Robust project scheduling with two constrained resources

Rong-Hwa Huang, Chang-Lin Yang*

Department of Business Administration, Fu Jen Catholic University, Taiwan

Email address:
026299@mail.fju.edu.tw(R. Huang), 051125@mail.fju.edu.tw(C. Yang)

To cite this article:

Abstract: Distinct from ordinary project scheduling to minimize project completion time, this study conducts robust project scheduling with two resources constraints. In this study, actual start time of each operation is dynamic. The latest finish time minus actual finish time is the slack time available to each operation. The objective of this study is to determine all project operation times and slack times. This study utilized a novel parallel tabu search scheme to simulate multiple CPUs searching for the optimum value. The parallel tabu search outperformed the conventional tabu search in terms of exploration. For model verification, test datasets from the project scheduling problem library (PSPLIB) were adopted. Analytical results show that parallel tabu search exceeded the conventional tabu search in optimizing the objective value.

Keywords: Resource Constraint, Project Scheduling, Parallel Tabu Search, Slack Time, Robustness, Project Completion Time

1. Introduction

Managers often consider situations and resource constraints during decision-making. However, as humans have limited rationality, perfect forecasting is generally impossible. Thus, each decision is accompanied by risk. In real business manufacturing environments, projects involve many decisions. An unexpected interference can result in enormous losses. Under extreme global competition, businesses must focus on project scheduling to enhance cost efficiency and service quality. Managers hope to achieve effective existing resource allocation in certain situations with the support of project scheduling.

Past studies demonstrated that solving the resource-constrained project scheduling problem (RCPSP) is difficult and complex. The two main models that have been applied to solve the RCPSP are mathematical and heuristic. According to Bell and Han (1991), for a project with more than 50 operations, mathematical models take considerable time to construct and compute. Solution effectiveness is therefore limited, and obtaining the optimum value is not guaranteed. Mathematical models only suit small-scale problems and are not utilized for practical applications.

Cogill and Hindi (2007) and Ibaraki et al. (2008) utilized integer programming to minimize completion time for the RCPSP. A large number of decision variables and complex constraints reduce solving effectiveness. Davis (1975) applied linear programming and dynamic programming to solve the RCPSP and encountered the same effectiveness problem. For large-scale problems, computation times of mathematical models grow exponentially as most scheduling problems are (NP)-hard. Therefore, heuristic methods outperform mathematical models by providing near optimum solutions in an acceptable amount of time, which is meaningful when compared to an optimum solution without time efficiency.

Kelley (1963), who first developed the RCPSP, showed that the parallel method performed better than the series method. Doctor (1990) tested the RCPSP with a case and developed an operational priority selection rule. With improvements in computer science and project scheduling techniques, several optimization algorithms have been developed and solving effectiveness and quality have progressed. Dorigo et al. (1997) and Merkel et al. (2002) developed the ant colony optimization (ACO) scheme that uses pheromone and the state transition rule to solve the RCPSP; they also compared the results with simulated annealing and tabu search. Glover (1977) used tabu search to solve the RCPSP. Multiple neighborhood searches are utilized to avoid local optimization. A flexible memory structure can record moving decisions of different cycles in

Most conventional studies assumed unlimited or one resource constraint to minimize project completion time. To further reflect real manufacturing conditions, this study considers two resource constraints to minimize project completion time and maximize slack time with a parallel tabu search. Three sets of weights, (0.25, 0.75), (0.5, 0.5), and (0.75, 0.25), are given to satisfy managers under different situations. The objective is to determine the most robust and efficient project schedule.

2. Integer Programming Model

This study considers the robustness of project scheduling with multiple resource constraints. All activities are non-preemptive and must be processed sequentially. Actual activity start times are dynamic and depending on previous activities. Resource consumption and processing time are known and cannot be split. The difference in most recent finish time and actual finish time is available slack for an activity. An adjusted parameter determines the duration of available slack among activities.

Since an activity with excessive available slack leads to unnecessary resource consumption, a weight is assigned, such that the effect of slack time on an objective function is a decreasing function of time. Finally, the objective function is to maximize the difference between the weighted sum of available slack and weighted project completion time.

2.1. Notification

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>total number of activities</td>
</tr>
<tr>
<td>( J_j )</td>
<td>activity of number ( j ), ( j = 1, 2, \ldots, n )</td>
</tr>
<tr>
<td>( J_0 )</td>
<td>dummy start activity</td>
</tr>
<tr>
<td>( J_{n+1} )</td>
<td>dummy end activity</td>
</tr>
<tr>
<td>( d_j )</td>
<td>processing time of activity ( J_j ), ( j = 1, 2, \ldots, n )</td>
</tr>
</tbody>
</table>

2.2. Mathematical Model

Objective function:

\[
\text{MaxZ} = w_1 \cdot \text{Robust} - w_2 \cdot C_{\text{max}} \quad (1)
\]

Constraints:

\[
\text{Robust} = \sum_{j=1}^{n} Robust_j \quad (2)
\]

\[
Robust_j = \sum_{x=1}^{d_j} [1 - (x-1) \cdot w_3] \quad (3)
\]

\[
D_j = LF_j - F_j \quad \forall j = 1, 2, \ldots, n \quad (4)
\]

\[
LF_j = \min \{ S_i \} \quad J_i \in \text{Succ}_j \quad i \neq j \quad \forall i = 0, 1, \ldots, n \quad (5)
\]
Objective function:

\[ F_j = S_j + d_j; \quad \forall j = 1, 2, \ldots, n \]  

(6)

\[ ES_0 = b \]  

(7)

\[ ES_j = \max \{ F_i \}; \quad J_i \in \text{Pred}_j; \quad i \neq j; \quad \forall i = 1, 2, \ldots, n+1; \quad \forall j = 1, 2, \ldots, n+1 \]  

(8)

\[ S_j \geq ES_j; \quad \forall j = 1, 2, \ldots, n \]  

(9)

\[ ES_{s+1} = C_{\text{max}} \]  

(10)

\[ C_{\text{max}} = LF_{s+1} \]  

(11)

\[ \sum_{j=1}^{n} r_{nj} \cdot k_{nji} \leq R_{n}; \quad k_{nji} = \begin{cases} 1 & \text{if } J_j \text{ is processing on time } t \\ 0 & \text{otherwise} \end{cases}; \quad t = 1, 2, \ldots, T \]  

(12)

\[ \sum_{j=1}^{n} r_{mj} \cdot k_{mji} \leq R_{m}; \quad k_{mji} = \begin{cases} 1 & \text{if } J_j \text{ is processing on time } t \\ 0 & \text{otherwise} \end{cases}; \quad t = 1, 2, \ldots, T \]  

(13)

\[ T = \sum_{j=1}^{n} d_j \]  

(14)

2.3. Interpretation

Objective function:
(1) Maximize project robustness and minimize project completion time.
(2) Constraints:
(3) Project robustness is the sum of available slack for all weighted activities.
(4) Weighted available slack of activity \( J_r \).
(5) Available slack of activity \( J_j \) is the difference between latest finish time and actual finish time.
(6) Latest finish time of activity \( J_j \) is the difference between minimum actual start time in its successor set.
(7) Actual completion time of activity \( J_j \) is the sum of its actual start time and processing time.
(8) Earliest start time of the dummy start activity is the confirmed project start time.
(9) Earliest start time of activity \( J_j \) is the maximum actual finish time in its predecessor set.
(10) Actual start time of \( J_j \) is not less than its earliest start time.
(11) Earliest start time of the dummy end activity equals project completion time.
(12) Project completion time equals latest finish time of the dummy end activity.
(13) Labor consumption of activity \( J_j \) is not larger than the labor limit per unit time.
(14) Consumption of machine hours by activity \( J_j \) does not exceed the machine hour limit per unit time.

(15) Overall project processing time equals the sum of processing times for all activities.

3. Algorithm

Tabu search has a better ability to avoid repeat solutions than other heuristics. However, tabu search can relatively easily fall into a local optimization. The parallel tabu search simulates multiple processors to randomly generate a start solution simultaneously, begins searching in neighborhoods, and then moves to the best solution in a neighborhood or a solution that satisfies the aspiration criterion. The route is memorized in a tabu list to avoid repeat searches.

3.1. Parallel Tabu Search Algorithm

Figure 1 shows the parallel tabu search procedure in this study.

![Figure 1. Procedures for solving robust project scheduling](image-url)

**Step 1:** Simulate multiple CPUs; each \( CPU_i \) generates a start solution \( S_i \).

**Step 2:** Search for the best solution, \( S_{\text{best}} \), in a neighborhood.

**Step 3:** Determine whether \( S_i \) is searched by other \( CPU_i \). If searched, go to step 8; otherwise, go to step 4.

**Step 4:** Determine whether \( S_i \) is in a tabu list. If yes, go to step 5; otherwise, go to step 6.
Step 5: Determine whether $S_k$ satisfies the aspiration criterion. If yes, go to step 6; otherwise, go to step 8.

Step 6: Determine whether $S_k$ is better than $S^*$. If yes, go to step 7; otherwise, go to step 8.

Step 7: Movement confirmed. Refresh the tabu list and let $S_k = S^*$.

Step 8: Determine whether the solution reaches the stop criterion. If yes, end project; otherwise, go to step 1.

3.2. Example

An illustration using the parallel tabu search to solve the available slack scheduling problem is presented below. Figure 2 shows the project network chart.

Assuming that search time of each CPU$_i$ is the same, the related terms are set as follows:
1. Two recoverable resources are available ($R_i$, $i = 1, 2$).
2. $w_1 = 0.75$, $w_2 = 0.25$, and $w_3 = 0.1$.
3. Set tabu list length ($TL = \sqrt{8}$, carrying 3).
4. Two processors ($CPU_i$, $i = 1, 2$).
5. Each CPU$_i$ randomly picks a start solution ($S_i$) to begin searching.

End search if any of the following situations are encountered.
1. Maximum start solution transferring time of CPU$_i$ is 1.
2. Unimproved times of CPU$_i$ are set to one;
3. Both CPU$_1$ and CPU$_2$ stop searching.

Begin search:
The CPU$_1$ randomly generates a start solution, $S_1 = (-1.075)$; the processing sequence is 1-2-3-4-6-7-8. When searching for best neighbor solution, $S_2 = 0.35$, the processing sequence is 1-2-3-4-6-5-7-8. This move is not repeated and not in the tabu list. Let $S_2 = S^* = 0.35$.

Thus, $S_1 = S^* = 0.35$, and the processing sequence is 1-2-3-4-6-5-7-8. When searching for the best neighbor solution, $S_1 = 0.6$, the processing sequence is 1-2-3-4-6-7-8. Movement is sustained since it is not repeated or in the tabu list. Let $S_1 = S^* = 0.6$.

MaxZ is unimproved, this satisfies the stop criterion and CPU$_1$ stops searching.

Notably, CPU$_2$ randomly generated a start solution $S_2 = S^* = 0.025$, and the processing sequence is
1-3-2-4-6-5-7-8. When searching for best neighbor solution, $S_k = 0.35$, the processing sequence is 1-2-3-4-6-5-7-8. Thus, $S_k$ repeats the preceding solution obtained by $CPU_1$, repeat the search.

$$k = 0.35$$

The processing sequence is 1-2-3-4-6-5-7-8. Thus, $k = 0.35$ repeats the preceding solution obtained by $CPU_1$, repeat the search.

$$J_j \text{Robust} = \sum_{j=1}^{n} \max_{j} C_j = \sum_{j=1}^{n} \text{Robust}_j - w_2 \cdot C_{\text{max}}$$

$J_3:1+0.9+0.8 \cdot 1+2.7 = 3.7$

$10.75x3.7 - 0.25x11 = 0.025$

$(2, 3)$

$J_2:1+0.9 \cdot 1.9+1.9 = 3.8$

$0.75x3.8 - 0.25x9 = 0.6$

$(6, 7)$

Notably, $CPU_2$ randomly generated a start solution $S_2 = S' = 0.275$, and the processing sequence is 1-2-3-4-7-6-5-8. When searching for the best neighbor solution, $S_2 = 0.6$, the processing sequence is 1-2-3-4-6-7-5-8. Since $CPU_2$ repeats the preceding best neighbor solution obtained by $CPU_1$, repeat the search. However, maximum start transferring times are set to 1 and satisfy the stop criterion; thus, $CPU_2$ stops searching.

$$J_j \text{Robust} = \sum_{j=1}^{n} \max_{j} C_j = \sum_{j=1}^{n} \text{Robust}_j - w_2 \cdot C_{\text{max}}$$

$J_3:1+0.9+0.8 \cdot 1+2.7 = 3.7$

$10.75x3.7 - 0.25x11 = 0.025$

$(2, 3)$

$J_2:1+0.9 \cdot 1.9+1.9 = 3.8$

$0.75x3.8 - 0.25x9 = 0.6$

$(6, 7)$

Table 1. Test data types

<table>
<thead>
<tr>
<th>Numbers of work (n)</th>
<th>Constraints of $R_1$</th>
<th>Constraints of $R_2$</th>
<th>Length of $TL (\sqrt{n})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>60</td>
<td>20</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>90</td>
<td>20</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>120</td>
<td>30</td>
<td>30</td>
<td>12</td>
</tr>
</tbody>
</table>

4.2. Algorithm Effectiveness

This research simulates 17 types of CPU numbers 1, 5, 10, 15, 20, 25, …, 80. Each type is given three different weights—$(w_1, w_2) = (0.75, 0.25)$, $(w_1, w_2) = (0.5, 0.5)$, and $(w_1, w_2) = (0.25, 0.75)$—for available slack and project completion time. Let $w_3 = 0.1$ for all situations. Table 2 shows the effect of CPU numbers on the optimum solution and computation time. Average solutions and computation time for the 30 datasets in each project scale and CPU numbers are as follows.

Figures 4 and 5 show data test results. The graphs indicate that when the number of CPUs exceeds 50, solutions tend to converge and computation time keeps increasing. For cost efficiency, this study suggests that 50 CPUs in a parallel tabu search is the most suitable number for the RCPSP.

4. Data Test and Analysis

This study obtains test data from the PSPLIB to verify algorithm effectiveness and robustness.

4.1. Test Data

To verify algorithm performance, 30 test datasets were acquired from the PSPLIB for projects with 30, 60, 90, and 120 activities. Each test dataset has a tabu list length and two recoverable resources, labor ($R_1$) and machine hour ($R_2$). Table 1 shows the test data types. The test program is compiled with C++ language and executed on an 2.21GHz AMD Athlon(tm) 64×2 Dual Core Processor 4200+, with 1.00GB RAM.
4.3. Algorithm Robustness

For the robustness test, one dataset is chosen for each of the four project scales. With 3 different weights of available slack and project completion time, each dataset is tested 30 times. Standard deviation is computed to verify robustness.

According to effectiveness test results, only 40, 50, 60, 70, and 80 CPUs are included in robustness test for time efficiency. Each number of CPUs is tested 30 times with four project scales of 30, 60, 90, and 120. Table 3 shows test results.

Standard deviation decreases as the number of CPUs increases (Fig. 6). Standard deviation increases as project scale increases. The explanation is that the size of the solution set increases when project scale increases. Notably, 50 CPUs was the most stable. In the effectiveness test, 50 CPUs in total was both effective and robust for parallel tabu search solving the RCPSP.

Table 2. Effectiveness test result

<table>
<thead>
<tr>
<th>Amount of CPU (n)</th>
<th>30</th>
<th>60</th>
<th>90</th>
<th>120</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>60</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>90</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>120</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 3. Robustness test result

<table>
<thead>
<tr>
<th>Numbers of works (n)</th>
<th>30</th>
<th>60</th>
<th>90</th>
<th>120</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 40</td>
<td>0.42</td>
<td>2.80</td>
<td>5.28</td>
<td>7.22</td>
</tr>
<tr>
<td>CPU 50</td>
<td>0.35</td>
<td>2.43</td>
<td>3.78</td>
<td>6.62</td>
</tr>
<tr>
<td>CPU 60</td>
<td>0.39</td>
<td>2.24</td>
<td>3.81</td>
<td>7.29</td>
</tr>
<tr>
<td>CPU 70</td>
<td>0.36</td>
<td>2.38</td>
<td>3.80</td>
<td>6.82</td>
</tr>
<tr>
<td>CPU 80</td>
<td>0.37</td>
<td>2.59</td>
<td>3.78</td>
<td>6.13</td>
</tr>
</tbody>
</table>

* Standard deviation $\sigma = \sqrt{\frac{1}{n-1} \sum (x_i - \bar{x})^2}$
5. Conclusion

Previous project scheduling studies only minimized project completion time. However, in real business situations, unexpected interference exists and projects can be delayed. In this study, a novel parallel tabu search algorithm was applied to solve the RCPSP with 2 robust two recoverable resources. With different weights, a researcher can decide whether to minimize delay or minimize project completion time. Under unstable or risky conditions, minimizing delay is recommended.

Test datasets are drawn from the PSPLIB. The four projects have 30, 60, 90, 120 activities; each is tested with 30 different datasets. Two recoverable resources exist. Let \( w_1 = 0.1 \). The researcher tested weights—(\( w_1, w_2 \)) = (0.75, 0.25), (\( w_1, w_2 \)) = (0.5, 0.5), and (\( w_1, w_2 \)) = (0.25, 0.75)—with the number of simulated CPU of 1, 5, 10, …, 80, totally 17 types of CPU number. The objective value improved as the number of CPUs increased. At 40 CPUs, the average solution converged while computation time continued increasing. With 50 CPUs, standard deviation was convergent in the robustness test. Therefore, for effectiveness and robustness, 50 CPUs is optimal for a robust parallel tabu search.

References


