Unrelated Parallel Machines Scheduling Problem Using an Ant Colony Optimization Approach

Y. K. Lin, H. T. Hsieh, F. Y. Hsieh

Abstract—Total weighted tardiness is a measure of customer satisfaction. Minimizing it represents satisfying the general requirement of on-time delivery. In this research, we consider an ant colony optimization (ACO) algorithm to solve the problem of scheduling unrelated parallel machines to minimize total weighted tardiness. The problem is NP-hard in the strong sense. Computational results show that the proposed ACO algorithm is giving promising results compared to other existing algorithms.

Keywords—ant colony optimization, total weighted tardiness, unrelated parallel machines.

I. INTRODUCTION

PARALLEL machine scheduling problems have been studied by many researchers ([6]–[25]) in recent years. Parallel machine scheduling problems are important because most real world manufacturing workgroups have more than one machine. Parallel machine algorithms can be reduced for the single machine problems and also many job shop algorithms, such as shifting bottleneck heuristics, also called parallel machine algorithms. Though there is extensive literature on parallel machine scheduling problems, most of it is limited to situations in which the processing times or speed rates are the same across all machines. When machines are not identical to one another and cannot be completely correlated by simple rate adjustments, they are said to be unrelated parallel machines. Based on the complex hierarchy of deterministic scheduling ([7]–[25]) among the traditional classification of parallel machine environments, unrelated parallel machines are some of the most difficult problems to solve.

This research considers the problem of scheduling unrelated parallel machines to minimize total weighted tardiness (TWT). TWT is defined as \( \sum w_j T_j \), where \( T_j = \max (C_j - d_j, 0) \), \( w_j \) is the weight of job \( j \) and \( d_j \) is the due date of job \( j \). TWT is a measure of customer satisfaction. Minimizing it represents satisfying the general requirement of on-time delivery.

Following the three-field notation of [12], we refer to this problem as \( R || \sum w_j T_j \). The problem is NP-hard in the strong sense, since its special case with \( m = 1 \) is already NP-hard in the strong sense ([15]). Research related to scheduling parallel machines to minimize total weighted tardiness is limited. Based on shortest processing time (SPT) and earliest due date (EDD) principles, [22] proposed a heuristic (PSK) to minimize the mean tardiness for the single-machine sequencing problem (1||\( \bar{T}_j \)). [14] proposed a heuristic (KPM) for the \( P||T_j \) problem, based on extending the PSK heuristic from the single-machine tardiness problem to a parallel-machine setting. [1] presented a modified due date (MDD) algorithm for the \( P||w_j T_j \) problem. [8] developed a heuristic algorithm based on tabu search to solve the problem of simultaneously selecting and scheduling parallel machines to minimize the sum of machine holding cost and job tardiness cost. [27] solved the \( P||T_j \) problem using a branch-and-bound algorithm, and [5] proposed several efficient heuristic algorithms for the \( P||T_j \) problem. According to [23], most unrelated parallel machine scheduling problems are NP-hard. [23] presented a survey of algorithms for single- and multi-objective unrelated parallel machine deterministic scheduling problems but indicated that unrelated parallel machine problems remain relatively unstudied. In particular, the study noted that there are a few solution approaches to minimize due-date-related criteria. [17] solved the \( R||\sum w_j T_j \) problem using a branch-and-bound algorithm that can solve a problem size up to 4 machines and 18 jobs. The Apparent Tardiness Cost (ATC) rule proposed by [29], originally designed for the \( 1||\sum w_j T_j \) problem, can be used to solve the \( R||\sum w_j T_j \) problem heuristically. [18] modified the ATC rule so that it can solve the \( R||\sum w_j T_j \) problem. They proposed a heuristic, ATC-I, for the \( R||\sum w_j T_j \) problem. Computational results show that the study’s proposed ATC-I outperformed other existing heuristics (MDD, KPM, and ATC).

ACO is a constructive meta-heuristic that has been used to solve scheduling problems in recent years. [2] used an ACO algorithm to solve the single machine total tardiness problem (1||\( \sum T_j \)). [4] proposed an ACO algorithm that incorporates local search for the single machine total weighted tardiness problem (1||\( \sum w_j T_j \)).
[19] proposed an ACO algorithm that uses global pheromone information instead of using only local pheromone information for the \( \sum w_i T_j \) problem. Similarly, [30] presented an ACO algorithm for the \( \sum w_i T_j \) problem that can effectively improve the robustness of various simple constructive heuristics. Moreover, [13] proposed a fast ACO algorithm for the \( \sum w_i T_j \) problem. [16] used an ACO algorithm for scheduling single machine tardiness problems with sequence-dependent setups (1|\( s_j \)|\( \sum w_i T_j \)). [26] used an ACO algorithm to minimize makespan for identical parallel machines with sequence-dependent setup problems (\( P|s_j|\sum w_i T_j \)). [31] and [20] both studied scheduling unrelated parallel machines to minimize total weighted tardiness using an ACO algorithm (\( R||\sum w_i T_j \)). [28] suggested an ACO algorithm to solve scheduling identical parallel machines to minimize total weighted tardiness (\( P||\sum w_i T_j \)). [21] used an ACO algorithm for scheduling jobs with incompatible families on parallel batch machines (\( P-batch,incompatible||\sum w_i T_j \)). [3] studied parallel machine scheduling problems with sequence-dependent setup times to minimize makespan (\( P|s_j|C_{\text{max}} \)) using an ACO, a simulated annealing (SA), and a variable neighborhood search (VNS) hybrid algorithm.

In this research, we present an ACO algorithm for the \( R||\sum w_i T_j \) problem.

II. APPLICATION OF ACO ALGORITHM

ACO ([9]–[10]–[11]) is a meta-heuristic for solving hard combinatorial optimization problems. It is inspired by the pheromone trail laying and following behavior of real ants, which uses pheromones as a communication medium. In a double bridge experiment conducted on ants, the result showed that after an initial transitory phase in which some oscillations can appear, eventually most of the ants choose the shortest path. The ACO algorithm imitates this optimization capability of selecting the shortest path from the ant colony system. The ACO algorithm is based on indirect communication within a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails.

The (artificial) ants in the ACO algorithm implement a randomized construction heuristic that probabilistically selects the solution components of the problem based on the pheromone trail and possibly available heuristic information, which is based on the input data of the problem to be solved. The (artificial) pheromone trails in the ACO algorithm serve as distributed, numerical information for the ants to probabilistically construct solutions to the problem being solved and to reflect their search experience. In this research, we apply the ACO algorithm to solve the problem of scheduling unrelated parallel machines to minimize total weighted tardiness. The following notation is used to define the problems.

\[
\begin{align*}
\text{m} & \quad \text{The number of machines} \\
\text{n} & \quad \text{The number of jobs} \\
\text{U} & \quad \text{The set of unscheduled jobs} \\
\text{p}_{ij} & \quad \text{The processing time of job } j \text{ on machine } i \\
\text{w}_j & \quad \text{The weight of job } j \\
\text{d}_j & \quad \text{The due date of job } j \\
\text{l}_i & \quad \text{Makespan of the scheduled jobs on machine } i \\
\text{\tau}_{ij}(t) & \quad \text{Pheromone trail intensity of assigning job } j \text{ on machine } i \text{ in iteration } t \\
\eta_j & \quad \text{The heuristic desirability of assigning job } j \text{ on machine } i \\
\alpha & \quad \text{Relative influence of heuristic information} \\
\beta & \quad \text{Relative influence of pheromone information} \\
\rho_{\text{local}} & \quad \text{Local pheromone evaporation rate (} 0 < \rho_{\text{local}} < 1 \text{)} \\
\rho_{\text{global}} & \quad \text{Global pheromone evaporation rate (} 0 < \rho_{\text{global}} < 1 \text{)} \\
\text{q}_{\text{m0}} & \quad \text{The user-specified number such that } 0 \leq q_{\text{m0}} \leq 1 \\
\text{q}_{\text{j0}} & \quad \text{The user-specified number such that } 0 \leq q_{\text{j0}} \leq 1 \\
\text{ANTS} & \quad \text{The number of ants in each ant colony} \\
\text{ITER} & \quad \text{Total number of iterations to be run} \\
\end{align*}
\]

The ACO algorithm

Step 1: Initialization. Set ACO parameters \( \alpha, \beta, \rho_{\text{local}}, \rho_{\text{global}}, q_{\text{m0}}, q_{\text{j0}}, \text{ITER}, \text{ANTS} \). Initialize, \( \text{TWT}^* = \infty \), \( \text{iter}=1 \), \( \text{ants}=1 \), \( U={1,\ldots,n_f} \), \( t_i=0 \), \( i=1,2,\ldots,m \).

Step 2: Machine selection. Let \( q_m \) be the random number generated from the uniform distribution [0,1] and \( q_{\text{m0}} \) be the user-specified number such that \( 0 \leq q_{\text{m0}} \leq 1 \). Select the machine \( i^* \) from the following equation,

\[
\begin{align*}
\text{select } i^* = \left\{ \begin{array}{ll}
\min \{t_i\}, & \text{if } q_m \leq q_{\text{m0}} \\
I, & \text{otherwise}
\end{array} \right.
\end{align*}
\]

where \( I \) is a probability distribution with \( P_i = \frac{1/t_i}{\sum_{p=1}^{m} 1/t_p} \), \( i=1,2,\ldots,m \).

Step 3: Job selection. Let \( q_j \) be the random number generated from the uniform distribution [0,1] and \( q_{\text{j0}} \) be the user-specified number such that \( 0 \leq q_{\text{j0}} \leq 1 \). Select the job \( j^* \) from the following equation,

\[
\begin{align*}
\text{select } j^* = \left\{ \begin{array}{ll}
\arg \max \{t_{ij}^* \}, & \text{if } q_j \leq q_{\text{j0}} \\
J, & \text{otherwise}
\end{array} \right.
\end{align*}
\]

where...
\[ \eta_{i,j}^* = 1 - e^{-\frac{w_j \max(d_j - p_{i,j} - t_{i,j}, 0)}{K\tau_{i,j}}} \quad (1) \]

and \( J \) is a probability distribution with

\[ P_{i,j} = \begin{cases} \frac{\tau_{i,j}^a(t) \eta_{i,j}^\beta}{\sum_{j \in U} \tau_{i,j}^a(t) \eta_{i,j}^\beta}, & \text{if } j \in U \\ 0, & \text{otherwise} \end{cases} \]

Step 4: Machine re-selection. For job \( j^* \), find machine \( i^* \) such that

\[ p_j^{i^*} = \min \left\{ p_j^i \mid t_i + p_j^i - d_j \leq 0, 1 \leq i \leq m \right\} \]

If \( i^* \) cannot be found from the above equation, conclude that there is no job that can be assigned without being tardy, then calculate \( i^* \) from \( i^* = \arg \min_{j \in U} (t_i + p_j^i - d_j) \).

Step 5: Assign job \( j^* \) on machine \( i^* \). Set \( U = U \setminus j^* \) and \( t_{i^*} = t_{i^*} + p_j^{i^*} \). Update local pheromone for the selected machine \( i^* \) and job \( j^* \) as

\[ \tau_{i,j} = (1 - \rho) \tau_{i,j} + \rho \tau_{i,j}^a(t) \]

where

\[ \tau_0 = 1/(ANTS \cdot TWTT) \quad \text{and} \quad TWTT(\text{AC-I}) \]

is the ATC-I ([18]) objective value.

Step 6: Repeat Steps 2-5 until all jobs are scheduled.

Step 7: Apply local search procedure to improve the given schedule.

Step 8: Find current TWT for the current solution. If \( TWTT < TWTT^* \), then \( TWTT^* = TWTT \). Update ants=ants+1.

If ants \( \leq \) ANTS, set \( U \setminus \{\ldots,n\} \) and \( t_i = 0, i = 1,2,\ldots,m \) and go to Step 2. If ants > ANTS, go to Step 9.

Step 9: Update the global pheromone of the best solution found in the current iteration as

\[ \Delta \tau_j(t + 1) = (1 - \rho) \Delta \tau_j(t) + \rho \tau_j(t) \]

where

\[ \Delta \tau_j(t) = 1/TWTT^* \]

and \( TWTT^* \) is the best TWT value up to the current iteration. Set iter=iter+1. If iter \( \leq \) ITER, then set ants=1, \( U = \{\ldots,n\} \) and \( t_i = 0, i = 1,2,\ldots,m \) and go to Step2. If iter > ITER, go to Step 10.

Step 10: Terminate the ACO algorithm.

In Step 2, when \( q_m \leq q_{m0} \), an ant selects the first available machine \( i^* \) that leads to the smallest makespan among the parallel machines (i.e., \( t_{i^*} = \min_{j \in U} t_j \)). When \( q_m > q_{m0} \), then an ant randomly selects machine \( I \) from the probability distribution formed by the probabilities \( P_I \).

In Step 3, once machine \( i^* \) is selected, the next decision of an ant is to select the job. Job selection considers both the heuristic information \( (\eta_{i,j}^*) \) and pheromone trails \( (\tau_{i,j}^a(t)) \).

\( \tau_{i,j}^a(t) \) is the pheromone trail associated with the assignment of job \( j \) to machine \( i^* \). The parameter \( t \) is used for the current iteration of the ACO algorithm. \( \eta_{i,j}^* \) is the heuristic desirability of assigning job \( j \) to the selected machine \( i^* \). \( \eta_{i,j}^* \) is a modification of assigning job \( j \) to the selected machine \( i^* \). \( \eta_{i,j}^* \) is a modification of ATC index defined by Eq.(1). More precisely, we add \( w_j \) into the exponential term of the original ATC index.

In Step 9, global pheromone updating is applied. Global pheromone updating is intended to provide a greater amount of pheromone to the schedule with the best performance. Therefore, only the ant that found the best schedule up to the current iteration is permitted to deposit pheromone.

III. COMPUTATIONAL RESULTS

In this section, we present several computational results on the performance of the proposed ACO algorithm. The ACO
algorithm was implemented in C++ and run on a computer with 2.5 GHz Pentium(R) Dual-Core CPU and 2G RAM. Processing times were generated from the uniform distribution [1,100]. The value of \( w_i \) for each \( j \) was chosen randomly from the uniform distribution [1.10]. Due dates were generated from the uniform distribution \( P(1 - T - R/2), P(1 - T + R/2) \), where \( P = \sum_{i=1}^{m} \sum_{j=1}^{m} p_{ij}/m^2 \), \( T \) is the average tardiness factor, and \( R \) is the relative range of due dates, as was done in [24]. We study cases with tight due dates since it is more challenging. Hence, due dates factor \( T \) was set to 0.8, and \( R \) was set to 0.4 and 0.8. We also used 4 machines with 20 jobs (4m20n) to represent small problem instances and used 10 machines with 100 jobs (10m100n) to represent large problem instances. For each combination of due date tightness and problem instance size, 20 problem instances were randomly generated. According to our extensive computational experimentation, ACO algorithm-related parameters were set to \( \alpha =1, \beta =3, \rho_{\text{local}} =0.01, \rho_{\text{global}} =0.01, q_{m0} =0.9, q_{j0} =0.9, \) ANTS =20, and ITER=250.

### Table I

<table>
<thead>
<tr>
<th>Instances</th>
<th>( T=0.8, R=0.4 )</th>
<th>( T=0.8, R=0.8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1118</td>
<td>1118</td>
</tr>
<tr>
<td>2</td>
<td>590</td>
<td>590</td>
</tr>
<tr>
<td>3</td>
<td>568</td>
<td>568</td>
</tr>
<tr>
<td>4</td>
<td>1045</td>
<td>1045</td>
</tr>
<tr>
<td>5</td>
<td>2689</td>
<td>2689</td>
</tr>
<tr>
<td>6</td>
<td>927</td>
<td>927</td>
</tr>
<tr>
<td>7</td>
<td>1557</td>
<td>1557</td>
</tr>
<tr>
<td>8</td>
<td>863</td>
<td>863</td>
</tr>
<tr>
<td>9</td>
<td>1188</td>
<td>1188</td>
</tr>
<tr>
<td>10</td>
<td>693</td>
<td>693</td>
</tr>
<tr>
<td>11</td>
<td>459</td>
<td>459</td>
</tr>
<tr>
<td>12</td>
<td>796</td>
<td>796</td>
</tr>
<tr>
<td>13</td>
<td>489</td>
<td>492</td>
</tr>
<tr>
<td>14</td>
<td>1468</td>
<td>1468</td>
</tr>
<tr>
<td>15</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>16</td>
<td>1528</td>
<td>1528</td>
</tr>
<tr>
<td>17</td>
<td>1375</td>
<td>1375</td>
</tr>
<tr>
<td>18</td>
<td>947</td>
<td>947</td>
</tr>
<tr>
<td>19</td>
<td>433</td>
<td>433</td>
</tr>
<tr>
<td>20</td>
<td>624</td>
<td>624</td>
</tr>
<tr>
<td>Average</td>
<td>974.20</td>
<td>974.35</td>
</tr>
<tr>
<td>(Algo.-opt.):opt.</td>
<td>-</td>
<td>0.0002</td>
</tr>
<tr>
<td>Num. of opt. found</td>
<td>-</td>
<td>19</td>
</tr>
</tbody>
</table>

When the due dates were very tight (\( T = 0.8, R = 0.4 \)), the ACO algorithm deviated 0.02% from the optimal solution, the ACO-SV algorithm deviated 16.09% from the optimal solution, and the GA deviated 3.51% from the optimal solution. Moreover, out of 20 problem instances, the ACO algorithm solved 19 problem instances to optimal solution, the ACO-SV solved 0 problem instances to optimal solution, and the GA solved 2 problem instances to optimal solution. When the due dates were tight (\( T = 0.8, R = 0.8 \)), the ACO algorithm deviated 0.03% from the optimal solution, the ACO-SV algorithm deviated 16.86% from the optimal solution, and the GA deviated 2.2% from the optimal solution. Moreover, out of 20 problem instances, the ACO algorithm solved 18 problem instances to optimal solution, the ACO-SV solved 0 problem instances to optimal solution, and the GA solved 8 problem instances to optimal solution.

For large problem instances, we compared the average relative percentage deviations from our proposed ACO algorithm and the number of instances solved to best heuristic solution. The results from this experimentation are given in Table 2; they show that the proposed ACO algorithm outperformed other existing algorithms.

### A. Comparison of algorithms

[28] proposed an ACO algorithm (named ACO-SV) for the \( P \| \sum w_j T_j \) problem. Lin et al. (2011) proposed a genetic algorithm (GA) for the \( R \| \sum w_j T_j \) problem. We compare the proposed ACO algorithm with these two existing algorithms. The two algorithms ACO-SV and GA have been re-implemented in our work. For the small problem instances, we compare the average relative percentage deviations from the optimal solution obtained from the branch-and-bound algorithm proposed by [17] to the number of instances solved to optimal solution. For the large problem instances we compare the average relative percentage deviations from our proposed ACO algorithm and the number of instances solved to best heuristic solution. For each problem instance, \( K \) value for AU index in the ACO-SV algorithm and Eq. (1) in our proposed ACO algorithm was set to 0.6. Table 1 gives computational results for small problem instances, which show that the ACO algorithm outperformed the ACO-SV algorithm and the GA in terms of total weighted tardiness and number of instances solved to optimal solution.
When the due dates were very tight ($T = 0.8, R = 0.4$), the ACO-SV algorithm deviated 19.68% from the ACO algorithm. Moreover, the ACO-SV algorithm deviated 29.18% from the ACO algorithm. When the due dates were tight ($T = 0.8, R = 0.8$), the ACO-SV algorithm deviated 171.23% from the ACO algorithm.

### IV. CONCLUSIONS

This research proposed an ACO algorithm for the problem of scheduling unrelated parallel machines to minimize total weighted tardiness. Computational results showed that the proposed ACO algorithm outperformed other existing algorithms (ACO-SV and GA) in terms of total weighted tardiness and the number of instances solved to best heuristic solution.

### REFERENCES


