Finding a Pure Nash Equilibrium for the Scheduling of Multi-Projects via Non-Cooperative Agents

Guillermo& De Ita* Pedro& Bello* Meliza& Contreras*

*Department of Computer Science, Universidad Autonoma de Puebla
Av. Sn. Claudio esq. 14 Sur C.U, Puebla, México

Abstract

We model a scheduling of multi-projects via intelligent agents, each one of which makes a project. Some tasks in this system can be done in parallel. We also consider that some tasks need common resources available to all multi-agent systems. The agents are non-cooperative, and they compete with the others for the common resources.

We analyze the global joint interaction using a network congestion game and seek to arrive at stable assignments of the scheduling of the tasks. For this class of congestion network, the stable assignments of the scheduling are modeled as Pure Nash equilibrium, and we show in this case, there is a guarantee that we shall obtain a pure Nash equilibrium via local search algorithms.

Keywords: *Intelligent Agents, Network Congestion Game, Pure Nash Equilibria*

1 Introduction

Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of n intelligent agents. We consider the problem of scheduling a set of projects $\mathcal{P} = \{P_1, \ldots, P_n\}$. Each project $P_i \in P$ has to be done by the agent $A_i \in \mathcal{A}$, $i = 1, \ldots, n$, and each project $P_i \in \mathcal{P}$ consists of a set of interdependent tasks.

Some tasks are executed with specialized equipment or by specialized employees. We consider such equipment or specialized personnel as common resources to be used by the agents in order to accomplish their projects. As usual, there is a limited number of employees and equipment to be used in the multi-agent system.

Let $\mathcal{T} = \{t_1, \ldots, t_l\}$ be the set of different tasks to be carried out so that each agent accomplishes his project. Let $\mathcal{R} = \{E_1, \ldots, E_k\}$ be the set of common resources to be used in the multi-agent system in order to execute the tasks. The agents are non-cooperative, and they compete with others for the use of the common resources. Scheduling is the problem of allocating limited resources to do all the tasks in a limited time of operation [4].

As it occurs in single scheduling problems, certain tasks depend on others; that is, they can not begin until all the tasks they depend on are completed. In fact, it is possible that two different agents have to carry out the same project; $P_i = P_j$, $i \neq j, i, j = 1, \ldots, n$. Although agents doing the same project could require different quantities of products associated with that project.

We can build a dependency graph or precedence graph (DAG) G_i for each project $P_i \in \mathcal{P}$. Each task represents a node of G_i and we join the last task of the project with a special node labeled as P_i. Each edge $(v, w) \in G_i$ meaning that the task w depends directly on the task v. Applying a topological order over each DAG G_i we can find the critical path C_i for each DAG $G_i, i = 1, \ldots, n$.

The resources are common to all agents but as it happens in practical situations, the same resource is only used for one agent at a particular time, then a queue of requirements for service is associated with each common resource. Furthermore, each resource $E_r \in \mathcal{R}$ determines an increasing cost function f_r which depends on the number of agents that want to use the resource E_r. The cost of using a 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.

In a single scheduling problem, we consider the problem of carrying out just one project $P_i \in \mathcal{P}$. But in a multi-scheduling scenario where all agents compete with each other in the use of common resources, we have to analyze the global joint interaction and seek arriving at stable assignments of the scheduling of the tasks. For this analysis, we model the stable assignments as Pure Nash equilibria of a network congestion game [2, 5].

2 A Congestion Network for the Multischeduling System

Let N_c be the congestion network formed by joining all final states of the DAG's G_i,$i = 1, \dots, n$ to a final node labeled by F. And where the initial state of each G_i, $i = 1, \dots, n$ is now an initial state of N_c. Then, $N_c = \vee_{i=1}^{n}(G_i \cup (P_i, F))$.

Usually, each project $P_i \in \mathcal{P}$ could be accomplished in different ways, that is, there could exist different paths in G_i from its initial state to its final state P_i, $i = 1, \dots, n$. We determine for each project $P_i \in \mathcal{P}$ the set of different paths to realize such project. So, for each agent $A_i \in \mathcal{A}$ a finite set $S_i = \{S_{i,1}, \dots, S_{i,k_i}\}$ is built. S_i is called the set of strategies of A_i, $i = 1, \dots, n$. Then, S_i contains the different paths on the congestion network N_c which allow A_i to carry out the project P_i.

In the sequence of tasks (a strategy) for doing a project, the order of tasks is relevant, so a different order of the tasks determines a different way of doing the same project and usually with different times. For example, $S_{1,3} = (t_1, t_2, t_4, t_5, t_7, t_8, t_9, t_{11})$ indicates the order of execution of the tasks to make the project P_1, according to the example 1. And each task of a stategy has associated the necessary resources for attaining such task. Then, each strategy indicates a path to realize a project as well as the necessary resources to be used in that project.

Regarding 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.

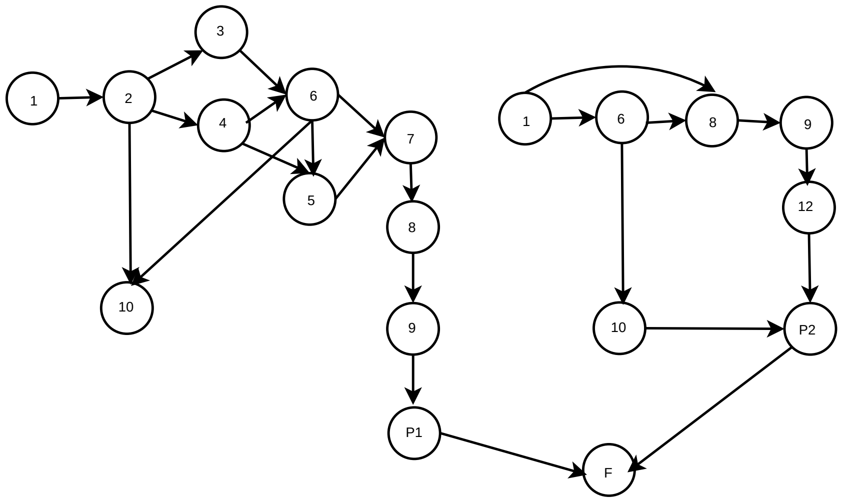

Figure 1: *The congestion network of the example 1 where the nodes are the tasks.*

And the collection $\mathcal{C} = (C_1, C_2, \ldots, C_n)$ of the n-minimal paths should not represent the optimum point in the scheduling of the multi-projects since the concurrent use of resources increase the cost of some (maybe all) minimal paths, in such a way that a different strategy $S_{i,j} \neq C_i$ for the agent A_i could be, in a global joint interaction, less expensive than the cost of C_i.

When all agents choose one of their strategies $s_i \in S_i$ $i = 1, \ldots, n$, a state (an action in the multi-agent system) is formed $e = (s_1, \ldots, s_n) \in S_1$ X $\ldots$X S_n. Let $S = \{e_1, \ldots, e_o\}$ be the state set of the multi-scheduling system, where each state $e_j, j = 1, \ldots, o$ is a configuration of the system. Then, $S = S_1$ X $\ldots$X S_n and the cardinality of S is given by $\mid S \mid = \mid S_1 \mid * \ldots * \mid S_n \mid = n_1 * \ldots * n_n$, where each $n_i, i = 1, \ldots, n$ is the number of different paths that the agent A_i has for realizing the project P_i.

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

Example 1. Let A_1, A_2 be two agents, A_1 is responsible for filling large containers with water, while A_2 is responsible for filling medium bottles. Each agent determines its strategies to accomplish its project. The correspondings DAG's; G_1 and G_2 are shown in figure 1 and the list of their tasks appears in table 1. The strategies $S_1 = \{s_{1,1}, s_{1,2}, s_{1,3}, s_{1,4}, s_{1,5}\}$ for the agent A_1, are:
$s_{1,1} = (t_1, t_2, t_3, t_6, t_7, t_8, t_9, t_{11})$
$s_{1,2} = (t_1, t_2, t_4, t_6, t_7, t_8, t_9, t_{11})$
$s_{1,3} = (t_1, t_2, t_4, t_5, t_7, t_8, t_9, t_{11})$
$s_{1,4} = (t_1, t_2, t_3, t_6, t_5, t_7, t_8, t_9, t_{11})$
$s_{1,5} = (t_1, t_2, t_4, t_6, t_5, t_7, t_8, t_9, t_{11})$
The strategies $S_2 = \{s_{2,1}, s_{2,2}\}$ for the agent 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 $S = S_1 X S_2$. And 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 execute $s_{2,l}$ and then $s_{1,j}$, $j = 1, \ldots, 5, l = 1, 2$.

Given this exponential number of possible configurations for the multi-agent system, we should apply computational methods which allow us to reduce the number of factible solutions as well as to address the searching for optimal configurations in an adecuate way.

In this global scenario, there are some tasks which can be done in parallel, using different equipment or different employees. Then in a global analysis, we have to capture the maximum of tasks which can be done in parallel. And under this consideration, the maximum number of tasks performing in parallel at the same time, coincides with the number of projects and for this reason, in this article, we are considering the same number of agents as of projects.

Each task $t \in \mathcal{T}$ has associated a cost function $T(t)$ representing the time for doing that task. Of course, if t is a single task then $T(t)$ is just a constant time. Although, it is common that $T(t)$ depends on the quantity of items which are manipulated through the task t.

However, there are delayed times when an agent has to wait for using busy equipment or to interact with busy employees. Regarding N_c, we have to consider that some tasks require common resources and then the cost of using the resource reflects different times for t according to the number of agents in the queue waiting for using the same resource.

In any case, we assume that $T(t)$ could be determined at any time during the scheduling system via a congestion function which depends on the number of items handled for the task t and also depends on the number of agents waiting to use the resource associated with t.

We model this system as a congestion network with common resources and with a maximum of n tasks to be carried out in parallel, as well as a set of n non-cooperative agents competing among themselves for the completion of their projects.

Task	**Description**	**Resources**	**Duration**
1	Tap Water reception	1 person	5 containers per sec.
2	Initial revision	1 person	5 containers per sec.
3	Using Washing Machine	washing machine	5 containers per min.
4	Using a big brush	1 person,brush	4 containers per min.
5	Tap Water Sterilization	1 Sterilizer Machine	5 containers per min. 20 bottles per min.
6	Second revision	1 person	5 containers per sec.
7	Rinse	1 hose	2 containers per sec.
8	Filling Station	1 Filling Machine	1 container per min. 10 bottles per min.
9	Capping Station	1 Capping Machine	2 container per min. 15 bottles per min.
10	Rejection	1 person	5 containers per min.
11	Delivery	1 person	5 containers per min.
12	Labeling	1 person	20 bottles per min.

Table 1: List of tasks

Given a state $e = (s_1, \ldots, s_n) \in S, s_i \in S_i, i = 1, \ldots, n$ if we consider that the sequence of tasks in e are done in parallel, maximizing at any time the number of tasks doing in parallel and if we suppose that not common resources are needed for performing the tasks, then it is not relevant the order of the execution of the n strategies and all permutations of e have equal completion time. Thus, due to the parallelism, the cardinality of the space of states is reduced to $|S| = n_1 * n_2 * \ldots * n_n$, with $n_i = |S_i|, i = 1, \ldots, n$.

Let $SP_i = \sum_{t_j \in S_i} T(t_j)$ be the completion time for performing all the tasks involved in a strategy $s_i \in S_i$, $i = 1, \ldots, n$. Due to the parallelism for doing the tasks, the completion total time associated with a state e, denoted by $T(e)$, is bounded by $T(e) \leq \sum_{s_i \in e} SP_i$. Furthermore, as each strategy s_i determines a sequence of tasks that they can not be done in parallel, then $max\{T(s_i) : s_i \in e\} \leq T(e)$.

When the tasks require common resources in order to be performed, then there are ‘delayed times’ associated with the schedule of the projects since the agents in the queue of a common resource E_r have to wait until E_r is liberated, and then the following agent in the queue can use now the resource E_r. We denote the cost as $Delay(A_{ij})$ (perhaps the time) that the agent A_i has to wait for using a common resource in order to do the task t_j.

Given a state $e = (s_1, \ldots, s_n) \in S$ the time associated with an agent $A_i, i = 1, \ldots, n$ for such state is determined as: $T(A_i, e) = \sum_{t_j \in s_i}(T(t_j) + Delay(A_{ij}))$. $T(A_i, e)$ is the time for realizing the project P_i and it is given as adding the delayed time that the agent has to wait for starting to perform each task plus the time for performing such tasks.

Note that the values $Delay(A_{ij})$ $i = 1, \ldots, n$ $j = 1, \ldots, n_i$ depend on the strategies of all agents in the state e and not only on the strategy choosed by A_i. Intuitively, each agent A_i can choose one strategy from among the set of strategies, but the time of the state depends on all strategies choosen by the agents.

The time associated with the state e is determined as: $T(e) = max\{T(A_i, e) : i = 1, \ldots, n\}$. So, the time of the state e should be the maximum completion time of the agents for performing the n projects.

In order to find the equilibrium point in this scenario, we consider that each agent $A_i \in \mathcal{A}$ chooses one of his 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 his strategy from s_i to s'_i changing to a new state e' and where the time $T(A_i, e')$ decreases with respect to $T(A_i, e)$. Thus, we can see the neighborhood of a state e consists of those states that deviate from e only in one agent's strategy. And the improvement of the time of an agent A_i is precisely $T(A_i, e') - T(A_i, e)$.

Each state $e \in S$ defines an instance of the *'job-shop'* problem where individual deadlines are not considered. There are a great number of algorithms for solving this problem, see for example [4, 6, 8]. In our research, we consider a variation of the NEH heuristic [8], since it has been one of the best polynomial time algorithm for a related problem; the flow-shop problem. In our proposal, instead of inserting a total project in the ordering projects such as occurs in the NEH algorithm, we were inserting interactively the set of tasks of the different jobs which coincide during the same interval of processing. Although, due that the job-shop problem is a NP-hard problem when more than two machines (agents) are considered [1], we should apply heuristics which allow us to compute $T(e)$ efficiently although we do not obtain the minimum exact value for $T(e)$.

In this multi-agent system, the desired equilibria are in one-to-one correspondence with the local optima for the potential function $T(e)$ under the 'change-one-agent's-strategy' neighborhood structure. It would then repeatedly update the current state e by replacing it with a neighbor which gives a better time, until it reaches such a state that no movement of a simple strategy could improve it, that is, one that was *locally optimal* with respect to the neighborhood structure. Note that so long as the solution space is finite, such local optima must exist [9].

Searching for an optimal interactive strategy is a hard problem because its effectiveness depends mostly on the strategies of all agents involved and mainly, in solving the instances of the flow-shop problem in the most precise way.

Notice that this network congestion is not symmetric since all agents have different starting nodes and different projects associated with them. But, a great advantage on this multischeduling sytem is that N_c is an acyclic graph and this guarantess that N_c has a pure Nash equilibria [3].

Although the problem of finding the equilibrium point is a PLS-complete problem [3]. Indeed, it is known that problems in PLS have a PTAS (a polynomial-time approximation scheme) [7]. By identifying that this problem is in the class PLS, we can apply smart algorithms designed to find equilibrium state (see for example [3, 5, 7]). There are a series of such algorithms looking for a Nash pure equilibrium, some of them working in sub-exponential time, and others working as polynomial approximation algorithms.

3 Conclusions

We illustrate a type of congestion network game for modelling the scheduling of n multi-projects via n non-cooperative agents. In this system, we assume that some tasks require using common resources and also, it is possible the parallelism for executing tasks among different projects. We show that for this kind of network there is a theoretical *Nash pure equilibrium*, that is, the multi-agent system reaches an equilibrium state $e = (s_1, \dots, s_n)$.

In a Nash pure equilibrium; $e = (s_1, \dots, s_n)$, each agent has chosen an optimal strategy $s_i, i = 1, \dots, n$ in the form of a best response to the choice in the total multi-agent system. The Nash equilibrium for this system of multischeduling is the assignment of strategies to agents such that no individual agent can reduce his associated time by changing his strategy. The Nash equilibria are the only fixed points of the dynamic defined by the improvement steps in this multi-agent system.

References

[1] M.R. Garey, D.S. Johnson, *Computers and Intractability*, Bell Labs. 1979.

[2] R. Elsässer, M. Gairing , T. Lücking , M. Mavronicolas , and B. Monien B, *A Simple Graph-Theretic Model for Selfish restricted Scheduling*, Lect. Notes in Computer Sciences **3828** (2005) pp. 195-209.

[3] A. Fabrikant , C. Papdimitriou and K. Talwar, *The complexity of Pure Nash Equilibria*, Proc. 34th ACM Symposium on Theory of Computing, STOC'04 (2004).

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

[5] D. Johnson, *The NP-Completeness Column:Finding Needles in Haystacks*, ACM Transactions on Algorithms **3:2** (2007).

[6] N. Melab, M. Mezmaz, E.-G. Talbi, *Parallel cooperative meta-heurisitcs on the computational grid. A case study: the bi-objective Flow-Shop problem*, Elsevier Parallel Computing 32 (2006), pp. 643-659.

[7] J.B. Orlin , A.P. Punnen and A.S. Schulz,*Approximate local search in combinatorial optimization*, Proc. of SODA (2004), pp.580-589.

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

[9] B. Vöcking, *Congestion games: Optimization in competition*, Proc. 2nd Algorithms and Complexity in Durham Workshop, Kings College Publications (2006), pp. 9-20.