IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 21, 2021, accepted May 13, 2021, date of publication May 17, 2021, date of current version May 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3081495

Integrated Scheduling of Tasks and Preventive Maintenance Periods in a Parallel Machine Environment With Single Robot Server

LOTFI HIDRI[1], KHALED ALQAHTANI[1], ACHRAF GAZDAR[2], AND AHMED BADWELAN[1]
[1]Industrial Engineering Department, College of Engineering, King Saud University, Riyadh 11421, Saudi Arabia
[2]Software Engineering Department, College of Computer and Information Science, King Saud University, Riyadh 11421, Saudi Arabia

Corresponding author: Lotfi Hidri (lhidri@ksu.edu.sa)

This work was supported by the Deanship of Scientific Research at King Saud University through the Research Group under Grant RG-1439-001.

ABSTRACT In this study, the objective of minimizing makespan has been considered for a scheduling problem of identical parallel machines with a single server and unavailability constraints. The unavailability constraints correspond to preventive maintenance periods. In this study, the jobs and the maintenance periods are scheduled simultaneously. This scheduling problem has a wide range of potential application areas in the manufacturing environment. In addition, the studied problem is a challenging one from theoretical point of view, due to its NP-Hardness. To conduct the study, a lower bound (*LB*) for the problem, and three metaheuristics namely Simulated Annealing (*SA*), Tabu Search (*TS*), and Genetic Algorithm (*GA*) have been proposed. The best parameters settings of the proposed algorithms were conducted using pilot runs with a Taguchi design. The algorithms performance has been assessed by using a set of test problems generated randomly. These test problems are based on a literature benchmark. The size of the instances, or number of jobs, were up to 500. Along with the performance analysis of the proposed algorithms, the effect of varying processing times and unavailability periods on the performance of the proposed algorithms is studied. The present work provides strong evidence of the efficiency and the performance of the proposed algorithms.

INDEX TERMS Scheduling, parallel machines, single server, maintenance, metaheuristic, optimal solution.

I. INTRODUCTION

Customer demands are constantly changing and the need to deliver better quality products are among the most important driving factors to improve production system performance. Production systems must have adaptive ability to meet the necessary manufacturing requirements in a timely manner. Manufacturing systems have achieved tremendous growth over the past three decades by introducing and implementing various management concepts and various tools. These concepts and tools control the production process related to different types of production environments, and coordinate necessary communication between different levels of organization using a variety of advanced information technology.

Scheduling is the allocation of limited resources over time to tasks. The resources take the form of machines and activities take the form of workers in a standard production environment. Machine scheduling is defined as assigning jobs to machines and specifying the sequence and processing times of machine jobs in order to optimize a performance criterion.

Parallel machine scheduling problem is one of the most challenging types of scheduling problems and a large number of studies have been conducted on various commercial, industrial and educational fields. Parallel machines can be categorized into three groups; identical parallel machines, uniform parallel machines and unrelated parallel machines [1]. The studies associated to minimize makespan in parallel machines scheduling problems with setup times have attracted a particular attention in recent years ([2]-[5]).

A common problem in manufacturing is the need to share a common server, such as a worker, a robot, a tool etc., by a number of parallel machines to carry out machine setups/loads. Scheduling problems with a single server have their

The associate editor coordinating the review of this manuscript and approving it for publication was Sun-Yuan Hsieh.

applications in several manufacturing systems such as manufacturing of automobile components, automated material handling systems, robotic cells, the semiconductor industry, etc. In this problem, each job is first loaded by the server (robot) on the available machine and then processed automatically and independently by that machine. In consequence, sharing the server resource results in machine idle time. Developing a good schedule will lead to reduce or eliminate the machine idle time.

However, scheduling on identical parallel machines with a single server assumes machines are always available, which is not realistic in practical cases. There will always be a possibility for machines maintenance, one of the most used maintenance actions in manufacturing is interval-based preventive maintenance. The findings in the general scheduling literature may become insufficient with machine availability constraints to find optimal solutions for many cases. Although scheduling with machines maintenance constraints has gained some popularity over the last decade, the related literature still has gaps with respect to numerous machines arrangements and objective functions.

This study considers identical parallel machine scheduling problem with a single server and availability constraints on each machine with an aim in minimizing makespan.

With the increasing uncertainty and complexity of modern production systems environments, many scheduling problems have been proven to be NP-Hard [6]. In addition, it is difficult to understand the solution space for the scheduling problem in order to develop the appropriate technique to achieve the optimal or near-optimal solution in the complex solution space. Therefore, the problem requires an appropriate and considerable amount of research effort to achieve its objective.

Approximation techniques, which belong to optimization algorithms, have played a main role in solving complex scheduling issues. Approximation methods like Iterative Search ITS, GA, SA, TS and Particle Swarm Optimization, and Harmony Search Algorithm, are generally found to be robust and produce good results. Their performance is satisfactory as long as the objectives specified in the Identical Parallel Machine Scheduling environment (IPMS) and unavailability period of the machines. This encourages researchers to apply metaheuristic techniques that support in providing near optimal solutions and reduce computational burden.

This study considers identical parallel machine scheduling problem with single server and availability constraints on each machine with the objectives of minimizing makespan. To the best of our knowledge, no reported studies have taken into account the machine unavailability constraints (i.e., interval-based preventative maintenance) when scheduling identical parallel machines using a common server.

This paper is organized as follows. In Section 2, an exhaustive literature review is presented. In Section 3, the studied problem is defined. Lower bounds are proposed in Section 4. The proposed metaheuristics, the initial solution, and the data generation are discussed in Section 5. Extensive experimental study, intended to assess the performance of the proposed procedures, is presented in Section 6. A conclusion summarizing the main findings and presenting future research directions, is finally presented.

II. LITERATURE REVIEW

The identical parallel-machine scheduling problem is denoted as $P_m||C_{max}$ where jobs (i.e., tasks) are processed on identical parallel machines in order to minimize makespan. This problem is shown to be NP-Hard [7]. Many methods were proposed in the literature to solve this problem. In practice, setups are a very significant issue and cannot be assumed as part of the processing times as in the $P_m||C_{max}$. Besides, the common server issue is one of important problems in parallel machines environments. Moreover, in basic parallel machines environments, machine availability is always assumed which is not realistic in practical cases. There will always be a possibility for machines maintenance, one of the most used maintenance actions in manufacturing is interval-based preventive maintenance.

Parallel machines scheduling problem is a commonly studied scheduling problems, and a substantial number of studies have been conducted on many commercial, industrial and academic fields [1]. Parallel machines scheduling problems can be roughly classified into three categories which are identical parallel machines, uniform parallel machines, and unrelated parallel machines [8].

Parallel machine scheduling problems have been widely studied in the literature due to their importance in manufacturing and planning [1]. In order to find the solution to these problems, several polynomial time algorithms have been suggested. Longest Processing Time (*LPT*) is one of the most popular rules [9]. The *LPT* rule received a lot of attention for solving single criteria makespan problems.

Gupta and Ho [10] considered the job scheduling problem on two identical parallel machines with minimization of total follow time. Liu and Wu [11] proposed a programming method to minimize the number of tardy jobs in the same parallel production line scheduling problem. Damodaran and Vélez-Gallego [12] used a SA algorithm to minimize the makespan on identical batch processing machines arranged in parallel where each job had an arbitrary processing time, non-identical size and non-zero ready time. Xu *et al.* [13] adopted an iterated local search (*ILS*) and a TS to find a near-optimal solution for two-parallel machine scheduling problem.

Ghalami and Grosu [14] proposed a parallel approximation algorithm to address the scheduling parallel identical machines problem with shared-memory systems in order to reduce makespan. Ozer and Sarac [15] studied an identical parallel scheduling problem with sequence-dependent initialization times multiple copies of shared resources.

Tanaka and Araki [16] suggested a new branch and bound algorithm for the scheduling problem class in order to minimize total tardiness on the identical parallel machines. The Lagrangian relaxation technique is used

in proposed algorithm to achieve a narrow lower bound. Ranjbar *et al.* [17] developed two branch and bound algorithms to find an optimal solution, with the aim of finding a robust schedule that maximizes the level of customer service, which is the likelihood that the makespan will not exceed the due date. Mensendiek *et al.* [18] regarded IPP scheduling to minimize total tardiness where jobs can only be provided at set delivery dates exogenously provided using the TS algorithm.

Schaller [19] described several procedures using TS to schedule identical parallel machines with family setup to minimize total tardiness. To minimize total weighted tardiness, Yeh *et al.* [20] applied SAA fuzzy to address parallel machine scheduling with learning to minimize makespan. Kim et al [21] used TS to determine the allocation and sequence of jobs on parallel machines to minimize total tardiness. Kim and Shin [22] opted to minimize the maximum lateness of the jobs by using a restricted TS algorithm that schedules jobs on parallel machines. The jobs have release times, and between the jobs there are sequence-dependent setup times. Regarding job processing times, the parallel machines are either identical or not identical.

Bilge *et al.* [23] suggested a TS approach for scheduling a set of independent jobs on a set of uniform parallel machines to minimize total tardiness. Jobs have non-identical due dates and arrival times. Tavakkoli-Moghaddam *et al.* [24] suggested an effective GA to solve bi-objectives, especially the number of tardy jobs and completion of all unrelated parallel machine scheduling tasks. Chaudhry and Drake [25] suggested a GA algorithm to minimize a set of duties for identical parallel machines and worker assignments.

Xu and Yang [26] addressed IPMS problems with the goal of minimizing total weighted completion time and makespan. The problem was formulated as mixed integer programming based on optimal schedule properties. Rajakumar *et al.* [27] used the GA to solve the manufacturing system's parallel machine scheduling problem with the workflow balancing goal. Chen *et al.* [28] presented a scheduling algorithm based on GA for the scheduling of workshop problems with parallel machine and the reentrant process. Balin [29] addressed parallel machine scheduling problems with fuzzy processing times with a robust GA approach embedded in a solution model to minimize the maximum completion time. Huo [30] studied the parallel machine scheduling problems subject to machine availability and total completion time constraint. Ma *et al.* [31] proposed a model based on Liu's chance theory for parallel machine scheduling problem with uncertainty and randomness simultaneously for processing times of jobs. The model's goal is to reduce the planned completion time.

In manufacturing systems, some resources such as a piece of equipment or a team of setup workers or a single operator may be required throughout the setup process. Each of these situations defines a scheduling problem with a common server. Server type may vary according to the production environment. Examples of this problem occur frequently in production environments such as manufacturing of automobile components and printing industries [3], automated material handling systems, semiconductor industry, or scheduling of maintenance and setups systems with limited staff number [32], textile industry [2], or network computing [33], and so on.

The scheduling identical parallel machines with single server problem for setting up/ loading the machines is denoted as $P_m, S1||C_{max}$ using standard scheduling notation when the objective is to minimize the maximum completion time (makespan). $P_2, S1||C_{max}$ was first presented by [34].

Several studies proposed formulations and algorithms to solve the general problem $(P_2, S1||C_{max})$ along with other several special cases of the problem. [35] developed an integer programming formulation to solve the general case of small-size problems up to 12 jobs using CPLEX. Two special cases of the problem were also considered; short processing times and equal length jobs. In addition, simple backward/forward $O(n \log n)$ heuristics were also proposed to solve the problem $(P_2, S1||C_{max})$. [36] proposed GA, greedy heuristic and Gilmore-Gomory algorithm to solve the general case problem with 50 and 100 job instances. [37] developed two mixed integer linear programming (MILP) formulations and two variants of a branch-and-price scheme for instances up to 100 jobs. [32] developed mixed integer linear program formulations and a hybrid heuristic of SA and TS for instances up to 40 jobs. The runtime was within 3,600s. [38] proposed MILP formulations based on decomposing a schedule into a set of blocks (for instances up to 250 jobs). [39] proposed SA and GA algorithms for large-scale instances up to 1,000 jobs. The run time was limited within 14,400s. In [40] authors developed a constructed algorithm named I-L algorithm to solve the general problem with large-scale instances up to 1,000 jobs. The performance of I-L algorithm was assessed by comparing its results with *SA* and *GA* results. It was reported that, in general, I-L algorithm outperformed other algorithms in terms of runtime. In addition, for large instances the I-L algorithm outperformed *SA* and *GA* in terms of runtime and the objective function. I-L algorithm was also used for instances of 10,000 jobs (with a runtime of 371.3s). [41] proposed an ant colony optimization (*ACO*) algorithm and assessed the *ACO* performance with branch and bound with instances up to 100 jobs. In addition, *ACO* algorithm was assessed with instances up to 1,000 jobs and compared its results with *GA* and *SA* algorithms results reported in [39]. It was reported that *ACO* algorithm outperformed *GA* and *SA* algorithms in large instances. [1] addressed the static m identical parallel machines scheduling problem with a common server and sequence dependent setup times. A mixed integer linear programming (*MILP*) model, SA and GA are presented to minimize the makespan for the problem. The performance of the proposed *MILP* model, *SA* and *GA* based solution approaches are compared with the performance of basic dispatching rules such as, shortest processing time first (*SPT*) and longest processing time first (*LPT*) over a set of randomly generated problem instances. The experimental results show that the proposed *GA* is generally more

productive and efficient in solving this problem compared to the proposed *MILP* model, *SA*, *SPT* and *LPT*. [42] proposed two algorithms namely TS and particle swarm optimization (*PSO*) to solve the general problem with large-scale instances up to 1,000 jobs. The results were then evaluated with *GA*, *SA* and *I-L* algorithms and reported in [40].

Scheduling problems usually take into account the static environment with a job set, deterministic processing times and no unexpected occurrences during schedule execution [43]. Allahverdi and Mittenthal [44] found dual-criteria scheduling on a two-machine flow-shop subject to random breakdowns with regard to the objective functions of both makespan and lateness. In a two-machine flow shop, they established an exclusion criterion when both machines were exposed to random breakdowns and showed that the longest processing time and the shortest processing time orders were optimal for both parameters in a two-machine flow shop when the first and second machine suffered stochastic breakdowns respectively. Sun and Xue [45] implemented a flexible adaptive production scheduling system to adjust the schedules that were originally created if these schedules could not be fulfilled due to changes in the production order include the deletion of an order that a customer cancels and the addition of an order that must be fulfilled in a short time, and machine breakdowns of machines and sudden sickness of worker.

Kaabi *et al.* [46] examined the problem of single machine scheduling where machine maintenance had to be carried out at certain times and therefore the machine was not available for maintenance. The lateness of the jobs was minimized in the production portion and the earliness/lateness of maintenance was minimized in the case of maintenance part. Lee and Kim [47] studied the problem of scheduling jobs on a single machine requiring periodic maintenance to minimize the number of late jobs.

Mellouli *et al.* [48] studied the identical parallel machine scheduling problem with a planned maintenance cycle on each machine to minimize completion times sum. They suggested three specific methods for solving the problem: a branch-and-bound method, a dynamic programming-based method and mixed integer linear programming methods. Liao *et al.* [49] considered two parallel machines problem where one machine was not available during a period of time. The available time period has been set and known in advance. Probably because it required preventive maintenance or periodic repair, a machine was not available. The objective was aimed at reducing making time. Saidy and Fard [50] addressed machine scheduling with availability constraints for different environments, constraints and performance measures for both deterministic and stochastic situations. Berrichi *et al.* [51] proposed a new bio-objective approach to the joint production and maintenance scheduling issue. This method allows decision makers to find acceptable options between the goals of development and the goals of maintenance. Models of reliability are used to consider the maintenance aspect of the problem. The purpose was to maximize the two requirements simultaneously: minimizing makespan for the output and minimizing system unavailability for maintenance. Berrichi *et al.* [52], Berrichi and Yalaoui [53] presented an ant colony optimization approach to solve the joint production and maintenance scheduling problem. This approach is developed to deal with the proposed model in [51] for the parallel machine. Xu and Yang [26] presented a computational programming model for two parallel scheduling machines where one machine is often unavailable, jobs are non-preemptive. Yoo and Lee [54] considered a problem of scheduling on parallel machines where each machine needs maintenance once over a given time period. The study's objective is to find a coordinated schedule for jobs and maintenance tasks to reduce the scheduling costs measured by either one of several objective measures, such as makespan. Shen and Zhu [55] studied a parallel-machine scheduling problem with preventive maintenance. Under deterministic availability constraints, Kaabi and Harrath [56] investigated the uniform parallel machine scheduling issue.

According to the reviewed literature, the studied problem was not addressed in previous works (to the best of our knowledge). The already studied problems represent particular cases of the current one. Indeed, maintenance constraints and single server resource were not considered simultaneously in these studies. In addition, and based on the presented literature, the proposed methods to solve the already addressed problems are of two categories: 1) Exact methods and 2) approximate methods. The approximate methods are composed of heuristics and metaheuristics. Since the already studied problems are NP-Hard, then the exact methods are used rarely to solve small size instances. This is because of the large consumed time while solving these problems. The heuristics are a low time consuming but with moderate solution quality. The metaheuristics are widely used due to the good quality of the produced solutions. Therefore, in this study metaheuristics will be developed to solve the current problem. Among these metaheuristics the GA, SA, and TS which were shown to be efficient solving the particular cases of the current studied problem.

III. PROBLEM FORMULATION

The studied problem is the identical parallel machines scheduling problem with common server. The maximum completion time (makespan) is the objective to be minimized. The machines are subject to periods of unavailability, which are intended to preventive maintenance. The problem is denoted as $P_m, S1|Maint|C_{max}$ using standard scheduling notation.

This problem can be formulated as follows. A set of n jobs $J = \{J_1, \ldots, J_n\}$ should be processed by a set of m identical parallel machines $M = \{m_1, m_2, \ldots, m_m\}$. Each job $j \in J$ requires a single operation and may be processed on any one of the m parallel machines. In the current study, the number of machines is restricted to $m = 2$. Each job should be processed on exactly one machine and all jobs are available at time zero. The setup (or equivalently, loading) is carried

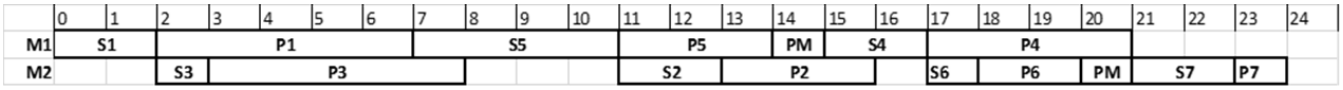

FIGURE 1. Gantt chart for a feasible schedule of example 1.

out by a common server. The loading procedure for a job $j \in J$, requires both the server and the machine for s_j time units which is known in advance. The server which could be a robot, cannot perform setup operation for a machine if the machine is processing a job or under *PM* operation (machine unavailability), whereas the machine can process a job without the server once the setup is completed. The processing time of job j assigned to any one of the machines m is p_j which is known beforehand. Processing times for all jobs are deterministic. The machines/server can only process one job/setup at any given time. Preemption is not allowed in the system. The availability interval of machines is deterministic and known in advance. The duration of *PM* activities on a specific machine is assumed to be deterministic and known in advance. The machine after maintenance will be in its best operational condition (i.e., as good as new). If the planning horizon is longer than the *PM* intervals, each machine would require at least one preventive maintenance (*PM*) operation. The *PM* activities cannot be delayed due to the fact that failure may occur, but it can be done early. The objective is to assign jobs to machines and to sequence setups on the server while considering *PM* operations in order to minimize the makespan of the system, i.e., the time of completion time of the final job in the system. A summary of notations is presented below.

n : Number of jobs.
m : Number of machines.
j : Job index, $j = 1, \ldots, n$.
i : Position index (for the single server), $i = 1, \ldots, n$.
k : Machine index, $k = 1, \ldots, m$.
p_j : Processing time of job j.
s_j : Setup time of job j.
t_0^k : The time for a *PM* action on machine k.
t_{PM}^k : The mean time of performing a *PM* on machine k.
C_{max} : The maximum completion time (Makespan).

We consider that all machines can be in one of two states, operational or down for maintenance. When any machine is in the operational state, it means that the machine is capable of processing jobs. However, when any machine is in the down for maintenance state, the machine is not working until *PM* is finished. The conducted *PM* maintenance plan is a time-based plan, meaning it is based on the operational time (or the age) of the machine. The machine age, which is the machine's operational time counter, should be set to zero after maintenance. For each machine m_k, we assumed that the time for *PM* action t_0^k and the required time to conduct that *PM* action on the machine (t_{PM}^k) are given. Then, using the given data, jobs are scheduled according to the periods of machine availability along with the single server.

Example 1: The following example demonstrates the problem's schedule construction. Consider two identical parallel machines (m_1 and m_2) with a set of seven jobs where the setup and processing times are presented in Table 1.

TABLE 1. Process and set-up times of an example 1.

j	1	2	3	4	5	6	7
s_j	2	2	1	2	4	1	2
p_j	5	3	5	4	3	2	1

The plans for *PM*, include the time at which a *PM* action is required, and the required time for conducting a *PM* action are provided in Table 2. The provided times are based on the operational machine times of the machines, i.e., idle times should not be included in the scheduling decisions for *PM* actions.

TABLE 2. The PM plans for machines 1 and 2.

	t_0^k	t_{PM}^k
m_1	10	1
m_2	10	1

Then, we can see that machine m_1 is available for 10 units of operational time (idle times are not included) before maintenance, and then is down for 1 unit of time for *PM*. Similarly, m_2 is available for 10 units of operational time, and then down for 1 unit of time for *PM*. Let the current job sequence π be *{1,3,5,2,4,6,}*. According to the given sequence, the jobs are assigned to the most available machine. The server should first load the job before operating it on the machine. The constructed schedule is presented on a Gantt chart as shown in Figure 1.

From the set π, job 1 is selected to be operated first and assigned to either machine m_1 or m_2 since both machines are available on the starting point ($t = 0$), here job 1 is assigned to m_1. The server has to load job 1 to machine m_1, which takes two unites of time (s_1). Once the server complete loading job 1 to machine m_1, job 1 is processed immediately for five units of time (p_1). In the meantime, since machine m_2 is available, the server starts to prepare it to receive the second selected job i π which is job 3; the preparation of machine m_2 takes one unit of time (s_3). The server will start loading job 3 after completing the loading process for job 1 (at $t = 2$), and then complete loading job 1 at point $t = 3$ on the timeline. Job 3 will be operated on machine m_2 for 5 units of time. The server will wait until one of the two machines is available to receive the third job in π (job 5). At point $t = 1$ on the

timeline, machine m_1 is available. The same procedure will continue until all jobs in π are scheduled.

Since the machines should have a *PM* after 10-unit of time, the *PM* for m_1 will be performed at $t = 14$ before loading job 4 despite that m_1 has been operated for only 8-unit of time. This is due to the fact that the next scheduled job, job 4, needs 4-units of time, which exceeds the period of availability and the *PM* should not be after 10 units of operating times for the possibility of failure. The *PM* for m_2 will be performed at $t = 23$ after 10 units time of operating. In this way, the constructed schedule is jobs *{1,5,4}* on machine m_1 and jobs *{3,2,6,7}* on machine m_2. The constructed schedule is presented on a Gantt chart as shown in Figure 1. The makespan value (C_{max}) is 23.

We should mention here that the machines availability periods are going to affect the generated schedule, since more waiting times will occur due to *PM* activities. For example, for machine m_1, the server waits from (14 -15) period to load m_1 while it is under *PM*. The same occurred for m_2 in period (20 – 21). In addition to waiting times due to *PM* activities, from the preceding example, it can also be noticed that two waiting types of waiting could happen in the schedule, which could probably be that the server is waiting for one of the machines to be available (either when the machine is working or under *PM*) or that one of the machines is waiting for the server availability. These two issues (the two types of waiting) can be encountered in Figure 1. For example, in periods (3-7, 13-15, and 18-21) the server waits for machine availability. Machine m_2 waits for server to be available in periods (8-11 and 16-17).

Proposition 1: The studied Problem $P_2, S1|Maint|C_{max}$ is NP-Hard in strong sense.

Proof: Indeed, a particular case $P_2, S1||C_{max}$ is NP-Hard in strong sense [34]. ■

IV. LOWER BOUND

A lower bound is proposed based on an existing one in [37]. The reported lower bound considers the scheduling problem of identical parallel machines with common server to minimize C_{max}. The proposed *LB* takes into account the periods in which machines are not available due to preventive maintenance actions. In the proposed *LB*, the unavailability periods of each machine have been added to that one from the literature. The reported lower bound in [37] is described as follow:

$$LB_{lit} = max\{LB_1, LB_2, LB_3\}$$

where:

$$\begin{aligned} LB_1 &= \frac{1}{2}\left\{\sum_{i\in J}(s_i + p_i) + min_{i\in J}(s_i)\right\} \\ LB_2 &= \sum_{i\in J}(s_i) + min_{i\in J}\{p_i\} \\ LB_3 &= min_{i\in J}\{s_i + p_i\} \end{aligned}$$

So, adding the unavailability periods for each machine, a new lower bound *LB* is obtained and expressed in **Proposition 2**.

Proposition 2: If

$$\begin{aligned} LB_1 &= \frac{1}{2}\left\{\sum_{i\in J}(s_i + p_i) + min_{i\in J}(s_i)\right\} \\ &\quad + t_{pm} \times \left\lfloor \frac{1}{2}\left(\sum_{i\in J}(p_i)/t_0\right)\right\rfloor \\ LB_2 &= \sum_{i\in J}(s_i) + min_{i\in J}\{p_i\} \\ &\quad + t_{pm} \times \left\lfloor \frac{1}{2}\left(\sum_{i\in J}(p_i)/t_0\right)\right\rfloor \\ LB_3 &= \sum_{i\in J}(s_i) + min_{i\in J}\{p_i\} + t_{pm} \times \left\lfloor \frac{1}{2}\left(\sum_{i\in J}(p_i)/t_0\right)\right\rfloor \end{aligned}$$

Then

$$LB = \max\{LB_1, LB_2, LB_3\}$$

is a valid lower bound for the problem $P_m, S1|Maint|C_{max}$, where t_0 is the available time (required time before a *PM* action is needed) and t_{pm} is the *PM* required time.

Proof: As maximum completion time can be calculated as follows:

$$C_{max} = \max_{i\in j}(C_i); \quad i \in J \tag{1}$$

where C_i is the job completion time i. And, as we have only single server and both machines are subject to maintenance, then

$$C_i \geq p_i + s_i + w_i + t_{pm} \tag{2}$$

where p_i is the processing time of job i, s_i is the setup time of job i, w_i is the waiting time if the server is busy, and t_{pm} is time for PM if the machine maintained before job i. Thus, according to equation 2 and 3, we have:

$$C_{max} \geq 0.5 \times \sum_{i\in J}(p_i + s_i)$$

and

$$C_{max} \geq 0.5 \times \left(\sum_{i\in J}(p_i + s_i) + \min_{i\in j}\{s_i\}\right) \tag{3}$$

As machines are subject to *PM*, there is a possibility of at least one *PM* action during the scheduling horizon. Considering that, we have a minimum *PM* time as follows:

$$t_{pm} \times \left\lfloor 0.5 \times \frac{\sum_{i\in J}(p_i)}{t_0}\right\rfloor \tag{4}$$

where t_0 is the machine availability time before a *PM* action is needed. As per equations 2 and 3, equation 4 can be rewritten as follows:

$$C_{max} \geq 0.5 \times \left(\sum_{i\in J}(p_i + s_i) + \min_{i\in j}\{s_i\} + t_{pm} \times \left\lfloor 0.5 \times \frac{\sum_{i\in J}(p_i)}{t_0}\right\rfloor\right)$$

Thus, $C_{max} \geq LB_1$.

Similarly, for LB_2 and LB_3, we have $C_{max} \geq LB_2$ and $C_{max} \geq LB_3$. ■

By applying the proposed LB for this **example 1** we get a lower bound of 20 as follows: $LB_1 = 20$, $LB_2 = 3$, $LB_3 = 8$, and $LB = \max\{20, 3, 8\} = 20$.

V. PROPOSED METAHEURISTICS

In this section, three metaheuristics are proposed. These metaheuristics are the SA, the TS, and the GA.

A. SIMULATED ANNEALING

SA algorithm, first proposed in 1982 and published in 1983 by [58] is a local search procedure capable of escaping from local optimum to solve combinatorial optimization problems. The *SA* is one of the popular metaheuristics effectively used in various combinatorial optimization problems. *SA* is a stochastic algorithm that allows for solution degradation under some conditions. The objective is to escape from local optima to delay the convergence. *SA* is a memoryless algorithm in the sense that the algorithm does not use any information gathered during the search. From an initial solution, *SA* proceeds in several iterations.

The *SA* improves a solution by iteratively moving the current solution S to a neighborhood solution $\acute{S}$, generated randomly. If $\acute{S}$ is better than S, the movement from S to $\acute{S}$ is accepted, i.e. S is replaced by $\acute{S}$. If $\acute{S}$ is worse than S, it is accepted with a probability of $e^{-\Delta E/T}$, called an uphill move, where ΔE represents a difference between the objective function values of S and $\acute{S}$, and T is a parameter called the temperature. As the algorithm advances, the probability that such moves are accepted decreases. This probability follows, in general, the Boltzmann distribution:

$$P(\Delta E, T) = e^{-\Delta E/T}$$

The higher the temperature, the more likely it is for a worst move to be accepted. At a given temperature, the lower the objective function increase is, the more significant the probability of accepting the move. T is initially set to T_0, T_{max} at the beginning and is decreased after every iteration. The algorithm is terminated if temperature T reaches to T_f, T_{min} as given in Algorithm 1. In addition to the current solution, the best solution found since the beginning of the search is stored.

Few parameters control the search progress, which are the temperature and the number of iterations performed at each temperature. The main elements of *SA* can be summarized as follows:

- The acceptance probability function: It is the main element of *SA* that enables non-improving neighbors to be selected.
- The cooling schedule: The cooling schedule defines the temperature at each step of the algorithm. It plays an essential role in algorithm efficiency and the effectiveness.

Algorithm 1 SA Algorithm Template

Input: Cooling schedule.
$S = S_0$; // Generation of the initial solution
$T = T_{\max}$; // Starting temperature
Repeat
  Repeat // At a fixed temperature
    // Generate a random neighbor S'
    $\Delta E = f\left(S'\right) - f(S)$;
    If $\Delta E \leq 0$ then $S := S'$
    //Accept the neighbor solution
    Else accept S' with a probability $e^{-\Delta E/T}$
  Until Equilibrium condition
    // e.g. a given number of iterations executed at each temperature T
    $T = \mathrm{g}(\mathrm{T})$; // Temperature update
Until Stopping criteria satisfied // e.g. $T < T_{min}$
Output: Best solution found

Regarding the stopping condition, theory suggests a final temperature equal to 0. In practice, the search can be stopped when the probability of accepting a move is negligible. The following stopping criteria may be used:

- Reaching a final temperature T_F is the most popular stopping criteria. This temperature must be low (e.g., $T_{min} = 0.01$), which is used in our code.
- Achieving a predetermined number of iterations without best-found solution improvement [61].
- When the objective function reaches a pre-specified threshold value (e.g. lower bound).
- Predetermined number of evaluations.

B. TABU SEARCH (TS) ALGORITHM

In 1989, [62] proposed a new local search method, called Tabu Search (*TS*). *TS* is an adaptive higher-level heuristic designed to guide other local search approaches to continue exploration without becoming confounded by the absence of improving moves, and without falling back into local optima from which it previously emerged. This is accomplished using a certain number of memories. It allows the deterioration of the current solution by accepting a worse solution when moving from one iteration to the subsequent one. Even if it is not improving the current solution, the new current solution will be the best solution found in the neighborhood. Obviously, this procedure can cycle, i.e., visit some solution more than once. In order to avoid this phenomenon, a tabu criterion is introduced to identify moves which are expected to lead to cycles. Such moves are then declared tabu and are added to the tabu list. As, however, forbidding certain moves could prohibit visiting "interesting" solutions, an aspiration criterion differentiates the potentially interesting moves from the forbidden ones. The basic idea of *TS* is to "remember" which solutions have been visited throughout execution of the algorithm, as to derive the promising directions for further search. Thus, the memory and not only the local investigation

of the neighborhood of the current solution drives the search as given in Algorithm 2.

Algorithm 2 TS Algorithm Template

Input: $S = S_0$; // Initial solution
Initialize the tabu list, medium-term and long-term memories;
Repeat
Generate a set "A" of solutions;
Find best neighbor $S^{'}$ of A; // non tabu or aspiration criterion holds
$S = S^{'}$;
Update tabu list and aspiration conditions;
If $f(S) < f(S_0)$ **Then** $S_0 := S$;
Until Stopping criteria satisfied
Output: S_0 is the best-found solution.

The main parameters of *TS* are the neighborhood structure, number of candidate solutions, the tabu list, the aspiration criterion and stopping criteria which are described below:

- **Initial solution**: since the *TS* is improvement class algorithms, a starting solution (initial solution) is needed to start out.
- **Neighborhood structure**: pairwise interchange and backward-shifted reinsertion are two classical and useful operators to construct promising neighborhood structures for generating new candidate solutions.
- **Number of generated candidates**.
- **Tabu list**: a list of forbidden or tabu moves, i.e., moves which are not allowed to be applied to the current solution. The goal of using tabus is to avoid cycles (prevent the search from revisiting previously visited solutions). The tabu list size may take different forms:
  - Static: The size of the tabu list a static value and determined in advance.
  - Dynamic: The size of the tabu list may change during the search without using any information on the search memory.
  - Adaptive: the size of the tabu list is updated according to the search memory. For instance, the size is updated upon the search performance in the last iterations [63].
- **Aspiration criteria**: as mentioned earlier (accept a forbidden move if it results in a solution that is better than the best solution found so far).
- **Stopping criteria**: in theory, the search could go on forever, unless the optimal value of the problem at hand is known beforehand. In practice, the search has to be stopped at some point. The most commonly used stopping criteria in TS are:
  - After a fixed number of iterations (or a fixed amount of CPU time);
  - After some number of consecutive iterations without an improvement in the objective function value (the criterion used in most implementations);
  - When the objective function reaches a pre-specified threshold value.

C. GENETIC ALGORITHMS (GA)

The Genetic Algorithms (*GA*), were developed by John Holland and his collaborators in the 1960s and 1970s. *GA* imitates the mechanics of natural evolution and selection [64]. It imitates the biological reproduction natural selection processes to solve for the 'fittest' solutions. Genetics is a biological term. Biologically, genes of a good parent produce better offspring. The same concept underlies the development of *GA*. GAs represent one branch of the field of study called evolutionary computation EAs. They are based on the evolution of a population of individuals. In the 1980s, Golberg [65] applied to optimization and machine learning. In order to apply *GA* to a problem, generally the problem's solution space is represented by a population of structures where each structure is a possible solution to the problem. A fitness value is associated with each structure. Then a certain number of structures are chosen to form the initial generation. The structures of the next generation are generated by applying simple genetic operators to the parent structures selected from the existing generation. According to the idea that 'good parents produce better offspring', a structure with higher fitness value in the current generation will have higher probability of being selected as a parent (similar to the concept of survival) [66]. This iteration represents a generation [59]. When we repeat this process, we can observe a continuous improvement in the structure's performance from one generation to the next. This process is iterated until a stopping criterion hold as given in Algorithm 3.

Algorithm 3 GA Algorithm Template

Generate $(p(0))$; /* Initial population */.
$t = 0$;
While not Termination Criterion $(P(t))$ **Do**
Evaluate $(P(t))$;
$p^{'}(t) =$ **Selection** $(P(t))$;
$p^{'}(t) =$ **Reproduction** $p^{'}(t)$; **Evaluate** $p^{'}(t)$;
$p(t+1) =$ **Replace** $(p(t), p^{'}(t))$;
$t = t + 1$;
End While
Output: Best individual or best population found

Since GAs are designed to simulate a biological process, much of the relevant terminology is borrowed from biology. The following are the terminology used in search components for designing the GAs.

1. **Representation:** This is a common search component for all metaheuristics. In *GA*, the encoded solution is referred as *chromosome* while the decision variables within a solution (chromosome) are genes. The encoding method of the string is comprised of the jobs' numbers

TABLE 3. Data generation.

Case	n	Processing time p_j ~(a , b)	setup times s_j	Available period t_0	PM period t_{pm}
Case 1	(10, 20, 30, 40,50, 100, 200, 300, 400, and 500)	(20,50)	(0, 0.25 ×b)	$(a+b)*n/6$	$(a+b)/2$
Case 2		(20,50)		$(a+b)*n/4$	
Case 3		(20,100)		$(a+b)*n/6$	
Case 4		(20,100)		$(a+b)*n/4$	

2. **Objective Function:** This is a common search component for all metaheuristics. the term fitness refers to the objective function.
3. **Selection strategy:** The selection strategy addresses the following question: "Which parents for the next generation are chosen with a bias toward better fitness?"
4. **Reproduction strategy**: The reproduction strategy consists in designing suitable mutation and crossover operator(s) to generate new individuals (offspring).
5. **Replacement strategy**: The new offspring compete with old individuals for their place in the next generation (survival of the fittest).

D. INITIAL SOLUTION

Since the three proposed algorithms, *SA*, *TS* and *GA* are improvement class algorithms, a starting solution or what so called initial solution is needed to start out the proposed metaheuristics. These metaheuristics will improve the initial solution through an iterative improvement approach until reach a high-quality feasible solution. In this study, the starting solution was generated using the longest processing time (*LPT*) dispatching rule, in which the jobs are sequenced in non-increasing processing time, for all proposed metaheuristics (*SA*, *TS* and *GA*).

E. DATA SET GENERATION

In order to investigate the proposed algorithms' performance (*TS, GA,* and *SA*), an experimental study is conducted using a set of benchmark instances. The generated data set for this study was randomly generated based on the previous studies' designed data set [49]. As stated in [49], processing times (p_j) were uniformly distributed in the intervals ($a = 20, b = 50$) and ($a = 20, b = 100$), setup times (s_j) were distributed uniformly in the interval $(0, b \times L)$, where L is a factor called the server's load value and was set to 0.25 and the b is corresponding to the generation of p_j, b equals to 50 and 100. For maintenance activities, some extensions have been conducted. The time t_{PM}^k, required to do a *PM* action on machine k was set to $(a + b)/2$. Two cases for machine k availability period t_0^k were generated, $(a+b) * n/6$ and $(a+b) * n/4$ where n is the number of jobs. Considering the machines are identical, t_0 and t_{pm} will be used from now on instead of t_0^k and t_{PM}^k, respectively.

For every combination of processing time and availability period (t_0), different problem sizes of jobs ($n = 10, 20, 30, 40,50, 100, 200, 300, 400$, and 500) were generated. A total of 400 problems have been obtained (40 types). A summary of the generated data is illustrated in Table 3.

Selecting (p_j) randomly and uniformly from ($a = 20, b = 50$) and ($a = 20, b = 100$) is explained by the need to have a large variety of processing times. Indeed, ($a = 20, b = 50$) represents the small processing time, while the second one ($a = 20, b = 100$) represents the large processing times. Since the processing times are distributed uniformly in (a, b), then $(a + b)/2$ represents the average processing time for each job. This average processing time is selected to be the duration of the preventive maintenance period. It is worth noting that $\left\{\frac{(a+b)}{2}\right\} * n$ is the average total processing time (for the n jobs), and $\frac{\left\{\frac{(a+b)}{2}\right\} * n}{2} = (a+b) * n/4$ is the average load for each machine. The period before preventive maintenance t_0^k is selected according to two scenarios. The first one is $t_0^k = (a+b) * n/4$, which means that the preventive maintenance begins when all the jobs are processed (in average). The second scenario corresponds to $t_0^k = ((a+b) * n/2)/3 = (a+b) * n/6$, which involves that the preventive maintenance begins while the processing of jobs is not yet finished.

F. PARAMETERS TUNING

In this section, several design parameters for the proposed metaheuristics are investigated and adjusted. For each metaheuristic algorithm, different parameter combinations return different results, meaning that the used parameter values in each algorithm affect its performance.

To classify the appropriate settings of each design parameter, a pilot run has been conducted based on screening and literature. Taguchi design of $L25$ was used to study the effect of the proposed algorithms parameters on the C_{max} and CPU time and the best settings for each proposed metaheuristic parameters are determined. A summary of the main parameters and their levels regarding *TS*, *SA* and *GA* are shown in Table 4.

VI. RESULTS AND DISCUSSIONS

In this section, performance of the proposed *SA*, *TS* and *GA* algorithms is evaluated by conducting computational experiments with 40 problem types of which have been randomly

TABLE 4. Summary of the main parameters and levels for SA, TS and GA.

Algorithm	Parameter	Considered Values	Selected Values
TS	Tabu List size	4, 8, 10, 12, 16,20	8
	No. candidates	10,20,30,40,50,60	30
	Neighborhood structure	Swap	
	Stopping Condition	Number of Evaluations (100K, 400K,700K,1M,1.3M) or LB	1M or *LB*
SA	Initial Temperature T_0	10, 60, 110, 160, 210	10
	Neighborhood Structure	Swap [1]	
	No. of neighbors	10, 50, 90, 130, 170	10
	Cooling Function	Geometric	
	Cooling rate α	0.91, 0.93, 0.95, 0.97, 0.99	0.95
	Equilibrium Condition	Static	
	Stopping Condition	Number of Evaluations (100K, 400K,700K,1M,1.3M) or LB	1M or *LB*
GA	Population Size	20, 60, 100, 140,180	100
	Selection Method	Roulette Wheel	
	Crossover Operator	Position Based Crossover (PBX)	
	Crossover Rate (Pc)	0.4, 0.55,0.7,0.85,0.95	0.95
	Mutation Operator	Swap, Insertion, and Inversion	
	Mutation Probability (Pm)	0.001, 0.005,0.01,0.05,0.055	0.055
	Mutation Rate	0.1 0.3 0.5 0.7 0.9	0.1
	Selection Pressure	2,6,10,14,18	14
	Stopping Condition	Number of Evaluations (100K, 400K,700K,1M,1.3M) or LB	1M or *LB*

generated and each problem has been replicated ten times (400 instances in total). The proposed algorithms have been coded using MATLAB software and the computational experiments for all instances have been conducted on a Dell computer with the following specifications. Processor: Intel (R) Core^{TM}i7- 4702MQ CPU at 2.2 GHz; RAM: 8 GB.

Based on the proposed lower bound (*LB*), a gap between the *LB* and the obtained C_{max} can be used to measure the proposed algorithms performance. The gap can be defined as follows:

$$Gap = C_{max}/LB$$

The computational results of minimum and average values of the *gap* and CPU time of the *SA*, *TS* and *GA* are shown in Table 5. Results are also presented. The following notations are used in Table 5:

- n denotes the number of jobs.
- p_j denotes the processing time.
- t_0 denotes the available period.
- Gap_{min} denotes the minimum value of the gap.
- Gap_{avg} denotes the average value of the gap.
- T_{min} denotes the minimum CPU time to reach the specified number of evaluations or reach LB.
- T_{avg} denotes the average CPU time to reach the specified number of evaluations or reach *LB*.

It can be seen from Table 5 that there are four cases based on the processing time and machine available times.

In case 1, the processing time is $\boldsymbol{p_j \sim (20, 50)}$ and $\boldsymbol{t_0 = (a+b) * n/6}$ and $\boldsymbol{a = 20}$ ***and*** $\boldsymbol{b = 50}$. In this case *SA* reaches ***LB*** in one instance where $n = 10$ and *GA* reaches *LB* in two instances where $n = 10$ and *20* while *TS* failed to reach ***LB*** in any instance. *GA* outperforms *SA* for instances with n less than 200 and outperforms *TS* for all instances in the minimum and average gap performances. The minimum and average gap of *GA* are very close to each other. *SA* outperforms *TS* in all instances except when n is 20 jobs and outperforms *GA* for large instances with $\boldsymbol{n}$ greater than to 100. In general, *GA* does better for small instances while *SA* has a good performance for large instances. Additionally, the gap becomes higher with increase in instance size and it is hard to reach *LB* for all algorithms as the number of jobs greater than 20 jobs. Regarding CPU time, *TS* has the least CPU time among the three algorithms except with when $n = 10$ and 500. *SA* outperforms *TS* and when $n = 10$ and 20 where *GA* reaches *LB*. *GA* takes much time than that of *SA*

TABLE 5. Computational results of SA, TS and GA.

No.	n	p_j	t_0		*SA*				*TS*				*GA*			
					Gap		Time (s)		Gap		Time (s)		Gap		Time (s)	
					Gap_{min}	Gap_{avg}	T_{min}	T_{avg}	Gap_{min}	Gap_{avg}	T_{min}	T_{avg}	Gap_{min}	Gap_{avg}	T_{min}	T_{avg}
1	10	*U(20,50)*	*(a+b)*n/6*	117	1.0000	1.1107	0.15	49.95	1.0043	1.2090	32.81	33.01	1.0000	1.0000	0.11	2.56
2	20			233	1.0784	1.2396	66.77	67.05	1.0412	1.1489	33.86	34.01	1.0000	1.0014	3.44	36.13
3	30			350	1.0749	1.2150	64.23	64.49	1.0868	1.2698	34.94	35.08	1.0015	1.0058	74.43	74.63
4	40			467	1.0508	1.2304	65.24	65.51	1.0745	1.2284	36.00	36.22	1.0181	1.0271	76.58	76.69
5	50			583	1.0350	1.2020	66.50	66.67	1.1068	1.1826	36.93	37.25	1.0120	1.0451	78.52	78.73
6	100			1167	1.1811	1.2261	72.74	72.97	1.1608	1.2704	44.14	44.33	1.1097	1.1416	88.78	88.90
7	200			2333	1.1645	1.2144	83.23	83.53	1.2146	1.2775	55.40	55.62	1.1957	1.2143	109.09	109.24
8	300			3500	1.2017	1.2306	91.82	92.03	1.3029	1.3472	67.03	67.50	1.2335	1.2512	129.62	129.92
9	400			4667	1.1985	1.2272	101.78	102.11	1.3145	1.3437	94.93	95.16	1.2579	1.2683	150.31	150.58
10	500			5833	1.1836	1.2147	112.15	112.64	1.3383	1.3713	124.17	124.90	1.2858	1.2922	171.48	171.70
11	10	*U(20,50)*	*(a+b)*n/4*	175	1.0000	1.2005	0.13	55.85	1.0199	1.2095	32.19	32.62	1.0000	1.0030	0.69	36.38
12	20			350	1.0208	1.1815	59.67	60.08	1.0833	1.2350	33.58	33.90	1.0000	1.0035	17.50	64.85
13	30			525	1.1107	1.2392	61.10	61.34	1.0813	1.1869	34.89	35.11	1.0013	1.0111	74.32	74.53
14	40			700	1.1280	1.2435	61.82	62.28	1.0723	1.2109	35.64	35.94	1.0118	1.0291	76.37	76.58
15	50			875	1.0582	1.1969	66.12	66.37	1.1360	1.2287	37.25	37.45	1.0308	1.0556	78.50	78.67
16	100			1750	1.0936	1.1913	72.41	72.77	1.1554	1.2396	43.62	44.16	1.1190	1.1445	88.73	91.28
17	200			3500	1.1361	1.2132	83.10	83.42	1.2218	1.2683	55.59	55.77	1.1938	1.2154	109.12	109.32
18	300			5250	1.1223	1.1949	91.71	92.05	1.3154	1.3379	66.94	67.41	1.2372	1.2539	129.47	129.80
19	400			7000	1.1555	1.2032	101.79	102.03	1.3318	1.3605	95.07	95.23	1.2618	1.2663	150.47	150.67
20	500			8750	1.2029	1.2314	111.29	112.01	1.3313	1.3607	123.85	124.53	1.2765	1.2903	171.45	171.71
21	10	*U(20,100)*	*(a+b)*n/6*	200	1.0024	1.0736	58.24	58.78	1.0168	1.1618	32.75	33.08	1.0024	1.0058	69.92	70.22
22	20			400	1.0363	1.1783	59.80	60.43	1.0969	1.2406	33.94	34.17	1.0000	1.0031	7.39	47.09
23	30			600	1.0009	1.0484	64.11	64.31	1.0043	1.0912	34.73	34.97	1.0000	1.0037	54.97	71.89
24	40			800	1.0126	1.1125	65.13	65.44	1.0096	1.1554	36.09	36.24	1.0096	1.0185	76.46	76.67
25	50			1000	1.0068	1.1340	66.25	66.54	1.0506	1.1632	37.04	37.33	1.0169	1.0400	78.61	78.80
26	100			2000	1.0092	1.1070	72.69	72.83	1.0754	1.1612	44.16	44.31	1.1050	1.1302	88.80	89.13
27	200			4000	1.0223	1.1002	83.10	83.36	1.2277	1.2530	55.57	55.86	1.1826	1.2124	109.24	109.66
28	300			6000	1.0495	1.1136	91.71	91.90	1.2311	1.3027	67.36	67.65	1.2495	1.2574	129.97	130.14
29	400			8000	1.0748	1.1134	101.63	102.03	1.2699	1.3335	94.94	95.23	1.2586	1.2827	150.68	150.97
30	500			10000	1.0786	1.1241	111.91	112.74	1.3341	1.3603	124.37	124.90	1.2902	1.2959	171.88	172.10
31	10	*U(20,100)*	*(a+b)*n/4*	300	1.0083	1.1429	61.35	61.84	1.0000	1.1626	0.07	29.67	1.0000	1.0017	0.38	28.81
32	20			600	1.0014	1.1253	62.88	63.30	1.0122	1.1441	33.46	33.81	1.0000	1.0046	56.96	70.87
33	30			900	1.0323	1.1658	63.81	65.02	1.0704	1.1196	34.75	35.07	1.0000	1.0042	63.63	73.49
34	40			1200	1.0205	1.1313	64.30	64.53	1.0616	1.1443	35.98	36.20	1.0048	1.0219	76.49	76.60
35	50			1500	1.0171	1.1209	65.41	65.58	1.1662	1.1973	37.25	37.39	1.0132	1.0427	78.50	78.69
36	100			3000	1.0356	1.1621	71.61	71.77	1.1006	1.1491	44.00	44.22	1.1193	1.1347	88.81	88.93
37	200			6000	1.0795	1.1314	81.99	82.17	1.1786	1.2255	55.52	55.73	1.1703	1.2066	109.38	109.60
38	300			9000	1.0824	1.1255	90.38	90.62	1.2697	1.3072	67.02	67.28	1.2355	1.2518	129.40	129.57
39	400			12000	1.0576	1.1167	100.06	100.46	1.3259	1.3394	94.53	95.17	1.2620	1.2787	150.53	150.68
40	500			15000	1.0753	1.1149	109.87	110.41	1.3304	1.3629	123.45	124.43	1.2801	1.2931	171.40	171.64

TABLE 6. Effects of the processing times and availability periods on the gap.

	SA		*TS*		*GA*	
Factor	Gap_{min}	Gap_{avg}	Gap_{min}	Gap_{avg}	Gap_{min}	Gap_{avg}
n	Significant		Significant	Significant	Significant	Significant
p_j	Significant	Significant	Significant	Significant		Significant
t_0						
$n * p_j$						Significant
$n * t_0$		Significant				
$t_0 * p_j$						Significant

and CPU time gap between *GA* and *SA* and *TS* becomes very high with increasing in the instances size.

In case 2, the processing time is $p_j \sim (20, 50)$ and $t_0 = (a+b) * n/4$ and $a = 20$ *and* $b = 50$. In this case *SA* reaches *LB* in one instance where $n = 10$ and *GA* reaches *LB* in two instances where $n = 10$ and *20* while *TS* failed to reach *LB* in any instance. *GA* outperforms *SA* for instances with n less than 100 and outperformed *TS* for all instances in

TABLE 7. Effects of the processing times and availability periods on the CPU times.

	SA		TS		GA	
Factor	$Time_{min}$	$Time_{avg}$	$Time_{min}$	$Time_{avg}$	$Time_{min}$	$Time_{avg}$
n	Significant	Significant	Significant	Significant	Significant	Significant
p_j	Significant					
t_0			Significant	Significant		
$n * p_j$	Significant					
$n * t_0$				Significant		
$t_0 * p_j$				Significant		

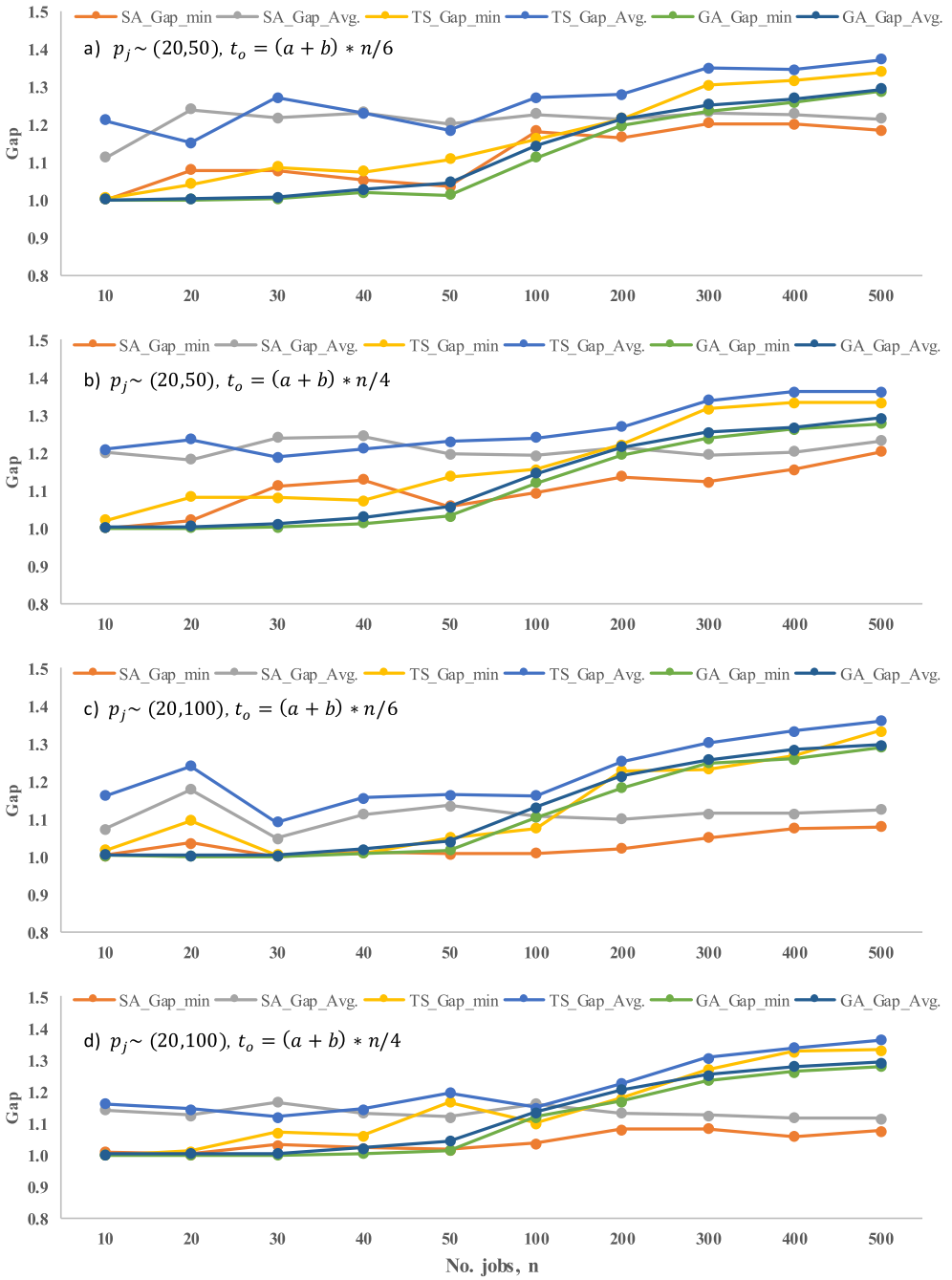

FIGURE 2. Minimum and average gap results of *SA*, *TS* and *GA*.

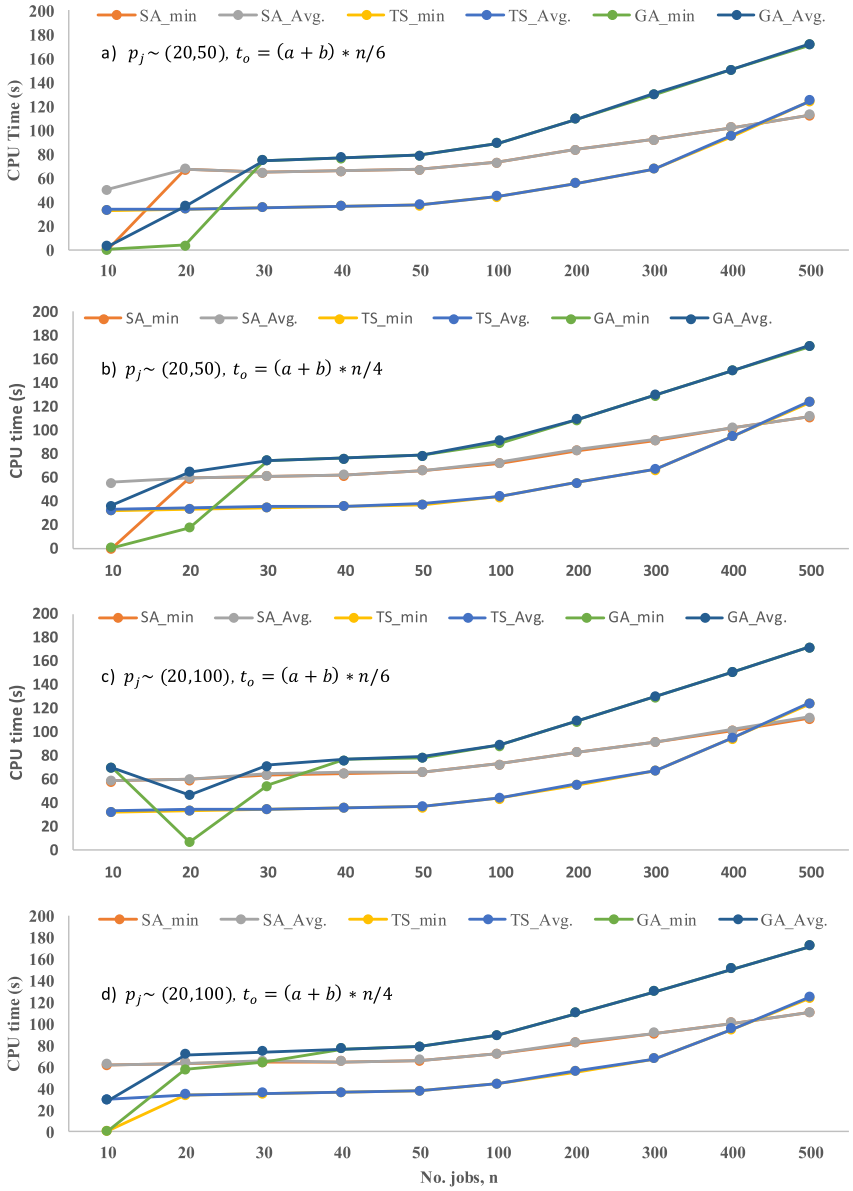

FIGURE 3. Minimum and average CPU time results of *SA*, *TS* and *GA*.

the minimum and average gap performances. The minimum and average gap of *GA* are very close to each other. *SA* outperforms *TS* in all instances except when n equals to 20 and 30 jobs and outperforms *GA* for large instances with n greater than to 50 in terms of minimum gap and with n greater than to 100 in terms of average gap. In general, *GA* does better for small instances while *SA* has a good performance for large instances, and it was hard to reach *LB* for all algorithms as the number of jobs was greater than 20 jobs. In addition, the gap becomes high with increase in instance size. Regarding CPU time, *TS* has the least CPU time among the three algorithms except with when $n = 10$ and 500. *SA* outperforms *TS* and when $n = 10$ and 20 where *GA* reaches *LB*. *GA* takes much time than that of *SA* and CPU time gap between *GA*, *SA*, and *TS* becomes very high with increasing in the instances size.

In case 3, the processing time is $\boldsymbol{p_j \sim (20, 100)}$ and $\boldsymbol{t_0 = (a+b) * n/6}$ and $\boldsymbol{a = 20}$ ***and*** $\boldsymbol{b = 100}$. In this case *SA* and *TS* failed to reach *LB* while *GA* reach the *LB* in two instances where $n = 20$ and 30. *GA* outperforms *SA* for instances with $\boldsymbol{n}$ less than 100 and outperformed *TS* for all instances in the minimum and average gap performances. The minimum and average gap of *GA* are very close to each other. *SA* outperforms *TS* in all instances and outperforms *GA* for large instances with n greater than to 50 in terms of minimum gap and with $\boldsymbol{n}$ greater than to 100 in terms of average gap. In general, *GA* does better for small instances while *SA* shows good performance for all instances, and it was hard to reach

TABLE 8. Computational results of ABC.

No.	n	p_j	t_0		*ABC*			
					Gap		Time (s)	
					Gap_{min}	Gap_{avg}	T_{min}	T_{avg}
1	10	*U(20,50)*	*(a+b)*n/6*	117	1.0000	1.0017	0.13	26.26
2	20			233	1.0247	1.0485	35.11	51.59
3	30			350	1.0599	1.0837	69.33	69.56
4	40			467	1.0982	1.1210	70.91	71.10
5	50			583	1.1280	1.1539	72.51	72.70
6	100			1167	1.2000	1.2168	80.76	80.94
7	200			2333	1.2545	1.2733	96.16	96.39
8	300			3500	1.2788	1.2979	110.72	110.98
9	400			4667	1.2966	1.3139	126.05	126.35
10	500			5833	1.3119	1.3253	141.82	142.17
11	10	*U(20,50)*	*(a+b)*n/4*	175	1.0000	1.0075	0.41	46.12
12	20			350	1.0370	1.0662	38.59	62.47
13	30			525	1.0760	1.0901	67.71	67.94
14	40			700	1.0841	1.1230	69.10	69.43
15	50			875	1.1249	1.1468	72.31	72.52
16	100			1750	1.1951	1.2244	80.57	82.03
17	200			3500	1.2628	1.2774	96.11	96.37
18	300			5250	1.2997	1.3076	110.59	110.93
19	400			7000	1.3030	1.3130	126.13	126.35
20	500			8750	1.3220	1.3300	141.37	141.86
21	10	*U(20,100)*	*(a+b)*n/6*	200	1.0024	1.0070	64.08	64.50
22	20			400	1.0169	1.0429	33.60	53.76
23	30			600	1.0803	1.0932	59.54	68.10
24	40			800	1.1053	1.1331	70.80	71.06
25	50			1000	1.1220	1.1459	72.43	72.67
26	100			2000	1.2101	1.2287	80.75	80.98
27	200			4000	1.2807	1.2896	96.17	96.51
28	300			6000	1.2915	1.3107	110.84	111.02
29	400			8000	1.3164	1.3275	126.16	126.50
30	500			10000	1.3325	1.3402	141.90	142.42
31	10	*U(20,100)*	*(a+b)*n/4*	300	1.0000	1.0053	30.87	45.33
32	20			600	1.0271	1.0537	59.92	67.09
33	30			900	1.0530	1.0802	63.72	69.26
34	40			1200	1.0900	1.1292	70.40	70.57
35	50			1500	1.1285	1.1505	71.96	72.14
36	100			3000	1.2075	1.2235	80.21	80.35
37	200			6000	1.2651	1.2793	95.69	95.89
38	300			9000	1.2971	1.3124	109.89	110.10
39	400			12000	1.3217	1.3320	125.30	125.57
40	500			15000	1.3293	1.3386	140.64	141.03

LB for all algorithms as the number of jobs is greater than 20 jobs. In addition, the gap becomes high with increase in instance size. Regarding CPU time, *TS* has the least CPU time among the three algorithms except when $\boldsymbol{n} = 20$ where *GA* had less CPU time. *GA* takes much time than that of *SA* and CPU time gap between *GA, SA,* and *TS* becomes very high with increasing in the instances size.

In case 4, the processing time is $p_j \sim (20, 100)$ and $t_0 = (a + b) * n/4$ and $a = 20$ *and* $b = 100$. In this case *SA* fails to reach *LB* while *GA* reaches *LB* in three instances where $n =$ *10*, 20 and 30. *TS* reaches *LB* when $n =$*10*. *GA* outperforms *SA* for instances with n less than 100 and outperforms *TS* for all instances in the minimum and average gap performances. The minimum and average gap of *GA* are very close to each other. *SA* outperforms *TS* in all instances except when $n =$ *10* in regards to minimum gap and outperforms *GA* for large instances with n greater than 50 jobs in terms of minimum gap and with n greater than to 100 in terms of average gap. In general, *GA* does better for small instances while *SA* has a good performance for all instances in terms of minimum gap and it was hard to reach *LB* for all algorithms as the number of jobs was greater than 30 jobs. In addition, the gap becomes high with increase in instance size. Regarding CPU time, *TS* has the least CPU time among the three algorithms except

with when $n = 10$ and 500. *SA* outperforms *TS* and when $n = 10$ and 20 *GA* outperforms *TS*. *GA* takes much time than that of *SA* and CPU time gap between *GA, SA,* and *TS* becomes very high with increasing in the instances size.

According to Figures 2-3, and among the four cases, *GA* minimum and average gap were very close to each other because of selection strategy. *TS* minimum and average gap were far away from each other because it accepts non-improving solutions. With the increase in the process time interval, case 2 and 3, *SA* gap becomes smaller and close to 1.

General linear model was used to study the effect of number of jobs n, processing time $\boldsymbol{p_j}$ and machine available period $\boldsymbol{t_0}$ on the *SA*, *TS,* and *GA* performance. A statistical analysis results summary of the n, $\boldsymbol{p_j}$ and $\boldsymbol{t_0}$ effects on gap and CPU time are presented in Table 6 and Table 7, respectively.

A. COMPARISON OF TS, SA, AND GA WITH A RECENT METAHEURISTIC

In order to have a general picture of the performance of the proposed algorithms (SA, TS and GA), a comparison study with a recent metaheuristic is performed. Among the most recent metaheuristic algorithms, the Artificial Bees Colony algorithm (ABC) [68]. In the sequel a brief description of the ABC is presented.

In ABC, afood source location is considered as a solution. The nectar quantity of a food source is the fitness of the corresponding solution. The bees are subdivided into three categories, which are the employed bees, the onlooker bees, and the scout bees. The employed bees are charged with searching food sources. The onlooker bees are waiting in hive, where they are making a decision in order to choose a food source. The scout bees are performing a random exploration for new source of food. For each type of bees, a phase is assigned.

During the employed bee phase, for each solution $\boldsymbol{s} \in \boldsymbol{P}$, a new solution $\boldsymbol{ns}$ is created using the expression $\boldsymbol{ns} = \boldsymbol{s} + \boldsymbol{\theta}(\boldsymbol{s} - \boldsymbol{s}')$ with $\boldsymbol{P}$ the population, $\boldsymbol{\theta}$ a random generated number in $[-\mathbf{1}, \mathbf{1}]$, and $\boldsymbol{s}' \neq \boldsymbol{s}$ a random selected solution from $\boldsymbol{P}$. If $\boldsymbol{ns}$ is better than $\boldsymbol{s}$ in terms of nectar amount, then $\boldsymbol{ns}$ replaces $\boldsymbol{s}$. In this case, the bee forgets $\boldsymbol{s}$ and memorizes $\boldsymbol{ns}$. In the opposite case, the bee keeps in memory $\boldsymbol{s}$. In onlooker bee phase, a roulette selection is used to allow to each onlooker bee to select a food source. An adapted ABC for the studied problem [68] is coded in MATLAB software and the same set of the instances is used in the experimental study. The detailed numerical results are presented in Table 8.

A comparison study of ABC with the already proposed metaheuristics (SA, TS, and GA) is carried out. This study is using two performance measurements: 1) The average gap **GAP**, and 2) the average time **TIME**. The results of this study are displayed in Table 9. According to Table 9, the recent metaheuristic (ABC) is ranked third by a $Gap = 1.1886$ behind GA and SA. In addition, in terms of average time, ABC is once again ranked third with $Time = 87.22\,s$. This means that the proposed metaheuristics (SA, TS, and GA) still useful and could provide performant results compared to the recent proposed metaheuristics (ABC).

TABLE 9. Comparison of SA, TS, and GA to ABC.

	GAP	TIME
SA	1.1662	77.48
TS	1.2415	**56.20**
GA	**1.1252**	96.96
ABC	1.1886	87.22

VII. CONCLUSION

The current study considered a scheduling problem of two identical parallel machines with single server and availability constraints with the objectives of minimizing makespan. This problem has many wide range potential application areas in the manufacturing environment. A lower bound (*LB*) for the problem has been proposed. Three metaheuristics namely SA, TS and GA have been proposed. The best parameters settings of the proposed algorithms were conducted using pilot runs with a Taguchi design. The algorithms performance has been evaluated using 400 instances generated randomly based on the literature. Four cases of instances were studied in which the processing times and availability periods of the machines were different. The size of the instances, number of jobs, were up to 500 jobs. Along with the performance analysis of the proposed algorithms, the effect of varying processing times and availability periods on the performance of the developed algorithms was studied. An intensive experimental study shows the effectiveness and performance of the proposed metaheuristics. In addition, there is no dominance of a particular metaheuristic on the others. Each one of these algorithms is doing well in a certain subset of instances.

For future studies, more objective functions could be investigated such as total tardiness and total completion time. In addition, an arbitrary number of machines could be considered instead of two machines. Furthermore, new metaheuristics and exact methods will be proposed and tested.

REFERENCES

[1] A. Hamzadayi and G. Yildiz, "Modeling and solving static m identical parallel machines scheduling problem with a common server and sequence dependent setup times," *Comput. Ind. Eng.*, vol. 106, pp. 287–298, Apr. 2017.

[2] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *Eur. J. Oper. Res.*, vol. 187, no. 3, pp. 985–1032, Jun. 2008.

[3] S. Huang, L. Cai, and X. Zhang, "Parallel dedicated machine scheduling problem with sequence-dependent setups and a single server," *Comput. Ind. Eng.*, vol. 58, no. 1, pp. 165–174, Feb. 2010.

[4] C. M. Joo and B. S. Kim, "Parallel machine scheduling problem with ready times, due times and sequence-dependent setup times using meta-heuristic algorithms," *Eng. Optim.*, vol. 44, no. 9, pp. 1021–1034, Sep. 2012.

[5] J. Behnamian, M. Zandieh, and S. M. T. Fatemi Ghomi, "Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm," *Expert Syst. Appl.*, vol. 36, no. 6, pp. 9637–9644, Aug. 2009.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W.H. Freeman, 1979.

[7] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM J. Comput.*, vol. 7, no. 1, pp. 1–17, Feb. 1978.

[8] T. C. E. Cheng and C. C. S. Sin, "A state-of-the-art review of parallel-machine scheduling research," *Eur. J. Oper. Res.*, vol. 47, no. 3, pp. 271–292, Aug. 1990.

[9] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.*, vol. 17, no. 2, pp. 416–429, Mar. 1969.

[10] J. N. D. Gupta and J. C. Ho, "Minimizing makespan subject to minimum flowtime on two identical parallel machines," *Comput. Oper. Res.*, vol. 28, no. 7, pp. 705–717, Jun. 2001.

[11] M. Liu and C. Wu, "Scheduling algorithm based on evolutionary computing in identical parallel machine production line," *Robot. Comput.-Integr. Manuf.*, vol. 19, no. 5, pp. 401–407, Oct. 2003.

[12] P. Damodaran and M. C. Vélez-Gallego, "A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times," *Expert Syst. Appl.*, vol. 39, no. 1, pp. 1451–1458, Jan. 2012.

[13] J. Xu, S.-C. Liu, C. Zhao, J. Wu, W.-C. Lin, and P.-W. Yu, "An iterated local search and tabu search for two-parallel machine scheduling problem to minimize the maximum total completion time," *J. Inf. Optim. Sci.*, vol. 40, no. 3, pp. 751–766, Apr. 2019.

[14] L. Ghalami and D. Grosu, "Scheduling parallel identical machines to minimize makespan:A parallel approximation algorithm," *J. Parallel Distrib. Comput.*, vol. 133, pp. 221–231, Nov. 2019.

[15] E. Akyol Ozer and T. Sarac, "MIP models and a matheuristic algorithm for an identical parallel machine scheduling problem under multiple copies of shared resources constraints," *TOP*, vol. 27, no. 1, pp. 94–124, Apr. 2019.

[16] S. Tanaka and M. Araki, "A branch-and-bound algorithm with lagrangian relaxation to minimize total tardiness on identical parallel machines," *Int. J. Prod. Econ.*, vol. 113, no. 1, pp. 446–458, May 2008.

[17] M. Ranjbar, M. Khalilzadeh, F. Kianfar, and K. Etminani, "An optimal procedure for minimizing total weighted resource tardiness penalty costs in the resource-constrained project scheduling problem," *Comput. Ind. Eng.*, vol. 62, no. 1, pp. 264–270, Feb. 2012.

[18] A. Mensendiek, J. N. D. Gupta, and J. Herrmann, "Scheduling identical parallel machines with fixed delivery dates to minimize total tardiness," *Eur. J. Oper. Res.*, vol. 243, no. 2, pp. 514–522, Jun. 2015.

[19] J. E. Schaller, "Minimizing total tardiness for scheduling identical parallel machines with family setups," *Comput. Ind. Eng.*, vol. 72, pp. 274–281, Jun. 2014.

[20] W.-C. Yeh, P.-J. Lai, W.-C. Lee, and M.-C. Chuang, "Parallel-machine scheduling to minimize makespan with fuzzy processing times and learning effects," *Inf. Sci.*, vol. 269, pp. 142–158, Jun. 2014.

[21] S.-I. Kim, H.-S. Choi, and D.-H. Lee, "Scheduling algorithms for parallel machines with sequence-dependent set-up and distinct ready times: Minimizing total tardiness," *Proc. Inst. Mech. Engineers, B, J. Eng. Manuf.*, vol. 221, no. 6, pp. 1087–1096, Jun. 2007.

[22] C. O. Kim and H. J. Shin, "Scheduling jobs on parallel machines: A restricted tabu search approach," *Int. J. Adv. Manuf. Technol.*, vol. 22, nos. 3–4, pp. 278–287, Sep. 2003.

[23] Ü. Bilge, F. Kraç, M. Kurtulan, and P. Pekgán, "A tabu search algorithm for parallel machine total tardiness problem," *Comput. Oper. Res.*, vol. 31, no. 3, pp. 397–414, Mar. 2004.

[24] R. Tavakkoli-Moghaddam, F. Taheri, M. Bazzazi, M. Izadi, and F. Sassani, "Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints," *Comput. Oper. Res.*, vol. 36, no. 12, pp. 3224–3230, Dec. 2009.

[25] I. A. Chaudhry and P. R. Drake, "Minimizing total tardiness for the machine scheduling and worker assignment problems in identical parallel machines using genetic algorithms," *Int. J. Adv. Manuf. Technol.*, vol. 42, nos. 5–6, pp. 581–594, May 2009.

[26] D. Xu and D.-L. Yang, "Makespan minimization for two parallel machines scheduling with a periodic availability constraint: Mathematical programming model, average-case analysis, and anomalies," *Appl. Math. Model.*, vol. 37, nos. 14–15, pp. 7561–7567, Aug. 2013.

[27] S. Rajakumar, V. P. Arunachalam, and V. Selladurai, "Workflow balancing in parallel machines through genetic algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 33, nos. 11–12, pp. 1212–1221, Aug. 2007.

[28] J. C. Chen, C.-C. Wu, C.-W. Chen, and K.-H. Chen, "Flexible job shop scheduling with parallel machines using genetic algorithm and grouping genetic algorithm," *Expert Syst. Appl.*, vol. 39, no. 11, pp. 10016–10021, Sep. 2012.

[29] S. Balin, "Parallel machine scheduling with fuzzy processing times using a robust genetic algorithm and simulation," *Inf. Sci.*, vol. 181, no. 17, pp. 3551–3569, Sep. 2011.

[30] Y. Huo, "Parallel machine makespan minimization subject to machine availability and total completion time constraints," *J. Scheduling*, vol. 22, no. 4, pp. 433–447, Aug. 2019.

[31] W. Ma, Y. Liu, and X. Zhang, "A new model and algorithm for uncertain random parallel machine scheduling problem," *Soft Comput.*, vol. 23, no. 15, pp. 6555–6566, Aug. 2019.

[32] M.-Y. Kim and Y. H. Lee, "MIP models and hybrid algorithm for minimizing the makespan of parallel machines scheduling problem with a single server," *Comput. Oper. Res.*, vol. 39, no. 11, pp. 2457–2468, Nov. 2012.

[33] C. A. Glass, Y. M. Shafransky, and V. A. Strusevich, "Scheduling for parallel dedicated machines with a single server," *Nav. Res. Logistics*, vol. 47, no. 4, pp. 304–328, Jun. 2000.

[34] S. A. Kravchenko and F. Werner, "Parallel machine scheduling problems with a single server," *Math. Comput. Model.*, vol. 26, no. 12, pp. 1–11, Dec. 1997.

[35] A. H. Abdekhodaee and A. Wirth, "Scheduling parallel machines with a single server: Some solvable cases and heuristics," *Comput. Oper. Res.*, vol. 29, no. 3, pp. 295–315, Mar. 2002.

[36] A. H. Abdekhodaee, A. Wirth, and H.-S. Gan, "Scheduling two parallel machines with a single server: The general case," *Comput. Oper. Res.*, vol. 33, no. 4, pp. 994–1009, Apr. 2006.

[37] H.-S. Gan, A. Wirth, and A. Abdekhodaee, "A branch-and-price algorithm for the general case of scheduling parallel machines with a single server," *Comput. Oper. Res.*, vol. 39, no. 9, pp. 2242–2247, Sep. 2012.

[38] K. Hasani, S. A. Kravchenko, and F. Werner, "Block models for scheduling jobs on two parallel machines with a single server," *Comput. Oper. Res.*, vol. 41, pp. 94–97, Jan. 2014.

[39] K. Hasani, S. A. Kravchenko, and F. Werner, "Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server," *Int. J. Prod. Res.*, vol. 52, no. 13, pp. 3778–3792, Jul. 2014.

[40] K. Hasani, S. A. Kravchenko, and F. Werner, "Minimizing the makespan for the two-machine scheduling problem with a single server: Two algorithms for very large instances," *Eng. Optim.*, vol. 48, no. 1, pp. 173–183, Jan. 2016.

[41] J.-P. Arnaout, "Heuristics for the two-machine scheduling problem with a single server," *Int. Trans. Oper. Res.*, vol. 24, no. 6, pp. 1347–1355, Nov. 2017.

[42] I. Alharkan, M. Saleh, M. A. Ghaleb, H. Kaid, A. Farhan, and A. Almarfadi, "Tabu search and particle swarm optimization algorithms for two identical parallel machines scheduling problem with a single server," *J. King Saud Univ.-Eng. Sci.*, vol. 35, pp. 330–338, Mar. 2019.

[43] M. Gholami, M. Zandieh, and A. Alem-Tabriz, "Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns," *Int. J. Adv. Manuf. Technol.*, vol. 42, nos. 1–2, pp. 189–201, May 2009.

[44] A. Allahverdi, "Dual criteria scheduling on a two-machine flowshop subject to random breakdowns," *Int. Trans. Oper. Res.*, vol. 5, no. 4, pp. 317–324, Jul. 1998.

[45] J. Sun and D. Xue, "A dynamic reactive scheduling mechanism for responding to changes of production orders and manufacturing resources," *Comput. Ind.*, vol. 46, no. 2, pp. 189–207, Sep. 2001.

[46] J. Kaabi, C. Varnier, and N. Zerhouni, "Genetic algorithm for scheduling production and maintenance in a flow-shop," (in French), Lab. Autom. Besancon, Paris, France, 2003.

[47] J.-Y. Lee and Y.-D. Kim, "Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance," *Comput. Oper. Res.*, vol. 39, no. 9, pp. 2196–2205, Sep. 2012.

[48] R. Mellouli, C. Sadfi, C. Chu, and I. Kacem, "Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times," *Eur. J. Oper. Res.*, vol. 197, no. 3, pp. 1150–1165, Sep. 2009.

[49] C.-J. Liao, D.-L. Shyur, and C.-H. Lin, "Makespan minimization for two parallel machines with an availability constraint," *Eur. J. Oper. Res.*, vol. 160, no. 2, pp. 445–456, Jan. 2005.

[50] H. R. D. Saidy and M. T. Taghavi-Fard, "Study of scheduling problems with machine availability constraint," *Ind. Syst. Eng.*, vol. 1, no. 4, pp. 360–368, Jan. 2008.

[51] A. Berrichi, L. Amodeo, F. Yalaoui, E. Châtelet, and M. Mezghiche, "Bi-objective optimization algorithms for joint production and maintenance scheduling: Application to the parallel machine problem," *J. Intell. Manuf.*, vol. 20, no. 4, pp. 389–400, Aug. 2009.

[52] A. Berrichi, F. Yalaoui, L. Amodeo, and M. Mezghiche, "Bi-objective ant colony optimization approach to optimize production and maintenance scheduling," *Comput. Oper. Res.*, vol. 37, no. 9, pp. 1584–1596, Sep. 2010.
[53] A. Berrichi and F. Yalaoui, "Efficient bi-objective ant colony approach to minimize total tardiness and system unavailability for a parallel machine scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 68, nos. 9–12, pp. 2295–2310, Oct. 2013.
[54] J. Yoo and I. S. Lee, "Parallel machine scheduling with maintenance activities," *Comput. Ind. Eng.*, vol. 101, pp. 361–371, Nov. 2016.
[55] J. Shen and Y. Zhu, "A parallel-machine scheduling problem with periodic maintenance under uncertainty," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 8, pp. 3171–3179, Aug. 2019.
[56] J. Kaabi and Y. Harrath, "Scheduling on uniform parallel machines with periodic unavailability constraints," *Int. J. Prod. Res.*, vol. 57, no. 1, pp. 216–227, Jan. 2019.
[57] N. G. Hall, C. N. Potts, and C. Sriskandarajah, "Parallel machine scheduling with a common server," *Discrete Appl. Math.*, vol. 102, no. 3, pp. 223–243, Jun. 2000.
[58] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[59] M. L. Pinedo, "Complexity theory," in *Scheduling*. Boston, MA, USA: Springer, 2012, pp. 589–602.
[60] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, Jun. 1953.
[61] E. G. Talbi, *Metaheuristics: From Design to Implementation*. Hoboken, NJ, USA: Wiley, May 2009.
[62] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, Aug. 1989.
[63] K. Nonobe and T. Ibaraki, "A tabu search approach to the constraint satisfaction problem as a general problem solver," *Eur. J. Oper. Res.*, vol. 106, nos. 2–3, pp. 599–623, Apr. 1998.
[64] N. Kundakcá and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," *Comput. Ind. Eng.*, vol. 96, pp. 31–51, Jun. 2016.
[65] D. Golberg, *Algorithms in Search, Optimization and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.
[66] O. Etiler, B. Toklu, M. Atak, and J. Wilson, "A genetic algorithm for flow shop scheduling problems," *J. Oper. Res. Soc.*, vol. 55, no. 8, pp. 830–835, 2004.
[67] L. R. Abreu, J. O. Cunha, B. A. Prata, and J. M. Framinan, "A genetic algorithm for scheduling open shops with sequence-dependent setup times," *Comput. Oper. Res.*, vol. 113, Jan. 2020, Art. no. 104793.
[68] D. Lei, Y. Yuan, and J. Cai, "An improved artificial bee colony for multi-objective distributed unrelated parallel machine scheduling," *Int. J. Prod. Res.*, vol. 13, pp. 1–13, Jun. 2020.

LOTFI HIDRI received the B.S. degree in mathematics from the Tunisian College of Science, in 1993, the M.S. degree in energetic engineering from the National Engineering School, in 1999, and the Ph.D. degree in operations research from the Tunisian High Institute of Management, in 2007. Since 2012, he has been a Faculty Member with the Industrial Engineering Department, King Saud University. His main research interests include scheduling and transportation.

KHALED ALQAHTANI received the B.Sc. degree in industrial engineering from King Khaled University, Abha, in 2013, and the M.S. degree in industrial engineering from King Saud University, Saudi Arabia, in 2020, where he is currently pursuing the Ph.D. degree with the Industrial Engineering Department, College of Engineering. His area of expertise is scheduling.

ACHRAF GAZDAR received the Ph.D. degree in computer science from Manouba University, Tunisia, in 2007. He is currently a Faculty Member with the Software Engineering Department, College of Computer and Information Systems, King Saud University. His research interests include the networking, video on demand systems, streaming architectures, peer to peer, and recommender systems.

AHMED BADWELAN received the B.Sc. degree in mechanical engineering from the Faculty of Engineering, University of Aden, in 2012, and the M.S. degree in industrial engineering from King Saud University, Saudi Arabia, where he is currently pursuing the Ph.D. degree with the Industrial Engineering Department, College of Engineering.

He is a Researcher with the Industrial Engineering Department, College of Engineering, King Saud University. His area of expertise is manufacturing systems. His main research interests include manufacturing, production, and quality.