A Heuristic for Two-Stage No-Wait Hybrid Flowshop Scheduling with a Single Machine in Either Stage

LIU Zhixin[1](刘志新), LI Jianguo[1]（李建国）, XIE Jinxing[1,*]（谢金星）,

DONG Jiefang[2]（董杰方）

[1] Department of Mathematical Sciences, Tsinghua University, Beijing 100084, China

[2] Wuhan Iron & Steel Group Company, Wuhan 430080, China

Abstract: This paper studies the hybrid flow-shop scheduling problem with no-wait restriction. The production process consists of two machine centers, one has a single machine and the other has more than one parallel machines. A greedy heuristic named Least Deviation (LD) algorithm is designed and its worst case performance is analyzed. Computational results are also given to show the algorithm's average performance compared with some other algorithms. LD algorithm outperforms the others in most practical cases discussed here, and it is of low computational complexity and is easy to carry out，thus it is of favorable application value.

Key Words: hybrid flowshop scheduling; no wait; heuristic; worst case analysis

Received: 2002-03-11; Revised: 2002-4-26

Supported by the National Natural Science Foundation of China (No. 69904007)

[*]To whom correspondence should be addressed. Tel: 86-10-6278 7812 E-mail: jxie@math.tsinghua.edu.cn

Introduction

Belonging to the class of NP-hard problems, the no-wait hybrid flowshop scheduling is among the most well studied combinational optimization problems. The no-wait restriction, as described in [1], occurs when the operations of a job have to be processed from start to end without interruptions between machines. This means, when necessary, the start of a job on a given machine is delayed in order that the operation's completion coincides with the start on the next operation on the subsequent machine. There are several industries where the no-wait flowshop problem applies. Examples include the metal, plastic, and chemical industries. For instance, in the case of steel production, the heated metal must continuously go through a sequence of operations before it is cooled in order to prevent defects in the composition of the material.

The no-wait hybrid flowshop scheduling problem generally can be described as follows. Given a list of machine centers, each having a fixed number of parallel machines, there are *n* jobs $\{P_i \mid 1 \le i \le n\}$ to be processed with the same fixed processing order on the machine centers, and each process must be carried out on at most one machine in every center, without any interruption either on or between machines during the process. The objective is to minimize the makespan of all the jobs, *i.e.* the time extent from the beginning time of the first job to the finished time of the last job being processed. The problem has attracted the attention of many researchers. A detailed survey of the application and research on the problem is given by Hall and Sriskandarajah [2]. Readers can also refer to [1], [3-9] for some new results in this field.

This paper evaluates the performance of some heuristics on a kind of relatively simple cases of the hybrid flowshop scheduling. Only two machine centers are taken into consideration and only a single machine is in one of the two centers, while it is still belonging to the NP-hard class, see [10] for details. During the heuristics we can take either center as the first one, which does not affect the makespan of any solution got by the heuristics. So to be convenient, we only discuss the case that one machine is always in the first center. Thus the problem can be denote as $F_2(m_1 = 1, m_2 = m > 1) \mid \text{no-wait} \mid C_{\max}$ [10] (problem F for shot), where F_2 denotes that the type of the problem is two-stage flowshop, no-wait describes the restriction, and $C_{\max}$ shows the optimal objective is to minimize the makespan.

1 Heuristic Design

For hybrid flowshop problem with no-wait restriction, among all the heuristics discussed here, the beginning time of any job on the first center should be restricted by the time point when there is at least one machine in the second center begins to be idle. This is in order to guarantee the no-wait restriction. The list scheduling algorithm (algorithm L for shot), originally designed as a heuristic for parallel machine scheduling in [11], is of broad applications in scheduling, and has been applied to hybrid flowshop scheduling in [12,13]. As for problem F, algorithm L is exactly the application of

the contrary deduction to get starting time point of every job for the machine in the first center for any given list of jobs.

In [13] another heuristic was also analyzed (named algorithm D here), which first sorts the jobs in non-increasing order of the second center processing time, and then uses algorithm L to arrange them to process. Besides, one can apply Johnson's Rule to arrange all the jobs during the first step and then use algorithm L to achieve the makespan, and we named this algorithm J. Johnson's Rule was designed in [14] for the classical two-stage flowshop problem without no-wait restriction, and is the exact algorithm for that problem.

All the above three heuristics consist of two steps, sorting and then determining the starting time of each job. However, the Least Deviation (LD for short) algorithm put forward in this paper is of different method. Algorithm LD sorts the jobs and determines the starting time simultaneously, *i.e.* the sorting process is dynamic. Let the machine in the first center be machine *a* and the *m* parallel machines in the second center be $\{b_1, b_2, \cdots, b_m\}$, then algorithm LD can be described as follows.

Algorithm LD. When the algorithm starts to run, all the machines in both machine centers are considered to be idle. For any time point when machine *a* begins to be idle, search the machine to be firstly idle among machines $\{b_j \mid 1 \le j \le m\}$ (any idle machine is naturally the first one; if there are more than one such kind of machines, randomly choose one). Let the processing time left to process the job on that machine be t. Among all the jobs to be processed, choose the job which has the closest first-center processing time to t (break ties arbitrarily). Suppose that the job chosen is job i^* and the processing time of job i^* on machine *a* is t_1. If $t \le t_1$, job i^* starts to process on machine *a* immediately; otherwise we have $t > t_1$, thus job i^* needs to start to process on machine *a* after $t - t_1$ waiting time.

Let p_{1i} and p_{2i} $(i = 1,...,n)$ be the processing times of job *i* in the first and second machine centers respectively. Then, the algorithm LD can be described in an exact way as below.

Step 0. Set $MA = 0$ and $MB_j = 0$ ($j = 1,...,m$); Set $J_i = L_i = K_i = 0$ ($i = 1,...,n$), $n^* = 1$ and $SP = 0$.

Step 1. Set $MB_j = \max(0, MB_j - MA)$ ($j = 1,...,m$) and $MA = 0$.

Step 2. Set $j^* = \arg\min\{MB_j \mid j = 1,...,m\}$, and set $t = MB_{j^*}$.

Step 3. Set $i^* = \arg\min\{\mid p_{1i} - t \parallel J_i = 0, i = 1,...,n\}$, and set $t_1 = p_{1i^*}$ and $J_{i^*} = 1$. Set $L_{n^*} = i^*$, $K_{n^*} = j^*$.

Step 4. Set $MA = \max(t, t_1)$ and $MB_{j^*} = MA + p_{2i^*}$.

Step 6. Set $n^* = n^* + 1$. If $n^* = n$, go to Step 7; Otherwise set $SP = SP + MA$ and go to

Step 1.

Step 7. Set $SP = SP + \max\{MB_j \mid j = 1,...,m\}$ and stop.

In the algorithm LD presented above, $L_i (i = 1,...,n)$ represents for the ordered processing list of all the jobs, $K_i (i = 1,...,n)$ represents for the corresponding machine for the jobs to be processed on the second center, and SP is the makespan.

For problem F, the computational complexity of algorithm L is $O(mn)$, and that of both algorithm D and algorithm J is $O(mn \lg n)$.

Theorem 1 For problem F with n jobs, the computational complexity of algorithm LD is $O(mn^2)$.

Proof There are n idle beginning time points for machine a. For each one of them, we should determine the machine in the second center, which need $O(m)$ searching. After that, the job of the least deviation processing time can be found in $O(n)$ time. So the total computational complexity is $O(mn^2)$.

Besides this, we can sort all the jobs according to their processing times in the first center, either in increasing order or decreasing order. Then the jobs can be searched by binary-search methods. Thus the average computational time can be reduced. Furthermore, according to advanced sorting theory, one can store the jobs in a more complicated data structure, such as binary search tree, and thus one can reduce the computational complexity of algorithm LD to $O(mn \lg n)$.

2 Worst Case Performance Analysis

As for scheduling problem, the worst case performance is among the most useful performance index of a heuristic, which describes the performance of a heuristic in the worst case. For a heuristic H, the worst case performance is the ratio of the makespan obtained by H and the optimal solution for the worst case instance, which is called tight if there is at least one instance to make the ratio hold.

For problem F, the worst case ratio of algorithm L is $3 - 1/m$, and that of algorithm D is 2. Both of them are tight [13]. Since algorithm J is a special case of algorithm L, its worst case ratio is obviously not larger than $3 - 1/m$.

Theorem 2 For problem F, the worst case ratio of algorithm LD is $3 - 1/m$, and it is tight in a general sense where some jobs can be processed only in one center.

Proof Since algorithm LD is a special case of algorithm L, the conclusion that its worst case ratio is not larger than $3 - 1/m$ can be directly drawn from the result for algorithm L, as was proved in [13].

For problem F in general sense, some jobs can be processed only in one center. An instance of

ratio $3-1/m$ is given as below. The job set is

$$J=\{J_a(i)\,|\,1\le i\le mN-(2m-1),\ J_b(i)\,|\,1\le i\le m-1,\ J_c(i)\,|\,1\le i\le m-1,\ J_d\},$$

and the processing times in both centers are

$$p[J_a(i)]=[\varepsilon,\phi],1\le i\le mN-(2m-1),$$

$$p[J_b(i)]=[\varepsilon,L],1\le i\le m-1,$$

$$p[J_c(i)]=[\varepsilon,(m-1)L],1\le i\le m-1,$$

$$p[J_d]=[\varepsilon,mL].$$

Here ϕ means the job need not to be processed on that center, ε <<L, and N is an integer big enough to make $N\varepsilon=L$.

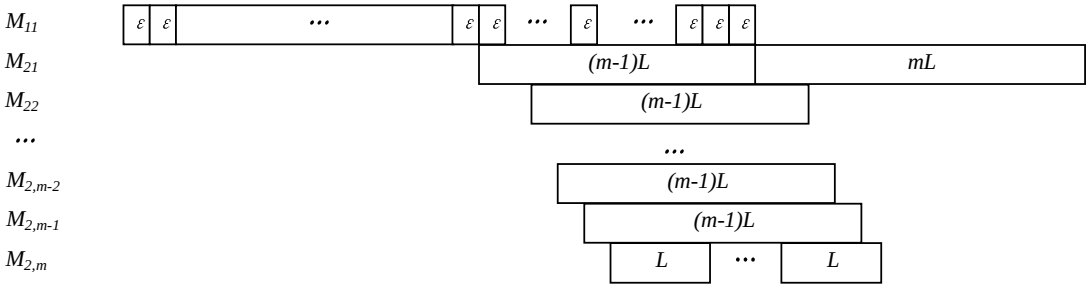

Fig. 1 Solution with the same makespan as algorithm LD, $f=(3m-1)L-(2m-2)\varepsilon$

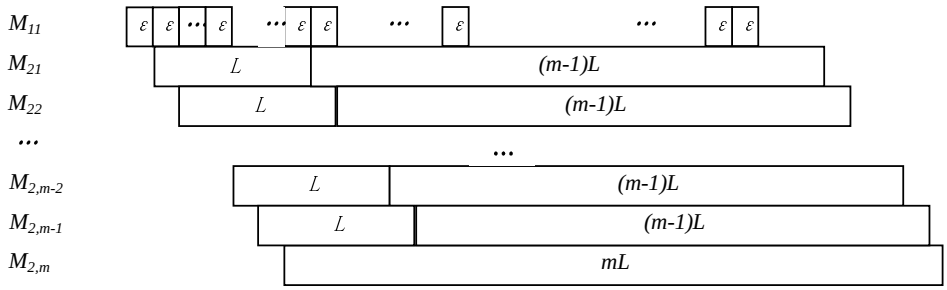

Fig. 2 Optimal solution, $f^*=mL+m\varepsilon$

From Fig. 1 and Fig. 2, we get,

$$\frac{f}{f^*}=\frac{(3m-1)L-(2m-2)\varepsilon}{mL+m\varepsilon}$$

$$=\frac{(3-1/m)-(2-2/m)\varepsilon/L}{1+\varepsilon/L}$$

Thus, $\lim\limits_{\varepsilon/L\to 0}\dfrac{f}{f^*}=3-\dfrac{1}{m}$.

If all the jobs need be processed on both centers, the instance with performance ratio of

$3-2/m$ can be shown as below, while whether there exists one with ratio $3-1/m$ is still open.

The job set is

$$J=\{J_a(i)\,|\,1\le i\le mN-(3m-5),\,J_b(i)\,|\,1\le i\le 2m-4,\,J_c(i)\,|\,1\le i\le m-2,\,J_d\},$$

and the processing time in both centers is,

$$p[J_a(i)]=[\varepsilon,\varepsilon],1\le i\le mN-(3m-5),$$

$$p[J_b(i)]=[\varepsilon,L],1\le i\le 2m-4,$$

$$p[J_c(i)]=[\varepsilon,(m-2)L],1\le i\le m-2,$$

$$p[J_d]=[\varepsilon,mL].$$

$\varepsilon \ll L$, and N is an integer big enough to make $N\varepsilon=L$.

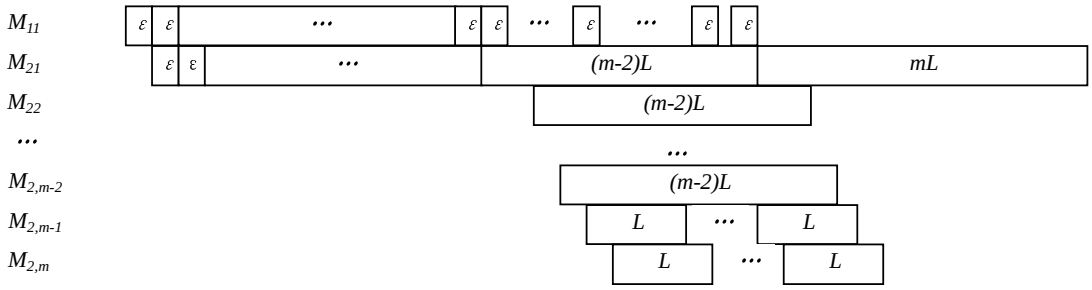

Fig. 3 Solution with the same makespan as algorithm LD, $f=(3m-2)L-(3m-6)\varepsilon$

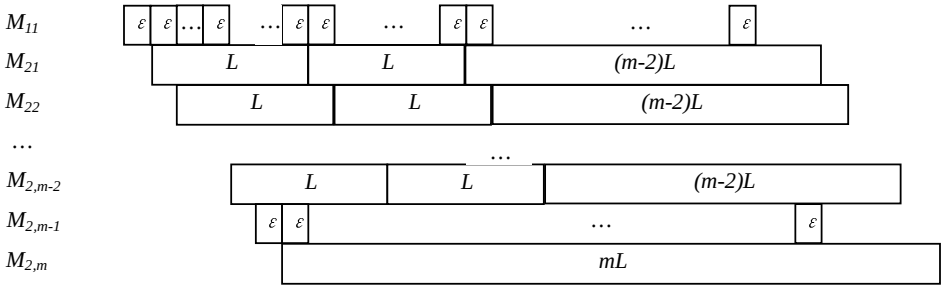

Fig. 4 Optimal solution, $f^*=mL+m\varepsilon$

From Fig. 3 and Fig. 4,

$$\frac{f}{f^*}=\frac{(3m-2)L-(3m-6)\varepsilon}{mL+m\varepsilon}$$

$$=\frac{(3-2/m)-(3-6/m)\varepsilon/L}{1+\varepsilon/L}$$

So, $\lim\limits_{\varepsilon/L\to 0}\dfrac{f}{f^*}=3-\dfrac{2}{m}$.

3 Average Case Performance Analysis

We evaluated the performance of all the four heuristic for problem F by a large number of randomly generated instances. Three cases with $m = 2,5,10$ were studied, and the processing times of the jobs were generated from uniform and normal distributions respectively.

Let p_{1i} and p_{2i} ($i = 1,2,...n$) denote the processing time on both the centers for the jobs.

For normal distribution, we designed the following instance sets:

I_1: both p_{1i} and p_{2i} are from $N(30,25)$,

I_2: both p_{1i} and p_{2i} are from $N(30,100)$,

I_3: p_{1i} is from $N(30,25)$ and p_{2i} is from $N(30m,25m^2)$,

I_4: p_{1i} is from $N(30,100)$ and p_{2i} is from $N(30m,100m^2)$.

For uniform distribution, we designed the following instance sets:

I_5: $0 < p_{1i}, p_{2i} \le 15$,

I_6, $0 < p_{1i}, p_{2i} \le 30$,

I_7: $0 < p_{1i} \le 15, 0 < p_{2i} \le 15m$,

I_8: $0 < p_{1i} \le 30$, $0 < p_{2i} \le 30m$.

Let nt denote problems tested in each instance (Here we have taken nt =100). Let f_r^H be the makespan, *i.e.* the $C_{\max}$, obtained by algorithm H (H=L, D, J, LD) for $I_r (r = 1,2,\cdots,8)$ instances, and let f_r^0 be a lower bound of the makespan of $I_r (r = 1,2,\cdots,8)$ of the optimal solution.

The lower bound f_r^0 of problem F is estimated as

$$f_r^o = \max\left\{\left(\sum_{i=1}^{n} p_{1i} + \min_i p_{2i}\right), \left(\frac{1}{m_2}\sum_{i=1}^{n} p_{2i} + \min_i p_{1i}\right)\right\}.$$

Let $e_r^H = \dfrac{f_r^H - f_r^o}{f_r^o} \times 100$ and $E_r^H = \dfrac{1}{nt}\sum_{r=1}^{nt} e_r^H$.

Thus, E_r^H (H=L, D, J, LD) indicates the relative performance of algorithms L, D, J and LD with respect to the lower bound. The heuristic with the least E_r^H has the closest solution to the optimal one, and has the best average performance compared with the others.

Table 1 Average performance evaluation of the heuristics

I	*n*	*m*	L	D	J	LD	O
I1	30	2	8.28	14.04	3.09	6.54	J
I2	30	2	7.15	13.60	2.50	6.84	J
I3	30	2	22.01	27.13	17.52	15.43	LD
I4	30	2	21.72	26.65	17.48	16.10	LD
I5	30	2	17.46	25.22	8.50	11.81	J
I6	30	2	17.86	26.67	8.40	12.64	J
I7	30	2	31.58	41.11	21.25	23.12	J
I8	30	2	30.73	41.12	20.63	19.55	LD
I1	300	2	5.89	11.01	1.98	6.17	J
I2	300	2	6.18	10.74	2.03	6.04	J
I3	300	2	21.52	25.95	18.07	14.24	LD
I4	300	2	22.11	26.31	18.70	15.25	LD
I5	300	2	12.36	17.76	5.87	8.42	J
I6	300	2	11.96	20.54	7.01	9.10	J
I7	300	2	29.52	38.62	20.61	18.80	LD
I8	300	2	28.78	38.26	20.60	17.63	LD
I1	60	5	4.81	9.45	2.25	5.03	J
I2	60	5	4.43	8.92	2.42	4.67	J
I3	60	5	34.30	42.31	33.38	28.32	LD
I4	60	5	34.16	43.27	30.12	27.55	LD
I5	60	5	7.72	13.61	1.96	6.48	J
I6	60	5	7.59	14.60	2.16	8.05	J
I7	60	5	51.39	64.19	40.33	41.18	LD
I8	60	5	50.11	63.20	37.79	40.86	J
I1	600	5	3.68	6.83	1.73	3.56	J
I2	600	5	3.21	6.47	1.79	3.13	J
I3	600	5	30.15	37.11	28.59	24.07	LD
I4	600	5	29.17	36.04	29.19	23.55	LD
I5	600	5	4.90	9.56	1.04	5.15	J
I6	600	5	5.13	9.80	1.33	4.55	J
I7	600	5	37.67	51.35	29.56	31.04	J
I8	600	5	40.71	52.39	31.14	31.50	J
I1	110	10	3.16	5.85	2.04	2.95	J
I2	110	10	3.02	5.76	1.89	2.63	J
I3	110	10	42.62	54.07	41.04	37.86	LD
I4	110	10	39.98	51.96	40.38	37.42	LD
I5	110	10	3.78	7.71	1.48	3.77	J
I6	110	10	4.52	8.03	1.34	4.02	J
I7	110	10	61.95	75.66	55.04	52.03	LD
I8	110	10	62.60	78.09	56.37	54.47	LD
I1	1100	10	2.29	4.32	1.34	2.25	J
I2	1100	10	2.15	4.26	1.28	2.21	J
I3	1100	10	35.03	44.05	35.39	30.45	LD
I4	1100	10	33.98	43.40	33.84	28.79	LD

I5	1100	10	2.75	5.18	1.12	2.58	J
I6	1100	10	3.09	5.57	1.01	2.66	J
I7	1100	10	48.19	59.92	42.19	39.12	LD
I8	1100	10	47.95	60.49	44.20	42.30	LD

The results are shown in Table 1, where m and n denote the number of machines in the second center and the number of the jobs, respectively. The last column lists the best heuristic among the four.

As shown in Table 1, compared with the ones with processing time of uniform distribution, the ones with processing time of normal distribution can be achieved closer makespan to the lower bound by the four heuristics discussed in this paper.

Among the four algorithms, obviously algorithms J and LD are relatively better than the other two. While for algorithm L and D, algorithm L is a little better. This result indicates that sorting the jobs by the processing time on the second center cannot improve the average performance even though it improves the worst case performance.

Furthermore, one can find a phenomenon for the performance of algorithms J and LD. For the cases the processing times in both centers are from the same distribution, *i.e.* I_1, I_2, I_5, I_6, algorithm J is better. While for the case the processing time correlates with the number of the machines in each center, algorithm LD is the better one. One should pay attention to that, in the real world, generally the processing times of the jobs are correlated to the number of machines where those jobs to be processed; Otherwise there maybe too many idle time on some machines in the second center.

4 Conclusions

For problem $F_2(m_1 = 1, m_2 = m > 1) \mid \text{no-wait} \mid C_{\max}$, we put forward the Least Deviation algorithm, analyzed its worst case performance, and then showed its efficiency by numerical experiments. The Least Deviation algorithm is of low computational complexity and is easy to be implemented, thus it is valuable to be used in real world applications.

Finally, we should point out that the generalization of this algorithm is straightforward to deal with the two-stage no-wait hybrid flow-shop scheduling problem with more than one parallel machines in both stages. That's to say, following the similar ideas and steps of the Least Deviation algorithm for the problem $F_2(m_1 = 1, m_2 = m > 1) \mid \text{no-wait} \mid C_{\max}$, we can design the Least Deviation algorithm for the generalized problem $F_2(m_1 \geq 1, m_2 > 1) \mid \text{no-wait} \mid C_{\max}$. In fact, we have also conducted numerical experiments for this generalized problem, and the results show that, compared with most of the other algorithms, the Least Deviation algorithm is very competitive under most of the cases we tested for the problem.

References

[1] Aldowaisan T, Allahverdi A. Total flowtime in no-wait flowshops with separated setup times. *Computers and Operations Research*, 1998, **25**: 757-765.

[2] Hall N G and Sriskandarajah C. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 1996, **44(3)**: 510-525.

[3] Agnetis A. No-wait Flow shop scheduling with large lot sizes. *Annals of Operations Research*, 1997, **70**: 415-438.

[4] Bianco L, Dell'Olmo P, Giordani S. Flow shop no-wait scheduling with sequence dependent setup times release dates. *INFOR*, 1999, **37**: 3-19.

[5] Ragendran C. Formulation and heuristic for scheduling in a Kanban flowshop to minimize the sum of weighted flowing, weighted tardiness and weighted earliness of containers. *International Journal of Production Research*, 1999, **37(5)**: 1137-1158.

[6] Kumar S, Bagchi T P, Sriskandarajah C. Lot streaming and scheduling heuristic for m-machine no-wait flowshops. *Computer and Industrial Engineering*, 2000, **38(1)**: 148-171.

[7] Abadi I N K, Hall N G, Sriskandarajah C. Minimizing cycle time in a blocking flowshop. *Operations Research*, 2000, **48(1)**: 177-180.

[8] Sawik T. Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. *Mathematical and Computer Modelling*, 2000, **31**: 39-52.

[9] Aldowaisan T. A new heuristic and dominance relations for no-wait flowshops with setups. *Computers and Operations Research*, 2001, **28**: 563-584.

[10]Garey M R, Johnson D S. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman San Francisco, 1979.

[11]Graham R L. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 1966, **45:** 1563-1581.

[12]Sriskandarajah C, Sethi S P. Scheduling algorithms for flexible flowshops: Worst and average case performance. *European Journal of Operational Research*, 1989, **43**: 143-160.

[13]Sriskandarajah C. Performance of scheduling algorithms for no-wait flowshops with parallel machines. *European Journal of Operational Research*, 1993, **70**: 365-378.

[14]Johnson S M. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1954, **1**: 61-68.