1. INTRODUCTION

I am going to analyze the following two articles:


Both of them make different approaches around the same topic: a branch-and-bound solving method for the resource constrained project scheduling problem (RCPSP).

The problem can be formulated in a very general way as the case of a company that has to develop a project. This project has been divided into single tasks or activities, each one of them has both time and resource requirements. At each period of time, the company has available a certain amount of resources. So, the objective is to minimize the makespan of the project.

This problem can be complicated by introducing new constraints that are specified by particular characteristics of a certain project, such as non-delay condition, headlines inside the project, financial requirements...

2. JUSTIFICATION OF BRANCH-AND-BOUND

Because RCPSP can be reduced to a job-shop scheduling problem, that belong to the class NP, then it is a NP-hard in the strong sense. That means that may not exit a polynomial algorithm that solves this kind of problem in a finite time, o the algorithm would not efficient. However, in this RCPSP, both authors have found a procedure for figure out, at least, a local optimal solution.

RCPSP can be evaluated with the length of the input L=L(X). It is obvious that this measure gets big values for great, and even for regular projects, with the number of activities can easily sum more that a hundred. This leads us to running time of algorithm that would be impossible to compute in an acceptable time for the mainframes.
The only way that we have for dealing with this kind of problem is to breaking into smaller parts, and then to analyze separately them. This is what branch-and-bound does. This technique breaks the feasible region of the problem, the size of each part depends on the information that we can obtain of the problem. When we discover a feasible solution, we can get a lower bound of each unexplored node. These nodes that are infeasible in any way, or its lower bound is worst than the feasible solution can be eliminated of the problem (fathom).

3. APPROACH OF BRUCKER

In their article, he introduces us his notation of the problem as we can read in the very first lines of the article. He define the problem as follows:

- **Definition of the variables**
  
  $S_i$ starting time of each activity, where $i$ goes from 1 to $n$.

- **Definition of the constraints**

  For any time and kind of resource, the system has a limit in the amount of resources that can be used.

- **Definition of the objective function**

  The purpose is to determine the starting times $S_i$, where $i$ is each of the $n$ possible activities and $p_i$ is the time which an activity lasts in being processed. With this information, he define the objective function as the minimum the makespan:

  $$\min_{S_i} \max_i (S_i + p_i)$$

Brucker implement a branch-and-bound algorithm into the RCPSP by setting up the idea of “schedule schemes”. A schedule scheme is a node of his branch-and-bound algorithm that corresponds to sets of feasible solutions. For that reason, he said that a schedule scheme can be divided into four disjoint relations C (conjunctions), D (disjunctions), N (paralleled relations), and F (flexibility relations), where we can place any element of the set $A$.

$$\begin{align*}
V &= \{A_{ij} | ij \in V\}
\end{align*}$$

where $V$ is the set of all activities, including the dummy activities of the start and end of the system (activities 0, and $n+1$, respectively).
The lower limit of each node is calculated by solving relaxed linear program of each feasible solution. This linear programming problem is:

\[
\min \sum_{j} q_j x_{jq} \geq \sum_{i} a_{ij} \quad \text{st} \quad x_{jq} = 1 \text{ if and only if the task } i \text{ is active at the time } j.
\]

where \( x_j \) is the number of units that are been processing at the same time \( j \), and \( a_{ij} \) is 1 if and only if the task \( i \) is active at the time \( j \).

The solution of this problem will be a lower bound of each node. This information will lead us to the resolution of the problem.

Additionally, Brucker provides to us different methods and criterias as with the Transitive Distances matrix, calculation of lower bound with a dynamic program algorithm, or the head and tail method, that help us to the elimination of some possible solution and in the search of the optimal schedule.

4. APPROACH OF DE REYCK

De Reyck studies the RCPSP with the help of the Generalized Precedence Relations (GPR). His start point is the treatment of this problem as a Critical Path Method with relaxations in the assumptions of unlimited resources, and precedence of activities (if \( i \) has a technological precedence relation over \( j \), then \( j \) must be started only after \( i \) completely ends). Both assumptions have been the key in the development and success of the CPM/PERT graphs as solution for this kind of problems. However, this sort of representation is not complete because it ignores this new constraints.

This approach is based in the definition of a time window that is the interval between the minimal and maximal time lag of two activities that have a precedence relation. The model can define these time windows in eight different ways as we can see in the page 154 of the paper.

The iteration of this approach begins with a temporal analysis of a certain schedule, defined by the starting time of all the activities, that satisfies all GPR
conditions, and that no activity starts before time zero. That is, we ensure that the solution is feasible in the time. This analysis can be performed with a distance matrix analysis. This method was used in the Brucker’s approach, too.

Then, we solve a linear programming in a resource analysis. With this operation, we can check that the solution meets all the resources requirements at any time.

The nodes of this branch-and-bound algorithm will be the feasible solutions, that have not been eliminated in the previous two analysis. With this start point, De Reyck begins his research of better solution by delaying activities and looking if this delays can release enough resources to minimize the system. This operation can be completely developed with a branch-and-bound computer program in an acceptable period of time. The model has another point of interest in the question of the reoptimization. De Reyck use Dynamic program for getting the lower bound of the new nodes by computing the longest path between the beginning (a dummy node) and the activity that is under analysis.

The author allow that the objective function takes different forms such as minimizing total project cost, maximizing the cash flow of the project, or minimizing the weighted sum of certain sets of activities that can be considered as subprojects inside a bigger one.

5. COMPARATION BETWEEN BOTH MODELS

It is evident that both models try to resolve the very same problem with a similar method but from different assumptions, notations, and approaches. When we face to this sort of dilemmas, it is interesting to compute the same problem with both models, and to compare then the results. We would certainly get different solutions in a hard case, but both of them will be acceptable.

Brucker presents to us a model where the problem is very clear. We can place the equations for a general form, and solve the model even without knowing the peculiar characteristic of the problem.

The advantage is that Brucker focuses in getting a model with excellent computational results in terms of power of calculation. That is, the model can easily introduce in a computer, and be solved in a short time, even the hardest case. The
computer has only to solve short linear programming, and then to compare, to fathom, to update, or to backtrack the solution. We also use elemental dynamic algorithm that eliminates many unexplored nodes as possible solution.

This model has disadvantages over the second one. Brcket ignores many real possibilities, and data of the original problem. For example, the delays in huge projects are so common as the restrictions that cannot be evaluated in terms of resources activity ready times, headlines of subprojects, or socio-political decisions. From this point of view, the solution, that this algorithms gives us, can be far away from the reality, or we would need to recalculate our schedule in short periods of time. This method misses a realistic sensitivity analysis, that ensure that we are going to be by feasible solution at a small variation.

On the other hand, De Reyck’s model gets a better solution in terms that the model computes more information from the original problem. Moreover, a sort of sensitivity analysis is made by the model, that can be very useful when we will move out of the optimum. Computational strentgh is not a quality of this model, however, we can obtain solution is acceptable period of time.

I can address as disadvantage of this model difficulties in the data entry. It is not easy to evaluate constraints such as inventory restriction, or setup times in critical resources. The model can be turned in a very complicated, almost without solution, problem only for inicializing the program. Moreover, once we operation outside the optimal interval, to recompute the system is a harder task than with the Bucker’s model.

6. CONCLUSION

I have examined two articles about branch-and-bound algorithm for RCPSP. One of the conclusions that I can name is that each type of problem has their own possible approaches with general tools (dynamic programming, linear programming, or Greedy algorithms). This three elements will be our weapons when we face to decisions in our jobs.

Furthermore, the result depends more on the notation, and on the approach than that in the method we choose to solve. In this two articles, one of the author, Brucker, gives priority to the number of computer calculations need to get a solution over the
number of data we take at the beginning. De Reyck build a more complicated model, that, in theory, will give us a more exactly result but losing more time, and this is money inside a company, in the calculation of the model.

Finally, I can say that both articles are very good examples for integer programming because they show how to deal with problems of a high grade of difficulty. How with basic ideas, we can get solutions for integer programming problems that apparently have no trivial solutions.

Pablo Gomez Corredera
December 4th