A NOVEL GREEDY HEURISTIC FOR THE RESOURCECONSTRAINED PROJECT SCHEDULING PROBLEM

De Ita Guillermo*, Moyao Yolanda* Soriano Marcela*, Catana Juan*

Abstract. *We model a scheduling of multi-projects via intelligent agents, each one of which has to perform a Project. We consider that some tasks need common and limited resources available to all multi-agent system.*
The agents are non-cooperative, and they compete with others for the common resources, forming so instances of the Resource Constrained Project Scheduling Problem (RCPS).
We design a novel greedy heuristic for solving the RCPS problem. 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.
Keywords: *Intelligent Agents, RCPS Problem, Job-Shop Problem, Greedy heuristic*

Resumen. *Modelamos una calendarización de multi-proyectos vía agentes inteligentes, cada uno de los cuales debe ejecutar un proyecto. Consideramos que algunas tareas necesitan recursos compartidos y limitados disponibles para todo el sistema de multi-agentes.*
Los agentes no son cooperativos entre sí, y ellos compiten con los demás por el recurso compartido, formando así instancias de Problemas de Calendarización de Proyectos con Recursos en Conflicto (RCPS por sus siglas en ingles).
Diseñamos una nueva y pretenciosa heurística para la resolución de problemas RCPS. Nuestra heurística trabaja de una forma incremental, construyendo calendarizaciones parciales mientras determine un orden para la ejecución de tareas en conflicto de traslape. El resultado del algoritmo tiene un tiempo de complejidad polinomial sobre el número de tareas y recursos compartidos.
Palabras clave: *Agentes inteligentes, Problema RCPS, Problemas Job-Shop, Heurística pretenciosa.*

1 INTRODUCTION

During the seventies, computer scientists discovered scheduling as a tool for improving the performance of computer systems. Furthermore, scheduling problems have been investigated and classified with respect to their computational complexity. During the last few years, new and interesting scheduling problems have been formulated in connection with flexible manufacturing.

An important problem in project management is the allocation of scare resources to competing activities in order to minimize overall project duration. The commonly used critical path method (CPM), assumes that unlimited resources are available, and that activities requiring a common resource can be carried out in parallel. In this article, we analyze the problems of scheduling a set of projects which use limited resources. The tasks of the projects share common resources and then, different sets of conflicting tasks are formed dynamically according with the order of performing of the previous tasks.

Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of n intelligent agents. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of n projects and such that each project $P_i \in \mathcal{P}$ has to be performed by the agent $A_i \in \mathcal{A}, i = 1, \ldots, n.$ as it is common, each project $P_i \in \mathcal{P}$ consists of a set of interdependent tasks.

Let $\mathcal{T} = \{t_1, \ldots, t_i\}$ be the set of different tasks to be carried out so that each agent accomplishes his project. Some tasks are executed with specialized equipment or by specialized employees. We consider such equipment or specialized personal as common resources to be used by the agents in order to accomplish their projects.

Let $\mathcal{R} = \{E_1, \ldots, E_k\}$ be the set of common resources of the multi-agent system. As it is usual, there is a limited number of employees and equipment to be used in the multi-agent system and as the agents are non-cooperative, they compete with others for the use of the limited resources. In general, the cost of using resource $E_r \in \mathcal{R}$ could represent the time, the price or any other measure that an agent has to pay for using that resource.

The resources are common to all agents but as it happens in practical situations, the same resource is used only for one agent at a particular time, and then a queue of requirements for service could be associated with each resource.

To analyze the affect of limiting resource to overall project performance has motivated a wide amount of researching; one of the related problems more widely studied is the Resource Constrained Project Scheduling Problem (RCPS).

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 can be seen as a generalization of the Job Shop Scheduling problem and so, this is NP-Hard in the strong sense [13]. For even moderately sized problems, finding an optimal solution in a reasonable amount of time can be very difficult. So, RCPS has been utilized as a model to analyze the effect of limiting resource to overall project performance [13].

An adequate review of early RCPS heuristics can be found in [5, 14]. Some versions-extensions of that problem include: multi-project scheduling problem, problems with resource duration interactions, time window constrains, cash flow restrictions and cost-related objectives [6, 9, 10, 13].

Since the RCPS is one of the most intractable problems in Operations Research, it has recently become a popular playground for the latest optimization techniques, including virtually all local search paradigms [11]. The last 20 years have witnessed a tremendous improvement of heuristics, meta-heuristics and exact solution procedures, e.g. see [1-5, 8, 10, 11, 13, 14, 15].

We present in this article, a novel heuristic for solving the RCPS problem, our heuristic has a polynomial time complexity over the number of tasks and resources, and it has shown to obtain good solutions.

2 THE JOB-SHOP SCHEDULING PROBLEM

A job-shop scheduling problem can be formulated as a set of *n* jobs (multi-project) $J = \{J_1, \ldots, J_n\}$ to be scheduled on *m* machines (or by *m* agents), in our case we will consider *n* = *m*. Each job J_i is formed by n_i consecutive tasks $J_i = \{t_{i1}, \ldots, t_{in_i}\}$. There is a sequential order among the tasks in the same project. The task (t_{ik}) represents the *k*-th task of the job J_i. Each task (t_{ij}) has associated a processing time u_{ij} and each job J_i must be achieved before a due time (dt_i). The total time that a job J_i needs for completing all its tasks is the completion time and it is denoted as $CT_i, i = 1, \ldots n$ (see figure 1).

Given a set $\mathcal{R}$ of finite resources, we consider that different tasks need common resources available to all multi-project system. As the resource *R* could be used at a particular time by only one task, an order for performing the conflicting tasks has to be built. When a set of tasks require the same resource $\mathcal{R} \in \mathcal{R}$ at the same time then a set CT_R of conflicting tasks is formed.

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.

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'. For example, for two tasks t_{ik} and t_{jl} of the projects P_i and P_j which require the set of resources; $\mathcal{R}_k$ and $\mathcal{R}_l$ respectively, they cannot overlap unless $\mathcal{R}_k \cap \mathcal{R}_l = \phi$. The capacity constraints give rise a list of disjunctive linear inequalities [8] of type: $\forall t_{ik} t_{jl}\ (\mathcal{R}_k \cap \mathcal{R}_l = \phi) \vee (t_{ik}$ is performed before $t_{jl}) \vee (t_{jl}$ is performed before $t_{ik})$

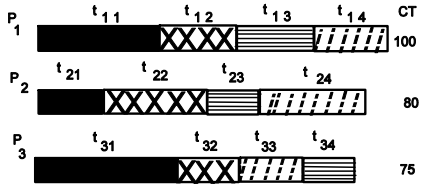

Fig. 1. A Gantt chart where same pattern mean same resource

Let $C_{\max}$ 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. If the task t_{ij} t_{ik} is being scheduled at time s_{ij}, the two objectives can be formulated as follows [12]:

$$f_1 = C_{\max} = Max\{s_{in_i} + w_{in_i} \mid i \in [1...n]\}$$

$$f_2 = TD = \sum_{i=1}^{n} [\max(0, s_{in_i} + w_{in_i} - d_{t_i})]$$

3 NEW GREEDY HEURISTIC FOR THE JOB SHOP PROBLEM

In combinatorial optimization, heuristics allow to iteratively solve in a reasonable time NP-hard complex problems. We propose a novel greedy heuristic for solving the RCPS problem in an incremental way. Our proposal is a constructive method for attacking the permutation problem which resides in sorting the tasks which are in a conflicting set.

Our proposal, called *Ordering*, works likely to the knowing heuristic NEH [15], which is one of the best polynomial time procedure applied for a related problem; the flow-shop problem. In our proposal, instead of inserting a total project in the *Ordering* of projects such as occurs in the NEH algorithm, we are inserting interactively the tasks of the different jobs which request the same resource during the same interval of time.

We use one pointer-time p_i for each job J_i; points to the following task in the project J_i which has to be scheduled. So, we have a *n*-tuple of *n*-pointers $\mathcal{P} = \{p_1, \ldots p_n\}$ indicating the set of the following tasks which are waiting for being scheduled.

At each iteration of our procedure, we determine the minimum time $T_c = Min\{p_i : p_i \mid \in P\}$. If the task t_s pointed by T_c plus its duration has not conflict with the other tasks pointed by P, that is, if $t_s + w_s \leq p_j(t)$ for $j = 1,\ldots,n, j \neq i$, or if the resource that t_s requires is not used by the other tasks pointed by P then p_s is updated as $p_s = p_s + w_s$. That means that the task t_s is not in conflicting with the other current tasks to be performed and then t_s is executed updating the pointer in P corresponding with T_c.

Otherwise, the task t_s is in conflict it current tasks of other projects. Then, at least two tasks pointed by P need the same resource and their respective performing times overlapping.

Let *CS* be the first conflicting set and let $N_{o_{iter}} = w_{in_i} \mid CS \mid$ be the number of tasks to be ordered. $N_{o_{iter}} - 1$ Determines the number of iterations of the procedure *Ordering* which builds an inverse order of execution for the conflicting tasks. In each iteration, *Ordering* chooses the project whose completion time increases minimally, when it's respective conflicting tasks is executed at the end of the all remaining tasks in *CS*.

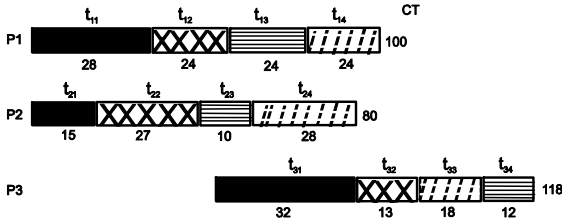

Fig. 2. First step of the procedure *Ordering*

Algorithm 1 Procedure *Ordering*

Input: CS {a set of conflicting tasks}
Initiate Order = " "{the inverse order of the conflicting tasks}
while (|cs| > 1) **do**
for each $s \in CS$ **do**
$Delay(s) = \sum_{c \in CS-\{s\}} Overlapping(t_s, t_c)$ {Sum of the overlapping if t_s is executed at the end of the tasks in conflict}
$CTimes(s) = TT_s + Delay(s)$ {As much as it will extend the completion time for this job}
end for
$p_i = \min\{CTimes(s) : s \in CS\}; \min t = index(p_i, CTimes(s));$
$d_i = \min\{Delay(s) : s \in CS\}; d\,\text{int} = index(d_i, Delay(s));$
if ($p_i == d_i$) **then**
$CS = CS - task_{\min t}; order = order + task_{\min t};$ {It is an optimal selection}
else
$CTT_i = \max\{CTimes(s) : s \in CS\};$ {Maximum completion time}
$DT_j = \max\{Delay(s) : s \in CS\};$ {Maximum delay}
$\Delta CompTime = CTT_j - CTimes(p_i);$ {Differential of completion times}
$\Delta Delay = DT_j - Delay(d_i);$ {Differential of delays}
if $(\Delta CompTime > \Delta Delays)$ **then**
$CS = CS - task_{\min t}; order = order + task_{\min t};$ {Choose task of the project with minimum growth in its completion time}
else
$CS = CS - task_{d\,\text{int}}; order = order + task_{d\,\text{int}};$ {Otherwise it is guided for minimum growth in delays}
end if
end if
end while {order the remaining tasks in CS}

The selected task t_s is deleted from *CS* and the iterative process continues *Ordering* the remaining conflicting tasks. So, in each iteration of the main while in *Ordering*, a conflicting task t_s is selected indicating that t_s has to be performed at the end of the all tasks in *CS*. *Ordering* gives us a total inverse order for performing the conflicting tasks of its input parameter *CS*.

Let us see how *Ordering* works over the example shown in figure 2. *Ordering* determines the order of execution of conflicting tasks in an inverse order, i.e. first, it is determined the task which going to be performed at the end of the conflicting set of tasks, after of this, *Ordering* find the task in the following position, and so on, until all the conflicting tasks are considered.

For determining which tasks is performed at the end of the set of conflicting tasks. *Ordering* estimates how its completion time increase if the task is displaces over the other tasks, and also computes its time of delay, this time going to be added at the total time of project. After this iteration, *Ordering* makes the same evaluation but with the next task, once obtain the three

new completion times, *Ordering* choose the task which is minimum over incremental completion times.

The Figure 2 represents the result of *Ordering* after the first iteration, where the tasks t_{31} was choose to be performed at the end of the other two conflicting tasks since its incremental completion time *TC3*= 75 +28+15 is minimal with respect to the other two possible incremental completion times.

Notice that to look for the task which minimizes its completion time via, *Ordering* realizes an interaction with just a few neighbors (the remaining conflicting tasks), which is a basic design principle to guarantee efficient use of resources in a distributed system [7].

The 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 p_1 maintains its initial completion time.

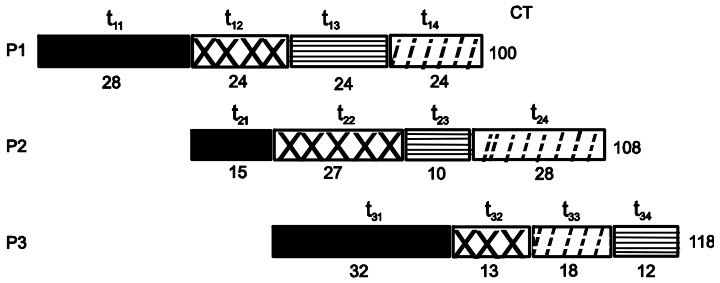

Fig. 3. Second step of the procedure *Ordering*

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.

It also important to note that if the differential of delays, which is the differential time between the last and the first task to be performed, is bigger than the differential of the incremental completion times, which is computed using the differential of the maximum and minimum completion time, both of them computed into conflicting set, then *Ordering* chooses the project which minimizes the 'delay time' of performing more than the completion time.

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.

4 CONCLUSIONS

We propose a greedy efficient procedure which in incremental way builds a scheduling for the RCPS problem. Our proposal executes at most $N = n_1 + ... + n_n$ 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 $k \leq n$, *Ordering* executes at most $O(k^2)$ basic operations. Then our heuristic has a polynomial complexity time of $O(N * n^2)$.

REFERENCES

[1] C. Artigues, P. Michelon, S. Reusser, Insertion Techniques for static and dynamic resource constrained project scheduling, European Journal of Operational Research 149, (2003), pp. 2 49-267.

[2] T. Baar, P. Brucker, S. Knust, Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem, Meta-heuristic: Advances and Trend in Local Search Paradigms for Optimization, Kluwer, (1998), pp. 1-18.

[3] J. Bautista, J. Pereira, ant Colonies for the RCPS Problem, LNCS Vol. 2504, Springer-Verlag, (2002), pp. 257-268.

[4] k. Bouleimen, H. Lecocq, A new efficient simulated annealing algorithm for the resorce-constrained project scheduling problem, Technical Report, Service the Robotique et Automatisation, Universit the Lige, (1988).

[5] E.W. Davis, J.H. Patterson, A comparison of heuristics and optimum solutions inresource constrained project scheduling, Management Science 21 (1975) pp. 944-955.

[6] R.F. Deckro, E.P. winkofsky, j.E. Herbert, R. Gangon, Decomposition approach to multi-project scheduling, Eur. J. of Oper. Res. 51 (1991), pp. 110-118.

[7] R. Elsasser, M. Gairing, T. Lucking, M. Mavronicolas, and B. Monien B, A simple Graph-Theretic Model for Selfish restricted Scheduling, Lect. Notes in Computer Science 3828 (2005) pp. 195-209.

[8] A. Garrido, M.A. Salido, F. Baber, M.A. López, Heuristic Methods for Solving Job-Shop Scheduling Problems, Citeseer 2000, url: citeseer.ist.psu.edu.

[9] S. Kim, R.C. Leachman, Multi-project scheduling with explicit lateness costs, IIE Transactions 25 (1993) 34.

[10] R. Kolisch, S. Hartmann, heuristic Algorithms for solving the resource-constrained project scheduling problem. Classification and computational analysis, Handbook on Recent Advances in Project Scheduling, Kluwer Amstterdam, (1988).

[11] R. Kolish, Serial and parallel resorce.constrained project scheduling methods revisited: Theory and computation, European Journal of Operational Research Vol. 90, (1996), pp. 320-333.

[12] N. melba, M. Mezmaz, E.-G. Talbi, Parallel cooperative meta-heuristic on the computational grid. A case study: the bi-objetive Flow-Shop problem, Elsevier Parallel Computing 32 (2006), pp. 643-659.

[13] K.S. Naphade, S.D. Wu, R.H. Storer, Problem space search algorithms for resource-constrained project scheduling, Annals of Operations research 70 (1997), pp. 307-326

[14] J.H. Patterson, A comparison of exact approaches for solving the multiple constrained resorce project scheduling problem, Management Science 30 (1984), pp. 854-867.

[15] M.G. Ravetti, F.g. Nakamura, c. Meneses, M. Resende, G. Mateus, P. Pardalos, Hybrid Heuristics for the permutation flow shop problem, Teach. Report AT&T Labs TD-6V9MEV, 2006.

Guillermo, De Ita Luna

Did his BTech in Computer Science in the Faculty of Computer Sciences BUAP, Puebla. The master and Ph. D. Program in Electrical Engineering in the Cinvestav – I.P.N., México. He has worked for 10 years as a developer and consulter for Database Systems and Geographic Information System in different enterprises in México.
He has been thesis advisor of 30 works for obtaining the BTech and Master Sciences in Computer Sciences. And he has published more than 35 scientific articles.
He has done researching stages in Chicago University, Texas A & M University, INAOEP – Puebla, and recently (2009) in the Computer Sciences, BUAP – Puebla.

Yolanda, Moyao Martínez

Did her BTech in Computer Science in the Faculty of Computer Sciences BUAP, Puebla. The master Sciences in patterns recognition in the Faculty of Computer Sciences BUAP, Puebla, México.

She has been thesis advisor of 15 works for obtaining the BTech in Computer Sciences. And she has published three scientific articles.
Currently she is a researcher – professor of the Faculty of Computer Sciences, BUAP Puebla.

Marcela Concepción, Soriano Orozco

Just finished her BTech in Computer Science Engineering in the Faculty of Computer Science BUAP, Puebla. She studied English as Second Language at Miracosta College, California.
Currently she is improving her skills to enroll at University of California, San Diego for getting a Master Degree in Computer Science.

Juan Carlos, Catana Salazar

Catana Juan graduate from the Faculty of Computer Sciences BUAP, Puebla. He is a Computer Technician for CBTis No. 44 Teziutlan, Puebla.
* Faculty of Computer Sciences, Universidad Autónoma de Puebla