Two-machine job shop scheduling problem with two competing agents

1st AZERINE Abdennour
Faculté de Mathématiques
USTHB
Alger, Algérie
abdennourazerine@gmail.com

2nd BOUDHAR Mourad
Faculté de Mathématiques
USTHB
Alger, Algérie
mboudhar@yahoo.fr

3rd REBAINE Djamal
Département d'informatique
UQAC
Chicoutimi, Canada

***Abstract*—Multiple-agent scheduling has attracted growing interest in recent years. Most of previous research focused on the single-machine environment. However, some situations make this assumption impractical, since some jobs require to be executed on different machines. In this work, we address a two-machine job shop scheduling problem with two competing agents. The main objective is to minimize the makespan of the first agent subject to an upper bound on the makespan of the second agent. In some situation we will consider the total completion time as an objective function for the second agent. We investigate the complexity of several problems. Then, we focus on the two-machine job shop problem with two agents and two operations. To solve this problem optimally for small instance, we provide a mathematical model and a branch and bound algorithm. In addition, we propose an heuristic with different priority lists, and a genetic algorithm to find near optimal solutions. Finally, we present analysis of computational experiments.**

***Index Terms*—job shop, two-machine, makespan, total completion time, multi-agent, two agents.**

I. Introduction

[1] and [2] introduced new area of research in scheduling theory with the concept of multi-agent scheduling, where two agents or more need to process their jobs on common shared resources, each agent wants to minimize his own objective function. Two approaches have been used in terms of objective function, the weighted sum combination where each agent objective is assigned some weights and combined into a single objective to minimize. The second approach is the ϵ-constraint where we minimize the objective function of an agent subject to an upper bound on the objective function for other agents.

In our research, we study job shop scheduling problem with two competing agents which raises when we have two users or more, each one has his own subset independent jobs and an objective function that depends on the completion times of his own subset of jobs. The two-machine flow shop problem with two competing agents which is special case of the job shop problem has been studied by [2] and [3] using the ϵ-constraint approach and the weighted combination approach respectively, and proved as NP-hard in both cases.

By Examining the literature on the complexity results for the two-machine job shop problem with makespan criteria, we found that the two-machine flow shop with single criteria is polynomial solvable [4]. [5] extended this results to the two-machine job shop with two operations. As mentioned Previously, [2] has shown that the two-machine flow shop problem become NP-hard when we have two competing agents for the ϵ constraint approach, and [3] has shown this problem is also NP-hard for the weighted sum approach. [6] studied the two machine flow shop problem with two competing agents to minimize the weighted sum of the makespan and the total completion time and show that this problem is NP-hard since the two-machine flow shop problem with total completion time is NP-hard. While [7] focus on the m-machine proportionate flow shop problem, they considered three objective functions of the first agent : minimum maximum cost of all the jobs, minimum total completion time, and minimum number of tardy jobs subject to an upper bound on the value of the maximum cost for the second agent. They has shown that this problems are polynomials solvable by extending extending the associated solutions from single-machine to the m-machine proportionate flow shop. In our study, we focus on the two-machine job shop scheduling problem with two operations and two agents, this problem has not been studied before.

II. Problem description

We describe the problem as follows. We have n jobs to be processed on two shared machines and two agents A and B. Each agent has a subset of non-preemptive jobs $J_X = \{J_1^X, J_2^X, \ldots, J_{n_X}^X\}$, the two subsets are independent. Each job i has a processing time p_{ij}^X, a completion time C_{ij}^X for $j = 1, 2$ and $X = A, B$. Let C_{max}^X and $\sum_{i \in J_X} C_i^X$ be the makespan and the total completion time of agent $X = A, B$ respectively. Our objective is to minimize the makespan of the first agent subject to an upper bound on the objective function for other agent, this problem will be denoted as $J2||C_{max}^A : \gamma^B \leq Q$ where $\gamma^B \in \{C_{max}^B, \sum_{i \in J_B} C_i^B\}$ and Q is a fixed number.

III. Complexity results

In our work, we focus on the two-machine job shop problem with two competing agents. It is clear that this problem is NP-hard since the two-machine flow shop problem with two competing agents is already NP-hard. Consider a setting in which two agents compete to perform their jobs on two shared machines. The processing time of each job is equal on the two machines. The first agent process his jobs on M_1 then on M_2, while the second agent process his jobs on M_2 then on M_1. Assume that all the jobs are ready at the beginning, and we want to minimize the makespan of the first agent while the objective of the other agent does not exceeds a fixed value Q. We face two problems according to the objective of the second agent. First, we consider the makespan as performance measure. According to the three-field notation scheme this problem is denoted as : $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_i|C^A_{max} : C^B_{max}$. The second problem is when we consider total completion time as an objective function, this problem is denoted as : $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_i|C^A_{max} : \sum C^A_i$.

Theorem 1: $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_i|C^A_{max} : C^B_{max} \leq Q$ and $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_i|C^A_{max} : \sum C^A_i \leq Q$ are NP-hard.

Proof 1: The proof is based on the two-partition problem.

Now consider a second setting where the first agent has n_A jobs pass on the first machine then on the second machine, while agent B schedules his jobs on the second machine only. We want to minimize the makespan for the first agent subject to an upper bound on the makespan for the second agent, we suppose that $\sum_{i \in J_B} p^B_{i2} \leq Q$.

Theorem 2: $J2|M_1 \mapsto M_2, M_2, p^A_{ij} = p^A_i|C^A_{max} : C^B_{max} \leq Q$ is NP-hard.

Proof 2: The proof is based on the two-partition problem.

Now, consider the same previous problem, and suppose this time that the first agent wants to minimize the total completion time without exceeding an upper bound on the value of the makespan for the second agent who process his jobs on M_2 only.

Theorem 3: $J2|M_1 \mapsto M_2, M_2, p^A_{ij} = p^A_i|\sum_i C^A_i : C^B_{max} \leq Q$ is NP-hard.

Proof 3: The proof is based on the two-partition problem.

In the next sections, we will study the two-machine job shop problem with two competing agents and two operations for each job, the objective is to minimize the makespan of the first agent, subject to an upper bound on the value for the second agent.

IV. Exact methods

A. mathematical model

This approach introduced by [8], it relies on precedence variable z_{ijk} define as :

The decision variables :

$$z_{ijk} = \begin{cases} 1 & if\ job\ i\ precedes\ job\ k\ on\ machine\ j \\ 0 & otherwise \end{cases}$$

$$x_{ij} = \ start\ time\ of\ job\ i\ on\ machine\ j$$

Objective function :

$$min\ C^A_{max}$$

Constraints :

$$\begin{cases} x_{i\sigma^i_h} \geq x_{i\sigma^i_{h-1}} + p_{i\sigma^i_{h-1}}\ i \in J;\ h = 2, ..., m & ...(1) \\ x_{ij} \geq x_{kj} + p_{kj} - V.z_{ijk}\ i, k \in J; i < k; j \in M & ...(2) \\ x_{kj} \geq x_{ij} + p_{ij} - V(1 - zijk)\ i, k \in J; i < k; j \in M & ...(3) \\ C^A_{max} \geq x_{i\sigma^i_m} + p_{i\sigma^i_m}\ i \in J_A & ...(4) \\ Q \geq x_{i\sigma^i_m} + p_{i\sigma^i_m}\ i \in J_B & ...(5) \\ x_{ij} \in \mathbb{N}\ i \in J; j \in M & ...(6) \\ z_{ijk} \in \{0; 1\}\ i, k \in J; j \in M & ...(7) \end{cases}$$

1 . Precedence constraint. It ensures that all operations of a job are executed in the given order.
2, 3 . Disjunctive constraint. Ensure that no two jobs can be scheduled on the same machine at the same time.
4 . It ensures that the makespan is at least the largest completion time of the last operation of all A-jobs.
5 . It ensures that Q is at least the largest completion time of the last operation of all B-jobs.
6 . Start time of each job is integer number.
7 . Binary variable.
V : Must be assigned to a large value to ensure the correctness of the constraints.

B. Branch-and-Bound

In order to find an optimal solution for this problem, we use a branch-and-bound algorithm (B & B). In any branch and bound algorithm, we need to define three components : a branching strategy, a strategy of search, and a lower bound. Our branching strategy for the branch and bound algorithm is based on the following proposition :

Proposition 1: For $J2|O \leq 2|C^A_{max} : C^B_{max} \leq Q$ there exists an optimal schedule such that the jobs are divide in two subsets S_1 and S_2, each subset follows Jackson's rule and the last operation of S_1 On machine M_1 or M_2 belongs to agent A or B which determines his makespan.

Suppose that we have $Q < C^J_{max}$, (we can use the same method for $Q \geq C^J_{max}$). According to the previous proposition, we have : all the jobs's operations of agent B belong to S_1 and there are some jobs's operations of agent A in S_1 also and the rest operations are in S_2. The two sets follow

Jackson's rule. Since we know the order of operations in S_1 and S_2, this problem becomes how to partition the operations of each job i, $i = 1, ..., n_A$ of the agent A in the two subsets. For each job we will give a value equals to :

0 : if all operations of job i belong to S_2.

1 : if the first operation of job i belongs to S_1 and the second operation to S_2.

2 : if all operations of job i belong to S_1.

The obtained solution is as figure IV-B.

We will donate by C^J_{max} the makespan of all job following Jackson's rule, C^B_{max} the makespan of the B-jobs only, and C^{A-B}_{max} the makespan we obtain by Jackson's rule starting by A-jobs then B-jobs.

It is clear that if $C^B_{max} > Q$ then there exist no feasible solution.

1) Lower Bound: To find a lower bound LB, we apply Jackson's rule, we will have two cases. First, if $C^J_{max} \leq Q \leq C^{A-B}_{max}$ then it's clear that we have a feasible solution, so the value of the makspan can not exceed this value, and we know that the value of the makespan can not be lower then C^A_{max} as results $LB = C^A_{max}$ and we apply the branching procedure according to the $B-$jobs. Secondly, if $C^B_{max} \leq Q < C^J_{max}$ then we are sure that $C^A_{max} \geq C^J_{max}$ and we have $C^A_{max} \geq C^{B-A}_{max}$ so $LB = C^J_{max}$ and we apply the branching procedure according to the $A-$ jobs.

2) Searching Strategy: In our algorithm, we use an array initial to 0 with the size of n_A, represents the situation of each job, in our case we have three situations, jobs with index 0 belong to S_2, jobs with index 2 belong to S_1 only, and finally jobs that have operation in S_1 and the second operation in S_2 with index 1.

To make the search faster we calculate the some of processing time on each machine, then we re-order the jobs following the non-decreasing processing time order on the machine that has the smallest sum of processing time.

In our branching, we used depth-first search as a searching strategy. For the case $Q < C^J_{max}$ we start from the left with array contains zero only which means all the jobs of agent A belong to S_2, for the second case where $C^J_{max} \leq Q < C^{A-B}_{max}$ we start from the right with an array contains 2 which means all the jobs of agent B belongs to S_1. The algorithm stops if we get a solution with value equal to a lower bound or all the feasible solutions have been visited.

3) Parallel branch and bound: Here we present our parallel $B\&B$ implementation, we aim to accelerate the search by executing in parallel a sub-sequence of the branching nodes. Parallelization is implemented by creating sub-problem, each of them to be solved by concurrent parallelization. In our branching strategy we generate for each node three children (with value 0,1 and 2). For the parallel version we will fix subsequence of the vector, let r be a level in our tree, so we will have 3^r fixed nodes and new 3^r new sub-problem, for each sub-problem we will have new vector with r-fixed elements, and we apply sequential $B\&B$ procedure to the rest of the vectors (see eg below with $r = 2$). We start the branching algorithm in parallel. We use a global variable to save the best obtained solution. For each thread we calculate new solution, If during the computation we find better solution we update the current best global solution. We stop all the thread when we find an optimal solution according to the lower bound, otherwise we wait till all the thread finish their branching.

4) Computational results: Here, we compare these the mathematical model with the $B\&B$ algorithm in terms of CPU time. We use Cplex to code the mathematic model, while c++ was used to code the proposed $B\&B$. We run the test on PC with Intel(R) Core(TM) i7-2670QM CPU 2.20 GHZ and 8.00 GB RAM on windows operating system.

We generate the instance randomly, and the runs are terminated after 3600 s. We calculate how many times the proposed method obtain the optimal solution.

First, we set the running time for the proposed model and method to one hour then we count the number of time that we obtain optimal solution with different size of interval and number of jobs. the obtained results are show in IV-B4.

Size	Range	MIP1	$B\&B$
5	[1,10]	20	20
	[1,20]	20	20
	[1,50]	20	20
	[1,100]	20	20
	[10,20]	20	20
	[20,50]	20	20
	[50,100]	20	20
10	[1,10]	20	20
	[1,20]	20	20
	[1,50]	20	20
	[1,100]	20	20
	[10,20]	20	20
	[20,50]	20	20
	[50,100]	20	20
15	[1,10]	8	20
	[1,20]	8	20
	[1,50]	8	20
	[1,100]	8	20
	[10,20]	3	20
	[20,50]	5	20
	[50,100]	4	20

TABLE I
PERFORMANCE OF THE MIP AND B&B ALGORITHM

In the case of the small size number of jobs ($n = 5$ and $n = 10$), We can achieve optimal solution with all two methods. However, the run time has grown at an exponential rate when we increase the value of the jobs to 15 or more for the first model. In the other hand $B\&B$ algorithm give optimal solution in smallest running time.

Now, we set $n_A = 25$ and we calculate the value of the upper bound for the agent B as : $Q = C^B_{max} + (C^J_{max} - C^B_{max}) * x\%$. The obtained results is shown in IV-B4 and IV-B4.

x	0	10	20	30	40	50
time	0.15	0.23	3.82	42.15	206.82	519.84

x	60	70	80	90	100
time	805.65	912.10	933.64	936.48	0.09

TABLE II
THE RESULTS OF $B\&B$ WITH DIFFERENT VALUE OF Q

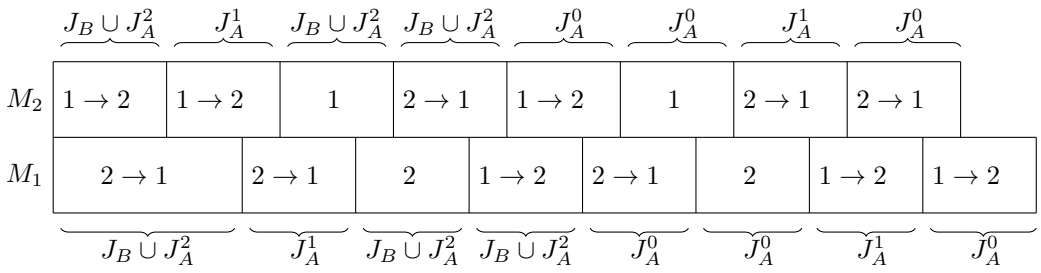

Fig. 1. scheduling the operations of each agent

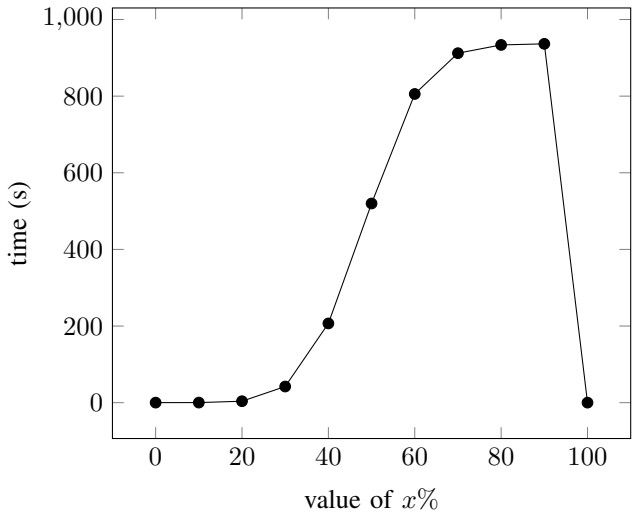

Fig. 2. The performance of $B\&B$ with different value of Q

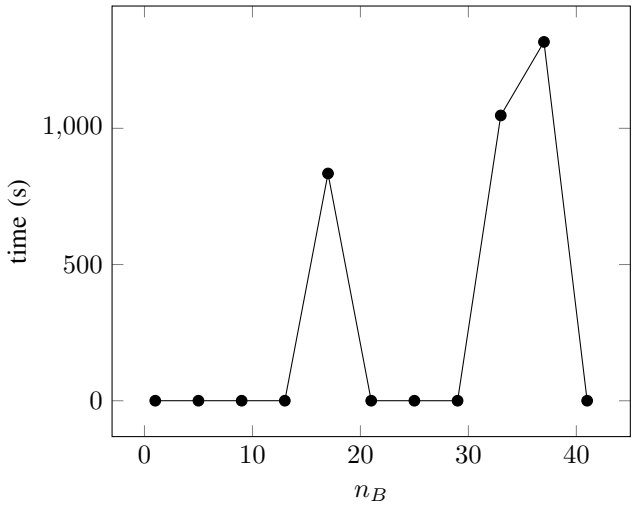

Fig. 3. The performance of $B\&B$ with different value of n_B

It is seen that the running time is increasing with the increase of the upper bound Q, so the instance becomes harder as the value of Q becomes closer to the makespan of all the jobs C^J_{max}. After reaching the maximum value C^J_{max} the running time becomes smaller.

Next, we address the performance of the $B\&B$ algorithm with different value of the second agent when we apply the branching for the first agent. We set $n_A = 25$ and the upper bound is fixed to $Q = C^B_{max} + (C^A_{max} - C^B_{max})/2$. The obtained results are shown in IV-B4 and IV-B4

n_B	1	5	9	13	17	21
time	0.1	0.14	0.13	0.15	834.23	0.13

n_B	25	29	33	37	41
time	0.09	0.09	1047.24	1317.2	0.13

TABLE III
THE RESULTS OF $B\&B$ WITH DIFFERENT VALUE OF n_B

Our results with the $B\&B$ suggest that there were no noticeable effects when we change the jobs numbers for the second agent if we are applying the branching for the first agent. Hence if we have instance with two two agents, it's better to fix an upper bound for the agent who has more jobs and minimize the makespan of the second.

V. HEURISTIC APPROACHES

The proposed exacts methods are often extremely time-consuming, So it is not sufficient to deal with large size problem instances. Therefore, we use heuristics to find feasible solutions, we propose an heuristic and we adapt a genetic algorithm to our problem.

A. Heuristics

We propose an heuristic 1 that depends on priority list, we call it H_1, we make use of the following priority lists :

R1 We choose the processing time of a machine k where $k = arg\{\min_j \sum_i p_{ij}, j = 1, 2\}$ and we apply the SPT rule.

R2 The order obtained from Jackson's rule on machine k where $k = arg\{\min_j \sum_i p_{ij}, j = 1, 2\}$.

The order obtained from Jackson's rule on machine k where $k = arg\{\max_j \sum_i p_{ij}, j = 1, 2\}$.

R4 Let $p_i = p_{i1} + p_{i2}$ and we apply the SPT rule following the new processing time.

R5 Let $p_i = p_{i1} + p_{i2}$ and we apply the LPT rule following the new processing time.

The basic idea of this heuristic is to start from a solution that has the B jobs only in S_1 and the A-jobs in S_2, then at each iteration we try to move an operation or two of job i for $i = 1, ..., n_A$ according to a predefined priority list, respecting the condition that the obtained solution still feasible and we decrease the value of the makespan.

Algorithm 1 Heuristic

1: Initial the vector solution to 0 and evaluate it.
2: i=0;
3: **while** $i < na$ **do**
4: Generate two new vectors by changing the i-th position to 1 and 2 according to the priority list;
5: evaluate the two obtained vectors;
6: save a feasible solution that has the best makespan;
7: i=i+1;
8: **end while**

B. Genetic algorithm

The most important implementation details of the GA algorithm include :

individuals : we define them as an array contains value 0, 1 or 2.

Initialization : we generate if possible $2 * n_A$ solution. Each solution contains one 1 or 2 then we add to the vector of solution a fixed number of 1 and 2 if it's possible. we add also a solution that contains 0 only.

Selection : for each individual, generate a random number, if it's less then a fixed given number (we set it to 0.3, the more we increase this value the more the algorithm consumes time), this individual becomes the first parent, we selected the second randomly.

Crossover : for each pair of parents that has been selected, we produce a new solution (child), we use two-point crossover that are picked randomly.

mutation : we generate a random priority list, and we try to increase the number of 1 and 2 if it decreases the value of the makespan. The mutation operator is executed with a probability. (We set the probability to perform mutation to 0.05).

Stopping condition : we use a fixed number of operations to stop the genetic algorithm.

Remark 1: If the obtained solution from crossover is not feasible, we decrease the number of 2 and 1 randomly until it becomes feasible.

Finally, we conduct computational tests in order to compare the heuristic and the genetic algorithm with lower bounds. We will use the heuristic with different priority lists to construct different solutions in order to improve the quality of the results. We set $n_B = 2 \times n_A$, and we fix the value of the upper bound to $Q = C^B_{max} + (C^J_{max} - C^A_{max})/2$.

Firstly, we try to generate random processing time from different intervals ($(p_{i1}, p_{i2}) \in$ { [1,20]× [1,20], [1,50]× [1,50], [1,20]× [1,50], [1,20]× [20,50], [20,50]× [20,50] }) with different size of problem. We generate 100 instance for each case, as a results 3000 instances has been generated. Our heuristic show high performance and obtained optimal solution in most cases. For meta-heuristic we get 100% times an optimal solution. We try to generate hard instance according to the problem $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_i|C^A_{max} : C^B_{max}$, we let $p_{i1} = p_{i2}$ and we generate the value of the upper bound Q between C^B_{max} and C^J_{max}. We test our heuristic and meta-heuristic with different numbers of jobs: 50, 100, 200 and 500. We calculate the relative deviance percentage as : $(H - LB)/H \times 100\%$, we present for each one of them the mean relative deviance and the relative deviance. We generate for each interval 100 random instances.

Size	Range		H1					GA
		Stats	R1	R2	R3	R4	R5	
50	[1,20]	mean	24,954	24,954	24,954	24,954	1,201	1,201
		max	24,961	24,961	24,961	24,961	1,848	1,848
	[1,50]	mean	24,982	24,982	24,982	24,982	1,302	1,302
		max	25,178	25,178	25,178	25,178	2,115	2,115
	[20,50]	mean	25,001	25,001	25,001	25,001	1,331	1,328
		max	25,807	25,807	25,807	25,807	25,250	2,201
100	[1,20]	mean	24,977	24,977	24,977	24,977	0,576	0,576
		max	24,980	24,980	24,980	24,980	0,913	0,913
	[1,50]	mean	24,990	24,990	24,990	24,990	0,657	0,657
		max	24,991	24,991	24,991	24,991	1,069	1,069
	[20,50]	mean	24,993	24,993	24,993	24,993	0,516	0,516
		max	24,993	24,993	24,993	24,993	0,718	0,718
200	[1,20]	mean	24,989	24,989	24,989	24,989	0,297	0,297
		max	24,990	24,990	24,990	24,990	0,469	0,469
	[1,50]	mean	24,997	24,997	24,997	24,997	0,303	0,303
		max	25,171	25,171	25,171	25,171	0,473	0,473
	[20,50]	mean	24,996	24,996	24,996	24,996	0,272	0,272
		max	24,997	24,997	24,997	24,997	0,352	0,352
500	[1,20]	mean	24,995	24,995	24,995	24,995	0,131	0,131
		max	24,996	24,996	24,996	24,996	0,181	0,181
	[1,50]	mean	24,998	24,998	24,998	24,998	0,148	0,148
		max	24,998	24,998	24,998	24,998	0,194	0,194
	[20,50]	mean	24,999	24,999	24,999	24,999	0,119	0,119
		max	24,999	24,999	24,999	24,999	0,141	0,141

TABLE IV
PERFORMANCE OF THE HEURISTIC WITH DIFFERENT PRIORITY LISTS AND THE GENETIC ALGORITHM.

It is seen that for H1 the LPT priority list outperform the rest of the priority lists, since they are using almost the same SPT order. the priority list that use the SPT rule outperform the lists that use LPT for the problem $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_i|C^A_{max} : C^B_{max}$. However, when we generate the instance randomly, the priority lists that are based on SPT rule outperform the lists based on LPT for H1. On the other hand, the meta-heuristic outperform the heuristic. Moreover, GA achieves small deviations to the lower bound for the problem $J2|M_1 \mapsto M_2, M_2 \mapsto M_1, p_{ij} = p_i|C^A_{max} : C^B_{max}$, and find optimal solution in most cases for the general problem with probability of 0.3 for crossover and 0.05 for mutation.

CONCLUSION

In this paper, we study two-machine job shop scheduling problem with two competing agents. We address the complexity of the proportionate two-machine job shop with two competing agents when the agent process their jobs in two different technological routes with the makespan and the total completion time using the ϵ-constraint approach.

In order to solve the two-machine job shop problem with two competing agents, we propose exact and heuristic methods. For the exact methods, we provide a mathematical model and a branch and bound algorithm. An heuristic based on different priority lists has been provided with a genetic algorithm.

The computational results showed that the mathematic model can solve instance when the number of jobs is fewer than 15. However, the B&B algorithm outperform the proposed mathematic model, it is able to solve optimally instance with 25 jobs for first agent regardless to the number of jobs of the second agent. Moreover, the proposed heuristic and genetic algorithm perform well in terms of efficiency and solution quality.

REFERENCES

[1] K. R. Baker and J. C. Smith, "A multiple-criterion model for machine scheduling," *Journal of scheduling*, vol. 6, no. 1, pp. 7–16, 2003.

[2] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici, "Scheduling problems with two competing agents," *Oper. Res.*, vol. 52, no. 2, pp. 229–242, Mar. 2004.

[3] W. Luo, L. Chen, and G. Zhang, "Approximation schemes for two-machine flow shop scheduling with two agents," vol. 24, 10 2012.

[4] S. Johnson, "Optimal two and three stage production schedules with set-up time included," vol. 1, pp. 61 – 68, 03 1954.

[5] J. R. Jackson, "An extension of johnson's results on job idt scheduling," *Naval Research Logistics Quarterly*, vol. 3, no. 3, pp. 201–203, 1956.

[6] B. Fan and T. Cheng, "Two-agent scheduling in a flowshop," *European Journal of Operational Research*, vol. 252, no. 2, pp. 376 – 384, 2016.

[7] B. Mor and G. Mosheiov, "Polynomial time solutions for scheduling problems on a proportionate flowshop with two competing agents," *Journal of the Operational Research Society*, vol. 65, no. 1, pp. 151–157, Jan 2014.

[8] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960.