Parallel-machine Scheduling with Precedence Constraints and Controllable Job-processing Times

Kailiang Xu[1], Rong Fei[2] and Gang Zheng[1]
[1]*School of Automation and Information Engineering, Xi'an University of Technology, Xi'an, Shaanxi, China*
[2]*School of Computer Science Engineering, Xi'an University of Technology, Xi'an, Shaanxi, China*
{*klxu, annyfei, zhenggang*}*@xaut.edu.cn*

Keywords: Parallel-machine, Precedence Constraint, Controllable Processing Time, Tabu-search.

Abstract: A parallel-machine scheduling problem with tree-formed precedence constraints is studied, where job-processing times can be compressed by allocating extra resource to the jobs. A tabu-search algorithm is developed in this paper, which searches for the near optimal solutions based on the structural characteristics of the problem. Experiment shows the algorithm is capable for providing satisfactory schedules for media and large-sized problems within acceptable computing time.

1 INTRODUCTION

This paper concerns on a parallel-machine scheduling problem with tree-formed precedence constraints, where job-processing times can be compressed by the consumption of extra resources, such as gas, fuel, electricity power, cash and labors. Vickson(Vickson, 1980) was one of the first researchers to consider the scheduling problems with controllable job-processing times. Motivated by his research work, a number of researchers focused their attention on scheduling problems with controllable job-processing times after 1980, and have achieved significant result (for example, Cheng et al.(Cheng et al., 2006), Janiak(Janiak, 1991), Shabtay and Kaspi(Shabtay and Kaspi, 2004; Shabtay and Kaspi, 2006), *etc*.). An excellent survey on this field has been given by Shabtay(Shabtay, 2007). Reader may refer to it for more information.

Alidaee and Ahmadian(Alidaee and Ahmadian, 1993) were the first to consider parallel machine system with controllable job-processing times. They considered two unrelated-machine problems. The first problem has an objective that equals total processing cost plus total flow time, and the second problem has an objective that equals total processing cost plus total weighted earliness and weighted tardiness. Under the assumption that all the jobs have the same linear compression rate, the problem is reduced to a transportation problem and is solved in $O(n^3m + n^2m\log(nm))$ time. Jansen and Mastrolilli(Jansen and Mastrolilli, 2004) successfully applied approximation schemes on several identical machine problems related with maximum completion time and total processing cost, including problem $P_m|lin, C_{\max} \leq K|\sum_{j=1}^{n} v_j u_j$, $P_m|lin, \sum_{j=1}^{n} v_j u_j \leq U|C_{\max}$ and $P_m|lin|C_{\max} + \sum_{j=1}^{n} v_j u_j$. Shabtay and Kaspi(Shabtay and Kaspi, 2006) studied parallel machine scheduling problems under the situation where operations are modeled by non-linear convex resource consumption functions. They showed that the general problem $P_m|conv|C_{\max}$ is $\mathcal{NP}$-hard, and obtained several polynomial time algorithms for special cases. The special case that preemption is allowed is also studied by several researchers. For example, Jansen and Mastrolilli(Jansen and Mastrolilli, 2004) showed that problem $P_m|lin, pmtn, C_{\max} \leq K|\sum_{j=1}^{n} v_j u_j$ can be solved in $O(n)$ time. Nowicki and Zdrzalka(Nowicki, 1995) provided an $O(n\max\{m, \log n\})$ time greedy algorithm to solve the uniform machine problem $Q_m|lin, a_j = 1, pmtn, C_{\max} \leq K|\sum_{j=1}^{n} v_j u_j$ problem.

Parallel machine scheduling with precedence constraints is widely studied in the context of the classical scheduling problems, but is not in literature under the condition that job-processing times are controllable. In this paper, we consider a parallel machine scheduling problem with precedence constraints and controllable job-processing times: Schedule a set of non-preemptive jobs $\mathcal{J} = \{1, 2, \ldots, n\}$ that have tree-formed precedence constraints on m identical machines. Job-processing times p_j are controllable, and can be modeled as the function of a continuously dividable resource u_j via a linear resource consumption

Xu, K., Fei, R. and Zheng, G.
Parallel-machine Scheduling with Precedence Constraints and Controllable Job-processing Times.
DOI: 10.5220/0006290804700476
In *Proceedings of the 6th International Conference on Operations Research and Enterprise Systems (ICORES 2017)*, pages 470-476
ISBN: 978-989-758-218-9

function,

$$p_j(u_j) = \bar{p}_j - \theta_j u_j \qquad 0 \le a_j \le p_j(u_j) \le b_j \le \bar{p}_j \tag{1}$$

where $\bar{p}_j$ is the non-compressed job-processing time and θ_j is the positive compression rate of job j. The objective is to determine the optimal schedule $\mathcal{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_m, p\}$, where $\sigma_i = \{\sigma_i(1), \sigma_i(2), \ldots, \sigma_i(n_i)\}$ is the optimal job-processing sequence on machine i, and $p = \{p_1, p_2, \ldots, p_n\}$ is the optimal job-processing times, such that the makespan of the jobs ($C_{\max} = \max_{j=1}^{n} C_j$) will not exceed the required deadline K, and the total resource consumption $U = \sum_{j=1}^{n} u_j$ is minimized. Using the three-field problem classification introduced by Graham *et al.*(Graham et al., 1979) and extended by Shabtay and Steiner(Shabtay, 2007), our problem can be denoted as $P_m \,|\, lin, tree, C_{\max} \le K \,|\, \sum_{j=1}^{n} u_j$.

The problem is strongly $\mathcal{NP}$-hard, which can be observed by the fact that it is still strongly $\mathcal{NP}$-hard to find a schedule whose makespan does not exceed deadline K when all the job-processing times are compressed to their lower bound a_j. Unless $\mathcal{P} = \mathcal{NP}$, it is impossible to find the optimal solution within acceptable computing time for large-scaled problems. Therefore, in this paper we designed a tabu-search algorithm to provide optimal or near optimal solutions for large-scaled problems.

Tabu-search(Glover, 1990) is essentially a meta-heuristic method that guides a heuristic local search procedure to explore the solution space beyond local optimality. A large number of successful tabu-search algorithms for scheduling problems can be found in literature (see, e.g., Dell'Amico and Trubian(Dell'Amico and Trubian, 1991), Bilge *et al.*(Bilge, 2004), Venditti *et al.*(Venidtti et al., 2010) and Xu et al.(Xu, 2010)).

2 TABU-SEARCH ALGORITHM

The scheduling algorithm needs to optimize job-processing sequence on the machines, as well as the processing times of the jobs. Suppose a job-processing sequence on the machines is decided, the optimal job-processing times can be calculated by the network flow method (NFM for convenience) introduced by Fulkerson(Fulkerson, 1961). Therefore, the tabu-search algorithm mainly concerns on searching for the optimal or near optimal job-processing sequences on the machines. In the following part of this section, the detail of the tabu-search algorithm will be discussed, which includes the initial solution generation, the neighborhood generation, the tabu mechanism, and the searching procedure.

2.1 The Objective Function

As the algorithm tries to minimize the resource consumption of the jobs, schedules are evaluated by their total resource consumption U. Therefore, the total resource consumption U is selected as the objective function in the tabu-search algorithm.

2.2 The Initial Solution

In this paper, the initial solution is constructed by the LTAP (Largest Total Amount Processing time first) method in the following way:

Algorithm 1: The LTAP method for initial solution generation.

1. Have all the job-processing times compressed to their lower bound a_j;
2. Sort jobs by their total amount of processing time, which is calculated by accumulating the processing time of jobs' successors and their own;
3. Assign jobs in the list to the machines. Each time, assign the first job in the list according to the following rules:
   (a) If there are one or more machines, where the last processed job is one of the direct predecessors of the job to be assigned, then assign it to the latest ready machine among them;
   (b) Otherwise, assign the job to the earliest ready machine.
4. Calculate the optimal processing times of the jobs by NFM.

Since parallel machine scheduling is $\mathcal{NP}$-hard, it shall be noted that the heuristic algorithm cannot guarantee a feasible initial solution when the deadline is very tight. However, as job-processing times are fully compressed, this method is still acceptable in most practical environment.

2.3 Neighborhood Generation

In each iteration, a number of generated neighboring schedules need be evaluated, which is time-consuming since objective values are calculated by solving the optimal resource allocation problem. Therefore, it is necessary to reduce neighboring schedules that need be evaluated. Tabu-search follows a so called "aggressive search" principle, that is, the

best schedule in the neighborhood will always be selected unless it is tabued. Therefore, if an operation leads to a neighboring schedule that is no better than another one, the operation need not be performed. If such dominated schedules can be identified by some simple rules before they are generated and evaluated, much computing time will be saved. In the following part, rules for "promising" neighboring schedules will be discussed based on the structural characteristic of the schedules and operations.

Neighboring schedules are normally generated by insert moves and swap moves. In this paper, insert moves are solely studied in neighborhood generation, which put single jobs to other processing positions either on their current machine or on other machines. As sequence constraint must be obeyed, it is supposed that jobs are always moved to feasible processing positions, such that they will not be processed earlier than their predecessors, or later than successors.

For parallel-machine problems with precedence constraints, critical path method (CPM) calculates the makespan of the jobs(Pinedo, 2012). By this method, jobs are classified as critical jobs and slack jobs. Critical jobs make up critical paths that decide the makespan, while slack jobs have no influence to the makespan. Moreover, it can be observed that $p_j = b_j$ for every slack job j. Therefore, when a slack job is moved to another processing position, the makespan will keep unchanged, if not be increased, which causes the total resource consumption not be reduced. The following lemma states this:

Lemma 1. Suppose job j is a slack job, then inserting job j to any other processing position cannot reduce the total resource consumption.

Therefore, insert moves are performed only on critical jobs.

Apart from this, other structural characteristics of the problem also help to reduce neighboring schedules. As is mentioned, an insert move may put a job to another feasible processing position on its current machine, or put it to another machine. The former operations are named **intra-machine insert moves**, while the latter **inter-machine insert moves**. The two kinds of operations will be discussed in the following.

Intra-machine Insert Moves. Suppose job j is inserted to another processing position on the same machine. If the processing of job j is advanced after insertion, it is inserted forward. Otherwise, it is inserted backward. If job j is processed immediately before job i, it is inserted to the front of job i. If job j is processed immediately after job k, it is inserted to the back of job k (As Fig.1 shows).

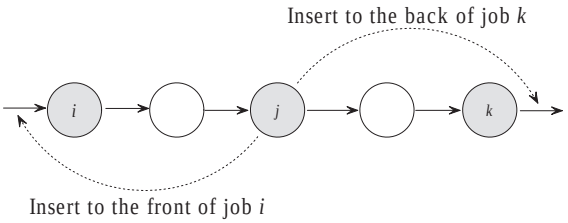

Figure 1: Intra-machine insert moves.

Suppose job i, j, k are processed on different machines, where job i is the direct predecessor and job k the direct successor of job j. In this case, it is said that job j has an emanating constraint to job k, and a sinking constraint from job i. If the precedence constraint $i \rightarrow j$ is part of a critical path, job j has a critical sinking constraint. If $j \rightarrow k$ is part of a critical path, then job j has a critical emanating constraint (As Fig.2 shows).

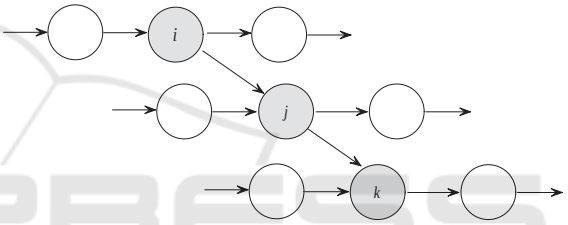

Figure 2: Critical sinking constraint and emanating constraint.

Lemma 2. Suppose job j has a critical emanating constraint in schedule $\mathcal{S}_1$. In the neighboring schedule $\mathcal{S}_2$ generated by inserting job j backward, job j still has a critical emanating constraint.

Lemma 3. Suppose job j has one or more critical sinking constraints in schedule $\mathcal{S}_1$. In the neighboring schedule $\mathcal{S}_2$ generated by inserting job j forward, job j still has one or more critical sinking constraints.

Corollary 1. Suppose job j has no critical sinking constraint in schedule $\mathcal{S}_1$. In the neighboring schedule $\mathcal{S}_2$ generated by inserting job j backward, job j has no critical sinking constraint.

Corollary 2. Suppose job j has no critical emanating constraints in schedule $\mathcal{S}_1$. In the neighboring schedule $\mathcal{S}_2$ generated by inserting job j forward, job j has no critical emanating constraint.

Lemma 4. Suppose there are jobs $j \prec j+1 \prec ... \prec k$ processed on the same machine in schedule $\mathcal{S}_1$, where job j has no critical sinking constraint, and jobs $j+1, ..., k$ have no critical emanating constraint. In this case, the total resource consumption of the neighboring schedule $\mathcal{S}_2$ generated by inserting job j to the back of job k is no less than that of $\mathcal{S}_1$.

Proof. (By contradiction) Assume the resource consumption of schedule $\mathcal{S}_2$ is smaller than that of the schedule $\mathcal{S}_1$, that is, $U_{\mathcal{S}_2} < U_{\mathcal{S}_1}$. Since job j has no critical sinking constraint in $\mathcal{S}_1$, it has no critical sinking constraint in $\mathcal{S}_2$. Similarly, job $j+1,...,k$ also have no critical emanating constraint in $\mathcal{S}_2$. Now construct a new schedule $\mathcal{S}_3$, where all the job-processing times are the same as those in $\mathcal{S}_2$, and job j is moved to its original position in $\mathcal{S}_1$. It can be easily observed that the makespan of $\mathcal{S}_3$ is no larger than K. Now generate a new schedule $\mathcal{S}_4$ by optimizing the job-processing times of $\mathcal{S}_3$. Obviously, there is $U_{\mathcal{S}_4} \leq U_{\mathcal{S}_3}$. However, since schedule $\mathcal{S}_4$ is exactly the same as $\mathcal{S}_1$, it causes a contradiction. The lemma is proved. □

Lemma 5. Suppose there are jobs $i \prec ... \prec j-1 \prec j$ processed on the same machine in schedule $\mathcal{S}_1$, where job j has no critical emanating constraint, and jobs $i,...,j-1$ have no critical sinking constraint. In this case, the total resource consumption of the neighboring schedule $\mathcal{S}_2$ generated by inserting job j to the front of job i is no less than that of $\mathcal{S}_1$.

Suppose there are job j and job k processed on different machines, where job k may or may not be the successor of job j. Because of the precedence constraints, when job j is inserted backward, job k must be processed after job j is completed. In this case, it is said that job j has an emanating constraint path to job k. In the similar way, it can also be defined that job j has a sinking constraint path from job i.

Lemma 6. Suppose there are job $j \prec j+1 \prec ... \prec k$ on the same machine in schedule $\mathcal{S}_1$, where job k has a critical emanating constraint to job l, while other jobs have no critical emanating and sinking constraint. Suppose job j has an emanating constraint path to job l. In this case, the total resource consumption of the neighboring schedule $\mathcal{S}_2$ generated by inserting job j to the back of job k is no less than that of $\mathcal{S}_1$.

Proof. (By contradiction) Assume the resource consumption of schedule $\mathcal{S}_2$ is smaller than that of the schedule $\mathcal{S}_1$, that is, $U_{\mathcal{S}_2} < U_{\mathcal{S}_1}$. Suppose job j has a direct successor job i processed on another machine (As Fig.6 shows). It can be easily seen that job j has a critical emanating constraint to job i, and no critical sinking constraint in schedule $\mathcal{S}_2$. It can also be seen that job $j+1,...,k$ have no critical emanating constraint in schedule $\mathcal{S}_2$. Now construct a new schedule $\mathcal{S}_3$, where all the job-processing times are the same as those in $\mathcal{S}_2$, and job j is moved to its original position in $\mathcal{S}_1$. It can be easily observed that the makespan of $\mathcal{S}_3$ is no larger than K. Now generate a new schedule $\mathcal{S}_4$ by optimizing the job-processing times of $\mathcal{S}_3$. Obviously, there is $U_{\mathcal{S}_4} \leq U_{\mathcal{S}_3}$. However, since schedule $\mathcal{S}_4$ is exactly the same as $\mathcal{S}_1$, it causes a contradiction. The lemma is proved. □

Lemma 7. Suppose there are job $j \prec j+1 \prec ... \prec k$ on the same machine in schedule $\mathcal{S}_1$, where job k has a critical emanating constraint to job l, while other jobs have no critical emanating and sinking constraint. Suppose job j has an emanating constraint path to job l. In this case, the total resource consumption of the neighboring schedule $\mathcal{S}_2$ generated by inserting job j to the back of job k is no less than that of $\mathcal{S}_1$.

Based on the above analysis, a set of rules for intra-machine insert moves are deduced in the following:

1. Job j has no critical emanating and sinking constraint.
   (a) It can be inserted to the back of job k, if job k is the first job with a critical emanating constraint, and job j has no emanating constraint path to the direct successor of job k;
   (b) It can be inserted to the front of job i, if job i is the first job with a critical sinking constraint, and job j has no sinking constraint path from the direct predecessor of job i.
2. Job j only has critical sinking constraint. It can be inserted to the back of the first job k with a critical sinking constraint.
3. Job j only has critical emanating constraint. It can be inserted to the front of the first job i with one or more critical emanating constraints.
4. Job j has critical sinking and emanating constraint. No intra-machine insert move shall be performed with it.

Inter-machine Insert Moves. Suppose there are job i and job j processed adjacently on the same machine, and job i or job j will be inserted to another machine. If job i is the predecessor of job j, the precedence constraint $i \prec j$ still exists after the operation. If not, the constraint will be eliminated. It can be observed, when a job is inserted to other machines, if no such precedence constraint can be eliminated, the resource consumption of the neighboring schedules will certainly not be reduced. Therefore, a job j can be inserted to other machines only if it satisfies one of the following conditions:

1. Job j is processed immediately after job i on the same machine, while job i is not a predecessor of job j;
2. Job j is processed immediately before job k on the same machine, while job k is not the successor of job j.

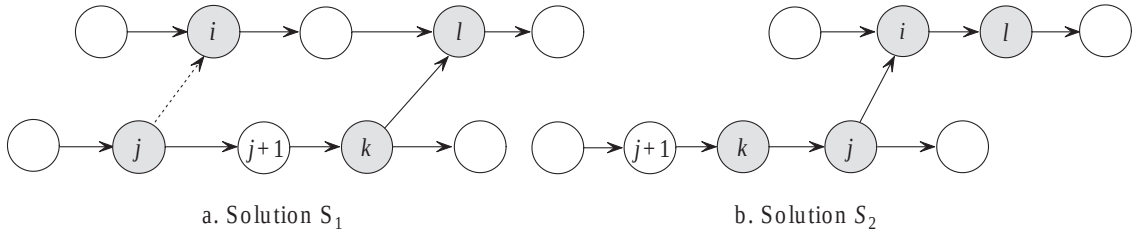

Figure 3: The illustration of lemma 6: (a) Schedule $\mathcal{S}_1$; (b) Schedule $\mathcal{S}_2$.

Otherwise, the operation should not be performed.

When job j is inserted to another machine m, it is first inserted to the earliest feasible processing position, and its optimal resource allocation is calculated. After that, it is inserted to other feasible positions using intra-machine insert moves, until the best processing position is determined, which will be recorded as the neighboring schedule generated by inserting job j to machine m.

2.4 The Tabu Mechanism

Tabu mechanism helps the searching procedure avoid being trapped in the local minimum. Typically, a list of mutations, which the procedure is not allow to make, is kept at any stage of the searching. Every time a mutation is made in the current schedule, the *reversed* mutation is entered at the top of tabu-list, while all other entries are pushed down one position and the bottom entry is deleted. In this paper, the schedule is modified by two kinds of moves, so the tabu-list needs to store them in different ways:

1. Intra-machine insert moves: When a selected schedule is generated by an intra-machine insert move performed on job j, the corresponding tabu-list entry is recorded as (i, j), where i is the job (may possibly be dummy) that is processed immediately before job j before the move is performed. Therefore, any intra-machine insert move that results in job j processed immediately after job i will be tabued.
2. Inter-machine insert moves: Suppose a schedule is generated by inserting job j from machine m to another machine. The tabu-list entry is recorded as (j, m), such that any inter-machine insert move that results in job j processed on machine m will be tabued.

2.5 The Tabu-search Algorithm

Based on the above discussion, the tabu-search algorithm is designed and presented in the following. The algorithm contains two input parameters that need be decided by users, which are:

- **MaxIter**: The algorithm stops when the objective function cannot be further improved after a number of iterations, defined by parameter MaxIter;
- **TabuDepth**: The maximum number of tabu entries that the tabu list contains, which is normally set between 6 and 10.

The tabu-search algorithm is formally stated as follows:

Algorithm 2: The tabu-search algorithm.

1. Have job-processing times compressed to their lower bound a_j, and schedule jobs by LTAP method. If the makespan is no larger than the deadline K, calculate the optimal job-processing times, and let the result be the initial solution.
2. Generate the initial solution by LTAP method. Let IterCounter $= 0$;
3. Improve the initial solution using insert moves:
   1. Generate neighboring schedules for every critical job using insert moves alone;
   2. Select among them the one with the smallest objective function value. If there are more than one such schedules, select one randomly;
   3. If the selected schedule is better than the current schedule, or the operation is not tabued, let the schedule replace the current schedule. Otherwise, remove the schedule from the neighborhood, and repeat Step 2.2;
   4. If the solution is improved, let IterCounter = 0. Otherwise, let IterCounter $=$ IterCounter $+ 1$;
   5. If IterCounter $>$ MaxCntInst, go to Step 3;
   6. If the objective function value of the solutions keeps unchanged over MaxCntObj times, push the objective function value of the selected schedule into the tabu-list. Otherwise, push the reversed operation that leads to the selected schedule into the tabu-list. Go to Step 2.1.
4. Select the best schedule obtained so far as the current schedule. Clear the tabu-list. Let IterCounter $= 0$;

5. Improve the current solution using insert and swap moves:
   1. Generate neighboring schedules for every critical job using insert moves and swap moves. If a schedule that is better than the current schedule is generated, then go to Step 4.2 directly;
   2. Select among them the one with the smallest objective function value. If there are more than one such schedules, select one randomly;
   3. If the selected schedule is better than the current schedule, or the operation is not tabued, let the schedule replace the current schedule. Otherwise, remove the schedule from the neighborhood, and repeat Step 4.2;
   4. If the solution is improved, let IterCounter = 0. Otherwise, let IterCounter = IterCounter + 1;
   5. If IterCounter > MaxCntSwp, go to Step 5;
   6. If the objective function value of the solutions keeps unchanged over MaxCntObj times, push the objective function value of the selected schedule into the tabu-list. Otherwise, push the reversed operation that leads to the selected schedule into the tabu-list. Go to Step 4.1.
6. Select the best schedule obtained so far as the solution, and exit.

3 NUMERICAL EXPERIMENT

A set of numerical experiments were performed in this section. The algorithm is implemented in C++ and is capable for parallel computing. The experiments were carried out on a personal computer with an Intel i7-2600 CPU with 4 cores and 8 independent threads. The algorithm was tested on problem instances generated by following method:

1. The minimum processing time a_j of the jobs are distributed randomly between [10, 50]. The maximum processing time $b_j = a_j + \delta_j$, where δ_j is also distributed between [10, 50];
2. The compression rate θ_j is distributed between [0.5, 1.5];
3. Each job has some predecessors, the number of them is distributed between [0, 5];
4. The maximum depth of the precedence tree is 5.

Because jobs are generated randomly, their number in each problem instance is different. For each instance, jobs are processed on 10 machines first, then on 25 machines. After the initial schedule is determined, a minimum makespan $T_{\min}$ is calculated with all the job-processing times $p_j = a_j$, and a maximum makespan $T_{\max}$ with $p_j = b_j$. The deadline $K = T_{\min} + (T_{\max} - T_{\min}) * \theta$, where θ is random between [0.3, 0.6].

Because no similar study on this problem is known in literature, the experimental result cannot be compared to other algorithms. Therefore, it is analyzed by comparing the scheduling result against the initial solution, which is shown Table 1 by following columns:

- n, m: The number of the jobs and the machines;
- Init, Rst: The resource consumption of the initial solution and that of the scheduling result;
- Imp: The improvement of the scheduling result against the initial solution, calculated as
$$Imp = \frac{Rst}{Init} 100\%$$
- Rst Time: The computing time when the best scheduling result is found;
- Total Time: The total computing time of the scheduling procedure.

The experimental result shows the heuristic initial solutions are significantly improved by the tabu-search algorithm. For most problem instances with job number around 200, the best scheduling results can be obtained normally within 60 minutes. Moreover, as the searching procedure is performed in parallel, the computing time can be easily reduced by using more powerful computers. Therefore, the algorithm is capable for providing satisfactory solutions for media and large-scaled problems within acceptable computing time.

4 SUMMARY

A parallel-machine scheduling problem is considered in this paper, where job-processing times can be compressed by allocating extra resource to the jobs. A tabu-search algorithm is designed to optimize job-processing sequence and processing times, such that the makespan does not exceed deadline K, while the total resource consumption can be minimized. Experiment shows the algorithm is capable for providing satisfactory near optimal solutions for media and large-scaled problems.

ACKNOWLEDGEMENTS

This paper is supported by the National Natural Science Foundation of China (No.61203183), and by CERNET Innovation Project (NGII20161201).

Table 1: Experimental results for problem instances generated with $[a,b] = [10,80]$, MaxPred = 6 and TreeDepth = 8.

n	m = 10					m = 25				
	Init	Rst	Imp	Rst Time	Total Time	Init	Rst	Imp	Rst Time	Total Time
51	1474.4	1314.8	89.2%	00:09:21	00:23:48	1179.8	991.0	83.9%	00:12:24	00:29:17
64	1808.2	1641.8	90.8%	00:11:54	00:35:12	1500.8	1375.5	91.7%	00:09:42	00:27:32
71	2073.4	1730.9	83.4%	00:11:06	00:43:21	1808.2	1616.4	89.4%	00:13:57	00:31:48
78	2152.6	1868.4	86.9%	00:14:13	00:36:04	1956.5	1774.5	90.7%	00:13:42	00:44:40
82	2301.8	1976.5	85.9%	00:13:43	00:42:19	2005.2	1758.6	87.7%	00:10:57	00:26:32
88	2475.6	2153.2	87.0%	00:16:43	00:45:19	2232.3	1959.9	87.8%	00:14:02	00:35:19
93	2481.2	2064.1	83.2%	00:17:23	00:41:14	2132.5	1937.8	90.9%	00:16:44	00:32:47
114	3415.5	3104.7	90.8%	00:20:41	00:53:34	3363.1	2845.1	84.6%	00:15:50	00:38:10
126	3865.7	3486.2	90.2%	00:23:57	00:57:22	3543.4	3029.6	85.5%	00:17:53	00:41:47
130	3644.3	3305.1	90.7%	00:19:31	00:54:22	3513.1	2954.4	84.1%	00:19:26	00:37:53
149	4715.5	3965.4	84.1%	00:19:15	00:58:10	3965.7	3429.7	86.5%	00:17:23	00:36:24
155	4455.8	3697.6	82.9%	00:31:12	01:05:24	4134.7	3596.6	87.0%	00:29:12	00:51:21
166	5158.4	4306.9	83.5%	00:33:52	01:04:13	4814.3	4259.5	88.5%	00:24:26	00:49:31
196	6043.5	5360.1	88.7%	00:42:25	01:23:04	5499.1	4844.6	88.1%	00:37:10	00:57:23
217	6401.7	5472.8	85.5%	00:42:35	01:35:43	5985.0	5081.3	84.9%	00:39:21	01:04:31
233	7024.6	6089.8	86.7%	00:57:51	01:54:53	6712.4	5752.2	85.7%	00:53:54	01:23:19
254	7485.8	6541.9	87.4%	00:55:26	01:56:27	6903.6	5867.6	85.0%	00:48:45	01:21:28

REFERENCES

Graham. R.L., Lawler. E.L., Lenstra. J.K., Rinnooy. A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, 5: 287-326.

Vickson., 1980. Two single machine sequencing problems involving controllable job processing times. AIIE Transactions, 12(3): 258-262.

Cheng. T.C.E, Kovalyov. M.Y., Shakhlevich. N., 2006. Scheduling with controllable release dates and processing times: total completion time minimization. European Journals of Operations Research, 175: 769-781.

Janiak. A., 1991. Single machine scheduling problem with common deadline and resource dependent release dates. European Journal of Operational Research, 53: 317-325.

Shabtay. D., Kaspi. M., 2004. Minimizing the total weighted flow time in a single machine with controllable processing times. Computers and Operations Research, 31(13): 2279-2280.

Shabtay. D., Kaspi. M., 2006. Parallel machine scheduling with a convex resource consumption function. European Journal of Operational Research, 173(1): 92-107.

Shabtay. D., Steiner. G., 2007. A survey of scheduling with controllable processing times. Discrete Applied Mathematics, 155(13): 1643-1666.

Alidaee. B., Ahmadian. A., 1993. Two parallel machine sequencing problems involving controllable job processing times. European Journal of Operations Research, 70: 335-341.

Jansen. K., Mastrolilli. M., 2004. Approximation schemes for parallel machine scheduling problem with controllable processing times. Computers and Operations Research, 31: 1565-1581.

Xu. K.L., Feng. Z.R., Ke. L.J., 2010, A tabu-search algorithm for scheduling jobs with controllable processing times on a single machine to meet due-dates. Computers and Operations Research, 37: 1924-1938.

Glover. F., 1990. Tabu search: a tutorial. Interfaces, 20(4): 665-679.

Dell'Amico. M., Trubian. M., Applying tabu-search to the job shop scheduling problems. Annals of Operations Research, 22: 231-252.

Bilge. U., Kirac. F., Kurtulan. M., Pekgun. P., 2004. A tabu search algorithm for parallel machine total tardiness problem. Computers and Operations Research, 31(3): 397-414.

Venditti. L., Pacciarelli. D., Meloni. C., 2010. A tabu search algorithm for scheduling pharmaceutical packaging operations. European Journal of Operations Research, 202(2): 538-546.

Nowicki., 1995. A bicriterion approach to preemptive scheduling of parallel machines with controllable job processing times. Discrete Appl. Math, 63: 23-256.

Fulkerson. D.R., 1961. A network flow computation for project cost curves. Management Science, 167-178.

Pinedo. M., 2012. *Scheduling: Theory, Algorithms, and Systems*. Springer-Verlag. New York, 4th edition.