Automatic Generator of Loading Rules and Its Applications on Logistics Operations

Anibal Tavares de Azevedo(✉)

State University of Campinas, Limeira, SP 13484-350, Brazil
atanibal@unicamp.br

Abstract. This paper presents an algorithm useful for many logistic processes: from loading containers into a ship to organizing cargo into stacks in a warehouse or packing cargo in a vehicle cargo. It was analyzed the feasible sequences to perform such operations and how it limits the possible sequences that could be created. Furthermore, it was showed the computational performance to propose a feasible solution for large-scale problems of regular spaces with 28,800 cells that are related to loading cargo into a container ship problem.

Keywords: Loading rules · Logistics operations · Stowage planning problem

1 Introduction

In several logistic processes, it is necessary to indicate how a given space should be occupied. Some examples of context and its application of such operation are in supply chain logistics, packing cargo, and warehouse operations [1, 3, 4, 6, 8]. The next subsections will detail these contexts and the article's contribution to them.

1.1 Motivation and Context

– Container port logistics

- Container ship: is necessary to know the order in which the cargoes will fill their regular spaces organized in stacks as described in Fig. 1 [1, 2];

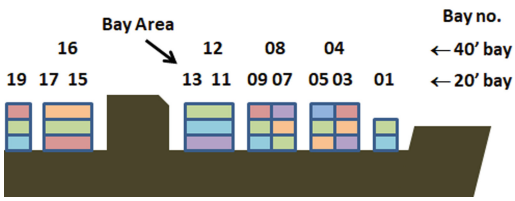

Fig. 1. Arrangement of containers in a ship.

S. R. González et al. (Eds.): DCAI 2021, LNNS 332, pp. 141–150, 2022.
https://doi.org/10.1007/978-3-030-86887-1_13

- Container port yard: this space serves as a place for the temporary storage of containers to facilitate importation and exportation flow in a port as described in Fig. 2. Several policies to organize containers could be employed like separating containers per blocks according to its destination [8];

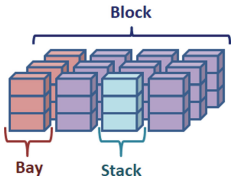

Fig. 2. Organization of containers in a port yard.

– Packing cargo

- On a truck: the containers should be organized in a manner that respects some stability and stacks constraints to avoid damage during cargo transportation [6] as shown in Fig. 3.

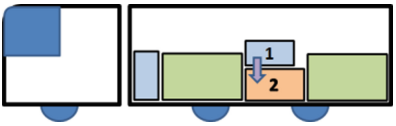

Fig. 3. Organization of packages in a truck.

- On a container: several packages and products should be organized inside a container as done in Fig. 4. Some important aspects to observe are: placing bigger boxes at the bottom of the container, a box organized in a stack should not overpass a weight limit [3, 6]. These containers will be transported by trucks or they will be loaded at a container ship.

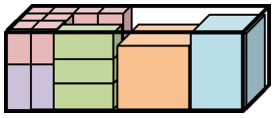

Fig. 4. Organizing packages in a container.

– Warehouse operations

- Storage area: the inbound flow of the products from the factory should be stored for future requisitions of client products [4] as shown in Fig. 5.

Fig. 5. Organization of packages into hacks of the storage area.

The picking area is an area where clients' orders are mounted from temporary stacks from the storage area as shown in Fig. 6. These orders will be grouped and loaded at a truck that will deliver orders [4].

Fig. 6. Order assembly using the temporary stacks of the picking area.

1.2 Contribution

A common situation to all problems described previously is the decision of which sequence should be used to organize the cargo in a given space divided as a grid. This approach could be applied to any regular space [7].

Although for each problem several mathematical formulations or simulation models were used to propose solutions that meet criteria of optimality or evaluation of these solutions, none of the methodologies was concerned with strictly generating a general and automatic procedure that could produce feasible solutions in any context.

The main contribution of this paper is the development of a general and automatic procedure that is context-free and could be employed in a wide array of problems.

Section 2 describes the problem, its complexity, and some computational results. Section 3 shows results for one context: the stowage planning problem. Section 4 presents conclusions and future works.

2 Problem Description

A feasible solution to organize cargo in a space divided as a grid consists in create a loading cargo sequence that should meet two physical constraints:

(i) One cargo cannot occupy the same space as another cargo at the same time;
(ii) A cargo that will occupy a position only if another cargo support it or the position is the bottom of a pile.

To create an algorithm that fulfills these two constraints, a study should be carried to know how they limit the possible sequences that could be generated. Subsection 2.1 introduces notation and Subsect. 2.2 employs it to explain the complexity of algorithms based on different strategies.

2.1 Notation and Definitions

Let it be a space that is composed of smaller units with the same dimensions, called cells, organized in m rows and n columns. Suppose the cargo is such that it occupies only the space of one cell. We want to apply an algorithm that generates a feasible sequence to fill the space.

Mathematically, this consists of generating a sequence of coordinates $((x_1, y_1), (x_2, y_2),\ldots, (x_n, y_n))$, where x_i is the row position and y_j is the column position, such that both criteria (i) and (ii) are met.

An example of the space-filling sequence is given in Fig. 7 for space with five columns (stacks), and with four rows (cargo capacity). The dimensions will be $m = 4$ and $n = 5$ and row and column zero index are in the left bottom of the space.

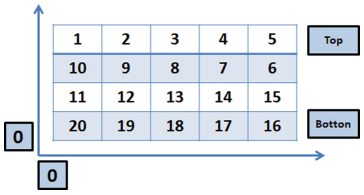

Fig. 7. An example of how to fill a regular space.

However, the filling sequence presented in Fig. 6 cannot be applied, since the first cargos should occupy the cells on the top of the stacks without any other cargo to support them.

There is no study in the literature of how many rules are possible without constraint (ii), or even how many rules can be generated considering constraints (i) and (ii) [1].

It is important to highlight that the possible number of combinations of filling in a $m \times n$ matrix is $(m \times n)!$, but without considering only the constraint (ii) as done in Fig. 6.

2.2 Strategies to Create Feasible Sequences and Their Complexity

To obtain sequences to fill a regular space considering both physical constraints, it is possible to apply a step-by-step process in which cargo is placed in each cell, and then the n candidate cells are listed for the next cargo to occupy. If space has dimension $m \times n$, then, at first, from n candidate cells to be occupied one will choose (cell (0, 2)), as illustrated in Fig. 8.

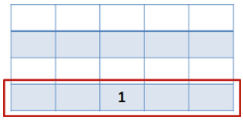

Fig. 8. Randomly choosing the first cell to put cargo from n possible cells.

After choosing the first place to put a cargo, then it is necessary to choose another place, from n possible cells, as done in Fig. 9.

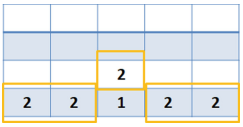

Fig. 9. Possible spaces for the second cell to put cargo from n possible cells.

If the option of filling the space per row is made ("Flat Way"), then, at each step there were n alternatives until $(m-1) \times n$ cells are filled and only n cells are left as provided in Fig. 10.

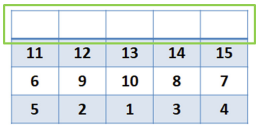

Fig. 10. Remaining spaces after employing filling according to "Flat Way".

From Fig. 10, the number of all possible feasible filling sequences, until n cells remain, is given by $n^{(m-1) \times n}$. For the remaining n remaining cells, the number of possible sequences is given by n!. Therefore, the total of possible filling sequences is given by $n^{(m-1) \times n} + n!$.

However, if a strategy of filling the space by columns is adopted ("Wall Way"), then, until the first column is filled at each step, there are n candidate cells as given in Fig. 11. After the first column is filled, m cells filled, the next m cells will have only n-1 cell alternatives for the next step.

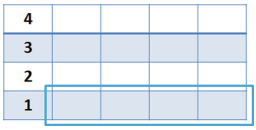

Fig. 11. The remaining space after filling m cells in one column.

Therefore, this filling strategy will result in $m \times (n^2 + n)/2$ possible filling sequences. It is important to note that the 'Wall Way' strategy is a special case of the 'Flat Way'

strategy since a column is selected it will be filled. Anyway, the 'Wall Way' strategy could be useful for certain purposes like group cargo by area.

Thus, there is a lower and an upper limit on the number of possible sequences to fill a space that meets physical constraints (i) and (ii). It is worth noting that this number is much smaller than the total of possible sequences that do not comply with either of the two fulfillment criteria (Complete Strategy) as given in Table 1.

Table 1. Several possible sequences for each strategy.

Strategy	Number of sequences
Complete	$(m \times n)!$
Flat Way	$n^{(m-1) \times n} + n!$
Wall Way	$m \times (n^2 + n)/2$

2.3 Computational Results

For illustration only, suppose that $m = 4$ and $n = 4$. Then, the total number of sequences in each strategy is given in Table 2.

Table 2. The number of possible sequences for each strategy in a space with dimensions 4×4.

Strategy	Number of sequences
Complete	20,922,789,888,000
Flat Way	16,777,240
Wall Way	40

In Table 2, it is possible to observe an advantage regarding the consideration of the two physical constraints: the considerable reduction in the maximum number of possible fill sequences from approximately 21 trillion to only approximately 17 million. Still, the number of possible sequences is considerably large.

Based on the previous analysis, to generate all possible combinations of sequences, that consider both constraints (i) and (ii), an algorithm to Generate Sequences for Filling Regular 3D Spaces (GSFIRES 3D), was created.

GSFIRES 3D consists of a step-by-step procedure in which cargo is placed in each cell and then the n candidate cells are listed for the next cargo to occupy. This cell selection process uses a random uniform distribution and its Python code is described in Fig. 12.

```
def GenRules(b,m,n):
  # Initial matrix M.
  M = [ [ [ (k,i,j) for i in range(m) ] for j in range(n) ] for k in range(b) ]
  # Random choosing the candidates available spaces from M to vR.
  vR = []
  for k in range(0,m*n*b):
    i = random.randint(0, len(M)-1)
    j = random.randint(0, len(M[i])-1)
    vR.append(M[i][j][0])
  return vR
```

Fig. 12. GSFIRES 3D for filling a 3D regular space with b × m × n dimension.

Table 3 gives the computational effort of GSFIRES 3D for several space sizes to create a filling rule. The environment used was Google Compute Engine Python 3 through Google Colab with 12 GB of RAM.

Table 3. The computational effort of GSFIRES 3D for each space size.

Dimension	Number of elements	Time (s)
1 × 3 × 4	12	0.0006
2 × 3 × 4	24	0.0010
4 × 3 × 4	48	0.0011
8 × 3 × 4	96	0.0011
16 × 3 × 4	192	0.0022
16 × 10 × 15	2400	0.0480
32 × 10 × 15	4800	0.1662
32 × 15 × 15	7200	0.2594
64 × 15 × 15	14400	0.9465
128 × 15 × 15	28800	3.9606

This computational effort study was important to show that this approach is viable to be employed in a Maritime Logistic Problem known as Stowage Planning [2].

3 Application on Stowage Planning

The implementation of the loading rules in the Stowage Planning depends on the ship dimensions. Typically, a container ship could store 20' feet containers in an amount between 2,400 to 28,800, and organized per bays, rows, and columns.

In the regular space of the container ship, each square with a number indicates that space is occupied with a container. The number inside a square specifies the destination port of a container.

Just to illustrate how the random generation of sequences will fill the container ship according to the setting of random seed coded in Python. Several instances of container ship dimensions and the number of containers are presented in Table 4.

Table 4. Features of container ship instances to test GSFIRES 3D.

Instance	Seed	# Bays	# Rows	# Columns	# Containers per port destination					
					Port 2	Port 3	Port 4	Port 5	Port 6	Port 7
1	0	2	3	4	2	4	3	2	1	1
2	1									
3	0	16	10	15	480	320	320	160	160	160
4	1									
5	0	128	15	15	720	480	480	240	240	240
6	1									

Figure 13 and 14 show results for instances 1 and 2, respectively. Table 5 presents the total computational time to create and employ the sequence generated to load a container ship.

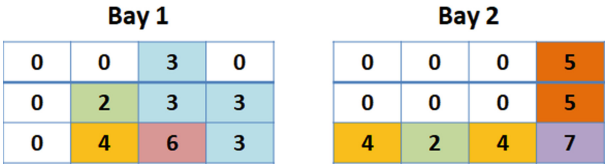

Fig. 13. GSFIRES 3D for filling a 3D regular space for instance 1.

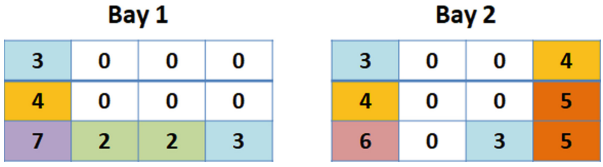

Fig. 14. GSFIRES 3D for filling a 3D regular space for instance 2.

Table 5. The computational effort of GSFIRES 3D for each container ship size.

Instance	Ship size	Total # of containers	Time (s)
1	$2 \times 3 \times 4$	24	0.0019
2			0.0023
3	$16 \times 10 \times 15$	2400	0.0528
4			0.0589
5	$128 \times 15 \times 15$	28800	3.6603
6			3.6881

4 Conclusions

It was proposed a general procedure to generate a feasible sequence to fill regular spaces with applications on several logistic processes: on ports: container ship and yard; on packing cargo: truck and container; on warehouse: storage and picking area. It was made a complexity analysis that helped to understand how certain filling strategies works and how to employ their properties to create randomly feasible filling sequences.

The complexity also enabled us to understand how the huge combinatorial number of strategies could be significantly reduced by the accomplishment of physical constraints related with all logistics context: a cargo that will occupy a position only if another cargo support it or the position is the bottom of a pile.

Furthermore, a code in Python programming language, called GSFIRES 3D, enables the random creation of feasible filling sequences, and its computational effort had been shown in a general filling problem.

The GSFIRES 3D was also adapted and computational tests carried attest to the algorithm speed and adequacy to propose and load a container ship ranging from 24 cells to 28,800 cells. For example, the algorithm was able to produce a feasible rule in less than four seconds for a container ship with a capacity of 28,800 containers.

For future works, this algorithm will be employed in a simulation-optimization scheme for port logistics planning. The program will be able to develop its own operating rules and select the best combination of them without human interference.

References

1. Azevedo, A.T., Salles, L.L.N., Chaves, A.A., Moretti, A.C.: Solving the 3D stowage planning problem integrated with the quay crane scheduling problem by representation by rules and genetic algorithm. Appl. Soft Comput. **65**, 495–516 (2018)
2. Azevedo, A.T., Ribeiro, C.M., Sena, G.J., Chaves, A.A., Salles, L.L.N., Moretti, A.C.: Solving the 3D Container ship loading planning problem by representation by rules and meta-heuristics. Int. J. Data Anal. Techn. Strat. **6**(3), 228–260 (2014)
3. Junqueira, L., Morabito, R.: Heuristic algorithms for a three-dimensional loading capacitated vehicle routing problem in a carrier. Comput. Ind. Eng. **88**(C), 110–130 (2015)

4. Karasek, J.: An overview of warehouse optimization. J. Adv. Telecommun. Electrotech. Signals Syst. **2**(3), 111–117 (2013)
5. Lee, B.K., Kim, K.H.: Optimizing the yard layout in container terminals. OR Spectr. **35**(2), 363–398 (2013)
6. Pollaris, H., Braekers, K., Caris, A., Janssens, G.K., Limbourg, S.: Vehicle routing problems with loading constraints: state-of-the-art and future directions. OR Spectr. **37**(2), 297–330 (2014). https://doi.org/10.1007/s00291-014-0386-3
7. Toledo, F.M.B., Carravilla, M.A., Ribeiro, C., Oliveira, J.F., Gomes, A.M.: The dotted-board model: a new MIP model for nesting irregular shapes. Int. J. Prod. Econ. **145**(2), 478–487 (2013)
8. Zhen, L., Jiang, X., Lee, L.H., Chew, E.P.: A review on yard management in container terminals. Ind. Eng. Manag. Syst. **12**(4), 289–305 (2013). Korean Institute of Industrial Engineers