American Journal of Applied Sciences 4 (11): 887-895 2007
ISSN 1546-9239
© 2007 Science Publications

Three Algorithms for Flexible Flow-shop Scheduling

[1]Tzung-Pei Hong, [2]Pei-Ying Huang, [3]Gwoboa Horng and [3]Chan-Lon Wang
[1]Department of Computer Science and Information Engineering,
National University of Kaohsiung, Kaohsiung 811, Taiwan, R.O.C.
[2]Department of Computer Science and Information Engineering
National Taiwan University, Taipei 106, Taiwan, R.O.C.
[3]Department of Computer Science
National Chung-Hsing University, Taichung 402, Taiwan, R.O.C.

Abstract: Scheduling is an important process widely used in manufacturing, production, management, computer science, and so on. Appropriate scheduling can reduce material handling costs and time. Finding good schedules for given sets of jobs can thus help factory supervisors effectively control job flows and provide solutions for job sequencing. In simple flow shop problems, each machine operation center includes just one machine. If at least one machine center includes more than one machine, the scheduling problem becomes a flexible flow-shop problem. Flexible flow shops are thus generalization of simple flow shops. In this paper, we propose three algorithms to solve flexible flow-shop problems of more than two machine centers. The first one extends Sriskandarajah and Sethi's method by combining both the LPT and the search-and-prune approaches to get a nearly optimal makespan. It is suitable for a medium-sized number of jobs. The second one is an optimal algorithm, entirely using the search-and-prune technique. It can work only when the job number is small. The third one is similar to the first one, except that it uses Petrov's approach (PT) to deal with job sequencing instead of search-and-prune. It can get a polynomial time complexity, thus being more suitable for real applications than the other two. Experiments are also made to compare the three proposed algorithms. A trade-off can thus be achieved between accuracy and time complexity.

Key words: scheduling, flexible flow shop, LPT scheduling, search, PT scheduling

INTRODUCTION

Scheduling is an important process widely used in manufacturing, production, management, computer science, and so on. In simple flow-shop problems, each machine center has just one machine[1,3,4,9-11]. If at least one machine center has more than one machine, the problem is called a flexible flow-shop problem. Flexible flow shops are thus generalization of simple flow shops[2]. Scheduling jobs in flexible flow shops is considered an NP-hard problem[8,12].

The problem addressed in the paper is a special case of the flexible flow shop problem. We assume each machine center has the same number of parallel machines which to the best of authors' knowledge is the first of its kind. This paper specifically focuses on minimizing the total completion time of flexible flow shop. Three algorithms have been developed to solve flexible flow-shop scheduling problems with more than two machine centers. The first one extends Sriskandarajah and Sethi's method by combining both the LPT[5] and the search-and-prune approaches to get a nearly optimal makespan. The LPT approach is first used to assign jobs to each machine group (flow shop). The search-and-prune approach is then used to deal with job sequencing. The second one is an optimal algorithm, entirely using the search-and-prune technique. The third one is similar to the first one except that it uses Petrov's approach (PT)[11] to deal with job sequencing instead of search-and-prune. Experimental results show that the third proposed algorithm can save much computational time when compared to the other two although its makespans may be a little larger. Particularly, the third one has the polynomial time complexity, avoiding the intractable problems occurring in the other two algorithms. In addition, the time complexities and makespans by the first algorithm lie between those by the other two. A trade-off for these three algorithms can thus be achieved between accuracy and time complexity.

In the past, Johnson first proposed an efficient algorithm which guaranteed optimality in a two-

Corresponding Author: Tzung-Pei Hong, Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan, R.O.C.

machine flow-shop problem[6]. Palmer, Petrov and Gupta then respectively proposed their algorithms for solving the flow-shop problems of more than two machines[4,10,11]. The three scheduling algorithms could process the job data in only one pass. Campbell, Dudek and Smith (CDS) then proposed a heuristic algorithm for achieving the same purpose[1]. It, however, needed to process the job data in multiple passes. Logendran and Nudtasomboon also proposed a multi-pass algorithm to solve it[7]. Sriskandarajah and Sethi then presented a heuristic algorithm based on the Johnson algorithm for solving flexible flow-shop problems of two machine centers with the same number of machines[12]. Many researches in this field are still in progress.

As mentioned above, flexible flow-shop problems are NP-hard. No algorithms can find the optimal solutions in polynomial time. In the past, Sriskandarajah and Sethi proposed a heuristic algorithm to solve the problem of two machine centers, and the completion time of the derived schedules was close to the optimum. In this paper, we generalize it and propose three algorithms to solve the flexible flow-shop problems of more than two machine centers. Some related scheduling algorithms are first introduced as follows.

The discovery of scheduling algorithms for a set of independent tasks with arbitrary execution time and an arbitrary number of processors is a classic sequencing problem of wide interest and application. Among the proposed scheduling algorithms, the LPT (Longest-Processing-Time-first) scheduling algorithm is the simplest one and is widely used in many real-world situations.

Given a set of n independent tasks (T_1 to T_n), each with arbitrary execution time (t_1 to t_n), and a set of m parallel processors or machines (P_1 to P_m), the LPT scheduling algorithm assigns the task with the longest execution time (among those not yet assigned) to a free processor whenever this processor becomes free. For cases when there is a tie, an arbitrary tie-breaking rule can be assumed. The finishing time by the LPT scheduling algorithm is in general not minimal. The computational time spent by the LPT scheduling algorithm is, however, much lower than that by an optimal scheduling algorithm.

The PT algorithm[11] was proposed by Petrov to schedule job sequencing for a flow shop with more than two machines. Given a set of n flow-shop jobs, each having m (m>2) tasks (T_{11}, T_{21}, … , T_{m1}, T_{12}, T_{22}, …, $T_{(m-1)n}$, T_{mn}) that must be executed in the same sequence on m machines (P_1, P_2, …, P_m), the PT scheduling algorithm seeks a nearly minimum completion time of the last job. It transforms the flow shop problems with more than two tasks into the ones with exactly two tasks and uses the Johnson algorithm to solve them.

Sriskandarajah and Sethi[12] proposed a heuristic algorithm for solving the flexible flow-shop problem of two machine centers and the completion time of the derived schedules was close to the optimum. Sriskandarajah and Sethi decomposed the problem into the following three subproblems and solved each heuristically:

Part 1: Form the machine groups, each of which contains a machine from each center.
Part 2: Use the LPT method to assign jobs to each machine group (flow shop).
Part 3: Deal with job sequencing and timing using the Johnson algorithm.

In this paper, we will extend above approaches to solve the flexible flow-shop problems of more than two machine centers.

Assumptions and Notation: Assumptions and notation used in this paper are described in this section.

Assumptions:

1. Jobs are not preemptive
2. Each job has m (m > 2) tasks with processing times, executed respectively on each of m machine centers.
3. All machine centers have the same number of parallel machines.

The Search-And-Prune Scheduling Procedure For Job Sequencing: The search-and-prune procedure proposed in this paper is used to schedule jobs sequencing for a flow shop with more than two machines. An upper bound is used to increase the performance of the procedure. The procedure will act as the third part in the first two algorithms proposed later. Given a set of n flow-shop jobs, each having m (m>2) tasks (T_{11}, T_{21}, …, T_{m1}, T_{12}, T_{22}, …, $T_{(m-1)n}$, T_{mn}) that must be executed in the same sequence on m machines (P_1, P_2, …, P_m), scheduling seeks the minimum completion time of the last job. The procedure is stated as follows.

The search-and-prune procedure with an upper bound for job sequencing:

Input: A set of n jobs, each having m (m > 2) tasks executed respectively on each of the given m machines

Output: A schedule with a minimum completion time of the last job.

Step 1: Set the initial upper bound $v_{\max}$ of the final completion time as ∞.
Step 2: For each possible permutation of task sequence, do the following steps.
Step 3: Set the initial completion time d_i of the machine M_i (i = 1 to m, m is the number of tasks in a job) to zero.
Step 4: Assign the first job J_j in its schedule sequence generated in Step 2 to the machines such that J_j 's first task T_{1j} is assigned to M_1, T_{2j} is assigned to M_2, …, T_{mj} is assigned to M_m.
Step 5: Add the processing time t_{1j} to the completion time d_1 of the first machine M_1; that is:

$$d_1 = d_1 + t_{1j}.$$

Step 6: If d_1 is larger than $v_{\max}$, go to Step 2 for trying another permutation.
Step 7: Set $d_{k+1} = \max(d_k, d_{k+1}) + t_{(k+1)j}$, for k = 1 to (m-1).
Step 8: If d_{k+1} is larger than $v_{\max}$, go to Step 2 for trying another permutation; otherwise, do the next step.
Step 9: Remove job J_j from the sequence.
Step 10: Repeat Step 4 to 9 until the job sequence is empty.
Step 11: Set the completion time d as the completion time d_m of its m-th machine.
Step 12: If d is smaller than $v_{\max}$, then set $v_{\max} = d$.
Step 13: Repeat Step 2 to Step 12 until all possible permutations have been tested.
Step 14: Set $v_{\max}$ as the final completion time of the job scheduling and save the schedule that gives the minimum total completion time.

After Step 14, scheduling is finished and an optimal completion time for a flow shop has been found.

The first algorithm for scheduling on a flexible flow shop with more than two machine centers: A heuristic algorithm for solving flexible flow-shop problems of two machine centers is proposed by Sriskandarajah and Sethi in 1989[12]. In this paper, we generalize it to solve flexible flow-shop problems of more than two machine centers. The proposed flexible flow-shop algorithm is based on the LPT and the proposed search-and-prune approaches to manage job scheduling. The algorithm is decomposed into three parts as Sriskandarajah and Sethi's method was. The first part forms the machine groups, each of which contains a machine from each center. The second part uses the LPT method to assign jobs to each machine group (flow shop). The third part deals with job sequencing and timing using the search-and-prune procedure for a flow shop. The proposed algorithm is stated below.

The proposed LPT_ Search-and-prune flexible flow-shop algorithm:

Input: A set of n jobs, each having m (m > 2) tasks, to be executed respectively on each of m machine centers with p parallel machines.

Output: A schedule with a suboptimal completion time.

Part 1: Forming the machine groups
Step 1: Form p machine groups, each of which contains one machine from each machine center. Each machine group can be thought of as a simple flow shop $F_1, F_2, \ldots, F_p$.
Step 2: Initialize the completion time $f_1, f_2, \ldots, f_p$ of each flow shop $F_1, F_2, \ldots, F_p$ to zero.

Part 2: Assigning jobs to machine groups
Step 3: For each job J_j, $1 \le j \le n$, find its total execution time $tt_j = t_{1j} + t_{2j} + \ldots + t_{mj}$.
Step 4: Sort the jobs in descending order of processing time tt_j; if any two jobs have the same tt_j values, sort them in an arbitrary order.
Step 5: Find the flow shop F_i with the minimum processing time f_i among all the flow shops; if two flowshops have the same minimum f_i value, choose one arbitrarily.
Step 6: Assign the first job J_j in the sorted list to the chosen flow shop F_i, which has the minimum completion time f_i, among all p flow shops.
Step 7: Add the total time tt_j of job J_j to the needed total time of the chosen flow shop, F_i; that is:

$$f_i = f_i + tt_j.$$

Step 8: Remove job J_j from the job list.
Step 9: Repeat Steps 5 to 8 until the job list is empty.

After Step 9, jobs are clustered into p groups and are allocated to the p machine flow shops.

Part 3: Dealing with job sequence in each flow shop
Step 10: For each flow shop F_i, set the initial completion time of the machines f_{ji} (j = 1 to m, i = 1 to p) to zero.
Step 11: Find the completion time of each flow shop f_i by the proposed search-and-prune procedure in Section 4.

Step 12: Find the final completion time $ff = \max_{i=1}^{p}(f_i)$ among the completion time of all the flow shops.

After Step 12, scheduling is finished and a total completion time *ff* has been found.

An Example For The Proposed Heuristic Algorithm: Assume five jobs, J_1 to J_5, each having three tasks (t_{1j}, t_{2j}, t_{3j}), are to be scheduled via three operations. Each operation is executed by a machine at the corresponding machine center. Each machine center includes two parallel machines. Assume the execution times of these jobs are listed in Table 1. The algorithm proceeds as follows.

Table 1: Processing times for the five jobs

Execution time / Job_j	t_{1j}	t_{2j}	t_{3j}
J_1	4	7	3
J_2	1	5	2
J_3	5	2	4
J_4	2	5	3
J_5	5	5	6

Part 1: Forming the machine groups*:*

Step 1: Form two machine groups, F_1 and F_2, each of which is thought of as a three-machine flowshop. Without lose of generality, we may assume the flowshops are constructed as follows:

$F_1 \rightarrow \{m_{11} + m_{12} + m_{13}\}$,

$F_2 \rightarrow \{m_{21} + m_{22} + m_{23}\}$,

where m_{ij} is the *i*-th machine in the *j*-th center.

Step 2: Initialize $f_1 = f_2 = 0$, where f_i is the initial completion time of F_i.

Part 2: Assigning jobs to machine groups:

Step 3: For each job J_j, $j = 1$ to 5, find its total execution time $tt_j = t_{1j} + t_{2j} + t_{3j}$. For example, the total processing time of job 1 is calculated as:

$$tt_1 = t_{11} + t_{21} + t_{31} = 4 + 7 + 3 = 14 .$$

The total processing times of the other jobs can be similarly found and the results are listed in Table 2.

Table 2: The total processing times of the five jobs

Job_j	*total processing time* tt_j
J_1	14
J_2	8
J_3	11
J_4	10
J_5	16

Step 4: Sort the jobs J_1 to J_5 in a descending order of the total processing time (tt_j). The following results are obtained:

Job list = $\{J_5, J_1, J_3, J_4, J_2\}$.

Step 5: Find the minimum f_i between the two flowshops F_1 and F_2. Since both the total processing times of the two flowshops are equal to zero, any arbitrary flowshop can be chosen. Without lose of generality, assume F_1 is chosen.

Step 6: Assign the first job J_5 in the sorted list to the chosen flowshop F_1.

Step 7: Add the total processing time tt_5 of job J_5 to the needed total time of the chosen flowshop F_1. Thus:

$$f_1 = f_1 + tt_5 = 0 + 16 = 16.$$

After Step 7, the results of allocating J_5 to the flowshop F_1 are shown in Table 3.

Step 8: Remove the job J_5 from the job list. After J_5 is removed, the job list is then as follows:

Job list = $\{J_1, J_3, J_4, J_2\}$.

Step 9: Repeat Steps 5 to 8 until the job list is empty. After Step 9, jobs are clustered into two groups and are respectively allocated to the two flowshops. Results are shown in Table 4.

Table 3: The flowshops with allocated jobs and total processing time

$Flowshop_i$	*allocated jobs*	*total processing time*
F_1	J_5	16
F_2	*None*	0

Table 4: The jobs in each flow shop

Flowshop_i	Jobs allocated
F_1	J_5, J_4
F_2	J_1, J_3, J_2

Part 3: Dealing with job sequencing in each flow shop:

Step 10: In each flow shop F_i, set the initial completion time of the machines $f_{ji} = 0$ ($j = 1$ to 3, i=1 to 2).

Step 11: Find the completion time of each flow shop f_i by the proposed search-and-prune procedure in Section 4. The results are found as follows:

$$f_1 = 20,$$
$$f_2 = 18.$$

Step 12: Find the maximal final completion time ff between the completion times of both the flow shops. We can thus get:

$$ff = 20.$$

ff has been found ff is then output as the final total completion time. The schedule obtained by the above steps is shown in Fig. 1.

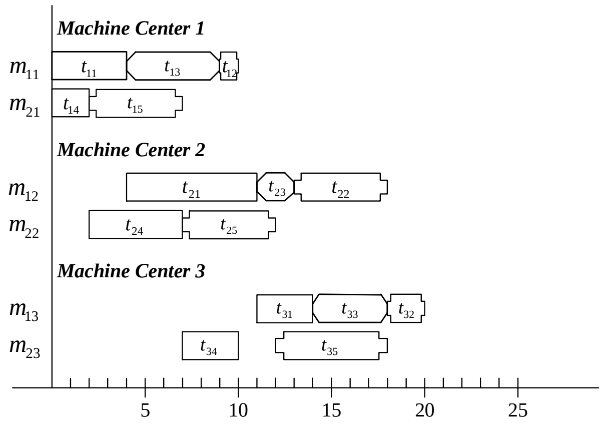

Fig. 1: The final scheduling result in the example

The Second Algorithm: In the first algorithm, the LPT method is used to assign jobs to machine groups. The job sequencing and timing in each group is then done by the search-and-prune procedure. The tasks in a set of clustered jobs are executed in the same machine group. The makespans obtained in the above way do not guarantee to be optimal. For getting an optimal schedule, the tasks in a set of jobs may be executed in different machine groups. In this section, we thus propose another scheduling algorithm based on the search-and-prune technique to get the optimal solutions, which can also be used to measure the performance of the first algorithm. The proposed optimal algorithm is stated below.

The proposed optimal flexible flow-shop algorithm:

Input: A set of n jobs, each having m ($m > 2$) tasks, to be executed respectively on each of m machine centers with p parallel machines.

Output: A schedule with an optimal completion time.

Step 1: Set the initial upper bound v_{max} of the final completion time as ∞ .

Step 2: For each possible combination of task allocation and permutation of task sequence, do the following steps.

Step 3: In each machine center, set the initial completion time of each machine to zero.

Step 4: Set the variable g to one, where g represents the number of the current machine center to be processed.

Step 5: Schedule the first tasks of all jobs in the machines of the first machine center. That is, for each task T_{1i} of the i-th job allocated to the j-th machine D_{j1} in the first machine center, do the following substeps according to the scheduling order in the permutation and combination generated:

(a)Add the processing time t_{1i} to the completion time d_{j1} of the machine D_{j1}. That is:

$$d_{j1} = d_{j1} + t_{1i,} \text{ and}$$
$$c_{1i} = d_{j1.}$$

(b)If d_{j1} is larger than v_{max}, neglect all the permutations and combinations with this sequence in the first machine center and go to Step 2 for trying another permutation and combination.

Step 6: Set $g = g + 1$.

Step 7: Schedule the g-th tasks of all jobs in the machines of the g-th machine centers according to the permutation and combination generated. For each task T_{gi} of the i-th job allocated to the j-th machine D_{jg} in the g-th machine center, do the following substeps in the scheduled order:

(a)Find the completion time d_{jg} of the machine D_{jg} as:

$$d_{jg} = max(d_{jg}, c_{(g-1)i}) + t_{gi}, \text{ and}$$
$$c_{gi} = d_{jg.}$$

(b)If d_{jg} is larger than v_{max}, neglect all the permutations and combinations with this sequence in the first g machine centers and go to Step 2 for trying another permutation and combination.

Step 8: Repeat Steps 6 and 7 until $g > m$.

Step 9: Set the completion time d_m of the current schedule $= \max_{j=1}^{p}\left(d_{jm}\right)$ among the p machines in the m-th machine center.

Step 10: If d_m is smaller than v_{max}, then set $v_{max} = d_m$.
Step 11: Repeat Steps 2 to 10 until all the possible permutations and combinations have been tested.
Step 12: Set the optimal final completion time of the job scheduling $ff = v_{max}$.

After Step 12, a globally optimal completion time ff has been found. In the above two algorithms, the permutations and combinations of task sequences or machine centers must be tested, causing the execution time is intractable in the worst case. Below, we propose another heuristic algorithm to reduce the computation time.

The Third Algorithm: The third algorithm is based on the PT approach to manage job scheduling. The algorithm is decomposed into three parts as the first algorithm. The first part forms the machine groups, each of which contains a machine from each center. The second part uses the LPT method to assign jobs to each machine group (flow shop). The third part deals with job sequencing and timing using the PT procedure for a flow shop. The proposed algorithm is stated below.

The proposed LPT_PT flexible flow-shop algorithm:
Input: A set of n jobs, each having m ($m > 2$) tasks, to be executed respectively on each of m machine centers with p parallel machines.
Output: A schedule with a nearly completion time.
Part 1: Forming the machine groups: The same as in the first algorithm.
Part 2: Assigning jobs to machine groups: The same as in the first algorithm.
Part 3: Dealing with job sequencing in each flow shop:
Step 10: For each flow shop F_i, set the initial completion time of the machines f_{ji} (j = 1 to m, i=1 to p) to zero.
Step 11: For each job j, calculate $t_j^C = \sum_{k=1}^{m/2} t_{kj}$ and $t_j^D = \sum_{k=(m/2)+1}^{m} t_{kj}$ for even m, and $t_j^C = \sum_{k=1}^{(m+1)/2} t_{kj}$ and $t_j^D = \sum_{k=(m+1)/2}^{m} t_{kj}$ for odd m.
Step 12: Schedule the jobs in each flow shop f_i according to the t_j^C and t_j^D values by the Johnson algorithm. Denote the schedule in F_i as Q_{Fi}.
Step 13: For each flowshop F_i, assign the first job J_j in Q_{Fi} to the machines such that J_{1j} is assigned to F_{1i}, J_{2j} is assigned to F_{2i}, …, and J_{mj} is assigned to F_{mi}.
Step 14: Add the processing time t_{1j} to the completion time of the first machine f_{1i}; that is:
$$f_{1i} = f_{1i} + t_{1j}.$$
Step 15: Set $f_{(k+1)i} = max(f_{ki}, f_{(k+1)i}) + t_{(k+1)j}$, for k =1 to (m-1).
Step 16: Remove job J_j from Q_{Fi}.
Step 17: Repeat Steps 13 to 16 until Q_{Fi} is empty.
Step 18: Set the final completion time of each flowshop f_i = the completion time of the m-th machine f_{mi}.
Step 19: Find the maximum final completion time $ff = \max_{i=1}^{p}(f_i)$ among the completion time of all the flowshops.

After Step 19, scheduling is finished and a total completion time ff has been found.

Experiments: This section reports on experiments made to show the performance of the proposed scheduling algorithms. They were respectively implemented by Visual C++ at an AMD Athlon(tm) XP 1800+ PC. In the first part of the experiments, five sets of problems were tested, respectively for 3 to 7 jobs. Each job has three tasks and each machine center has two parallel machines. The execution time of each task was randomly generated in the range of 5 to 50. Each set of problems was executed for 20 tests and the makespans and computation times were measured. The proposed optimal approach did not work for more than seven jobs in limited time of 10 hours in our environments due to the large amount of computation time.

The optimal approach considered all possible combinations and used a pruning technique to increase its efficiency. The makespans obtained in this way were optimal. The makespans for problems of three to seven jobs by the three proposed methods are shown in Fig. 2 to 4.

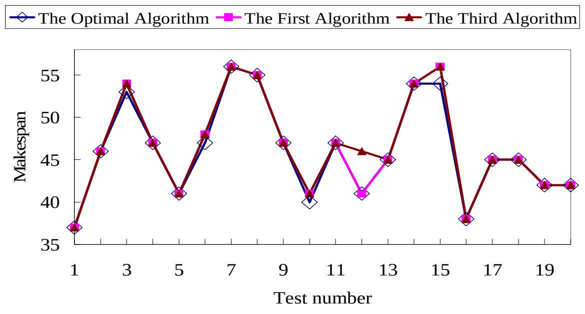

Fig. 2: Makespans of 20 tests for three jobs

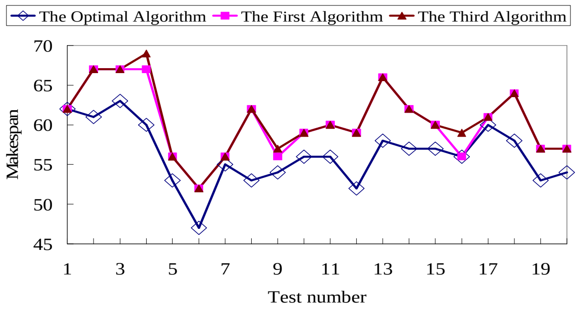

Fig. 3: Makespans of 20 tests for five jobs

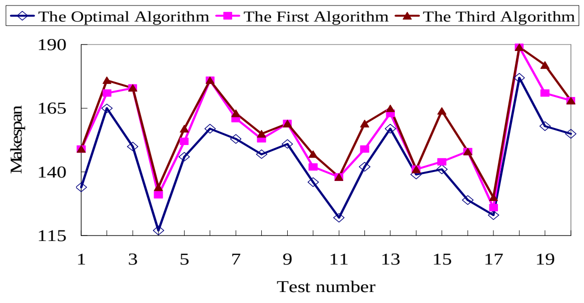

Fig. 4: Makespans of 20 tests for seven jobs

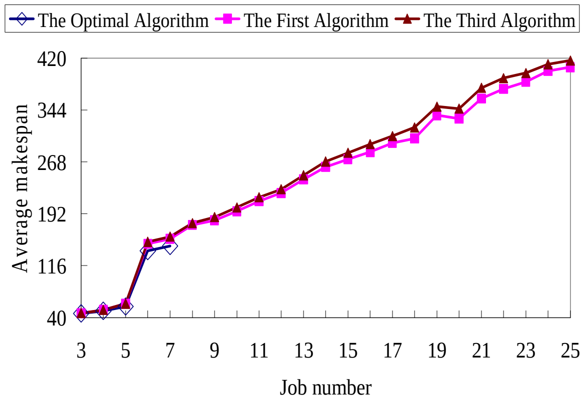

Fig. 5: Average makespans obtained by the three proposed algorithms

From Figs. 2 to 4, it is easily seen that the makespans by the proposed three algorithms have the following relation: Algorithm 3 > Algorithm 1 > Algorithm 2. It is totally consistent with our expectation. The deviation percentages for the first and the third algorithms from the optimal algorithm for processing different numbers of jobs are shown in Table 5. The average deviation percentage for the first and the third proposed heuristic algorithm from the optimal algorithm is respectively 5.05% and 6.19%. Note that the deviation rate for the second algorithm is 0% since it is an optimal approach.

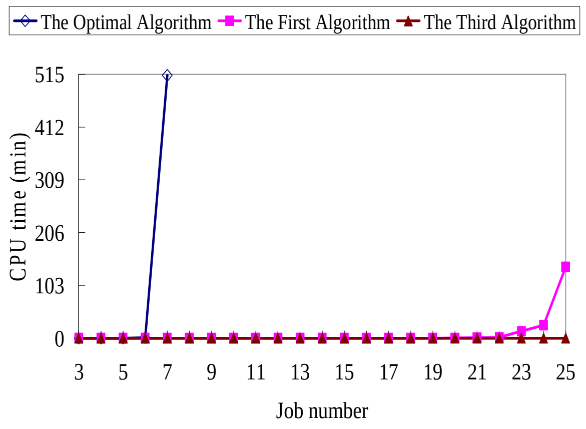

Fig. 6: The average CPU times for processing different numbers of jobs

Table 5: The distribution of deviation rates for different numbers of jobs and the run number is 20

Problem Size *n*	Run number	The first algorithm			The third algorithm		
		No. Optimals	Largest Deviation (%)	Average Deviation (%)	No. Optimals	Largest Deviation (%)	Average Deviation (%)
3	20	16	3.70	0.51	15	12.20	1.12
4	20	11	13.33	2.47	11	13.33	2.58
5	20	2	16.98	7.28	1	16.98	7.80
6	20	0	14.93	7.80	0	27.89	9.89
7	20	0	14.73	7.17	0	16.31	9.57
Total	100	29		5.05	27		6.19

In the second part of the experiments, we extend the job number to 25. The average makespan for problems with three to twenty-five jobs are shown in Fig. 5 for comparison. Note that the optimal approach can process no more than seven jobs in this environment.

The average CPU times for problems of three to twenty-five jobs are shown in Fig. 6. The optimal algorithm proposed cannot run over seven jobs in ten hours due to its high time complexity.

From Figs. 5 and 6, it is easily seen that the first and the third algorithms got a little larger makespans than the second one did. The computational time needed by the second algorithm was, however, much larger than that needed by the other two approaches, especially when the job number was large. Actually, since the flexible flow-shop problem is an NP-hard problem, the second approach can work only for a small number of jobs.

As to the first and the third algorithms, the latter got a little larger makespan but used less computational time than the former one. The former can be applied for solving a medium-sized problem.

At the last part, experiments for large job numbers ranging from 3 to 8000 were executed for verifying the efficiency of the third approach. The average CPU times for different jobs are shown in Fig. 7. It can be observed that all the execution times are less than 0.8 seconds. Hence, the third approach is feasible and efficient even for a large number of jobs. It is thus more suitable than the other two proposed approaches for real applications.

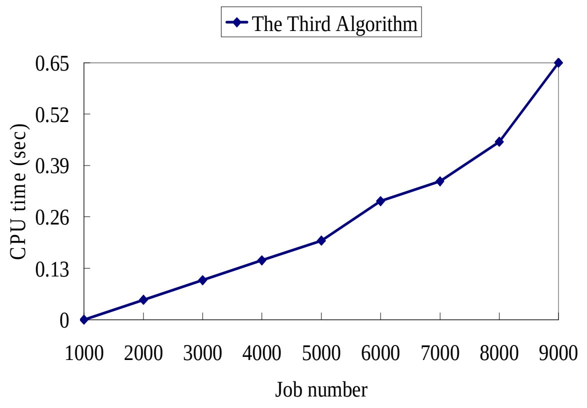

Fig. 7: The average CPU times for processing 3 to 8000 jobs by the third approach

CONCLUSION

Appropriate scheduling cannot only reduce manufacturing costs but also reduce the material handling cost and time. Finding good schedules for given sets of jobs can thus help factory supervisors control job flows and provide for good job sequencing. Scheduling jobs in flexible flow shops has long been known an NP-hard problem. In this paper, we propose three algorithms to solve flexible flow-shop problems of more than two machine centers. The first one extends Sriskandarajah and Sethi's method by combining both the LPT and the search-and-prune approaches to get a nearly optimal makespan. It is suitable for a medium-sized number of jobs. The second one is an optimal algorithm, entirely using the search-and-prune technique. It can work only when the job number is small. The third one is similar to the first one, except that it uses Petrov's approach (PT) to deal with job sequencing instead of search-and-prune. It can get a polynomial time complexity, thus being more suitable for real applications than the other two. Experimental results show that the computational times by the proposed three algorithms have the following relation: Algorithm 3 < Algorithm 1 < Algorithm 2, and the makespans have the following relation: Algorithm 3 > Algorithm 1 > Algorithm 2. It is totally consistent with our expectation. A trade-off can thus be achieved between accuracy and time complexity. The choice among the three proposed approaches to solve a flexible flow-shop problem thus depends on the problem size, the allowed execution time and the allowed error. In the future, we will consider other task constraints, such as setup times, due dates and priorities.

REFERENCES

1. Campbell, H. G., R. A. Dudek and M. L. Smith, 1970. A heuristic algorithm for the *n* job, *m* machine sequencing problem. Management Science., 16: B630-B637.
2. Chung, S. C. and D. Y. Liao, 1992. Scheduling flexible flow shops with no setup effects. The 1992 IEEE International Conference on Robotics and Automation., pp: 1179-1184.
3. Dudek, R. A., S. S. Panwalkar and M. L. Smith, 1992. The lessons of flowshop scheduling research. Operations Research., 40: 7-13.
4. Gupta, J. N. D., 1971. A functional heuristic algorithm for the flowshop scheduling problem. Operations Research., 40: 7-13.
5. Hong, T. P., C. M Huang and K. M. Yu, 1998. LPT scheduling for fuzzy tasks. Fuzzy Sets and Systems., 97: 277-286.
6. Johnson, S. M., 1954. Optimal two- and three-stage production schedules with set-up times included. Naval Research Logistics Quarterly., 1: 61-68.
7. Logendran, R. and N. Nudtasomboon, 1991. Minimizing the makespan of a group scheduling problem: a new heuristic. International Journal of Production Economics., 22: 217-230.
8. Morton, T. E. and D. W. Pentico, 1993. Heuristic Scheduling Systems with Applications to Production Systems and Project Management. John Wiley & Sons Inc., New York.

9. Nawaz, M., J. E. E. Enscore and I. Ham, 1983. A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. Omega., 11(1): 91-95.
10. Palmer, D. S., 1965. Sequencing jobs through a multi-stage process in the minimum total time- a quick method of obtaining a near optimum. Operational Research Quarterly., 16(1): 101-107.
11. Petrov, V. A., 1966. Flow Line Group Production Planning. Business Publications, London.
12. Sriskandarajah, C. and S. P. Sethi, 1989. Scheduling algorithms for flexible flow shops: worst and average case performance. European Journal of Operational Research., 43: 143-160.