A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints

Chunfeng Liu
School of Management, Hefei University of Technology, Hefei, China
Tourism College of Zhejiang, Hangzhou, China
Email: lcf_spring@163.com

Shanlin Yang
School of Management, Hefei University of Technology, Hefei, China
Email: hgdysl@gmail.com

Abstract— **The paper presents a priority rule-based heuristic serial schedule (SS) algorithm for a deterministic scheduling problem where multiple jobs with arbitrary precedence constraints are processed on multiple unrelated parallel machines. The objective is to minimise makespan. The priority rule employs the arithmetic mean and deviation of the processing times to determine the prior job-machine pair. Moreover, at each iteration, the algorithm can schedule the prior job on the prior machine as early as possible to prevent certain machines from standing idle by the greatest extent. The proposed algorithm is demonstrated in detail through a test instance. Computational experiments are conducted to show that the new polynomial-time algorithm is effective in reducing makespan and efficient in shortening runtime.**

Index Terms— **unrelated machines, scheduling, precedence constraints, makespan, priority rule, heuristics**

I. INTRODUCTION

In real industry such as production plan, controlling process, job shop scheduling, it often happens that a task can be separated into several subtasks with precedence constraints between them, and then the bid invitation and bidding or other means are adopted to select the best resources (or machines) to undertake each subtask separately, in order to maximise utilization of these resources, improve productivity and reduce overall cost. Usually, precedence constraints can be summarized into two categories. The one is standard precedence constraints, in which one subtask may have to be completed before another subtask can be started. Standard precedence constraints include chains, out-tree, in-tree, forest, special constraints, arbitrary precedence constraints and etc. The other is s-precedence constraints, in which one subtask is constrained to be started no sooner than another subtask to be started.

As in the literature, there are some studies about standard precedence constraints such as follows. For single machine problems of minimising total completion time, a polynomial-time algorithm has been known when the precedence graph is a set of disjoint trees (a forest) [1]. Lawler [2] has proved that the problem with arbitrary precedence constraints is NP-hard. Adolphson and Hu [3] have showed that such a problem is equivalent to the optimal linear ordering problem, and showed that the complexity of Horn's algorithm is $O(n \log n)$. Sidney [4] has suggested the concept of ρ-maximal initial sets, which can be used to decompose the arbitrary precedence constraints. Chudak and Hochbaum [5] have suggested 2-approximation algorithm using a simple and compact linear programming (LP) relaxation.

For multi-machine problems of minimising makespan, Baev et al. [6] have compared four lower bounds (*Hu*, *RJ*, *LC* and *BR*), and showed that *LC* computing in $O(n^3)$ time can achieve tighter lower bound than others. Chudak and Shmoys [7] have presented a new $O(\log m)$-approximation algorithm for uniform parallel machine problem. Aho and Mäkinen [8] have provided a method to solve parallel machine problem in polynomial-time in the special case where the size of the task precedence graph is bounded by maximum degree and by maximum path length.

For multi-machine problems of minimising total completion time, the problem on identical parallel machines is strongly NP-hard even when the precedence graph is a set of chains [9]. Chang and Hsu [10] have proposed a scheduling approach combining searching algorithm of artifical intelligence (AI) with branch-and-bound approach in order to efficiently find an optimal schedule in a search tree for arbitrary precedence graph.

For multi-machine problems of minimising total weighted completion time, Ramachandra and Elmaghraby [11] have offered a binary integer program (BIP) and a dynamic program (DP) to solve two machine problem. They have also introduced a genetic algorithm (GA) procedure that is capable of solving any problem size. Queyranne and Schulz [12] have presented a 4-approximation algorithm for the parallel machine problem with precedence delays. In that problem each precedence constraint is associated with a certain amount of

doi:10.4304/jsw.6.6.1146-1153

time that must elapse between the completion and start times of the corresponding jobs. Chudak and Shmoys [7] have presented a new $O(\log m)$-approximation algorithm for uniform parallel machine problem. Hall et al. [13] have introduced a general framework for designing on-line algorithm in scheduling environments with release dates. The technique yields the first constant performance guarantee for a variety of scheduling models. Queyranne and Sviridenko [14] have considered a general class of multiprocessor shop scheduling problems, preemptive or non-preemptive with job or operation release dates, and presented a general approximation method combining a linear programming relaxation in the operation completion times, with any algorithm for the makespan version of these problems without release dates.

The other studies about s-precedence constraints can be found in recent literature. Kim and Posner [15] have considered the multiple identical parallel machine scheduling problem of minimising makespan subject to s-precedence constraints, and introduced a list scheduling heuristic that schedules the job maximising the sum of processing times of its successors along any path whenever a machine becomes available. Kim et al. [16] have also considered the parallel machine problem of minimising total completion time, and formulated it as a LP problem with preemption allowed. To solve the LP problem efficiently, they have developed a cutting plane approach in which a pseudo-polynomial dynamic programming algorithm is derived to solve the involved separation problem.

As mentioned above, there has been no study yet on the unrelated parallel machine scheduling problem subject to arbitrary precedence constraints. Such a problem typically occurs in an office or project management environment, where machines are workers who have different skills for office scheduling problem, and machines are types of resources which are allocated to activities for multi-mode project scheduling problem. To the best of our knowledge, it is only Herrmann et al. [17] who have considered the special case of unrelated parallel machine scheduling problem with chains. Motivated from the facts, we propose a heuristic serial schedule method (SS algorithm) for the unrelated parallel machine scheduling problem of minimising makespan subject to arbitrary precedence constraints.

The remainder is organized as follows. The problem is presented in Section II. In Section III, we propose a priority rule-based heuristic serial schedule (SS) algorithm. In Section IV, the SS algorithm is demonstrated with a test instance. In Section V, several numerical experiments are conducted to evaluate the performance of the proposed algorithm. Finally, the paper closes with a general discussion of the proposed approach as well as a few remarks on research perspectives in Section VI.

II. Problem Formulation

We consider the following scheduling problem P. A set $J = \{1, \ldots, n\}$ of n jobs has to be processed on m unrelated parallel machines $M = \{1, \ldots, m\}$. Each machine can process at most one job at a time. Each job is processed on only one machine and non-preemptive during the processing period. Each job j has a positive processing time p_{jv} when processed on machine v. There are arbitrary standard precedence constraints between jobs. The constraints force a job not to be started before all its predecessors are completed. The objective is to find a feasible schedule that minimises makespan $C_{\max} = \max_{j \in J} FT_j$. Here, FT_j denotes the completion time of job j. In standard scheduling notation [18], this problem P is known as $Rm|prec|C_{\max}$, where R denotes unrelated parallel machines, and $prec$ denotes arbitrary standard precedence constraints. Ullman [19] has proved the parallel machine problem $P|p_i = 1; prec|C_{\max}$ with unit processing time is NP-hard. Consequently, the problem P is also NP-hard evidently.

The problem can be represented as a mathematical formulation as follows [20]:

$$\text{Min} \quad C_{\max} = \max_{j \in J} FT_j \tag{1}$$

$$\text{st.} \quad \sum_{v=1}^{m} \sum_{r=1}^{UB} x_{jvr} = 1, \forall j \in J \tag{2}$$

$$\sum_{j=1}^{n} x_{jvr} \leq 1, \forall r \in R, \forall v \in M \tag{3}$$

$$\sum_{i=1}^{n} x_{ivr} - \sum_{j=1}^{n} x_{j,v,r-1} \leq 0, \quad \forall v \in M, \forall r \in \{2, \ldots, UB\} \tag{4}$$

$$FT_j - FT_i + L(2 - x_{jvr} - x_{i,v,r-1}) \geq p_{jv}, \quad \forall i, j \in J, i \neq j, \forall v \in M, \forall r \in \{2, \ldots, UB\} \tag{5}$$

$$FT_j \geq \sum_{r=1}^{UB} p_{jv} x_{jvr}, \forall j \in J, \forall v \in M \tag{6}$$

$$FT_j - FT_i \geq \sum_{v=1}^{m} \sum_{r=1}^{UB} p_{jv} x_{jvr}, \forall i \in P_j \tag{7}$$

$$x_{jvr} \in \{0, 1\}, FT_j \geq 0, \forall j \in J, \forall v \in M, \forall r \in R \tag{8}$$

The input parameters of the model include:

J	the job set, $J = \{1, \ldots, n\}$.
M	the machine set, $M = \{1, \ldots, m\}$.
UB	the maximum number of positions on each machine that jobs are placed on them. It is computed as follows: $UB = n - m + 1$ (i.e., the maximum machine utilization is met, so that all machines are used).
R	the position set, $R = \{1, \ldots, UB\}$.
p_{jv}	the processing time of job j on machine v.
P_j	the set of immediate predecessors of job j.
L	A large positive number.

The decision variables of the model include:

x_{jvr}	equals 1 if job j is processed in the position r on machine v and 0 otherwise.
FT_j	the completion time of job j.

The objective function (1) minimises the the makespan. Constraints (2) ensure that each job is assigned to one of the existing positions on the machines. Constraints (3) guarantee that at most one job can be assigned to each position. Constraints (4) ensure that until one position on a machine is occupied, jobs are not assigned to subsequent positions. Constraints (5) ensure that the completion time of a job in sequence on a machine is at least equal to the sum of the completion time of the preceding job and the processing time of the present job. Constraints (6) measure completion time for each job on each machine. Constraints (7) observe precedence relationships. Constraints (8) define the type of decision variables.

III. Priority Rule-based Heuristic Serial Schedule (SS) Algorithm

The SS algorithm consists of n iterations. At each iteration, an eligible job is selected according to its priority and inserted inside a partial schedule on the earliest eligible machine (respecting the precedence constraints), while keeping unchanged the start time of the already scheduled jobs. A job is eligible if all its predecessors have already been scheduled. A machine is eligible if the machine is idle from the start of schedule time point t.

A. priority rule

In this subsection, we first introduce a job-on-node (JON) network to represent the entire schedule. The nodes of this graph represent the jobs. Arc (i, j) means that job i must be completed before job j can be started. In addition, The dummy jobs $s = 0$ and $e = n+1$ correspond to the start and end of schedule respectively such that $\forall v \in M, p_{sv} = p_{ev} = 0$.

Then, three job-sets and one machine-set associated with each iteration are defined. Jobs which are completed up to the schedule time are in the completion job-set C_u. Jobs which are already scheduled are in the total scheduled job-set H_u. Jobs which are available for scheduling w.r.t. precedence constraints but yet unscheduled are in the decision job-set D_u. Finally, machines which are idle from the start of schedule time point t are in the idle machine-set W_u. A priority rule is used to select an eligible job of D_u and an eligible machine of W_u, which can be described as follows: if there is only one machine in W_u, the job of D_u having the minimum processing time on the machine is selected; otherwise, the job of D_u having the maximum deviation of the processing times on the machines of W_u is selected, meanwhile, the machine of W_u processing the job having the minimum processing time is selected.

The variables used for the SS algorithm are summarized as follows:

u	the counter of iteration.
H_u	the total scheduled job-set at u iteration.
C_u	the completion job-set at u iteration.
D_u	the decision job-set at u iteration, $D_u = \{j \mid j \notin H_u, P_j \subseteq C_u\}$.
W_u	the idle machine-set at u iteration.
$\lvert W_u \rvert$	the number of elements of set W_u.
I_v	the start idle time of machine v.
(j^*, v^*)	the prior job-machine pair (j^* denoting the selected prior job, and v^* denoting the selected prior machine).
E_j	the arithmetic mean of the processing times of job j on the machines of W_u, $E_j = \frac{1}{\lvert W_u \rvert} \sum_{v \in W_u} p_{jv}, j \in D_u$.
σ_j^2	the deviation of the processing times of job j on the machines of W_u from E_j, $\sigma_j^2 = \frac{1}{\lvert W_u \rvert} \sum_{v \in W_u} (p_{jv} - E_j)^2, j \in D_u$.
t	the schedule time point.

B. SS algorithm

In this subsection, we give a pseudo-code description of the heuristic serial schedule procedure which consists of n iterations (cf. SS algorithm). Step 1 initialises some variables u, t, H_u, C_u, W_u, I_v and computes D_u. In step 2 each job is scheduled at each iteration until $u \leq n$ does not hold. If $|W_u| = 1$ holds, the prior job-machine pair (j^*, v^*) can be directly determined according to the minimum processing time of the jobs $j \in D_u$ in step 2.1.1. Else, the arithmetic mean E_j and the deviation σ_j^2 of the processing times of job j are computed in step 2.1.2, then the job of D_u having the maximum σ_j^2 is selected in step 2.1.3, meanwhile, the machine of W_u having the minimum p_{j^*v} is determined in step 2.1.4. In step 2.2 the start time ST_{j^*} and the completion time FT_{j^*} of job j^* are saved. In step 2.3 u is increased for the next iteration. In step 2.4 the total scheduled job-set H_u and the start idle time I_{v^*} of machine v^* are updated. In step 2.5 the minimum start idle time of the machines is selected as the current schedule time point t. The idle machine-set W_u, the completion job-set C_u and the decision job-set D_u are computed in steps 2.6–2.8. In step 2.9 the schedule time point t is postponed to the next minimum start idle time until D_u is not empty. Finally, the makespan $C_{\max}$ is set equal to the maximum completion time of the jobs $j \in J$. The principle of SS algorithm just outlined is illustrated in Figure 1.

As one can notice, the SS algorithm is a polynomial-time algorithm. Step 2.9 executes n iterations at most for each iteration of step 2. Since the algorithm stops after n iterations when all the jobs have been scheduled, its total time complexity is $O(n^2)$.

IV. Test Instance

In order to test the SS algorithm on the instance that Herrmann et al. [17] have given, we consider a special case of the problem P, which changes the condition of arbitrary precedence constraints to a set of chains and maintains the others unchanged. Let P′ denote the revised

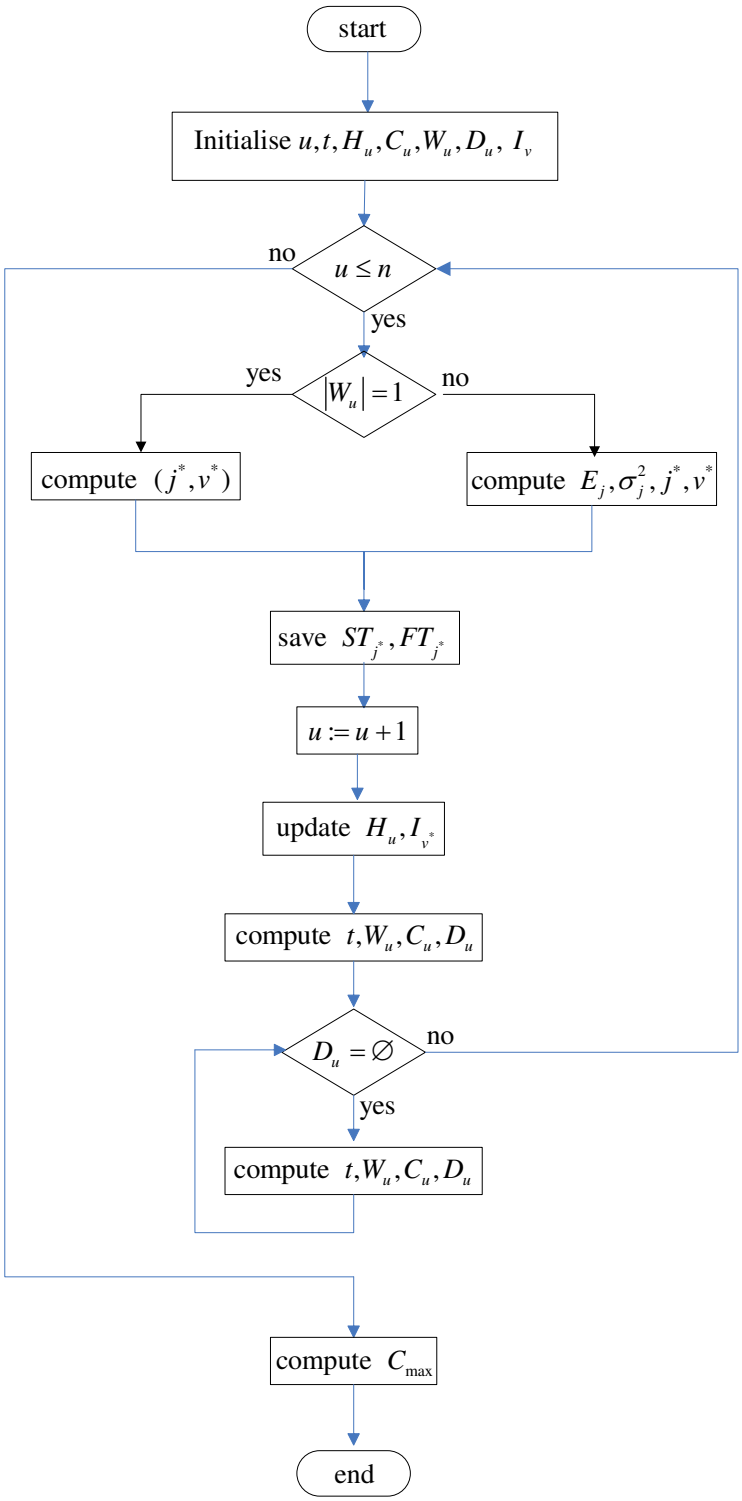

Figure 1. Program flow chart

SS algorithm

1. Initialise: $u = 1, t = 0, H_u = C_u = \{\mathrm{s}\}, W_u = M$, compute D_u;
   $I_v = 0, \forall v \in M$
2. **WHILE** $(u \leq n)$ **DO**
   2.1. **IF** $(|W_u| = 1)$
   2.1.1. $(j^*, v^*) : p_{j^*v^*} = \min\{p_{jv} | j \in D_u, v \in W_u\}$
   ELSE
   2.1.2. $E_j = \frac{1}{|W_u|} \sum_{v \in W_u} p_{jv}, \sigma_j^2 = \frac{1}{|W_u|} \sum_{v \in W_u} (p_{jv} - E_j)^2, \forall j \in D_u$
   2.1.3. $j^* : \sigma_{j^*}^2 = \max\{\sigma_j^2 | j \in D_u\}$
   2.1.4. $v^* : p_{j^*v^*} = \min\{p_{j^*v} | v \in W_u\}$
   2.2. $ST_{j^*} = t, FT_{j^*} = t + p_{j^*v^*}$
   2.3. $u := u + 1$
   2.4. $H_u := H_u \cup \{j^*\}, I_{v^*} = FT_{j^*}$
   2.5. $t = \min\{I_y | y \in M\}$
   2.6. $W_u = \{y | I_y = t, y \in M\}$
   2.7. $C_u = \{j | FT_j \leq t, j \in H_u\}$
   2.8. compute D_u
   2.9. **WHILE** $(D_u = \emptyset)$ **DO**
   2.9.1. $M' = M \backslash W_u$
   2.9.2. $t = \min\{I_y | y \in M'\}$
   2.9.3. $W'_u = \{y | I_y = t, y \in M'\}$
   2.9.4. $W_u := W_u \cup W'_u$
   2.9.5. $C_u = \{j | FT_j \leq t, j \in H_u\}$
   2.9.6. compute D_u
3. $C_{\max} = \max_{j \in J}\{FT_j\}$

TABLE I.
PROCESSING TIMES

	job1	job2	job3	job4	job5	job6	job7
machine1	3	4	8	2	5	9	3
machine2	9	5	2	6	10	4	8

problem. Herrmann et al. have considered the problem P′, and provided a heuristic which assigns the jobs to the machines and defines the schedule simultaneously. We denote it as HH algorithm for convenience. Herrmann et al.'s instance can be described as follows. There exists seven jobs and two unrelated parallel machines. The processing times are given in Table I. Furthermore the following precedence constraints need to be considered: $S_3 = \{7\}$, $S_1 = \{3\}$, $S_2 = \{6\}$.The solution to this problem is reached after seven iterations using the SS algorithm. The results obtained at each iteration are gathered in Table II.

When $u = 1$, the selected prior job $j^* = 1$ is processed on the selected prior machine $v^* = 1$, started at time $ST_{j^*} = 0$, and completed at time $FT_{j^*} = 3$. After increasing u the variables $W_u = \{2\}$, $C_u = \{\mathrm{s}\}$ and $D_u = \{2, 4, 5\}$ are computed in steps 2.6–2.8, which are to be used at the next iteration. The following iterations are similar to the first iteration.

Both the final solution using the SS algorithm and the one using the HH algorithm are presented by Gantt graphs in Figures 2 and 3, respectively. It is easy to observe that the makespan $C_{\max} = 13$ using the SS algorithm is less than the one using the HH algorithm by 2 units.

V. COMPUTATIONAL EXPERIMENTS

The following numerical experiments are conducted to evaluate the performance of the SS algorithm compared to the HH algorithm for the problem P′. The performance is to be evaluated by use of several impact factors including number of jobs (n), number of machines (m), number of constraint chains (*NC*) and processing times (p_{jv}).

To test the effects of varying n, m and *NC*, four different values of n are used, including 30, 60, 90 and 120, four different values of m are used, including 5, 8, 12 and 15, and four different values of *NC* are used, including 50, 60, 70 and 80. Moreover, to determine whether the range of p_{jv} may have any impact on the performance of the SS algorithm, four different distributions of p_{jv} are used, including $p_{jv} \sim \mathrm{DU}[10, 15]$, $p_{jv} \sim \mathrm{DU}[10, 20]$, $p_{jv} \sim \mathrm{DU}[10, 25]$ and $p_{jv} \sim \mathrm{DU}[10, 30]$, where $\mathrm{DU}[a, b]$ represents a discrete uniform distribution with a range from a to b.

Four sets of numerical experiments are conducted. In the first set, n is allowed to vary, given $m = 5$, $NC = 25$ and $p_{jv} \sim \mathrm{DU}[10, 15]$. In the second set, m is allowed to vary, given $n = 90$, $NC = 80$ and $p_{jv} \sim \mathrm{DU}[10, 15]$. In the third set, *NC* is allowed to vary, given $n = 90$, $m = 5$ and $p_{jv} \sim \mathrm{DU}[10, 15]$. In the fourth set, the distribution of generating p_{jv} is allowed to vary, given $n = 90$, $m = 5$ and $NC = 80$. The four experiment results are presented in Tables III–VI, respectively.

As the performance measures, the average relative error $hhGap$ of the HH algorithm is used which is defined as $100 * (C_{hh} - LB) / LB$, and the average relative error $ssGap$ of the SS algorithm is used which is defined as $100 * (C_{ss} - LB) / LB$, where C_{hh}, C_{ss} and LB represent the solution value of the HH algorithm, the one of the SS algorithm and a lower bound of the problem P′, respectively. To allow other researchers to compare their results with those reported here, we chose an easy lower

TABLE II.
SUMMARY OF ITERATIONS

u	j^*	v^*	ST_{j^*}	FT_{j^*}	u	W_u	C_u	D_u
1	1	1	0	3	2	{2}	{s}	{2,4,5}
2	2	2	0	5	3	{1}	{s,1}	{3,4,5}
3	4	1	3	5	4	{1,2}	{s,1,2,4}	{3,5,6}
4	3	2	5	7	5	{1}	{s,1,2,4}	{5,6}
5	5	1	5	10	6	{2}	{s,1,2,4,3}	{6,7}
6	6	2	7	11	7	{1}	{s,1,2,4,3,5}	{7}
7	7	1	10	13	8	{1,2}	{s,1,2,4,3,5,6,7}	{e}

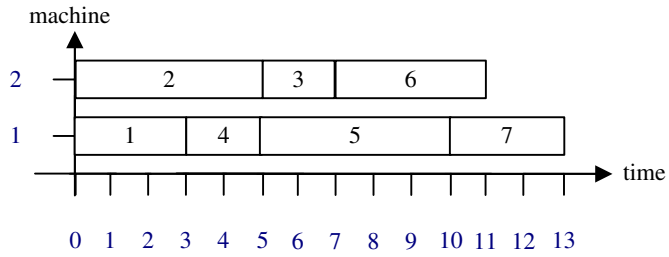

Figure 2. Gantt chart using SS algorithm

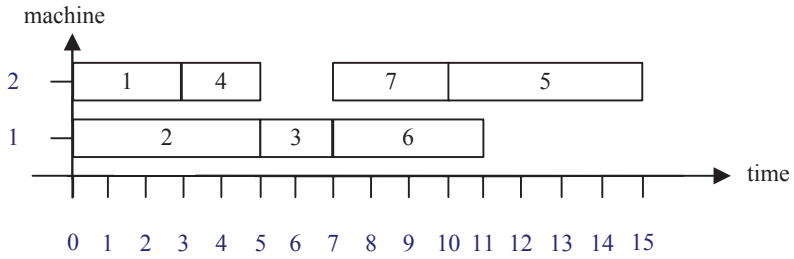

Figure 3. Gantt chart using HH algorithm

TABLE III.
PERFORMANCE COMPARISON BETWEEN THE HH AND SS ALGORITHMS FOR DIFFERENT NUMBER OF JOBS (n)

$m = 5$, $NC = 25$, $p_{jv} \sim \mathrm{DU}[10, 15]$		*hhGap* (%)			*ssGap* (%)			*decrease-AveGap* (%)	*hhCPU* (s)	*ssCPU* (s)	*decrease-CPU* (%)
		MIN	MAX	AVE	MIN	MAX	AVE				
n	30	14.06	34.49	23.39	4.69	38.63	16.17	30.9	0.03	0.02	41.89
	60	15.14	25.80	20.78	8.42	24.80	14.94	28.09	0.15	0.08	48.89
	90	16.51	26.33	22.47	10.53	49.47	20.81	7.42	0.43	0.24	44.53
	120	17.20	25.49	22.18	6.53	45.33	18.29	17.54	0.84	0.47	43.75

bound:

$$LB = \frac{\sum_{j=1}^{n} \min_{v \in \{1,\ldots,m\}} p_{jv}}{m}. \quad (9)$$

Because the lower bound LB is not greater than the optimal solution, the relative distance between the solution of the SS (HH) algorithm and the optimal solution is not greater than $ssGap$ ($hhGap$). Each table entry represents the minimum, maximum and average of its associated 10 instances. Let $decreaseAveGap$ denote the percentage that $ssGap$ is less than $hhGap$. Let $ssCPU$ ($hhCPU$) denote the CPU time of the SS (HH) algorithm without including input and output time. Let $decreaseCPU$ denote the percentage that $ssCPU$ is less than $hhCPU$.

The experiments have been performed on a Pentium-based Lenovo-compatible personal computer with 2.40 GHz clock-pulse and 496 MB RAM. The HH and SS algorithms have been coded in C++, compiled with the Microsoft Visual C++ 6 compiler, and tested under Microsoft Windows XP Professional SP3.

It can be observed from Tables III and IV that, there are no strong correlations between the average of the average relative error ($ssGap$) and the number of jobs (n), but the average $ssGap$ decreases as m decreases. This may

TABLE IV.
PERFORMANCE COMPARISON BETWEEN THE HH AND SS ALGORITHMS FOR DIFFERENT NUMBER OF MACHINES (m)

$n = 90$, $NC = 80$, $p_{jv} \sim \mathrm{DU}[10, 15]$		*hhGap* (%)			*ssGap* (%)			*decrease-AveGap* (%)	*hhCPU* (s)	*ssCPU* (s)	*decrease-CPU* (%)
		MIN	MAX	AVE	MIN	MAX	AVE				
m	5	22.89	30.64	27.61	1.596	7.03	4.54	83.55	0.37	0.21	43.56
	8	23.54	33.05	28.19	6.90	17.49	9.33	66.92	0.44	0.22	50.50
	12	18.81	25.69	23.15	8.97	23.14	14.85	35.84	0.52	0.23	56.26
	15	22.52	27.34	24.76	4.30	33.81	18.96	23.43	0.59	0.20	66.19

TABLE V.
PERFORMANCE COMPARISON BETWEEN THE HH AND SS ALGORITHMS FOR DIFFERENT NUMBER OF CONSTRAINT CHAINS (NC)

$n = 90$, $m = 5$, $p_{jv} \sim \mathrm{DU}[10, 15]$		*hhGap* (%)			*ssGap* (%)			*decrease-AveGap* (%)	*hhCPU* (s)	*ssCPU* (s)	*decrease-CPU* (%)
		MIN	MAX	AVE	MIN	MAX	AVE				
NC	50	22.65	29.07	25.42	3.36	21.07	11.16	56.11	0.38	0.21	45.73
	60	21.34	28.36	25.75	4.43	17.07	7.52	70.81	0.37	0.21	42.69
	70	23.02	33.37	26.95	1.37	13.47	5.29	80.38	0.37	0.22	40.76
	80	25.13	30.04	28.43	2.86	7.56	5.02	82.35	0.37	0.22	40.93

TABLE VI.
PERFORMANCE COMPARISON BETWEEN THE HH AND SS ALGORITHMS FOR DIFFERENT PROCESSING TIMES (p_{jv})

$n = 90$, $m = 5$, $NC = 80$		*hhGap* (%)			*ssGap* (%)			*decrease-AveGap* (%)	*hhCPU* (s)	*ssCPU* (s)	*decrease-CPU* (%)
		MIN	MAX	AVE	MIN	MAX	AVE				
p_{jv}	$\mathrm{DU}[10, 15]$	24.34	30.83	27.33	1.35	7.33	4.30	84.27	0.38	0.22	42.12
	$\mathrm{DU}[10, 20]$	35.56	48.35	40.69	3.16	12.56	6.40	84.27	0.37	0.21	43.02
	$\mathrm{DU}[10, 25]$	44.69	55.87	49.00	3.84	15.63	7.43	84.84	0.38	0.21	45.25
	$\mathrm{DU}[10, 30]$	42.56	65.81	57.30	4.97	14.50	7.91	86.19	0.39	0.21	46.44

due to the fact that a job is easier to be assigned on certain machine as m decreases, which results in a shorter makespan.

In Table V, the average of the average relative error $(ssGap)$ decreases as the number of constraint chains (NC) increases. Because the jobs become weaker in precedence constraints and easier to be scheduled as the number of constraint chains increases, which also results in a shorter makespan.

In Table VI, the average of the average relative error $(ssGap)$ increases slowly as the range of processing times (p_{jv}) increases. Since the lower bound of the problem P′ relaxes the processing time of each job j to the minimum processing time $\min_{v\in\{1,\ldots,m\}} p_{jv}$, LB gives a slacker lower bound as the range of processing times (p_{jv}) increases, which brings about a greater average $ssGap$.

Moreover, referring to Tables III–VI, it can be concluded that the average $ssGap$ is less than the average $hhGap$ by 7%–87% (see $decreaseAveGap$). The reason why the SS algorithm gets better solution than the HH algorithm can be explained as follows. First, the SS algorithm computes the idle machine-set W_u having the minimum of start idle time in steps 2.6 and 2.9.4, which can prevent certain machines from standing idle by the greatest extent in the next iteration of scheduling and lead to a shorter makespan. However, it can be observed from the above test instance that the HH algorithm is easy to make certain machine idle and results in a longer makespan. Second, the SS algorithm employs heuristic priority rule when selecting the prior job and machine in step 2.1. The job having the largest range of processing times is processed in the minimum processing time, which shortens the makespan. In addition, the average $ssCPU$ is less than the average $hhCPU$ by 40%–67% (see $decreaseCPU$). Therefore the SS algorithm can run more quickly than the HH algorithm.

VI. CONCLUSIONS

This paper considers an unrelated parallel machine scheduling problem of minimising makespan subject to arbitrary precedence constraints. A priority rule-based heuristic serial schedule (SS) algorithm is proposed for the NP-hard problem. The priority rule can select the prior job and machine at each iteration. The algorithm can schedule the jobs on the idle machines as early as possible.

In order to evaluate the effectiveness and efficiency of the proposed algorithm, four sets of numerical experi-

ments are conducted. Experiment results show that the proposed SS algorithm performs better as the number of machines (m) decreases. The SS algorithm gets better solution than the HH algorithm. Moreover, the SS algorithm is a polynomial-time algorithm that can run in $O(n^2)$, and consumes shorter CPU time than the HH algorithm. So it can be embedded in more sophisticated heuristics or metaheuristics for determining initial feasible schedules that can be improved in further stages, or for computing effective upper bounds which allow to cut parts of search trees during branch-and-bound procedures.

As a further study, it would be interesting to consider various priority rules as heuristic information, and it also seems to be worthwhile to extend the problem to an unrelated parallel machine scheduling problem subject to s-precedence constraints.

ACKNOWLEDGMENT

This research is supported by National Natural Science foundation of China under grant number 70631003, and in part by National High Technology Research and Development Program 863 of China under grant number 2008AA042901.

REFERENCES

[1] W. A. Horn, "Single-machine job sequencing with treelike precedence ordering and linear delay penalties," *SIAM Journal on Applied Mathematics*, vol. 23, pp. 189–202, 1972.

[2] E. L. Lawler, "Sequencing jobs to minimize total weighted completion time subject to precedence constraints," *Annals of Discrete Mathematics*, vol. 2, pp. 75–90, 1978.

[3] D. Adolphson and T. C. Hu, "Optimal linear ordering," *SIAM Journal on Applied Mathematics*, vol. 25, pp. 403–423, 1973.

[4] J. B. Sidney, "Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs," *Operations Research*, vol. 23, pp. 283–298, 1975.

[5] F. A. Chudak and D. S. Hochbaum, "A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine," *Operations Research Letters*, vol. 25, pp. 199–204, 1999.

[6] I. D. Baev, W. M. Meleis, and A. Eichenberger, "Lower bounds on precedence-constrained scheduling for parallel processors," *Information Processing Letters*, vol. 83, pp. 27–32, 2002.

[7] F. A. Chudak and D. B. Shmoys, "Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds," *Journal of Algorithms*, vol. 30, pp. 323–343, 1999.

[8] I. Aho and E. Mäkinen, "On a parallel machine scheduling problem with precedence constraints," *Journal of Scheduling*, vol. 9, pp. 493–495, 2006.

[9] J. Du, J. Y. T. Leung, and G. H. Young, "Scheduling chain-structured tasks to minimize makespan and mean flow time," *Information and Computation*, vol. 92, pp. 219–236, 1991.

[10] J. M. Chang and C. C. Hsu, "Task scheduling with precedence constraints to minimize the total completion time," *International Journal Systems Science*, vol. 26, pp. 2203–2217, 1995.

[11] G. Ramachandra and S. E. Elmaghraby, "Sequencing precedence-related jobs on two machines to minimize the weighted completion time," *International Journal of Production Economics*, vol. 100, pp. 44–58, 2006.

[12] M. Queyranne and A. S. Schulz, "Approximation bounds for a general class of precedence constrained parallel machine scheduling problems," *SIAM Journal on Computing*, vol. 35, no. 5, pp. 1241–1253, 2006.

[13] L. A. Hall, A. S. Schulz, and D. B. Shmoys, "Scheduling to minimize average completion time: offline and online algorithms," *Mathematics of Operations Research*, vol. 22, pp. 513–544, 1997.

[14] M. Queyranne and M. Sviridenko, "Approximation algorithms for shop scheduling problems with minsum objective," *Journal of Scheduling*, vol. 5, pp. 287–305, 2002.

[15] E. S. Kim and M. E. Posner, "Parallel machine scheduling with s-precedence constraints," *IIE Transactions*, vol. 42, pp. 525–537, 2010.

[16] E. S. Kim, C. S. Sung, and I. S. Lee, "Scheduling of parallel machines to minimize total completion time subject to s-precedence constraints," *Computers & Operations Research*, vol. 36, pp. 698–710, 2009.

[17] J. Herrmann, J. M. Proth, and N. Sauer, "Heuristics for unrelated machine scheduling with precedence constraints," *European Journal of Operational Research*, vol. 102, pp. 528–537, 1997.

[18] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[19] J. D. Ullman, "NP-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, pp. 384–393, 1975.

[20] R. Tavakkoli-Moghaddam, F. Taheri, M. Bazzazi, M. Izadi, and F. Sassani, "Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints," *Computers & Operations Research*, vol. 36, pp. 3224–3230, 2009.

Chunfeng Liu was born in 1977. He received his B.S. degree from School of Mechanical Engineering of Nanjing University of chemical industry in 1999, and M.S. degree from School of Management of Zhejiang University in 2004.

He serves as an associate professor at Tourism College of Zhejiang China. In addition, he is currently a doctoral student at School of Management of Hefei University of Technology. His major research interests are production planning and scheduling of discrete manufacturing systems.

Shanlin Yang was born in 1948. He received his B.S. and M.S. degrees from School of Computer and Information Engineering of Hefei University of Technology in 1982 and 1985, respectively.

He has been a professor and Ph.D. advisor at Hefei University of Technology since 1997, where he has served successively as a dean of School of Management, vice president, director of Institution of Computer Network Systems since 1994. He conducted cooperating research at the University of Melbourne, Australia, from 1986 to 1987. His research interests include information management, decision support systems, scheduling, supply chain management and so on. He has chaired more than 30 projects from National Natural Science Foundation of China, National High Technology Research and Development Program 863 of China and so on. Moreover, he has won the second prize of China National Prize for Progress in Science and Technology.