this method is quite useful since it is not necessary to solve the integer problem, but instead, by relaxing certain variables, solutions can be reached more efficiently. The complication of this method is that non-feasible solutions can be chosen, so it is necessary to add some valid inequalities that facilitate the elimination of non-integer solutions. The sets of valid inequalities are meant to face this problem.

3.1. Initial valid inequalities

A valid inequality for a mixed-integer program is any constraint that tries to remove parts of the LP feasible region but does not eliminate any feasible integer solutions. It is also called a cutting plane or cut.

In this section, we construct the valid inequalities that represent the characteristics of complete graphs. Indeed each order can be represented by a complete graph, in which each node symbolizes the position of the items requested.

Complete graphs have some properties that can be used to create a set of valid inequalities for the SSA-SPD problem. For example, in each graph, the total number of edges (E) can be computed using the following formula: $\frac{n(n-1)}{2} = E$, where n is the total number of nodes (positions) in each graph (order). Considering the previous property and the number of positions that each order needs (which is known), we can define four parameter sets that can be used to create new valid inequalities.

We define parameter mif_k to be the total number of edges (E) in each order k. Parameter min_k represents the number of product types that an order k requests. Parameter mv_i is defined as the number of orders requesting product type i. Parameter $mvis_i$ is defined as the minimum number of positions that the product type i requires.

Using the parameters defined above, the next constraints are valid for the initial formulation:

$$\sum_{i \in I} \sum_{s \in S} y_{isk} = min_k \qquad \forall k \in K \tag{7}$$

$$\sum_{s \in S} \sum_{k \in K} y_{isk} = mv_i \qquad \forall i \in I \tag{8}$$

$$\sum_{s \in S} x_{is} \geq mvis_i \qquad \forall i \in I \tag{9}$$

$$\sum_{s \in S} \sum_{r \in S} v_{srk} = \frac{(min_k - 1)}{2} \sum_{i \in I} \sum_{s \in S} y_{isk} \qquad \forall k \in K \tag{10}$$

$$\sum_{r \in S | s \neq r} (v_{srk} + v_{rsk}) = (min_k - 1) \sum_{i \in I} y_{isk} \qquad \forall s \in S, k \in K \tag{11}$$

$$\sum_{s \in S} \sum_{r \in S} v_{srk} = mif_k \qquad \forall k \in K \tag{12}$$

$$y_{isk} \leq x_{is} \qquad \forall i \in I, s \in S, k \in K \tag{13}$$

Restrictions (7) establish a minimum number of positions for each order. Constraints (8) and (9) force to strictly comply with the number of positions required for each type of product. Constraints (10)-(11) relate variables x_{is} and v_{srk}, and force them to take specific values according to the number of type of products requested by each order. Constraints (12) result from the graph-theoretical property mentioned above. Finally, constraints (13) impose on the variable x_{is} to take integer values. This constraint is important since x_{is} is relaxed and has the freedom to take fractional values but will not do so since y_{isk} is an integer variable. In the end, when solving the initial formulation with this type of inequalities, restrictions (1)-(5) and (7)-(13) are applied together.

3.2. Knapsack cover inequalities

Our second group of inequalities derives from the knapsack problem. A set of items, each with a weight and a profit, must be selected to maximize the total benefit of the chosen items while respecting the knapsack's total capacity. The SSA-SPD has a similar structure to the knapsack problem since the product type, and associated order quantities must be assigned to each position while respecting its capacity.

The set of valid inequalities proposed in this section are related to the new lifting procedure developed by Letchford and Souli (2019b) for the knapsack problem. In the SSA-SPD, we generate cover constraints for each pair location-product (s, i) to check whether product i can be stored in slot s. These cuts aim to define at least one facet of the polytope generated by this problem's knapsack restrictions.

In the SSA-SPD, we define the concept of minimal cover C_{is} as the set of orders k that can store their product i in position s without exceeding its capacity. For example, suppose orders 1, 2, 3, and 4 demand product i quantities 2, 1, 2, and 3, and the capacity of position s for product i is 4. According to this information, a maximum of two orders can be stored in that position, which implies that $|C_{is}| = 2$. In short, the most robust cover inequalities are obtained when C is minimal (see Letchford and Souli (2019a)).

Let C_{is} be a minimal cover if product i is stored in position s. The valid knapsack inequality that can be applied is the following:

$$\sum_{k \in C_{is}} y_{isk} \leq |C_{is}| \quad \forall i \in I, s \in S \tag{14}$$

For $w = \{1, \ldots, |C_{is}|\}$, let $G(w)$ be the sum of the w smallest u_{ik} values over the members of C_{is}, then the next constraint is also valid for the SSA-SPD:

$$\sum_{k \in K \setminus C_{is}} \alpha_{isk} y_{isk} + \sum_{k \in C_{is}} y_{isk} \leq |C_{is}| \quad \forall i \in I, s \in S \tag{15}$$

Where α_{isk} is a parameter calculated as follows: if $u_{ik} + G(w) > Q_{is} \quad \forall i \in I, k \in K \setminus C_{is}$ then, $\alpha_{ik} = |C_{is}| - w$. On the other hand, if $u_{ik} > Q_{is} \quad \forall i \in I, k \in K \setminus C_{is}$ then, $\alpha_{ik} = 0$.

This second set of valid inequalities works quite well for the SSA-SPD. This performance is because the products that are outside the minimal cover can be affected by a much larger parameter, making the selection space for the y_{isk} variable more bounded.

On the other hand, if for a certain pair of different orders k and j, position s, and product i the following codifications are met: $\alpha_{isk} + \alpha_{isj} \leq |C_{is}|$ and

$u_{ik} + u_{ij} > Q_{is}$, then increase by one both parameters α_{isk} and α_{isj}, and the minimal cover $|C_{is}|$. This resulting inequality is also valid for the SSA-SPD, and it is at least as strong as (15):

$$\sum_{k \in K \setminus C_{is}} \lambda_{isk} y_{isk} + \sum_{k \in C_{is}} y_{isk} \leq |C_{is}| + 1 \quad \forall i \in I, s \in S \tag{16}$$

where $\lambda_{isk} = \alpha_{isk} + 1$ for all those products, orders and positions that met the previous condition. When applying these types of inequalities, restrictions [1-5] and [15-16] are applied together.

4. Numerical experiments

In this section, we describe the numerical experiments. Section 4.1 describes the instance generation, while Section 4.2 is dedicated to the main results obtained by applying the different set of cuts developed in Section 3.

4.1. Instances generation

Unfortunately, no established testbed for the SSA-SPD exists. This Section explains how our test instances have been generated. The process to create the set of instances is as follows: The products that each order will request are selected with a probability of 50%. The quantity demanded per product is discrete uniformly distributed over [0,5], where 0 means that the order does not require the item i. The storage capacity of each position for the different types of products is generated discrete uniformly over [0,10], where 0 means that the position cannot store the item.

After generating all the orders, it is verified which orders request more than 60% of the product types. If this is the case, the products with the highest demand are eliminated until no order contains more than 60% of the types of the product. For example, suppose there is a portfolio of ten products. For a specific order, the product quantities are 2, 1, 2, 3, 0, 7, 5, 6, 0, and 4, meaning 80% of the types of products are requested. In this case, the types of products numbers 6 and 8 are eliminated for that specific order. Due to the complexity of the SSA-SPD and the difficulty of achieving the optimal solution, only small data sets are included in the experiments. Table 1 shows the parameters used for the 20 instances tested in this paper.

The experiments executed in this article are carried out, taking into account a traditional parallel aisles warehouse layout (Figure 1). This type of layout has the following characteristics:

Instance	# Positions	# Orders	# Type of products	Total demand
1				56
2				140
3	20	10	10	130
4				125
5				143
6				110
7				163
8				130
9	20	15	10	96
10				155
11				157
12				154
13				176
14				203
15	30	15	10	161
16				228
17				242
18				183
19	80	25	15	392
20				455

Table 1: Instances description

- Picking aisles are straight and parallel to each other.
- If present, cross aisles are straight and meet the picking aisles at right angles.
- Positions are unique in terms of geographic location, which implies no positions on top of each other.

In total, we generated 20 instances, which vary between 20, 30, and 80 positions; 10, 15, and 25 orders, and finally, between 10 and 15 types of products.

4.2. Results

The experiments described in this section aim to test the impact of the valid inequalities on the instances solved.

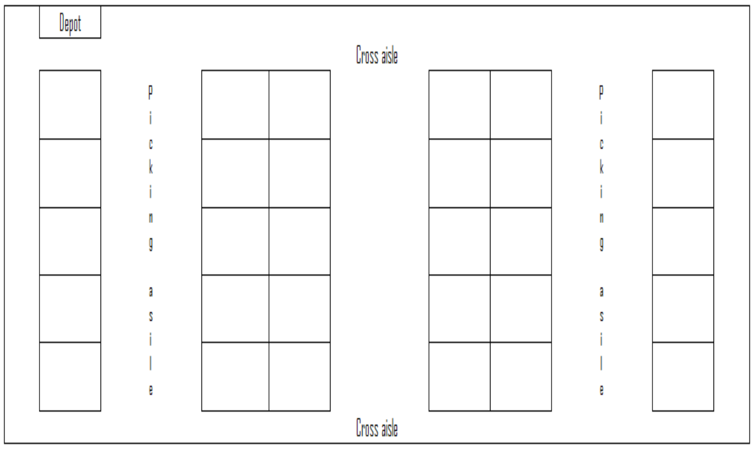

Figure 1: Warehouse layout

First, we use the solver Cplex in its standard configuration on the initial model without valid inequalities. Then, we solve for the same instances, the other models with different combinations of valid inequalities. We evaluated the following models:

- Model I: Initial formulation (constraints: (1)-(6))
- Model II: Initial formulation including only the first set of valid inequalities (constraints: (1)-(13))
- Model III: Initial formulation including the second set of valid inequalities (constraints: (1)-(6), and (15)-(16))
- Model IV: Initial formulation including both sets of valid inequalities (constraints: (1)-(13), and (15)-(16))
- Model V: Initial formulation, relaxing the variables x_{is} and v_{srk}, and including the first set of valid inequalities (constraints: (1)-(5), and (7)-(13))
- Model VI: Initial formulation, relaxing the variables x_{is} and v_{srk}, and including the second set of valid inequalities (constraints: (1)-(5), and (15)-(16))

- Model VII: Initial formulation, relaxing the variables x_{is} and v_{srk}, and including both sets of valid inequalities (constraints: (1)-(5), (7)-(**??**), and (15)-(16))

The experiments have been performed on an Intel(R) Core(TM) i7-8850H, 2.60 GHz, 16.00 GB of RAM. The mathematical formulation has been coded in C++ using Cplex 12.9 with the default settings, and maximum computing time of one hour.

4.2.1. Comparison of valid inequalities

Instance	I	II	III	IV	V	VI*	VII
1	37	37	37	37	37	*37*	37
2	311	303	294	295	245	*179*	241
3	212	235	274	210	177	*144*	181
4	163	176	182	151	159	*143*	157
5	298	243	297	243	237	*154*	237
6	113	93	104	93	93	*88*	93
7	315	279	N/sln	271	271	*251*	271
8	201	146	181	160	147	*96*	123
9	87	90	86	90	72	*66*	70
10	313	288	274	350	290	*242*	271
11	230	227	230	227	227	*141*	227
12	N/sln	313	412	309	303	*189*	315
13	334	365	398	352	274	*236*	256
14	756	569	647	554	428	*315*	451
15	415	303	239	273	219	*174*	203
16	1072	723	1001	1041	679	*517*	496
17	N/sln	996	N/sln	N/sln	893	*777*	888
18	253	249	280	262	191	*177*	183
19	4427	N/sln	4132	5816	5863	*1690*	5471
20	3279	N/sln	3842	N/sln	N/sln	*1302*	4153
Avg.	318	270	302	289	232	*181*	215
Imprv(%).	-	15.1	5.0	9.1	27.0	43.1	32.4

Table 2: Sum of the pairwise intra-order item distances obtained after calculation time (< 1 hour)

Table 2 shows the initial results for each instance in terms of the SPD. The last row of the table is the SPD average of the solutions obtained using each model. This value is calculated, taking into account only instances that have feasible solutions for all the models. So the results of instances 7, 12, 17, 19, and 20 are not considered in the average because not all models resulted in a feasible solution for those instances. Model VI does not yield any feasible solutions since the results of the variables are fractional. The reason is that only the cuts related to the second group of valid inequalities do not force the variables x_{is} to strictly take integer values, as it happens when using the first group of valid inequalities.

Model VII is most effective since it obtained an average value of 215, a smaller value than the other models. Model V gets the second-best result with an average value of 232. The results for models II and IV (without relaxing variables) show the first group of valid inequalities' effectiveness. These valid inequalities help to reduce the polyhedral structure of the initial formulation and, therefore, are much more effective in finding good solutions. However, by including the valid inequalities in models II, III, and IV, the results improve considerably compared to the initial formulation.

Moreover, the combination of both sets of valid inequalities works much better when the variables are relaxed. When the integer variables are not relaxed, the combination of both sets of inequalities is not as effective in comparison with only using a single set of valid inequalities, as seen in the average solutions obtained by models II and IV. The opposite happens when the variables are relaxed (models V and VII). In this case, the combination of both sets of inequalities does help to improve the results.

We notice that in the case of the largest instances (19 and 20), some models struggled to find feasible solutions. Models I-IV failed in 2 out of 20 instances in obtaining at least one feasible solution, while model V failed in only one instance, and model VII solved all instances. We also see that the results obtained by the models that include relaxation of variables are better compared to those that do not include it. These results imply that relaxing variables and then forcing their values to binary (0-1) with valid inequalities is a useful strategy to improve the tested instances. Finally, the last row of the table shows that all models (except VI, not feasible) improved the initial formulation results between 5% and 32%.

Table 3 shows the optimality gap reported by Cplex, which is the difference between the solution reached after the maximum computation time and the linear relaxation. Models V and VII had the lowest average optimality

Instance	I	II	III	IV	V	VI*	VII
1	56.76	0	59.46	0	0	48.65	0
2	68.72	40.90	72.01	39.53	29.16	45.48	24.42
3	71.44	39.60	73.33	31.75	21.41	51.51	22.55
4	48.08	19.98	51.77	5.40	13.78	36.55	10.83
5	67.50	13.20	68.58	12.10	0	53.37	0
6	63.98	0	62.50	0	0	43.18	0
7	93.75	9.93	N/sln	9.97	7.09	97.56	6.65
8	98.51	35.96	99.45	41.27	39.53	88.02	27.28
9	95.40	29.53	97.09	29.91	23.82	79.55	9.75
10	73.63	45.11	70.57	55.81	46.30	95.46	41.20
11	59.77	0	41.47	0	0	93.74	1.81
12	N/sln	28.08	81.92	26.08	21.08	62.30	30.72
13	96.86	42.55	100	40.26	27.94	96.61	23.18
14	99.17	55.72	100	54.51	44.64	96.30	47.36
15	98.19	44.15	100	38.24	29.61	97.70	24.09
16	99.30	62.44	100	73.89	62.33	99.23	48.01
17	N/sln	69.95	N/sln	N/sln	67.97	98.46	67.74
18	97.23	37.39	98.39	40.61	25.34	95.90	22.33
19	100	N/sln	100	85.14	84.80	100	90.55
20	100	N/sln	100	N/sln	N/sln	100	90.13
Avg.	79.64	31.10	79.64	30.89	24.26	74.75	20.18

Table 3: Optimality Gap (%)

gap. Both models managed to obtain better lower bounds and better results in terms of the final solution. In the case of models II and IV, the latter obtained better lower bounds than model II, despite not having achieved the best results in terms of the final solution. Finally, the optimality gap does not drop below 20.18%, which implies that trying to reach the optimum in most instances is challenging to solve. The models V and VII were the only ones that managed to solve 4 out of 20 instances to optimality, while models II and IV did it in only three instances.

4.2.2. SPD comparison with traditional and random storage allocation

To analyze the SPD savings results, we will compare the SSA-SPD (results of model VII) policy with a traditional and random storage policy. The traditional storage allocation (TSA) assigns products of the same type to

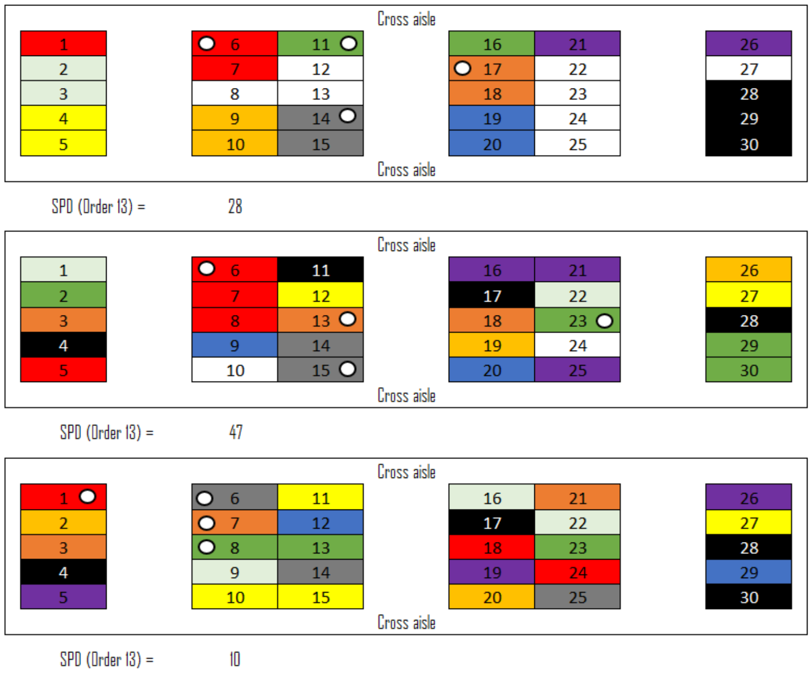

Figure 2: Example of SPD for Order 13 for traditional storage (upper part) and scattered storage (lower part)

warehouse positions close to each other. A procedure in two stages is performed to face this policy. In the first stage, an optimization model is used to minimize the SPD between items of the same type of product. The structure of this optimization model is similar to the original model proposed in this paper (constraints: (1)-(6)), but instead of variable v_{srk} (which is 1 if order k uses positions s and r), we define v_{sri} as 1 if type of product i uses positions s and r. Another modification is constraint (5). In this case, we use the following constraint:

$$x_{is} + x_{ir} - 1 \leq v_{isr} \quad \forall (s,r) \in S | s \neq r, i \in I \tag{17}$$

This model returns the values for the variable x_{is} that represent the positions for all product types. In the second stage, our SSA-SPD model is solved (constraints: [1-6]), taking into account the fixed values of variable x_{is} defined in the first stage.

In the random storage allocation (RSA) policy, the items are assigned randomly through the warehouse. Once items have been assigned and we already know how many items are in each position (like the first phase of TSA), we perform the model using again (constraints: [1-6]), taking into account the fixed values of variable x_{is} defined in the random stage.

Figure 2 shows an example for Order 13 of instance number 15. The upper part of the figure shows the TSA, where the same products are stored together. The middle part shows RSA where the items are randomly allocated, while in the lower part of the figure, the items are assigned via our SSA-SPD strategy. Each position's color represents the type of product, while the numbers are the positions' labels. Suppose that Order 13 needs the following products: one red, one green, one orange, and one gray. If the TSA is used, the products located in positions 6, 11, 14, and 17 would be required. In that hypothetical scenario, the SPD for Order 13 would be 28. If RSA had been the storage allocation policy, the products located in positions 6, 13, 15, and 23 would be required. In that case, the SPD would be 47. Finally, if SSA-SPD would be the situation, the products located in positions 1, 6, 7, and 8 would be required. The SPD, in this case, would be 10. In this sense, the savings in terms of the SPD are quite significant.

Table 4 shows the total SPD of the different instances for the SSA-SPD, TSA, and RSA policy. The last two columns represent the SPD savings in percentage between (SSA-SPD and TSA), and (SSA-SPD, and RSA). We can see that the average difference in total SPD is remarkable and exceeds 40% in both cases. We can also notice that RSA is sightly better than TSA in terms of SPD. Although RSA items are randomly allocated, there are always a few probabilities of finding an item close when orders require different types of items.

5. Conclusions

This paper has investigated a scattered storage assignment strategy that spreads a single SKU all around the warehouse. The scatteredness of items aims to reduce unproductive picker travel because, for any order, there is probably some item close by at the time of picking. Our approach seeks to minimize the SPD of the expected orders, meaning the total sum of the distance between the locations of the items of each order. We have presented a mathematical formulation of the problem together with different classes of valid inequalities.

Instance	SSA-SPD	TSA	RSA	S-TSA (%)	S-RSA (%)
1	37	185	144	80.0	74.3
2	245	631	520	61.2	52.9
3	185	515	443	64.1	58.2
4	157	447	398	64.9	60.6
5	237	463	348	48.8	31.9
6	93	345	268	73.0	65.3
7	271	431	337	37.1	19.6
8	123	352	312	65.1	60.6
9	75	267	181	71.9	58.6
10	291	439	331	33.7	12.1
11	227	286	272	20.6	16.5
12	318	498	473	36.1	32.8
13	288	1028	721	72.0	60.1
14	496	1123	733	55.8	32.3
15	240	868	649	72.4	63.0
16	583	1294	883	54.9	34.0
17	888	1366	989	35.0	10.2
18	183	758	601	75.9	69.6
19	3902	6118	5051	4.0	22.7
20	3279	4251	3720	22.9	11.9
Avg.	606	1083	869	52.5	42.4

Table 4: Total SPD savings between TSA, RSA and SSA-SPD

We studied the effectiveness of those valid inequalities by performing tests with different problem formulations. Computational results show that valid inequalities improve the performance of the solution method. However, the most significant effect is found when both sets of valid inequalities are included in the relaxed model, improving the objective by 32% compared to the initial formulation. We also examined the benefit of scattered storage over traditional and random storage policies. The advantage of applying SSA-SPD results in a reduction of 44% in the total SPD compared to the TSA and 30% compared to the RSA policy.

Lastly, there remains the challenge of deriving scattered storage assignments in massive warehouses where vast assortments of SKUs are stored. For these cases, further algorithmic developments (heuristics or meta-heuristics) are necessary to successfully face the assignment of thousands of SKUs in multiple storage positions.

References

Balas E. Facets of the knapsack polytope. Mathematical programming 1975;8(1):146–64.

Bartholdi III JJ, Hackman ST. WAREHOUSE & DISTRIBUTION SCIENCE Release 0.98.1, 2019.

Beeketing . Future of Ecommerce 10 international growth trends. `https://beeketing.com/blog/future-ecommerce-2019/`; 2019. Accessed: 2020-04-08.

Boysen N, de Koster R, Weidinger F. Warehousing in the e-commerce era: A survey. European Journal of Operational Research 2019;277(2):396–411.

Bureau L. What is slotting and why is it an important productivity tool? `https://logisticsbureau.com/what-is-slotting-and-why-is-it-an-important-productivity-tool/`; 2011. Accessed: 2019-10-18.

Letchford AN, Souli G. New valid inequalities for the fixed-charge and single-node flow polytopes. Operations Research Letters 2019a;47(5):353–7.

Letchford AN, Souli G. On lifted cover inequalities: A new lifting procedure with unusual properties. Operations Research Letters 2019b;47(2):83–7.

Pansart L, Catusse N, Cambazard H. Exact algorithms for the order picking problem. Computers & Operations Research 2018;100:117–27.

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

Theys C, Bräysy O, Dullaert W, Raa B. Using a TSP heuristic for routing order pickers in warehouses. European Journal of Operational Research 2010;200(3):755–63.

Van Gils T, Ramaekers K, Caris A, de Koster RB. Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. European Journal of Operational Research 2018;267(1):1–15.

Weidinger F. Picker routing in rectangular mixed shelves warehouses. Computers & Operations Research 2018a;95:139–50.

Weidinger F. A precious mess: On the scattered storage assignment problem. In: Operations Research Proceedings 2016. Springer; 2018b. p. 31–6.

Weidinger F, Boysen N. Scattered storage: How to distribute stock keeping units all around a mixed-shelves warehouse. Transportation Science 2018;52(6):1412–27.

Weidinger F, Boysen N, Schneider M. Picker routing in the mixed-shelves warehouses of e-commerce retailers. European Journal of Operational Research 2019;274(2):501–15.