Hybridization of GRASP algorithm with genetic algorithm.

Christian Perez, Miguel A. Salido
[1]Universitat Politécnica de Valencia
Camí de Vera s/n
Valencia, Spain
cripeber@doctor.upv.es, msalido@dsic.upv.es

Abstract

(Por escribir)

Introduction

(Por Escribir)

Problem specification

This section features the most relevant aspects of the problem specification proposed by the company with the intention of providing a better comprehension of the presented algorithm. The objective of the problem to solve is to minimize the transport cost and the stock cost by assigning each travel to a warehouse, taking into account every characteristic of each travel and warehouse. The minimum cost of transportation is obtained by finding the trip with the minimum value of transportation, which makes this process relatively easy to perform. However, a problem arises when looking for the minimum stock cost due to the fact that there must be a balance between the stock of each type of container. In order to solve this problem, it is needed to stablish a multi-objective function by assigning weights to the stock cost and the transport cost in order to determine the fitness function. The objective of this function is to maintain a stable transport cost while distributing homogeneously the stock between the warehouses. Thus, the most important variables in the problem are taken into account.

- **Transport cost** (TC): Cost asociated with an order and based on the distance between the warehouse and the starting point. This is one of the values that is taken into account in the calculation of the final cost in the fitness function.
- **Stock cost** (SC): If an order is assigned to be loaded at a warehouse where the current stock is less than the stock requested, the negative stock remaining given by this load operation must be replaced. Thus, the cost of the stock is proportional to the product of the cost of replacing each of the items and the negative stock.

- **Delay of delivery** (DD): Each warehouse that can be associated with an order is located in a different place, such that the distance between each warehouse and the starting point is different. Therefore, the delay of delivery value depends on the warehouse chosen by the order. This delay affects how much stock is reduced from the warehouse from the scheduled date to the actual delivery date.
- **Single load point** (SLP): For each order there is a number of possible warehouses where it can be loaded. All the items in an order must be loaded in the same location, creating the constraint that each order can only be loaded at one warehouse.

The problem is formed by a dataset (DS) that is based on the combination of three main elements. This three elements contain all of the information necessary to represent an instance of the problem (see Equation 1). Hence, each problem instance is formed by a group of orders (O) each of which represent a trip to transport items, all of the warehouses that have every item requested by the order (W), and a set of prices (P) for each item.

$$DS = \{O, W, P\} \tag{1}$$

The set $O = \{o_1, o_2, \ldots, o_N\}$ contains all orders, where N is the number of orders. Each order $o_i \in O$ is formed by three parameters $o_i = [D_{o_i}, Ir_{o_i}, W_{are_{o_i}}]$:

- D_{o_i} is the delivery date of o_i.
- Ir_{o_i} is the set of items requested in the order o_i (see Equation 2). Each element $ir_{o_i j} \in Ir_{o_i}$ represents the number of items that have to be loaded.

$$Ir_{o_i} = \{ir_{o_i 1}, ir_{o_i 2} \ldots, ir_{o_i P}\} \tag{2}$$

- $W_{are o_i}$ is the set of possible warehouses that have every item requested in order o_i (see Equation 3).

$$Ware_{o_i} = \{Ware_{o_i 1}, Ware_{o_i 2}, \ldots Ware_{o_i Q}\} \tag{3}$$

Each $W_{o_i j}$ is formed by three values:

  - $ava_{o_i j}$ represents if the warehouse j is available to load all of the items requested by the order o_i. Therefore, $ava_{o_i j} = 1$ means that the warehouse is available, and $ava_{o_i j} = 0$ means that it is not.

– $pr_{o_i j}$ contains the transport cost of order o_i from warehouse j. If $ava_{o_i j} = 0$, this value is null.
– $dl_{o_i j}$ is the delay to load the order o_i from warehouse j. If $ava_{o_i j} = 0$, this value is null.

At the beginning of each week and in order to control the amount of stock, the current status of each warehouse is provided by the company through the use of a three-dimensional matrix W (see Equation 4).

$$W = \{w_1, w_2, w_3, \dots w_P\} \tag{4}$$

where each $wst_i \in W$ is composed of a bidirectional matrix (item x week days). Each of the logistic centers (warehouses) has the ability to replace the lack of items by shipping them. This allows the company to generate negative stock in the warehouses asociated to an order $wst_{ijk} \in \mathbb{N}$.

Finally, P is the set of unitary price for each items (see Equation 5).

$$P = \{p_1, p_2, p_3, \dots, p_i, \dots p_P\} \tag{5}$$

Solving techniques

This section explains the different algorithms being used in the evaluation of the solving of the problem proposed by the company, pointing out the the final method used, which is an hybridization of the GRASP algorithm and a Genetic Algorithm (GA). The resolution techniques presented are the Greedy algorithm, the GRASP algorithm and the Genetic Algorithm.

The Greedy algorithm

Currently, the purpose of the company is to plan orders each week that are based on the demand of customers, considering the expected available stock for each day. The solution for this purpose is based on a greedy technique. The availability of each item is obtained from a given loading plan that its updated each week. This update consists on the stock remaining from the week before, the stock obtained by shipments, etc. This algorithm is determined by different constraints, imposed by factors such as customers, warehouses and geographical areas:

- There are clients who only allow loading at specific warehouses due to convenience factors and distance.
- The assingments of warehouses to orders have a priority that is dependent on the geographical area or zone of influence.
- The order is restricted by the type of items it contains, thus it can not load at a warehouse that does not have every item requested.
- Theq quantity of products (vegetables, fruits, fishes) is not constant thorughout the year, therefore the warehouses have different delays for each product depending on the time of the year.

The current algorithm developed by the company is composed of two layers. First, the algorithm uses a heuristic called closeness centrality that obtains a measure of centrality of a transport network by adding the distance between the different nodes, obtaining the central nodes (warehouses) as those that have minimum distance to the other nodes. Then, all orders are classified according to the item with the largest part of the order.

In the second layer, if an order does not meet the restrictions of the first layer it is not assigned to a warehouse. These orders are assigned warehouses with larger stock that are less central. The orders that are still unassigned can be reordered using the customers in order to improve the warehouse system or to make up for the lack of stock. However, there may be still unassigned orders which are assigned after an expert human looks for combinations that improve the global transport cost withouth increising stock cost. It is necessary for an expert to review all of the trips in order to obtain a better solution for the company.

GRASP

A GRASP algorithm has been developed taking into account the drawbacks of the Greedy algorithm used by the company, in order to obtain in terms of stock balance and travel cost a balanced solution.

Algorithm 1 GRASP

1: *input:* All orders O, All stock matrix W, Item price vector P, Size of LCR list n
2: *output:* Optimized solution s^*
3: $i \longleftarrow 0$
4: $t \longleftarrow [1, 2, 3, \dots, |O|]$
5: $LCRordering(\alpha)$
6:
7: $s \longleftarrow [\quad]$
8: $LCR \longleftarrow [o_1, \dots, o_n]$
9: **while** $|s| \neq |LCR|$ **do**:
10: $j \longleftarrow 0$
11: $lcr \longleftarrow [\quad]$
12: **while** $j < |LCR|$ **do**:
13: $lcr \longleftarrow lcr \cup$ **LocalSearch**$(LCR_j, \alpha, |LCR|)$
14: $j \longleftarrow j + 1$
15: **end while**
16: $s \longleftarrow s \cup lcr*$
17: $LCR \longleftarrow LCR \setminus lcr*$
18: $LCR \longleftarrow LCR \cup o_{n+1}$
19: **end while**

The proposed Algorithm 1 is composed of two parts. First, the set of orders is sorted with purpose of improving the efficiency of the algorithm (line 5). This type of sorting is based on the idea of constrainedness, meaning giving the algorithm more decision power in the last iterations by analyzing the most restrictive orders in the beginning. Second, in the iterative part of GRASP (from line 9 to 19), a list (called LCR) that contains the candidates to assign a warehouse given by the algorithm is obtained. Later, through the use of local search the best warehouse stablished. When the best warehouses for all the trips are known, the warehouse with the lowest cost is selected, deleting the order from the LCR list and adding the next one.

LRC ordering: The sorting of the LCR list is carried out to give more flexibility and to test the behavior of the algorithm with different inputs. This pre-processing auto-allocates all of the trips that have only one available warehouse due to availability or customer restrictions.

- **Sort by number of warehouses:** The list is ordered by the number of possible warehouses, that is, the first order an the list will have fewer warehouses, allowing the algorithm to assign them with fewer of decisions to make (constrainedness).
  - **Sort by number of warehouses and items:** The trips ordered by warehouses and with the same value are ordered by the lowest number of items required for the trip.
  - **Sort by number of warehouses and delay:** The trips ordered by warehouse are ordered according to the number of days of delay for that order. This is because the more days of delay there are, the more blocked stock remains during the week.
- **Sort by Delay and Items:** In this case, orders are classified by the sum of the delays of all their warehouses and by the amount of items required for the trip.
- **Sort by warehouses plus standardized delay and items:** This sorting is carried out by mixing the three criteria previously defined. To do this, we add the standardization of the delay and the number of warehouses and then order by the number of items.
- **Sort by number of weight warehouses and delay:** This sorting is based on the previous one by adding a beta weight to the delay. Then, the sorting is done by the number of items.

After carrying out some tests to verify the viability of all sorting types, it was found that sorting by the number of warehouses(more specifically, sorting by number of warehouses and items) gave the best results without negatively affecting time or computer costs.

Local search

Once the order list is sorted, we iterate it to execute a local search on each order in order to obtain the best assignation. This local search is based on obtaining the objective function for each of the possible warehouses in a trip, by generating a list of values for each warehouse. Once this list is obtained, we obtain the minimum value on the list and the selected warehouse is assigned to the trip.

Algorithm 2 Local Search

1: *input:* A order o, List of item price P, Alpha value for objective function α, Length of LCR list $nLCR$
2: *output:* Index of warehouse chosen j, Minimum cost of warehouse chosen $Q*$
3: $Q \longleftarrow [\quad]$
4: $I \longleftarrow Ir_o$
5: $M \longleftarrow W_o$
6: **For** j **in** $\{0, \ldots, |M|\}$:
7: $\quad Cs \longleftarrow []$
8: $\quad Ct \longleftarrow$ **getTransportCost**(M_j)
9: $\quad$ **For** i **in** $\{0, \ldots, |I|\}$:
10: $\quad\quad Cs_i \longleftarrow P_i * \sum_{k=D_o}^{|wst_{ij}|} wst_{ijk} \mid wst_{ijk} < 0$
11: $\quad\quad Cs \longleftarrow Cs \cup Cs_i$
12: $\quad$ **end for**
13: $\quad Q_j \longleftarrow \alpha * Ct + (1 - \alpha) * \sum Cs$
14: $\quad Q \longleftarrow Q \cup Q_j$
15: $\quad F_\alpha \longleftarrow (1 - \alpha)/nLCR$
16: $\quad$ **if** $\sum Cs < 0$ **then**:
17: $\quad\quad \alpha \longleftarrow \alpha + F_\alpha$
18: $\quad$ **else:**
19: $\quad\quad \alpha \longleftarrow \alpha - F_\alpha$
20: $\quad$ **end if**
21: **end for**

In each iteration, one of the orders an the LCR list is analyzed, and the objective function is calculated with all the items that compose the order and the possible warehouses in order to obtain the best one. The objective function is based on calculating the transport cost and the stock cost and weighting them with an alpha value in order to give more weight to the transport cost which generates more costs (see Line 15). The transportation cost of a trip is obtained by adding the price of traveling to a warehouse for each order (see Line 8). The stock cost is obtained by subtracting the quantity of an item required in a warehouse an a certain day, multiplying it by the price of manufacturing the item in the case that the stock is negative. If the stock remains positive the stock cost will be zero (see Line 11). Finally, the warehouse assignments for each trip are saved.

Improvements

During the development of the algorithm, several problems appear that negatively affect the optimization of the solution. To overcome these problems, parts of the main code have been improved:

- **LCR sorting:** As mentioned above, the ordering of the LCR list improves the allocation of the order with the most restrictive characteristics, leaving those with a larger set of solutions for the end.
- **Shuffle:** In each one of the iterations of the GRASP algorithm, it is observed that the LCR list sorting generates the same order of the orders. A shuffle method has been developed that randomizes the LCR list from the criteria used in LCR sorting. This method allows us to generate in each of the iterations of the algorithm a different LCR list from the others in order to avoid identical solutions.

- **Scheduler:** The objective function seeks to minimize both the transport cost and the stock cost. To do this, an alpha value is applied to each of the variables to be minimized in order to find the optimal point. During the execution of the tests, it is observed that to greater amount of items with negative stock in the warehouses the objective function is polarized losing the optimal value. This is because of a fixed alpha value that reduces the dynamism of the function and, therefore, worsens the results. For this purpose, a *scheduler* that modifies the alpha value in each of the iterations has been developed. This scheduler increases or decreases the alpha variable regardless of whether or not the assigned order is in a warehouse with negative stock. The factor that is used in the *scheduler* is calculated by dividing the rest of alpha minus 1 by the number of trips that remain to be assigned. This allows the *scheduler* to minimize the stock cost more easily.

Evaluation

This section presents the results of each of the proposed improvements, as well as the results of the comparison of the greedy algorithm and the GRASP algorithm with all of the implemented improvements.

The first part of the algorithm to be improved is the sorting of the list of possible candidates. This is done by obtaining the different costs for each of the models explained (see Fig 1).

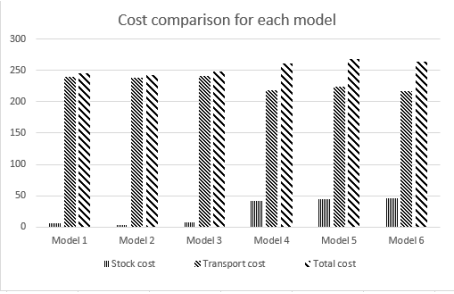
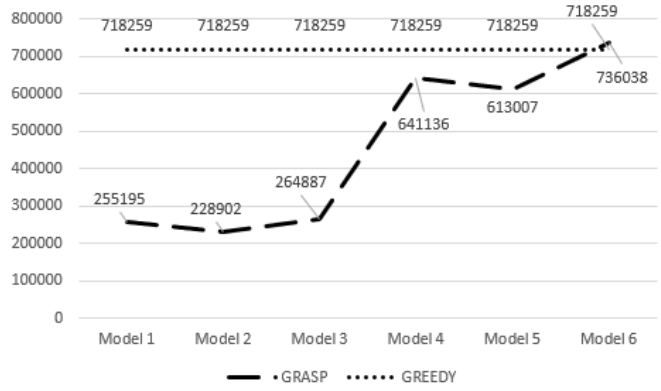


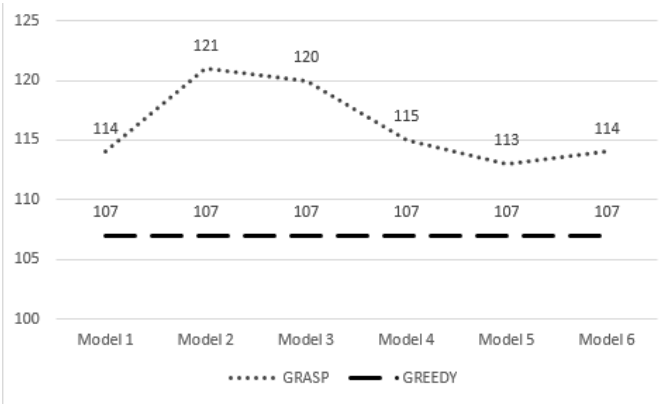
Figure 1: Cost comparison for each model.

These results indicate that in models where delay is prioritized over warehouses or items, transport cost improves but stock cost worsens. To contrast in those models where the number of available warehouses is prioritized and the models are combined with other parameters, a stable transport cost and a significantly lower stock cost are obtained. In order to check these results, a series of metrics is obtained that allows a deeper evaluation of the stock balance in the warehouses by comparing these metrics in the greedy algorithm and the proposed one. To do this, we first obtain the average negative stock of all platforms (see Fig 2a).

The average number of items with negative stock reinforces our hypothesis that the number of warehouses available per trip is a determining factor. In order to decide which model to use for the LCR list, we obtain the number of items where the stock is negative (see Fig 2b))



(a) Average



(b) Quantity

Figure 2: Metrics of items with negative stock for each model.

Finally, we decided to use the ordering model based on quantity of available warehouses and amount of items in the order. We selected this model because it is the one that generates the least number of items with negative stock and the lowest average of negative stock, and it keeps the cost of transport constant.

The set of improvements optimize the scheduling of the algorithm and maintain balanced stock. These improvements are evaluated by measuring metrics that are similar to those used previously. Each one of these improvements implements the previous one, because they depend on each other. In all of the improvements applied to the algorithm, they are implemented with a LCR ordering model 2. Basically, we obtain the different transport and stock costs and compare them in order to have a general view of the impact of the improvement on the solution of the algorithm (see Fig 3).

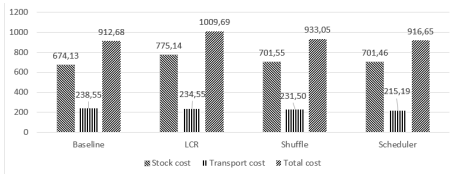
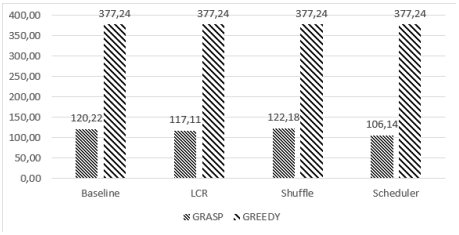
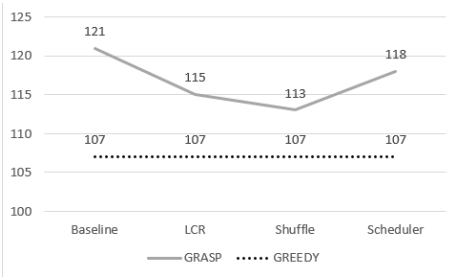


Figure 3: Different costs for each improvements.

As shown, the test results do not show significant differences in costs; only the implementation of all of the improvements slightly reduces the total cost. To ensure that the improvements used optimize the solution, we obtain metrics based on stock balancing (see Fig 4).



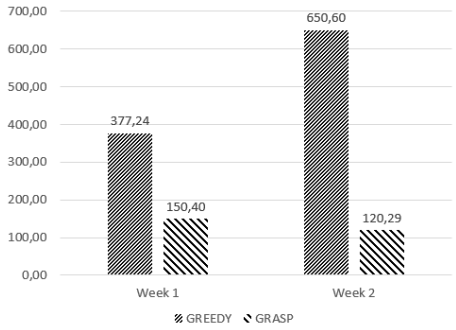
(a) Average number of items with negative stock for each improvement.



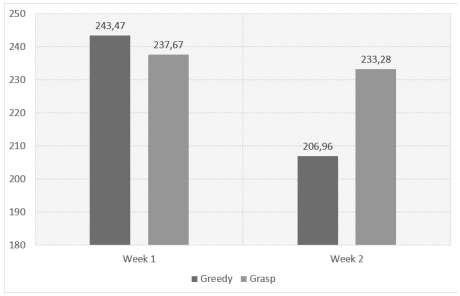
(b) Amount of items with negative stock for each improvement

Figure 4: Metrics based on stock balancing

The results obtained from the metrics for the evaluation of the balance of the stock show us that the average stock of the items in the warehouses remains stable with a difference of between 67% and 71%, while the number of items with negative stock increases with respect to the algorithm already implemented. This is a good result. If we compare the average and the amount negative stock of items at warehouses, we determine that there are more items with negative stock but the value of these is much lower. Therefore, the negative stock is balanced with the new improvements. Finally, the two algorithms are compared completely, using the cost and stock balancing metrics used previously. First, the total cost is evaluated for the first two weeks of July 2020 (see Fig 5b). The results indicate that the algorithm improves the results of the first week but worsens the second week. This is because GRASP was developed to improve the balance at the expense of the transport cost. Second, the stock metrics are evaluated to determine whether or not the algorithm balances the stocks correctly.



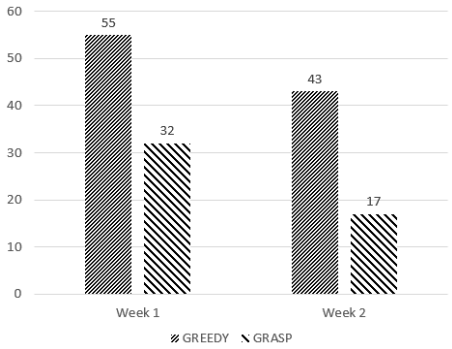
(a) Average number of negative stock items.



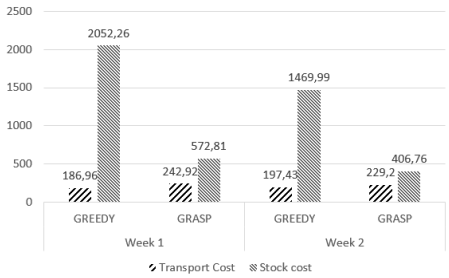
(b) Total cost comparison.

Figure 5: Metrics in the first half of July 2020.

First, a metric is shown that calculates the average negative stock amount for all of the warehouses (see Graph 5a). This metric is very important because it allows us to see how the stock is distributed in the warehouses. The improvement provided by the GRASP in the average negative stock quantity is between 60% and 81%. On the other hand the number of items with negative stock (see Graph 6a) is also a metric that helps us to measure the dispersion of stock among the warehouses. In this case, you can see an improvement by the GRASP of between 41% and 60% based on data from the greedy algorithm. Finally, the costs of the two weeks are obtained to evaluate them jointly (see Graph 6b)



(a) Quantity of negative stock items in the first half of July 2020



(b) Stock and transport cost comparison between the greedy algorithm and GRASP

Figure 6: Metrics in the first half of July 2020.

The results show a large difference in the stock cost between the different algorithms for the two weeks shown. It can also be observed that even through the transport cost is higher in GRASP. The penalty is relatively low compared to the improvement in the stock cost. The stock cost improves by about 72% and the transport cost worsens by between 13% and 21%. The computational time taken by the greedy algorithm is about 20 to 30 minutes, while GRASP takes between 0.45 and 1.5 minutes.

Conclusions and future work

Transport companies need to optimize their logistics infrastructures and strategies in order to be more efficient in a more competitive world. This work tries to merge two different problems, the warehouse stock management problem and the routing problem in order to minimize both the negative stock and the transport cost. To do this, a GRASP-based metaheuristic has been developed to improve the greedy algorithm that is currently being used by the company. The results in several case studies show that the greedy algorithm had a better behavior in the transport cost, since it is specially guided by a heuristic. However, the proposed GRASP algorithm overcomes the results obtained by the greedy algorithm in stock balancing, improving the negative average stock by up to 82%.

In future works, we will improve the proposed algorithm by combining the proposed GRASP algorithm with a genetic algorithm (GA). Thus, the solutions obtained by the GRASP algorithm will participate as a subset of the initial population for the genetic algorithm. This could improve the quality of the solutions due to the high capability of the GA to combine previous solutions and avoid blockage in the local optimal.

Acknowledgements

The paper has been partially supported by the Spanish research project TIN2016-80856-R.

References