FINAL PROJECT OF MASTER IN INDUSTRIAL ENGINEERING

BY

CARLES NIUBÒ BERMEJO

A HEURISTIC APPROACH TO SOLVE DISASSEMBLY LOT SIZING PROBLEMS

JANUARY 2017

Tutor of the project: Lionel Amodeo

Receiving institution: UTT

Sending institution: UPC (ETSEIB)

Abstract

In disassembly lot sizing problems the final objective is organize the schedule for disassembly operations in order to recover parts from an initial product in order to satisfy a given demand. This kind of reverse logistic procedures are becoming more and more important every day mainly as a consequence of environmental issues. When a product has arrived at the end of its life their parts can be recycled or used again but we need to organize the operations required in order to obtain the maxim profit and to respect some important constrains. The one studied is a two levels problem without commonality parts. In this project we propose two different methods to solve this disassembly lot sizing problem. The method 1 is divided in three different stages. During the first one, an initial solution is obtained using a fix-and-optimize heuristic. Since the solution after this algorithm can be feasible or not, an algorithm to fix it is proposed in the second stage. In the last stage we propose a tabo research in order to improve the solution. We have used another method which includes exact resolution with integer relaxation and simulated annealing to improve this initial solution. In the last section we show how both methods perform and we compare them.

Keywords: disassembly lot sizing problem; reverse logistic; fix-and-optimize; tabu search; simulated annealing.

Table of contents

List of figures

List of tables

1. Introduction

In this section we are going to talk about the importance of reverse logistic and being more precise, disassembly operations. First of all, it could be necessary to know what does it means: disassembly can be defined as the process of separating products into parts or subassemblies with necessary inspection and sorting operations (Lee, Kang, Doh, & Park, 2012). Hence, knowing the meaning of disassembly we can start talking about the purpose.

Disassembly managing is becoming every day more important for companies. The main reason is consequence of environmental issues: the short time developing new products, the shortage of dumping sites and new legislations to preserve the environment (Ma, Jun, Kim, & Lee, 2011), since some decades ago many countries have created new laws in order to force the companies to take care of their products during the entire life-cycle. It means that when a product is created it cannot be just managed as trash because it is actually composed of some different components which have been assembled by the company and they must be treated separately if it is required for the legislation of the country. An example is a new legislation in Germany which made the automotive companies to recover and recycle their products upon disposal (Johnson & Wang, 1998). Other countries, for example U.S.A., are doing also an effort to decrease the environmental effects caused by a bad management of the products during their life-cycle.

Knowing the effect of the environmental legislation constrains in companies it is necessary to know which is the relationship between them and disassembly lot sizing problems, the topic of this project. The objective of disassembly operations is to recover all the components, or some of them, of the final product. It has been demonstrated that Material Recovery Opportunities (MRO) (Johnson & Wang, 1998) is not only interesting from materials disposal point of view, it is also useful to recover components in order to use them in other process. Hence, there is not necessity to obtain new materials, process and transport, and it is not even required to assembly them so the environmental effect caused by the production of a new components is removed if companies take advantage of old components.

As a result of the increasing importance in material and product recovery, several researches about disassembly operations have been done this last decades. In the next section we are going to make a quick literature review starting with a brief introduction in lot sizing problems, in order to make easy to understand the different problems studied by the researchers.

2. General problem description

In this section, there is an introduction on this field of study. The vocabulary used in this study is already invented and used in all or almost all the papers about disassembly lot-sizing problems (DLSP). In this type of problems the main objective is to know how many products are going to be disassembled each period to satisfy the demand (or to optimize the procedure) and usually we must know which disassembly operation operations are going to be made in each period over a planning horizon.

DLSP can be divided depending of the suppositions and constrains considered in the study. Here we are going to explain this different assumptions and to show studies for each one. Obviously, many hypothesis can be supposed in only one problem.

First of all, we can classify the problems depending on:

1. Number of products
2. Number of levels of the structure
3. With or without parts commonality
4. Constrains

2.1. Number of products

We can differentiate two groups here: mono-product and multi-products. In the first group all the demand is obtained from a single product while in the second group we need two or more products to satisfy the demand. Hence, there are some components obtained from one product and others components obtained from others products. The structure of each one can be different so as more products we have more difficult it is going to be to solve the problem.

2.2. Number of levels of the products

From now, we are going to define properly all the elements of the problem. A common vocabulary is used in every paper about disassembly problems. In this project, we are going to use the same lexis, those are the definitions:

- Parents: Products from you can obtain items (components) after a disassembly process.
- Children: Items obtained from a parent after a disassembly process.
- Root items: They are the products to be disassembled. Not a single disassembly operation has been done to them before. They are parents but they cannot be children.
- Leaf items: Items obtained when all the possible disassembly operations have been done, they cannot be disassembled further. They are children but they cannot be parents.
- Subassembly: items with parents and children.
- Yield: number of items of each type obtained from a disassembly operation.

Knowing the basic vocabulary used in this field of study we can start defining what “levels of products” means.

The number of levels of products is a characteristic defined by the disassembly bill of material (d-BOM structure) (Ma, Jun, Kim, & Lee, 2011). The structure is always given, it is data required to solve the problem so it is known. The most simplified problem is with only two levels. It means that there are not subassemblies in the structure. We can only find roots which have children but not parents and leaves which have parents but not children. Figure 1 shows an example in order to make easier to understand the problem.

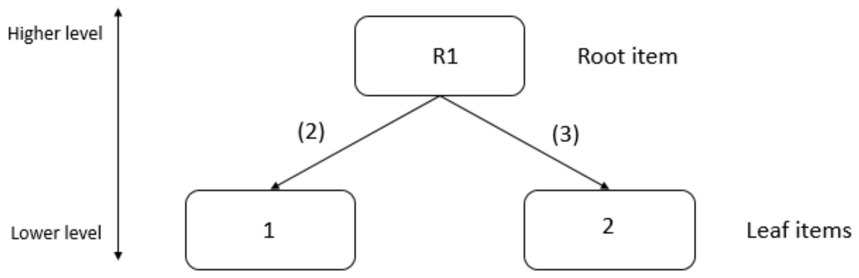

Figure 1: one level d-BOM structure

As we can see in the picture there are only two leaf items (1 and 2) with one parent and without children. On the other hand, there is one root item (1) with two children and without parents. The yield of item 1 from R1 is 2 and the yield of item 2 from R1 is 3. Hence, 2 units of item 1 and 3 units of item 2 are obtained when 1 unit of R1 is disassembled.

This is actually a very simplified two levels structure to make it easy to understand. However, it could has a big amount of children and keep being a two levels structure. The level of the root item is always the higher one and the level of the leaf items the lower one. In two-level product structures the demand is defined only for leaf items.

The second type of d-BOM structure has more than two levels, so it is not actually a structure with a specific number of levels. Figure 2 shows a simplified example.

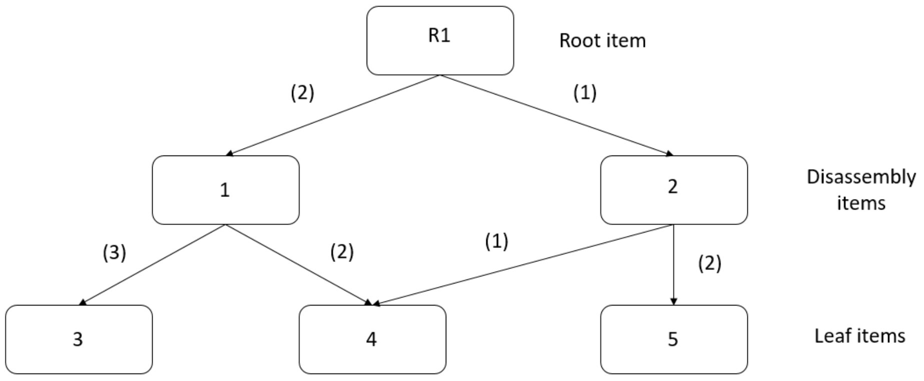

Figure 2: two levels d-BOM structure

This one is a three levels d-BOM structure. The root item is the only one placed at level 1. They children are items 1 and 2 who are parents too. This type of items did not appear in the two levels structure, they are called subassemblies. At the third and last level there are located items 3, 4 and 5. They are the only leaf items in the structure.

This type of structures are more difficult to manage because it is necessary more disassembly operations to get the leaf items. Furthermore, one or more subassembly items can be required for the demand, so products of the same type must be disassembled differently depending on the demand.

The three or more levels structures make the solution complicated. To solve the two levels type it is only necessary know how many roots are going to be disassembled each period. This second type requires to know how many roots and subassembly items are going to be disassembled each period. In that way, a new constriction appears, i.e., perhaps it is not possible to obtain an item even if the root item is disassembled. Hence, it’s necessary at least two operations to obtain a leaf item.

2.3. With or without parts commonality.

Until now all the examples had been without commonality parts. It means that all the subassemblies and leaf items have only one parent, so there is only one path to obtain each child.

On the other hand, commonality parts implies that subassemblies and/or leaf items can be obtained from different parents. It is a good characteristic because there are more than one paths to obtain a product so we can chose the one which reduce the total cost. However, it increase the complexity of disassembly problems significantly because there are many different ways to solves one problem.

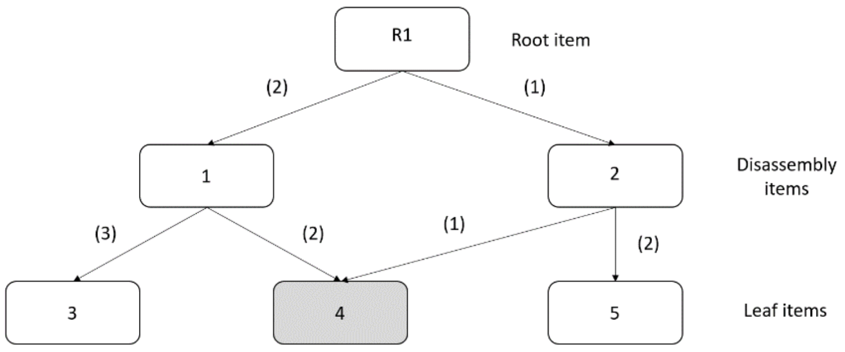

Figure 3: d-BOM structure with commonality parts

In figure 3 there is an example of a d-BOM structure with commonality parts. The item 4 is the common part which can be obtained from two subassembly items, 1 and 2. In this example both parents are in the structure of the same root, i.e., we can obtain item 4 from 2 different items but in any case we need R1. Nevertheless, a common part can be obtained from different roots.

2.4. Constrains

Many constrains can be considered in a DLSP. One of the most commons is the capacity constrain. In this type of problems disassembly quantities are limited in each period by disassembly capacity, which can be different for each one. The capacity available can be defines for many parameters, but the most common is establish a time limit. For example, if the capacity of a period *t* is 20 minutes and a disassembly operation requires 10 minutes it is only possible to disassembly twice. In this project we are going to solve the problem having in mind this constrain.

3. Literature review

Several studies have been made for many researches about this field the last decades. In this section, we are going to analyse some of them in order to know which the state of the art is at this moment. We are going to show how the different problems have been solved.

3.1. Mono-product problems

Taleb and Gupta 1994 (Gupta & Taleb, 1994) made one of the most important research papers within this field. Several authors have used this algorithm in order to find an initial solution for their own problems, for instance Jeon, Kim, Kim and Lee (Jeon, Kim, Kim, & Lee, 2006). Taleb and Gupta presented an algorithm capable to find all of the disassembly operations required to cover the demand of each period. The algorithm works starting from the lower level where there are only leaf items and it analyses the demand of this leaf items parent in order to cover the demand. This levels are solved for each period. Hence, a demand of this items parent is found all periods. Then, the same method is used for other group of brothers with a different parent (called module in the paper) at the same level, if there are no more items at the same level following level is analysed where there previous parents are now the brothers. This process is done until the number of roots required is found. The f shows the sequence followed by the algorithm.

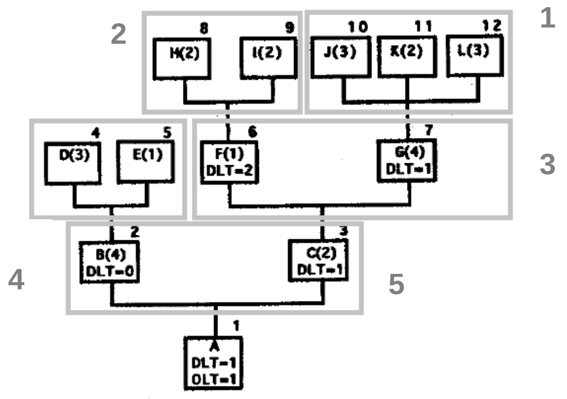

Figure 4: sequence of GT algorithm

One of the peculiarities of this algorithm is that the disassembly operation time is considered and the minimum unit is one period. However, there are no capabilities constrains. In this problem are considered receipts from external sources which are items not obtained from disassembly operations.

The following paper solves the problem without common parts but adding a new capacity constrain. This one is an interesting constrain because the problem becomes more real. As everybody knows all the procedures in a factory are constrained by several factors, in instance, human resources, budget, time, etc. Here, it is considered one of the most important factors. The capacity can be measured with different units such as time or space. The most common unit is time and it is actually the unit of measure used in the following study. This constrain can easily be found in a real factory. If there are two workers in the disassembly department and they are working during 8 hours every day the disassembly time available each day is 16 hours. Hence, if the time required to cover all the demand is higher than 16 hours, the demand cannot be totally covered that day.

Jeon, Kim, Kim, and Lee (Jeon, Kim, Kim, & Lee, 2006) suggested a solution of the problem that we have just talked about. They are keeping in mind the available time in each period and the fact that each disassembly operation consumes part of this time. As in most of the papers, the objective is minimize the cost of the procedure but without overtaking the capacity of each period. In this paper an integer programming model is proposed in order to solve it, where all the costs and constrains are considered. This model can be solved using a commercial software package but it can require a big amount of computational time as the authors specify in the paper. Once more, they suggest a heuristic method in order to find a good and feasible solution with a reasonable computational time.

This method is divided in two stages. During the first one, a feasible solution is found and the second one is used to improve the solution given by the first stage.

The initial solution is obtained using the GT Algorithm (Gupta & Taleb, 1994). As we have explained before this algorithm is not considering any capacity constrain. As a consequence the author adds a modification in order to get a feasible solution. This modification is based on the backward move. That procedure is used by several authors since it is very useful to convert a non-feasible solution to a feasible one. It consist in the movement of one or more disassembly operations of the period t to the period $t-1$ until the overload capacity is eliminated. The total cost is going to increase if it is necessary since the main objective is convert this solution in a feasible one. If the solution obtained from the GT algorithm is feasible no backward movements are required and the solution can be improved during the second stage.

During the second stage both forward and backward moves are considered in order to minimize as much as possible the total cost of the procedure. The solution obtained is also feasible since the number of disassembly operations is the same of the one in the first stage. Forward and backward movement are done simultaneously and the results obtained are significantly better than the ones obtained in the previous stage as the authors show in the computational experiments section of the paper.

3.2. Multi-products problems

From now on, all the studies have been made with multi-products. In the paper (Hrouga, Godichaud, & Amodeo, 2016) the problem is without commonality parts, with two levels structure and with capacity constrains. First of all, the authors present an integer program where the capacity constrains are considered. The function objective is minimize the total cost of the procedure having in mind the holding, disassembly and lost sales cost. This problem could be represented also with a maximizing function objective where the revenues generated by the sales were included. Instead of that, the authors preferred to use a minimizing function which is equally useful.

A heuristic is also explained. The method starts with a solution which can be feasible or not. Then backward moves are applied so disassembly operations can be either moved from one period to the previous one or withdrawn. It should be pointed that in this method is considered not only the overload capacity, the movements are going to be done considering the cost of this movements as well. So if there are more than one options to avoid the overload capacity the cheapest one is going to be used even if it consists in withdrawing some disassembly operations.

When the backward moves have been applied in all the periods (except in period one as we could expect) the forward moves are applied. As a consequence of this two movements, a big amount of disassembly operations can be placed within the last period generating overload capacity. For this reason, a third move is used withdrawing any overload capacity.

A fix and optimise heuristic is proposed to improve the solution given by the previous method. The way the solution is optimized is by fixing the value of all the variables allowing the modification of only one of them. Two versions of fix and optimize are used. The first one is product oriented so one of the roots is fixed while the others can be modified in order to minimize the total cost. This procedure is used with all the roots. In the same way, the period oriented version fix the values of one period and modify the other values if it means a decrease in the total cost. Once again, the procedure is repeated with all periods.

Finally, a genetic algorithm is proposed with different methods to obtain a feasible and optimized solution. For more details go to (Hrouga, Godichaud, & Amodeo, 2016).

The next problems are the most complex since multi-products and more than two levels are suggested.

The problem presented by Kim, Lee, Xirouchakis and Zust (Kim, Lee, Kirouchakis, & Zust) includes a multi-level structure, multi-products, commonality parts but without capacity constraints. Once more, the authors propose an integer program to solve the problem as well as a heuristic. In this case, both methods are used to solve the problem.

First of all, the integer program is solved after the integer constrains are relaxed. It means that the inventory level, number of disassembly operations and the binary variable can be a non-integer number. This relaxation technique transforms an NP-hard problem into a problem easy to solve using commercial linear programming software. But the solution is not feasible since is not possible to make, for example, 1,145 disassembly operations and that is exactly why a modification is required. This modification consists in round-down the solution obtained. The method is explained in section 3 of the paper (Kim, Lee, Kirouchakis, & Zust)[24].

A different way to solve the same problem is proposed by Kang, Doh, Park and Lee (Lee, Kang, Doh, & Park, 2012). They solve the multi-products and multi-levels problem without commonality parts first. Then there is an extension where common items are considered.

The authors propose a model which can be solved with a commercial software package. As this is a NP-Hard problem a heuristic algorithm is proposed too. This heuristic consists in two phases where an initial solution is obtained and then improved. First of all, the initial solution is obtained using the previous algorithm explained in the same paper. As we have explained this first method does not consider the commonality parts so the solution given is actually non feasible or not good enough since it is not satisfying the demand of the items with more than one parent. To obtain the final solution, the common items are analysed in the second phase. During this phase the problem is solved using the same method as in the first phase but only for the common items. For each common item the heuristic is solved for all the parents. Hence, for one common item we have as many solutions as number of parents it has and the solution chosen is the best one. This procedure is used for all the common items.

Adding capacity constraints, the problem is becoming more difficult and several researchers have worked in this problem as well. One example is the study made by Kim, Lee and Xirouchakis (H.-J. Kim, Lee, & Xirouchakis, 2006). They propose an integer program which can be solved by a commercial software but as a lot of time can be required to solve it an algorithm is proposed too. This algorithm as many others consists in two phases: an initial solution and an improvement of it. Once more, a linear-programming relaxation is used to find the first solution. Then the disassembly operation and inventory values are modified in order to satisfy the integer constrains. In this way a feasible solution is obtained as an initial one.

Although the first solution is feasible, the authors propose a second phase where it is improved using a dynamic programming algorithm with look-ahead check. The basic idea is to decompose the problem for each parent item into T sub-problems and solve all of them. To know more about the method go to (H.-J. Kim et al., 2006). The tests made during the paper showed that this heuristic gave very good solutions and very close to the optimal ones.

Another way to solve the same problem is proposed by Melike Kaya (Kaya, 2011) [16] in his thesis. He presents an integer program where the objective is maximize the profit since revenues obtained by sales are considered as well as all the costs. The sign of revenue value is positive and the costs values are negative. As we said before, this function can be written as a minimizing function if all the costs are considered positive and instead of revenues the cost caused by lost sales is included. Many authors have chosen this last way but it is actually not important, both of them are solved likewise.

Knowing the complexity of the problem, the author propose a heuristic solution algorithm. This algorithm divide the problem in T sub-problems being T the total number of periods. During the first step a solution of the relaxed problem is obtain for $t = 1$ using a commercial program. As we have seen before, the relaxation consists in to allow non-integer values as solutions of the variables. This solutions are rounded-down in order to obtain possible large integer values of the variables. During the following step new constrains are added to solve the problem. The rounded-down values are uses as a boundary for the next solution so if the number of disassembly is 5 during the first step this number is going to be the lower boundary. On the other hand, if the value given by step 1 is 0, the value of the final solution is going to be that one. Solving again (with a commercial software) the problem with this new restrictions, we obtain the final solutions for this period. Then the inventory level is recalculated with this new values and the process explained is repeated for all the periods progressively until $t = T$. When the algorithm computes the problem for T, the final solution of the whole problem in given.

This algorithm works actually very good as we can verify in the chapter 4 of his paper. Looking at the diagrams, the deviation from the optimal solution is almost 0 after 1 hour of computing time.

Finally we would like to present a last paper. We found this one interesting because the authors use a heuristic technic that any of the previous author have applied. This method is called "constrained-based simulated annealing" (CBSA) (Prakash, Ceglarek, & Tiwari, 2012). They propose an exact method and an algorithm as well.

The algorithm proposed is based in the simulated annealing (SA) technic as we has just said. This one is a stochastic procedure as the genetic algorithm (GA). The point of use the first one instead of GN is that it requires a lot of computing time if there is a large search, i.e., if the solution has a lot of possible mutations. SA is also a stochastic technique where new values are obtained from an initial solution using a random procedure. It is also a metaheuristic and the objective is find the global minimum. Other heuristics work improving the solution in each iteration. However, this heuristic allows worse solutions in order to escape from local minima. It means that with an unlimited computation time the optimal solution is going to be found. The next pictures show how it works.

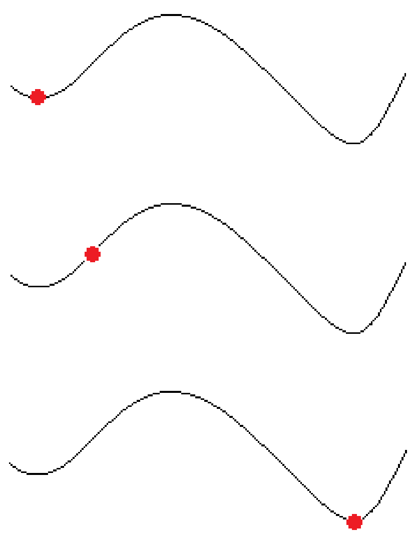
Figure 5: solution evolution

In this picture the solution indicated with a red point is a local minimum.

All the neighbours are a worse solution since the minimizing function is increasing.

As worse solutions have been allowed the algorithm has been able to find the best one after some iterations.

It should be pointed out that we are considering the global minimum as the best solution because this is a minimizing function so as smaller is the value better is the solution. On the other hand, if the objective of the function was maximize the value the best one would be the highest and to escape from local optimal the decrease of the objective function should be allowed.

As I have just explained with SA the uphill moves are allowed. However, the movement has attached a probability. That means that each movement to another neighbourhood is allowed but some have more probability than other since the probabilities are not the same. The value of this probability is defined by two values: the difference between energy function of current state and the previous solution and a parameter known as temperature. The last parameter is going to have a high value at the beginning and the iterative process is going to finish when this value becomes equal to zero.

To know how exactly the authors have performed the algorithm go to (Prakash et al., 2012).

Those papers previously exposed are some examples of disassembly lot sizing problems where the main objective is to obtain an optimized schedule for disassembly operations. It should be noted that in any of this problems have been considered resource capacity constraints, stochastic demands or defective parts.

4. Case of study

Within the disassembly lot sizing problems field there are many different problems depending on the constrictions and assumptions we apply. In this section we present the problem we have worked in and all the methods proposed.

4.1. Assumption

There are some assumptions we have done to simplify the problem:

a) Demands of leaf items are given and deterministic.
b) We do not consider defective parts.
c) The root item can be supplied whenever they are ordered.
d) All disassembly operations are done with no errors.
e) The disassembly operation time is known and deterministic.
f) When an item is disassembled we obtain all the children.
g) Backlogging is not allowed. An unsatisfied demand becomes to lost demand.
h) All disassembly operation times are less than a period.

4.2. Mathematical model

This objective function have been used in several papers before in a similar way. Exactly this one has been obtained from (Hrouga, Godichaud, & Amodeo, 2016).

Index

r	Index for root items, $r = 1,2, \dots, R$
i	Index for leaf items, $i = 1,2, \dots, N$
t	Index for periods, $t = 1,2, \dots, T$

Parameters

s_r	Setup cost of parent item r.
C_t	Capacity available, in time, in period t.
φ_i	Parent of leaf item i.
g_r	Disassembly operation time of root item r.
h_i	Inventory holding cost of item i.
p_i	Lost sales cost of item i.
d_{it}	Demand of item i in period t.
a_{ri}	Number of unit of items i obtained by disassembly of one unit of its parent item r.

I_{i0}	Initial inventory of item i.
M	Large Number.

Variables

Y_{rt}	Binary setup variable of root item r in period t.
X_{rt}	Disassembly quantity of root item r through period t.
I_{it}	Inventory level of leaf item i at the end of period t.
L_{it}	Lost sales for each leaf i period t.

The function objective is formulated as follow.

$$Min \sum_{t=1}^{T}\sum_{r=1}^{R} s_r * Y_{rt} + \sum_{t=1}^{T}\sum_{i=1}^{N}(h_i * I_{it} + p_i * L_{it})$$

Constrains:

$I_{it} = I_{it-1} + L_{it} + a_{\varphi_i i}X_{\varphi_i t} - d_{it}$

For $i = 1,2,\dots,N$ and $t = 1,2,\dots,T$

$X_{rt} \leq M \cdot Y_{rt}$ for $r = 1,2,\dots,R$ and $t = 1,2,\dots,T$

$\sum_{r=1}^{R} g_r \cdot X_{rt} \leq C_t$ for $t = 1,2,\dots,T$

$L_{it} \leq d_{it}$ for $t = 1,2,\dots,T$ and $i = 1,2,\dots,N$

$I_{it,}L_{it,}X_{rt} \geq 0$ for $t = 1,2,\dots,T$, $i = 1,2,\dots,N$ and $r = 1,2,\dots,R$

$Y_{rt} \in \{0,1\}$ for $r = 1,2,\dots,R$ and $t = 1,2,\dots,T$

All the exposed constrains are properly described in the paper (Hrouga, Godichaud, & Amodeo, 2016). Look at section 2.2 for more information.

4.3. First method proposed

First of all, we propose a method based on several heuristics. With the first one we obtain an initial solution which can be feasible or not. With the second algorithm we fix the first solution if it is necessary. Finally, with the third stage we improve the solution.

4.3.1. Initial solution

The objective of the first algorithm is to obtain an initial solution. This one does not need to be feasible since we are relaxing the capacity constrain. If it is not feasible the second part of the procedure is going to fix this problem.

The first option was to use the GT Algorithm. This algorithm was proposed by Gupta and Taleb (Gupta & Taleb, 1994) as we explained before. It has been used by several authors as (Jeon, Kim, Kim, & Lee, 2006) and (Kim, Jeon, Kim, Lee, & Xirouchakis, 2006) in order to obtain an initial solution.

The objective of this algorithm is to cover the whole demand of every period t. It does not keep in mind the total cost but since there is a lost sales cost in the objective function it is obvious that we do not want lost sales and as there is also a holding cost the best option is to obtain the item the same t that it is required. To sum up, this algorithm provides a disassembly schedule for all root items over the planning horizon avoiding lost sales.

As many authors have used it, we tried to obtain an initial solution using this algorithm as well. Actually, this method has a multi-level approach but as two levels structure is a simplification of the first one the algorithm can be used in the second type too.

Eventually, this algorithm was not used in this project. Although the CPU time required was better for all the instances the solution in some of them was quite bad compared to other proposed algorithms. The reason is that the GT Algorithm does not have a costs approach and minimising the cost is actually the objective of the function. When one specific root is disassembled, several items are obtained, i.e., if we want to obtain one type of item we are going to obtain more types even if we do not need them. As a result, the holding cost is going to increase since we have a lot of items that we do not actually need. Therefore, we can conclude that sometimes is better not to cover the demand in order to reduce the total cost.

It has been proposed a fix-and-optimize heuristic in order to obtain this initial solution. This kind of heuristics only modify a group of variables while the others cannot be changed. It starts with an initial bad solution which is going to be improved. This first one consist in zero disassembly operations for all periods and roots. Hence, the total cost obtained at the beginning is going to be very high since the number of lost sales is the highest possible.

As in (Hrouga, Godichaud, & Amodeo, 2016) we propose a heuristic with period approach. In this case, the periods are analysed separately starting with the last one ($t = T$). The disassembly operation chosen is going to be the one which generates the higher decrease in the total cost value (of all the periods $\sum_{t=1}^{T} TC$). When the total cost is not decreasing whichever the root selected is, the next period is going to be analysed ($t = t - 1$). The algorithm ends when all the periods t have been analysed.

As we can see this algorithm uses the total cost as an indicator and try to minimize it with every new solution's modification. When the solution cannot be improved the heuristic stops and a local optimal solution has been reached.

The steps of the algorithm are the following ones:

Step 1. Data input.

Step 2. At the beginning all the elements of the matrix X are 0. The Y, I and L matrix are calculated. The total cost with the initial values is calculated too, now named $old\ TC$.

Step 3. $r = 1$ and $t = T$.

Step 4. The value of $X(r, t)$ is increased in 1 unit and Y, I, L and $TC(r, t)$ are calculated. Then the last unit added is removed. If $r < R$ increase the value of r in one unit and repeat the Step 4, otherwise go to Step 5.

Step 5. If the minimum value of the vector $TC(:, t)$ is lower than the $old\ TC$ add one unit in the position (r', t') where the minimum total cost is located. This unit is immutable and now the minimum cost found has become the TC , then go to Step 6. If all the values of the vector $TC(:, t)$ are higher than $old\ TC$ go to Step 6 without any change.

Step 6. If $t > 1$ decrease in one unit the value of $t, r = 1$ and go to step 4. Otherwise this is the end of the first heuristic.

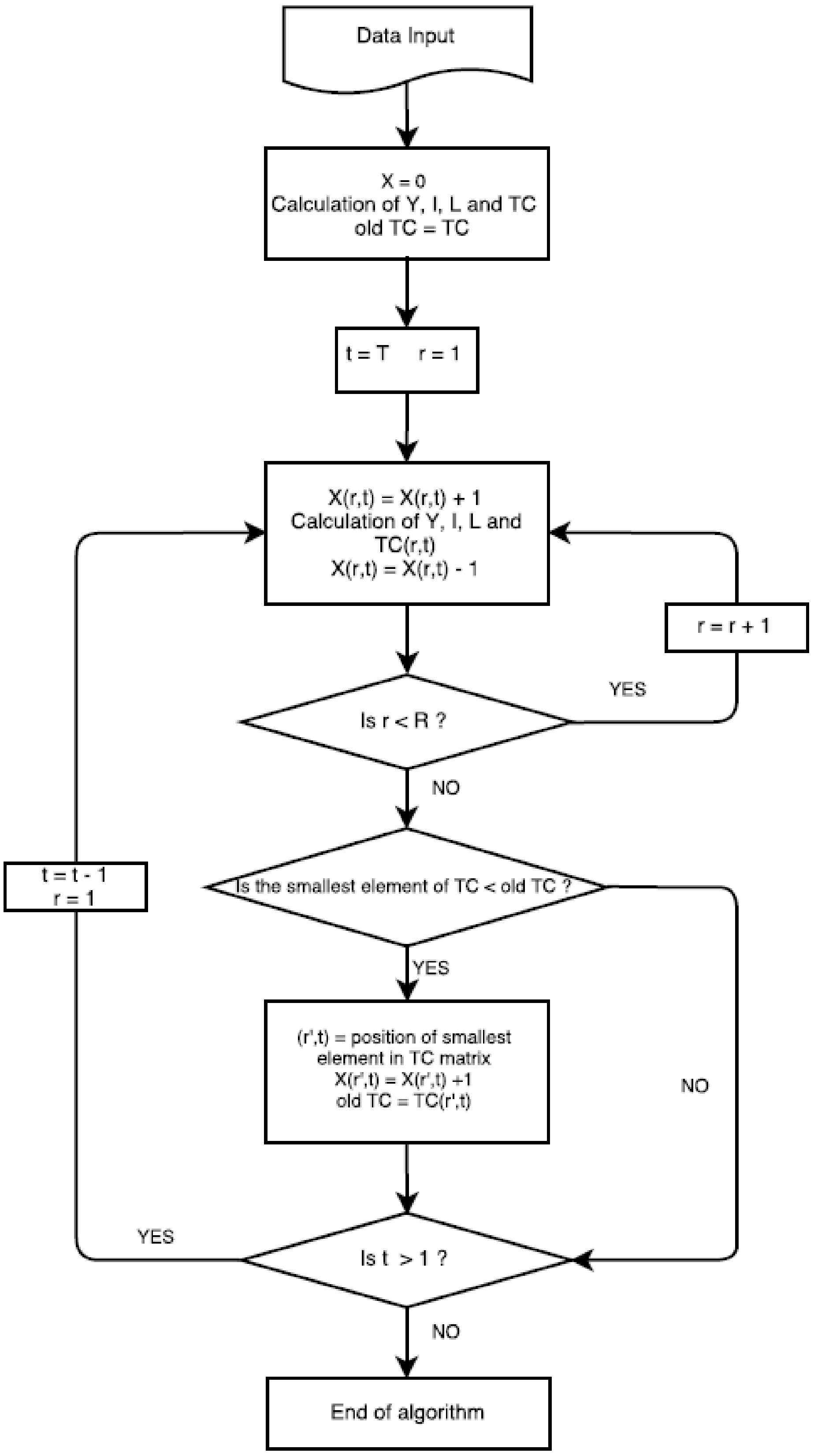

Figure 6: initial solution flowchart

4.3.2. Backward and forward moves

This algorithm is only required if the obtained solution is not feasible. It means that the capacity constrain is not being kept in mind and the total time required for one or more periods is higher than the capacity.

The algorithm works removing or moving to the adjacent previous period the disassembly operations which generate the overload. The algorithm is going to choose the option which reduce the total cost: either to move the operations to the previous period or to remove them. The operations modified are going to be chosen following also the total cost criteria.

The backward moves finish when $t = 1$ since it is impossible to move operations to another previous period. At this point the forward moves start.

During the forward moves procedure the algorithm works alike the previous one but it starts in $t = 1$ and the operations are moved to the adjacent next period instead of the previous one.

Finally, if there is still overload (only possible in the las period $t = T$) some operations are going to be removed until the overload disappears. The method followed is once more to remove the operation which generates the minimum total cost. The algorithm ends when overload disappears.

Backward

Step 1. Data input. Calculate Y, I, L, TC and OC. $t = T$ and $r = 1$, $old\ TC = TC$.

Step 2. Obtain the number of roots which are generating the overcapacity $dX(r,t)$.

Step 3. If $r < R$, $r = r + 1$ and go to Step 2. Otherwise $r = 1$ and go to Step 4.

Step 4. Remove $dX(r,t)$ from $X(r,t)$ and add it to $X(r,t-1)$ and calculate $Y, I, L, TC(1,r)$ and $dC(1,r) = old\ TC - TC$. Then recover old values before this previous action. The number 1 in TC and dC indexes indicates that an interchange has been made.

Step 5. Remove $dX(r,t)$ from $X(r,t)$ and calculate $Y, I, L, TC(1,r)$ and $dC(2,r) = old\ TC - TC$. Then recover old values before this previous action. The number 2 in TC and dC indexes indicates that a withdrawal have been made.

Step 6. If $r < R$, $r = r + 1$ and go to Step 4. Otherwise go to Step 7.

Step 7. The option which generates the highest dC is chosen (x, r'). The second index of dC indicates which root is going to be modified and the first index whether it is going to be moved to the previous period or suppressed. Obtain the new X matrix.

Step 8. Calculate Y, I, L, TC and OC. If $OC(t) > 0$, go to Step 2. Otherwise go to Step 9.

Step 9. If $t > 1, t = t - 1$ go to Step 2. Otherwise go to Step 10.

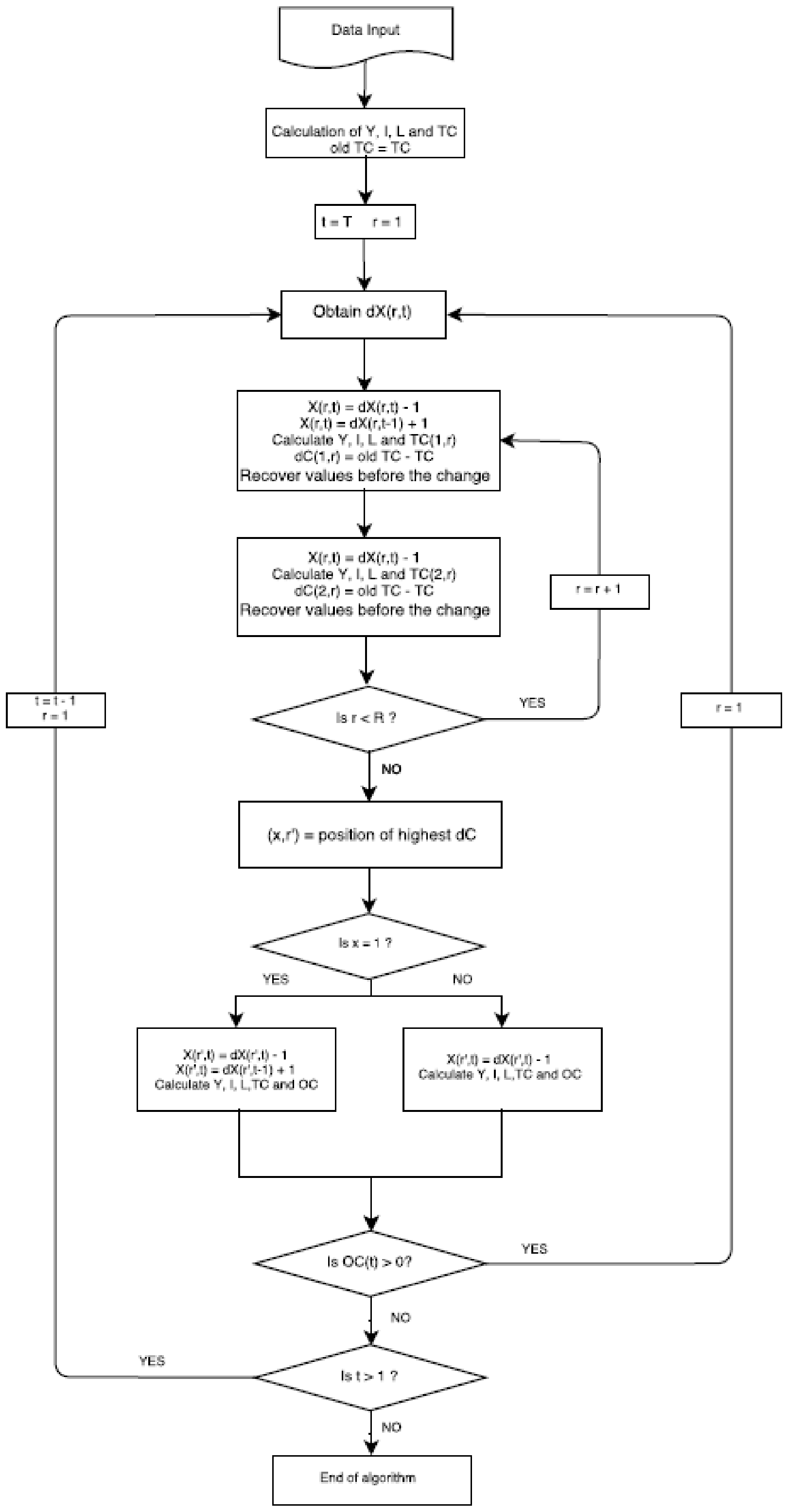

Figure 7: backward moves flowchart

Forward

Step 10. Repeat the same procedure but with forward moves instead of backward moves.

Withdrawal

Step 11. $r = 1$.

Step 12. Obtain the number of roots which are generating the overcapacity $dX(r, T)$. Go to step 13.

Step 13. If $r < R$, $r = r + 1$ and go to Step 12. Otherwise $r = 1$ and go to Step 14.

Step 14. If $X(r, T) > 0$ remove one unit from $X(r, T)$ and calculate $Y, I, L, TC(r, T)$. Then add again the removed unit to $X(r, T)$ and go to step 15. Otherwise do nothing and go to Step 15.

Step 15. If $r < R$, $r = r + 1$ and go to Step 14. Otherwise $r = 1$ and go to Step 16.

Step 16. If $dX(r, T) = 0$, $TC(r, T) = 2 * (highest\ number\ of\ vector\ TC)$ and go to Step 17. Otherwise nothing happens and go to Step 16.

Step 17. If $r < R$, $r = r + 1$ and go to Step 16. Otherwise go to Step 18.

Step 18. Find the minimum TC and remove one unit of r' related with this TC. Calculate again Y, I, L, OC and TC. If $OC > 0$ go to Step 11, otherwise the actual X is the solution. End of algorithm.

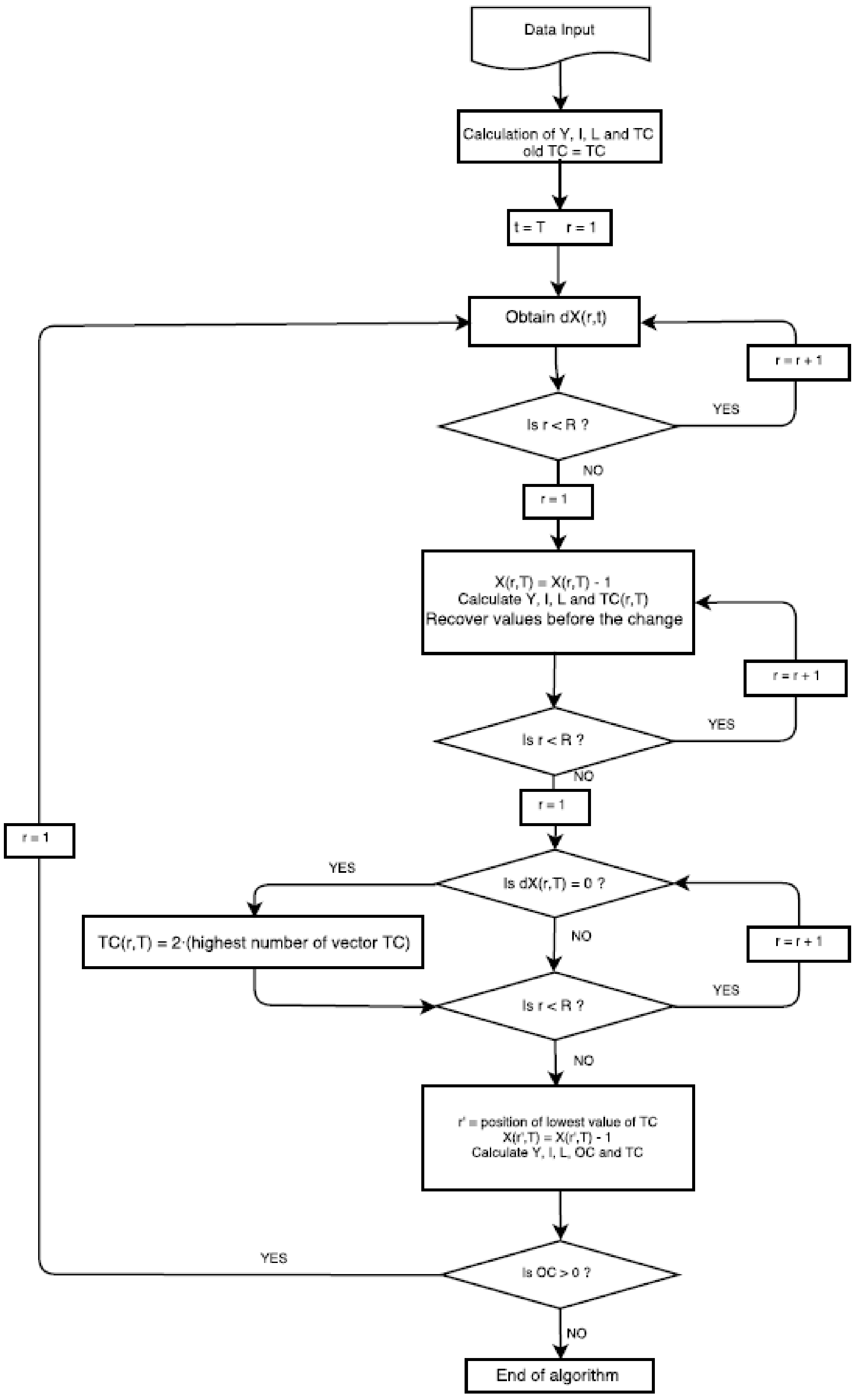

Figure 8: withdrawal flowchart

4.3.3. Improving the solution with a tabu search approach

Finally a metaheuristic with a tabu search approach is proposed to improve the solution.

As we have explained before, the main characteristic of a metaheuristic algorithm is that it is theoretically able to reach the global optimal. The metaheuristic allows the deterioration of the solution in order to escape the local optimal which is a new advantage but there are some disadvantages too. On one hand, the algorithm can start a infinite loop since the old and worse solution can be the new solution and after, the old better solution can be the new solution again. On the other hand as the algorithm can increase and decrease the solution, there is not a well-defined end, i.e., it is necessary to include a stop mechanism. In the algorithm designed the iterations are limited to a specific number. Using the time as a loop controller is a good option as well.

The tabu search algorithm proposed has a stochastic approach. Firstly, one disassembly operation is removed from a specific r and t. This values are selected randomly using a uniform distribution with 1 as a lower boundary and R and T as a higher boundary respectively. Then the removed operation is added to each root and period, being either r or t different from the original place, and the position selected is going to be the one which generates the minimum total cost even if this minimum is higher than the total cost before the movement. Each move is allowed only if the capacity constrain is not violated. This method is called tabu search because it actually have a tabu list. The objective is to avoid repeating old steps. Each change cannot be done again the next three iterations.

Step 1. Data input, $iteration = 0$ and $best\,TC = old\,TC = TC$.

Step 2. If $iteration < IT$ go to step 3, otherwise go to Step 11.

Step 3. Assign a uniform distributed random value between $[1, R]$ to $r1$ and another one between $[1, T]$ to $t1$. If $X(r1, t1) > 0$ remove one unit from $X\,(r1, t1)$, $t2 = 1$, $r2 = 1$ and go to Step 4, otherwise repeat again the Step 3.

Step 4. If $r1 \neq r2$ or $t1 \neq t2$ add one unit to $X(r2, t2)$. Calculate Y, I, L, TC and OC. Calculate also the cost variation between $old\,TC$ and TC, $\Delta TC(r, t)$. Suppress the unit previously added to $X(r2, t2)$. If $r1 = r2$ and $t1 = t2$ go directly to Step 5.

Step 5. If $r2 < R$, increase the value of $r2$ in one unit and go to Step 4, otherwise $r2 = 1$ go to Step 6.

Step 6. If $t2 < T$ increase the value of $t2$ in one unit and go to Step 4, otherwise go to Step 7.

Step 7. (r', t') is the position of the element with the highest value in matrix ΔTC

Step 8. Decrease in one unit the value of $X(r1, t1)$ and increase in one unit the value of $X(r', t')$. If the change do not generate overload capacity, $r1 \neq r'$ or $t1 \neq t'$, and the interchange is not in the tabu list go to Step 9. Otherwise (r', t') is the position of the next highest value and Step 7 is repeated again.

Step 9. Save now $(r1, t1)$ and (r', t') in the tabu list. This list does not allow interchanges between $(r1, t1)$ and (r', t') during 3 iterations.

Step 10. Calculate Y, I, L, TC with the new solution. If $TC < old\ TC$, save the new X matrix in $best\ X$ matrix, $iteration = iteration + 1$. Go to Step 2.

Step 11. $best\ X$ is the final solution. End of algorithm.

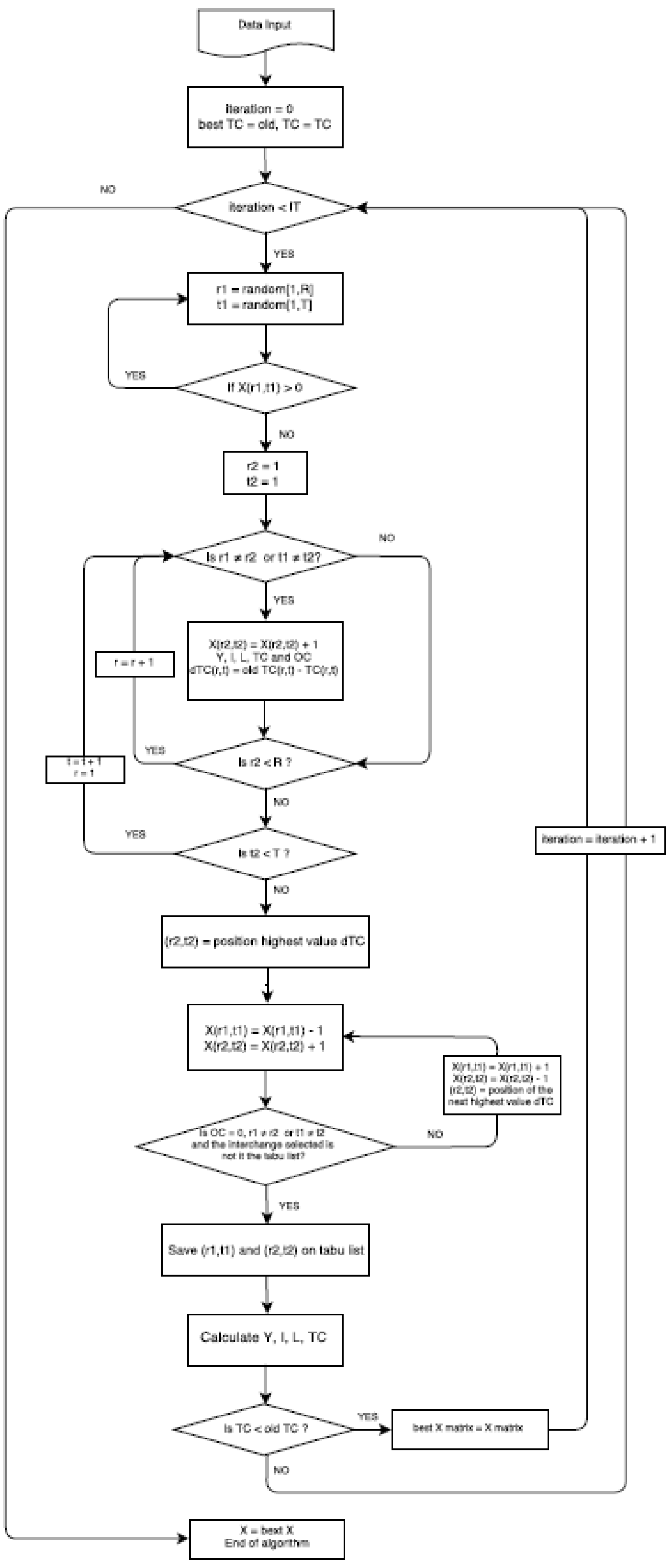

Figure 9: tabu search flowchart

4.4. Second method proposed

After the computational study of the first method another one has been proposed in order to improve the results, especially the computational time.

4.4.1. Initial solution

In the first method almost all the time required to obtain the final solution is spent obtaining a first feasible solution. Knowing that, the main objective of the second method has been to reduce the time required to obtain this initial solution.

One interesting method to obtain an initial solution which has been used by many authors before and provide good results is to obtain a solution from the commercial software relaxing before the integer variables (X_{rt}, I_{it}, L_{it}). As we know, the amount of time needed by commercial software to solve a complex problem can be very high but with this relaxation the CPU time is extremely reduced.

Once we have used the commercial software we have as a result a non-feasible solution since the number disassembly operations, as the inventory level and lost sales, must be integer values. To solve this problem all the values of the X matrix are rounded down and I and L matrix are recalculated again as integer values from this new X matrix. The solution is rounded down instead of rounded up because there is still the capacity constrain and using this procedure we avoid the violation of this constrain.

As we are going to see in the computational study section, the CPU time decrease in all the instances and the objective function as well.

4.4.2. Improving the solution with a simulated annealing approach

In the first method proposed a tabu search approach has been proposed in order to improve the initial solution. In the computational study section we are going to check that although the tabu search is a good tool which helps us to improve the solution but the improvement is little disappointing. As a consequence we have looked for another metaheuristic method called simulated annealing.

Simulated annealing is a probabilistic technic that as the tabu search has a global optimum approach. The name come from annealing in metallurgy where in order to increase the size of its crystals a thermic procedure is applied. From an initial solution a new one is obtained introducing random changes in the previous solution, to determine this new one an energy function is used.

To have a control in the different changes of the solution we use the next expression:

$$u < \exp(-\frac{\delta}{Temp})$$

Where u is a random value obtained from a discrete uniform distribution with values between 0 and 1. The value of δ is a maximum deterioration chosen by us. It is the difference in energies between the two states (before and after the change). The initial value of T which is the variable of control is chosen using the next procedure:

$$\exp\left(-\frac{\delta}{Temp}\right) = 0.99 \rightarrow Temp = -\frac{\delta}{\log(0.99)}$$

As we can see the initial value of T depends on the value of δ. The energy function is equal to 0.99 at the beginning in order to allow us a movement above all the possible feasible solutions so the first value of T is going to be high and is going to decrease in each iteration being more unlikely to accept the new solution in each iteration.

The parameter T is going to decrease following the next equation:

$$Temp_{new} = \alpha \cdot Temp_{old}$$

Where α is a parameter with a value between 0 and 1. As higher is the value of α faster decrease the probability to choose the solution and therefore faster is going to finish the iteration. Hence, it could be better to use a low value allowing more iterations and more possible solutions.

Knowing the parameters of the method and the purpose of them we are going to describe briefly how this algorithm generally works.

First we have the initial solution and we find in the neighbourhood a new one. This new solution is going to be accepted only whether the value of the objective function is better or it is worst but the value of u is lower than the value of the control function. If we are in one of this two states we accept that change and otherwise we do not. Then we update the new T and we start again. The algorithm stops when the value of T is near to 0.

We have added another improvement to this algorithm. In the tabu search every change is the same, although it is randomly the pattern followed consists in the exchange of the assembly operation between two elements of the X matrix. In the simulated annealing method we have performed there are three different possible changes: to exchange one disassembly operation as in tabu search algorithm, to add one operation or to remove one. The option is selected randomly following a uniform distribution. The probability of each option is explained in the computational study section.

The algorithm usually includes a stop mechanism in case the solution is not improving during several iterations. In this project we have included one which stops the algorithm when the solution has not almost improved during several iterations.

In the next lines we describe all the steps of the algorithm applied in this project.

Step 1. Data input. Find the initial value of δ and $Temp$. $old\ X = X$.

Step 2. Assign a uniformly distributed random number between $[1, R]$ to $r1$ and another one between $[1, T]$ to $t1$ and another between [0,1] to $prob$. If $prob \leq p1$ go to step 3, if $p1 < prob \leq p2$ go to step 4 and if $prob > p2$ go to step 5.

Step 3. If $X(r1, t1)$ is a positive number remove one unit and go to step 6, otherwise go to Step 2.

Step 4. Add one unit to $X(r1, t1)$. Calculate OC. If it is higher than zero recover the state before the change and go to Step 2. Otherwise go to Step 6.

Step 5. Assign a uniformly distributed random number between $[1, R]$ to $r2$ and another one between $[1, T]$ to $t2$. If $X(r1, t1)$ is a positive number remove one unit and add it to $X(r2, t2)$ and go to Step 6. If $X(r1, t1)$ is 0 go to Step 2. Calculate OC. If it is higher than zero recover the state before the change and go to Step 2. Otherwise go to Step 6.

Step 6. $new\ X = X$. Calculate $Y, I, L\ and\ TC$. $new\ TC = TC$ and $X = old\ X$. Calculate $Y, I, L\ and\ TC$. $old\ TC = TC$ and $\delta = new\ TC - old\ TC$. If $\delta < 0$, $old\ X = new\ X$ and go to Step 8. Otherwise go to Step 7.

Step 7. Assign a uniformly distributed random number between $[0,1]$ to U. If U is smaller than $exp(-\delta/Temp)$, $old\ X = new\ X$ and go to Step 8. Otherwise go to Step 8.

Step 8. $X = old\ X$. Calculate Y, I, L, TC with the new solution. If $TC < best\ TC$, $best\ TC = TC$ and go to Step 9. Otherwise, go to Step 9.

Step 9. If $\left[\frac{TC(iteration-1) - TC(iteration)}{TC(iteration-1)}\right] < 0{,}00000001$ add 1 to the counter and go to Step 10. Otherwise, do not change the value of the $counter$ and go to Step 10.

Step 10. $Temp = \alpha \cdot Temp$. If $Temp > 1{,}0 * 10^{-30}$ or $counter < 3500$ go to step 2. Otherwise, $best\ TC$ is the solution. End of the algorithm.

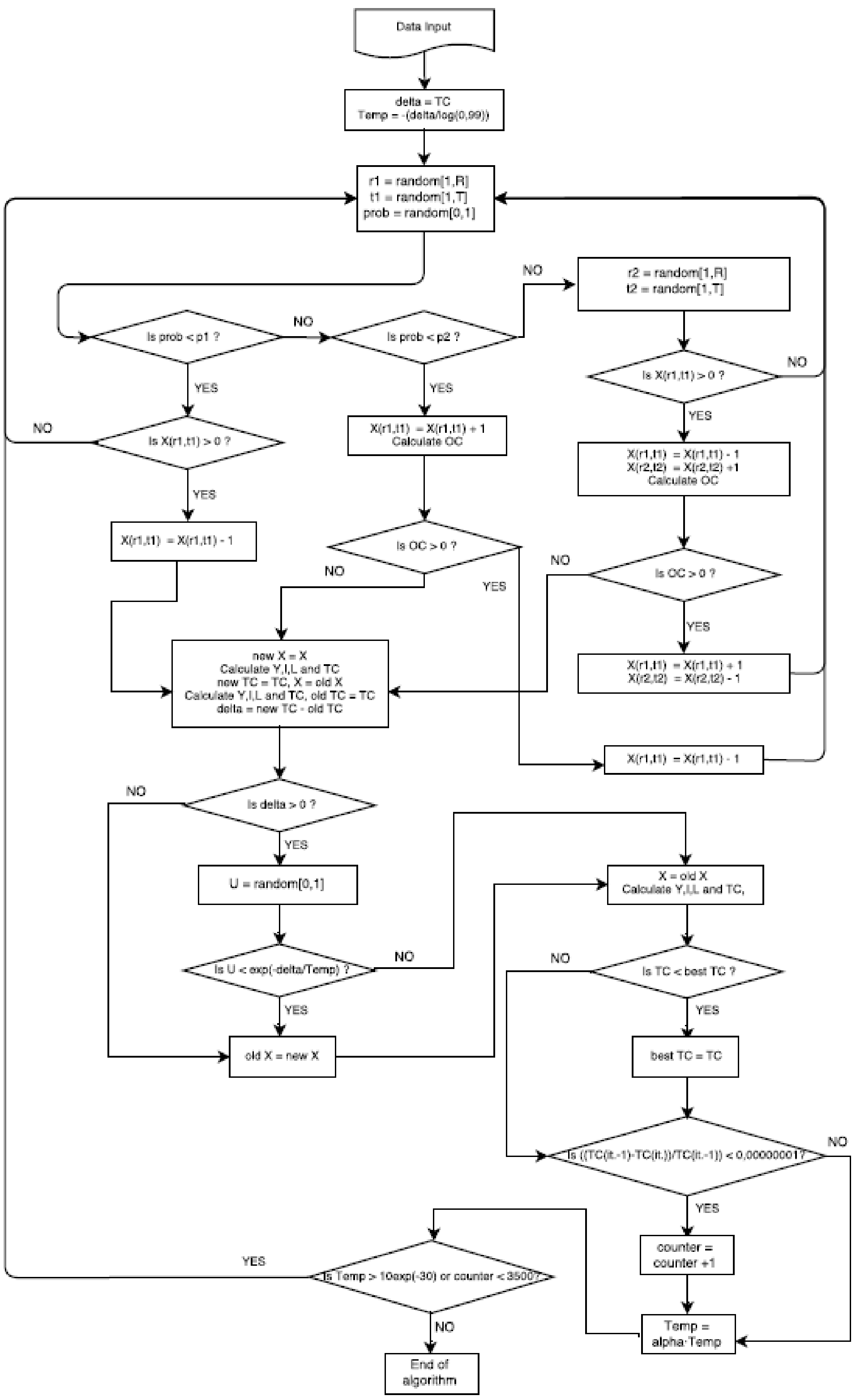

Figure 10: simulated annealing flowchart

5. Computational study

A computational study has been done in order to analyse the performance of the heuristics using several test instances with different problem size. The algorithm has been coded using MATLAB and run on a PC with an Intel Core processor operating at 2.40 GHz clock speed.

5.1. Input data used

The sample size is an important factor in this kind of empirical studies since we are obtaining a conclusion from the output generated, as bigger the sample size is more reliable are going to be them. On the other hand, the simulation time is a problem since many hours are required in some instances to obtain the solution. The CPU time to obtain a solution can vary depending on the clock speed on the computer so all the samples have been run using the same computer. It means that we only have one device to make all the experiments and therefore the time is an important parameter and a limitation as well. We eventually decided that the number of instances for each group of input data is five.

The different groups of instances are clustered depending on three variables: number of periods (20, 30 and 40), number of roots (5, 10 and 20) and number of items per root (5, 10 and 15). As the number of items depends on the number of roots as more roots are in the sample more items are too. Hence, the minimum number of items is 25 and the maximum number of items is 300.

The different values of the parameter of each instance have been selected randomly using a discrete uniform distribution where all the numbers within the range are equally likely. The range $[a, b]$ used is similar and in some cases exactly the same that the ones used by (Hwa-Joong Kim & Xirouchakis, 2010). Actually, the number of instances and the different group of samples have been selected following the structure of this paper. Set up cost was generated from $DU(300, 500)$, capacity available from $DU(800, 1100)$, disassembly operation time from $DU(2, 8)$, holding inventory cost from $DU(5\ 10)$, lost sales cost from $DU(100, 200)$, number of unit of item i obtained from each parent from $DU(2, 10)$, initial inventory from $DU(20, 100)$ and demand from $DU(50, 200)$.

5.2. Solutions obtained using the first method

In the next table we can see the solutions obtained using the whole algorithm, i.e., initial solution algorithm, backward and forward moves and tabu search. The table shows the minimum and maximum values of the results as well as the average since for each different parameters 5 instances have been used.

			Solution			Time(sec)		
T	**R**	**N/R**	**min**	**mean**	**max**	**min**	**mean**	**max**
20	**5**	**5**	1902777,00	2126486,20	2344681,00	61,90	89,21	108,48
		10	4444553,00	5005503,60	5800313,00	106,35	144,82	169,85
		15	6802445,00	7552713,40	8441342,00	189,72	217,77	231,93
	10	**5**	5209783,00	6950051,20	8904135,00	844,76	901,08	955,35
		10	9933998,00	11105599,40	12120312,00	1294,03	1424,45	1565,62
		15	16556183,00	18329571,00	20868105,00	1936,65	2010,15	2139,40
	20	**5**	22114970,00	23639894,40	24942912,00	6686,95	7308,26	7716,21
		10	45481607,00	47350371,80	48645755,00	12615,97	16013,50	24450,92
		15	66637563,00	69396397,60	71935694,00	18542,78	24906,51	38753,74
30	**5**	**5**	3367932,00	3898999,80	4419649,00	172,54	209,12	249,23
		10	7714998,00	8573090,60	8917734,00	304,15	315,47	327,86
		15	12652606,00	12914283,20	13256662,00	257,57	302,79	333,65
	10	**5**	7826786,00	9147203,00	10681099,00	1029,26	1330,46	1641,12
		10	16338812,00	17799876,80	18693962,00	2321,89	2505,30	2796,96
		15	24972006,00	26823943,20	28794339,00	3272,87	4151,12	5571,98
	20	**5**	31506016,00	33420030,60	35696597,00	13162,40	15460,64	17292,78
		10	65194370,00	70270462,80	71922891,00	24095,81	33732,90	44193,05
		15	97720025,00	101056646,20	105695453,00	36170,71	41476,53	48317,20
40	**5**	**5**	4266130,00	5519520,40	6039147,00	258,98	319,50	346,85
		10	11017067,00	12060323,80	14014328,00	429,35	473,62	504,78
		15	16930874,00	17855293,60	18503114,00	397,90	590,97	693,85
	10	**5**	10305101,00	12117855,60	14590626,00	2194,78	2322,16	2451,05
		10	23544027,00	25624244,40	27047940,00	3259,49	3620,67	3843,82
		15	37252282,00	38285408,00	40393493,00	5215,03	5411,81	5599,54
	20	**5**	40597159,00	46071141,40	50646941,00	20156,51	21051,45	22593,03
		10	83627142,00	91056453,40	99415034,00	34769,27	37541,66	40409,50
		15	129193515,00	137345143,40	141957777,00	56315,00	57993,84	58902,48

Table 1: results of first method

In this table we can see the solutions and time required to obtain them but it is not actually very useful. From now on, we are going to use this values and other ones to discover which the behaviour of this algorithm is.

5.2.1. Correlation

With the amount of data obtained it is possible to deduce information which can be useful for samples different from the ones we have used.

There are many parameters which can affect the solutions and CPU time required. Although all the input data has been obtained using the same uniform distribution, the values of the parameters are different among the instances. However, the random values have been obtained from a uniform distribution and this distribution has been the same for all the instances. The variable are definitely the factors which have more influence in the results.

The correlation is any relationship between one or more variables which can be obvious or not so sometimes it is required to analyse the output data to discover correlations which can be impossible to foresee before the experimentation. It is very useful to know how this dependency works since it makes easier to predict future results.

As it is difficult to know how the correlation works from the values in the table 1, we have plotted a heat map with the correlation between all the variables.

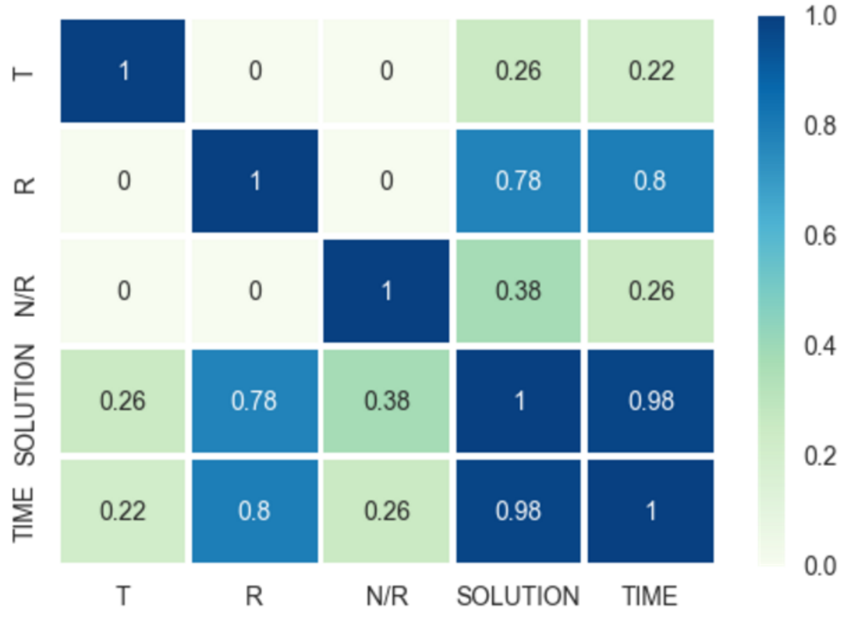

Figure 11: correlations heat map

First of all, we can see an obvious dependency between time and solution. As bigger is the sample, higher is going to be the value of the solution and the time as well.

Secondly we can see that there is no relationship between time-solution and the variables period and number of items per root.

Finally, the most interesting conclusion we can obtain is the dependency between the outputs and the number of roots. There is a direct proportionality among this variables so when the number of roots increases we can assure that the solution and time required are going to increase in the same way.

This conclusion is actually a bit strange. Despite the relationship between outputs and number of roots was easy to foresee it was not very likely not to find a hard correlation between outputs and the other two variables ($T, N/R$). Therefore, it is very interesting to realize and exhaustive analyse of the data since sometimes the algorithm has a difficult to predict behaviour.

5.2.2. Comparing integer programming results and method 1

There are many ways to find a solution for this kind of problems. Usually, the best solution or globally optimal solution, can be found using an optimization software package as CPLEX. The main disadvantage of this commercial programs is the CPU time required to solve big size samples. This is why in this project as in many other before, a heuristic method has been proposed. Usually this last one does not give us the best solution but it allows us to have a good solution (depending on the quality of the algorithm) in a reasonable amount of time.

	TC GAP			Time GAP		
T-R-N	**min**	**mean**	**max**	**min**	**mean**	**max**
20-5-25	5,70	8,20	11,60	521,47	656,11	900,76
20-5-50	4,80	5,90	8,60	342,11	765,77	1127,21
20-5-75	5,00	5,70	6,60	228,65	811,22	1104,06

Table 2: GAP between method 1 and integer programming results

In the previous table are printed the GAP between the CPLEX solutions and the ones obtained from the heuristic. The GAP is the percent deviation of the heuristic solution from the optimal one and to find it the next formula have been used:

$$GAP = \left(\frac{HS - OS}{OS}\right) * 100$$

HS: Heuristic solution

OS: Optimal solution

In order to analyse it the table has been plotted as a bar chart.

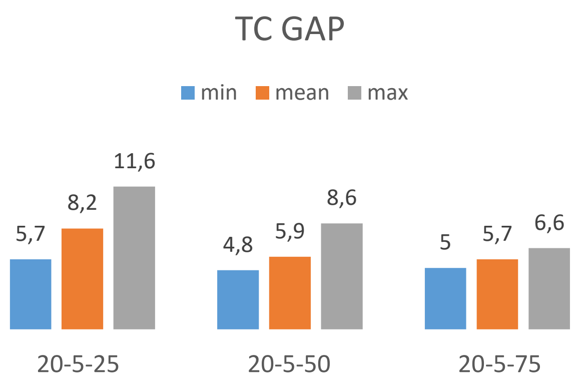

Figure 12: TC GAP between method 1 and integer programming results

This previous plot shows the GAP between the total cost obtained using the commercial software and the total obtained using the heuristic procedure. As we can see, the GAP decrease when the number of items increase, i.e., the performance of the algorithm improve for bigger samples size. This is actually a good characteristic since the commercial software normally requires too much CPU time to solve big size instances. Therefore, we can conclude that for very small size instances the commercial software give us better results and using less CPU time. For big size instances the heuristic is going to provide us good solutions since the GAP tends to decrease when the size of the instances decreases.

The values of the GAP are pretty high if we compare the results with other paper as (Hrouga, Godichaud, & Amodeo, 2016) and (Hwa-Joong Kim & Xirouchakis, 2010). That means that despite the fact the solutions found are close to the optimal solution, the algorithm does not work as good as other algorithms from other researchers.

In the next chart the time GAP data showed in the table is plotted as we made with total cost GAP.

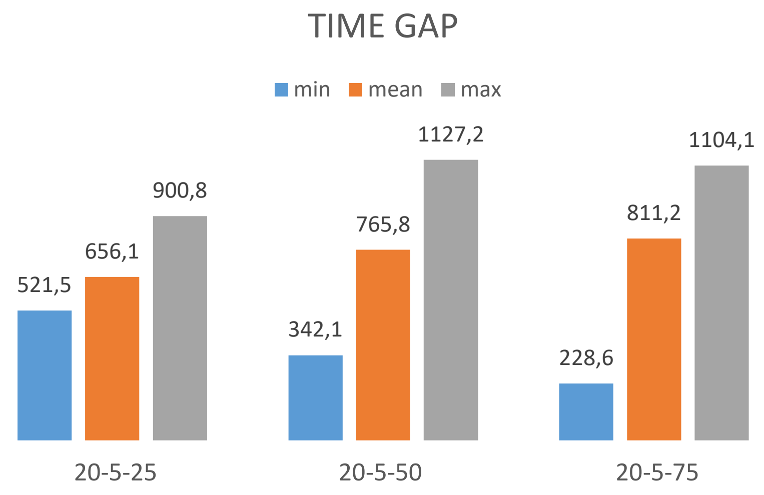

Figure 13: time GAP between method 1 and integer programming results

As we can see, the GAP is increasing proportionally to the size of the sample. That means that as bigger is the size more time is required for the heuristic method in comparison to the integer method.

None of the previous sample have given as a result a GAP equal to zero. Hence, the algorithm has not found the global optimal solution in any case. Since the las part of the algorithm consists on a tabu search where the solution is improved using a specific number of iterations, there is the possibility to get the global optimal solution with a higher number of iterations. In order to know whether it is possible or not, one sample of each size has been simulated until the solution remains invariable, i.e., until the solution converge.

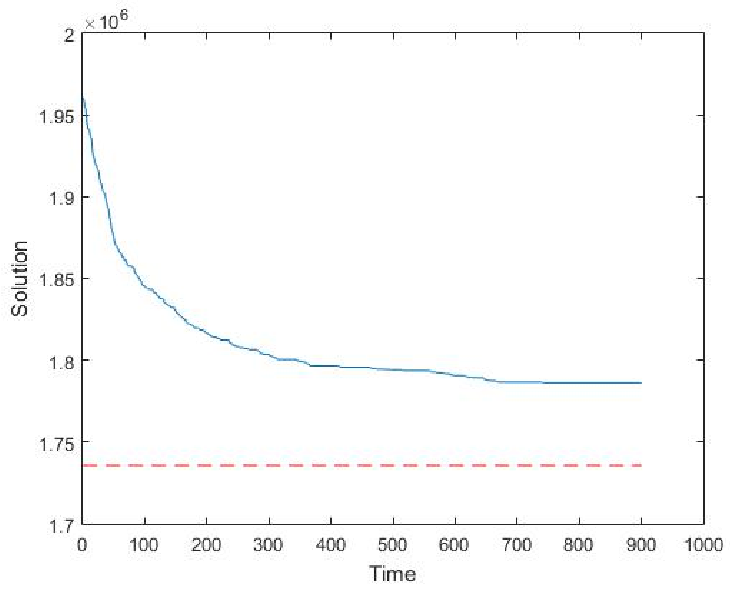

Figure 14: temporal evolution of the solution

In the previous plot we can see the temporal evolution of the solution provided by the algorithm. The red line shows the optimal solution found by the commercial software. In the example exposed the solution converge in some point close to $1{,}77 * 10^6$ while the optimal solution is about $1{,}74 * 10^6$. The time required for the algorithm to converge is about $750\ s$.

Knowing that the solution is not converging in the optimal value provided by the commercial software we can assure that the algorithm is not able to find the best solution even with infinite time. Hence, we conclude that another type of more powerful metaheuristic should be applied in order to improve the results.

5.2.3. Performance of the tabu search algorithm

The first feasible solution is the one obtained after backward and forward moves since this part of the algorithm check if the solution is feasible and it makes the required changes to achieve it. Hence, the comparison is made between the solution after tabu search and the solution after backward and forward moves.

On one hand, the GAP method has been used again to analyse the solution. On the other hand, we have thought that the best way to analyse the CPU time required by the tabu search is compare it with the total time required by the whole algorithm to obtain the last solution.

The method used here for the CPU time is different from the one used in the previous section since in this one we are analysing improvements of the same solution. It means that we need the first solution to obtain the second one. However, in the previous section we have compared the solutions obtained with different methods.

$$\%\,Time = \left(\frac{T_{TS}}{T_{IS} + T_{BF} + T_{TS}}\right) * 100$$

T_{TS}: Tabu search time

T_{IS}: Initial solution time

T_{BF}: Backward and forward moves time solution

			TC GAP			% Time		
T	**R**	**N/R**	**min**	**mean**	**max**	**min**	**mean**	**max**
20	**5**	**5**	1,96	2,79	3,48	26,66	30,18	40,21
		10	1,42	2,14	3,01	26,77	27,21	27,67
		15	1,58	2,23	2,51	21,61	27,05	29,71
	10	**5**	1,60	2,06	2,78	13,40	14,30	15,72
		10	1,01	1,66	2,73	15,34	16,12	16,60
		15	0,69	1,21	1,96	15,03	15,65	16,67
	20	**5**	0,83	0,95	1,10	7,65	8,12	8,61
		10	0,65	0,91	1,15	4,39	7,01	8,17
		15	0,70	0,78	0,85	4,41	11,65	32,13
30	**5**	**5**	1,07	2,08	3,12	24,00	29,12	34,65
		10	1,19	2,03	3,13	28,35	29,64	31,17
		15	1,00	1,57	2,30	26,18	29,55	38,38
	10	**5**	0,60	1,01	1,26	14,61	17,37	18,92
		10	0,69	0,81	0,92	14,89	17,00	18,36
		15	0,65	0,90	1,09	10,49	14,56	17,99
	20	**5**	0,50	0,74	0,95	7,37	8,50	9,24
		10	0,53	0,61	0,68	4,83	6,72	8,82
		15	0,47	0,55	0,63	7,79	11,25	22,53
40	**5**	**5**	0,96	1,96	3,02	27,48	29,87	35,56
		10	1,08	1,98	2,53	29,91	31,30	34,21
		15	1,36	1,88	2,41	25,65	30,50	33,27
	10	**5**	0,93	0,99	1,09	14,67	17,07	18,81
		10	0,55	0,70	0,88	18,00	18,87	19,92
		15	0,47	0,76	1,10	17,33	17,78	18,33
	20	**5**	0,48	0,52	0,57	8,72	9,19	9,50
		10	0,26	0,45	0,54	8,50	9,40	9,99
		15	0,34	0,37	0,42	8,81	9,14	9,69

Table 3: GAP between solutions before and after tabu search

Once more it is difficult to analyse the output data directly from the table so we are going to obtain information from the following plots.

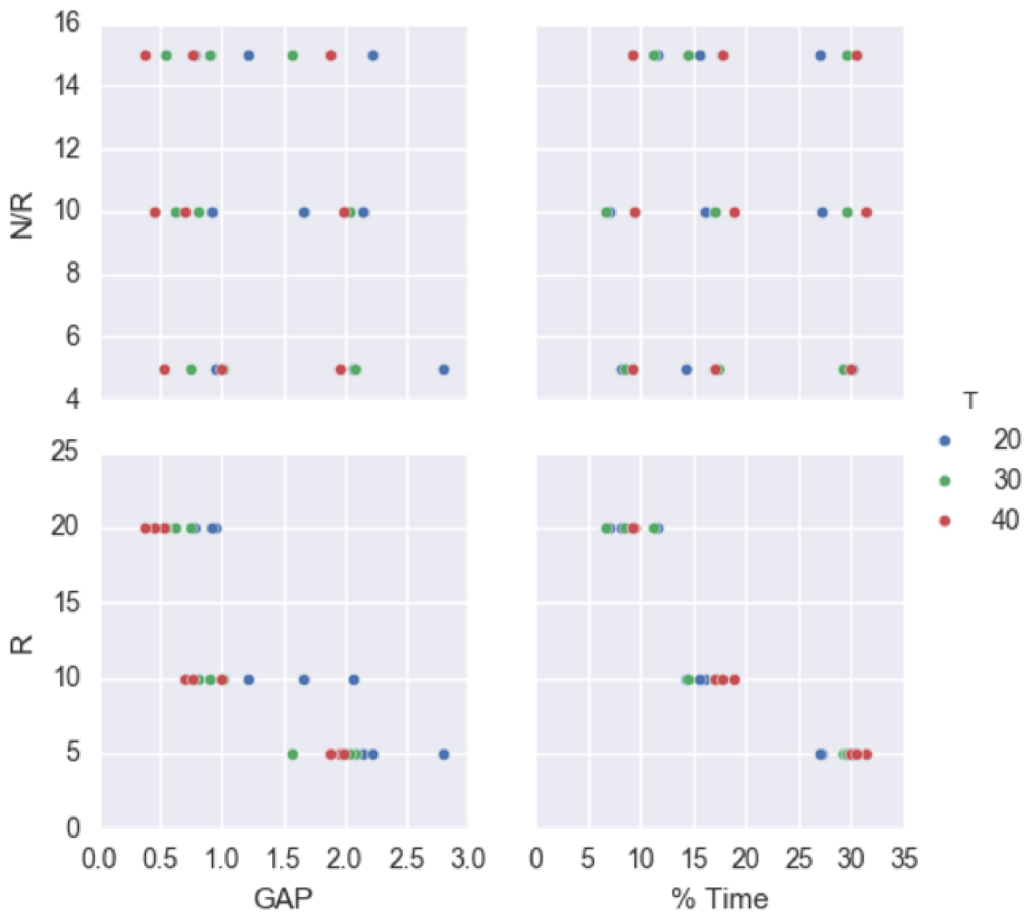

Figure 15: GAP between solutions before and after tabu search

In this scatter graph we can see the behaviour of the algorithm. In the first row the output values of each N/R value are plotted. In the second one, the outputs values are plotted for ach R value. In both rows, the colours of the dots fix the number of periods T.

First of all, in the GAP plots it is easy to see that as higher is the value of R smaller is the GAP, it actually follows a very clear pattern. Within this context, a small GAP is a negative result since it means that the solution has not changed a lot with the tabu search so the improvement has been very poor. On the other hand, we can see that neither the variable N/R nor variable T have a hard correlation with the GAP.

Secondly, in $\%\,Time$ plots we can see a similar behaviour. The GAP has a hard relationship with the R value but it has not with the R/N variable and neither with T. The $\%\,Time$ decrease when the value of R increase. The number of iterations of the tabu search have been set instead of the running time. As the GAP results are low and the $\%\,Time$ is low too, it could be necessary to allow more iterations for a big sizes of R value since as higher it is more possible solutions are and more difficult is to find a better one.

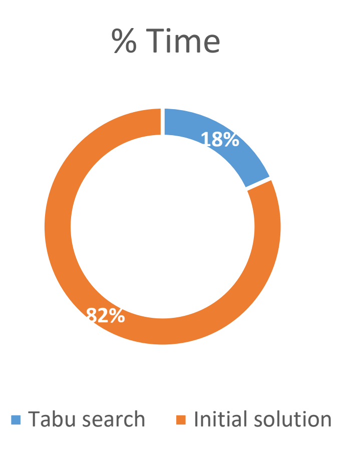

Figure 16: % of computational time

In this graph we can see the $\% \, Time$ average spent before and after the tabu search. The amount of time used to obtain the initial solution is very high knowing that it is not very accurate.

5.3. Solutions obtained using the second method

In the next table we can see the solutions obtained using the simulated annealing algorithm to initial solutions obtained from the commercial software. The table shows the minimum and maximum values of the results as well as the average since for each different parameters 5 instances have been used likewise the first method section.

			Solution			Time(sec)		
T	**R**	**N/R**	**min**	**mean**	**max**	**min**	**mean**	**max**
		5	1723835,00	1992269,20	2229293,00	24,81	36,82	44,93
	5	**10**	4115938,00	4758080,80	5531312,00	47,19	56,70	71,51
		15	6411755,00	7181571,20	8075453,00	82,49	130,96	147,59
		5	4413964,00	5339852,00	6599283,00	102,26	105,18	112,14
20	**10**	**10**	8949386,00	9304069,40	10090343,00	164,53	182,23	197,95
		15	15417161,00	15943130,40	16584734,00	81,61	115,92	229,83
		5	16124154,00	17508532,80	19792654,00	79,36	89,05	95,54
	20	**10**	33912689,00	35456828,80	36761558,00	148,51	201,44	232,65
		15	49060519,00	54316152,40	59083865,00	255,30	285,80	306,98
		5	3150740,00	3650075,20	4219462,00	29,34	37,21	41,09
	5	**10**	7139975,00	8082995,80	8532728,00	36,28	47,41	61,16
		15	11910099,00	12178933,40	12434982,00	90,85	111,19	132,31
30		**5**	6934469,00	7973816,40	8474504,00	46,48	79,10	138,87
	10	**10**	15338925,00	15809083,40	16983944,00	195,29	224,36	245,15
		15	23175699,00	24969522,40	26118746,00	167,90	307,58	370,22
	20	**5**	22430271,00	24429330,20	28132726,00	80,42	104,83	132,51

		10	48633845,00	53665922,40	57379245,00	198,10	202,99	210,63
		15	75583215,00	78893981,60	83551932,00	275,24	298,85	322,34
40	**5**	**5**	3827406,00	5180966,00	5744795,00	46,09	58,88	70,12
		10	10090732,00	11215745,60	13336128,00	51,75	71,83	79,27
		15	15512739,00	16599437,60	17556322,00	62,44	82,60	98,71
	10	**5**	9547721,00	11019958,40	13371875,00	86,92	94,65	106,65
		10	21899373,00	23521760,40	24700167,00	90,36	138,85	194,57
		15	34373781,00	35583848,60	37490435,00	189,19	233,37	262,91
	20	**5**	28233221,00	34128650,00	39210390,00	95,79	136,50	161,74
		10	63881968,00	70472989,60	79310000,00	188,12	202,18	236,14
		15	103540975,00	110419431,20	113956980,00	265,97	325,71	461,72

Table 4: results of second method

5.3.1. Parameters of the Simulated Annealing algorithm

To perform the simulated annealing algorithm we have had to choose the value of some parameters that have not appeared in the previous algorithm. In this section we are going to explain the reason of each choice.

First of all, we have selected a very small and close to zero value as a boundary of $Temp$ within the loop. The chosen value has been $1{,}0 * 10^{-30}$. This value allows to the algorithm to iterate for a long in order to find a very accurate solution. The problem is that in some cases the solution converge and the algorithm spend too much time obtaining very little improvements. To control this situation a stop mechanism has been included. It evaluates the variation in the solution of the last two ones following the next system:

$$if\ abs\left(\frac{solution(t-1) - solution(t)}{solution(t-1)}\right) < 1{,}0 * 10^{-08} \ => counter = counter + 1$$

When the difference is very small the solution has not change too much and when it happens a many times means that the solution is converging and it is not going to improve enough, therefore the algorithm must finish the iterations. The number of $counter$ allowed is 3500. Over this value the algorithm is going to stop. To select this specific value several tests with different values and different instances have been made. This value is high enough to be sure that the solution is not going to improve too much but also we can guarantee that it is very close to the best solution that we are able to obtain using this algorithm.

Another important parameter in Simulated Annealing is α. As we have explained in the section 4.4.2, as smaller is this value faster is going to end the iteration. In order to analyse a big amount of neighbour solutions it has been selected 0,99 as a value of the α parameter.

As we explained before, three possible changes have been considered in each iteration. Each option has attached a probability and to choose them, different instances have been tested with three groups of probabilities. In the next table we can see the results:

	OPTION 1		OPTION 2		OPTION 3	
	TC	Time(sec)	TC	Time(sec)	TC	Time(sec)
Test 1	16421542,00	141,52	16626429,00	138,39	16420035,00	142,79
Test 2	16741480,00	158,51	17017061,00	138,58	16722522,00	179,66
Test 3	17152847,00	143,87	17301592,00	136,56	17095536,00	140,29
Test 4	15380147,00	143,08	15626398,00	136,87	15391118,00	139,56
Test 5	16970969,00	142,55	17096223,00	151,13	16978033,00	132,37
Mean	16533397,00	145,91	16733540,60	140,31	16521448,80	146,93

Table 5: results with each option

And in the next table we can see how the probability is distributed in each option:

	OPTION 1	OPTION 2	OPTION 3
Remove operation	25%	50%	25%
Exchange operation	25%	25%	50%
Add operation	50%	25%	25%

Table 6: probabilities distribution

The instances selected are all of the same sample size since there is no reason to think that the result depends on the size.

The option which give us a minimum total cost is the option three and it has been the one chosen. Hence, it is more likely that and exchange operation happens than the other two options.

5.3.2. Comparing integer programming results and method 2

Once more, as we showed in the method 1 section, the total cost and time GAP are attached in the next table.

	TC GAP			TIME GAP		
T-R-N	min	mean	max	min	mean	max
20-5-25	1,08	1,17	1,41	207,19	254,72	335,31
20-5-50	0,49	0,62	0,79	133,56	263,14	404,26
20-5-75	0,45	0,50	0,60	100,52	488,85	779,66

Table 7: GAP between integer programming and method 2 results

Comparing this results with the ones obtained using the first method we can observe a great improvement. The values shown here are eight or more times lower than the other ones so even not being able to reach the global optimal, this method achieve very close results.

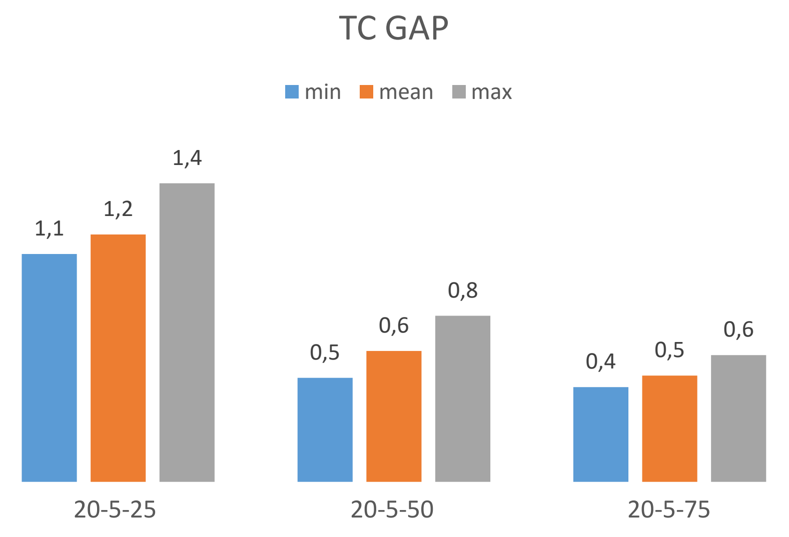

Figure 17: total cost GAP between integer programming and method 2 results

In this chart we can see the total cost GAP plotted. The behaviour of this method looks similar than the first one since the GAP decrease when the size of the sample increase which makes us think that it is going to improve with higher samples as well. The difference that we can notice besides the improvement in the results is the fact that the results are more uniform. It means that the performance of the algorithm is more or less the same no matter what sample are you testing. This stable behaviour makes easier to predict how good are going to be the results with any sample.

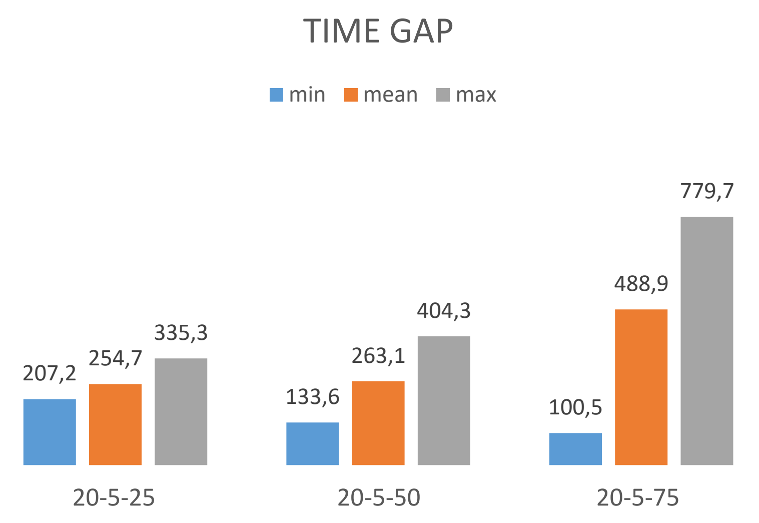

Figure 18: time GAP between integer programming and method 2 results

The time GAP chart does not show a big difference between the one obtained from the first method. The values are much lower in every case and the uniformity of the results tends to disappear when the size of the sample increases.

5.3.3. Comparing first method and second method

Looking the performance of the first method we decided to use another one since the results were a bit disappointing. The next table show as the GAP obtained between them.

			TC GAP			Time GAP		
T	**R**	**N/R**	**min**	**mean**	**max**	**min**	**mean**	**max**
20	**5**	**5**	10,38	6,74	5,18	149,51	142,30	141,43
		10	7,98	5,20	4,86	125,37	155,44	137,51
		15	6,09	5,17	4,53	130,00	66,29	57,14
	10	**5**	18,03	30,15	34,93	726,08	756,67	751,96
		10	11,00	19,36	20,12	686,51	681,67	690,91
		15	7,39	14,97	25,83	2273,06	1634,02	830,84
	20	**5**	37,15	35,02	26,02	8325,92	8106,46	7976,67
		10	34,11	33,54	32,33	8395,13	7849,34	10409,73
		15	35,83	27,76	21,75	7163,26	8614,62	12524,15
30	**5**	**5**	6,89	6,82	4,74	488,15	461,96	506,62
		10	8,05	6,06	4,51	738,29	565,40	436,10
		15	6,23	6,04	6,61	183,52	172,32	152,18
	10	**5**	12,87	14,72	26,04	2114,62	1582,03	1081,73
		10	6,52	12,59	10,07	1088,91	1016,63	1040,93
		15	7,75	7,43	10,24	1849,24	1249,59	1405,03

	20	**5**	40,46	36,80	26,89	16267,93	14648,85	12950,35
		10	34,05	30,94	25,35	12063,47	16518,36	20881,60
		15	29,29	28,09	26,50	13041,45	13778,69	14889,54
40	**5**	**5**	11,46	6,53	5,12	461,89	442,62	394,66
		10	9,18	7,53	5,09	729,61	559,32	536,80
		15	9,14	7,57	5,39	537,28	615,48	602,93
	10	**5**	7,93	9,96	9,11	2425,10	2353,51	2198,31
		10	7,51	8,94	9,51	3507,36	2507,59	1875,57
		15	8,37	7,59	7,74	2656,50	2218,99	2029,84
	20	**5**	43,79	34,99	29,17	20943,39	15322,14	13869,16
		10	30,91	29,21	25,35	18382,68	18468,27	17012,17
		15	24,78	24,38	24,57	21073,66	17705,15	12657,08

Table 8: GAP between first and second method

In the table we can see how the solution and computational time from the second method have improved in relation to the first method (in percentage). For example, the average of the total cost for $T = 20, R = 5$ and $N/R = 5$ is 6,74% lower using the second method than using the first one and the average of the computational time is 142,30% better using the second method as well.

It is easy to see that the total cost has decreased significantly but the best improvement is the reduction of computational time in all the samples.

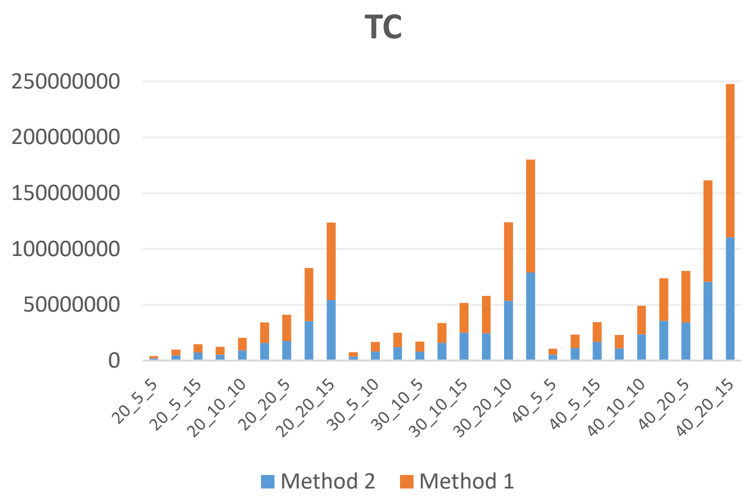

Figure 19: total cost obtained in first and second method

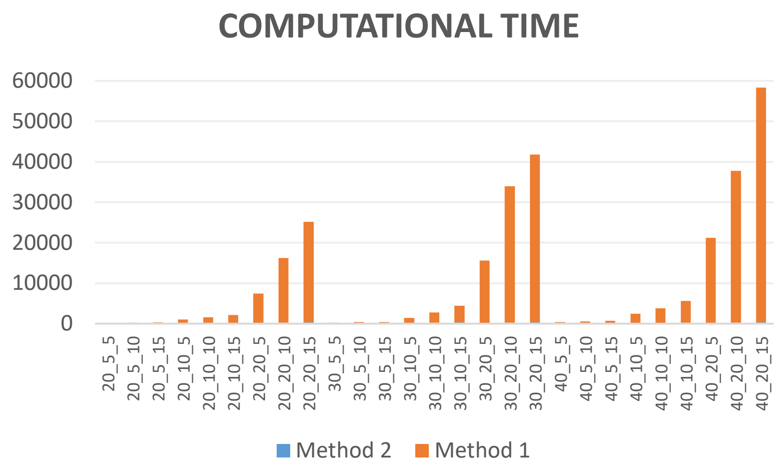

Figure 20: computational time obtained in first and second method

In the first chart the total cot obtained using both methods is plotted and in the second one the computational time required.

The total cost has decreased in all the samples but observing the second chart we can realize how much better is the second method. The time is so higher in the first method that it is impossible to see the time from the second one. We have to zoom in to notice that method 2 is actually plotted.

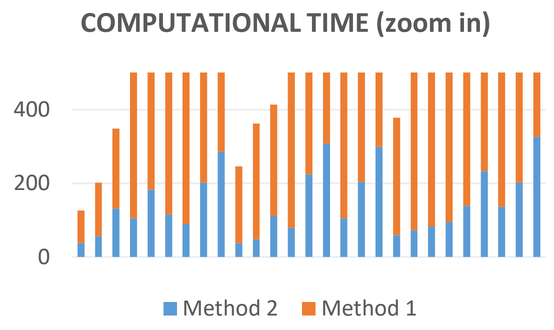

Figure 21: computational time obtained in first and second method (zoom in)

5.3.4. Performance of simulated annealing algorithm

In this section we are comparing the solution given by CPLEX (which has been rounded down) and the solution obtained after apply the simulated annealing algorithm to the first one. The next table shows the TC GAP and the proportion of time used by the simulated annealing algorithm, in the same way we have done with the tabu search method.

			TC GAP			%Time		
T	**R**	**N/R**	**min**	**mean**	**max**	**min**	**mean**	**max**
20	**5**	5	1,95	1,35	0,46	82,02	86,23	87,94
		10	1,12	0,78	0,47	88,58	90,13	91,78
		15	0,70	0,58	0,43	93,26	95,68	96,06
	10	5	1,04	1,57	1,20	92,92	92,71	92,84
		10	1,16	1,24	1,07	96,06	96,29	96,35
		15	0,64	0,92	0,62	91,47	93,84	96,80
	20	5	2,35	1,59	1,03	91,80	91,47	90,65
		10	2,06	1,94	2,12	94,88	96,13	96,47
		15	2,13	1,93	1,69	96,58	96,76	96,75
30	**5**	5	2,11	1,66	0,47	82,44	85,45	86,44
		10	1,88	1,31	0,77	84,57	87,76	90,25
		15	0,34	0,57	0,88	93,33	94,28	94,63
	10	5	2,07	1,79	1,71	86,40	90,43	93,81
		10	1,10	1,36	1,26	96,37	96,72	96,95
		15	0,74	0,92	1,27	95,33	97,35	97,71
	20	5	2,05	1,90	1,52	90,46	92,41	93,65
		10	2,29	1,92	1,63	95,41	95,26	95,23
		15	2,13	1,88	1,84	96,21	96,11	96,14
40	**5**	5	1,51	1,50	1,45	88,13	90,39	91,40
		10	1,22	1,25	0,82	87,58	90,60	90,56
		15	1,03	0,86	0,91	89,08	91,48	92,62
	10	5	2,62	1,83	1,30	92,76	92,38	92,02
		10	1,30	1,59	1,40	91,23	94,17	95,70
		15	1,10	1,14	1,03	95,33	95,93	96,11
	20	5	2,43	1,82	1,68	90,26	91,73	91,87
		10	2,19	2,26	1,87	94,32	94,39	94,88
		15	1,91	2,12	2,04	95,30	96,04	97,11

Table 9: GAP between solutions before and after simulated annealing

As we made before, we are going to graph the results to make easier to analyse them.

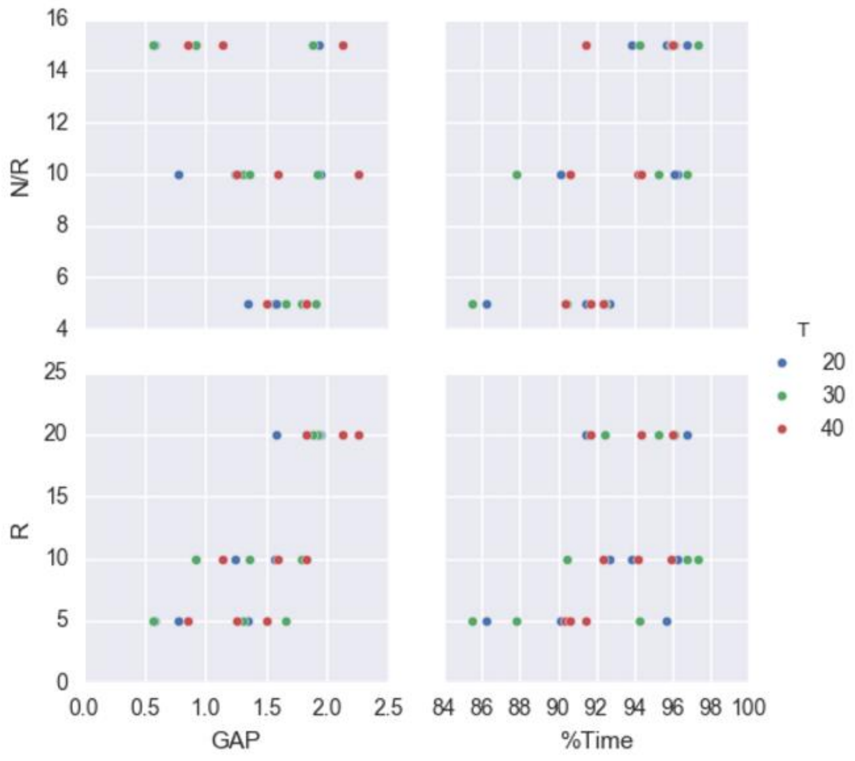

Figure 22: GAP between solutions before and after simulated annealing

It is not easy to obtain information from this graph since there is not any very well defined pattern. The simulated annealing algorithm shows a different behaviour than the tabu search method. The GAP increase when the R grows too. It means that as higher is the number of roots of the sample more significant is the effect of the simulated annealing algorithm. As we can expect, the amount of time required increase as well but not in the same magnitude. The number of periods keeps not having any effect neither in GAP nor in $\%Time$ while the variable N/R increase the $\%Time$ that the algorithm requires. On the other hand, as higher is the value of N/R more dispersed are the spots in the GAP graph and therefore the effect of this variable in the response decrease.

5.3.5. Comparing simulated annealing with tabu search

When we have compared the first method with the second one in previous sections we have noticed how much better is the second one but with the initial solution we have already obtained a significant improvement and we do not know if the simulated annealing algorithm is better yet.

In this section we are going to compare this two metaheuristics. In order to omit the effect of the initial solution algorithm of each method we have used the one obtained in the first one and we have applied the two metaheuristics to this first initial solution, the results are plotted in the next chart.

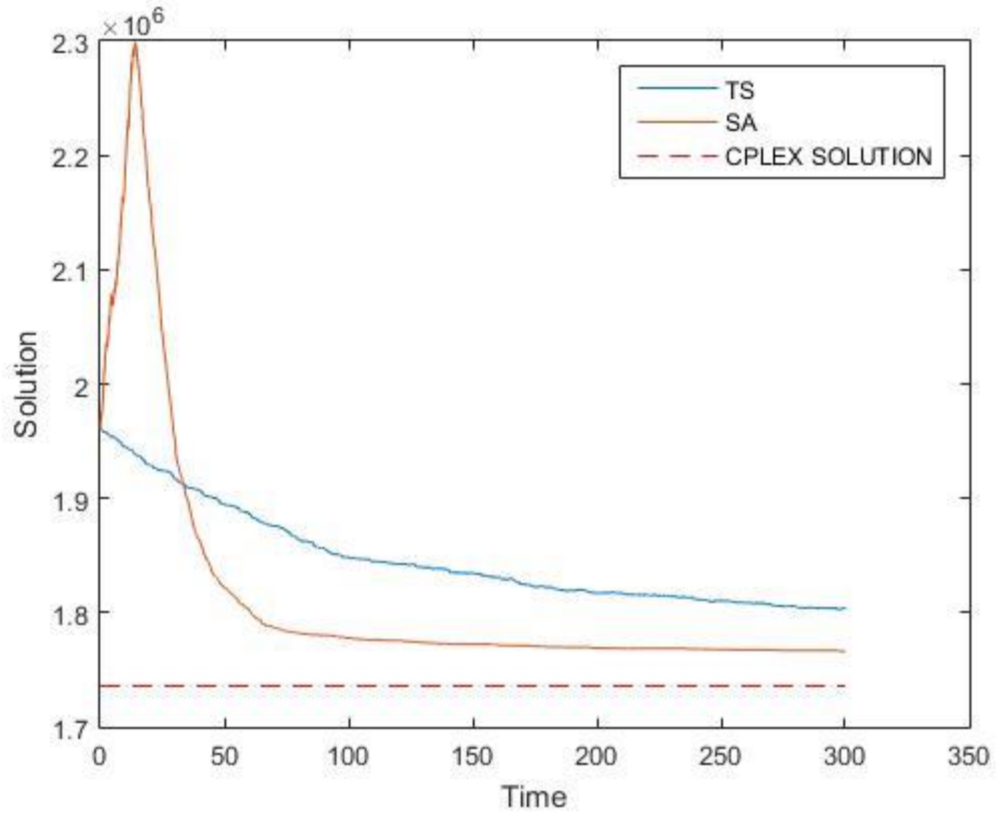

Figure 23: temporal evolution of the solution with tabu search and simulated annealing algorithm

We have tested the sample $T = 20$, $R = 5$, and $N/R = 5$. The first characteristic we can notice is the capacity of the simulated annealing algorithm to deteriorate the solution. However, it can find a better solution much faster than the tabu search and despite the fact that simulated annealing is giving a worse solution during the first 40 seconds it is leading during the rest of the time and reach the convergence in about 80 seconds while tabu search required much more time. As we have seen in previous sections, the GAP between the two methods increase when the value R is high so although we can notice a significant difference between methods in this algorithm, it is actually bigger in other samples.

On the other hand, even improving the result tit keeps being a bit far from the optimal one given by the commercial software. In any case, this graph shows us how much more powerful and effective is the simulated annealing method than the tabu search.

6. Conclusions

During this project we have solved a disassembly lot sizing problem. It has been a two levels problem without commonality parts and some assumptions have been done as we have could notice. Since the amount of time required by the commercial software it too high we decided to solve it using heuristics and metaheuristics which should give us a good solution in a reasonable amount of time.

In the first method we have used a fix–and–optimize heuristic in order to obtain an initial solution. Since it can be feasible or not, a second algorithm based in three stages is used to obtain a feasible solution and finally a tabu search is applied to improve the solution. The solutions obtained from this method are a bit far from the ones obtained using the commercial software as we can notice in the 5.2 section. The worst part of the solution has been the computational time. It has been very high especially with big size samples reaching values of more than 16 hours. However, even being this ones very high values they can be useful in a factory since with a maximum computational time of 16 h we are going to have the disassembly schedule of the next 40 periods. Hence, although it looks very high, it is still a reasonable amount of time.

In order to improve the computational time required we decided to go ahead with an alternative method. In the second one we have proposed an initial a feasible solution obtained from rounding down the solution given by the commercial software applying integer relaxation. This values have been actually better than the ones obtained from the whole first method and the computational time has been very low. After it, we applied a simulated annealing algorithm which shows a very good performance as well. Hence, we have noticed that it is difficult to find a good initial solution in a very short time using heuristics and although it can be useful, it seems better to use the commercial software to solve a relaxed version of the problem and improve it with a heuristic after.

It can seems that the first method has been a waste of time but on the contrary it has been very useful. As a final master project, the objective of this project is purely educational and with both methods several different kinds of heuristics have been used or studied. The first method helped us to understand how some heuristics works and to realize how difficult can be to obtain a good one. The second method was an improvement of the first, it could not be used without the bad results obtained in the first one.

7. References

Hrouga, M., Godichaud, M., & Amodeo, L. (2016). Heuristics for multi-product capacitated disassembly lot sizing with lost sales. *Elsevier*, 628-633.

Jeon, H.-B., Kim, J.-G., Kim, H.-J., & Lee, D.-H. (2006). A Two-Stage Heuristic for Disassembly Scheduling with Capacity Constrains. *International Journal of Management Science*, 95-112.

Johnson, M., & Wang, M. (1998). Economical evaluation of disassembly operations for recycling, remanufacturing and reuse. *International Journal of Production Research*, 26. doi:10.1080/002075498192049

Kaya, M. (2011). *A heuristic approach for profit oriented disassembly lot-sizing problem.* Master Thesis, School of Natural and Applied Sciences, Industrial Engineering.

Kim, H.-J., Lee, D.-H., Kirouchakis, P., & Zust, R. (s.f.). Disassembly Scheduling with Multiple Products Types.

Kim, J.-G., Jeon, H.-B., Kim, H.-J., Lee, D.-H., & Xirouchakis, P. (2006). Disassembly scheduling with capacity constraints: minimizing the number of products disassembled. *IMechE 2006*, 1473-1481.

Lee, D.-H., Kang, K.-W., Doh, H.-H., & Park, J.-H. (2012). Disassembly leveling and lot sizing for multiple product types: a basic model and its extension. *Springer-Verlag London 2012*, 1463–1473.

Gupta, S. M., & Taleb, K. N. (1994). Scheduling disassembly. *International Journal of Production Research*, *32*(8), 1857–1866. http://doi.org/10.1080/00207549408957046

Kim, H.-J., Lee, D.-H., & Xirouchakis, P. (2006). Two-phase heuristic for disassembly scheduling with multiple product types and parts commonality. *International Journal of Production Research*, *44*(1), 195–212. http://doi.org/10.1080/00207540500244443

Kim, H.-J., & Xirouchakis, P. (2010). Capacitated disassembly scheduling with random demand. *International Journal of Production Research*, *48*(23), 7177–7194. http://doi.org/10.1080/00207540903469035

Ma, Y.-S., Jun, H.-B., Kim, H.-W., & Lee, D.-H. (2011). Disassembly process planning algorithms for end-of-life product recovery and environmentally conscious disposal. *International Journal of Production Research*, *49*(23), 7007–7027. http://doi.org/10.1080/00207543.2010.495089

Prakash, P. K. S., Ceglarek, D., & Tiwari, M. K. (2012). Constraint-based simulated annealing (CBSA) approach to solve the disassembly scheduling problem. *International Journal of Advanced Manufacturing Technology*, *60*(9–12), 1125–1137. http://doi.org/10.1007/s00170-011-3670-2