Paper

Dynamic Scheduling of Large-scale Flow Shops Based on Relative Priority Approach

Yang Jianhua* Student Member
Yasutaka Fujimoto* Member

A relative priority approach is proposed to solve dynamic scheduling problem for flow shop problem in this paper. Two neighbor jobs' relationship is represented by a relative priority, which implies permutation obstacle between them. An algorithm is developed to calculate relative priority matrix on jobs and a near-optimal result can be obtained with the calculation iteration of relative priority matrix. A printed circuit board production line, which is composed of 31 works, is used to demonstrate the efficiency of proposed approach. It shows that an excellent final sequence can be obtained while initial sequence is derived from rule-based result.

Keywords : Dynamic scheduling, Flow shop, Performance improvement

1. Introduction

Scheduling problems arise when a manufacturing system has possibilities of processing various jobs, which have different due-dates and a variety of product characteristics. In modern industry there exist volume production lines, which are referred to as flow shops where jobs go along the same route. The scheduling problem of flow shop is generally also NP-hard [1][2] although it seems to be simpler than that of job shop. Herein, we get a definition of a NP-hard problem in the static sense that deterministic jobs and machines are considered. In an actual factory a more realistic flow shop is a dynamic one where the jobs continuously arrive and machines may be reduced due to breakdowns. Generally a dynamic flow shop is much more difficult to get an optimal or near optimal solution than a traditional NP-hard problem in a static sense and a different study policy should be taken into account.

Moreover we usually consider special scheduling policy according to the size and complexity of a problem. For a small or intermediate size problem, we are often interested in finding its minimal or maximal value of objective function. But for a large-scale problem, obtaining an exact optimal value becomes almost impossible. As a consequence, macro viewpoints are taken and if just common characteristics of jobs are considered system will be simply modeled and easy to obtain a theoretical result. Nevertheless parameters of system model derived from statistical data of long history often involves a possibility of big error due to unstable system input and environment, with which obtained results may be far from performance improvement of current system.

In an industrial flow shop, we think, there simultaneously exist deterministic and stochastic elements. A deterministic element is decided in the sense that its fluctuation can be ignored with concerning overall system behavior. The more deterministic parameters are used, the more precise the system model is. But the system model size may become larger due to more detailed information. To an acceptable degree it is possible to face such a large-scale model using modern computer technique. As a practical instance hereafter studied, a printed circuit board production is considered, where there are about a hundred machines and thousands of work-in-process and preparation jobs.

In this paper we propose an approach based on relative priority to solve the above large-scale dynamic flow shop problem. Not only the current deterministic jobs but also predictable jobs in the future are considered. We try to obtain adaptable sequence preference, i.e. relative priority between jobs on every machine. A motivation of our research is the hard combination of heuristic rules and we expect to solve the problem based on inner nature of sequence characteristic.

The remainder of the paper is organized as follows. In Section 2. we review the relevant researches especially suitable for a dynamic environment. The basic concept of relative priority is described in Section 3. and a simple example is given in order to illustrate its nature. Then the calculation of relative priority is discussed and an algorithm is given. For demonstration of our proposed algorithm a printed circuit board production line is applied and some data results are given in Section 4. Concluding remarks appear in Section 5.

2. Literature Review and Notations

Perhaps Johnson's work in 1954 [3], which obtained a greedy solution for 2-machine flow shop, was a milestone of modern scheduling study. After that many studies have been expanding and attempted to solve more complicate cases but reached a disappointed result that most of them were NP-hard problem. No perfect approach is found to get an optimal solution for intermediate size problem although nowadays techniques of solving NP-hard problem constantly appear.

Under the consideration of large-scale dynamic flow shop scheduling problem, heuristic rules are widely adopted because of its simplicity and easy implementation [4]. Of the early works on heuristic rules, the algorithm of Giffler and Thompson [5] is

* Dept. of Electrical & Computer Eng., Yokohama National University
79-5 Tokiwadai, Hodogayaku, Yokohama 240-8501

considered as the basis of all other rules. And as a survey of scheduling heuristic rules, Panwalker and Iskander [6] reviewed 113 rules.

Many experimental studies [7]~[10] on heuristic rules for practical manufacturing system have revealed that a single heuristic rule is of poor ability to get a good result and hybrid policy is necessary. Unfortunately, the combination of rules is often hard. One of most popular hybrid policies is artificial intelligence, which stems from natural evolution. Neural network and genetic algorithm [11][12] are often applied to make such a decision. The heuristic rules are feasible and effective in a real factory, and heuristic rules based result is regarded as the start point of our research in this paper.

On the other hand, we adopt the standard notation to represent dynamic flow shop problems suggested by Graham et al [13]. The triple formulation $\alpha|\beta|\gamma$ indicates machine configuration, job characteristic and optimal criteria respectively. For instance, $F_m \| C_{\max}$ denotes minimization of make-span for an m-machine flow shop problem, concerning with a blocking environment. Generally notation F denotes flow shop and $C_{\max}$ the completion time of the last job.

By default, the attribute β of the formulation $F_m \| C_{\max}$ represent that it is busy-scheduling where a machine will start processing if its buffer is not empty, and it is non-preemptive where each job must be processed without any interruption. A special term should be added while additional characteristics are included. For example, $F_m | dynamic | C_{\max}$ indicates continuous arrival of jobs, i.e., the dynamic case.

Usually we assume that in a flow shop all jobs follow the same route and each job has exactly one operation on each machine. Considering specific cases, some modifications are allowed, for example a flow shop may be composed of a series of works, each of which consists of several similar machines. A detailed description of such variations of a practical flow shop is given in Section 4.

3. Methodology

3.1 Basic Problem One-machine problem is the most primitive in scheduling field. So it is often used to illustrate the efficiency and feasibility for proposed approach because of its simplicity and easiness. Static scheduling of one-machine problem can be described as follows.

Given a set of jobs $N=\{1,2,\cdots,n\}$. The due-dates of jobs and the processing time on the unique machine are represented by $D=\{d_1,d_2,\cdots,d_n\}$ and $T=\{\tau_1,\tau_2,\cdots,\tau_n\}$, respectively. The complete time of job $j\in N$ is represented by c_j. Concerning the makeshifts of job j, namely d_j-c_j, the job j is late if $d_j-c_j<0$, or it is spare. As an example, $D=\{d_1,d_2,\cdots,d_{50}\}$ and $T=\{\tau_1,\tau_2,\cdots,\tau_{50}\}$ is given using following formula

$$d_j = 450(j/10+1) \quad \tau_j = 100r \cdots\cdots (1)$$

where r is a random number between 0-1 and operator / truncates fractional part. Table 1 shows an instance of generated data.

One of the most common scheduling problems is how to reduce the number of late jobs and their late degree. Firstly dispatching rules are considered and applied to solve above scheduling problems. Min-Slack (the shorter slack time $d_j-\tau_j$ the earlier mount on machine) rule, which is hereby equivalent to combination of LPT (the longer processing the earlier mount) rule and EDD (the earlier due-date the earlier mount) rule, is one of possible solutions. Based on sequence of Table 1 and Min-Slack dispatching rule, both results of job makeshift are shown in Fig.1.

Unfortunately, Fig.1 shows that the Min-Slack rule is even worse than the sequence of Table 1, a random one. From Fig.1, a question appears: whether is there a way to get the best solution? When the system scale is small enough, it is possible to analyzed and get a greedy solution, using existed approaches, such as integer programming. But with the increase of system scale, it becomes hard. Consequently, above question becomes: whether is there a way to find a good enough solution? Nowadays, local search methodologies have prosperously proposed to answer the question. Among various local search of iteration for improving system performance, threshold algorithm [14], genetic algorithm [15] and Tabu algorithm [16] are usually applied. In a traditional local search algorithm, objective function is enhanced by investigating

Table 1. An instance of one-machine problem.

τ_{1-10}	71	50	59	20	23	49	75	77	60	10
d_{1-10}	450									
τ_{11-20}	28	16	95	60	50	24	69	26	73	39
d_{11-20}	900									
τ_{21-30}	60	61	26	39	10	3	98	3	66	64
d_{21-30}	1350									
τ_{31-40}	45	37	15	4	57	38	54	33	16	14
d_{31-40}	1800									
τ_{41-50}	43	44	30	38	5	80	62	74	7	36
d_{41-50}	2250									

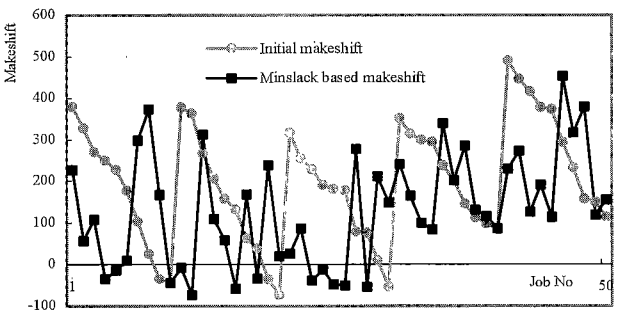


Fig. 1. Result based on Min-Slack dispatching rule Vs Initial result.

to a group of neighbor, which is derived from random permutation. One obstacle of these local search methods is how to reduce time cost because neighbors are nearly out of count for a large-scale system. Another one is that it is difficult to deal with dynamic case.

3.2 Concept of Relative Priority At first, we try to solve above problem using general local search. Neighbors of current sequence are randomly selected. If a better one appears, goes to it. Obviously such a policy takes less time than searching an appropriate one from a group of better neighbors, but probably reaches a not good enough optimal solution.

For the problem of section 3.1, objective function f is defined on jobs' makeshifts $\{d_j - c_j\}_n$.

$$f = \sum_{j=1}^{n} \lambda_j (d_j - c_j - \sigma)^2 \quad \cdots\cdots (2)$$

$$\lambda_j = 0 \quad \text{if} \quad d_j - c_j - \sigma \geq 0$$
$$\lambda_j = 1 \quad \text{if} \quad d_j - c_j - \sigma < 0$$

where σ is referred to as spare time factor, which is introduced to improve spare time of a job even if no job lateness occurs.

Thus the problem of improvement of jobs' spare time and decrease of jobs' lateness is equivalent to minimization of f. Let $\sigma = 300$, which means that job makeshifts will be closer to 300 when iteration reaches a better solution. Fig.2 shows two simulation results based on random local search. It shows that there might exist several local minimization points and some of them are far from the best one. Therefore, it is important to make a way to get the good enough one.

Herein, we focus on the worse solution and generation of random permutation. Concept of relative priority is introduced as follows. The initial sequence is represented by sequence vector $\pi = [k_1, k_2, \cdots, k_n]$, $(k_u \in N, \quad u = 1,2,\cdots,n)$, which specifies that the job will be processed with the sequence $k_1, k_2, \cdots, k_n$. A vector of relative priority $P = \{p_1, p_2, \cdots, p_n\}$ is defined on real number, where element $p_j (j = 1,2,\cdots,n)$ represents the priority relationship between jobs $k_u k_{u+1}$, where $j = k_u$. Given two integers $u, v \in \{1,2,\cdots,n\}$. If $p_u > p_v$, it means that the gap between jobs $k_u k_{u+1}$ is deeper than that between jobs $k_v k_{v+1}$, i.e., the permutation between $k_u k_{u+1}$ will be more difficult than that between $k_v k_{v+1}$, Vice versa.

Given the initial sequence $\pi_0 = [k_1^0, k_2^0, \cdots, k_n^0]$ and its corresponding objective value f_0. Let the initial relative priority vector $P^0 = [p_1^0, p_2^0, \cdots, p_n^0] = [0,0,\cdots,0]$. A random permutation occurs between jobs $k_1^0 k_2^0, k_2^0 k_3^0, \cdots, k_{n-1}^0 k_n^0$, producing a new sequence π_1 and its objective value f_1. Then relative priority vector is refreshed by

$$P^1 = [p_1^1, p_2^1, \cdots, p_n^1] \quad \cdots\cdots (3)$$

$$p_j^1 = \begin{cases} 0 & if \quad k_u^0 k_{u+1}^0 = 0 \\ \delta_1 & if \quad k_u^0 k_{u+1}^0 = 1 \quad j = k_u^0 \\ -\delta_1 & if \quad k_u^0 k_{u+1}^0 = 1 \quad j = k_{u+1}^0 \end{cases} \quad \cdots\cdots (4)$$

where $k_u^0 k_{u+1}^0 = 1$ stands for permutation between jobs $k_u^0 k_{u+1}^0$ and $k_u^0 k_{u+1}^0 = 0$ for no permutation occurrence. On the other hand, we have $k_u^0 k_{u+1}^0 = 0$ if $k_{u-1}^0 k_u^0 = 1$ because a job can be only permuted one times for each step.

δ_1 is referred to as relative priority factor, which can be calculated by

$$\delta_1 = \frac{f_1 - f_0}{f_1 + f_0} \quad \cdots\cdots (5)$$

The next neighbor π_2 will be generated from sequence π_1 even if $f_0 < f_1$. However the permutations at next step will be select using following formula

$$k_u^1 k_{u+1}^1 = \begin{cases} 0 & if \quad r \geq 0.5(1+\theta_j^1) \\ 1 & if \quad r < 0.5(1+\theta_j^1) \end{cases} \quad \cdots\cdots (6)$$

where r is random number between 0-1 and θ_j is a threshold, calculated by

$$\theta_j^1 = \begin{cases} (p_j^1 - p_{\min}^1)/(p_{\max}^1 - p_{\min}^1) & p_j^1 \neq 0 \\ 0 & p_j^1 = 0 \end{cases} \quad \cdots\cdots (7)$$

$$p_{\max}^1 = \max p_j^1, \quad p_{\min}^1 = \min p_j^1 \quad \cdots\cdots (8)$$

Similarly, relative priority vector is refreshed by

$$P^2 = [p_1^2, p_2^2, \cdots, p_n^2] \quad \cdots\cdots (9)$$

$$p_j^2 = \begin{cases} 0 & if \ k_u^1 k_{u+1}^1 = 0 \text{ and } p_j^1 = 0 \\ \mathrm{sgn}(\delta_2) & if \ k_u^1 k_{u+1}^1 = 1 \text{ and } p_j^1 = 0 \\ \dfrac{p_j^1 + \mathrm{sgn}(\delta_2)}{2} & if \ k_u^1 k_{u+1}^1 = 1 \text{ and } p_j^1 \neq 0 \end{cases} \quad \cdots\cdots (10)$$

where

$$\delta_2 = \frac{f_2 - f_1}{f_2 + f_1} \quad \cdots\cdots (11)$$

$$\mathrm{sgn}(\delta_2) = \begin{cases} \delta_2 & if \quad j = k_u^1 \\ -\delta_2 & if \quad j = k_{u+1}^1 \end{cases} \quad \cdots\cdots (12)$$

For the new sequences $\pi_3, \pi_4, \pi_5, \cdots$, the same process as the above is applied.

Finally, for the change of objective value, two results are shown in Fig.2. One of improved makeshift results is shown in Fig.3 and Table 2. Fig.2 clearly indicates that the relative priority approach can more quickly and stably close to final solution, implying that

the relative priority based search have the possibility of overcome the local minimization. Moreover, its final solution is better than that of random local search. Note that compared to the initial result, not only the so-called job lateness has been improved in Fig.3, but also has the spare time of jobs been improved. The number of jobs that spare time is more than 100 is increased by 8 and the number of jobs that spare time is more than 200 is increased by 9.

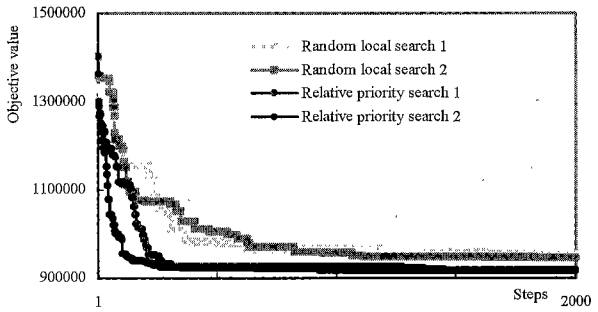


Fig. 2. Relative priority search Vs random local search.

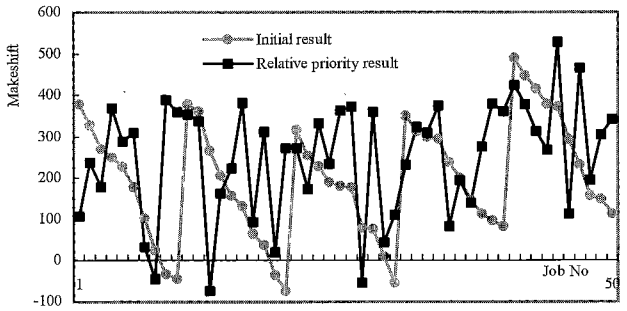


Fig. 3. Result based on relative priority approach Vs Initial result.

Table 2. Relative priority based result ($s_j = d_j - c_j$).

u	j	τ_j	d_j	c_j	s_j	u	j	τ_j	d_j	c_j	s_j
1	9	60	450	60	390	26	24	39	1350	1115	235
2	4	20	450	80	370	27	22	61	1350	1176	174
3	10	10	450	90	360	28	30	64	1350	1240	110
4	6	49	450	139	311	29	29	66	1350	1306	44
5	5	23	450	162	288	30	27	98	1350	1404	-54
6	2	50	450	212	238	31	39	16	1800	1420	380
7	3	59	450	271	179	32	34	4	1800	1424	376
8	1	71	450	342	108	33	40	14	1800	1438	362
9	7	75	450	417	33	34	32	37	1800	1475	325
10	8	77	450	494	-44	35	33	15	1800	1490	310
11	16	24	900	518	382	36	38	33	1800	1523	277
12	11	28	900	546	354	37	31	45	1800	1568	232
13	12	16	900	562	338	38	36	38	1800	1606	194
14	18	26	900	588	312	39	37	54	1800	1660	140
15	20	39	900	627	273	40	35	57	1800	1717	83
16	15	50	900	677	223	41	45	5	2250	1722	528
17	16	60	900	737	163	42	47	62	2250	1784	466
18	17	69	900	806	94	43	41	43	2250	1827	423
19	19	73	900	879	21	44	42	44	2250	1871	379
20	13	95	900	974	-74	45	50	36	2250	1907	343
21	26	3	1350	977	373	46	43	30	2250	1937	313
22	25	10	1350	987	363	47	49	7	2250	1944	306
23	28	3	1350	990	360	48	44	38	2250	1982	268
24	23	26	1350	1016	334	49	48	74	2250	2056	194
25	21	60	1350	1076	274	50	46	80	2250	2136	114

The principle of relative priority method depicted above is simple. A job, which is selected to be processed early or late, should be suppressed by its corresponding relative priority. A threshold is calculated from relative priorities of all jobs, determining which jobs are more *possibly active* to be permuted, which jobs are more *possibly inactive* to be permuted. The basic idea of relative priority method is stemmed from genetic principle [15], where the gene (relative priority) more adaptive to circumstance might be more possibly inherited (inactive to be permuted), and the Tabu search method [16], where a taboo list is introduced to limit the permutation. On investigation to these methods, we try to overcome two obstacles, the explosive model size and the permutation with numerous possibilities, in order to solve the large-scale case. Hereby, the binary model, the basic genetic algorithm model, is replaced by job's relative priority description. And the Tabu is replaced by the possibility of permutation, a more flexible limitation. Compared to traditional genetic algorithm and Tabu search, both of which also belong to variants of basic random search, the relative priority method has at least four advantages: (1) the model size is reduced and it become possible to deal with large-scale case; (2) the search efficiency is enhanced, which is also a key factor because numerous permutations appear if we consider a large scale case; (3) the local optimal point can be gotten over; (4) Uncertain jobs can also be included as described in the next Section.

3.3 $F_m \mid Dynamic \mid f_{\min}$ **Problem** A $F_m \mid Dynamic \mid f_{\min}$ problem is described as follows. Given a set of machines $V = \{1,2,\cdots,m\}$ and a set of buffers $B = \{b_1, b_2, \cdots, b_m\}$, which implies that there is a buffer for each machine. Given a set of jobs $N = \{1,2,\cdots,n\} = N_w \cup N_d \cup N_s$, where N_w stands for a set of work-in-process jobs, N_d for deterministic jobs ready to enter and N_s for stochastic jobs having a possibility of being cancelled or with a changeable due-date. $D = \{d_1, d_2, \cdots, d_n\}$ stands for due-dates of all jobs, and $T = \{\tau_{ij}\}_{m \times n}$ for processing time of job $j \in N$ on machine $i \in V$. Objective function f is used to evaluate all possible sequences and the goal is to get minimization of f.

For dynamic system, i.e., continuous arrivals of jobs considered, the first question is how to evaluate its performance, namely the construction of objective function f. As the evaluation of deterministic part, an objective function can be constructed on N_d and N_w. For example,

$$f_d = f_d(d_\alpha, d_\beta, c_\alpha, c_\beta) \quad \cdots\cdots (13)$$

where d_α, d_β denotes due-date of N_d, N_w and c_α, c_β denotes complete time of N_d, N_w, respectively. But we know that an optimal schedule over N_d and N_w might grow worse

because the N_s might have an effect on c_α, c_β. Therefore, for dynamic case, we do not try to get an absolute minimization of objective vale but to get a stable relative priority matrix

$$\Delta P < \varepsilon \quad \text{such that} \quad \Delta f_d < \lambda \qquad \cdots\cdots (14)$$

where ε is a tiny constants and λ denotes the variation range of objective value.

The second question is how to use relative priority approach effectively because a job might be permuted again with another job on another machine even if it has been permuted. One of feasible policies is that only first machine's jobs can be permuted and after that FIFO rule is used. In this paper, we make a modification to this policy, that just bottleneck machines are considered, where permutations are allowed.

Therefore, we get a calculation iteration of relative priority matrix for m-machines case as follows. The job sequence i.e., the schedule

$$\pi = [k_{ij}]_{m\times n'}, \qquad \cdots\cdots (15)$$

$$i \in V,\ k_{ij} \in N_w \cup N_d,\quad n' = d + w$$

is defined on jobs. The smaller the subscript j is, the earlier the job k_{ij} will be mounted. A matrix of relative priorities $P = [p_{ij}]_{m\times n'}, (n' = d + w)$ is defined on real number, such that

$$P^0 = [0]_{m\times n'} \qquad \cdots\cdots (16)$$

$$P^v = [p_{ij}^v]_{m\times n'} \qquad \cdots\cdots (17)$$

$$p_{ij}^v = \begin{cases} 0 & if\ k_{iu}^{v-1} k_{i(u+1)}^{v-1} = 0 \text{ and } p_{iu}^{v-1} = 0 \\ \mathrm{sgn}(\delta_v) & if\ k_{iu}^{v-1} k_{i(u+1)}^{v-1} = 1 \text{ and } p_{iu}^{v-1} = 0 \\ \dfrac{p_{ij}^v + \mathrm{sgn}(\delta_v)}{2} & if\ k_{iu}^{v-1} k_{i(u+1)}^{v-1} = 1 \text{ and } p_{iu}^{v-1} \neq 0 \end{cases} \qquad \cdots\cdots (18)$$

$$\delta_v = g(f_{\pi_v}, f_{\pi_{v-1}}) \qquad \cdots\cdots (19)$$

$$\mathrm{sgn}(\delta_v) = \begin{cases} \delta_v & if \quad j = k_{iu}^{v-1} \\ -\delta_v & if \quad j = k_{i(u+1)}^{v-1} \end{cases} \qquad \cdots\cdots (20)$$

$$p_{ij}^v \equiv p_{(i-1)j}^v \qquad \cdots\cdots (21)$$

$$i \notin V^t \cup V^b \text{ and } j \in N_w \cup N_d$$

where π_v, π_{v-1} stand for two neighbor sequences, respectively. The factor δ_v implies deterioration or improvement from π_{v-1} to π_v. V^t and V^b denote throwing machines and bottleneck machines, respectively. The permutation of jobs will be selected using the same formula as section 3.2 proposed.

4. A Printed Circuit Board Flow Shop

4.1 Description FW denotes a flow shop composed of works. Each work might be composed of several parallel machines. In this paper, we consider a case of $FW \mid dynamic \mid f_{\min}$, characterized by

① There are 31 works, composed of 103 machines;
② Processing time is dependent on job sequences;
③ Three works can be used to enter ordered jobs.

Generally the whole line is divided into two stages. One is inner layer stage, which is responding for processing cores, such as chemical surface treating, dry film laminating, etching, striping, etc. And the other is outer layer stage, which is responding for processing bases, whose some of processes are similar to that of inner layer stage. Before the completion of inner layer stage, cores are stick to a base. At one of outer works, a base is cut into several panels, i.e. printed circuit boards.

The processing time of a job on all machines is depicted in Fig.4. A bar indicates a core, a base or a printed circuit board.

For core processing, we calculate a job k_j's processing time on machine w_i by

$$\tau_{ij} = t(size, i)(q(l-2)-2)/2 - 1) + d + \eta \qquad \cdots\cdots (22)$$

where $t(\mathrm{size}, i)$ stands for tact time which is related to product size, q for job size, l for the number of core layers of a panel, d for lead time, and η for arrange time. η might occur, for example, on etching work when the thickness of copper foil of the core differs from the previous one.

For base processing and final panel processing, we get similar formulations

$$\tau_{ij} = t(\mathrm{size}, i)(q-1) + d + \eta \qquad \cdots\cdots (23)$$

$$\tau_{ij} = t(\mathrm{size}, i)(qb-1) + d + \eta \qquad \cdots\cdots (24)$$

respectively, where b represents the number of boards on a panel. Also might η occur due to the change of base size, job, cage etc.

Define a work's processing speed υ by $\upsilon = \kappa / \tau$, where κ is the number of machines of the work and τ represents the mean processing time for a job. Deriving from 1626 sampling jobs, all works' processing speeds are depicted in Fig.5.

4.2 Simulation and Data Results At first we obtain 1063 jobs lateness from 1626 jobs (2 weeks' load and 563 predicted jobs) simulation result, as shown in Fig.6, based on FIFO and EDD rules, currently used in factory. The result shows

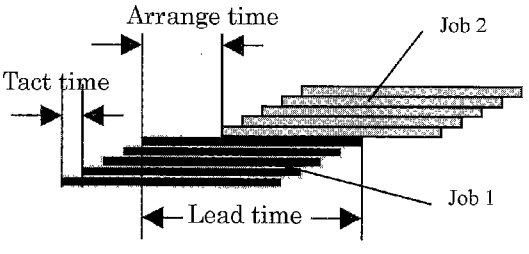


Fig. 4. processing time of two jobs.

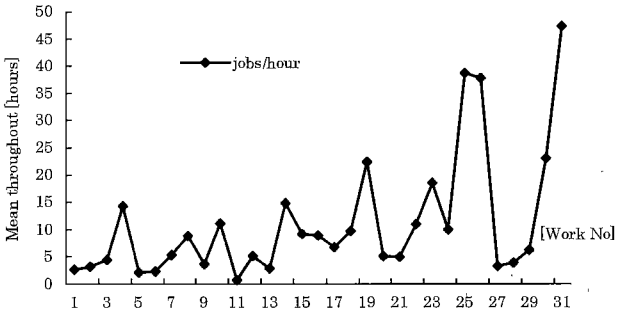


Fig. 5. Mean throughout for each work.

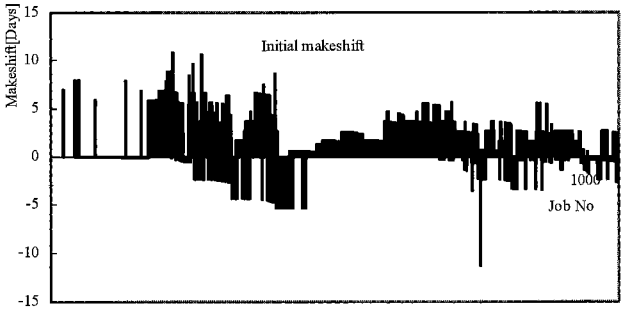


Fig. 6. 1063 jobs lateness using FIFO and EDD rules.

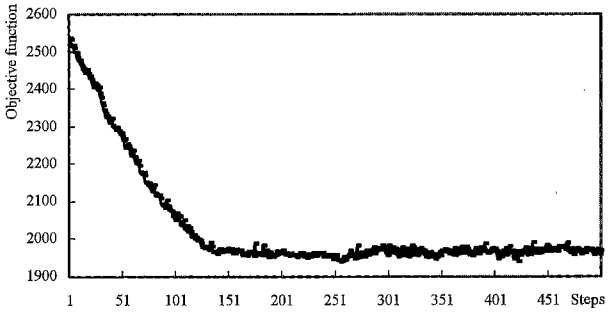


Fig. 7. Improvement of objective function.

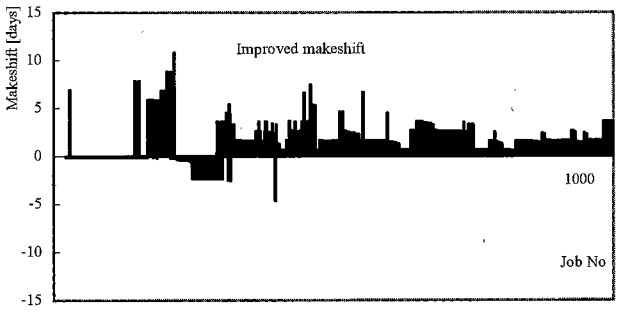


Fig. 8. 1063 jobs lateness using relative-priority algorithm.

that some jobs are finished several days earlier than expected but some jobs delay. Therefore it is possible to reduce the jobs' delay by reducing foregoing jobs' speed simultaneously.

Based on relative priority algorithm, proposed in Section 3, the jobs delay can be decreased with the iteration of calculating relative priorities for jobs in all buffers of 31 works. In practical computing process, only bottleneck works (5,21,27), whose processing speed is obviously lower than previous work and next work, and 3 input works (1,4,11) are considered.

Define objective function as

$$f = \sum \lambda_j (d_j - c_j - 1)^2 \quad \cdots\cdots (25)$$

$$\lambda_j = 0 \quad if \quad d_j - c_j - 1 \geq 0$$
$$\lambda_j = 1 \quad if \quad d_j - c_j - 1 < 0$$

where all time units are represented by days and $\sigma = 1$ implies that a job, whose spare time is less than a day, might be improved. The change of objective function value is shown in Fig.7 with the progress of iteration using algorithm given in Section 3.3.

The final result of job makeshift is given in Fig.8. It shows that the most jobs lateness can be greatly reduced but some of them nearly unchangeable because they are work-in-process jobs, which have passed a few works and have been late before the start point of simulation.

It takes about 2765 seconds for all 500 steps and about 730 seconds, i.e. 150 steps, to get the final stable priority set using Linux operation system on Intel Pentium 4/1.8GHz personal computer. Also Fig.8 shows the objective function value might be slightly change due to the influence of predicated jobs. Rescheduling is needed after the system load is greatly changed, i.e. a considerable predicated jobs have been determined. One of advantages of relative priority approach is that the old result can remain and it might take less time to get a new stable relative priority matrix.

On the other hand, in above printed-circuit-board production line arrange time should be cut down because it will reduce the working efficiency of machines. It is possible to decrease the arrange time using relative priority approach. The jobs, whose relative priorities are less than certain valve, can be mounted according the sequence where less arrange time occurs.

5. Conclusion

Relative priority approach is proposed to solve a large-scale dynamic flow shop problem in this paper. In relative priority approach, the job sequence is described as a matrix of relative position possibilities, which are in proportion to their effects on objective function. The original motivation stems from the fact that simple rule-based scheduling is usually unable to improve effectively the performance. And the obtained result shows that relative priority approach is a possible solution to similar problems.

(Manuscript received June 24, 2002, revised December 5, 2002)

References

(1) J. D. Ullman : "NP-complete scheduling problems", *Journal of Computer and System Science*, Vol. 10, pp.384-393 (1975)

(2) H. Kamoun and C. Sriskandarajah : "The complexity of scheduling jobs in repetitive manufacturing systems", *European Journal of Operation Research*, Vol.70, No.3, pp.350-364 (1993)

(3) S. M Johnson : "Optimal two and three-stage production schedules with setup times includes", *Nav. Res. Logist. Quart*, Vol.1, pp.61-68 (1954)

(4) A. Reisman, A. Kumar, and J. Motwani : "Flowshop scheduling/sequence research: A statistical review of literature 1952-1994", *IEEE Trans. on Engineering Management*, Vol. 44, No. 3, pp.316-329 (1997-3)

(5) B. Giffler and G. L. Thompson : "Algorithm for solving production scheduling problem ", *Operation Research*, Vol. 8, No.4, pp.487-503 (1960-4)

(6) S.S. Panwalker and W. Iskander: "A survey of scheduling rules", *Operation Research*, Vol.25, No.1, pp.45-61 (1977-1)

(7) Lawrence M. Mein : "Scheduling semiconductor wafer fabrication", *IEEE Trans. on Semiconductor Manufacturing*, Vol.1, No.3 pp.115-130 (1988-3)

(8) G. Habchi : "Study of the effect of process time variability on production flow of a manufacturing", *Simulation Series*, Vol.30, No.3, pp.221-226 (1998-3)

(9) Steve C. H. Lu, D. Ramaswamy, and P. R. Kumar : "Efficient scheduling policies to reduce mean and variance of cycle-time in semiconductor manufacturing plants", *IEEE Trans. on Semiconductor Manufacturing*, Vol.1, No.3 pp 374-385 (1998-3)

(10) M. T. Costa and J. S. Ferreira : "A simulation analysis of sequence rules in flexible flowline", *European Journal of Operation Research*, Vol.119, pp.440-450 (1999)

(11) U. Dorndorf and E. Pesch : "Evolution based learning in a job-shop scheduling environment", *Computers and Operation Research*, Vol. 22, No. 1, pp.25-40 (1995-1)

(12) S. Y. Kim, Y. H. Lee, and D. Agnihotri : "A hybrid approach for sequencing Jobs using Heuristic rules and neural networks", *Production Planning and Control*, Vol.6, No.5, pp.445-454 (1995-5)

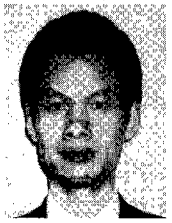
(13) R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G Rinnooy Kan : "Optimization and approximation in deterministic sequencing and scheduling: A survey", *Annals of Discrete Math.*, Vol.5, pp.287-326 (1979)

(14) E. H. L. Arts, P.J.M. Van Laarhoven, J. K. Lenstra, and N. L. J.Ulder : "A computational study of local search algorithms for job shop scheduling", *ORSA Journal on Computing*, Vol.6, No.2, pp.118-125 (1994-2)

(15) R. Nakano and T. Yamada : "Conventional genetic algorithm for job shop problem", in Proc. Of the 4th International Conference on Genetic Algorithm, San Diego, CA pp.474-479, (1991)

(16) M. Dell'Amico and M. Trubian : "Applying tabu search to the job shop scheduling problem", *Annals of Operations Research*, Vol.41, pp231-252 (1993)

Yang Jianhua (Student member) was born in Jiangsu province, China, in 1968. He received the B.S. degree from South-east University, Nanjing, China in 1990 and the M.S. degree from Tsinghua University, Beijing, China in 1993. He had worked in Automation Department of Tsinghua University from 1993 to 2000. He has been a doctorial candidate of the Department of Electrical and Computer Engineering, Yokohama National University since 2001. His research interests include flexible manufacturing system, shop-floor control and Petri Nets. Mr. Yang is a student member of the Institute of Electrical Engineers of Japan.



Yasutaka Fujimoto (Member) was born in Kanagawa prefecture, Japan, in 1971. He received B.E., M.E., and Ph.D. degrees in electrical engineering from Yokohama National University, Japan, in 1993, 1995, and 1998, respectively. In 1998, he joined the Department of Electrical Engineering, Keio University. Since 1999, he has been with the Department of Electrical and Computer Engineering, Yokohama National University, where he is currently an Associate Professor. His research interests include manufacturing automation, discrete event systems, motion control, and robotics. He is a member of IEEE and Robotics Society of Japan.