Evolutionary Algorithm and Ant Colony Optimization for a Container Stowage Problem

Marcelo Pinheiro Leite Benedito
Universidade Federal de Viçosa
Viçosa - MG - Brasil
marcelo.benedito@ufv.br

André Gustavo dos Santos
Universidade Federal de Viçosa
Viçosa - MG - Brasil
andre@dpi.ufv.br

ABSTRACT

In the Container Stowage Problem, a container-ship must visit a sequence of ports, transporting containers. At each port there are containers destined to other ports visited ahead. Upon arrival at a port, the containers with destination to that port must be unloaded. If there are containers on top of them, they must be unloaded and reloaded again. The unloading and loading operations are very costly, so the problem aims to find a solution which minimizes the total number of undesirable movements. A heuristic that have never been used in this problem is presented, along with one that already has been. We show that an integer linear programming formulation of the problem is now able to solve real-sized instances, but not enough for all the cases. We follow the representation by rules presented in the literature, proposing new ones, more effective, that lead to better solutions for all the instances.

KEYWORDS. Container Stowage. Combinatorial Optimization. Evolutionary Algorithm. Ant Colony Optimization.

Main Area: MH - Metaheuristics; OC - Combinatorial Optimization

1. Introduction

Marine shipping, used for centuries, has as one of its principal activities the transportation of goods. Speed, safety and efficiency during the displacement of large loads are determining factors for a good deal, therefore, this type of transport is the subject of studies that aims to improve these activities, in order to increase profitability.

In this paper, we deal with the Container Stowage Problem, a marine shipping method consisting in transporting goods in containers. We must define a plan to stowage containers in a ship during a predefined route of ports, knowing the numbers of containers to be loaded and unloaded at each port. The balance of the ship is not taken into account, so there could be stacks on just one of the sides of the ship without compromising it. The plan must minimize the number of shifts during the loading/unloading operations, which is the number of containers unloaded before their destination port because of blocking other containers. These containers must be reloaded and this is a costly operation that should be avoided as much as possible.

In section 2 we formally define the problem as it was introduced by Avriel & Penn (1993), and discuss the results of the heuristics that have been proposed since then. The most successful heuristics use a representation by rules, which define for each port one of predefined schemes for loading and unloading containers. The rules already proposed are detailed in section 3; at the end of the section we describe the new rules we are proposing. Then, in section 4 we describe the Genetic Algorithm (GA) and the Ant Colony Optimization (ACO) heuristics we propose to solve the problem; both heuristics use a representation by rules. In section 5 we show the results of computational experiments of our heuristics and the one we found in the literature that yielded the best results so far; the results show that our heuristics substantially outperform the literature; besides that we show that an ILP formulation constantly used in several papers only to formalize the problem, is able nowadays to prove optimal solutions for instances with up to 25 ports when the destination of the containers are in average at a short distance from the origin port. Finally, we conclude this work at section 6.

2. Problem definition

The Container Stowage Problem studied here was introduced by Avriel & Penn (1993). It was shown to be NP-hard by Avriel, Penn, & Shpirer (2000), and since then several heuristics were proposed. The problem is defined as follows.

There is a container ship consisting of a regular bay for container stowage, and it is assumed that all containers are of the same standard size. The bay may support at most C stacks of containers, each one with at most R containers, so the bay may contain simultaneously at most $R \times C$ containers. The container ship must travel through a fixed route departing from port 1 and visiting sequentially ports $2, \ldots, N-1$ and finally reach port N. The ship starts empty at port 1 and finishes empty after visiting port N. At each port $i = 1, \ldots, N-1$ containers with destination to ports $j = i+1, \ldots, N$ can be loaded. A container may be loaded only at the top of a stack of containers, or on the bay ground (on an empty stack). When visiting a port, all containers with destination to that port must be unloaded and do not come back to the ship again. Containers with destination to a future port may be unloaded and loaded again for rearrangement. Particularly, containers that are blocking the containers for that port (i.e., if they are on top of them) are forced to be unloaded and then reloaded. The operation of unloading and reloading a container is called a shift. This is an expensive operation and the objective of the problem is to minimize the number of shifts. The input is the number N of ports, and the transportation matrix T, which is a $(N-1) \times (N-1)$ matrix whose entry T_{ij} is the number of containers at port i that must be transported to port j (assuming $T_{ij} = 0, \forall j \leq i$). The transportation matrix is an input for the problem, i.e., it is known before the trip starts, and the output is a container stowage planning: which containers are loaded and unloaded at each port, and where in the bay each container must be loaded. As mentioned, the container stowage planning should minimize the number of shifts.

Avriel & Penn (1993) defined the problem and also proposed an integer linear programming (ILP) formulation to solve it. The proposed model uses a 5-index binary variable x_{ijk}^{rc} to state that a container with origin i to destination j positioned at row r of stack c is unloaded on port v. An auxiliary variable binary y_i^{rc} is used to control if upon departure from port i the row r of column c is occupied by a container.

They show a solution for a 5-ports 19-containers instance, considering a (2×5)-container ship (i.e., a ship with 2 stacks with up to 5 containers each). This instance is of course a toy instance, used for illustration purposes. However, at that time, ILP solvers could not solve larger instances, then they proposed a heuristic method to decompose the transportation matrix of a given instance into smaller sub-matrices, which are solved by the ILP model. The heuristic is called Whole Column Heuristic Procedure, and its the main idea is to build stacks filled with containers of same origin at destination. The containers of those stacks can be loaded at the origin port and unloaded at the destination port with no shift at any intermediary port. These stacks can be loaded anywhere in the bay, then only the stowage planning of the remaining containers must be solved by the ILP. They show a solution for a 5-ports 162-containers instance, considering a (15×5)-ship.

The proposed ILP model was repeated in Avriel *et al.* (1998), Azevedo, Ribeiro, & Deus (2010), Ribeiro & Azevedo (2010) and Carraro *et al.* (2013), but not used in any of them. In this paper, we show that nowadays solvers are able to solve the model for larger instances. The 10-ports $\approx$ 1000-containers 6×50-ship instances from a benchmark set were solved to optimality in seconds. Some larger instances whose origin and destination for most of the containers are not far apart could also be solved in short computational time. However, only a few of the 15-ports instances without that characteristic could be solved. For most of the larger instances, not even a feasible solution was found within 1 hour. Then, heuristics should still be considered.

2.1. Heuristics from the literature

Avriel *et al.* (1998) proposed a heuristic called Suspensory Heuristic Procedure (SH). The heuristic uses function rules to decide for voluntary shifts in order to reduce the total number of shifts. The main idea is to rearrange some containers of some stacks with voluntary shifts (non necessary shift at that port) in order to avoid costlier shifts (i.e., involving more containers) in future ports. They compare the number of voluntary and necessary shifts and show the influence of the number of ports, the size of the ship and other characteristic of the instances, but do not compare the results with other methods.

Azevedo, Ribeiro, & Deus (2010) and Ribeiro & Azevedo (2010) proposed heuristics based respectively on Genetic Algorithm (GA) and Beam Search (BS), both using a novel solution representation: instead of working with each container or stack of the bay individually, there is a set rules for loading/unloading the containers from/to each port. These rules are chosen in advance and are also an input to the algorithm. What the algorithm has to decide is which loading and which unloading rule should be applied in each port. This leads to a compact representation, as it can be represented by a sequence of $N - 1$ rules, one for each port (no rule is needed for the last port, as the ship is emptied at that port). The drawback is that the optimal solution may be complex and not reachable by the set of rules, if they are too simple. They use the benchmark set of instances now publicly available by Azevedo (2012), but do not not compare their results. They conclude that the representation by rules is a good technique to speed up the solution of this problem and may lead to better results, however more elaborate rules are needed for larger instances. The next section describes their proposed loading/unloading rules and new ones proposed in more recent works.

The representation by rules was also used in Azevedo et al. (2011). They proposed a heuristic based on Simulated Annealing (SA) using 12 combination of rules: the 4 loading rules and the 2 unloading rules proposed by Ribeiro & Azevedo (2010) and 1 new unloading rule. The results are compared to the BS of Ribeiro & Azevedo (2010). For the same benchmark, the proposed SA outperforms the BS, reaching better results in 30 out of 45 instances, and the same result on the remaining instances.

Carraro *et al*. (2013) propose some new rules and two heuristics, one based on GA and other based on Artificial Bee Colony algorithm (ABC). For calibration purposes, the results of both heuristics for the 3 smallest instances were compared to each other and to the results of the GA proposed by Azevedo, Ribeiro, & Deus (2010). They report the solutions to only 6 out of the 45 instances of the benchmark and compare their GA and ABC but not compare the results to those of the literature. The results found by the GA were better or equal to the ones found by ABC.

The following section details the loading and unloading rules proposed in the cited papers from the literature, and the new rules we are proposing in this work.

3. Solution representation by rules

In this section we present all the loading and unloading rules found in the literature for the Container Stowage Problem. As mentioned in the previous section, the representation by rule leads to a compact representation and reduces considerable the search space. The drawback is that this representation does not map all possible real solutions into feasible candidates, because the loading and unloading is done only be means of the rules, which are decided per port, not per container. The search space is reduced substantially, with the cost of not including all feasible solutions, which means that the optimal solution may be missing. Nevertheless, they also facilitates to incorporate knowledge from experts because new rules may be created based on their experience, so it is expected to get better results if one has a better set of rules.

Firstly, we present the rules found in the literature and then propose a set of new rules. Using our proposed rules we could reach better solutions, including optimal solutions not found before.

3.1. Rules from Azevedo, Ribeiro, & Deus (2010) and Ribeiro & Azevedo (2010)

The first set of rules was proposed by Azevedo, Ribeiro, & Deus (2010) and Ribeiro & Azevedo (2010). Although from the same group of researchers, they are independent papers, with a slightly different set of rules. As they are similar, we describe them together.

Loading rules

- L_1: This rule fills the occupation matrix row by row, from left to right. Containers are inserted in decreasing order of destination, so that containers with destination to nearest ports are positioned on top.
- L_2: This rule fills the occupation matrix column by column, from left to right, until row θ_p is reached. The value θ_p is computed for each port, by the following equation, which determines the number of rows needed to accommodate all containers loaded on the ship when departing from port p. As in the previous rule, containers are inserted in decreasing order of destination.

$$\theta_p = \left\lceil \frac{\sum_{i=1}^{p} \sum_{j=p+1}^{n} T_{ij}}{C} \right\rceil \quad (1)$$

- L_3: same as L_1, but from right to left (used in Ribeiro & Azevedo (2010))
- L_4: same as L_2, but from right to left. (used in Ribeiro & Azevedo (2010))

Unloading rules

- U_1: Upon arrival at port p, all containers with destination to that port are unloaded, together with the containers blocking them in the stacks, i.e., all containers with destination p and those on top of them. Then, the blocking containers are reloaded together with the containers from that port.

- U_2: All containers are unloaded. i.e. the ship is fully emptied, which allows a rearrangement of the containers. The containers not destined to that port are then reloaded together with the containers coming from that port.(only in Azevedo, Ribeiro, & Deus (2010))

- U_2': All containers with destination to that port are unloaded, together with the containers of their stacks. i.e. all stacks with containers to port p are fully unloaded, which allows a rearrangement of the stacks. The containers not destined to that port are then reloaded together with the containers coming from that port.(only in Ribeiro & Azevedo (2010))

The SA from Azevedo et al. (2011) uses all the rules above.

3.2. Carraro *et al* 2013 rules

Carraro *et al*. (2013) proposed 2 new loading rules and modify the previous unloading rules.

Loading rules

- L_3: same as L_1, but from right to left (already proposed by Ribeiro & Azevedo (2010))

- L_4: same as L_2, but from right to left (already proposed by Ribeiro & Azevedo (2010))

- L_5: This rule fills the occupation matrix column by column, from left to right, as rule L_1, but until column β_p is reached, similar to rule L_2. The value β_p is computed for each port by the following equation. As in the previous rules, containers are inserted in decreasing order of destination.

$$\beta_p = \left\lceil \frac{\sum_{i=1}^{p} \sum_{j=p+1}^{n} T_{ij}}{R} \right\rceil \tag{2}$$

- L_6: same as L_5, but from right to left.

Unloading rules

- U_1': as in unloading rule U_1, containers with destination to the current port are unloaded, and the containers blocking them. Instead of reloading the blocking containers together with the containers from that port using the chosen loading rule, they are loaded using rule L_1, i.e., row by row, from left to right.

- U_2': as in unloading rule U_2, all containers are unloaded. As in unloading rule U_1' they are reloaded using rule L_1, before the containers from that port.

- U_3': as unloading rule U_1', but the blocking containers are reloaded as in rule L_2, column by column, left to right, filling the empty spots until the top, without a row limit θ_p.

3.3. New rules

The rules proposed in the literature are simple and provide good results. However, none of them uses the valuable information of the stowage planning on the last visited port. We propose three new loading rules, described in the following. The first one load the containers based on the number of containers currently on the stacks, the second is based on the destination port of the containers currently at the top of the stacks, and the third combines information from the stacks and the destination of the loaded container.

The experimental results on Section 5 shows that these new rules contributes significantly to reduce the number of shifts in the final solution.

New loading rules

- L_7: Let η_c be the number of containers currently at column c. Load containers column by column, until the top, in increasing order of η_c. As in other rules, containers are inserted in decreasing order of destination.

- L_8: Let δ_c be the destination of the container on top of column c (with $\delta_c = \infty$ if c is empty). Load containers in decreasing order of destination, column by column, until the top, in decreasing order of δ_c.

- L_9: Load containers in decreasing order of destination. The column chosen to load each container is the one that minimizes $\delta_c - d$, where d is the destination of the loaded container, i.e., load the container on the column whose top container remains less time after the container is unloaded. If no such column exists, then the container is loaded on the column with minimum δ_c.

3.4. Combinations of rules

To simplify the assignment of a loading and an unloading rule to a given port we represent every unique combination of the two types in a single variable; by doing that we facilitate some sections of the solution algorithms.

4. Proposed heuristics

Since the Container Stowage Problem is an NP-hard optimization problem (Avriel, Penn, & Shpirer (2000)), it generally is computationally hard to find exact solutions for big instances. When facing such problems we usually use heuristics, procedures capable of generating a sufficiently good solution that solves the problem without spending all the available resources. In the next sections we introduce the Genetic Algorithm and the Ant Colony Optimization, two heuristics that are widely used in the optimization field.

4.1. Genetic Algorithm

The Genetic Algorithm (GA) was designed to mimic the natural selection process, as theorized by Charles Darwin, by performing an adaptive search from a population of initially random solutions, attempting to find good ones. In the GA nomenclature, the solutions are called individuals and these are selected from the population, then reproduced, generating children that have some of its parents' features. Besides, the children can also mutate, that is, have a slight change on one of its features.

Initially, the random population is evaluated by a fitness function, which classifies the individuals by its fitness value. Then, the population is selected using an operator, by taking individuals and usually selecting the fitter one. The crossover combines individuals from the new population, creating a child with some of its features. The last step in a cycle is the mutation that has a chance to modify a feature in a child, according to an operator. After, a new cycle unfolds, exposing the new population to the same steps, until reaching a stop criteria. A pseudocode of the whole algorithm is shown in Algorithm 1. In our algorithm a solution is represented by a sequence of N values, representing the loading/unloading combination to be used on each port.

Next, we have the chosen values of the parameters. The size of the population is fixed in 80 individuals and the maximum number of cycles is 1200. The selection operator is the tournament selection (Mitchell (1996)), which selects two random individuals and picks the fitter with a 75% probability. The crossover operator is the uniform crossover (Michalewicz (1996)), which two individuals are selected to build the child's features, randomly choosing one of them with the same probability to give the feature. The mutation operator is the uniform mutation (Mitchell (1996)), which chooses a feature of an individual, with a probability of 35%, and change its value.

Algorithm 1 Genetic Algorithm

```
1:  procedure GA()
2:      P ← initialPopulation()                ▷ Build a set of N_pop random individuals
3:      repeat N_gen times
4:          R ← ∅
5:          repeat N_pop/2 times
6:              parent_1 ← selection(P)
7:              parent_2 ← selection(P)
8:              child_1, child_2 ← crossover(parent_1, parent_2)
9:              mutation(child_1)
10:             mutation(child_2)
11:             R ← R ∪ {child_1} ∪ {child_2}
12:         end repeat
13:         P ← best(P ∪ R)                    ▷ Select the best N_pop individuals
14:     end repeat
```

4.2. Ant Colony Optimization

As the previous heuristic, Ant Colony Optimization (ACO) is inspired by nature phenomenon: the ant's trail system. Some species of ants lay pheromone from its colony to a possible place with food. If indeed there is food in that place, more ants are likely to go there and increase the pheromone trail, since the first returned with food; if there is not, the trail will evaporate over time until becoming unnoticeable. This idea is used to build solutions based on the paths taken by the previous ones in the problem's graph, where the paths that originated better solutions are more probable to be chosen. The evaporation is also applicable since a better path can be discovered and have its pheromone increased, whilst the old one decays for not being so used anymore.

The graph we propose for the Container Stowage Problem is dependent on the quantity of ports and the number of combinations of rules. Each port is represented in a node. Each node is linked to its successors node and the number of links to each one is equal to the number of combinations of rules. A link between a node i and a node j contains the combination of rule that will be used in node i, so the first node ignores its unloading rule whilst the last node ignores its loading rule. In this way, we can represent all the possibilities of any set of combinations to be used. A representation of this graph is given in Figure 1.

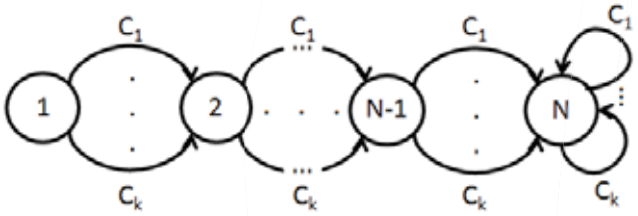

Figure 1: Representation of the graph used in ACO with N ports and k combinations of rules

Initially, every arc has the same probability to be chosen, so their pheromone values are the same. Then, a fixed number of ants are released in the graph, on the first node, walking on the graph until the last node, building solutions based on the arc's probabilities. When all the solutions are built, a fitness function evaluates them and increases the pheromone of the arcs that the best ones went through. Before beginning a new cycle, the pheromone on the arcs evaporates at a fixed rate. The whole procedure is shown in Algorithm 2.

Next, we have the chosen values of the parameters. The number of ants is 120 and the

number of cycles is 800. The nodes have an initial pheromone value of 5, while the increase value in the arc is 40. The number of solutions that have its path's pheromone updated is 2 and, in the end of each cycle, 1% of an arc's pheromone value is evaporated.

Algorithm 2 Ant Colony Optimization

```
1: procedure ACO()
2:     G ← createGraph()          ▷ Creates the graph describing the problem
3:     repeat N_gen times
4:         S ← buildSolutions()
5:         rewardSolutions(S)
6:         updatePheromone(G, S)
7:     end repeat
```

5. Computational experiments

As defined in Avriel *et al*. (1998), there are three types of instances of the problem that are related to the average period of time of a container on board:

1. Mixed Distance: containers are assigned random time periods
2. Long Distance: containers have long time periods
3. Short Distance: containers have short time periods

Besides being separated by the permanency of the container in the ship, the instances have also different number of ports, ranging from 10 to 30. The instances of the problem provided by Azevedo (2012) have ships with different sizes, that is, with different number of rows and columns. In the first analysis, the instances chosen were those with 6 columns and 50 rows, the same configuration used by Carraro *et al*. (2013). The proposed solution methods were compared to an integer linear programming (ILP) formulation of the problem (Avriel *et al*. (1998)) and to a Genetic Algorithm based on Carraro *et al*. (2013). These paper's algorithms were implemented in an Intel(R) Core(TM) i7 CPU 2.83GHz, 8.0 GB of RAM, Windows 7 operating system with C++ programming language.

Table 1 in the following summarizes the results of all solution methods, where ILP93 is the ILP formulation solved by CPLEX 12.4, with a one-hour time constraint; GA_18 is the Genetic Algorithm based on Carraro *et al*. (2013), therefore, with 18 combinations of rules; GA_45 is the Genetic Algorithm with 45 combinations of rules here proposed and ACO is the Ant Colony Optimization, with all the combinations as well:

Noticeably, the ILP formulation is capable of finding optimal results for some instances, but in the bigger ones it runs out of time. The model usually finds a solution to the instances of type 3 because it is the short distance transportation matrix type, so it is easier to manage the containers through the process. Though a solution was found to various short-distance kind of instances, it failed to find one for the 30 ports case, so the use of heuristics is well justified.

Comparing both Genetic Algorithms, we can see that the new set of rules has unarguably increased the quality of solutions without taking much longer. These rules examine the current state of the ship and take advantage on it to load containers, instead of placing them blindly. This examination process is the major time-consuming task in the algorithm, so it is efficiency is indispensable. Another performance improving technique is the use of generic unloading rules: it is far better than applying a specific rule to reload the containers, as in Carraro *et al*. (2013); instead of that, all loading rules can be applied in any reload procedure.

The difference between GA_45 and ACO is very tenuous, since their differences are not big, but, generally, ACO has found solutions with lower number of shifts in less time. Trying to

Table 1: Results of 6x50 instances: time (in seconds), average number of shifts for 30 runs and standard deviation

	ILP93		GA_18			GA_45			ACO		
ID	Time(s)	Shifts	Time(s)	Shifts	(σ)	Time(s)	Shifts	(σ)	Time(s)	Shifts	(σ)
10/1	12	0	8	8,2	(1,8)	0	0,0	(0)	0	0,0	(0)
10/2	10	0	6	5,1	(0,7)	10	3,1	(0,3)	10	3,1	(0,3)
10/3	1	0	0	0,0	(0)	0	0,0	(0)	0	0,0	(0)
15/1	82	0	11	21,9	(4,4)	1	0,1	(0,7)	1	**0,0**	(0)
15/2	1921	0	8	9,9	(2,3)	13	1,3	(0,5)	13	**1,0**	(0)
15/3	1	0	0	0,0	(0)	0	0,0	(0)	0	0,0	(0)
20/1	1155	0	15	30,6	(6)	24	4,6	(2,5)	25	**3,4**	(1,1)
20/2	†	-	11	21,3	(4,6)	17	8,9	(1,9)	18	**7,3**	(1,4)
20/3	1	0	0	0,0	(0)	0	0,0	(0)	0	0,0	(0)
25/1	†	-	16	24,5	(7,9)	25	11,0	(2,3)	25	**8,8**	(1,8)
25/2	†	-	13	17,1	(7,3)	19	2,0	(1,6)	15	**1,0**	(1)
25/3	2	0	0	0,0	(0)	0	0,0	(0)	1	0,0	(0)
30/1	†	-	22	41,1	(11,1)	33	8,7	(2,5)	32	8,7	(2,4)
30/2	†	-	17	41,8	(7,9)	26	**12,0**	(3,3)	26	14,7	(1,9)
30/3	†	-	0	0,0	(0)	0	0,0	(0)	0	0,0	(0)
Average time & total shifts			8,5	221,4		11,2	51,8		11,1	48,0	

† No solution within 1 hour.

Table 2: Total number of shifts per instance type

	ILP93	GA_18	GA_45	ACO
1 - Mixed distance	-	126	24	21
2 - Long distance	-	95	27	27
3 - Short distance	-	0	0	0
Total of best solutions	9	5	8	13

find the best algorithm, we ran tests for the instances with 6 rows by 100 columns and 6 rows by 150 columns; their results are in tables 3 and 4.

In the instances with higher number of rows and columns, GA_45 and ACO still reaches with similar results. The average time to find a solution increased, since we now have bigger ships therefore more containers to manage. Comparing the sets of instances, the results of the 6x50-instances tend to have small amounts of shifts in every instance, whereas the 6x100 and 6x150 have found more optimal solutions. It can be explained by the size of the ships: the more you have, the easier the management; on the other hand, if the number of containers in the instance is close to the viable maximum, any shift may have a large impact in the total value.

The line chart of Figure 2 shows the total number of each loading rule used in the current solution of the GA population at each iteration, for two different instances. At the beginning each rule is used about the same number of times, since the initial population is generated randomly. During the process, the loading rules L_7, L_8 and L_9 tends to be more frequent in the population. In fact, for these instances the rules L_7 and L_9 end up to be the most frequent rules in the last iterations. This show that, at least in these instances, the new rules we introduce in this paper tends to be frequently used by the best solutions.

Table 3: Results of 6x100 instances: time (in seconds), average number of shifts for 10 runs and standard deviation

	GA_45			ACO		
ID	Time(s)	Shifts	(σ)	Time(s)	Shifts	(σ)
10/1	0	0,0	(0)	0	0,0	(0)
10/2	17	**4,0**	(0)	17	4,2	(0,6)
10/3	0	0,0	(0)	0	0,0	(0)
15/1	0	0,0	(0)	1	0,0	(0)
15/2	29	5,1	(1,4)	30	**4,8**	(1,1)
15/3	0	0,0	(0)	0	0,0	(0)
20/1	1	0,0	(0)	5	0,0	(0)
20/2	37	10,0	(1,3)	37	**8,8**	(0,4)
20/3	0	0,0	(0)	1	0,0	(0)
25/1	61	10,5	(0,8)	58	**9,9**	(1,3)
25/2	54	**2,5**	(0,8)	49	3,1	(1,7)
25/3	1	0,0	(0)	2	0,0	(0)
30/1	64	**1,7**	(1,7)	65	2,7	(2,5)
30/2	57	**7,1**	(6,2)	53	8,6	(2,5)
30/3	1	0,0	(0)	3	0,0	(0)
Average time & total shifts	21,6	40,9		21,4	42,1	

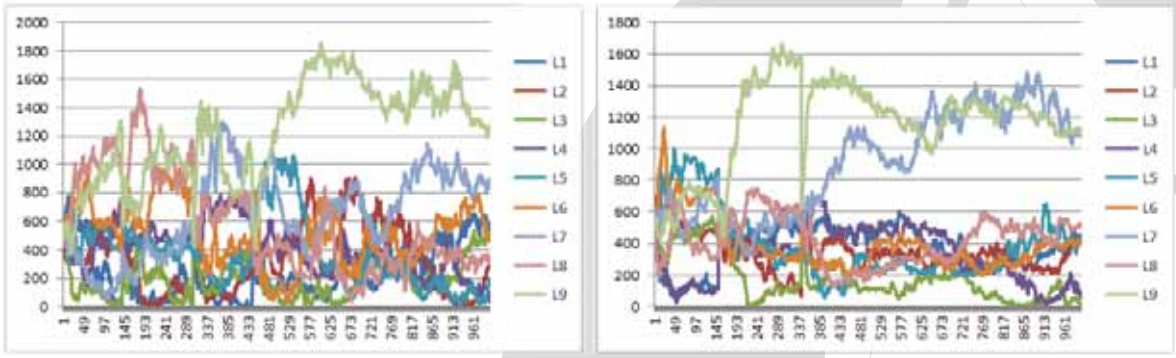

Figure 2: Line chart of the total number of each loading rule used over time by GA for 6×50 instances $20/2$ and $30/1$

For a complete view regarding the loading rule usage, the pie chart of Figure 3 represents the average number of each loading rule at the end of the execution of instances. The instances of type 3 were ignored in this comparison since they can be solved within a few cycles, so the quality of the rules does not apply as much as the ones that take the entire cycle. Representing almost 50% of the total, the proposed rules take a big part in building better solutions, showing their importance in the quality of the results found.

Table 4: Results of 6x150 instances: time (in seconds), average number of shifts for 10 runs and standard deviation

	GA_45			ACO		
ID	Time(s)	Shifts	(σ)	Time(s)	Shifts	(σ)
10/1	0	0,0	(0)	0	0,0	(0)
10/2	0	0,0	(0)	0	0,0	(0)
10/3	0	0,0	(0)	0	0,0	(0)
15/1	1	0,0	(0)	2	0,0	(0)
15/2	39	1,0	(0)	37	1,0	(0)
15/3	0	0,0	(0)	0	0,0	(0)
20/1	3	0,0	(0)	7	0,0	(0)
20/2	62	4,3	(2,8)	61	**3,3**	(2,3)
20/3	1	0,0	(0)	2	0,0	(0)
25/1	10	0,0	(0)	27	0,0	(0)
25/2	29	0,4	(1)	36	**0,0**	(0)
25/3	2	0,0	(0)	3	0,0	(0)
30/1	112	**5,5**	(0,5)	95	7,4	(1,8)
30/2	66	5,7	(1,4)	68	**4,4**	(1)
30/3	2	0,0	(0)	8	0,0	(0)
Average time & total shifts	21,9	16,9		23,1	16,1	

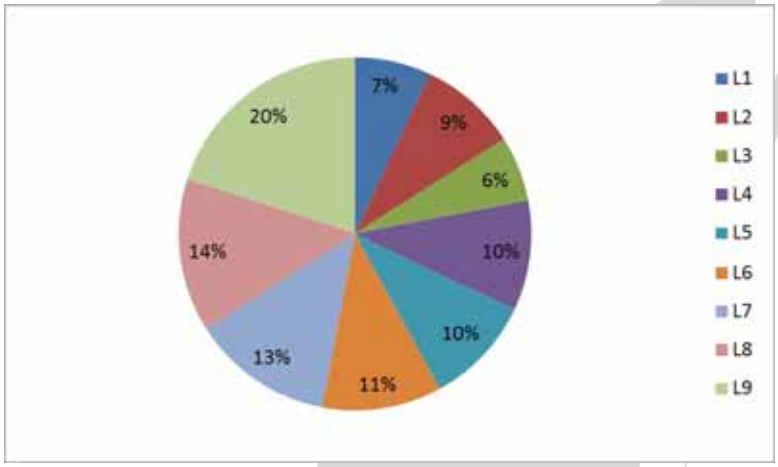

Figure 3: Pie chart showing average number of each loading rule used

6. Conclusions

In this paper, we discussed the Container Stowage Problem, modeling a situation where a container-ship must visit a sequence of ports, carrying containers from their initial ports to a final destination. The objective is to minimize the amount of unnecessary displacement of containers, a costly operation. Our contributions improves the methods of solution seen in the literature so far, which gives us more profitable, less time-consuming processes in the daily routine of a marine shipping environment. The new rules presented are capable of analysing the ship's state and direct the search towards a better solution. Though these rules cause the algorithm to run slower, its gains are incomparable, since our heuristics substantially outperform the literature.

We show that the integer linear programming formulation of the problem is now practicable to solve larger instances than before, which suggests that an exact approach could be taken to find even better solutions.

The implementation of the Ant Colony Optimization algorithm opened up a new range of possibilities, showing that there is an alternative to the previously tried heuristics, for the results found are comparable both in quality and time.

Acknowledgement

The first author worked on this project funded by CNPq. The authors also thank to CAPES and FAPEMIG for the partial financing of the project.

Avriel M., Penn M., Exact and approximate solutions of the Container Ship Stowage problem, *Computers and Industrial Engineering*, 1993.

Avriel M., Penn M., Shpirer N., Witteboon S., Stowage Planning for Container Ships to Reduce Number of Shifts, *Annals of Operations Research*, 1998.

Avriel, M., Penn, M., Shpirer, N., Container ship stowage problem: complexity and connection to the coloring of circle graphs, *Discrete Applied Mathematics*, 103, 271– 279, 2000.

Azevedo, A. T., Problemas & Dados, Instâncias empregadas nos artigos do SIMPEP 2009, Universidade Estadual Paulista, (https://sites.google.com/site/projetonavio/programas-dados), 2009.

Azevedo, A. T., Ribeiro, C. M. e Deus, N. M. R. , Resolução do Problema de Carregamento e Descarregamento de Contêineres em Terminais Portuários via Algoritmo Genético, *Revista Ingerpro – Inovação, Gestão e Produção*, 2, 38-51, 2010.

Azevedo, A. T., Sena, G. J., Chaves, A. A., Ribeiro, C. M., O problema de carregamento e descarregamento de contêineres em terminais portuários: uma abordagem via Simulated Annealing, *Simpósio Brasileiro de Pesquisa Operacional*, 2011.

Carraro, L. F., Chiwiacowsky, L. D., Gómez, A. T., Oliveira, A. C. M., Uma aplicação das metaheurísticas Algoritmo Genético e Colônia Artificial de Abelhas através da codificação por regras para resolver o Problema de Carregamento de Navios-Contêineres, *Simpósio Brasileiro de Pesquisa Operacional*, 2013.

Michalewicz, Z., Genetic Algorithms + data structures = evolution programs, *Springer-Verlag*, Berlin, 1996.

Mitchell, M., An introduction to genetic algorithms, *MIT Press*, Cambridge, 1996.

Ribeiro, C. M., Azevedo, A. T.,, Resolução do problema de carregamento e descarregamento de contêineres em terminais portuários via Beam Search, *Simpósio Brasileiro de Pesquisa Operacional*, 2010.