Science Journal of Business and Management
2013; 1(1): 19-25
Published online June 30, 2013 (http://www.sciencepublishinggroup.com/j/sjbm)
doi: 10.11648/j.sjbm.20130101.14

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:
Rong-Hwa Huang, Chang-Lin Yang. Robust Project Scheduling with Two Constrained Resources. *Science Journal of Business and Management.* Vol. 1, No. 1, 2013, pp. 19-25. doi: 10.11648/j.sjbm.20130101.14

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 companied 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. Boctor (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 the short and long term to escalate from local optimization. Notably, tabu search is widely applied to scheduling and NP-hard problems. Minghe (2006) solved the factory position problem with a tabu search and compared tabu search with other heuristics. Huang and Liao (2008) combined ACO and tabu search to optimize the scheduling problem. Taillard (1991) and Firchter (1994) simulated multiple processors in tabu search to accelerate the solving process and obtain better solving quality compared to that of conventional tabu search. Bozejko and Wodecki (2004) utilized block property to distribute computation tasks and reduce communication among processors during tabu search. Al-Fawzan and Mohamed (2005) solved the multi-objective RCPSP of minimizing project completion time and maximizing slack time using tabu search. Wang (2005) developed constraint satisfaction problems (CSPs) to overcome uncontrollable interference as constraint functions change and repair a possible breach of a scheduling solution. This increased the flexibility of conventional scheduling tools.

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

n	= total number of activities
J_j	= activity of number j, $j = 1, 2, \cdots, n$
J_0	= dummy start activity
J_{n+1}	= dummy end activity
d_j	= processing time of activity J_j, $j = 1,2,...,n$, $(d_j \geq 0)$
S_j	= start time of activity J_j, $j = 1,2,...,n$
F_j	= finish time of activity J_j, $j = 1,2,...,n$
r_{hj}	= labor consumed by activity J_j per unit time, $j = 1,2,...,n$
r_{mj}	= machine hourly consumption of activity J_j per unit time, $j = 1,2,...,n$
R_h	= labor limit per unit time
R_m	= machine hour limit per unit time
b	= confirmed project start time
t	= scheduling time of a project, $t = 1,2,...,T$
T	= overall project processing time
D_j	= available slack for activity J_j
$Robust_j$	= weighted available slack of activity J_j, $j = 1,2,...,n$
$Robust$	= sum of slack for all activities with weight w_3
$C_{\max}$	= project completion time (makespan)
ES_0	= earliest start time of the dummy start activity
ES_{n+1}	= earliest start time of the dummy end activity
ES_j	= earliest start time of activity J_j, $j = 1,2,...,n$
LS_j	= latest start time of activity J_j, $j = 1,2,...,n$
LF_j	= latest finish time of activity J_j, $j = 1,2,...,n$
LF_{n+1}	= latest finish time of the dummy end activity
$Pred_j$	= immediate predecessors set of activity J_j
$Succ_j$	= immediate successors set of activity J_j
w_1	= weight of *Robust*
w_2	= weight of $C_{\max}$
w_3	= decreasing rate of activity J_j available slack

2.2. Mathematical Model

Objective function:

$$MaxZ = w_1 \cdot Robust - w_2 \cdot C_{max} \tag{1}$$

Constraints:

$$Robust = \sum_{j=1}^{n} Robust_j \tag{2}$$

$$Robust_j = \sum_{x=1}^{D_j} \left[1 - (x-1) \cdot w_3\right] \tag{3}$$

$$D_j = LF_j - F_j\,; \quad \forall j = 1,2,...,n \tag{4}$$

$$LF_j = \min\{S_i\}\,; \quad J_i \in Succ_j\,; \quad i \neq j\,; \quad \forall i = 0,1,,...,n\,; \quad \forall j = 0,1,...,n \tag{5}$$

$$F_j = S_j + d_j\,;\ \ \forall j = 1,2,...,n \tag{6}$$

$$ES_0 = b \tag{7}$$

$$ES_j = \max\{F_i\}\,;\ \ J_i \in \Pr ed_j\,;\ \ i \neq j\,;\ \ \forall i = 1,2,...,n+1\,;$$

$$\forall j = 1,2,...,n+1 \tag{8}$$

$$S_j \geq ES_j\,;\ \ \forall j = 1,2,...,n \tag{9}$$

$$ES_{n+1} = C_{max} \tag{10}$$

$$C_{max} = LF_{n+1} \tag{11}$$

$$\sum_{j=1}^{n} r_{hj} \cdot k_{hjt} \leq R_h\,;\, k_{hjt} = \begin{cases} 1 & J_j \text{ is processing on time } t \\ 0 & \text{otherwise} \end{cases}\,;$$

$$t = 1,2,...,T \tag{12}$$

$$\sum_{j=1}^{n} r_{mj} \cdot k_{mjt} \leq R_m\,;\, k_{mjt} = \begin{cases} 1 & J_j \text{ is processing on time } t \\ 0 & \text{otherwise} \end{cases}\,;$$

$$t = 1,2,...,T \tag{13}$$

$$T = \sum_{j=1}^{n} d_j \tag{14}$$

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_j.
(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.

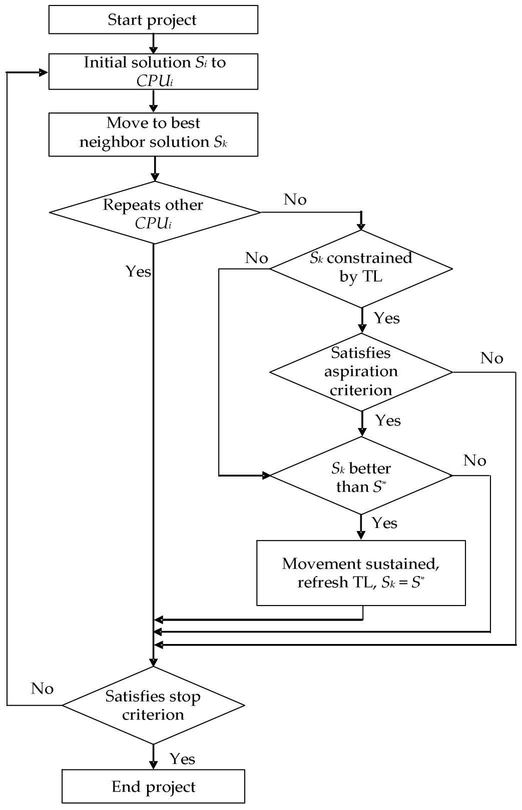
Start project
Initial solution S_i to CPU_i
Move to best neighbor solution S_k
Repeats other CPU_i
No
Yes
No
S_k constrained by TL
Yes
Satisfies aspiration criterion
No
Yes
S_k better than S^*
No
Yes
Movement sustained, refresh TL, $S_k = S^*$
No
Satisfies stop criterion
Yes
End project

Figure 1. *Procedures for solving robust project scheduling*

Step 1: Simulate multiple CPUs; each CPU_i generates a start solution S_i.
Step 2: Search for the best solution, S_k, in a neighborhood.
Step 3: Determine whether S_k is searched by other CPU_i s. If searched, go to step 8; otherwise, go to step 4.
Step 4: Determine whether S_k 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.

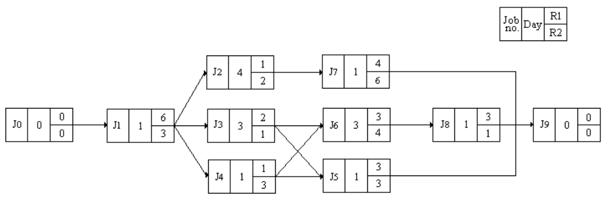

Figure 2. *Project network chart*

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 , $i = 1,2$). Resource limits are $R_1 = 6$ and $R_2 = 6$.
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, $S_1 = S^* = (-1.075)$; the processing sequence is 1-2-3-4-5-6-7-8. When searching for best neighbor solution, $S_k = 0.35$, the processing sequence is 1-2-3-4-6-5-7-8. This move is not repeated and not in the tabu list. Let $S_k = S^* = 0.35$.

Sequence(*Jj*)	$Robust_j$	$\sum_{j=1}^{n} Robust_j$	C_{max}	$MaxZ = w_1 \cdot \sum_{j=1}^{n} Robust_j - w_2 \cdot C_{max}$	*TL*
1-2-3-4-5-6-7-8	J_4:1+0.9	1.9	10	0.75x1.9-0.25x10=(-1.075)	(5,6)
1-2-3-4-6-5-7-8	J_2:1+0.9 J_4:1+0.9	1.9+1.9=3.8	10	0.75x3.8-0.25x10=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, $S_k = 0.6$, the processing sequence is 1-2-3-4-6-7-5-8. Movement is sustained since it is not repeated or in the tabu list. Let $S_k = S^* = 0.6$.

Sequence(*Jj*)	$Robust_j$	$\sum_{j=1}^{n} Robust_j$	C_{max}	$MaxZ = w_1 \cdot \sum_{j=1}^{n} Robust_j - w_2 \cdot C_{max}$	*TL*
1-2-3-4-6-5-7-8	J_2:1+0.9 J_4:1+0.9	1.9+1.9=3.8	10	$0.75 \times 3.8 - 0.25 \times 10 = 0.35$	(5,7)
1-2-3-4-6-7-5-8	J_2:1+0.9 J_4:1+0.9	1.9+1.9=3.8	9	$0.75 \times 3.8 - 0.25 \times 9 = 0.6$	

Thus, $S_1 = S^* = 0.6$, and the processing sequence is 1-2-3-4-6-7-5-8. When searching for best neighbor solution, $S_k = 0.6$, the processing sequence is 1-2-3-4-6-7-8-5. Since MaxZ is unimproved, this satisfies the stop criterion and CPU_1 stops searching.

Sequence(*Jj*)	$Robust_j$	$\sum_{j=1}^{n} Robust_j$	C_{max}	$MaxZ = w_1 \cdot \sum_{j=1}^{n} Robust_j - w_2 \cdot C_{max}$	*TL*
1-2-3-4-6-7-5-8	J_2:1+0.9 J_4:1+0.9	1.9+1.9=3.8	9	0.75x3.8-0.25x9=0.6	(5,8)
1-2-3-4-6-7-8-5	J_2:1+0.9 J_4:1+0.9	1.9+1.9=3.8	9	0.75x3.8-0.25x9=0.6	

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.

Sequence(Jj)	$Robust_j$	$\sum_{j=1}^{n} Robust_j$	C_{max}	$MaxZ = w_1 \cdot \sum_{j=1}^{n} Robust_j - w_2 \cdot C_{max}$	TL
1-3-2-4-6-5-7-8	J_3:1 J_4:1+0.9+0.8	1+2.7=3.7	11	0.75x3.7-0.25x11=0.025	(2,3)
1-2-3-4-6-5-7-8	J_2:1+0.9 J_4:1+0.9	1.9+1.9=3.8	10	0.75x3.8-0.25x10=0.35	

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, $S_k = 0.6$, 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 satisfie the stop criterion; thus, CPU_2 stops searching.

Sequence(Jj)	$Robust_j$	$\sum_{j=1}^{n} Robust_j$	C_{max}	$MaxZ = w_1 \cdot \sum_{j=1}^{n} Robust_j - w_2 \cdot C_{max}$	TL
1-2-3-4-7-6-5-8	J_3:1 J_4:1+0.9+0.8	1+2.7=3.7	10	0.75x3.7-0.25x10=0.275	(6,7)
1-2-3-4-6-7-5-8	J_2:1+0.9 J_4:1+0.9	1.9+1.9=3.8	9	0.75x3.8-0.25x9=0.6	

Both CPU_1 and CPU_2 stop searching, end the algorithm. The objective value found by CPU_1 is 0.6 (Fig. 3).

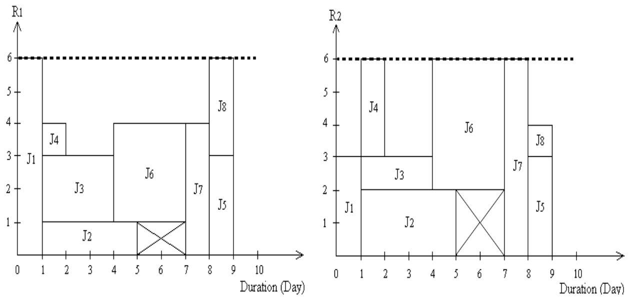

Figure 3. *Tabu search result*

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 complied with C++ language and executed on an 2.21GHz AMD Athlon(tm) 64×2 Dual Core Processor 4200+, with 1.00GB RAM.

Table 1. *Test data types*

Numbers of work (n)	**Constraints of R_1**	**Constraints of R_2**	**Length of TL ($\sqrt{n}$)**
30	20	20	6
60	20	20	8
90	20	20	10
120	30	30	12

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.

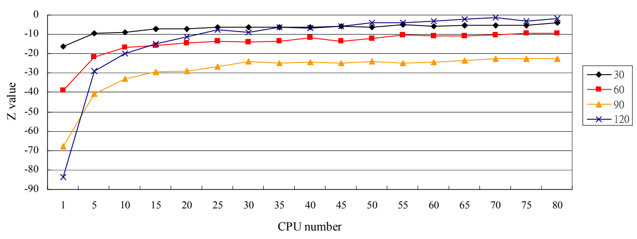

Figure 4. *Effective analyses – Z value*

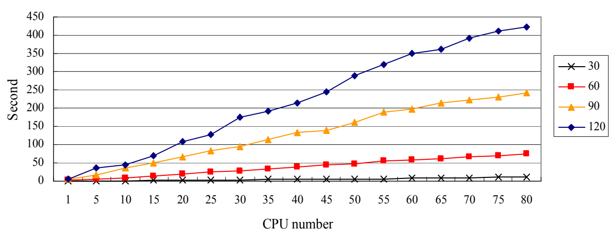

Figure 5. *Effective analyses – execution time*

Table 2. *Effectiveness test result*

No. of work (n) / Amount of CPU	30		60		90		120	
	Z Value	Execution Time (sec)	Z Value	Execution Time (sec)	Z Value	Execution Time (sec)	Z Value	Execution Time (sec)
1	-16.14	0.25	-38.79	2.00	-67.71	5.70	-83.48	6.74
5	-9.67	0.68	-21.84	5.04	-40.76	16.13	-28.75	36.16
10	-8.86	1.25	-16.65	9.62	-33.23	36.87	-20.11	45.04
15	-7.04	1.81	-15.87	14.28	-29.22	48.85	-15.09	69.76
20	-7.41	2.44	-14.36	18.79	-28.85	67.64	-11.21	107.15
25	-6.31	2.88	-13.51	24.66	-26.87	82.44	-7.60	128.15
30	-6.18	3.67	-13.88	26.78	-23.91	95.50	-9.21	175.36
35	-6.49	4.26	-13.38	32.55	-24.83	114.57	-6.20	190.86
40	-6.23	4.80	-11.76	38.32	-24.57	132.10	-6.61	215
45	-5.91	5.44	-13.48	44.50	-24.98	137.92	-5.86	245.71
50	-6.19	5.98	-12.25	47.43	-24.04	160.22	-3.97	289.15
55	-4.98	6.53	-10.56	54.55	-24.76	189.00	-4.05	319.71
60	-6.02	7.56	-10.87	57.76	-24.26	196.69	-2.96	350.45
65	-5.49	8.12	-10.84	61.26	-23.38	213.36	-2.20	359.78
70	-5.57	9.53	-10.56	65.50	-22.55	221.95	-1.35	390.51
75	-5.38	10.24	-9.54	68.41	-22.52	229.37	-3.06	410.08
80	-4.21	11.50	-9.62	75.31	-22.65	242.18	-2.00	421.46

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 3. *Robustness test result*

Numbers of works (*n*)		**30**	**60**	**90**	**120**
CPU Number	CPU 40	0.42	2.80	5.28	7.22
	CPU 50	0.35	2.43	3.78	6.62
	CPU 60	0.39	2.24	3.81	7.29
	CPU 70	0.36	2.38	3.80	6.82
	CPU 80	0.37	2.59	3.78	6.13

* Standard deviation= $s=\sqrt{\frac{1}{n-1}\sum_{n=1}^{n}\left(x_i-\overline{x}\right)^2}$

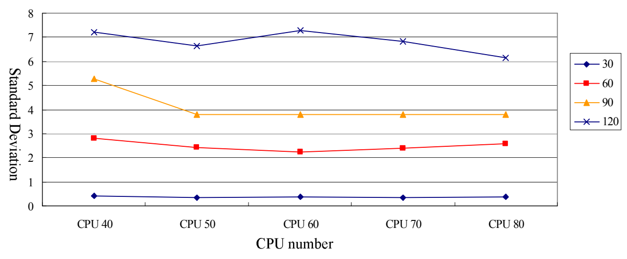

Figure 6. *Standard deviation of each CPU numbers - number of work is from 30 to120*

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_3 =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

[1] Al-Fawzan, M.A. and Mohamed, H. (2005), “A bi-objective model for robust resource-constrained project scheduling”, International Journal of Production economics, 96(2), 175-187.

[2] Bell, C.E. and Han, J., (1991), “A new heuristic solution method in resource-constrained project scheduling”, Naval Research Logistics, 38(3), 315-331.

[3] Boctor, F., (1990), “Some efficient multi-heuristic procedures for resource-constrained project scheduling”, European Journal of Operational Research, 49(1), 3-13.

[4] Bozejko, W. and Wodecki, M., (2004), Parallel tabu search method approach for very difficult permutation scheduling problems’, Parallel Computing in Electrical Engineering 2004 International Conference, 156-161.

[5] Cogill, R. and Hindi, H., (2007), “Optimal routing and scheduling in flexible manufacturing systems using integer programming”, IEEE Conference on Decision and Control, 4095-4102.

[6] Davis, E.W. and Patterson, J.H., (1975), “A comparison of heuristic and optimum solutions in resource-constrained project scheduling”, Management Science, 21(8), 944-955.

[7] Dorigo, M. and Gambardella, L.M., (1997), “Ant colony system: a cooperative learning approach to the traveling salesman problem”, IEEE Transactions on Evolutionary Computation, 1(1), 53-66.

[8] Fiechter, C.N., (1994), “A parallel tabu search algorithm for large traveling salesman problems”, Discrete Applied Mathematics, 51(3), 243-267.

[9] Glover, F., (1977), Tabu Search, Kluwer Academic Publishers, Boston.

[10] Huang, K.L. and Liao, C.J., (2008), “Ant colony optimization combined with tabu search for the job shop scheduling problem”, Computers & Operations Research, 35(4), 1030-1046.

[11] Ibaraki, T., Nomura, T. and Sasaki, M., (2008), “Integer programming approaches to the problem of network upgrading”, The 2008 International Symposium on Applications and the Internet, 229-232.

[12] Kelley, J.E., (1963), The critical path method: resources planning and scheduling, Prentice-Hall, Englewood Cliffs, New Jersey.

[13] Minghe, S., (2006), “Solving the incapacitated facility location problem using tabu search”, Computers & Operations Research, 33(9), 2563-2589.

[14] Merkle, D., Middendorf, M. and Schmeck, H., (2002), “Ant colony for resource constrained project scheduling”, IEEE Transactions on Evolutionary Computation, 6(4), 333-346.

[15] Taillard, E., (1991), “Robust tabu search for the quadratic assignment problem”, Parallel Computing, 17(4-5), 443-455.

[16] Wang, J., (2005), “Constraint-based schedule repair for product development projects with time-limited constraints”, International Journal of Production Economics, 95(3), 399-414.