Metaheuristics for optimizing unrelated parallel machines scheduling with unreliable resources to minimize makespan

Husam Kaid¹,², Abdulrahman Al-Ahmari¹,², Adel Al-Shaye¹, Emad Abouel Nasr¹,³, Ali K Kamrani⁴ and Haitham A Mahmoud¹,³

Abstract
Parallel machines scheduling problems with continuous availability of machines are NP-hardness (non-deterministic polynomial-time hardness) and have become very popular for the last decade; there is still very limited literature on this problem. The purpose of this paper is to focus on the problem of scheduling n independent jobs to be processed on m unrelated identical parallel machines with availability constraints to minimize the maximum completion time of jobs (makespan). For this NP-hard problem, a mixed-integer linear programming (MILP) model is proposed to find an optimal solution for this problem. Two metaheuristics, tabu search (TS) and simulated annealing (SA) are proposed to solve large scale problem. Moreover, the performance of the solution obtained by the proposed metaheuristics is evaluated based on a lower bound, which decreases the time required to find the optimal solution. Extensive experiments are carried out to assess the performance of all proposed metaheuristics. The computational results highlight the ability of the proposed metaheuristics to obtain optimal solutions for most of the instances compared with the solutions of the proposed MILP model and lower bounds. Moreover, SA and TS can provide good efficiency for the problem in any jobs size and any machine size, but TS provides worse CPU time as the size of jobs become large.

Keywords
Tabu search, simulated annealing, scheduling, parallel machines, makespan

Date received: 13 October 2021; accepted: 5 April 2022

Handling Editor: Chenhui Liang

Introduction
Parallel machine scheduling has been an interesting topic of research due to the breadth of its potential applications,¹–⁶ which include scheduled drilling mechanisms in printed circuit board manufacturing facilities, textile industry activities, dicing of semiconductor wafers used in computer manufacturing, and ship docking systems.⁷ This popularity has grown significantly in recent years as a result of the development of parallel processor computing systems.⁸ A parallel bank of machines is a significant configuration from both a theoretical and practical perspective. It is an
extension of the single machine and a subset of the flexible flow shop from a theoretical viewpoint. It is significant from a practical perspective since the presence of resources in parallel is frequent in the real world. Additionally, parallel machine approaches are frequently employed in decomposition procedures for sequential systems.

In this article, we considered the problem of scheduling unrelated identical parallel machines with availability constraints. In this problem, there are \( n \) independent jobs and \( m \) unrelated identical parallel machines. Job \( j \) \((j = 1, 2, \ldots, n)\) becomes available for processing at time zero and requires a processing time \( p_{ij} \) if job \( j \) is assigned to exactly one machine \( i \) \((i = 1, 2, \ldots, m)\). Preventive maintenance actions are often carried out to keep the manufacturing system at the desired level of operation by preventing failures before they happen. There is only one maintenance period that occurred for each machine \( i \) \([t_{Bi}, t_{El}]\). The main objective is to determine the set of job sequences that processes all jobs and minimize makespan, \( C_{max} \), which is the completion time of the latest finished job. The Graham scheduling notation \( \alpha | \beta | \gamma \) for the problem is represented by \( R_m | \text{availability} | C_{max} \).

Several studies,\(^{5,10–18}\) have been proposed to solve parallel machine scheduling using different metaheuristic algorithms to minimize different objective functions such as makespan, total tardiness, the weighted sum of completion times, and the number of tardy jobs. Computational studies were carried out to determine the heuristics’ performance and execution time. It observed that TS and SA outperform other metaheuristic algorithms, and as a result, TS and SA are recommended for this problem. Fanjul-Peyro and Ruiz\(^{19}\) proposed a set of simple iterated greedy local search-based metaheuristics for unrelated parallel-machine scheduling for makespan minimization, which produces solutions of remarkable quality in a very short amount of time. Vallada and Ruiz\(^{20}\) applied a genetic algorithm (GA) to the scheduling of unrelated parallel machines subject to machine and job sequence setup times. The proposed GA consists of a simple local search and a crossover operator improved with local search; it showed excellent performance as a model for large- and small-scale experiments. Fanjul-Peyro and Ruiz\(^{21}\) presented mathematical formulations for mixed-integer programming based on two generalizations of non-identical parallel machine scheduling for makespan minimization. The first is a condition under which not all parallel machines should be employed but a subset. The second is that there is no need for managing all available jobs. Extensive testing demonstrates that the two suggested methods significantly improve the results. Balin\(^{22}\) developed a GA and fuzzy processing time simulation technique for scheduling a set of unrelated parallel machines. The obtained results were compared to those obtained when the LPT rule was used.

The results of the proposed method were excellent and easy to achieve. There are two metaheuristics proposed by Joo and Kim\(^{23}\): the self-evolution algorithm and a GA for assessing job allocation and non-identical parallel-machine scheduling for the aim of minimizing the makespan. Several randomly generated instances were used to compare the meta-heuristic algorithms’ efficiencies with optimal solutions. Fleszar et al.\(^{24}\) hybridized a variable neighborhood descent search algorithm with mathematical programming for scheduling jobs on unrelated parallel machines in order to minimize the makespan. Joo and Kim\(^{25}\) developed a hybrid GA with dispatching rules for scheduling a series of jobs on non-identical parallel machines with sequence- and machine-dependent setup times and machine-dependent processing times with the aim of minimizing the makespan. Geyik and Elia\(^{26}\) developed a fuzzy linguistic technique for solving non-identical parallel-machine scheduling considering the learning effect and uncertain processing times.

It is well known that parallel machines scheduling problems with continuous availability of machines are NP-hard.\(^{27}\) With availability constraints, unrelated parallel machine scheduling problems are NP-hard for the makespan objective.\(^{28–30}\) Although scheduling problems with availability constraints have become very popular for the last decade, there is still very limited literature on this problem. Therefore, this paper aims to propose a mathematical model to optimize the unrelated parallel machines scheduling problem under availability constraint to minimize makespan. To do this, a new mixed-integer linear programming (MILP) model is developed to find an optimal solution for this problem. In the second step, two metaheuristics, tabu search and simulated annealing are presented to solve the MILP model in the case of larger-scale problems and the obtained results compared with a proposed lower bound to assess the effectiveness of the proposed metaheuristics solutions.

The main contributions of the present research are as follows.

1. A new mixed-integer linear programming (MILP) model is developed for unrelated parallel machine scheduling problems to minimize the makespan with availability constraints. It can handle all unreliable machines in the system.
2. Preventive maintenance actions are developed for unreliable machines.
3. Efficient metaheuristic algorithms are suggested to solve the scheduling problem.
4. A lower bound is developed to evaluate the solution obtained by the proposed metaheuristics.

The rest of the paper is organized as follows. Section 2 the problem and presents the formulation of the developed mixed-integer linear programming model, the synthesis of lower bound, and numerical example. Sections 3, 4, and 5 display the suggested metaheuristics, computational analysis and results, and sensitivity analysis, respectively. Finally, Section 6 presents conclusions and future work.

Problem formulation

Problem definitions

The considered problem is depicted as follows. There are \( n \) jobs, which are processed on \( m \) unrelated parallel machines. The scheduling problem has following assumptions on jobs and machines.

1. At most one machine can process just one job at a time;
2. At time zero, all jobs are available for processing and all machines are available;
3. No jobs can be processed on more than one machine at a time, and each job operation must be completed;
4. The processing times and PM windows are fixed and known in advance.

The main objective is to minimize the makespan for all jobs under availability constraint of machines. The Graham scheduling notation is represented by \( R_m | \text{availability} | C_{max} \). The problem \( R_m | \text{availability} | C_{max} \) is a classical unrelated parallel machine problem and NP-hard,\(^{14}\) where there is no availability constraint. Since our problem \( R_m | \text{availability} | C_{max} \) is more general than \( R_m | C_{max} \) it is obviously NP-hard.

Formulation of the mixed integer linear programming model

This section proposes a mathematical model (using some constraints from Al-Ahmari et al.\(^{31}\)) for unrelated parallel machine scheduling problem under availability constraint to minimize makespan. The mixed integer linear programming model is stated as follows:

\[
\text{Min } z = C_{max} \tag{1}
\]

\[
\text{Subject to:}
\]

\[
\sum_{i=1}^{m} X_{ji} = 1, \forall j = 1, \ldots, n. \tag{2}
\]

\[
\sum_{j=1}^{n+m} X_{ji} = 1, \forall i = 1, 2, \ldots, m. \tag{3}
\]

\[
\sum_{j=1}^{m} X_{ji} = 1, \forall j = n + 1, \ldots, n + m. \tag{4}
\]

\[
\sum_{j=1}^{m} Z_{ji} = 1, \forall j = n + 1, \ldots, n + m. \tag{5}
\]

\[
Y_{jki} + Y_{kji} = X_{ji} \forall j, k = 1, 2, \ldots, n; \forall i = 1, 2, \ldots, m; j \neq k. \tag{6}
\]

\[
Y_{jki} + Y_{kji} = X_{ji} + X_{ki} - 1 \forall j, k = 1, 2, \ldots, n; \forall i = 1, 2, \ldots, m; \tag{7}
\]

\[
c_{ji} \geq s_{ji} + p_{ji}X_{ji} \forall j = 1, 2, \ldots, n; \forall i = 1, 2, \ldots, m. \tag{8}
\]

\[
st_{ji} + p_{ji}X_{ji} \leq s_{ki} + Q(1 - Y_{jki}) \forall j, k = 1, 2, \ldots, n + m; \forall i = 1, 2, \ldots, m; j \neq k. \tag{9}
\]

\[
st_{ji} + p_{ji}X_{ji} - Z_{ji} > 0, \forall i = 1, 2, \ldots, n; \forall j = n + 1, \ldots, n + m. \tag{10}
\]

\[
st_{ji} - Q(1 - Z_{ji}) \leq t_{ki}, \forall i = 1, 2, \ldots, m; \forall j = n + 1, \ldots, n + m. \tag{11}
\]

\[
C_{max} \geq c_{ji}, \forall j = 1, 2, \ldots, n; \forall i = 1, 2, \ldots, m. \tag{12}
\]

\[
st_{ji} \geq 0 \forall j = 1, \ldots, n + m; \forall i = 1, 2, \ldots, m. \tag{13}
\]

\[
X_{ji} = 0 \text{ or } 1 \forall i = 1, 2, \ldots, m; \forall j = 1, \ldots, n + m. \tag{14}
\]

\[
Y_{jki} = 0 \text{ or } 1 \forall i = 1, 2, \ldots, m; \forall j, k = n + 1, \ldots, n + m; j \neq k. \tag{15}
\]

\[
Z_{ji} = 0 \text{ or } 1 \forall i = 1, 2, \ldots, m; \forall j, k = 1, 2, \ldots, n + m; j \neq k. \tag{16}
\]

Objective (1) minimizes the maximum completion time of the solution. Constraint (2) guarantees that each original job \( j \) in \( n \) is performed on exactly one machine. Constraints (3, 4, and 5) guarantee that periods of unavailability are completed and only one unavailability period is allocated to each machine \( i \). Constraints (6 and 7) confirm that assigned jobs must appear once in their sequence on the same machine. The job \( j \) completion time is determined by constraint (8). Constraints (9 and 10) confirm the correct order of the jobs and the completion time of job \( k \) should be greater or equal
than the job completion time. Constraints (11 and 12) confirm that the starting time and completion time of the dummy jobs have to respect the preventive maintenance period \([t_{Bi}, t_{Ei}]\). Finally, constraints (13–17) define the non-negativity and integrality of the variables.

### Lower Bound

In this section, we propose the LBs – LB1 and LB2, which we use to assess the performance of the metaheuristic method used to find the best makespan. Let \(p_j\) is the minimum processing time for each job such that

\[
p_j = \min_{1 \leq i \leq m} p_{ji}, \text{ for } j = 1, \ldots, n
\]  

The valid first lower bound \(LB_1\) on \(C_{max}\) is expressed as

\[
LB_1 = \frac{1}{m} \left[ \sum_{j=1}^{n} p_j + \sum_{i=1}^{m} (t_{Ei} - t_{Bi}) \right]  
\]  

Since each job must be scheduled, a second lower bound \(LB_2\) is expressed as

\[
LB_2 = \max_{1 \leq i \leq m, \forall j} p_{ji}, \text{ for } j = 1, \ldots, n + m
\]  

Therefore, a valid lower bound \(LB\) on \(C_{max}\) is expressed as

\[
LB = \max(LB_1, LB_2)
\]

### Numerical Example

Consider an unrelated parallel machine scheduling problem that consists of three machines \(i = 1, 2, 3\) operating nine original jobs \(j = 1, 2, \ldots, 9\) and three dummy jobs \(j = 10, 11, 12\) denoting to preventive maintenance of machines. The job processing times on machines \(p_{ji}\) and the periods of preventive maintenance \((t_{Bi}, t_{Ei})\) are presented in Tables 1 and 2, respectively. To solve the example presented in Figure 5 by using the developed MILP model, we used Lingo optimizer. Lingo optimizer is a mathematical programming solver for continuous and integer variables that resolve linear and quadratic problems. The optimizer offers a good branch and bound framework. Table 3 presents the MILP model optimal solution and decision variables for the problem. Figure 1 shows the Gantt chart of the optimal sequence for jobs.

### Metaheuristic Algorithms

**Tabu Search**

Tabu search is a popular metaheuristic search approach using local search techniques for optimization problems Glover. Tabu search is carried out using a neighborhood or local search mechanism to iteratively move from one feasible solution A to an improved solution B in the neighborhood of A until a stop condition is satisfied. In order to prevent solutions previously visited, tabu search uses memory named tabu list to store
results about the search process. The tabu list is updated in each TS iteration. The risk of rejecting solutions, which have not yet been created may occur due to the restrictions in the tabu list. Tabu results are then tested for such parameters, which are known as aspiration criteria. If the tabu value is better than the best objective value found earlier, the tabu will accept the solution and delete its move from the tabu list. TS has many factors or parameters, which must be tuned before its searching starts. Initial solution, the next element, the tabu list, the aspiration condition, and the stopping condition are main stopping of the TS. In this study, the initial solution for TS has been set by the dispatching rule longest processing time (LPT) policy, which sequences tasks or jobs in a descending order of processing time. In addition, to generate new candidate solutions for TS algorithm, we used the pairwise interchange operator. According to preliminary experiments, the number of candidates generated was 50. The size of the tabu list is set to 10 elements and the first-in-first-out (FIFO) technique is used to update the tabu list based on the same preliminary experiments. As previously reported, the aspiration criteria were applied (when a move leads to a better solution than the previously found best solution value). Finally, the stopping criteria is then defined such that computational times (in s) are similar in each algorithm or if the lower bound is reached. The general TS algorithm is constructed in Algorithm 1:

**Simulated annealing**

Simulated annealing is a popular metaheuristic and has been applied in several combinatorial problems. Algorithm 2 includes the implemented SA algorithm for estimating the makespan. Simulated annealing improves several initial solutions to achieve the stopping criterion and the output is the best global solution found. For each individual solution, the following procedures are stated: (1) obtain the initial solution using LPT rule; (2) set the initial temperature \( t_0 \); (3) improve the solution based on an exploration criteria of the problem; (4) assess the new solution according to a fitness function; (5) the new solution becomes the current solution when the metropolitan condition accepts the new solution which accepts improvements and worse solutions can be accepted on probabilistic depending on the degradation of the solution and the current temperature; (6) reduce the temperature by a geometric cooling scheme, where the typically high cooling rate of \( \zeta \) (i.e. \( \zeta \approx 0.9 \)); (7) repeat until the stopping condition has been achieved. The general SA problem is constructed in Algorithm 2:

| Machine | Scheduled job | Max \( (C_j) \) |
|---------|---------------|-----------------|
| 1       | J9, PM(J11), j2 | 31              |
| 2       | J4, PM(J10), j6 | 29              |
| 3       | J8, J11, PM(J12), J5, J7, J3 | 31 |

\[ X_{2,1} = 1, X_{9,1} = 1, X_{11,1} = 1 \]
\[ X_{4,2} = 1, X_{6,2} = 1, X_{10,2} = 1 \]
\[ X_{13} = 1, X_{12,3} = 1, X_{8,3} = 1, X_{18,3} = 1, X_{12,3} = 1 \]
\[ Z_{10,2} = 1, Z_{11,1} = 1, Z_{12,3} = 1 \]

**Algorithm 1:** Construction of TS algorithm.

**Input:** \( n, m, p_j, t_B, t_E \).

**Initialization:** Let the best solution is to be \( s_0 \), the initial solution is to be \( s \), \( s_0 = s \), set the tabu list, medium-term and long-term memories.

**Step 1:** while \( s_0 > L_B \) or No. of iterations < maximum iterations do

1. Create a set \( \Psi \) of solutions;
2. Find best neighbor \( s' \) of \( \Psi \);
3. \( s = s' \);
4. Update aspiration conditions and tabu list;
5. If \( f(s) < f(s_0) \), Then \( s_0 = s \);
end while

**Output:** The best solution \( s_0 \).

**Computational Results**

The proposed metaheuristic algorithms and the computational experiments for all instances has been coded and implemented using MATLAB R2015a on the computer with Intel(R) Core (TM) i7-4702MQ CPU @ 2.20 GHz, 16 GB RAM.

**Data generation**

The metaheuristic algorithms proposed in this paper are conducted and evaluated computational experiments were implemented using randomly generated test 32 instances with fixed the number of jobs 10, 12, 15, 20, 50, 70, 100, 150 and number of machines 2, 3, 5, 7. The jobs processing times are integer values and generated randomly according to the uniform distribution as \( U(1, 100) \). The starting times of the preventive maintenance are drawn from the uniform distribution on \( U(\min p_j, \max p_j) \) where \( \min p_j \) and \( \max p_j \) are the smallest and largest processing times, respectively. The ending times of the preventive maintenance period \( t_E \) are computed from \( (t_B + l_i) \), where \( l_i \) is a length of preventive maintenance, denoted as \( l_i = (L + U)/2 \), where \( L \) and \( U \) are the lower and upper limits of the uniform distribution of job processing times, respectively.
Parameter tuning

In metaheuristics design, tuning parameters is an important issue. Each metaheuristics has its parameter values that affect its effectiveness. For this, experimental runs are conducted for parameters values of algorithms, and Table 4 shows the best setting of parameters for two algorithms.

Experimental results

There are two types of instances, which are small sized instances with \( n = \{10, 12, 15\} \), \( m = \{2, 3, 5, 7\} \) and large sized instances with \( n = \{20, 50, 70, 100\} \), \( m = \{2, 3, 5, 7\} \). Table 5 shows the experiments results of small sized instances. Table 5 display the optimal MILP model solution by Lingo and relative percentage deviation (RPD) of 15 replications computed with the equation (22) by metaheuristic algorithms for each test instance, where “Average RPD,” “Min RPD,” and Max RPD are the average, minimum, maximum RPD of the metaheuristic algorithm solution from the optimal solutions.

\[
\text{RPD} = \frac{\text{Method}_{\text{sol}} - \text{Opt}_{\text{sol}}}{\text{Opt}_{\text{sol}}} \times 100 \tag{22}
\]

where \( \text{Method}_{\text{sol}} \) is the makespan obtained by the proposed metaheuristics and \( \text{Opt}_{\text{sol}} \) is the obtained optimal solution by Lingo solver.

In order to validate lower bounds shown in equations (21) and (22), it is important to investigate if the observed differences in the optimal solution and the lower bound values of small sized instances that computed by MILP presented in Figure 2 are statistically significant. The significant differences can be computed using two-sample \( t \)-test. Table 6 displays the mean, standard deviation, and \( p \)-value of \( t \)-test at the 95% confidence level. The \( t \)-test indicates that statistically significant differences are not occurred with \( p \)-value = 0.781 between the optimal solution and lower bound

---

**Table 4.** The best setting for SA and TS parameters.

| Algorithm | Parameter   | Value   |
|-----------|-------------|---------|
| SA        | Tabu list size | 10      |
| TS        | Candidate list size | 50      |
| TS        | Stopping criteria | LB or no. of iterations (200n) |
| TS        | Initial temperature | 20      |
| TS        | Cooling rate | 0.9     |
| TS        | Stopping criteria | LB or no. of iterations (200n) |

---

**Algorithm 2:** Construction of SA algorithm.

**Input:** \( n, m, p_{ji}, t_{Bi,t}, E_{i} \).

**Initialization:** Let the best solution is to be \( s_0 \), the initial solution is to be \( s \), \( s_0 = s \), the initial temperature is to be \( t_0 > 0 \), and the temperature change function is to be \( \xi(T) \).

**Step 1:** while \( s_0 \leq \text{LB} \) or No. of iterations \(<\) maximum iterations do

1. Randomly choses \( s \in \Psi(s_0) \);
2. \( \lambda = f(s) - f(s_0) \);
3. if \( \lambda < 0 \) then \( s_0 = s \)
4. else generate random \( x \) uniformly in the range \([0, 1]\);
5. if \( x < \exp\left(-\frac{1}{T_{\text{em}}}ight) \), Then \( s = s_0 \)
6. \( T_{\text{em}} = \xi(T_{\text{em}}) \);

**Output:** The best solution \( s_0 \).

---

**Table 5.** Comparisons metaheuristics for small sized instances.

| No. | Problem size \( n \times m \) | SA RPD% Ave. Min. Max. | CPU time (s) Ave. Min. Max. | TS RPD% Ave. Min. Max. | CPU time (s) Ave. Min. Max. |
|-----|-----------------------------|------------------------|-----------------------------|------------------------|-----------------------------|
| 1   | 10 \times 2                | 5.617 0 11.165 1.842 0.040 2.152 | 1.872 0 3.398 0.799 0.072 | 1.045 0.072 0.934 |
| 2   | 10 \times 3                | 3.016 0 15.556 1.889 0.341 2.150 | 0.476 0 1.111 0.523 0.072 | 0.915 0.072 0.915 |
| 3   | 10 \times 5                | 9.643 0 17.500 1.885 0.322 2.156 | 2.143 0 5 0.512 0.067 | 0.920 0.067 0.920 |
| 4   | 10 \times 7                | 6.642 0 13.158 1.508 0.465 2.169 | 0 0 0 0.263 0.098 | 0.755 0.098 0.755 |
| 5   | 12 \times 2                | 14.610 1.364 21.818 0.937 0.024 2.139 | 7.857 5.455 10.455 0.056 | 0.046 0.063 | 0.063 |
| 6   | 12 \times 3                | 8.844 0 16.190 0.135 0.053 0.241 | 7.687 1.429 11.905 0.074 | 0.063 0.084 | 0.084 |
| 7   | 12 \times 5                | 10.647 6.604 13.208 1.902 0.281 2.189 | 7.547 2.830 10.377 0.467 | 0.213 0.918 | 0.918 |
| 8   | 12 \times 7                | 0.000 0 2.173 2.155 2.180 | 0.0 0 0 0.954 0.939 | 0.965 0.939 0.965 | 0.939 |
| 9   | 15 \times 2                | 2.421 0 4.237 1.389 0.315 2.143 | 1.332 0 4.237 0.610 0.131 | 0.935 0.131 0.935 |
| 10  | 15 \times 3                | 5.149 0 11.168 1.445 0.367 2.164 | 1.523 0 5.584 0.449 0.217 | 0.920 0.217 0.920 |
| 11  | 15 \times 5                | 14.694 7.619 31.429 2.161 2.155 2.165 | 8.027 5.714 8.571 0.923 | 0.921 0.925 | 0.925 |
| 12  | 15 \times 7                | 10.467 3.960 15.842 2.183 2.177 2.191 | 10.750 3.960 12.871 0.903 0.797 | 0.922 0.797 0.922 |
Therefore, the developed lower bound is valid and can be applied to different sized instances.

Table 7 shows the experiments results of large sized instances, the lower bound, and relative percentage deviation (RPD) of 15 replications computed with the equation (23) by metaheuristic algorithms for each test instance.

$$\text{RPD} = \frac{(\text{Method}_{\text{opt}} - \text{LB})}{\text{LB}} \times 100 \quad (23)$$

Where $\text{Method}_{\text{opt}}$ is the makespan obtained by metaheuristic and LB obtained by equation (21).

To evaluate the proposed algorithms, it is important to investigate if the observed differences in the makespan and CPU time values for proposed algorithms with small and large sized instances are statistically significant. The significant differences also can be computed using two sample t-test. Table 8 displays the mean, standard deviation, and $p$-value of t-test at the 95% confidence level. The t-test indicates that statistically significant differences are not occurred in makespan with $p$-value = 0.057 and 0.205 for proposed algorithms with small and large sized instances, respectively. Meanwhile statistically significant differences are not occurred in CPU time values with $p$-value $<0.0000$ and $<0.0000$ for proposed algorithms with small and large sized instances. Thus, SA and TS give good performance on both jobs size per machine and any machine size. However, TS provides worse CPU time as jobs size per machine and machine size become large as shown in Figures 3 to 8.

Figures 3 to 6 display the Box Plot of the overall average RPD for $C_{\text{max}}$ of small sized instances, CPU times of small sized problem, $C_{\text{max}}$ of large sized problem, and CPU times of large sized problem, respectively. We can clearly see from Figures 3 and 4 that in respect of $C_{\text{max}}$ and CPU time, TS has a better performance than the SA in small sized problem. Meanwhile in large sized problem, SA has a better performance than the TS in respect of $C_{\text{max}}$ and CPU time as shown in Figures 5 and 6. Figures 7 and 8 also present the Box Plot of the average RPD values for $C_{\text{max}}$ of large sized instances versus the number of machines ($m$) and number of jobs ($n$) per machines, respectively. It can be noticed that the $C_{\text{max}}$ of the proposed algorithms increases significantly, as the jobs number per machine and the machines number increases, and SA provides better performance in any machine size and any jobs size than TS.

**Sensitivity analysis**

A two-factor analysis of variance is used to study the effect of the number of jobs and the number of machines on the average relative percentage deviation RPD of the algorithms. Average relative percentage deviations were obtained for the eight different jobs sets (10, 12, 15, 20, 50, 70, 100, 150) and four different machines sets (2, 3, 5, 7), each with five replications. Table 9 summarizes these results. As the table indicates, the $p$-values of $n$, $m$, and their interaction $nm$ are less than 0.05, therefore, the factors $n$, $m$, and $nm$ have the greatest effect on RPD. This fact is illustrated in Figures 9 and 10. As the diagrams demonstrate, increasing the number of jobs and machines increases the RPD. The reason is that by increasing the size of sets $n$ and $m$, the probability of discovering feasible interchanges increases.

**Conclusions**

This paper proposes an optimal solution to unrelated parallel machine scheduling problems subject to preventive maintenance to minimize the maximum completion time of jobs. The first step is to formulate the problem using a mixed-integer programming model in order to find the optimal solution. The problem is NP-hard, which means that it cannot be solved in polynomial time with jobs $n > 15$. In many cases, the time
required to solve this type of problem increases exponentially as \( n \) increases. To improve solution efficiency, we proposed two metaheuristics: tabu search and simulated annealing. The parameters of the proposed two algorithms were defined in such a way that they were compatible with the problem structure, and then the optimal set of parameters for the two algorithms was identified using experimental design. A lower bound is proposed to assess the accuracy of the obtained solution by the proposed metaheuristics to reduce the time required for searching for the optimal solution. In addition, the \( t \)-test is used to test the significant differences between the proposed metaheuristics results. The performance of the proposed metaheuristics is examined and validated with two instance groups. The main results of the present research are as follows.

1. The proposed metaheuristics are very effective and efficient to obtain optimal solutions for most of the cases compared with the solutions of the proposed MILP and lower bounds.

2. The overall average relative percentage deviation RPD of SA with small-sized problems is 7.646\%, while TS is 4.101\%. Therefore, TS outperforms SA in considered small-sized problems.

3. For large-sized problems, the overall average relative percentage deviation RPD of SA is 4.377\%, while TS is 5.946\%. Therefore, SA outperforms TS in considered large-sized problems.

4. SA has a better performance and consumes less time than TS as the size of jobs becomes large.

Further study should be considered for the assessment and implementation of some useful information in metaheuristics to evaluate how SA and TS work with

| Table 7. | Comparisons metaheuristics for large sized instances. |
| --- | --- |
| No. | Problem size \( n \times m \) | SA | CPU time (s) | TS | CPU time (s) |
| RPD% | Ave. | Min. | Max. | Ave. | Min. | Max. | RPD% | Ave. | Min. | Max. | Ave. | Min. | Max. |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 20 \( \times \) 2 | 4.889 | 1.111 | 7.778 | 0.123 | 0.065 | 0.271 | 4.422 | 0 | 9.286 | 0.178 | 0.095 | 0.248 |
| 2 | 20 \( \times \) 3 | 1.667 | 0.417 | 2.500 | 1.098 | 0.799 | 1.371 | 1.905 | 1.25 | 2.500 | 1.098 | 0.799 | 1.371 |
| 3 | 20 \( \times \) 5 | 2.708 | 0 | 5.882 | 2.314 | 1.444 | 2.697 | 4.202 | 0 | 11.765 | 1.152 | 1.124 | 1.173 |
| 4 | 20 \( \times \) 7 | 6.310 | 0 | 11.667 | 2.147 | 1.390 | 2.286 | 6.667 | 0 | 17.500 | 4.518 | 4.084 | 4.761 |
| 5 | 50 \( \times \) 2 | 0.372 | 0 | 0.987 | 0.665 | 0.465 | 0.810 | 0.167 | 0 | 0.538 | 2.971 | 2.438 | 3.347 |
| 6 | 50 \( \times \) 3 | 0.727 | 0.526 | 1.053 | 1.231 | 1.000 | 1.495 | 3.684 | 1.579 | 5.614 | 4.802 | 4.726 | 4.876 |
| 7 | 50 \( \times \) 5 | 6.845 | 2.083 | 10 | 2.689 | 2.689 | 2.689 | 9.940 | 8.75 | 10.833 | 2.432 | 2.038 | 2.521 |
| 8 | 50 \( \times \) 7 | 5.801 | 3.030 | 7.879 | 1.905 | 1.404 | 2.689 | 8.485 | 7.879 | 9.091 | 2.289 | 2.282 | 2.295 |
| 9 | 70 \( \times \) 2 | 0.072 | 0 | 0.288 | 0.599 | 0.507 | 0.772 | 0.740 | 0 | 1.871 | 5.304 | 4.836 | 5.621 |
| 10 | 70 \( \times \) 3 | 0.870 | 0 | 2.369 | 2.275 | 1.969 | 2.392 | 4.712 | 3.141 | 8.901 | 10.762 | 10.565 | 10.850 |
| 11 | 70 \( \times \) 5 | 4.118 | 0.882 | 6.176 | 2.301 | 2.234 | 2.396 | 5.252 | 4.118 | 8.529 | 10.104 | 9.844 | 10.285 |
| 12 | 70 \( \times \) 7 | 5.952 | 1.667 | 8.333 | 2.340 | 2.324 | 2.352 | 8.571 | 3.750 | 10.833 | 10.366 | 9.972 | 10.841 |
| 13 | 100 \( \times \) 2 | 0.359 | 0.109 | 0.796 | 0.978 | 0.876 | 1.198 | 1.493 | 0.930 | 3.063 | 11.509 | 11.355 | 11.586 |
| 14 | 100 \( \times \) 3 | 0.955 | 0 | 1.980 | 2.229 | 1.583 | 2.345 | 2.746 | 1.815 | 4.538 | 11.478 | 11.287 | 11.677 |
| 15 | 100 \( \times \) 5 | 7.720 | 3.404 | 12.340 | 2.281 | 2.148 | 2.400 | 9.483 | 6.383 | 14.468 | 11.735 | 11.547 | 11.882 |
| 16 | 100 \( \times \) 7 | 8.958 | 1.622 | 15.135 | 2.189 | 2.162 | 2.201 | 11.390 | 4.865 | 14.595 | 11.222 | 11.208 | 11.241 |
| 17 | 150 \( \times \) 2 | 0.591 | 0.215 | 0.862 | 2.255 | 2.237 | 2.266 | 1.348 | 0.819 | 1.853 | 13.788 | 13.566 | 13.967 |
| 18 | 150 \( \times \) 3 | 6.923 | 2.125 | 8.942 | 2.257 | 2.253 | 2.264 | 8.333 | 6.374 | 9.870 | 14.025 | 13.637 | 14.445 |
| 19 | 150 \( \times \) 5 | 10.743 | 9.067 | 12.800 | 2.268 | 2.262 | 2.274 | 11.524 | 8.667 | 14.533 | 14.252 | 13.961 | 14.456 |
| 20 | 150 \( \times \) 7 | 10.969 | 7.627 | 13.051 | 2.249 | 2.235 | 2.260 | 13.860 | 12.281 | 15.088 | 0.903 | 0.797 | 0.922 |

Table 8. The \( t \)-tests with 95\% confidence on the \( C_{max} \) and CPU time for all problem instances and Algorithms.

| Problem size | Algorithm | N | Mean | SD | T-value | p-Value |
| --- | --- | --- | --- | --- | --- | --- |
| Small \( C_{max} \) | SA | 12 | 7.65 | 4.66 | 2.02 | 0.057 |
| | TS | 12 | 4.10 | 3.91 | 2.35 | 0.021 |
| CPU time | SA | 12 | 1.621 | 0.600 | 5.52 | <0.000 |
| | TS | 12 | 0.544 | 0.311 | 3.05 | 0.004 |
| Large \( C_{max} \) | SA | 20 | 4.38 | 3.63 | -1.29 | 0.205 |
| | TS | 20 | 5.95 | 4.05 | 1.32 | 0.189 |
| CPU time | SA | 20 | 1.809 | 0.727 | -4.70 | <0.000 |
| | TS | 20 | 7.22 | 5.10 | 1.34 | 0.188 |
Figure 3. Box plot of overall average RPD for $C_{\text{max}}$ of small sized instances.

Figure 4. Box plot of overall average for CPU time values of small sized problem.

Figure 5. Box plot of overall average RPD for $C_{\text{max}}$ of large sized problem.

Figure 6. Box plot of overall average for CPU time values of large sized problem.

Figure 7. Box plot of average RPD for $C_{\text{max}}$ versus a number of machines of large sized problem.

Figure 8. Box plot of average RPD for $C_{\text{max}}$ versus a number of instances of large sized problem.
other optimization techniques. Moreover, the proposed method can be extended for the considered scheduling problem with limited buffer capacity constraints.

Declaration of conflicting interests
The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding
The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The authors are grateful to the Raytheon Chair for Systems Engineering for funding.

ORCID iD
Husam Kaid https://orcid.org/0000-0003-3608-013X

References
1. Bitar A, Dauzère-Pérès S and Yugma C. Unrelated parallel machine scheduling with new criteria: complexity and models. Comput Oper Res 2021; 132: 105291.
2. Zhang X, Liu S-C, Lin W-C, et al. Parallel-machine scheduling with linear deteriorating jobs and preventive maintenance activities under a potential machine disruption. Comput Ind Eng. 2020; 145: 106482.
3. Anghinolfi D, Paolucci M and Ronco R. A bi-objective heuristic approach for green identical parallel machine scheduling. Eur J Oper Res 2021; 289: 416–434.
4. Rivera G, Porras R, Sanchez-Solís JP, et al. Outranking-based multi-objective PSO for scheduling unrelated parallel machines with a freight industry-oriented application. Eng Appl Artif Intell 2022; 108: 104556.
5. Asadpour M, Hodaei Z, Azami M, et al. A green model for identical parallel machines scheduling problem considering tardy jobs and job splitting property. Sustainable Oper Comput 2022; 3: 149–155.
6. Keshavarz T, Karimi E and Shakhshi-Niaei M. Unrelated parallel machines scheduling with sequence-dependent setup times to minimize makespan and tariff charged energy consumption. Adv Ind Eng 2021; 55: 91–113.
7. Jouhari H, Lei D, AA Al-qaness M, et al. Sine-cosine algorithm to enhance simulated annealing for unrelated parallel machine scheduling with setup times. Mathematics 2019; 7: 1120.
8. Azizoglu M. Parallel machine scheduling to minimize total cost functions. PhD Doctoral Program, Middle East Technical University, 1994.
9. Pinedo M and Hadavi K. Scheduling: theory, algorithms and systems development. In: Gaul W, Bachem A, Habenicht W, et al. (eds) Operations research proceedings 1991, vol. 1991. Berlin, Heidelberg: Springer.
10. Sarıçeki and Çelik C. Two meta-heuristics for parallel machine scheduling with job splitting to minimize total tardiness. Appl Math Model 2011; 35: 4117–4126.

11. Yeh W-C, Lai P-J, Lee W-C, et al. Parallel-machine scheduling to minimize makespan with fuzzy processing times and learning effects. Inf Sci 2014; 269: 142–158.

12. Alharkan I, Saleh M, Ghaleb MA, et al. Tabu search and particle swarm optimization algorithms for two identical parallel machines scheduling problem with a single server. J King Saud Univ Eng Sci 2020; 32: 330–338.

13. Krim H, Zufferey N, Potvin J-Y, et al. Tabu search for a parallel-machine scheduling problem with periodic maintenance, job rejection and weighted sum of completion times. J Sched 2022; 25: 89–105.

14. Noman MA, Alatefi M, Al-Ahmari AM, et al. Tabu search algorithm based on lower bound and exact algorithm solutions for minimizing the makespan in non-identical parallel machines scheduling. Math Probl Eng 2021; 2021: 1856734.

15. Wang S, Zou H and Wang S. A Tabu-GA-based parallel scheduling with restrained tool resources. Proc Inst Mech Eng Part B: J Eng Manuf 2022; 236: 39–50.

16. Moussa M, Azar A and Ghatari AR. A new approach based on matching theory and simulated annealing for parallel machine scheduling problems. 2021. DOI: 10.21203/rs.3.rs-919404/v1.

17. Balin S. Non-identical parallel machine scheduling using genetic algorithm. Expert Syst Appl 2011; 38: 6814–6821.

18. BilgeU, Kiraç F, Kurtulan M, et al. A tabu search algorithm for parallel machine total tardiness problem. Comput Oper Res 2004; 31: 397–414.

19. Fanjul-Peyro L and Ruiz R. Iterated greedy local search methods for unrelated parallel machine scheduling. Eur J Oper Res 2010; 207: 55–69.

20. Vallada E and Ruiz R. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. Eur J Oper Res 2011; 211: 612–622.

21. Fanjul-Peyro L and Ruiz RJ. Scheduling unrelated parallel machines with optional machines and jobs selection. Comput Oper Res 2012; 39: 1745–1753.

22. Balin S. Non-identical parallel machine scheduling with fuzzy processing times using genetic algorithm and simulation. Int J Adv Manuf Syst 2012; 61: 1115–1127.

23. Joo CM and Kim BS. Non-identical parallel machine scheduling with sequence and machine dependent setup times using meta-heuristic algorithms. Ind Eng Manage Syst 2012; 11: 114–122.

24. Flieszar K, Charalambous C and Hindi KS. A variable neighborhood descent heuristic for the problem of makespan minimization on unrelated parallel machines with setup times. J Intell Manuf 2012; 23: 1949–1958.

25. Joo CM and Kim BS. Hybrid genetic algorithms with dispatching rules for unrelated parallel machine scheduling with setup time and product availability. Comput Ind Eng 2015; 85: 102–109.

26. Geyik F and Elibal K. A linguistic approach to non-identical parallel processor scheduling with fuzzy processing times. Appl Soft Comput 2017; 55: 63–71.

27. Sevindik K. Parallel machine scheduling subject to machine availability constraints. Ankara: Bilkent University, 2006.

Appendix

Notations

PM preventive maintenance
MILPM mixed integer linear programming model
LPT longest processing time
LB lower bound
α the environment of the machine
β the job characteristics and constraints
γ The objective function
n number of jobs
m number of machines
j, k the index of job, j, k = 1, 2, ..., n + m
i the index of machine, i = 1, 2, ..., m
pji processing time of job j on machine i
stji starting time of the job j on machine i
Cji job / completion time on machine i
Cmax maximum completion time or makespan
Q big number
lBi starting time of the PM on machine i
lEi ending time of the PM on machine i
Xji
\[
\begin{align*}
X_{ji} &= \begin{cases} 
1 & \text{if job } j \text{ is processed on machine } i \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

Lee C-Y. Machine scheduling with an availability constraint. J Global Optim 1996; 9: 395–416.

Al-Harkan IM and Qamhan AA. Optimize unrelated parallel machines scheduling problems with multiple limited additional resources, sequence-dependent setup times and release date constraints. IEEE Access. 2019; 7: 171533–171547.

Al-Shayea AM, Saleh M, Alatefi M, et al. Scheduling two identical parallel machines subjected to release times, delivery times and unavailability constraints. Processes 2020; 8: 1025.

Al-Ahmari A, Kaid H, Li Z, et al. A new MINLP continuous time formulation for scheduling optimization of oil refinery with unreliable CDUs. Math Probl Eng 2022; 2022: 1298495.

Glover F. Tabu search—part I. ORSA J Comput 1989; 1: 190–206.

Kaid H, Alharkan I, Ghaleb A, et al. Metaheuristics for identical parallel machines scheduling to minimize mean tardiness. In: 2015 international conference on industrial engineering and operations management (IEOM), Dubai, UAE, 2015, pp.1–6. IEEE.

Kirkpatrick S, Gelatt CD and Vecchi MP. Optimization by simulated annealing. Science 1983; 220: 671–680.

Metropolis N, Rosenbluth AW, Rosenbluth MN, et al. Equation of state calculations by fast computing machines. J Chem Phys 1953; 21: 1087–1092.
$Y_{jki}$: \begin{align*} &1 \text{ if job } k \text{ is processed after job } j \text{ on machine } i \\ &0 \text{ otherwise} \end{align*}

$Z_{ji}$: \begin{align*} &1 \text{ if preventive maintenance on machine } i \text{ is at job } j \\ &0 \text{ otherwise} \end{align*}

$t_0$: the initial temperature

$s_0$: the best schedule solution in the scheduling problem

$s$: the initial schedule solution in the scheduling problem

$f(s_0)$: the fitness function of the best solution in the scheduling problem

$f(s)$: the fitness function of the initial solution in the scheduling problem

Method$_{sol}$: the makespan obtained by metaheuristic

Opt$_{sol}$: the obtained optimal solution by Lingo solver