Non-disjoint Multi-agent Scheduling Problem on Identical Parallel Processors

F Sadi, van Ut Tran, N Huynh Tuong, Ameur Soukhal

Non-disjoint multi-agent scheduling problem on identical parallel processors

F. Sadi[a,b], T. Van Ut[a,c], N. Huynh Tuong[d], and A. Soukhal[a]

[a]Laboratory of Computer Science (EA 6300),
Team Recherche Opérationnelle, Ordonnacement et Transport
ROOT ERL-CNRS 6305), 64 Avenue Jean Portalis, 37200 Tours
Université François Rabelais Tours,
[b]INSA Centre Val de Loire, 3 rue de la chocolaterie, 41000 Blois
[c]Can Tho University of Technology (CTUT),
256 Nguyen Van Cu Street, Ninh Kieu District, Can Tho City, Vietnam
[d]Faculty of Computer Science & Engineering,
Ho Chi Minh City University of Technology, VNU-HCM, 268 Ly Thuong Kiet
Street,Ho Chi Minh City 740500, Vietnam
sadi.faiza@gmail.com
vanut.tran@etu.univ-tours.fr
htnguyen@hcmut.edu.vn
ameur.soukhal@univ-tours.fr

Abstract. Scheduling problems in which agents (users, customers, application masters, resource manager, etc.) have to share the same set(s) of resources are at the frontier of combinatorial optimization and cooperative game theory. This paper deals with scheduling problems arising when two agents, each with a set of nonpreemptive jobs, compete to perform their respective jobs on two common identical parallel machines. Each agent aims at minimizing a certain objective function that depends on the completion times of its jobs only. The objective functions we consider in our study are makespan and number of tardy jobs. The agents may share some jobs and this problem is called *non-disjoint multi-agent scheduling problem* (Agnetis et al. 2014). Finding the optimal solution for one agent with a constraint on the other agent's cost function is known to be $\mathcal{NP}$-hard. To obtain best compromise solutions for each agent, we propose polynomial and pseudo-polynomial heuristics. Two mixed integer linear programming models are developed to calculate exact non-dominated solutions. Experimental results are conducted to measure the solutions quality given by heuristics.

Keywords: Multicriteria optimization, Multiagent scheduling, parallel processors, heuristics, Dynamic programming, linear mathematical programming.

1 Introduction

Efficient management of large-scale job processing systems is a challenging problem, particularly in the presence of multi-users. In addition, real world systems

require processing of jobs on common resources and considering different objective functions. Most conventional algorithms are designed for multi-criteria scheduling problems where each measure is applied on the whole set of jobs without any distinction. However, these classical multi-criteria scheduling problems are not well appropriated to the more general resource allocation problem across heterogeneous networks frequently encountered in real applications. It is useful to classify jobs as either time constrained, which should be scheduled as soon as possible, or non-time-constrained, which should simply be processed before their due date. This is the subject of multi-agent scheduling (Agnetis et al. 2014, Sadi et al. 2014, Huynh Tuong et al. 2012). Indeed, in (Peha 1995) the author considered an integrated-services packet-switched networks such as ATM (Asynchronous Transfer Mode). Information carried by the network are first split into smaller messages called *packets*. The data comes from different types, such as voice, video, image and so on. Each packet is wrapped with the essential information needed to get it from its source to the correct destination. In the case of audio and video data the authors show that minimizing the number of late delivered packets is more relevant when for the other types of data, the minimizing delay queuing is more suitable. Delay queuing is commonly expressed in the scheduling literature by the total completion time. Peha's results provide polynomial time algorithms to schedule a set of n jobs on m identical parallel machines with assumption of unit processing times.

In this paper, we study the problem of scheduling jobs on identical parallel processors. The model is featured by agents (local decision makers) - each of which is associated with a subset of jobs to perform, and each one has its own objective function depending only on the completion times of its jobs. The agents share not only the common resources but also some common jobs. This problem has been introduced by (Agnetis et al. 2014) and called *non-disjoint multi-agent scheduling problem*. These problems belong to a particular class of multi-criteria scheduling problems where their practical and theoretical benefits are highlighted in (Agnetis et al. 2014). During this past decade, such a class has drawn a significant interest to researchers dealing with scheduling problems and from operational research domain.

Depending on the agent's relationships, three scenarios have been defined in (Agnetis et al. 2014): COMPETING (CO), INTERFERING (IN) and NON-DISJOINT (ND). According to our knowledge, except few results appeared in (Agnetis et al. 2014), the non-disjoint scenario is not already studied in the literature. However, the competing scenario is no doubt the most studied scenario until now. It was introduced in (Baker et al. 2003), the authors considered two disjoint sets of jobs, each one is associated with one agent and one objective function. The jobs have to be executed on the single machine and the goal is to find the best compromise solutions between the two agents. When the ε-constraint approach is used, a polynomial time algorithm is proposed to minimizing the number of tardy jobs of each agent. In (Ng et al. 2006), the authors study Peha's problem introduced in (Peha 1995) by considering any processing times (not necessary identical). They present an NP-hardness proof and propose a

dynamic programming algorithm to calculate a non-dominated solution. The interfering job set scheduling problems is particularly studied in (Huynh et al. 2012) and (Sadi et al. 2014). Polynomial and pseudo-polynomial time algorithms are derived for settings with various combinations of the objective functions in the case of single processor and parallel processors.

The rest of this paper is organized as follow. We define the problem in Section 2. In Section 3, we propose a discussion on the appropriate solutions structure. Section 4 is dedicated to the mathematical programming formulations which determine strict pareto solutions. Two polynomial heuristics are proposed and presented in Section 5. We also develop two pseudo-polynomial heuristics in Section 6. A comparison between the exact and approximate methods is illustrated in Section 7. Conclusion and future researches are presented in Section 8.

2 Problem definition and notations

We consider two competitive agents A and B sharing the same machines. Each agent is owning a set of jobs. We denote by $\mathcal{J}^A = \{J_1^A, J_2^A, \ldots, J_{n_A}^A\}$ a job set associated with agent A, while $\mathcal{J}^B = \{J_1^B, J_2^B, \ldots, J_{n_B}^B\}$ is the job set associated with agent B. The agents can share some jobs, that means that $\mathcal{J}^A \cap \mathcal{J}^B$ is not necessary empty. The whole set of jobs is denoted by $\mathcal{J}$ such as $|\mathcal{J}| = n$, that is given by $\mathcal{J}^{\mathcal{A}} \cup \mathcal{J}^B$. The machine environment is composed of two identical parallel machines, denoted by M_i that are always available, for $i = 1, 2$. Preemption is not allowed and each machine can process only one job at a time. The processing time of J_j denoted by p_j is known and given, where C_j is its completion time, $\forall j \in \mathcal{J}$. We assume that all jobs are available at time zero, and jobs within agent B are subject to a common due date denoted by d^B. In this study, the objective function to be minimized of agent A is the makespan of its jobs: $C_{\max}^A = \max_{J_j^A \in \mathcal{J}^A} C_j$, while agent B minimizes the number of its tardy jobs: $\sum U_j^B = \sum_{J_j^B \in \mathcal{J}^B} \mathbb{1}_{\{C_j > d^B\}}$.

According to the three fields notation of the multi-agent scheduling problems introduced by (Agnetis et al. 2014), the problem addressed in this paper is denoted by $P2|ND, d^B|(C_{max}^A, U_j^B)$, where ND means that the job sets are non-disjoint. When the ε- constraint approach is considered, the studied problem is denoted by $P2|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A$. In this case the problem is to find a schedule that minimizes the objective function of agent A, while keeping the objective function of agent B less than a given threshold Q_B.

3 Structure of the non-dominated solutions

Schedule σ is called Pareto solution if there does not exist another solution that dominates it. On the basis of the studied problem properties, we want to determine the overall structure of the Pareto solutions. Some of these properties are generalization of classical single machine scheduling problems. In fact, as jobs of agent B are submitted to one common due date, it is easy to see that they

have to be sequenced on each machine according to their shortest processing time order, thus to minimize the number of tardy jobs of agent B. It can also be shown that on a given machine, the tardy jobs of agent B that belong to $\mathcal{J}^B \backslash \{\mathcal{J}^A \cap \mathcal{J}^B\}$, have to be scheduled last, otherwise the makespan of the agent A can be increased. So, we can write the following proposition.

Proposition 1 *If problem* $Pm|ND, d^B, \sum U_j^B \leq Q_B|C_{max}^A$ *admits a feasible solution, then it is possible to build an optimal solution such that on each machine we have:*

1. *Jobs of agent* B *appear in* SPT *order.*
2. *Tardy job* J_j *within* $\mathcal{J}^B \backslash \{\mathcal{J}^A \cap \mathcal{J}^B\}$ *is scheduled after the jobs of agent* A.

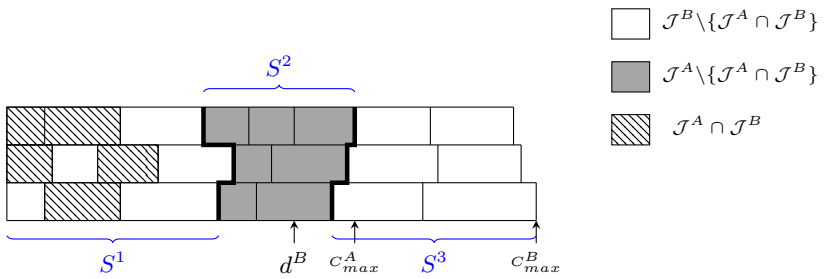


Fig. 1: Structure of non-dominated solution for the $Pm|ND, d^B|C_{max}^A, \sum U_j^B$.

Notice that the jobs of agent B should be scheduled before the common due date D^B. Thereby, to minimize the number of tardy jobs (maximize the number of early jobs) we only have to schedule jobs according to SPT rule. Therefore, there is an optimal schedule, if there exists, with structure similar to the one presented in figure 1: a sub-schedule S^1 containing jobs of agent B and sequenced according to SPT order on each machine, followed by a sub-schedule S^2 consisting of the remaining jobs of agent A, ended by a sub-schedule S^3 including the remaining jobs within agent B (some agent B jobs in S^3 may be early jobs).

4 Integer programming formulation

We present in this section two mixed integer linear programming formulations to solve the ϵ-constraint non-disjoint multi-agent scheduling problem. These models are tailored for the problem considering m identical parallel machines ($m > 0$). The first type of our proposed mathematical model is based on assignment of the jobs to machines, it is presented in section 4.1. The second formulation is based on the time indexed, it is presented in section 4.2.

To generate the set of strict Pareto solutions (Pareto front), we solve the problem with different values of Q_B. At the first iteration, Q_B is set to the upper bound of the objective function $\sum U_j^B$ (at worst case this value is equal to n_B). The obtained solution is denoted by $(\hat{\alpha}^A, \hat{\alpha}^B)$. We then solve the symmetric problem, denoted by $Pm|C_{max}^A \leq \hat{\alpha}^A| \sum U_j^B$. The obtained solution is hence strict Pareto solution, denoted by $(\hat{\alpha}^A, \hat{\alpha}^{B'})$. It is then added to the current set of non-dominated solutions. Next we iterate with $Q_B = \hat{\alpha}^{B'} - 1$; If no feasible solution is obtained then the procedure is stopped.

4.1 Assignment-based formulation

Let us consider the following decision variables:

- $x_{i,j}$ is a binary variable that takes value 1 if job J_j is scheduled on machine M_i; 0 otherwise.
- $y_{j,k}$ is a binary variable that is equal to 1 if job J_j is executed before job J_k on the same machine; 0 otherwise.
- z_j is a binary variable that is equal to 1 if job J_j is scheduled after its due date d_B; 0 otherwise.

We also need to define continuous variables: C_j is the completion time of job J_j and C_{max}^A is the value of the agent A objective function. We define HV some positive high value.

$$(MILP - Assign) \quad Min \quad C_{max}^A$$

$$s.t. \begin{cases} \sum_{i=0}^{m} x_{i,j} & = 1, \quad \forall J_j \in \mathcal{J} & (1) \\ C_j - \sum_{k=1}^{n} p_k \times y_{j,k} & \geq p_j, \quad \forall J_j \in \mathcal{J} & (2) \\ y_{j,k} + y_{k,j} & \leq 1, \quad \forall J_j, J_k \in \mathcal{J} & (3) \\ x_{i,j} + x_{i,k} - y_{j,k} - y_{k,j} & \leq 1, \quad \forall J_j, J_k \in \mathcal{J} \; i = 1, \ldots, m & (4) \\ x_{i,j} + y_{j,k} - x_{i,k} & \leq 1, \quad \forall J_j \in \mathcal{J} \; i = 1, \ldots, m & (5) \\ y_{j,k} + y_{k,l} - y_{j,k} & \leq 1, \quad \forall J_j, J_k, J_l \in \mathcal{J} & (6) \\ C_j - d_B - HV z_j & \leq 0, \quad \forall J_j \in \mathcal{J}^B & (7) \\ \sum_{J_j \in \mathcal{J}^B} z_j & \leq Q_B & (8) \\ x_{i,j}, y_{j,k}, z_j \in \{0,1\}, C_j \geq p_j & \forall J_j, J_k \in \mathcal{J} \; i = 1, \ldots, m & (9) \end{cases}$$

Constraints (1) impose that each job J_j should be assigned to one and only one machine. Constraints (2) mean that each jobs' completion time is at least

equal to the summation of the processing times of the jobs scheduled before it on the same machine plus its own processing time. Constraints (3) avert solutions that consider job J_j scheduled before job J_k, concurrently job J_k scheduled before job J_j. Constrains (4) impose that if two jobs are on the same machine, then one must be scheduled before another. Constraints (5) express that if J_j is executed on machine M_i and J_j is before J_k, than J_k must be scheduled on machine M_i also. Transitivity constraints are expressed by formula (6), which mean that if J_j is executed before J_k, and J_k is executed before J_l than J_j is executed before J_l. Constraints (7) fixe the values of the binary variables z_j: if it equals to 0 then we have $C_j \leq d^B$, otherwise the job is late and the constrain is still valid. Constraints (8) express the ε-approach bound and constraints. Constraints (9) are the integrity ones.

4.2 Time-based formulation

In this section, we propose a second mathematical formulation to solve the considered scheduling problem. We consider $s_{j,t}$ a new binary variables that are time indexed. $s_{j,t}$ takes as a value 1 if job J_j starts its processing at time t; 0 otherwise. Thereby, we have $n \times (T+1)$ binary variables, which is pseudo-polynomial. The general formulation is the following one.

$$(MILP - Time) \quad Min \quad C^A_{max}$$

$$s.t. \begin{cases} \sum_{t=0}^{T} s_{j,t} = 1, & \forall J_j \in \mathcal{J}^A & (10) \\ \sum_{J_j \in \mathcal{J}} \sum_{l=\max\{0,t-p_j+1\}}^{t} s_{j,l} \leq m, & \forall t = 0, \ldots, T & (11) \\ C^A_{max} - \sum_{t=0}^{T-p_j} (t+p_j) s_{j,t} \geq 0, & \forall J_j \in \mathcal{J}^A & (12) \\ \sum_{t=0}^{T-p_j} (t+p_j) s_{j,t} - HV z_j \leq d^B, & \forall J_j \in \mathcal{J}^B & (13) \\ \sum_{J_j \in \mathcal{J}^B} z_j \leq Q_B, & & (14) \\ s_{j,t} \in \{0,1\}, z_j \in \{0,1\}, & \forall J_j \in \mathcal{J} \quad t = 0, \ldots, T, & (15) \end{cases}$$

Constraints (10) impose that each job J_j starts at a given time t. Constraints (11) ensure that no more than m jobs are scheduled simultaneously, it avoids the jobs overlapping at any time t. Constraints (12) determine the makespan value of jobs of agent A. Constraints (13) fixe the values of the binary variables z_j: if $z_j = 0$ then we have $C_j \leq d^B$, otherwise job J_j is late and the constrain is still valid. Constraints (14) express the ε-approach bound. Constraints (15) are the integrity ones.

5 Polynomial heuristics

The studied scheduling problems are NP-hard (Sadi et al. 2014), we propose computationally efficient heuristics technical. To illustrate our approach, let consider the case of two machines ($m = 2$). Our approach can be easily extended to multiple processors environment. Our goal is to generate the set of non-dominated solutions, where the decision maker can evaluate the tradeoffs in the criteria. This is a posteriori approach in which the decision maker makes his choice only after a set of points is presented. Of course, our resolution methods can be also used as interactive approach where the decision maker specifies the maximum allowed number of tardy jobs of agent B and seek for the optimal value for the makespan of agent A.

Since makespan is equivalent to the decision problem involving a common deadline (i.e. whether a feasible schedule can be obtained such that all jobs finish before a common deadline) the set of non-dominated solutions can give the decision-maker important information on whether jobs for one set can be finished by a given time and the resulting compromise or effect on the number of tardy jobs for the other agent.

We consider the problem where the agent B objective function is bounded by Q_B. The goal is to obtain at least $n_B - Q_B$ early jobs. Since Property 1 specifies that the jobs of agent B have to be sequenced in their smaller processing time order, we consider in the following subset of jobs E containing the smallest $n_B - Q_B$ jobs of agent B. These jobs should be scheduled early so as to have a feasible solution. In order to obtain an approximate Pareto front, we solve the problem with different values of Q_B, such that $Q_B \in [0, n_B]$. Then after, dominated solutions are removed.

5.1 Heuristic 1

We focus on minimizing the makespan of agent A, using the heuristic *LPT-FAM* that is a well heuristic method for solving classical scheduling problem $Pm||C_{\max}$. It sorts the jobs in non-increasing processing times order (LPT) and assigns them to the first available machine (FAM).

The heuristic presented in this section is made up of three phases, where *LPT+FAM* is used (see Algorithm 1). It starts by scheduling jobs of E, followed by the remaining jobs of agent A from $\mathcal{J}^A \backslash \{\mathcal{J}^A \cap E\}$, and finishes by sequencing jobs of agent B not already scheduled. The remaining jobs of agent B have no influence on the makespan we can execute them at the end of any machine, but in order to optimize the number of tardy jobs it would be more efficient to use *LPT+FAM*. This heuristic is presented in Algorithm 1 and has $O(n_A log(n_A) + n_B log(n_B))$ time complexity.

5.2 Heuristic 2

In this section we develop the precedent heuristic by allowing the jobs reassignment. Indeed, instead of scheduling the jobs using *LPT-FAM*, this heuristic tries

Algorithm 1 LPT-FAM

Sort jobs in J^B according to SPT rule;
Set $E = \{J_1^B \ldots, J_{n_B - Q_B}^B\}$;
Schedule the jobs in E using *LPT-FAM*;
if at least one job is late **then**
  Stop; // this heuristic cannot find a feasible solution
else
  Schedule jobs in $\mathcal{J}^A \backslash \{\mathcal{J}^A \cap E\}$ using *LPT-FAM*;
  Schedule jobs in $\mathcal{J}^B \backslash \{\mathcal{J}^B \cap E\}$ using *LPT-FAM*;
  Return the resulting solution;

progressively to schedule jobs of $E \cup \mathcal{J}^A$, so as to minimize the makespan, and when a job of E is scheduled late, then it is rescheduled until is executed early. Alternatively, this heuristic cannot find a feasible solution. The complexity is still $O(n_A log(n_A) + n_B log(n_B))$, since the step 8 of Algorithm 2 can be done in constant time.

Algorithm 2 LPT-FAM with jobs rescheduling

Sort the jobs in J^B according to SPT rule;
Set $E = \{J_1^B \ldots, J_{n_B - Q_B}^B\}$;
Set $S = E \cup \mathcal{J}^A$ in LPT order;
Set $E^B = 0$; // the number of early jobs;
while $S \neq \emptyset$ and $E^B < n_B - Q_B$ **do**
  Schedule J_j using *LPT-FAM*;
  if J_j is late **then**
    Remove largest job J_k already scheduled, $J_k \notin E$;
    Put $S = S \backslash \{J_k\}$
    Reschedule J_j using *LPT-FAM*;
  else
    Set $E^B = E^B + 1$;
if $E^B = n_B - Q_B$ **then**
  Schedule jobs of $\mathcal{J}^A$ not already scheduled using *LPT-FAM*;
  Return the resulting solution;
else
  Stop; // This heuristic cannot find a feasible solution;

6 Pseudo-polynomial heuristics

In general, classical heuristics for solving scheduling problems are greedy algorithms based on priority rules. These approaches guaranty an efficient computational time but local optimality. In this section we try to overcome the gap by solving in the exact way the problem minimizing the makespan, that

is $P2||C_{max}(S)$ such as $S \subseteq \{\mathcal{J}^A \cup \mathcal{J}^B\}$. We will use the following pseudo-polynomial time algorithm based on dynamic programming (Blaziwicz et al. 2007):

Let P_j the makespan on machine M_i (i=1,2), and $F_j(P_1, P_2)$ is a recursive boolean function, it is equal to $true$ if jobs $J_1, \ldots, J_j$ of S can be scheduled on M_1 and M_2 in such away that each machine M_i is busy in interval $[0, P_i]$ $(i = 1, 2)$, and $false$ otherwise.

Applying $F_0(0, 0) = true$ and $F_0(P_1, P_2) = false\ \forall (P_1, P_2) \neq (0, 0)$. $F(j, t_j) = min(F(j-1, p_j - t_j); F(j-1, t_j) + p_j)$, the recursive function is given as follows:

$$F_j(P_1, P_2) = \big(F_j(P_1 - p_j, P_2) \wedge F_j(P_1, P_2 - p_j)\big),$$

$$\forall j = 1, \ldots, n,\ \forall P_i \in [0, UB], \forall i = 1, 2.$$

This dynamic programming algorithm (DP) determines the assignment of jobs to machines, which is sufficient to compute an optimal schedule that minimizes a makespan. The optimal makespan value is given by

$$C_{max}(S) = \min(\max_{\forall P_i \in [0,UB]}(\{P_1, P_2\} | F_j(P_1, P_2) = true)$$

This algorithm runs in $O(nUB^2)$ time, where UB is the upper bound of the makespan.

6.1 Heuristic 3

This heuristic start by scheduling the $n_B - Q_B$ first jobs of agent B, by applying the previous DP. If the optimal value of the obtained makespan is greater than d_B then there is no feasible solution for the problem. Otherwise jobs are scheduled early and they are followed by the remaining jobs of agent A and after by the remaining jobs of agent B, by always applying this DP. This algorithm runs in $O(n^2 + nUB^2)$ time, where UB is the upper bound of the makespan.

6.2 Heuristic 4

This heuristic is inspired from Heuristic 2, but instead of using *LPT-FAM* for assigning the jobs, it uses the dynamic program presented above. This algorithm runs in $O(nlogn + nUB^2)$ time, where UB is the upper bound of the makespan.

Algorithm 3 Heuristic based on dynamic programming

Sort the jobs in J^B according to SPT rule;
Set $E = \{J_1^B \ldots, J_{n_B - Q_B}^B\}$;
Optimally solve problem $P2||C_{max}$ considering only jobs of E by the DP
if $C_{max}(E) > d^B$ **then**
    Stop; // This problem has no feasible solution;
Optimally solve problem $P2||C_{max}$ considering only jobs of $(J^A \backslash \{J^A \cap E\})$ by the DP and taking into account no-availability machines at time zero (jobs of E have been already scheduled)
Optimally solve problem $P2||C_{max}$ considering only jobs of $(\mathcal{J}^B \backslash \{\mathcal{J}^A \cup E\})$ by the DP and taking into account no-availability machines at time zero (previous jobs have been already scheduled)
Try to schedule tardy jobs of agent B earlier without increasing makespan value by moving them to the left before d^B
Return the resulting solution;

Algorithm 4 LPT-FAM-Dynamic programming

Sort the jobs in J^B in SPT order;
Set $E = \{J_1^B \ldots, J_{n_B - Q_B}^B\}$;
Set $S = E \cup \mathcal{J}^A$ in LPT order;
Set $E^B = 0$; // the number of early jobs;
while $S \neq \emptyset$ and $E^B < n_B - Q_B$ **do**
    Schedule J_j using *LPT-FAM*;
    if J_j is late **then**
        Remove J_k the largest job already scheduled;
        Put $S = S \backslash \{J_k\}$
        Reschedule J_j using *LPT-FAM*;
    else
        Set $E^B = E^B + 1$;
if $E^B = n_B - Q_B$ **then**
    if some jobs of E^B are late **then**
        Stop; // This problem has no feasible solution;
    else
        Use DP to schedule the jobs of $\mathcal{J}^A$ not already scheduled;
        Use DP to schedule the jobs of $\mathcal{J}^B$ not already scheduled;
Return the resulting solution;

7 Computational results

The algorithms under study are coded in C language and executed on a workstation with a 2.4 Ghz Intel Core i5 processor with 8 GB of memory running Mac OS X Lion 10.7.5. Cplex version 12.6.2 was used to solve the mathematical model. The computation time limit has been fixed to 3600 seconds for each value (getting only one Pareto solution), where the Pareto fronts are determined as described in Section 4.

In this section, we compare Pareto fronts generated by each heuristic presented in Sections 5 and 6 to the exact Pareto fronts generated by the mathematical integer linear programs (presented in Section 4). The used instances are generated with different settings. Processing times, are generated using a discrete uniform distribution from 1 to 10. For each value of n such that $n \in \{10, 20, 30, 40, 50, 70\}$, thirty instances are generated. For each instance, the jobs are assigned randomly to the agents. We generate uniformly a value in the interval $[1, 3]$, for the value 1, the job belongs to $\mathcal{J}^A \backslash \{\mathcal{J}^A \cap \mathcal{J}^B\}$; 2, the job belongs to $\mathcal{J}^A \backslash \{\mathcal{J}^A \cap \mathcal{J}^B\}$ and 3, the job belongs to $\{\mathcal{J}^A \cap \mathcal{J}^B\}$.

We use three different metrics in order to evaluate the solutions quality:

1. First, for each exact and approximate solution, we calculate the cardinalities of the generated Pareto fronts. We denote by $|\mathcal{S}^*|$ the cardinality of the exact Pareto front $\mathcal{S}^*$, and by $|\mathcal{S}|$ the cardinality of the near Pareto front $|\mathcal{S}|$.

2. Using the cardinalities, we define $\%S$, this metric calculates the number of exact solutions generated by each heuristic, it is given by $|\mathcal{S} \cap \mathcal{S}^*|/|\mathcal{S}|$.

3. GD is a generational distance, it calculates the average of the minimum Euclidian distances between the approximate solutions and the exact ones. It is given by $GD = \frac{1}{|\mathcal{S}|}\left(\sum_{s_1 \in \mathcal{S}} \min_{s_2 \in \mathcal{S}^*} d_{s_1,s_2}\right)$ where d_{s_1,s_2} is the Euclidian distance between the element $s_1 \in \mathcal{S}$ and the element $s_2 \in \mathcal{S}^*$.

4. $\mathcal{H}$ is the *Hypervolume*, it calculates the area dominated by some front. In the following, we give the ratio of the area dominated by the approach pareto front and not by the exact Pareto front. Even when GD are small, they may be a poor indicator of the quality of the front $\mathcal{S}$. Therefore we introduced the metric $\mathcal{H}$ in order to estimate the repartition of the approach solutions on the exact Pareto front (see figure 2).

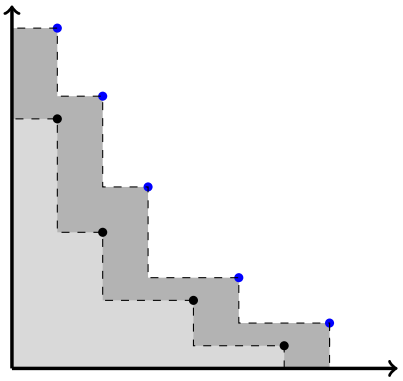


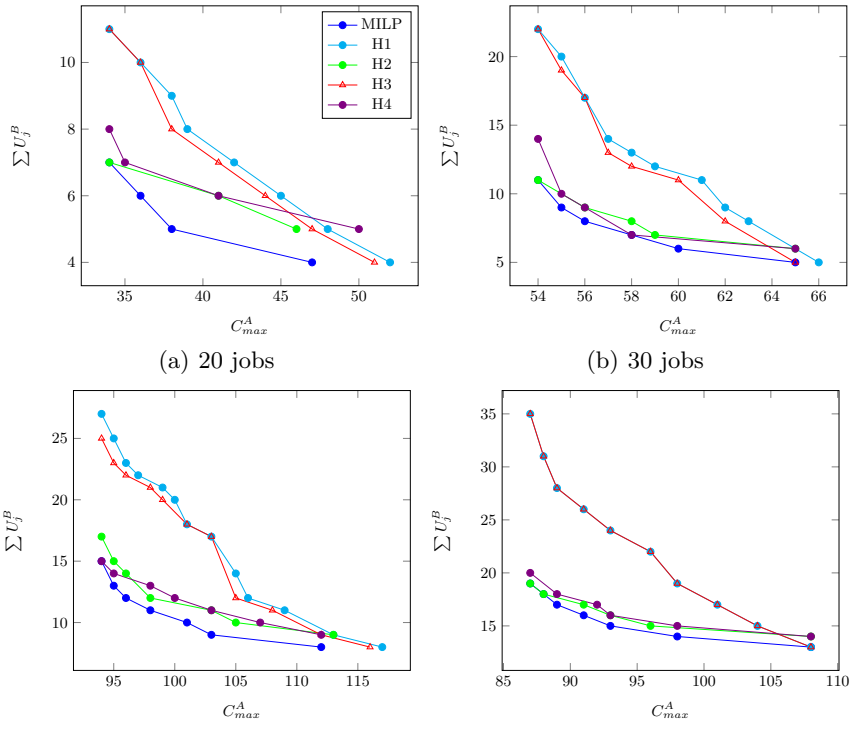
Fig. 2: Hypervolume measures representations

5. Furthermore, we calculate the CPU running time needed for each method.

In the following, we first analyze the results graphically in the objective space (see Figure 3) by considering four representative instances of obtained solutions. We also report complete results in tables 1, 2 and 3 with some discussions.

7.1 Graphical representation for four instances

In this section, we provide some graphical examples of the Pareto fronts generated by the exact and the heuristics methods. The chosen instances are fairly representative of the other instances. We can see in figures 3(a), 3(b), 3(c) and 3(d) five curves, such that the blue one gives the exact Pareto front, where the others are resulting from the heuristics. From this graphical, we can easily see that the green and purple curves are better that the others. They are the results of the heuristics 2 and 4.



(a) 20 jobs (b) 30 jobs

(c) 40 jobs (d) 50 jobs

Fig. 3: Example of the obtained Pareto fronts with four instances with 20, 30, 40 and 50 jobs.

In the next section we analyze and discuss all the results and give the average of the metrics previously introduced.

7.2 Result tables and discussions

In this section, we present the quantitative comparison results.

Table 1 gives the performances to the proposed mathematical formulations (MILP-Time and MILP-Assign).

	MILP-Time		*MILP-Assign*	
n	CPU	$\|\mathcal{S}^*\|$	CPU	$\|\mathcal{S}\|$
10	0.01	2.37	1.281	2.37
20	0.69	4.07	708.39	4.07
30	2.65	4.87	-	-
40	12.20	6.13	-	-
50	78.00	7.50	-	-
70	4664.16	9.766		

Table 1: Comparison of the performances of the MILPs.

About MILPs performances, we can easily see that the time indexed formulation is better than the assigned formulation, since its solves instances with 70 jobs in 1 hour and 18 minutes on average, the average time spent to obtain the whole Pareto front. Unfortunately, the assigned indexed formulation shows weaker results since its cannot solve instances with more than 20 jobs. We note that the time indexed formulation has a pseudo-polynomial binary variables, so the present performances are due to the small values of the processing times.

Table 2 represents the CPU average and the cardinality $|\mathcal{S}|$ of each heuristic.

Table 3 is dedicated to the comparison using the metrics $\%S$, GD and $\mathcal{H}$.

	H1		H2		H3		H4	
n	CPU	$\|\mathcal{S}\|$	CPU	$\|\mathcal{S}\|$	CPU	$\|\mathcal{S}\|$	CPU	$\|\mathcal{S}\|$
10	0.00	2.87	0.00	2.43	0.000	2.97	0.000	2.53
20	0.00	5.63	0.00	4.07	0.000	5.40	0.000	4.03
30	0.00	7.50	0.00	4.90	0.001	7.13	0.000	4.87
40	0.00	9.67	0.00	6.20	0.002	9.30	0.001	5.97
50	0.00	11.53	0.00	7.27	0.006	11.40	0.003	6.77
70	0.00	15.23	0.00	9.60	0.019	15.13	0.005	8.63

Table 2: Performance comparison using the CPU and $|\mathcal{S}|$.

	H1			H2			H3			H4		
n	$\%S$	GD	$\mathcal{H}$	$\%S$	GD	$\mathcal{H}$	$\%S$	GD	$\mathcal{H}$	$\%S$	GD	$\mathcal{H}$
10	29.83	0.86	24.21	36.94	0.68	17.90	33.28	0.82	25.66	28.33	0.89	21.11
20	14.40	1.39	37.39	28.85	0.91	11.52	18.85	1.39	36.31	12.08	1.24	19.95
30	6.09	1.86	40.94	25.19	1.06	10.37	7.08	1.89	40.54	9.82	1.44	21.23
40	1.78	2.02	41.13	27.04	1.12	7.86	1.84	2.06	41.58	9.66	1.43	20.07
50	1.01	2.47	44.52	37.08	1.01	5.65	1.21	2.48	44.50	14.50	1.53	17.69
70	0.70	3.09	45.30	35.74	0.96	4.77	0.70	3.10	45.29	14.13	1.39	10.51

Table 3: Performance comparison using the $\%S$, GD and $\mathcal{H}$.

Table 1 shows that the number of compromise solutions increases with the increase of the number of jobs. This is also true for the heuristics (see Table 2), where more compromise solutions are found using the heuristics 1 and 3. However, this does not necessarily mean that these solutions are better than the solutions found by the heuristics 2 and 4. In fact, we have seen in Figure 3, that more solutions are proposed by H1 and H3, but the corresponding curves are farther from the exact curve.

Although the criteria values are not normalized, the generational distances are still small for heuristics 2 and 4. In fact, for the instances with 70 jobs, solutions given by H2 are in average within distance 0.92 from the exacts Pareto solutions. And for the same instances solutions given by H4 are in average within distance 1.39 from the exact Pareto solutions. In the case of H1 and H3, the distances are slightly more significant. In fact, the values reach 3 for the instances with 70 jobs.

The generational distance values show that the compromise solutions proposed by the heuristics are close to the exact Pareto solution obtained by the MILPs. However, when analyzing the multi-criteria heuristic, we need to know whether the near Pareto solutions are well distributed in the criteria space or not. The *Hypervolume* can be a good gauge to measure the repartition of the solutions obtained by heuristics. Table 3, shows that H2 is the best method according to metric $\mathcal{H}$. Surprisingly, the values of H2 and H4 are decreasing with the increasing of the number of jobs.

8 Conclusion and perspectives

This paper tackled multi-agent scheduling problem, which is featured by two non-disjoint agents A and B, competing to perform their jobs on two identical parallel machines. Agent A aims at minimizing the maximum completion times of its jobs, whereas agent B tries to minimize the number of its tardy jobs.

The studied problem result from the application of the well known ε-constraint approach. This mono-criteria problem minimizes agent A makespan under a bounded constraint on agent B objective function. We proposed two types of mathematical programming formulation. The first one, is based on precedence decision variables, while the other is based on time indexing decision variables. The empirical results showed that when the processing times were in $[1, 10]$ with two identical machines, the time indexed formulation allows to obtain Pareto front for large instances.

In the second part of this research we proposed four heuristics for this NP-hard problem. The first and second heuristics are polynomial, they are based on LPT-FAM order. The third and fourth heuristics are inspired by the first two ones, instead of using an approach method for minimizing the makespan, it uses an exact algorithm based on dynamic programming. These heuristics are finally compared with the MILPs, using generational distance, hypervolume and cardinality metrics, the results showed that the heuristics 2 and 4 give better solutions. All proposed methods can be used to solve the case of m identical parallel machines. Thereby, we have conducted some other experimental results with 4 and 6 identical parallel machines and not presented here. In this case, the preliminary obtained results display the same conclusions.

For further research, we will propose a genetic algorithm starting from the solutions obtained by the heuristics. It would be also interesting to seek for a pseudo-polynomial time algorithm.

acknowledgment

This research was partially funded by the European Union through the Erasmus STA program.

References

Agnetis, A., Mirchandani, P., Pacciarelli, D., Pacifici, A., (2000). *non-dominated schedules for a job-shop with two competing users.* Computational and Mathematical Organization Theory, pp. 191–217.

Agnetis, A., Mirchandani, P., Pacciarelli D. and Pacifici, A., (2004). *Scheduling problems with two competing agents.* Operations research. 52, pp. 229–242.

Agnetis, A., Billaut, J.-C., Gawiejnowicz, S., Pacciarelli D., Soukhal, A., (2014). *Multiagent Scheduling, Models and Algorithms.* Springer. (271), pp. 401–415.

Baker, K. R., Smith, J. C., (2003). *A multiple-criteria model for machine scheduling.* Journal of Scheduling (6), pp. 7–16.

Balasubramanian, H., Fowler, J., Keha A., Pfund, M., (2009). *Scheduling interfering job sets on parallel machines.* European Journal of Operational Research (199), pp. 55–67.

Blazewicz J., Ecker K.H., Pesch E., Schmidt G., Weglarz J., (2007). *Handbook on scheduling: From Theory to Applications.* International handbooks on information systems. *Springer.*

Agnetis, A., Billaut, J.-C., Gawiejnowicz, S., Pacciarelli D., Soukhal, A., (2014). *Multiagent Scheduling, Models and Algorithms.* Springer. (271), pp. 401–415.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T., (2002). *A fast and elitist multiobjective genetic algorithm: NSGA-II.* IEEE Transactions on Evolutionary Computation 6, pp. 182–197.

Huynh Tuong, N., Soukhal, A., and Billaut, J. C. (2012). *Single-machine multi-agent scheduling problems with a global objective function.* Journal of Scheduling, 15(3), pp. 311–321.

Kung H. T., Luccio F. and Preparata F. P, (1975). *on finding the maxima of a set of vectors.* Journal of association for computing machinery, Vol 22, No 4, pp. 469–476.

C.T. Ng and T.C.E. Cheng and J.J. Yuan. (2006) *A note on the complexity of the problem of two-agent scheduling on a single machine.* Journal of Combinatorial Optimization, 12(4), pp. 387-394.

Peha J.M., (1995). *Heterogeneous-criteria scheduling: minimizing weighted number of tardy jobs and weighted completion time.* Computer operational research, Vol 22, No 10, pp 1089–1100.

Sadi, F., Soukhal, A., Billaut, J-C., (2014). *Solving multi-agent scheduling problems on parallel machines with a global objective function.* RAIRO - Operations Research, 48(2), pp. 225-269.