International Journal of Industrial Engineering, 28(6), 577-593, 2021

A BRANCH-AND-BOUND ALGORITHM FOR TWO-COMPETING-AGENT SINGLE-MACHINE SCHEDULING PROBLEM WITH JOBS UNDER SIMULTANEOUS EFFECTS OF LEARNING AND DETERIORATION TO MINIMIZE TOTAL WEIGHTED COMPLETION TIME WITH NO-TARDY JOBS

Tugba Danaci[1, 2, *] and Duran Toksari[3]

[1]Fatma Senses VS Business Administration Department
Kırıkkale University
Kırıkkale, Turkey
[*]Corresponding author's e-mail: tugbadanaci@kku.edu.tr

[2]Forte Information Communication Technologies and Defense Industry Inc.
Ankara, Turkey

[3]Department of Industrial Engineering
Erciyes University
Kayseri, Turkey

Recent scheduling studies focus on variable job-processing times and multi-agent problems simultaneously, but none of them studied with jobs under the simultaneous effect of learning and deterioration. This paper studies a two-competing-agent single-machine scheduling problem with jobs under simultaneous learning and deterioration effect. The goal is to find an optimal solution to minimize the total weighted completion time for the first agent, subject to the restriction that no tardy job is allowed for the second agent. According to our current literature knowledge, this paper will be the first one with these specifications. For this problem, a two-stage methodology is developed in the study. In the first stage, heuristics are proposed to find the near-optimal solution of which is used as input for the second stage. For the second stage, a branch-and-bound algorithm along with several dominances and a lower bound is developed to find the optimal solution. Computational experiments are provided to further measure the performance of the proposed algorithms.

Keywords: Multi-agent scheduling, Variable processing times, Learning effect, Deterioration effect, Branch-and-bound algorithm

(Received on July 25, 2021; Accepted on December 31, 2021)

1. INTRODUCTION

In classical scheduling problems, processing times of jobs do not vary and are stable. However, in the real world, process times can vary because of job-repeating or the number of resources used. In literature, change in processing times is studied under three main groups: time-dependent processing times, position-dependent processing times, and controllable processing times (Agnetis *et al.*, 2014).

The processing time of the jobs with time-dependent processing times is a function of the time when the job is started to be processed. The processing time of the jobs with position-dependent processing times is a function of the position or row the job is started to be processed. Sometimes the processing time could be varied by changing the number of resources used. These kinds of jobs are known as jobs with controllable processing times.

Time-dependent processing times result from job characteristics and are divided into two groups: non-decreasing and non-increasing time-dependent processing times. Jobs with non-decreasing time-dependent processing times, also known as jobs under deterioration effect, and the later the job is processed, the more the processing time will increase. Extinguishing fire could be given as a sample. Jobs with non-increasing time-dependent processing times are also known as jobs under shortening effect, and the later the job is processed, the more the processing time will decrease. Jobs like annealing could be given as a sample.

Position-dependent processing times result from worker experience and are separated into two groups: non-decreasing and non-increasing position-dependent processing times. Jobs with non-increasing position-dependent processing times are also known as jobs under the learning effect. Jobs with non-decreasing position-dependent processing times are also known as jobs under the aging effect.

The processing times of some jobs can be varied by the amount of the resource used, and these kinds of jobs are known as jobs with controllable processing times. Jobs under project management topics could be given as samples.

Detailed grouping for the jobs with varying process times is given in Figure 1.

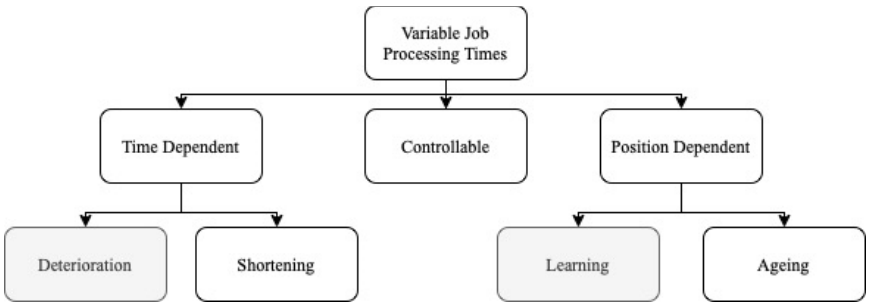


Figure 1. Groups of Jobs with Variable Processing Times (Agnetis *et al.*, 2014)

In this paper, scheduling problems for jobs under the simultaneous effect of deterioration and learning will be studied. In other words, the later the job is processed, the more the process time will decrease because of learning effect, while on the other hand, the more it will increase because of the deterioration effect.

Jobs under the simultaneous effect of learning and deterioration are encountered, especially in the production processes of the defense industry and metal industry. Jobs in the process of coating warheads with polymer liner before they are filled with explosives can be given as a sample. As time passes, the chemical properties of the polymer material used begin to deteriorate. Due to the deterioration of chemical properties, the processing time of coating gets increases, and the quality level decreases. However, on the other hand, the process shortens due to both the learning of the workers and the temperature reaching the desired level. As another example, hot forging can be given. The list can be extended.

The literature for multi-agent scheduling problems can be divided into two main groups based on jobs processing-time-feature: problems with non-variable processing times and problems with variable processing times. The works of Agnetis and others (Agnetis *et al.*, 2000; Agnetis *et al.*, 2001) and Baker & Smith (Baker *et al.*, 2003) may be considered as pioneer studies for multi-agent scheduling problems. While examining the literature studies, papers that studied the same objective function as ours - minimization of the total weighted completion time- are reviewed.

One of the papers that studied scheduling problems of jobs with non-variable processing times belongs to Soltani and others (Soltani *et al.*, 2010). They studied to minimize the total weighted completion time for the first agent while minimizing maximum lateness for the second agent. To improve the parameters, Taguchi methods were used. They developed a genetic algorithm and a hybrid kangaroo algorithm to solve the problem. For small-size problems, the hybrid kangaroo method gave better results. On the other hand, the genetic algorithm gave better results for big-size problems.

In another paper, Cheng and others studied jobs with non-variable processing times and release dates (Cheng *et al.*, 2013). They tried to minimize the total weighted completion time for the first agent with a threshold for the maximum lateness value of the second agent. They proposed a two-stage methodology. In the first stage, a simulated annealing method was used to obtain an initial solution. In the second stage, they used branch and bound algorithms. Via this method, the optimal solution could be obtained for less than 24 jobs with a 0,5% mean error. The same problem is studied by Yin and others too (Yin *et al.*, 2015 (3)). They also proposed a two-stage methodology: the honeybee algorithm for the initial solution and the branch and bound algorithm for the optimal solution. With the method, the optimal solution could be obtained for less than 24 jobs.

Lee and Wang (2014) studied the scheduling of jobs with non-variable processing times for three agents. They tried to minimize the total weighted completion time for the first agent with a threshold for the total completion time value of the second agent while the maintenance process of the third agent should have been finished in a predefined time interval. They also proposed a two-stage methodology. In the first stage, to obtain an initial solution, a local search and genetic algorithm were used. In the second stage, they used branch and bound algorithms. The method was useful for problems with 24 or fewer jobs.

Lee and others tried to minimize the total weighted completion time not only for the first agent but also for the second agent too (Lee *et al.*, 2009). They reduced the problem to the MOSP (multi-objective shortest path) problem and developed a solution approach with polynomial time.

Another paper belongs to Choi and Chung (2014). They tried to minimize not only the total weighted completion time but also the weighted number of tardy jobs of the first agent subject to the restriction of the weighted number of just-in-time jobs of the second agent. They studied the complexity level of the problem.

Zhang and others studied three-agent scheduling on a single machine in which the criteria of the three agents are to minimize the total weighted completion time, the weighted number of tardy jobs, and the total weighted late work, respectively (Zhang et al., 2020). Since the problem was NP-hard, they studied the problem under the restriction that the jobs of the first agent have inversely agreeable processing times and weights. The smaller the processing time of a job was, the greater its weight was. They presented a pseudo-polynomial-time algorithm to find the Pareto frontier.

The papers mentioned above are not the only ones that studied multi-agent scheduling problems with non-variable processing times. Feng and others tried to minimize the total completion time and rejection cost for the first agent while the second agents weighted the number of late jobs (Feng *et al.,* 2015). Agnetis and others tried to minimize max weighted completion time (Agnetis *et al.,* 2009). Some studied about tardiness and earliness values (Lee *et al.,* 2013; Elvikis *et al.,* 2011; Li *et al.,* 2020; Mor *et al.,* 2010; Li *et al.,* 2015). The list could be extended for many different objective functions and the number of agents.

When examining the studies focused on variable job processing times and multi-agent problems simultaneously, papers can be grouped according to the effects under which the processing time of jobs varied.

In some papers, jobs that belong to all agents are under the effect of deterioration. Yin and others investigated the complexity and solvability of problems for one agent while keeping an upper limit for maximum earliness cost for the other agent (Yin *et al.,* 2015). The objective functions for the first agent studied were maximum cost, total weighted completion, discounted total weighted completion, maximum earliness cost, total earliness, and total weighted earliness. In another study, the linear aging effect with job-dependent aging ratios for both agents is studied (Choi, 2015). It was tried to minimize the total weighted completion time for the first agent while keeping makespan below the upper limit for the second agent. It was proved the problem is NP-hard, obtained three initial solutions using SA and then used branch bound algorithm. Li and Hsu studied the log-linear learning effect for jobs of both agents and tried to minimize total weighted completion time for both agents (Li *et al.,* 2011). In each iteration, they tried to keep the makespan value of the other agent less than a certain value. They used the genetic algorithm for the initial solution and then used the branch and bound algorithm. The system worked fine up to 18 jobs. In another study, both agents' past sequence-dependent learning effect is studied (Wu, 2014). The total weighted completion time was aimed to be minimized for both agents, and the makespan value was limited below a certain value for the other agent. Solutions for up to 20 jobs were obtained with less than 0.076% error using SA and branch and bound algorithm.

Papers with the same objective functions of this study-minimization of total weighted completion time for the first agent and no-tardy jobs for the second agent- are (Cheng *et al.,* 2011; Cheng *et al.,* 2011(2); Lee *et al.,* 2010; Wu *et al.,* 2013 (2); Wu W.H. *et al.,* 2013; Wu, 2013). Cheng and others studied the log-linear learning effect for the first agent and the log-linear aging effect for the second agent simultaneously (Cheng *et al.,* 2011). SA was used for the initial scheduling and branch, and the bound algorithm was used for the optimal solution. A solution for up to 16 jobs was obtained using branch and bound algorithm, and by also using heuristics, a solution for up to 50 jobs was obtained with an error of less than 0.5%. In another paper, the past sequence-dependent learning effect for both agents were studied (Cheng *et al.,* 2011(2)). It was mentioned that if there were too many jobs with very long process times, the actual processing time would converge to 0 when the general learning affect function is used. And this would not be realistic and possible in real life. Therefore, the learning function with a control parameter was used. SA heuristics was utilized for the initial solution. When SA was constructed, jobs of agent A were aligned using WPST and jobs of agent B were aligned using EDD, and results were improved using neighbor search. SA1 was obtained using backward shifted reinsertion, SA2 was obtained using forward-shifted reinsertion, and SA3 was obtained using pairwise interchange. Initial solutions were used as inputs for branch and bound algorithms. The method worked fine for up to 20 jobs. Lee and others used three heuristics for the initial solution and used a branch and bound algorithm for the optimal solution for jobs with linear deterioration (Lee *et al.,* 2010). For the calculation of the lower bound in branch and bound algorithm, they considered the weights of the jobs for the first agent and deadlines for the second agent. The algorithm worked for up to 20 jobs, and the mean error was less than 1.64%. Wu and others studied jobs with log-linear learning for both agents (Wu *et al.,* 2013). The initial solution was obtained using four different heuristics, and then, branch and bound algorithms were used. Past sequence-dependent learning effect for both agents was studied in (Wu *et al.,* 2012) and (Wu, 2013). To solve the problem, three different simulated annealing algorithms with branch and bound algorithms were developed (Wu *et al.,* 2012): (i) For the first simulated annealing algorithm jobs were sequenced randomly; (ii) For the second SA algorithm jobs of the first agent were sequenced by EDD and second agent's jobs were sequenced by SPT rule; (iii) For the third SA algorithm, jobs of the first agent were sequenced by EDD and second agent's jobs were sequenced by WSPT rule. To have a better solution, the results of the second and third SA algorithms were applied NEH algorithm. In (Wu, 2013), ten different ant colony algorithms were used to have initial solutions, and branch and the bound algorithm were used to optimize the initial solutions. The method developed was useful for the problems containing 15 or fewer jobs. The list could be extended for many different objective functions and the number of agents.

On the other hand, based on the literature knowledge, there is not any paper that combined multi-agent scheduling problems with jobs under log-linear learning and linear deterioration effects simultaneously. The problem is NP-hard, and as observed from the literature review, the approaches adopted to solve problems include more than one step. In this paper, a two-stage methodology is developed. In the first stage, a heuristic is proposed to find the initial solution of which is used as input for the second stage. For the second stage, a branch-and-bound algorithm along with several dominances and a lower bound is developed to find the optimal solution. Lastly, computational experiments are provided to further measure the performance of the proposed algorithms.

2. PROBLEM DESCRIPTION

The scheduling problem under consideration is a single-machine scheduling problem with two competing agents, A and B. There is a total number of $n = n_A + n_B$ jobs of which n_A jobs belong to agent A and n_B Jobs belong to agent B. Jobs are ready to be processed at time t=0. There is no permission for interruptions. Jobs are under the effect of position-based learning and time-based deterioration effect simultaneously. While the learning effect shows itself log-linearly, the deterioration effect shows itself linearly. The processing time for jobs under effects is formulated as:

$$p_j(r,t) = (p_j + \beta t) r^{\alpha} \ .$$

The objective function to minimize is the total weighted completion time of agent A under the restriction that no tardy jobs are allowed for agent B. The problem studied is:

$$1/CO; p_j(r,t) = (p_j + \beta t) r^{\alpha}; \sum_{j=1}^{n} U_j^B(S) = 0 / \sum_{j=1}^{n} w_j^A C_j^A(S) \ .$$

The notations and variables are used throughout this paper, and then the problem is defined formally. The defined variables and notations are listed in Table 1:

Table 1. Variables and Notations

J_A	The job set for agent A
J_B	The job set for agent B
$J = J_A \cup J_B$	The set for the sum of jobs
n_A	The number of jobs for agent A
n_B	The number of jobs for agent B
$n = n_A + n_B$	The total number of jobs
p_j	Normal processing time for job j
d_j	The due date for job j
w_j^A	Weight for job j of agent A
$w_j^B = 1$	Weight for job j of agent B
θ	Interpolation coefficient
β	Deterioration coefficient ($\beta \geq 0$)
α	Learning coefficient ($\alpha \leq 0$)
r	Position in the schedule
t	The time when job j is started to be processed
$p_j(r,t) = (p_j + \beta t) r^{\alpha}$	The processing time when job j is started to be processed in position r and time t
S	The present schedule
$C_j^A(S)$	The completion time of the job j of agent A in the schedule S
$C_j^B(S)$	The completion time of the job j of agent B in the schedule S
$T_j(S) = max\{0, C_j(S) - d_j\}$	The value of tardiness for the job j in the schedule S
$U_j(S) = \begin{cases} 1 if T_j(S) > 0 \\ 0 if not \end{cases}$	It's 1 if the job j is tardy in the schedule S, it's 0 if the job j is not tardy
$OFV(S)$	Objective Function Value of agent A in schedule S
$k = k_A + k_B$	The number of assigned jobs to the node in branch and bound algorithm

k_A	The number of unassigned jobs belonging to agent A to the node in branch and bound
k_B	The number of unassigned jobs belonging to agent B to the node in branch and bound
$\hat{C}_{[r,t]}$	The completion time of the job when started to be processed in position r and time t
UB	Upper bound value

The problem is NP-hard. So, to solve the problem, a methodology based on branch and bound algorithm (BBA) is developed. Firstly, a heuristic is developed to get an initial solution. The objective function value obtained from the initial solution will be used as the upper bound value of the branch and bound algorithm.

3. HEURISTICS FOR INITIAL SOLUTION

First, a basic heuristic is developed to obtain the initial solution with the help of two well-known methods, WSPT and EDD. In a single machine - single agent environment, if the objective is to minimize the total weighted completion time, the jobs are sorted with respect to adjusted processing times (obtained by dividing processing times by weights) from smaller to higher. The method is known as WSPT (weighted shortest processing time). The other method used is EDD (earliest due date), and it is used in a single machine - single agent environment if the objective is to schedule jobs so that there will not be any tardy jobs.

The efficiency of the branch-and-bound algorithm depends on the initial solution. The objective function value of the initial solution is used as the first upper bound value for the branch and bound algorithm. Three separate job sets will be used for the initial solution: unassigned-jobs set, tardy-control set, and partial schedule S.

Unassigned-jobs set includes all the jobs that are not assigned to partial schedule S. Processing times of jobs for each position in the set is calculated. New processing times of jobs belonging to agent A is calculated by $pd_j(r, t) = (p_j+\beta t)r^{\alpha}/w_j$ where *r* shows the position in the schedule and t shows the time when the job is started to be processed. For jobs belonging to agent B, new processing times are calculated by $pd_j(r, t) = \theta(p_j+\beta t)r^{\alpha}+(1-\theta)d_j$ where θ demonstrates the weight between the due date and real processing time. Jobs are sorted in ascending order by $pd_j(r, t)$.

The second set is the tardy-control set. It includes jobs belonging to Agent B and not assigned to partial schedule S. The jobs in the tardy-control set should not become tardy and sorted in ascending order by deadlines.

Each job in the unassigned-jobs set will be assigned to the partial schedule. In each iteration, we must ensure that none of the unassigned jobs belonging to agent B fall into tardy status when the assignment occurs. To check tardiness status, assign the jobs in the tardy-control set in the same order to the position right after the job that will be scheduled and calculate the variable processing times of the jobs. If any of the unassigned jobs belonging to agent B fall into tardy status, the first job in the set will be scheduled instead of the previous one.

The pseudo-code of the heuristics developed to obtain the initial solution is given in Algorithm 1 and the flow chart in Figure 2.

Algorithm 1. Heuristics for Initial Solution

Input: $J=\{J_1,J_2,...,J_n\}$: all jobs
S={…}: partial schedule
t=0 : completion time
For (r←1**ton**), **do**

1. Set new processing times for each job with respect to the following: If $J_j \in J_A \cap J$ then $pd_j(r,t)=(p_j+\beta t)r^{\alpha}/w_j$; If $J_j \in J_B \cap J$ then $pd_j(r, t) = \theta(p_j+\beta t)r^{\alpha}+(1-\theta)d_j$ where θ demonstrates the weight between the due date and real processing time for agent B.
2. Select the job with min{pd(u)} value for $J_u \in J$.
3. Calculate the new processing time $pd_j(r, t)$ and completion time $\hat{C}_{[r, t]}=pd_j(r, t) + t$ of J_u as assigned to position *r* in partial schedule S.
4. Check whether the unscheduled jobs belonging to the agent B gets tardy by using Algorithm 2, in case the job J_u is added to the partial schedule S.
5. If number-of-tardy-jobs=0, assign the job J_u to r^{th} place into the partial schedule S and take this job out of the set J. Go to step 9.
6. Else, assign the job with the smallest due date in set tardy-control={j|j∈J_B∩J,j∉S}and remove the job from the set and apply Algorithm 2.

7. If number-of-tardy-jobs=0, assign the job r^{th} place into the partial schedule S and take this job out of the set J. Go to step 9.
8. Else, the solution is INFEASIBLE.
9. Calculate the completion time of the partial schedule and update *t.*
10. r = r+1

End

Output $S=\{J_{[1]},J_{[2]},\ldots,J_{[n]}\}$

Output $UB = OFV(S) = \sum w_j^A C_j^A(S)$

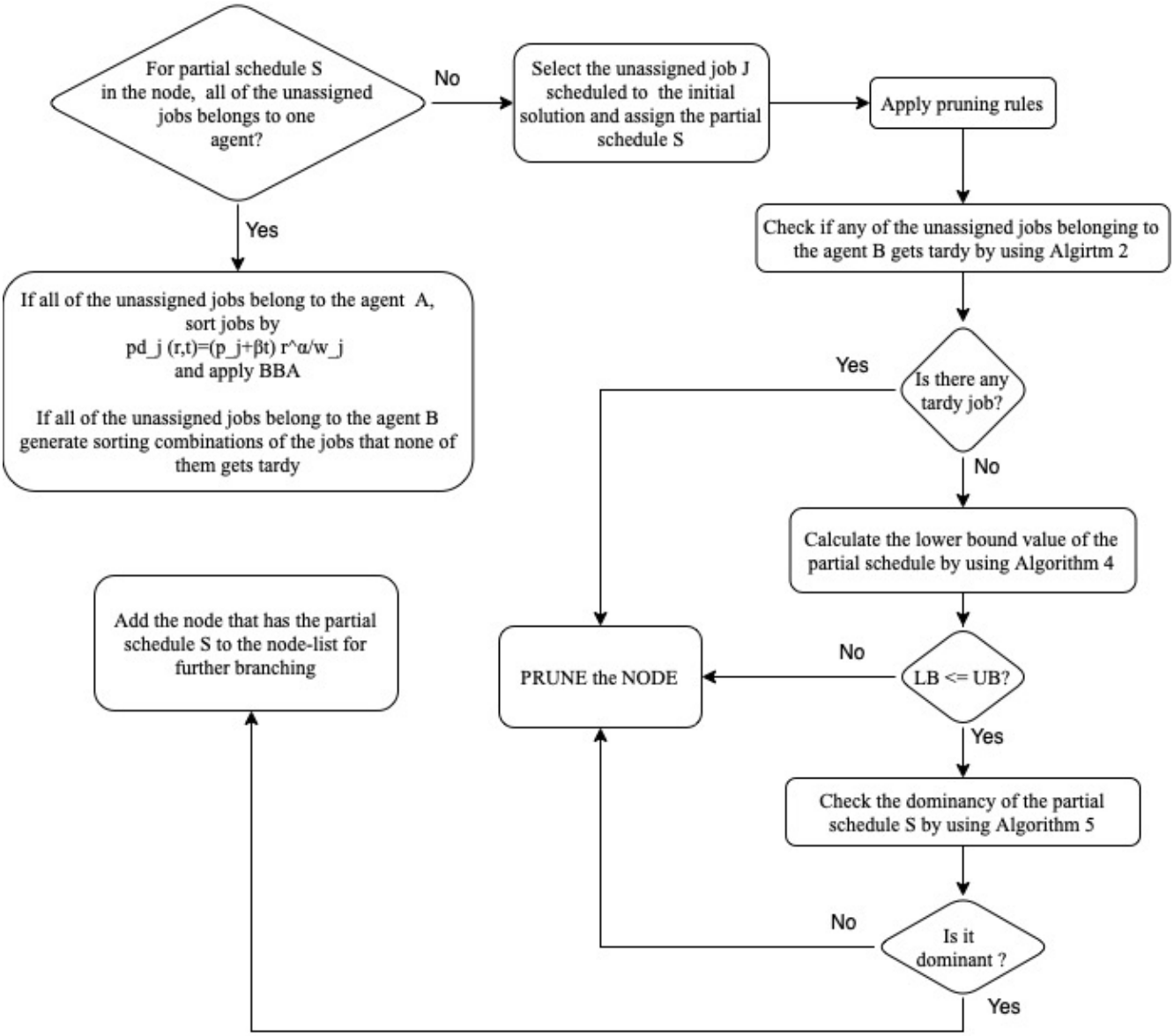


Figure 2. Flowchart for Initial Solution

Algorithm 2. Tardy control algorithm

Input: $J = \{J_1, J_2, \dots, J_n\}$
$S = \{\dots, J[r-2], J[r-1]\}$
$tardy\text{-}control = \{j \mid j \in J_B \cap J, j \notin S\}$
$number\text{-}of\text{-}tardy\text{-}jobs = 0$
1. Sort the jobs in set "tardy-control" in ascending order by deadlines.
2. Schedule the "tardy-control" set to the partial schedule S right after the job J_u.
For $j \in tardy - control \cap S$ **do:**
1. Calculate the completion time of job j
if $C_j(S) > d_j$ **:**
$number - of - tardy - jobs = +1$
Break
End
Output number_of_tardy_jobs

4. THE BRANCH AND BOUND ALGORITHM

The problem studied in this paper is NP-hard. Therefore, a branch-and-bound algorithm might be a good way to derive the optimal solution. In this section, several pruning rules with dominance properties are developed to speed up the algorithm and reduce the searching scope. Besides, as a branching strategy, a dept-search methodology is used.

4.1 Pruning Rules

To prune the nodes, some rules are developed and applied:

- ***Pruning by bound:*** The first rule is that the calculated lower bound value of the partial schedule in the node shall be less than or equivalent to the best known upper bound value. Otherwise, the node is pruned. As the first upper bound value, the objective function value of the initial solution is assigned. Calculation details of the lower bound value of each node are given in subtitle 4.2 and in Algorithm 4.
- ***Pruning by feasibility:*** The study aims to minimize the total weighted completion time for the first agent while no tardy job is allowed for the second agent. If any of the unassigned jobs of the second agent get tardy by the partial schedule in the node, the solution will be infeasible, so the node will be pruned. Checking the tardiness of unassigned jobs to the node is detailed in Algorithm 2.
- ***Pruning by optimality:*** The study aims to minimize the total weighted completion time for the first agent while there is no tardy job for the second agent. If all the jobs belonging to the first agent is assigned to the node, the node is pruned, and the remaining jobs which are belonged to the second agent are assigned by ascending order by deadlines. If all the unassigned jobs just belong to the first agent, the node is pruned. The unassigned jobs are sorted by $pd_j(r,t) = (p_j + \beta t)r^{\alpha}/w_j$, values and add to the node. By this method, the number of nodes that will be created will decrease dramatically compared to the classical branching method.
- ***Pruning by dominancy:*** To reduce the searching scope non-dominant nodes are pruned. Dominance properties are detailed in subtitle 4.3 and in Algorithm 5.

The pseudo-code of the branch and bound algorithm is given in Algorithm 3. The flowchart is given in Figure 3.

Algorithm 3. Branch and Bound Algorithm

Input: $J = \{J_1, J_2, \dots, J_n\}$
node-list = {Partial schedules obtained by branching nodes in branch and bound algorithm}
UB = objective function value of the schedule obtained by using Algorithm 1
For $(r \leftarrow 1 \boldsymbol{to} n)$ **do:**
For a node in node-list, do:
1. For each partial schedule, create the set of "unassigned jobs."

2. For the unassigned job that belongs to Agent B, create an "unassigned-B-agent-jobs" set.
3. Sort jobs in "unassigned-B-agent-jobs" set by deadlines.

if *unassigned-jobs* $\subset J_A$**:**

1. Sort jobs by $pd_j(r,t) = (p_j + \beta t)r^{\alpha}/w_j$
2. Obtain schedule S that contains all the jobs by adding the jobs in "unassigned-jobs" set in the same order. Calculate OFV (S).

if OFV(S) <= UB:

1. Equal UB to OFV(S)
2. Add schedule S to the "feasible-solutions" set.

Elif *unassigned-jobs* $\subset J_B$**:**

1. Generate sorting combinations of jobs and add each combination to the partial schedule in the node.
2. Check if any of the jobs are tardy by Algorithm 2
3. Add schedules that none of the jobs are tardy to the "feasible-solutions" set.

if OFV(S) <= UB:

a. Equal UB to OFV(S)

Else:

For each job in "unassigned-jobs," do:

1. Sort jobs by $pd_j(r,t) = (p_j + \beta t)r^{\alpha}/w_j$
2. Add a job with min $pd_j(r,t)$to the schedule in the node and create partial schedule S.
3. Calculate the lower bound value of S with Algorithm 4.

If LB <= UB

Check if any of the unassigned jobs belonging to agent B gets tardy by using Algorithm 2.

If number_of_tardy_jobs = 0:

Apply Algorithm 5 to check if the partial schedule S is dominant.

If S is dominant in node-list:

Add S to the node list**.**

If s(S/J) = 0:

Equal LB to UB

Add schedule S to the "feasible-solutions" set.

Else Prune the node

Else Prune the node

Else Prune the node

End

Output node-list

End

End

Output feasible-solutions

1.Sort schedules in "feasible-solutions" set in ascending order of objective function values.

2.Add schedule or schedules with the smallest objective function value to the "optimal-solution" set.

End

Output optimal-solution

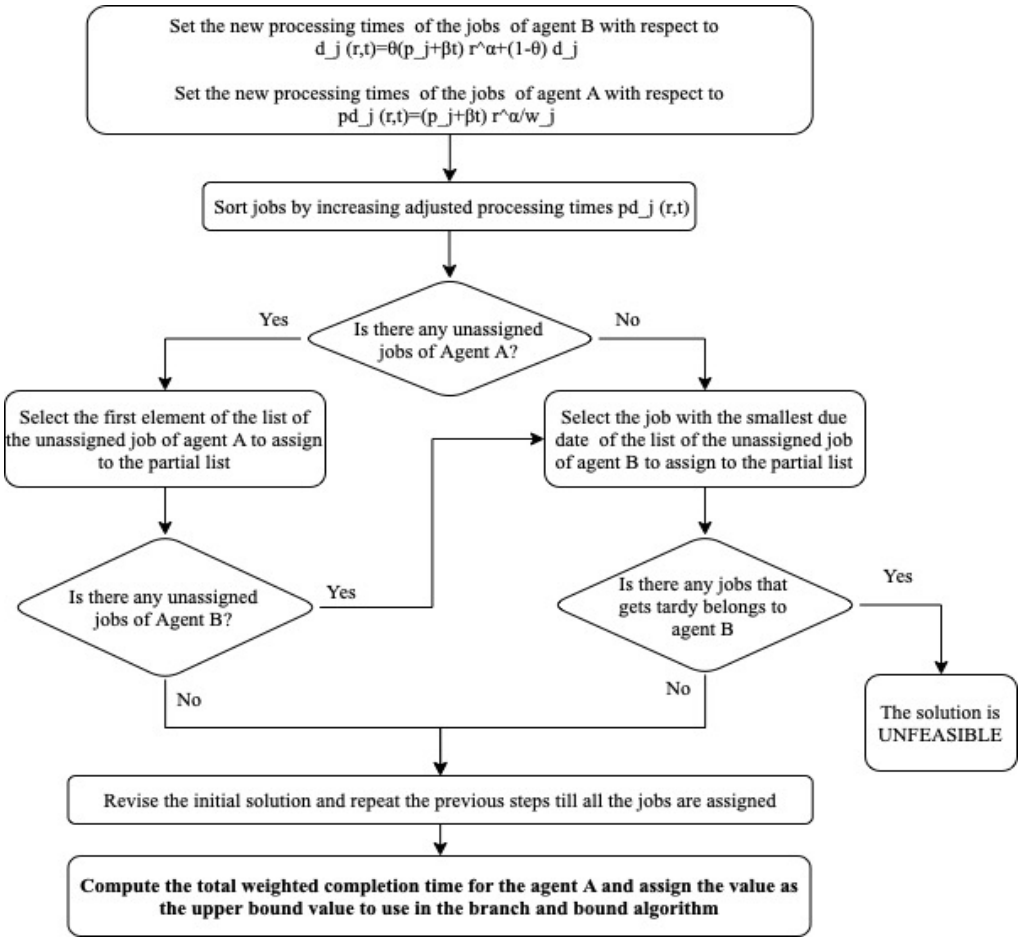


Figure 3. Pruning Rules

4.2 Calculation of Lower Bound

The efficiency of the branch-and-bound algorithm also depends on the lower bound of the partial sequence. Assume that S is a partial schedule in which the order of the first k jobs is determined. There are (n − k) jobs unassigned to the partial schedule in the node. The completion time for the partial schedule S is represented by $C_{[k]}$. Among the unscheduled jobs, there are k_B jobs belonging to the Agent B. So, there are $n - k - k_B$ jobs belonging to agent A.

All the jobs are under the simultaneous effect of learning and deterioration. Due to the deterioration effect, the later position a job is assigned, the longer the processing time; however, the processing time is also reduced due to the learning effect. In our problem, it is hard to decide if the processing time of the job will increase or decrease by the position assigned. Besides, there is a tardiness constraint for jobs of agent B.

To calculate the lower bound, assign unscheduled jobs of Agent B to the partial schedule S by ascending order by deadlines and by assuming their processing times are equal to 0 (zero). Thus, we meet the tardiness constraint. Position $(k + k_B + 1)$ will be the first position that the unscheduled jobs of Agent A will be assigned. We also ensure that the unscheduled jobs of Agent A are not affected by the deterioration effect in any way and benefit from the learning effect to the maximum extent. The pseudo-code for calculating the lower bound value is given in Algorithm 4.

Algorithm 4. Calculation of Lower Bound

Create an "unassigned-A-agent-jobs" set for jobs belonging to agent A that are not assigned to the partial schedule in the node.
$unassigned\text{-}A\text{-}agent\text{-}jobs = \{j \mid j \in J_A \cap J, j \notin S\}$
$C_{[k]}$: the completion time of partial schedule S

For $(r \leftarrow 1 to n-k-k_B)$ **do:**

1. For each job "unassigned-A-agent-jobs" set, calculate $pd_j(r,t) = (p_j + \beta t)(r+k+k_B)^{\alpha}/w_j$, and sort ascending.
2. Assign the job with minimum value $pd_j(r,t)$ to the position $(k+k_B+r)$ of partial schedule S and remove the job from the "unassigned-A-agent-jobs" set.
3. For $\hat{C}_{[0]} = C_{[k]}$, calculate the completion time for the job with $\hat{C}_{[r]} = \left(p_j + \beta \hat{C}_{[r-1]}\right)(k+k_B+r)^{\alpha}$
4. r = r +1

Calculate lower bound:

$LB = \sum_{j=1}^{k} w_{[k]}^{A} C_{[k]}^{A} + \sum_{r=1}^{n-k-k_B} w_{[r]} \hat{C}_{[r]}$

End

Output $LB = \sum_{j=1}^{k} w_{[k]}^{A} C_{[k]}^{A} + \sum_{r=1}^{n-k-k_B} w_{[r]} \hat{C}_{[r]}$

4.3 Dominance Rules

Consider two partial schedules: S and S'. The only difference between them is a pairwise interchange of two adjacent jobs *i* and *j*. In partial schedule S job *i* processed in the r^{th} position and the job *j* processed in $(r+1)^{th}$position. Let *t* denote the completion time of the job scheduled to $(r-1)^{th}$position.

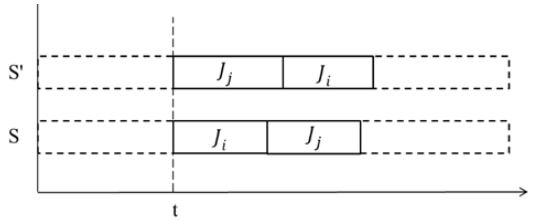


Figure 4. Gannt Diagram of Partial Schedules S and S'

In accordance with the dominant rules, new job assignments will be continued on the node with the dominant schedule; the node with the non-dominant schedule will be pruned. Jobs are under the effects of learning and deterioration simultaneously; so, two parameters determine the dominance: the objective function value of partial schedule and the completion time of the last job scheduled in the partial schedule.

Partial schedule S is dominant to S' if one of the conditions listed below is met. For other combinations of conditions, it is hard to decide which partial schedule is dominant.

- $OFV(S) < OFV(S')$ and $C_j(S) < C_i(S')$
- $OFV(S) < OFV(S')$ and $C_j(S) = C_i(S')$
- $OFV(S) = OFV(S')$ and $C_j(S) < C_i(S')$

By using the equations [1-6], the dominance conditions described will be detailed. To explain the dominance rules as agent-based, variable I_j will be used. I_j indicates the agent to which the job belongs. If job $J_j \in J_A$, then $I_j = 0$; otherwise $I_j = 1$.

$$OFV(S) = \sum_{r}^{r-1} w_{[r]}^{A} C_{[r]}^{A} + w_i C_i(S)(1-I_i) + w_j C_j(S)(1-I_j) \quad (1)$$

$$OFV(S') = \sum_{r}^{r-1} w_{[r]}^{A} C_{[r]}^{A} + w_j C_j(S')(1-I_j) + w_i C_i(S')(1-I_i) \quad (2)$$

$$C_i(S) = t + (p_i + \beta t) r^{\alpha} \quad (3)$$

$$C_j(S) = C_i(S) + \left(p_j + \beta C_i(S)\right)(r+1)^{\alpha} \quad (4)$$

$$C_j(S') = t + (p_j + \beta t) r^{\alpha} \quad (5)$$

$$C_i(S') = C_j(S') + \left(p_i + \beta C_j(S')\right)(r+1)^{\alpha} \quad (6)$$

Proposition 1. If $(p_j - p_i) > 0$ then $C_j(S) - C_i(S') > 0$

Proof. By using equations from 1 to 6; the difference between completion times of partial schedules S and S' is obtained in Equation 7.

$$C_j(S) - C_i(S') = (p_j - p_i)((r+1)^\alpha + \beta r^\alpha (r+1)^\alpha - r^\alpha) \quad (7)$$

In Equation 7, $(r+1)^\alpha + \beta r^\alpha (r+1)^\alpha - r^\alpha > 0$; so, if $(p_j - p_i) > 0$ then $C_j(S) - C_i(S') > 0$ too.

Proposition 2. If $J_j \in J_B$ and $J_i \in J_B$ then $OFV(S) - OFV(S') = 0$

Proof. By using equations from 1 to 6; the difference between objective function values of partial schedules S and S' is obtained in Equation 8.

$$OFV(S) - OFV(S') = w_j(1 - I_j)(C_j(S) - C_j(S')) + w_i(1 - I_i)(C_i(S) - C_i(S')) \quad (8)$$

If $J_j \in J_B$ and $J_i \in J_B$ then $I_i = I_j = 1$. So $OFV(S) - OFV(S) = 0$

Proposition 3. If $J_j \in J_B$ and $J_i \in J_A$ and $C_i(S) < C_i(S')$, then $OFV(S) - OFV(S') < 0$. So S dominates S'.

Proof. Equation 8 is used again. If $J_j \in J_B$ and $J_i \in J_A$ then $I_j = 1$ and $I_i = 0$.

$$OFV(S) - OFV(S') = w_i(C_i(S) - C_i(S'))$$

Proposition 4. If $J_j \in J_A$ and $J_i \in J_B$ and $C_j(S) > C_j(S')$, then $OFV(S) - OFV(S') > 0$. So S' dominates S.

Proposition 5. If $J_j \in J_A$ and $J_i \in J_A$ and $(p_i - p_j)(w_j - w_i) > 0$ and $C_j(S) - C_i(S') > 0$ then S' dominates S.

Proof. By using Equation 8:

$$OFV(S) - OFV(S') = w_j\left(C_j(S) - C_j(S')\right) + w_i\left(C_i(S) - C_i(S')\right)$$
$$OFV(S) - OFV(S') = (p_i - p_j)r^\alpha(w_j - w_i) + \beta r^\alpha (r+1)^\alpha(w_j p_i + w_i p_j) + (r+1)^\alpha(w_j p_j + w_i p_i)$$
$$+\beta t(r+1)^\alpha(w_i + w_j)(1 + \beta r^\alpha) \quad (9)$$

In Equation 9, all the elements are positive but $(p_i - p_j)r^\alpha(w_j - w_i)$. So, if $(p_i - p_j)(w_j - w_i) > 0$ then $OFV(S) - OFV(S') > 0$; if $(p_i - p_j)(w_j - w_i) < 0$ then $OFV(S) - OFV(S') < 0$.

Proposition 6. If $J_j \in J_A$ and $J_i \in J_A$ and $(p_i - p_j)(w_j - w_i) < 0$ and $C_j(S) - C_i(S') < 0$ then S dominates S'.

The pseudo-code for dominance rules application is given in Algorithm 5.

Algorithm 5. Dominance Rules

Input: $S = \{\dots, J_{[r-2]}, J_{[r-1]}, J_{i,[r]}\}$
$S' = \{\dots, J_{[r-2]}, J_{[r-1]}, J_{j,[r]}, J_{j,[r+1]}\}$
node-list = {Partial schedules obtained by branching nodes in branch and sound algorithm}

For J_j **do:**

1. Add job J_j to partial schedule S, position (r+1)

2. Get $S = \{\dots, J_{[r-2]}, J_{[r-1]}, J_{i,[r]}, J_{j,[r+1]}\}$

If S' **in node_list:**

Check if S dominates S' by using propositions from 1-6

If partial schedule S is dominant to S'

1. Add partial schedule S to node-list set.

2. Remove partial schedules S' from node_list set

Elif partial schedule S' is dominant:

1. Do not add partial schedule S to the node-list set.

Else:

1. Add partial schedule S to node-list set.

Else:

1. Add partial schedule S to node-list set.

End

Output node-list

6. COMPUTATIONAL RESULTS

All algorithms developed are coded in the JetBrains PyCharm Professional 2020.2 environment, and experiments are performed on a personal computer powered by a 2.6 GHz 6-Core Intel Core i7 processor and a 16 GB 2667 MHz DDR4 memory personal computer with a macOS Catalina operating system. The job processing times and due dates were generated randomly from a uniform distribution over the integers (0,100) and [1, 5], respectively. The experiments were conducted with different sample sizes and different deterioration coefficients, learning coefficient, and interpolation coefficient parameters. Total jobs belonging to the agents were chosen as 5, 10, 15, 20, and 25. While deterioration coefficient (β) was chosen as 0,1 and 0,2; learning coefficient (α) were taken into consideration for %70, %80 and %90. The interpolation coefficient, the weight between the due date and the real processing time for agent B, was taken as 0, 0.5, and 1. Considering the parameter combinations, performance evaluation was made on a total of 90 different combinations. For each combination, experiments were repeated ten times; in total conducted by 900 times.

The data gathered from the experiments are nodes-ratio, error-percentage, and CPU-time. Nodes-ratio shows how the developed algorithm reduces the number of nodes that will be created in the classical branching method and is given by

$$Nodes - ratio = TN^*/TN\ , \tag{10}$$

where TN and TN* are denoted as the nodes created in the classical branch-and-bound algorithm and the total nodes created by the proposed algorithm, respectively.

The percentage error of a solution is given by:

$$Error - percentage = \frac{(OFV^* - OFV)}{OFV^*}, \tag{11}$$

where OFV and OFV^* are denoted as the total weighted completion time of the initial solution and the branch-and-bound algorithm solution, respectively.

CPU-time shows the amount of time used for processing the algorithm developed. It should not be forgotten that CPU times are not only related to the success of the algorithm but also to how the algorithm is encoded.

The efficiency of the proposed methodology depends on some parameters: total number of jobs, slack times of jobs belonging to the agent that no tardy job is allowed, values of learning coefficient, and deterioration coefficient. The relation between the parameters and outputs (processing time and the number of nodes) was studied. ANOVA and the Pearson correlation were studied with a 95% significance, and it was found that as the number of total jobs increases; so, does the processing time and number of nodes too. The relations are shown in Figure 5, Figure 6, Figure 7, and Figure 8. Besides, in Table 2, performance evaluation results are given for error percentage, CPU-time, and nodes-ratio. The metrics are minimum, maximum, and mean values.

Table 2. Performance Evaluation Results

		Total Number of Jobs (n)				
		5	10	15	20	25
Error-percentage	Minimum	-0,17908	-0,27489	-0,34361	-0,37943	-0,48822
	Mean	-0,08012	-0,10649	-0,13311	-0,14913	-0,18956
	Maximum	0,00000	0,00000	0,00000	0,00000	0,00000
CPU-time	Minimum	0,0006	0,0061	0,0079	0,0219	425,0399
	Mean	0,0016	304,8133	435,4453	1219,2484	2438,4953

	Maximum	0,0069	3697,9903	5282,8334	10565,67	28489,92
Node-ratio	Minimum	9,23E-01	2,85E-06	3,00E-10	6,99E-18	3,52E-22
	Mean	3,19E+00	1,28E-02	1,28E-06	2,99E-14	2,79E-18
	Maximum	5,54E+00	6,10E-02	7,31E-06	1,70E-13	8,34E-18

The error-percentage value is directly affected by the total number of jobs and the interpolation coefficient. The error-percentage value gets higher for the higher values of the total number of jobs or for the lower values of the interpolation coefficient. It also gets lower for the higher values of the learning effect and the lower values of the deterioration effect. It is also found that the value of the node-ratio depends on the error-percentage value, value of the learning effect, value of the deterioration effect and the total number of the jobs. It gets lower for the lower values of the error-percentage. The value of the learning effect and the value of the deterioration effect alone do not affect the algorithm's efficiency, but their combination does; because they directly affect slack time and the number of jobs that can be processed in the slack time interval. The difference between the deadline and the processing time of these jobs is called slack time, and if one or more jobs can be processed in the slack time interval, this will make the algorithm work slower, increase CPU- time and the total node number. Job processing times are getting shorter for the higher values of the learning effect and the lower values of the deterioration effect. Therefore, slack time will increase. Also, the number of jobs that can be processed during slack-time intervals, CPU time, and node ratio will increase even more. Job processing times are getting longer for the lower values of the learning effect and the higher values of the deterioration effect. Therefore, slack time will decrease. Also, the number of jobs that can be processed during slack-time intervals, CPU time, and node ratio will decrease even more.

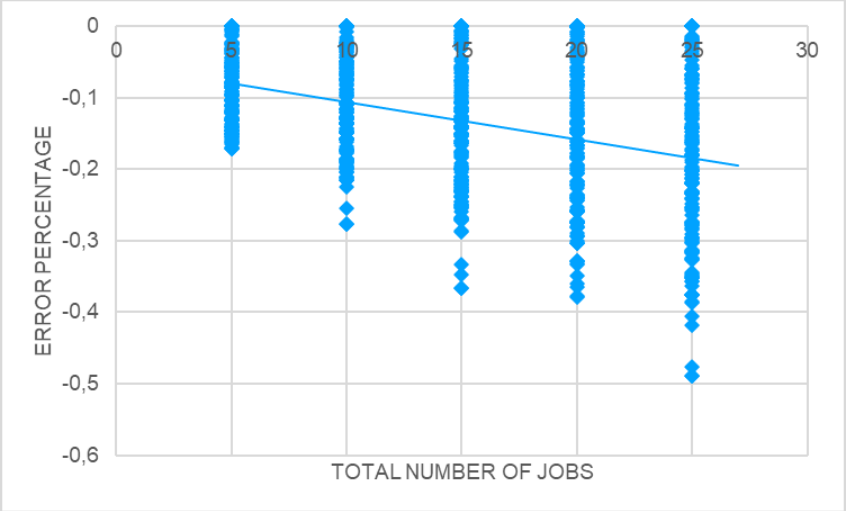


Figure 5. Relation Between Total Number of Jobs and Processing Time and Error Percentage

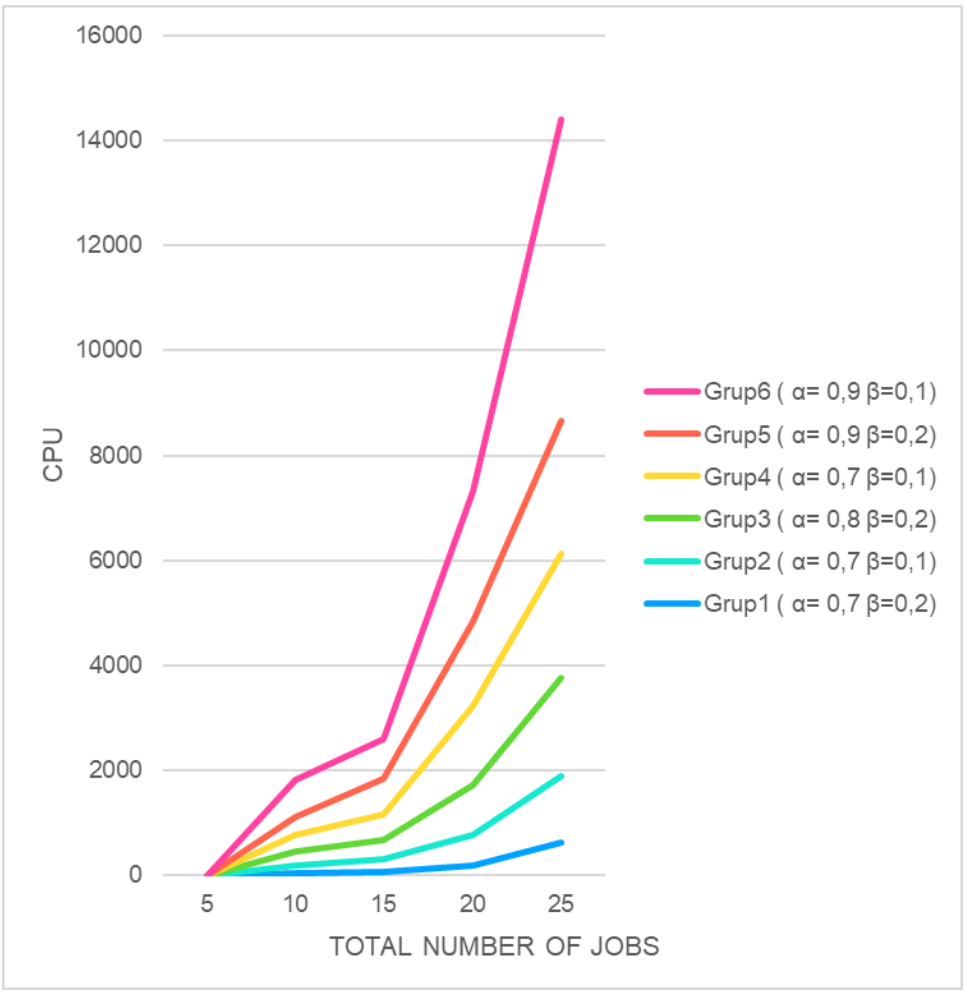


Figure 6. Relation Between Total Number of Jobs and CPU Time for Different Learning and Deterioration Coefficient

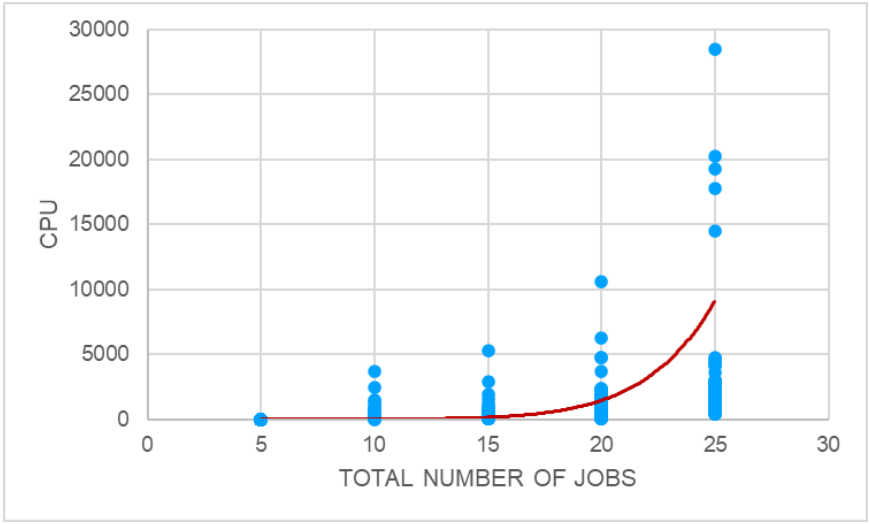
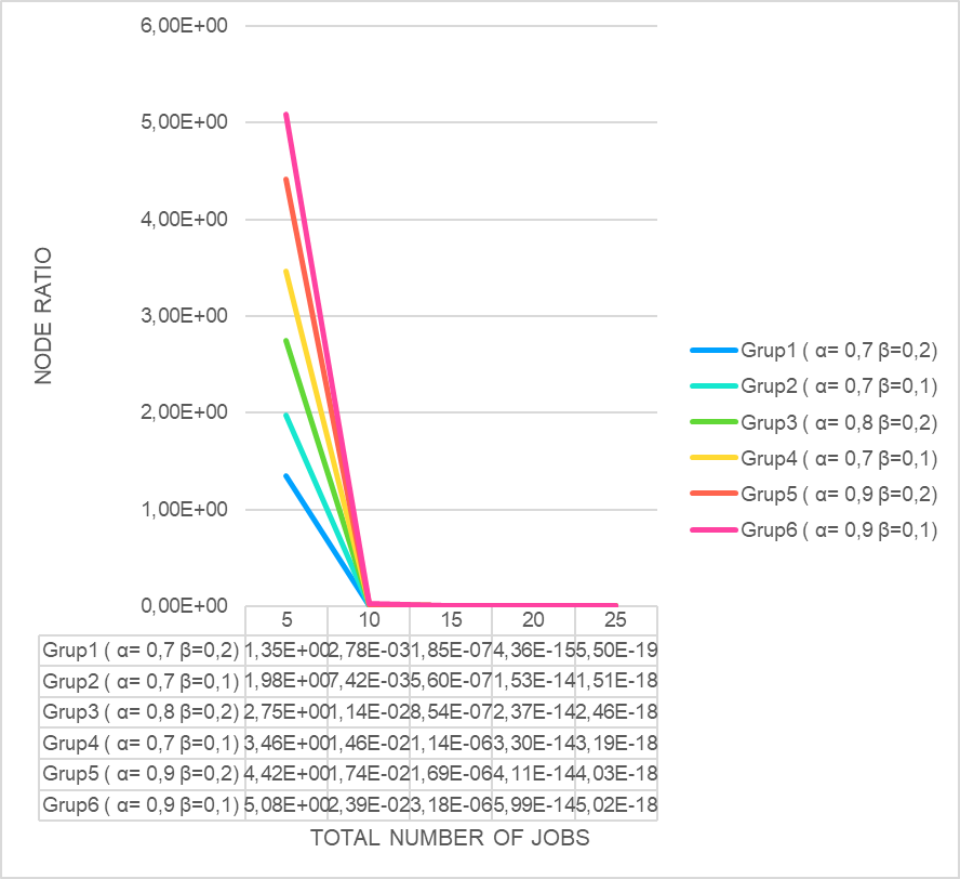


Figure 7. Relation Between Total Number of Jobs and CPU time



	5	10	15	20	25
Grup1 (α= 0,7 β=0,2)	1,35E+00	2,78E-03	1,85E-07	4,36E-15	5,50E-19
Grup2 (α= 0,7 β=0,1)	1,98E+00	7,42E-03	5,60E-07	1,53E-14	1,51E-18
Grup3 (α= 0,8 β=0,2)	2,75E+00	1,14E-02	8,54E-07	2,37E-14	2,46E-18
Grup4 (α= 0,7 β=0,1)	3,46E+00	1,46E-02	1,14E-06	3,30E-14	3,19E-18
Grup5 (α= 0,9 β=0,2)	4,42E+00	1,74E-02	1,69E-06	4,11E-14	4,03E-18
Grup6 (α= 0,9 β=0,1)	5,08E+00	2,39E-02	3,18E-06	5,99E-14	5,02E-18

Figure 8. Relation Between Total Number of Jobs and Node-Ratio for Different Learning and Deterioration Coefficient

7. CONCLUSION

This paper proposed a methodology to solve a multi-agent scheduling problem for jobs under the simultaneous effect of learning and deterioration. The objective of the problem was to minimize the total weighted completion time for the first agent with the restriction that no tardy job is allowed for the second agent. The proposed methodology consists of 2 phases: a heuristic for the initial solution and a branch-and-bound algorithm for the final and optimized solution. Some dominance rules and pruning rules were developed to accelerate the branch-and-bound algorithm phase and reduce the searching scope. The computational results showed that with the help of the proposed heuristic initial solution, the branch-and-bound algorithm performs well in terms of the number of nodes and execution time. The proposed methodology was tested for up to 25 jobs. But it can also be used effectively for problems involving much more jobs with smaller learning coefficients and bigger deterioration coefficients.

REFERENCES

Agnetis, A., Mirchandani, P. B., Pacciarelli, D., and Pacifici, A. (2000). Nondominated schedules for a job-shop with two competing users. *Comput. Math. Organ. Theory*, 191-217.

Agnetis, A., Mirchandani, P. B., Pacciarelli, D., and Pacifici, A. (2001). Scheduling problems with two competing users. *UniversitaÁ di Roma "Tor Vergata," Centro Vito Volterra, Working Paper No. 452.*

Agnetis, A., De Pascale, G., and Pacciarelli, D. (2009). A lagrangian approach to single- machine scheduling problems with two competing agents. *Journal of Scheduling*, 12(4): 401–415.

Agnetis, A., Billaut, J. C., Gawiejnowicz, S., Pacciarelli, D., and Soukhal, A. (2014). *Multi-agent scheduling models and algorithms*. Springer publishing.

Baker, K. R. and Smith, J. C. (2003). A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6(1): 7-16.

Cheng, T. C. E., Wu, W. H., Cheng, S. R., and Wu, C. C. (2011). Two-agent scheduling with position-based deteriorating jobs and learning effects. *Applied Mathematics and Computation*, 217(21): 8804–8824.

Cheng, T. C. E., Cheng, S. R., Wu, W. H., Hsu, P. H., and Wu, C. C. (2011(2)). A two-agent single-machine scheduling problem with truncated sum-of-processing-times- based learning considerations. *Computers and Industrial Engineering*, 60(4): 534–541.

Cheng, T. C. E., Chung, Y. H., Liao, S. C., and Lee, W. C. (2013). Two-agent single-machine scheduling with release times to minimize the total weighted completion time. *Computers and Operations Research*, 40: 353-361.

Choi, B. C. and Chung, J. (2014). Two-agent single-machine scheduling problem with just-in-time jobs. *Theoretical Computer Science*, 543: 37–45.

Choi, J. Y. (2015). Minimizing total weighted completion time under makespan constraint for two-agent scheduling with job-dependent aging effects. *Computers and Industrial Engineering*, 83: 237–243.

Elvikis, D., Hamacher, W., and T'kindt, V. (2011). Scheduling two agents on uniform parallel machines with makespan and cost functions. *Journal of Scheduling*, 14: 471–481.

Feng, Q., Fan, B., Li, S., and Shang, W. (2015). Two agent scheduling with rejection on a single machine. *Applied Mathematical Modelling*, 39: 1183-1193.

Lee, K., Choi, B. C., Leung, J. Y. T., and Pinedo, M. L. (2009). Approximation algorithms for multi-agent scheduling to minimize total weighted completion time. *Information Processing Letters*, 109(16): 913–917.

Lee, W. C., Chen, S. K., and Wu, C. C. (2010). Branch-and-bound and simulated annealing algorithms for a two-agent scheduling problem. *Expert Systems with Applications*, 37(9): 6594–6601.

Lee, W.C., Chung, Y.H., and Huang, Z. R. (2013). A single-machine bi-criterion scheduling with two agents. *Applied Mathematics and Computation*, 219: 10831-10841.

Lee, W. C. and Wang, J. Y. (2014(2)). A scheduling problem with three competing agents. *Computers and Operations Research*, 51: 208–217.

Li, D. C. and Hsu, P. H. (2011). Solving a two-agent single-machine scheduling problem considering learning effect. *Computers and Operations Research*, 39(7): 1644–1651.

Li, S. S., Chen, R. X., and Feng, Q. (2015). Scheduling two job families on a single machine with two competitive agents. *Journal of Combinatorial Optimization, Springer*.

Li, S. S., Yuan, J. J. (2020). Single-machine scheduling with multi-agents to minimize total weighted late work. *Journal of Scheduling*, 23: 497–512.

Mor, B. and Mosheiov, G. (2010). Scheduling problems with two competing agents to minimize minmax and minsum earliness measures. *European Journal of Operational Research*, 206(3): 540–546.

Soltani, R., Jolai, F., and Zandieh, M. (2010). Two robust meta-heuristics for scheduling multiple job classes on a single machine with multiple criteria. *Expert Systems with Applications*, 37(8): 5951–5959.

Wu, W. H., Xu, J., Wu, W. H., Yin, Y., Cheng, I. F., and Wu, C. C. (2013). A tabu method for a two-agent single-machine scheduling with deterioration jobs. *Computers and Operations Research*, 40: 2116-2127.

Wu, W. H. (2013(2)). A two-agent single-machine scheduling problem with learning and deteriorating considerations. *Mathematical Problems in Engineering*, 8.

Wu, W. H. (2013). An exact and meta-heuristic approach for two-agent single-machine scheduling problem. *Journal of marine science and technology*, 21(2): 215-221.

Wu, W. H. (2014). Solving a two-agent single-machine earning scheduling problem. *International Journal of Computer Integrated Manufacturing*, 27: 20-35.

Yin, Y., Cheng, T. C. E., Wan, L., Wu, C. C., and Liu, J. (2015). Two-agent single machine scheduling with deteriorating jobs. *Computers and Industrial Engineering*, 81: 177-185.

Yin, Y., Wu, W. H., Cheng, T. C. E., Wu, C. C., and Wu, W. H. (2015(3)). A honey-bees optimization algorithm for a two-agent single-machine scheduling problem with ready times. *Applied Mathematical Modelling*, 39: 2587–2601.

Zhang, Y., Yuan, J., Ng, C., and Cheng, T. C. (2020). Pareto-optimization of three-agent scheduling to minimize the total weighted completion time, weighted number of tardy jobs, and total weighted late work. *Naval Research Logistics*, 68: 378-393.