ISSN 2007-9737

Finding Pure Nash Equilibrium for the Resource-Constrained Project Scheduling Problem

Guillermo De Ita Luna, Fernando Zacarias-Flores, and L. Carlos Altamirano-Robles

Computer Science Department, Autonomous University of Puebla (BUAP), Mexico

{deita, altamirano}@cs.buap.mx, fzflores@yahoo.com.mx

Abstract. The paper focuses on solving the Resource-Constrained Project Scheduling (RCPS) problem with a method based on intelligent agents. Parallelism for performing the tasks is allowed. Common and limited resources are available to all agents. The agents are non-cooperative and compete with each other for the use of common resources, thereby forming instances of RCPS problem. We analyze the global joint interaction of scheduling via a congestion network and seek to arrive at stable assignments of scheduling. For this class of network, stable assignments of scheduling correspond to a pure Nash equilibrium, and we show that in this case there is a guarantee of obtaining a pure Nash equilibrium in polynomial time complexity. The pure Nash equilibrium point for this congestion network will be a local optimum in the neighborhood structure of the projects, where no project can improve its completion time without negatively affecting the completion time of the total system. In our case, each state of the neighborhood represents an instance of the RCPS problem, and for solving such problem, we apply a novel greedy heuristic. It has a polynomial time complexity and works similar to the well-knowing heuristic NEH.

Keywords. Intelligent agents, congestion network, pure Nash equilibrium, RCPS problem, multi-scheduling, greedy heuristic NEH.

1 Introduction

In the seventies, computer scientists proposed scheduling as a tool for improving the performance of computer systems. Furthermore, scheduling problems have been studied 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.

Scheduling is a decision-making process that has as a goal the optimization of one or more objectives. 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. An important problem in practical scheduling scenarios is the allocation of scarce resources for competing activities in order to minimize overall project duration.

As a single scheduling problem, we consider the problem of carrying out just one project. However, in a multi-project scenario, where there is a set of agents performing a set of projects, and such agents are non-cooperative since they compete with each other in order to use common limited resources, it is necessary to analyze the global joint interaction and seek to arrive at stable task scheduling assignments.

In this paper we consider scenarios where tasks are performed through specialized equipment or by specialized employees. We consider equipment and specialized personnel as the common resources to be used by agents in order to accomplish their projects. Let $\mathcal{R} = \{E_1, \ldots, E_k\}$ be a set of common resources to be used in a multiagent system.

As it is common in scheduling problems, there is a limited number of employees and equipment, therefore these resources are required by different agents at the same time; since the agents are noncooperative, they compete for the limited resources. Each resource $E_r \in \mathcal{R}$ has a cost associated to it. This cost can represent time, price, or any other entity that an agent has to invest or pay for using a particular resource.

A lot of researches have been studying the effect of limited resources in project scheduling, and the

doi: 10.13053/CyS-19-1-1921

Resource-Constrained Project Scheduling (RCPS) is a typical problem of this kind. The RCPS problem has shown to have a great variety of applications.

The RCPS is a well-known challenging problem in combinatorial optimization which can be considered as a generalization of the Job Shop Scheduling problem. The RCPS problem consists in finding a schedule of tasks in a multi-project system with a minimal completion time satisfying the constraints defined over the limited resources. The RCPS problem is a strongly NP-hard problem [19].

2 Related Work

The Resource-Constrained Project Scheduling problem has been designed as a model to analyze the effect of limited resources on the overall project performance. An adequate review of early RCPS heuristics can be consulted in [7]. Some versions and extensions of that problem include multi-project scheduling problems, problems with resource duration interactions, time window constrains, cash flow restrictions, and cost-related objectives [8, 13, 14, 19, 18, 24].

Some versions of the RCPS problem has been proved to be of great practical interest, for example, [24] presents a multi-level technical data model which has been useful for creating hierarchical production planning as well as for modeling scheduling in an industrial area. Similarly, [18] shows a version of the RCPS problem considering resource transfers, where the cost of idleness and transfer of resources among projects are analyzed.

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 [14]. It has also been used for proving the advantages of some metaheuristcs like the ant-colony algorithms versus typical local searches [4, 5, 6, 7, 12, 14, 19, 23, 27]. The last two decades have witnessed a tremendous improvement of heuristics, meta-heuristics, and exact solution procedures for solving the RCPS problem and some of its versions, see, e.g., [2, 5, 12, 15, 17, 19, 23, 27].

Pure Nash equilibrium as a tool for finding stable scheduling for selfish users and individual machines was considered in [22, 10]. Since the publication of the above referenced works, several scheduling proposals have been designed based on Nash equilibrium. Research has also been done on analyzing the hardness of finding a pure Nash equilibrium for certain classes of scheduling problems [25, 16, 17, 18, 21, 3, 27].

For example, in [1] Nash equilibrium is characterized in terms of the existence of certain types of cuts on the project network, and in [9] a solver specifically designed to find an equilibrium in concave games is introduced; it is based on the primal-dual interior-point method for nonlinear programming.

In this article we propose to model the RCPS problem (in fact, some versions of scheduling problems) as a non-cooperative game in such a way that a pure Nash equilibrium of the game corresponds to the local optima, in this case, to the minimal values of the total completion time of scheduling based on a neighborhood structure, where no agent can improve its completion time without negatively affecting the total completion time of the multi-project system.

We also present a new greedy heuristic for solving instances of the RCPS problem; our heuristic has a polynomial time complexity on the number of tasks and resources, and it has shown to obtain good solutions. Our heuristic works similar to the well-knowing NEH heuristic, and it is performed with the purpose of accelerating the search for minimal completion times in the main procedure which looks for a pure Nash equilibrium.

3 A Congestion Network for a Multi-Scheduling System

Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of n intelligent agents. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n projects. Each project $P_i \in \mathcal{P}$ is performed by an agent $A_i \in \mathcal{A}$. Each project $P_i \in \mathcal{P}$ consists of a series of consecutive and interdependent tasks $(t_1, \ldots, t_{m_i})$. By the notation t_{ij} we emphasize that the task j is being performed by the agent A_i. There is an order among the tasks of each project based on their interdependency in such a way that

a task cannot be started until all the tasks that it depends on are completed.

A dependency graph or precedence graph (DAG) $G_i = (\mathcal{T}, E(G_i))$ is built for representing the order in which the tasks are executed in a project $P_i \in \mathcal{P}$. The nodes in G_i represent tasks, and we join the last task of the project with a special node labeled as P_i. Each edge $(v, w) \in E(G_i)$ means that the task w depends directly on the task v. The precedence constraints for tasks are represented in this DAG by means of edges, that is, an edge represents a precedence relationship between the corresponding tasks.

The duration of each task is known, and a task may require a common resource throughout its performance. Applying the commonly used critical path method (CPM), we can find the critical path C_i for each DAG $G_i, i = 1, \ldots, n$, assuming that there are unlimited resources available for performing the tasks.

In practical situations, it is possible that two different agents have to carry out similar projects, $P_i = P_j$, $i \neq j, i, j = 1, \ldots, n$, although agents doing similar projects may require different quantities of products associated to their tasks. Some scheduling problems are usually studied in a single-objective deterministic way whereas they are multi-objective by nature.

Nowadays, the importance of multi-objective optimization is widely recognized [16]. Furthermore, taking into account the multi-objective character of a system and various kinds of constraints, we propose a decision maker with a more realistic solution.

In order to analyze the global interaction in a multi-objective project, a congestion network denoted by N_c is formed by joining all final states of the individual DAGs G_i,$i = 1, \ldots, n$ into a final node labeled by F. So the initial state of each G_i, $i = 1, \ldots, n$ is now the initial state of N_c. Then, $N_c = \vee_{i=1}^{n}(G_i \cup (P_i, F))$ (see Figure 1).

In practice, each project $P_i \in \mathcal{P}$, $i = 1, \ldots, n$ could be accomplished in different ways, that is, there could exist different paths in N_c from its initial state t_{i1} to its final state P_i. Let $\mathcal{S}_i = \{s_{i,1}, \ldots, s_{i,n_i}\}$ be a set of different paths for performing P_i on a congestion network N_c. Then, each $s_{i,j} \in \mathcal{S}_i, j = 1, \ldots, n_i$ denotes a series of interdependent tasks for performing the project $P_i, i = 1, \ldots, n$.

In fact, $\mathcal{S}_i$ represents a set of strategies which an agent A_i, $i = 1, \ldots, n$ could utilize for realizing its corresponding project, and each strategy $s_{i,j} \in S_i$ is an ordered sequence of tasks that realizes a project.

For instance, according to Example 1, $s_{1,3} = (t_1, t_2, t_4, t_5, t_7, t_8, t_9, t_{11})$ is one of the possible paths (strategies) for the agent A_1 performing the project P_1. Each task in a particular strategy is linked to resources necessary to fulfill it.

When each resource of the system has a certain finite capacity, idle times are to be considered while scheduling the tasks. Tasks from different projects may require the same resource and within the same interval of time, so different sets of conflicting tasks are formed dynamically according to the order of task fulfillment.

Furthermore, the order of the tasks in a strategy $s_{i,j}$ of an agent $A_i, i = 1, \ldots, n$ is not the only variable for determining the task completion time, since the strategies of other agents also determine different tasks in conflict with those in $s_{i,j}$.

Concerning a single project $P_i \in \mathcal{P}$, its minimal path C_i in G_i is now just one of the possible strategies of the agent A_i in the congestion network N_c.

The collection $\mathcal{C} = (C_1, C_2, \ldots, C_n)$ of the original n-minimal paths may not represent the optimum point in multi-project scheduling, since the concurrent use of resources increases the cost of some (maybe all) original minimal paths, and in such case, a different strategy $s_{i,j} \neq C_i$ of the agent A_i could be, in a global joint interaction, less expensive than the cost of C_i.

Example 1. In an enterprise, there are two employees represented by two agents $A_1 and A_2$. A_1 is responsible for filling large containers with water, while A_2 is responsible for filling medium-size bottles. Each agent determines a strategy to accomplish his daily project. The corresponding DAGs G_1 and G_2 are shown in Figure 1, and the catalog of their tasks is presented in Table 1.

The different strategies for the agent A_1, $\mathcal{S}_1 = \{s_{1,1}, s_{1,2}, s_{1,3}, s_{1,4}, s_{1,5}\}$, are
$s_{1,1} = (t_1, t_2, t_3, t_6, t_7, t_8, t_9, t_{11})$,
$s_{1,2} = (t_1, t_2, t_4, t_6, t_7, t_8, t_9, t_{11})$,
$s_{1,3} = (t_1, t_2, t_4, t_5, t_7, t_8, t_9, t_{11})$,

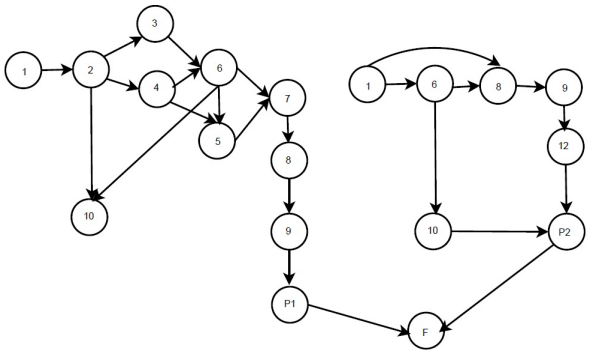

Fig. 1. The congestion network of Example 1

$s_{1,4} = (t_1, t_2, t_3, t_6, t_5, t_7, t_8, t_9, t_{11})$,
$s_{1,5} = (t_1, t_2, t_4, t_6, t_5, t_7, t_8, t_9, t_{11})$.
And the possible strategies $\mathcal{S}_2 = \{s_{2,1}, s_{2,2}\}$ for A_2 are
$s_{2,1} = (t_1, t_6, t_8, t_9, t_{12}, t_{11})$,
$s_{2,2} = (t_1, t_8, t_9, t_{12}, t_{11})$.

The space of states is $S = \mathcal{S}_1 X \mathcal{S}_2$. Considering a sequential processing, for each state $e \in S$ there are two possible sequences for scheduling tasks: e.g., to execute $s_{1,j}$ and afterwards $s_{2,l}$, or to execute $s_{2,l}$ and then $s_{1,j}$, $j = 1, \ldots, 5, l = 1, 2$.

We assume that all agents choose only one of their strategies $s_i \in \mathcal{S}_i$ $i = 1, \ldots, n$, and then a state (an action of the multi-agent system) is formed as $e = (s_1, \ldots, s_n) \in \mathcal{S}_1$ X ... X $\mathcal{S}_n$. Let $S = \{e_1, \ldots, e_o\}$ be a set of different states that can be formed in a multi-project system, each state $e_j, j = 1, \ldots, o$ being a configuration of the multi-agent system.

Then $S = \mathcal{S}_1$ X ... X $\mathcal{S}_n$ and its cardinality is given by $\mid S \mid = \mid \mathcal{S}_1 \mid * \ldots * \mid \mathcal{S}_n \mid = n_1 * \ldots * n_n$, where $n_i, i = 1, \ldots, n$ is the number of different strategies that an agent A_i has in order to complete the project P_i assigned to him.

In a sequential processing, given a state $e = (s_1, \ldots, s_n) \in S$, there are $n!$ possible ways to execute all the strategies of e since any permutation of $s_1 s_2 \ldots s_n$ results in a different way of realizing n projects. If we do not consider parallelism for performing the tasks, then the total number of possible configurations for a multi-agent system is $(n_1 * n_2 * \ldots * n_n) * n!$.

Table 1. Catalog of tasks for two different projects

Task	*Description*	*Resource*	*Units/Min*
1	reception	1 person	300 containers
2	initial revision	1 person	300 containers
3	washing machine	washing machine	5 containers
4	big brush	1 person brush	4 containers
5	sterilization	sterilizer machine	5 containers 20 bottles
6	second revision	1 person	5 containers
7	rinse	1 hose	120 containers
8	filling station	filling machine	1 container 10 bottles
9	capping station	capping machine	2 containers 15 bottles
10	rejection	1 person	5 containers
11	delivery	1 person	5 containers
12	labeling	1 person	20 bottles

Given this exponential number of possible configurations for a multi-agent system, we have to apply computational methods that allow us to reduce the number of solutions, as well as to address the search for optimal configurations in an appropriate manner.

Starting from here, we denote as t_{ik} the k-th task of the project P_i. Each task t_{ij} has a processing time w_{ij} associated to it, and each project P_i must be completed before its deadline dt_i. Let s_{ij} be a start time for scheduling the task t_{ij}. For $i = 1, \ldots, n$ the total time that a project P_i requires for completing all its tasks is its completion time denoted as CT_i, while TD_i denotes the total tardiness spent by A_i in order to fulfill the last task P_i.

Considering a multi-agent system, let C_{max} be the makespan (the total completion time of a multi-project system) and TD be the total tardiness of the system. The multi-objective optimization problem consists roughly in finding a schedule of n projects that minimizes the makespan and the total tardiness. Both objectives can be formulated as minimizing the following functions:

$f_1 = C_{max} = Max\{s_{in_i} + w_{im_i} | i \in [1 \dots n]\}$,
$f_2 = TD = \sum_{i=1}^{n} [max(0, s_{im_i} + w_{im_i} - dt_i)]$.

The goal of our proposal consists in finding a state e (one strategy for each agent in order to perform its corresponding project) that minimizes the makespan function and, for the same state e, avoids achieving a maximized value of the tardiness function.

We consider the case where two or more states have the same minimal value of the makespan of a set of projects, and in this case, we establish the total tardiness time of each project as a second parameter for decision making.

4 Finding a Pure Nash Equilibrium for the RCPS Problem

We model each instance of the RCPS problem determined by a state $e \in S$ via a congestion network with common resources and a maximum of n tasks to be carried out in parallel. The maximum number of tasks to be performed simultaneously in parallel equals the total number of agents.

Given a state $e = (s_1, \dots, s_n) \in S, s_i \in \mathcal{S}_i, i = 1, \dots, n$, if we consider parallelism among the projects allowing a maximum number of tasks to be performed at any time and if we suppose that the resources are unlimited, then the order of execution of n strategies is not relevant and all permutations of e have equal completion time. Thus, due to parallelism, the cardinality of the state space is $|S| = n_1 * n_2 * \dots * n_n$, with $n_i = |\mathcal{S}_i|, i = 1, \dots, n$.

A parallel scheduling scheme is easier to implement, it's more intuitive and has good results with classic scheduling rules [5]. Furthermore, Kolish has shown that a parallel scheduling scheme builds solutions in a set of non-delay schedules. However, it may occur that an optimal solution will not be included in the non-delay set, so a parallel scheduling scheme might not generate an optimal solution even if all possible schedules are generated [14].

Indeed, we must consider the delay time of an agent that wants to use an occupied equipment or to interact with a busy employee while we try to minimize the makespan C_{max}. Given a specific state e, if a set of current tasks to be performed requires the same resource $R \in \mathcal{R}$, a set CS_R of conflicting tasks is formed by those tasks.

We associate with each task $t_{ij} \in CS_R$ a 'delay time' denoted by $Delay(A_{ij})$, which is the time that an agent A_i has to wait until it can use a common resource $R \in \mathcal{R}$ in order to perform its current task t_{ij}.

The value of $Delay(A_{ij})$ for a task $t_{ij} \in CS_R$ depends on the number of agents in the queue waiting for their usage of the resource R, and for different orders of execution of the tasks in CS_R, different values of $Delay(A_{ij})$ are obtained. Furthermore, $Delay(A_{ij})$ $i = 1, \dots, n$, $j = 1, \dots, n_i$ depends on the strategies chosen by all the agents in the state e and not only on the strategy chosen by A_i.

We define a real-valued cost function $T(t_{ij})$ on the set of tasks. $T(t_{ij})$ is the function which computes time required to perform each task, including its delay time. Of course, if t_{ij} is a current task which is not in conflict with any other current task of some other project, then $T(t_{ij})$ is w_{ij} - just the processing time for performing t_{ij}.

It is common that $T(t_{ij})$ depends on the quantity of items that are manipulated while performing the task, but its main dependence is on its delay time $Delay(A_{ij})$. In any case, we assume that $T(t_{ij})$ can be determined at any time during the scheduling system operation according to the number of items handled by the task t_{ij} and the number of agents waiting to use the same resource required by t_{ij}.

The time associated with an agent $A_i, i = 1, \dots, n$ in a state $e = (s_1, \dots, s_n) \in S$ is determined as $T(s_i, e) = \sum_{t_{ij} \in s_i} T(t_{ij})$. Then, $T(s_i, e)$ denotes the completion time of A_i for performing the project P_i and is given by the sum of the delay time for performing each task and the duration of each task. Let $T(e)$ be the maximum completion time of n-agents while performing their projects. This value coincides with the makespan C_{max} after the strategies used by the agents are fixed. Then, given a fixed state e, $T(e) = C_{max} = max\{T(s_i, e) : i = 1, \dots, n\}$ represents the makespan of the multi-agent system.

Due to parallelism in performing the tasks, the total completion time $T(e)$ associated with a state e is

upper bounded as $T(e) \leq \sum_{s_i \in e} T(s_i, e)$. $T(e)$ represents an exact potential function for this congestion game. The optimization goal of this scheduling problem consists in finding a state e (strategies for all agents) which minimizes the value of $T(e)$.

A *pure Nash equilibrium* for this congestion game is an assignment of strategies to n agents such that no individual agent can reduce its completion time by changing its strategy. And in order to find a pure Nash equilibrium in this scenario, we must build a neighborhood structure for the set of states of a multi-agent system.

We consider that each agent $A_i \in \mathcal{A}$ chooses one of its strategies $s_i \in S_i$,$i = 1, \ldots, n$ forming a state $e = (s_1, \ldots, s_n) \in S$. An *improvement step* of an agent A_i is a change of its strategy from s_i to s'_i, thus achieving a new state e' and causing a decrease of time $T(s'_i, e')$ with respect to $T(s_i, e)$.

Therefore, we can see the neighborhood of a state e consisting of such states that deviate from e only in one of the agent's strategies. The improvement of the time of an agent A_i is precisely $T(s'_i, e') - T(s_i, e)$.

An improvement movement for finding equilibrium points is given by determining, starting from a state e, a new state e' such that $T(e') < T(e)$ and e' is a neighbor of e since they differ in only one strategy. Then, the current state e will be updated repeatedly by replacing it with a neighbor that gives a better time until such a state is reached which cannot be improved by any simple strategy, that is, *locally optimal* with respect to the neighborhood structure.

Therefore, the desired equilibriums in this multi-agent system are in one-to-one correspondence with the local optima of the potential function $T(e)$ under the 'change-one-agent's-strategy' neighborhood structure. Notice that as long as the solution space is finite, such local optima must exist.

Notice as well that this network congestion is not symmetric since all agents have different starting nodes and different projects associated with them. A great advantage on this multi-scheduling system is that N_c is an acyclic graph and the sequences of improvement steps do not run into cycles. This ensures the possibility to reach a pure Nash equilibrium after a finite number of steps [11].

Searching for an optimal interactive strategy is a complex problem because the effectiveness of its solution depends mostly on the strategies of all agents involved and, mainly, on solution of instances of the RCPS problem. Notice that each state $e \in S$ defines an instance of the RCPS problem where individual deadlines are not considered and there is a fixed order among the tasks of a single project.

It is known that the problem of finding an equilibrium point in a congestion network is a PLS-complete problem [11]. Indeed, it is known that problems in PLS have a PTAS (a polynomial-time approximation scheme) [22]. For determining if such problem is in the PLS class, we apply a greedy heuristic designed specially with the purpose to achieve an equilibrium state for the RCPS problem.

5 A Greedy Heuristic for the RCPS Problem

Several approaches for finding exact solutions of the RCPS problem have been developed, some of the most effective are techniques based on branch and bound procedures [23, 26]; although due to the NP-hardness of the problem, these exact procedures have exponential time complexity. Also, several heuristics and metaheuristics have been developed [4, 5, 6, 12, 15]. Among the heuristic approaches, the greedy heuristics based on progressive construction of the solution using a priority rule and a serial or parallel scheduling scheme are the most accepted methods [14].

Given a state $e \in S$, and in order to preserve a polynomial time complexity for finding a pure Nash equilibrium, it is necessary to apply polynomial procedures for solving each instance of the RCPS problem.

In this article, we present a novel greedy heuristic for solving the RCPS problem. Our proposal is a constructive method for attacking the permutation problem which resides in sorting the conflicting tasks.

Our heuristic, called $Ordering$, works similar to the well-known heuristic NEH [14, 20], which is one of the best polynomial time procedures applied

in a related problem, the flow-shop problem. In our proposal, instead of inserting a total project in the ordering of projects such as it occurs in the NEH algorithm, we interactively insert the conflicting tasks of different projects which request the same resource within the same time interval.

We use one pointer-time p_i for each project P_i; p_i points to the current task of the project P_i which has to be scheduled. An n-tuple $P = (p_1, p_2 \ldots, p_n)$ is formed with n-pointers to the current tasks still are to be scheduled.

The minimum time $T_c = Min\{p_i : p_i \in P\}$ of a set of time pointers is determined in each iteration of the procedure $Ordering$. If a task t_{ij} pointed by T_c plus its duration is not in conflict with the other tasks in P, i.e., if $T_c + w_{ij} \leq p_j$ for $j = 1, \ldots, n, j \neq i$, or if a resource that t_{ij} requires is not needed for the other tasks pointed by P, then t_{ij} is executed updating its corresponding time pointer and the iteration continues with the following task to be performed.

Otherwise, the task t_{ij} is in conflict with the current tasks of other projects. Then, at least two tasks pointed by P need the same resource and their respective times of execution overlap. In this case, the set CS is formed. CS contains all the conflicting current tasks which are aimed via the tuple P. Let $No_{iter} = |CS|$ be the number of tasks to be ordered. $No_{iter} - 1$ is the total number of iterations of the main while-loop in the procedure $Ordering$.

In each iteration of the main while-loop, $Ordering$ estimates, for each conflicting task in CS, to what extent its completion time increases if the performance of the corresponding conflicting task is displaced to the end of all remaining tasks in CS. After this estimation, $Ordering$ makes the same evaluation but for the next task in the conflicting set CS.

Once the new completion times are obtained, $Ordering$ chooses the task t_{ij} whose completion time of its project TC_i increases to a minimal extent with respect to the new completion times of the other projects involved in CS. When t_{ij} is fixed to be performed at the end of the tasks in CS, t_{ij} is deleted from CS and the iterative process continues to order the remaining conflicting tasks.

Figure 2 shows the result of $Ordering$ after the first iteration, where the first tasks t_{11}, t_{21}, t_{31} are in conflict. In this example, the initial completion times for each project are $TT_1 = 100, TT_2 = 80$, and $TT_3 = 75$, while the processing time for the tasks are $w_{11} = 28, w_{21} = 15$, and $w_{31} = 32$. In this case, t_{31} was chosen to be performed after the other two conflicting tasks because its corresponding incremental completion time $TC_3 = 75 + 28 + 15 = 118$ is minimal with respect to the other two possible incremental completion times ($TC_1 = 100 + 15 + 32 = 147$ and $TC_2 = 80 + 28 + 32 = 140$).

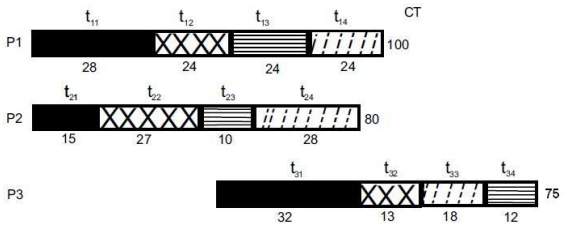

Fig. 2. Gantt chart where each pattern means a different resource

Notice that $Ordering$ determines the order of execution of conflicting tasks in an inverse order, i.e., first it determines the task which is going to be performed at the end of the conflicting set of tasks, then $Ordering$ finds the task in the penultimate position, and so on, until all the conflicting tasks are considered.

To search for the task with a minimal completion time requires an analysis of a few neighbors, i.e., only the set of conflicting tasks whose maximum cardinality is n; this is the basic design principle to guarantee an efficient use of resources in a distributed system [10].

Now we consider the case in which two or more completion times have the same minimal value, and for such case, we consider the delayed time of each project as a second parameter for decision making. So, we check the delay generated by displacing each conflicting task and the task with minimum delay is chosen to be performed before the other tasks.

It is important to note that if the differential of delays, which is the differential of time between the last and the first task to be performed, is bigger

than the differential of the incremental completion time, which is computed using the differential of the maximum and minimum completion times, both of them computed for the conflicting tasks set, then $Ordering$ chooses the project that minimizes the performance 'delay time' more than the completion time.

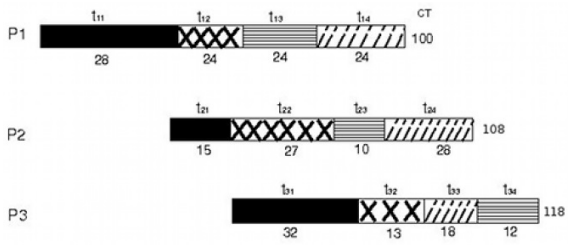

Fig. 3. End of the processing of the first conflicting tasks

Figure 3 shows the order of execution for the first tasks (black tasks) of each project until they don't have more conflicts. Notice how the completion time of some projects is increased and how they are dynamically updated in each iteration of our procedure.

An optimal movement in each iteration of $Ordering$ is obtained if the selected task infers the lowest increment of its respective completion time and has the minimal increment over its increased delay time.

Looking for a stable point (a pure Nash equilibrium) in this multi-project system, iterations of at most $N = n_1 + \ldots + n_n$ are executed, and in each iteration, the procedure $Ordering$ is called if there exists a set of conflicting tasks, to obtain an order for performing the 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 the time complexity of our procedure is $O(N * n^2)$ which is polynomial over the number of tasks of the projects.

Algorithm 1 Procedure $Ordering$

Input: CS {a set of conflicting tasks}
Initiate $Order = \text{“ ”}$ {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
  $p_i = min\{CTimes(s) : s \in CS\}; mint = index(p_i, CTimes(s))$;
  $d_i = min\{Delay(s) : s \in CS\}; dint = index(d_i, Delay(s))$;
  if $(p_i == d_i)$ **then**
    $CS = CS - task_{mint}; order = order + task_{mint}$; {It is an optimal selection}
  else
    $CTT_j = max\{CTimes(s) : s \in CS\}$; {maximum completion time}
    $DT_j = max\{Delay(s) : s \in CS\}$; {maximum delay}
    $\Delta CompTime = CTT_j - CTimes(p_i)$; {differential of completion times}
    $\Delta Delays = DT_j - Delay(d_i)$; {differential of delays}
    if $(\Delta CompTime > \Delta Delays)$ **then**
      $CS = CS - task_{mint}; order = order + task_{mint}$; {choose the task of the project with the minimum increment in its completion time}
    else
      $CS = CS - task_{dint}; order = order + task_{dint}$; {otherwise it is guided for the minimum increment in delays}
    end if
  end if
end while

6 Conclusion and Future Work

We illustrate a type of congestion network game which models the scheduling of n projects via n non-cooperative agents. In this system, we consider parallelism among tasks of different projects and the use of common limited resources. This congestion network is suitable for modeling the Resource-Constrained Project Scheduling Problem (RCPS).

The heuristic applied to solve instances of the RCPS problem preserves a polynomial time com-

plexity and works similar to the well-known NEH heuristic. We show that for this kind of networks there is a theoretical *pure Nash equilibrium*, that is, when a multi-agent system reaches an equilibrium point (pure Nash equilibrium).

In a pure Nash equilibrium $e = (s_1, \ldots, s_n)$, each agent A_i has chosen an optimal strategy $s_i, i = 1, \ldots, n$ for performing the project tasks assigned to him as the best response in the total multi-agent system. Then, the Nash equilibrium for this scheduling problem is the assignment of strategies to agents such that no individual agent can reduce its project completion time by changing its strategy. Nash equilibriums are the only fixed points of a dynamic multi-agent system defined by improvement steps.

The greedy heuristic for solving instances of the RCPS problem interactively inserts tasks in the set of conflicting tasks such that in each step of the procedure an optimal inverse order to perform the current conflicting tasks is looked for.

The use of a congestion network for modeling the RCPS problem through the competing intelligent non-cooperative agents as well as the application of a heuristic for solving instances of the RCPS problem allow us to fix a polynomial time upper bound to achieve stable configurations of the system, i.e., determine minimal makespan values of scheduling based on a neighborhood structure; no project can improve its completion time affecting negatively the total completion time of the multi-project system. In fact, our method can be applied for solving different versions of scheduling problems.

Acknowledgements

The authors would like to recognize SNI-CONACyT and the academic group of Combinatorial Algorithms of the Autonomous University of Puebla for their financial support. Also, the authors would like to acknowledge many helpful suggestions and valuable comments on this article given by the anonymous reviewers and by the Editor of this Journal.

References

1. **Agnetis, A., Briand, C., Billaut, J., & Sucha, P.** (**2014**). Nash equilibria for the multi-agent project scheduling problem with controllable processing times problem. *Journal of Scheduling*, Vol. 1, pp. 125–142.
2. **Artigues, C., Michelon, P., & Reusser, S.** (**2003**). Insertion techniques for static and dynamic resource constrained project scheduling. *European journal of Operational Research*, Vol. 149, pp. 249–267.
3. **Averbakh, I.** (**2010**). Nash equilibrium in competitive project scheduling. *European journal of Operation Research*, Vol. 205, pp. 552–556.
4. **Baar, T., Brucker, P., & Knust, S.** (**1998**). *Meta heuristics: Advances and Trends in local search paradigms for optimizatiion*, chapter Tabu search algorithms and lower bounds for the resource constrained project scheduling problem. Kluwer, pp. 1–18.
5. **Bautista, J. & Pereira, J.** (**2002**). Ant colonies for the RCPS problem. *Lecture notes in computer science*, Springer.
6. **Bouleimen, K. & Lecocq, H.** (**1998**). *Technical report, service de robotique et automatisation*, chapter A new efficient simulated annealing algorithm for the resource constrained project scheduling problem. Universit de Lisge, pp. 1–20.
7. **Davis, E. & Patterson, J.** (**1975**). A comparison of heuristics and optimum solutions in resource constrained project scheduling. *Management Science*, Vol. 21, pp. 944–955.
8. **Deckro, R., Winkofsky, E., Herbert, J., & Gangon, R.** (**1991**). Decomposition approach to multi project scheduling. *European journal of Operation Research*, Vol. 51, pp. 110–118.
9. **Drwal, M., W., R., Ganzha, M., & Paprzycki, M.** (**2014**). Equilibria in concave non-cooperative games and their applications in smart energy allocation. *Internet and Distributed Computing Systems, LNCS*, volume 8729, pp. 409–421.
10. **Elsasser, R., Gairing, M., Lucking, T., Mavronicolas, M., & Monien, B.** (**2005**). A simple graph theoric model for selfish restricted scheduling. *Lectures notes in computer science*, Springer.
11. **Fabrikant, A., Papdimitriou, C., & Talwar, K.** (**2004**). The complexity of pure Nash equilibria. *Proceedings 34th ACM Symposium on Theory of Computing, STOC04*, ACM.

12. **Garrido, A., M.A.Salido, Baber, F., & Lopez, M.** (**2000**). Heuristic methods for solving job shop scheduling problems. *Proc. ECAI-2000 Workshop on New Results in Planning, Scheduling and Design (PuK2000)*, pp. 44–49.
13. **Kim, S. & Leachman, R.** (**1993**). Multi project scheduling with explicit lateness costs. *IIE Transactions*, Vol. 25, pp. 98–108.
14. **Kolisch, R.** (**1996**). Serial and parallel resource constrained project scheduling methods resisited: Theory and computation. *European journal of Operation Research*, Vol. 90, pp. 320–333.
15. **Kolisch, R. & Hartmann, S.** (**1998**). *Handbook on recent advances in project shceduling*, chapter Heuristic algorithms for solving the resource constrained project scheduling problem: Classification and computational analysis. Kluwer, pp. 1–20.
16. **Liefooghe, A., Basseur, M., Jourdan, L., & El-Ghazali, T.** (**2007**). Combinatorial optimization of stochastic multi objective problems: and application to the flow-shop scheduling problem. *Lecture notes in computer science*, Springer.
17. **Ma, Y., Gao, Y., & Wang, L.** (**2009**). A pcu resource scheduling algorithm based on Nash equilibrium. *Proceedings of the future information networks*, FIN.
18. **Mittal, M. & Kanda, A.** (**2009**). Scheduling of multiple projects with resource transfers. *International journal of mathematics in operational research*, Vol. 1, pp. 303–325.
19. **Naphade, K., Wu, S., & Storer, R.** (**1997**). Problem space search algorithms for resource constrained project scheduling. *Annals of operations research*, Vol. 70, pp. 307–326.
20. **Nawaz, M., Enscore Jr., E., & Ham, I.** (**1983**). A heuristic algorithm for the m-machine n-job flowshop sequencing problem. *OMEGA International Journal of Management Science*, Vol. 11, pp. 91–95.
21. **Nguyen, K. T.** (**2009**). NP-Hardness of pure Nash equilibrium in scheduling and connection games. *Lecture notes in computer sciences*, Springer.
22. **Orlin, J., Punnen, A., & Schulz, A.** (**2004**). Approximate local search in combinatorial optimization. *Proceedings of SODA*, SODA.
23. **Patterson, J.** (**1984**). A comparasion of exact approaches for solving the multiple constrained resourve project scheduling problem. *Management Science*, Vol. 30, pp. 854–867.
24. **Pereyrol, F., Dupas, R., Alix, T., & Bourrieres, J.** (**1996**). Serial and parallel resource constrained project scheduling methods resisited: Theory and computation. *European journal of Operation Research*, Vol. 90, pp. 320–333.
25. **Ravetti, M., Nakamura, F., Meneses, C., Resende, M., Mateus, G., & Pardatlos, P.** (**2006**). Hybrid heuristics for the permutation flow shop problem. *Technical report, AT&T Labs.*, AT&T Labs TD-6V9MEV.
26. **Simpson, W. & Patterson, J.** (**1996**). A multiple tree search procedure for the resource constrained project scheduling problem. *European journal of Operation Research*, Vol. 89, pp. 525–542.
27. **Venkataramana, M. & Raghavan, N.** (**2010**). Ant colony based algorithms for scheduling parallel batch processors with incompatible job families. *International journal of mathematics in operational research*, Vol. 2, pp. 73–98.

Guillermo De Ita Luna received his B.Sc. in Computer Science from the Faculty of Computer Science of the Autonomous University of Puebla (BUAP), Mexico, his M.Sc. and Ph.D. in Electrical Engineering from CINVESTAV-IPN, Mexico. He has worked for 10 years as a developer and consulter for Database Systems and Geographic Information Systems in various enterprises in Mexico. He has done research stays at Chicago University, Texas A&M, INAOEP-Puebla, and recently (2009) in the INRIA Institute in Lille, France. Currently, he is a researcher-professor at the Faculty of Computer Science, BUAP-Puebla.

Fernando Zacarias-Flores is researcher and professor of Computer Science at the Autonomous University of Puebla. He is a researcher in practical and theoretical Computer Science and Mobile Technologies, and has conducted R&D projects in this area since 2000. Results from these projects have been reported in more than 60 national and international publications. Professor Zacarias serves in the editorial board of the following journals: IEEE Latin America Transactions, Engineering Letters, International Transactions on Computer Science and Engineering, Common Ground Publishing - Technology, Learning and Social Sciences. He holds a Ph.D. degree in Computer

Science from UDLAP, M.Sc. in Electrical Engineering from CINVESTAV-IPN and B.Sc. in Computer Science from BUAP.

Luis C. Altamirano-Robles received his B.Sc. and M.Sc. in Computer Science from the Autonomous University of Puebla, Mexico, and his Ph.D. in Computer Science from the National Polytechnic Institute (IPN), Mexico. Currently, he works at BUAP and coordinates the Postgraduate Section of the Computer Science Faculty.

Article received on 12/12/2013, accepted on 27/11/2014.
Corresponding author is Fernando Zacarias-Flores.