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 problem specification proposed by the company with the intention of providing a better comprehension of the presented algorithm.

- **Transport cost** (TC): this is the cost associated with an order. This cost is based on the distance between the starting point and the warehouse, which is proportional to the distance. This is taken into account when calculating the final cost in the fitness function.
- **Stock cost** (SC): If an order composed of N items to be delivered is loaded at a warehouse where the current stock is less than the one required, the remaining negative stock quantity must be replaced. Therefore, the cost of the stock will be proportional to the negative stock multiplied by the cost of replacing each of the items.
- **Delay of delivery** (DD): Each of the possible warehouses associated with an order are located at a certain distance from the origin. Therefore, depending on the warehouse chosen for loading the order, a different delay will be assigned. This delay affects the amount of stock that is reduced from the warehouse from the planned date to the real delivery date.
- **Single load point** (SLP): Each order has a number of locations to which it can be loaded. Each of these locations has a number of important decision features. An order can only be loaded at one location, with the constraint that all of the items needed in the order must be loaded at the same location. Therefore, it is not allowed to load at different locations.

The problem is composed of a dataset (DS) that is based on the combination of three main elements it contains all of the information necessary to represent a problem instance (see Equation 1). Thus, each problem instance is composed of a set of orders (O) each of which represent a trip to transport items, a set of available warehouses (W) to load the items, 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 of the information about the orders, where N is the number of orders. Each order $o_i \in O$ is composed of three parameters $o_i = [D_{o_i}, Ir_{o_i}, W_{o_i}]$:

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

$$Ir_{o_i} = \{ir_{o_i1}, ir_{o_i2} \ldots, ir_{o_iP}\} \tag{2}$$

- W_{o_i} is a set of possible warehouses where order o_i can load all of the required items (see Equation 3).

$$Ware_{o_i} = \{Ware_{o_i1}, Ware_{o_i1}, \ldots Ware_{o_iQ}\} \tag{3}$$

  Each element W_{o_ij} is composed of by three values:

  - ava_{o_ij} represents the availability of platform j to load all of the items of order o_i. Thus, $ava_{o_ij} = 1$ means the warehouse is available, and $ava_{o_ij} = 0$ means that is not available.
  - pr_{o_ij} contains the transport cost of order o_i from platform j. If $ava_{o_ij} = 0$, this value is null.
  - dl_{o_ij} is the delay to load the order o_i from platform j. If $ava_{o_ij} = 0$, this value is null.

To control the amount of stock at the beginning of each week, the company provides a three-dimensional matrix with the current status of each warehouse W (see Equation 4).

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

where each $wst_i \in W$ is composed of a bidirectional matrix (item x week days). The model of the company, allows negative stock to be generated in the warehouses associated

to an order $wst_{ijk} \in \mathbb{N}$. This is allowed because each of the logistic centers (warehouses) can replace the lack of containers by shipping them.

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

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

Solving techniques

This section explains the current algorithm being used by the company, pointing out the structure and specifications of the implemented greedy algorithm as well as the characteristics that make it up. The proposed GRASP algorithm, local search, and several improvements are detailed below.

The Greedy algorithm

Today, the objective of the company is the planning of weekly orders based on the demand of customers taking into account the available stock expected for each day of the week. The company has developed its own algorithm to obtain an optimized solution based on a greedy technique with a post-processing phase. The stock available in each item model in each day of week is obtained from a weekly load plan. This stock is obtained from the sum of the remaining containers from the previous week, the containers that have been successfully refilled and those obtained by shipments, etc. The greedy algorithm has additional constraints imposed by the perspective of the customer, warehouses, and geographical areas, as well as to other factors:

- A certain number of clients only allow loading in a certain warehouse due to distance and convenience factors.
- The zone of influence or geographical area has a higher priority in the assignment of orders by each warehouse.
- The loading of an order is restricted to the type of items it contains, so there are certain warehouses that do not serve a particular item.
- The warehouses or loading centers have independent delays for each product at different times of the year because the quantity of products (fruit, vegetables, fishes) is not constant throughout the year.

These constraints make the generated search tree too large to be addressed in one go. Therefore, the current algorithm developed by the company is split into several layers, which are put together in a post-processing step.

First, the algorithm performs an ordering according to priorities. It uses a heuristic called closeness centrality that is applied to network theory. This measure allows us to obtain a measure of centrality of a transport network by adding the length of the paths between the different nodes, obtaining the central nodes as those that are closer to the other nodes. Later, the orders are classified according to the item with the greater portion of the order, assigning all if the available stock of the nearest warehouses by this article model.

In the second layer, there are orders that are unassigned to a warehouse because they have failed to meet any of the restrictions of the first layer. These remaining orders are reordered using the centrality closeness heuristics to assign them to less central warehouses with larger stock. The orders that have not been assigned can be renegotiated using the customers in order to improve the response of the warehouse system or to make up for the lack of stock by manufacturing the containers needed to meet the customers' needs. However, after this second layer, there may be unassigned orders which are assigned to the most optimal warehouse in a post processing. These are marked so that an expert human can review them, looking for combinations that improve the cost of global transport without increasing the stock cost to manufacture containers. The constraints that are joined in the greedy algorithm by layers only take into account the cost generated by the transport and the requests of the clients. This makes it necessary for an expert to review all of the trips to be able in order to obtain a better solution for the company.

GRASP

Taking into account the structure of the above greedy method and its drawbacks in to obtaining a good solution, a GRASP algorithm has been developed to obtain a balanced solution in terms of travel cost and stock balance. To do this, we focus on obtaining an efficient solution and keeping the stock as balanced as possible among the warehouses.

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: $t \longleftarrow [1, 2, 3, \ldots, |O|]$
5: **while** empty(t) **do**:
6: **if** sort-criterion(i) **then**:
7: swap(o_{i-1} , o_i)
8: $t \longleftarrow t \setminus i$
9: $i \longleftarrow i + 1$
10: **end if**
11: **end while**
12:
13: $s \longleftarrow [\ \ ]$
14: $LCR \longleftarrow [o_1, \ldots, o_n]$
15: **while** $|s| \neq |LCR|$ **do**:
16: $j \longleftarrow 0$
17: $lcr \longleftarrow [\ \ ]$
18: **while** $j < |LCR|$ **do**:
19: $lcr \longleftarrow lcr \cup$ **LocalSearch**$(LCR_j, \alpha, |LCR|)$
20: $j \longleftarrow j + 1$
21: **end while**
22: $s \longleftarrow s \cup lcr*$
23: $LCR \longleftarrow LCR \setminus lcr*$
24: $LCR \longleftarrow LCR \cup o_{n+1}$
25: **end while**

The proposed Algorithm 1 is based on two steps. First, the list of orders is sorted to improve the response of the algorithm (from lines 5 to 11). This assumption is based on the idea of constrainedness, analyzing the most restrictive orders first in order to give the algorithm more decision power

in the final iterations. Second, the iterative part of GRASP (from line 15 to 25) obtains a list that a fraction of the size of the list of requests (called LCR). This list contains the algorithm's candidates to assign a warehouse. Then, a local search is generated in which the best warehouse is obtained. Once we have the best warehouses for all of the trips, we select the warehouse with the lowest cost, eliminating the order from the LCR list and adding the next one from the sorting list.

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.

This type of ordering is modular for easier testing. For this purpose, a sorting algorithm has been developed based on a criterion (see Line 6). Once all of the values an the list are correctly sorted, a list with these values is returned.

- **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: $Q \longleftarrow [\ \ ]$
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).

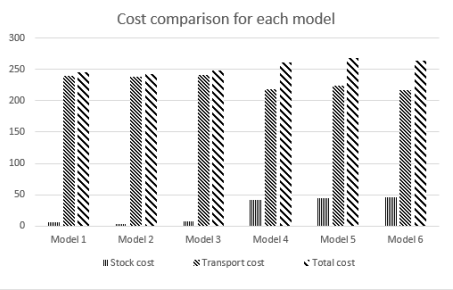
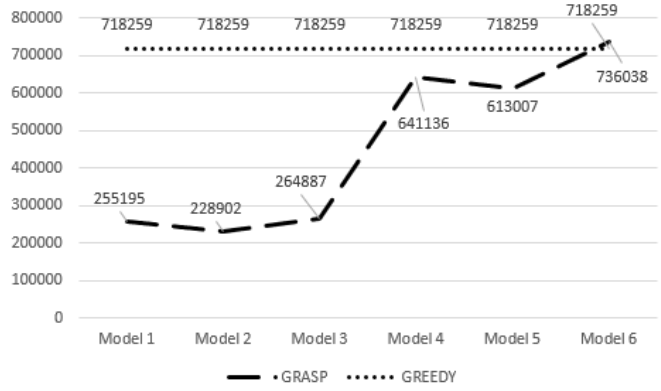


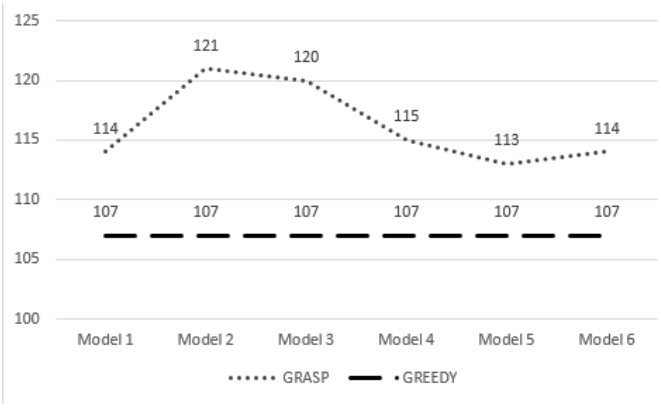
Figure 1: Cost comparison for each model.

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).

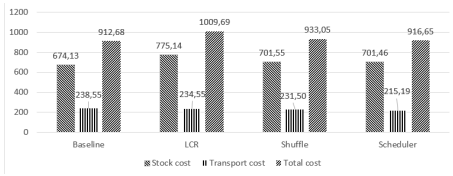
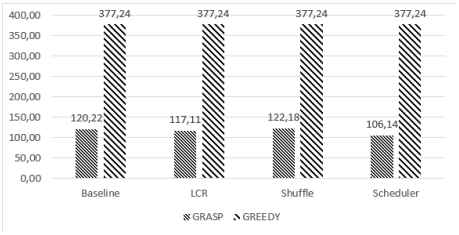
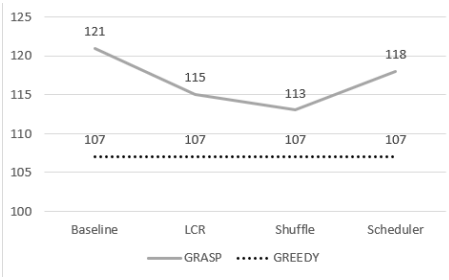


Figure 3: Different costs for each improvements.

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).



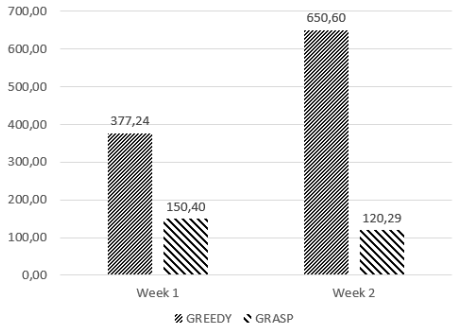
(a) Average number of items with negative stock for each improvement.



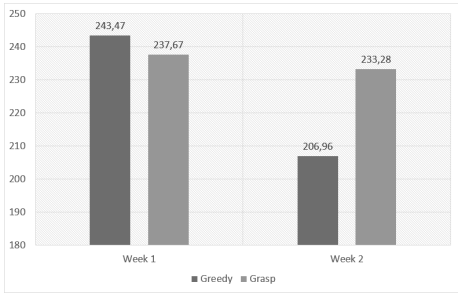
(b) Amount of items with negative stock for each improvement

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.



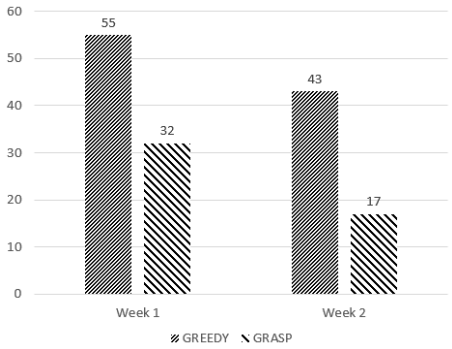
(a) Average number of negative stock items.



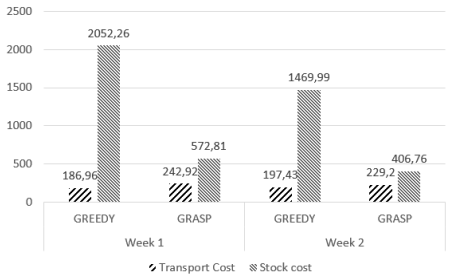
(b) Total cost comparison.

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