A NOVEL GREEDY HEURISTIC FOR THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

De Ita Guillermo*, Moyao Yolanda* Soriano Marcela*, Catana Juan*
deita, ymoyao{@cs.buap.mx},
radiomarch, c18a{@hotmail.com}

Abstract.

- *We model a scheduling of multi-projects via intelligent agents, each one of which has to perform a Project.*
- *The agents are non-cooperative, and they compete with others for the common resources, forming instances of the Resource Constrained Project Scheduling Problem (RCPS).*
- *We propose a novel greedy heuristic for solving the RCPS problem.*

Proposal

- *Our heuristic works in an incremental way, building partial scheduling while it determines an order for performing overlapping conflicting tasks.*

- *The resulting algorithm has polynomial time complexity over the number of tasks and shared resources.*

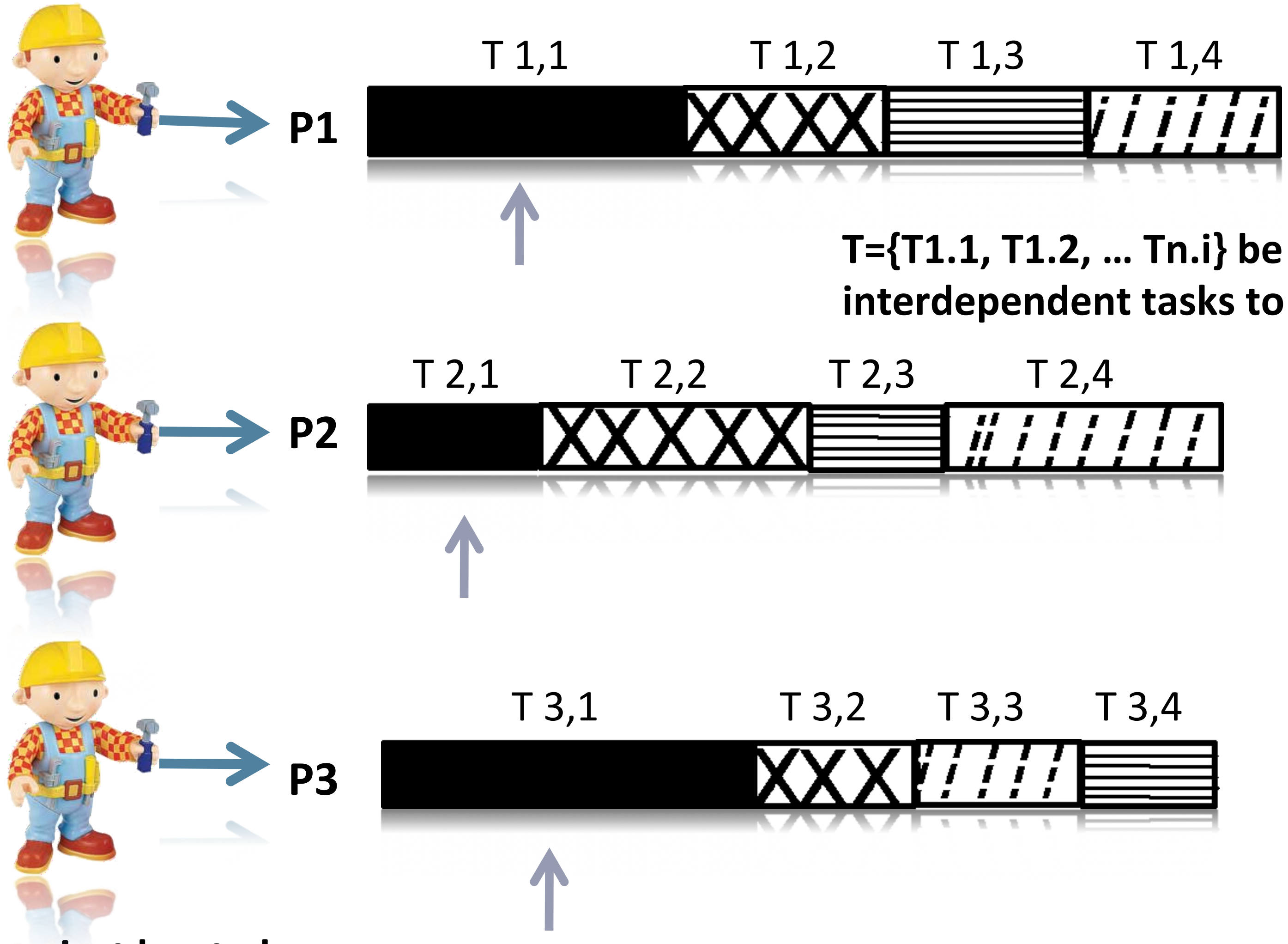
INTRODUCTION

- In the last two decades, computer scientists discovered scheduling as a tool for improving the performance of computer systems.
- An important problem in project management is the allocation of scare resources to competing activities in order to minimize overall project duration.
- In this article, we analyze the problems of scheduling a set of projects which use limited resources, that is, we attack the RCPS problem.

Let A={A1,…,An} be a set of n agents

Let P={P1,…,Pn} be the set of n projects

Each project consist of a set of tasks

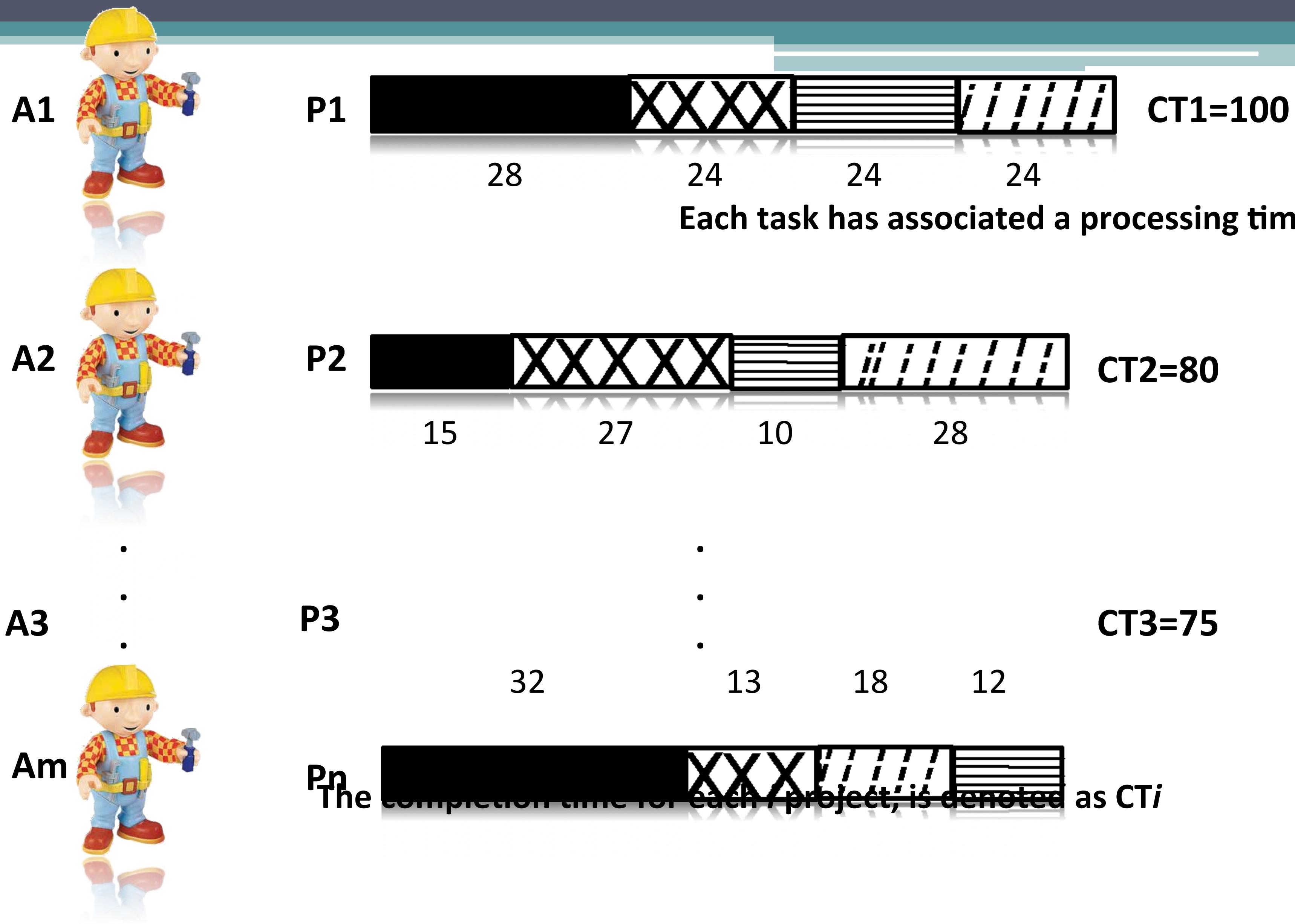


T={T1.1, T1.2, … Tn.i} be the set of interdependent tasks to be carry out.

Each project has to be performed by one agent

R={E1, E2, … Ek} be the set of common resources of the system.

- The RCPS problem consists in finding a schedule of the tasks of a multi-project system with minimal completion times of the projects and into the constraints of the capacity of the resource of the system.
- The RCPS problem is a well know and challenging combinatorial optimization problem which is NP-Hard in the strong sense. For even moderately sized problems, finding an optimal solution in a reasonable amount of time can be very difficult.



Each task has associated a processing time

The completion time for each *i* project, is denoted as CT*i*

The scheduling problem can be formulated as a set of *n* jobs or projects to be scheduled on *m* machines or agents, in our case we will consider *n* = *m*.

- For example, in figure 1 we can see the different conflicting sets:

$CS1 = \{t_{11}, t_{21}, t_{31}\}$ $CS2 = \{t_{12}, t_{22}, t_{32}\}$ $CS3 = \{t_{13}, t_{23}, t_{34}\}$ $CS4 = \{t_{14}, t_{24}, t_{33}\}$

which are the initial conflicting set of tasks.

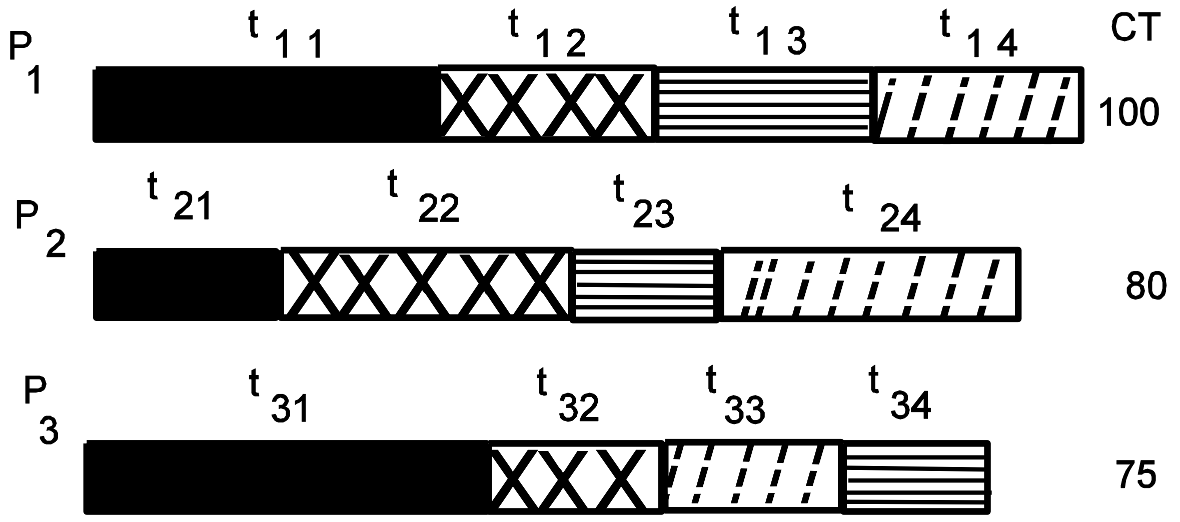


Fig. 1. A Gantt chart where same pattern means same resource.

- The RCPS problem continues being a NP-hard problem since the possibilities of permutations of conflicting tasks which need the same resource. Although for this problem, the explosive number of permutations depends mainly on the number of sharing resources and the number of tasks in conflict.

- The restriction for using sharing resources by just one task at a particular time is called 'Capacity constraint'

- Let be the make span (total completion time of the all project), and *TD* be the total tardiness for the multi-project system

- The multi-objective optimization problem consists roughly in finding a schedule of the *n* jobs that minimizes the make span and the total tardiness.

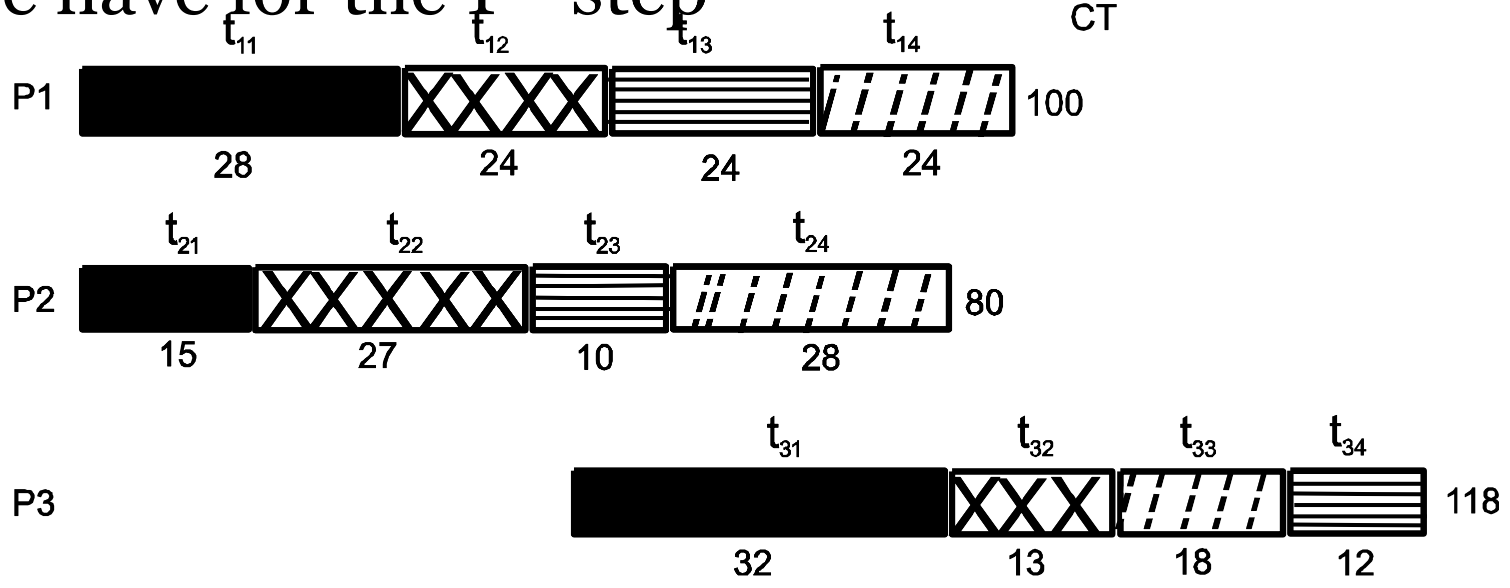
NEW GREEDY HEURISTIC FOR THE JOB SHOP PROBLEM

- Our proposal is a constructive method for attacking the permutation problem which resides in sorting the tasks which are in a conflicting set.

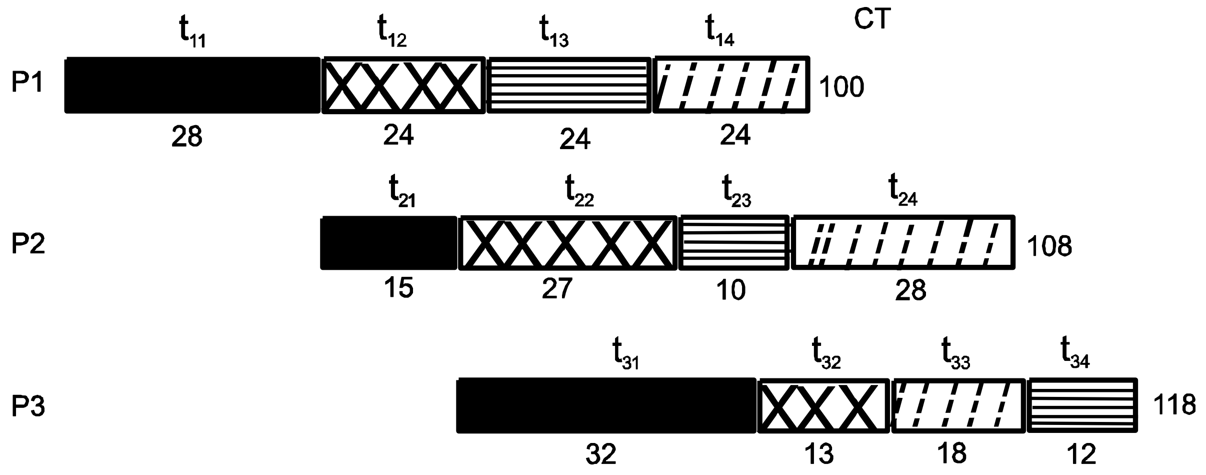
- In our proposal, we are inserting interactively the tasks of the different jobs which request the same resource during the same interval of time.

An example

- Taking the Figure 1. like the initial configuration, we have for the 1st step



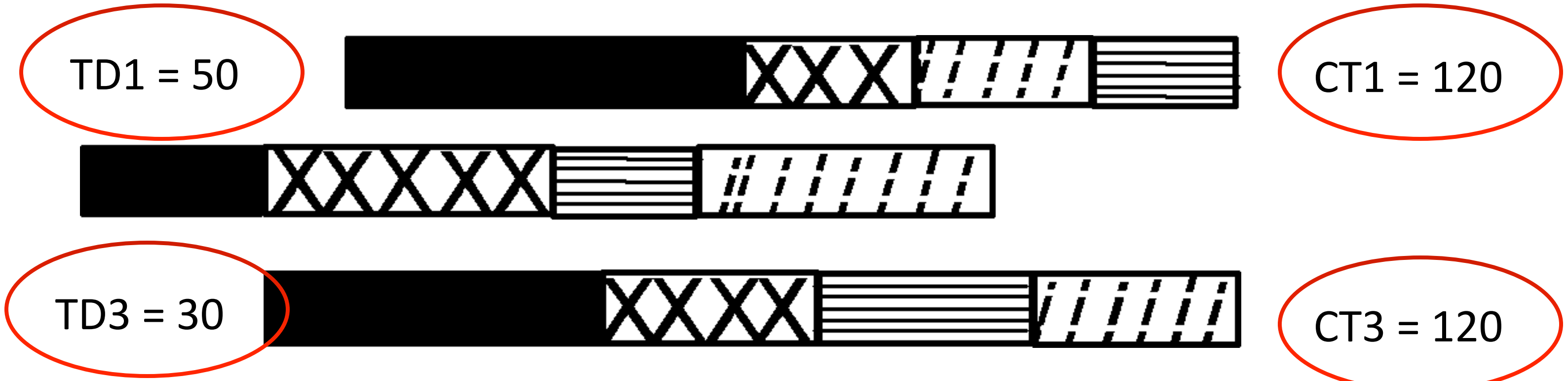
• 2nd step of the Ordering procedure



Figures 3 shows the order of performing for the first tasks of each project. We can see that the first tasks (black tasks) don't have more conflicts. Notice how the completion time of same projects are increased and how they are updating dynamically in each iteration of our procedure. In Figure 3, we note that only the project maintains its initial completion time.

What happens when?

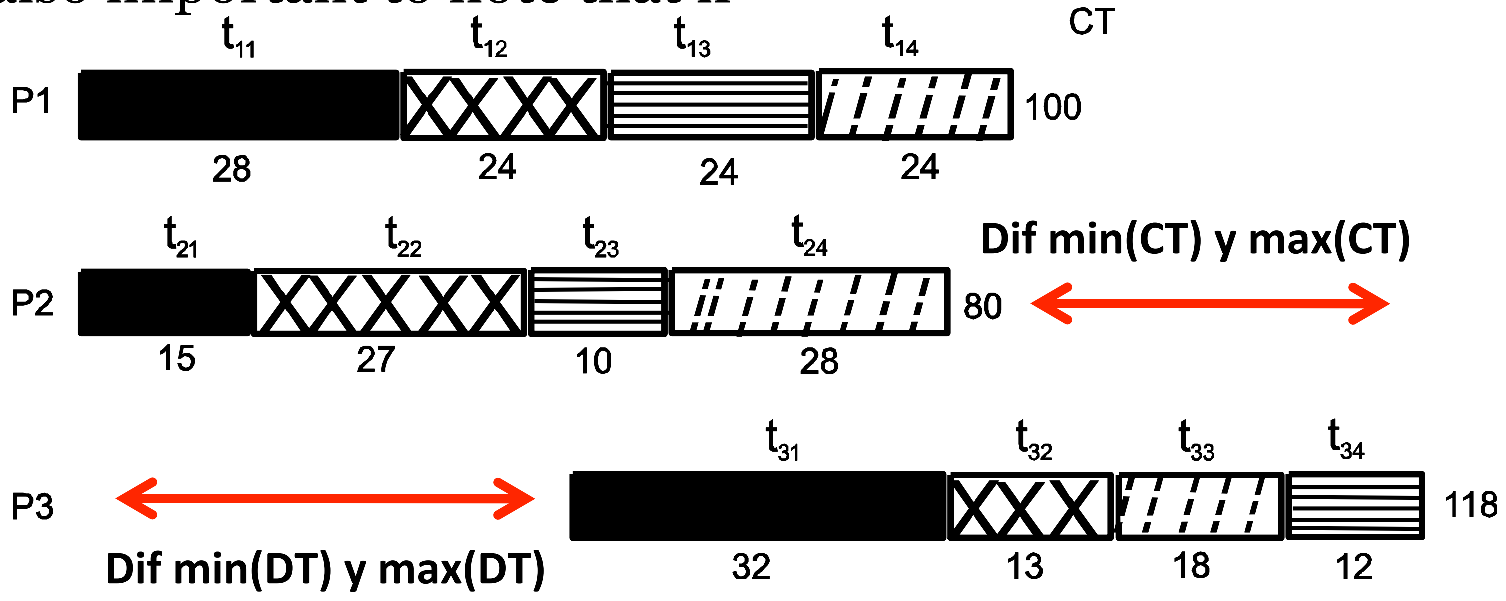
- In each displacement we consider the case that two or more completion times have the same minimal value, and in the case, we consider as a second parameter for deciding, the delayed times in each project.



•So, we check the delay generated by the displacement of each conflicting task, and tasks with minimum delay are chosen.

An optimal movement in each iteration of *Ordering* is obtained if the task choosed infer the lowest growth for its respective completion time as well as it has a minimal growth over its increased delay time.

It also important to note that if



If **Dif min(CT) y max(CT) >= Dif min(DT) y max(DT)**
Choose task of the project with minimum CT

else
Choose task of the project with minimum DT

CONCLUSIONS

- We propose a greedy efficient procedure which in incremental way builds a scheduling for the RCPS problem. Our proposal executes at most *N* = iterations, and in each iteration, the procedure *Ordering* is called if exist a set of conflicting tasks. *Ordering* determines an inverse order for performing a set of at most n conflicting tasks.

- Given a conflicting set of *k* tasks, notice that *Ordering* executes at most $O(N^*n^2)$ basic operations, then *Ordering* has a polynomial complexity time.