TREE BASED ALGORITHM FOR THE FLOW-SHOP MODEL TO MAXIMIZE NUMBER OF JUST-IN-TIME JOBS

NAOYA HONDA [a,*], HIROAKI ISHII [a], TERUO MASUDA [b]

Received October 1, 2004

ABSTRACT. We study the flow-shop problem nonpreemptively scheduling n jobs on m machines to maximize the number of just-in-time jobs. This network type algorithm is based on a binary tree structure. We define the superiority relation of a partial schedule, and propose an efficient rule that searches only partial schedules extensible to optimal schedules. We show that this algorithm finds an optimal schedule for this problem in a low polynomial time.

1 Introduction The paper deals with flow-shop scheduling problem to maximize the number of just-in-time jobs. Each of the n jobs J_i $(i = 1, \cdots, n)$ consists of a chain in a linear order of m operations $(O_{i1}, O_{i2}, \cdots, O_{im})$ that have to be completed exactly at their due dates. We are interested in maximizing the number of jobs completed just-in-time. We show that this problem is solvable in a low polynomial time by using a new network type algorithm.

There is now a substantial body of literature on just-in-time production systems since the publication of the first paper by Sugimori et al. in 1977 on the Toyota Production System [9]. Baker and Scudder [1] reviewed the literature on scheduling models with earliness and tardiness penalties in 1990. These authors pointed out that the single-machine scheduling problem with a restricted common due date has never been addressed in the literature. Numerous results on other related single machine JIT scheduling problems can be found in the survey paper by Li et al. [7], Davis and Kanet [2], Federgruen and Mosheiov [3], Liman and Lee [6], Li et al. [8], and Kim and Yano [4], among others. K. Hiraishi et al. [5] proposed an algorithm for the parallel identical machine problem of maximize the weighted number of just-in-time jobs, which is solvable in a polynomial time.

2 Problem formulation We consider the following permutation flow-shop problem:

- There are m machines $M_1, M_2, \ldots, M_m$, and n jobs $J_1, J_2, \ldots, J_n$ to be processed on these machines.
- At each machine, processing order is equal for all jobs, i.e., permutation schedule.
- Processing time of each job on each machine is known.
- Each job can be processed on any machine but it must be processed without preemption.
- A nonnegative due date d_{ij} is associated with each job J_i on M_j $(j = 2, 3, \ldots, n)$.
- If $d_{kj} \leq d_{lj} (k \neq l)$ holds, $d_{k(j+1)} \leq d_{l(j+1)}$ also holds due to the permutation flow-shop.

2000 *Mathematics Subject Classification.* 90B35.

Key words and phrases. Just-in-time, Flow-shop, Binary tree, Polynomial algorithm, Non-dominated solution.

We use the following notations:

t - Time.
J_i - Job i $(i = 1, 2, \ldots,)$.
M_j - Machine j $(j = 1, 2, \ldots, m)$.
p_{ij} - Processing time of job i on machine j.
d_{ij} - Due date of job i on machine j $(j = 2, 3, \ldots, n)$.
S_{ij} - Start time of job i on machine j.
C_{ij} - Completion time of job i on machine j.
π - A certain permutation schedule.
$J_{\pi(i)}$ - i-th processed job on schedule π

The objective is to maximize the number of jobs completed on at their due dates. Therefore the job that is not enough just in time on the due date is processed by postponement. We consider the problem that the due date exist in each machine except M_1. Therefore all jobs on machine M_1 is processed in no-wait.
The example of feasible schedule is shown in the following Gantt chart.

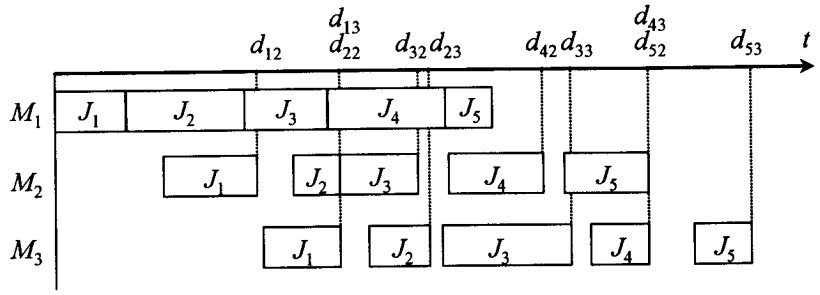

Figure 1: Feasible schedule

3 Solution algorithm At first n jobs are sorted by EDD rule. Sorted jobs are described as $J_1, J_2, \ldots, J_n$ $(d_1 \leq d_2 \leq \ldots \leq d_n)$ without any loss of generality by changing index of job if necessary. The objective is to maximize the number of just-in-time job. We define a state of i-th job as $b(i) = \{0, 1, ?\}$ $(i = 1, 2, \ldots, n)$, when $b(i) = 1$ is completed just-in-time. We define a binary sequence of length n for a schedule. The binary sequence with the maximum number of elements of g1h is the optimal schedule. The schedule is called a partial schedule that g?h is included in a sequence. For example, it is assumed that there is the following schedule. J_1, J_2 are processed just-in-time. J_3 is processed by postponement. Processing of J_4 and J_5 is not determined yet. Then that partial schedule is expressed as $\{1, 1, 0, ?, ?\}$.

Fig. 2 shows binary tree structure as a method to enumerate $\{0, 1\}$ sequence. In Fig. 2, there is one partial schedule in each node. In node A, J_1 is processed just-in-time, J_2 processed just-in-time, too, J_3 processed by postponement, and J_4, J_5, $\cdots$, J_n not fixed. This partial schedule is expressed as $\{1, 1, 0, ? \ldots ?\}$. A enumeration of all schedules is expressed by node expansion with such a binary tree structure. Fig. 3 shows the new network structure compressed from tree structure in Fig. 2. Each level means the number of just-in-time jobs. Set of partial schedules is attached to each node. Label in a node shows the number of the contained in it partial schedule. For example, in node D shown

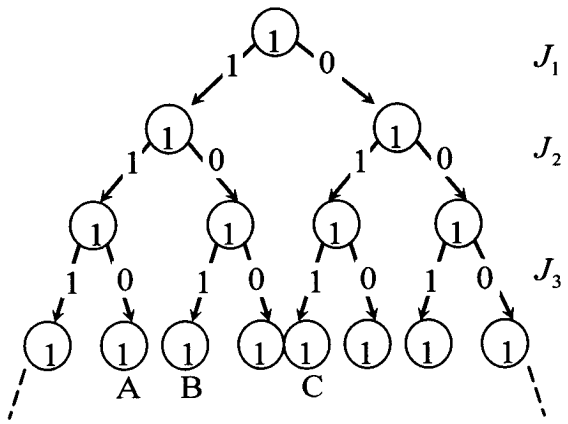

Figure 2: Binary tree structure

in Fig. 3, there are three partial schedules with two just-in-time jobs among J_1, J_2 and J_3. That partial schedule is expressed as $\{0,1,1,?\ldots?\}$, $\{1,0,1,?\ldots?\}$, $\{1,1,0,?\ldots?\}$. These partial schedules correspond to three nodes A,B,C shown in Fig. 2.

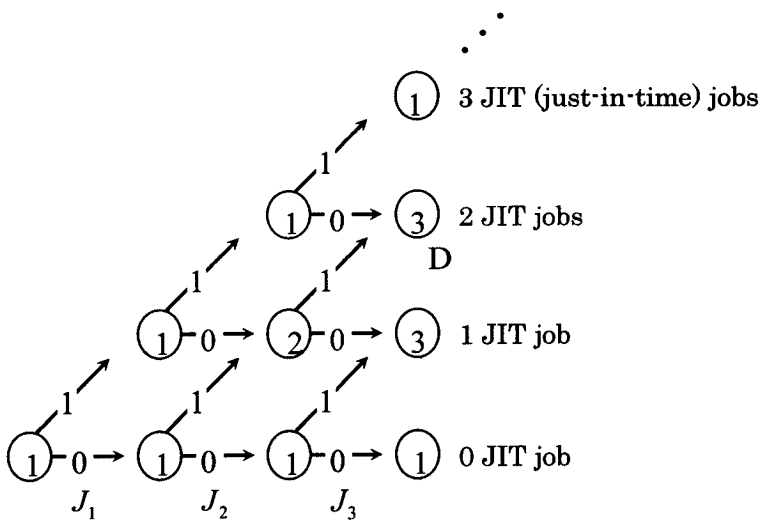

Figure 3: Network model transformed from binary tree

Fig. 4 shows the network model with labeled nodes and arcs. We define the notation used in Fig. 4.

a_{pq} - Node at the level p in phase q. There are partial schedules with p just-in-time jobs among $J_1 \ldots J_q$.

p - Level p of a vertical direction $(p \leq q \,|p = 0, 1, \ldots, n)$.

q - Phase q of a horizontal direction $(p \leq q \,|q = 0, 1, \ldots, n)$.

X_{pq} - Arc from $a_{p(q-1)}$ to a_{pq} $(p \neq q)$.

Y_{pq} - Arc from $a_{(p-1)(q-1)}$ to a_{pq} $(p \neq 0)$.
π_R^{pq} - A certain partial schedule in node a_{pq}.
C_{Rj}^{pq} - Completion time of the last just-in-time job on machine M_j in the partial schedule π_R^{pq}.
$\pi_R^{pq \to \mathrm{JIT}}$ - A certain partial schedule in node a_{pq} where J_{q+1} can be processed just-in-time.
$\Pi^{pq \to \mathrm{JIT}}$ - A set of partial schedules in node a_{pq} where J_{q+1} can be processed just-in-time.
$C_{Rj}^{pq \to \mathrm{JIT}}$ - Completion time of the last just-in-time job on M_j in the partial schedule $\pi_R^{pq \to \mathrm{JIT}}$.

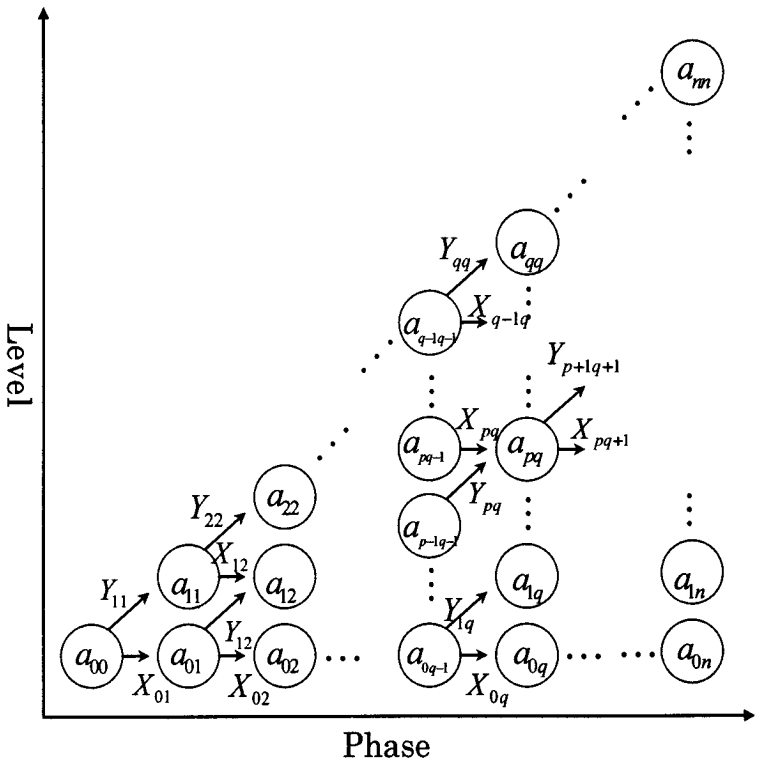

Figure 4: Network model

Rule of node expansion:

In node expansion X_{pq} (from $a_{p(q-1)}$ to a_{pq}), J_q is not processed just-in-time for all partial schedules in $a_{p(q-1)}$. J_q is processed by postponement. Therefore, $b(q) = 0$ is set for all partial schedules to satisfy

$$\sum_{i=1}^{q-1} b(i) = p,\ b(q) = b(q+1) = \cdots = b(n) = ?$$

In node expansion Y_{pq} (from $a_{(p-1)(q-1)}$ to a_{pq}), J_q is is processed just-in-time for all partial schedules in $a_{p(q-1)}$. Therefore, $b(q) = 1$ is set for all partial schedules to satisfy

$$\sum_{i=1}^{q-1} b(i) = p-1,\ b(q) = b(q+1) = \cdots = b(n) = ?$$

Thus, the enumeration of all the schedules can be expressed by node expansion from phase 1 to phase q. The schedule which satisfies $\max \sum_{i=1}^{n} b(i)$ is the optimal schedule in the combination of all schedules. A characteristic of network structure is shown in Fig. 4, there are partial schedules with the same numbers of just-in-time jobs in each node. When the node is expanded to the last phase, a combination of all schedules is expressed. However, when n is large, it is unrealistic to expand all node because There exists 2^n schedule combinations. We suggest improvement rule of node expansion. Only a node with optimal possibility is expanded, but the rest are cut off.

Definition 1 *Domination of a partial schedule*

In node a_{pq}, π_L^{pq} is superior to π_R^{pq} when corresponding m-dimensional vectors $(C_{L1}^{pq}, \ldots, C_{Lm}^{pq})$, $(C_{R1}^{pq}, \ldots, C_{Rm}^{pq})$ satisfy

$$C_{Lj}^{pq} \leq C_{Rj}^{pq} \quad (j = 1, \ldots, m), \quad \pi_L^{pq} \neq \pi_R^{pq}$$

In this case we say π_L^{pq} dominates π_R^{pq}.

Proposition 1

In the above case, even if partial schedule π_R^{pq} is deleted, an optimal schedule can be obtained.

Proof. Each schedule in node a_{pq} has a common property that $\sum_{i=1}^{q} b(i) = p$ and $b(q+1) = b(q+2) = \cdots = b(n) = ?$. A different point is combination of $\{b(1), b(2), \ldots, b(q)\}$. Processed job just-in-time from among $J_{q+1}, J_{q+2}, \ldots, J_n$ is influenced by the complete time of a last just-in-time job in $J_1, J_2, \ldots, J_q$ since all jobs are sorted in an EDD rule. If an optimal schedule is expanded from π_R^{pq}, the partial schedule after J_{q+1} is feasible also in π_L^{pq} shown in Fig. 5. This completes the proof.

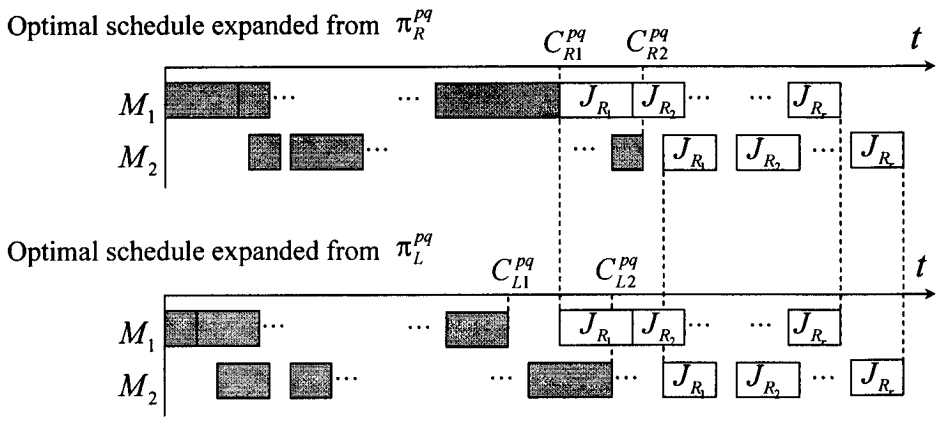

Figure 5: Optimal schedule expanded from π_R^{pq} and π_L^{pq}

Proposition 2

In node expansion Y_{pq} (from $a_{(p-1)(q-1)}$ to a_{pq}), only one partial schedule is expanded which satisfy $\min_{R} \left\{ C_{R1}^{(p-1)(q-1)\to \mathrm{JIT}} \middle| \pi_R^{(p-1)(q-1)\to \mathrm{JIT}} \in \Pi^{(p-1)(q-1)\to \mathrm{JIT}} \right\}$.

Proof. If J_q is processed in each schedule $\pi_R^{(p-1)(q-1)\to \mathrm{JIT}} \in \Pi^{(p-1)(q-1)\to \mathrm{JIT}}$, completion time of each schedule on $M_2, \ldots, M_m$ is same, but completion time on M_1 is different as shown in Fig. 6. According to proposition 1, we can obtain optimal schedule even if schedules which is not satisfy $\min_{R} \left\{ C_{R1}^{(p-1)(q-1)\to \mathrm{JIT}} \right\}$ are cut off. This completes the proof.

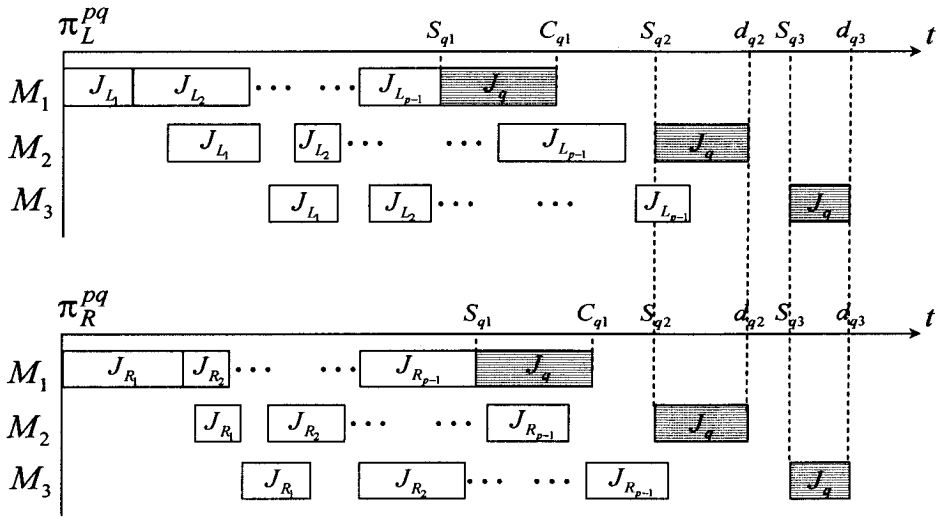

Figure 6: Node expansion in Y_{pq}

Maximum number of expanded partial schedule in each node is shown in Fig. 7.

Number of partial schedules expanded in node a_{pq} is $q - p + 1$ at the most. Expanded schedule in node a_{0q} is one. Total maximum number of expanded partial schedule in phase q $(a_{0q}, a_{1q}, \ldots, a_{pq}, \ldots, a_{qq})$ is

$$1 + \sum_{p=1}^{q} (q - p + 1) = \frac{1}{2} q (q + 1) + 1$$

Grand total maximum number of expanded partial schedules of the whole graph is

$$\sum_{q=1}^{n} \left(1 + \sum_{p=1}^{q} (q - p + 1) \right)$$

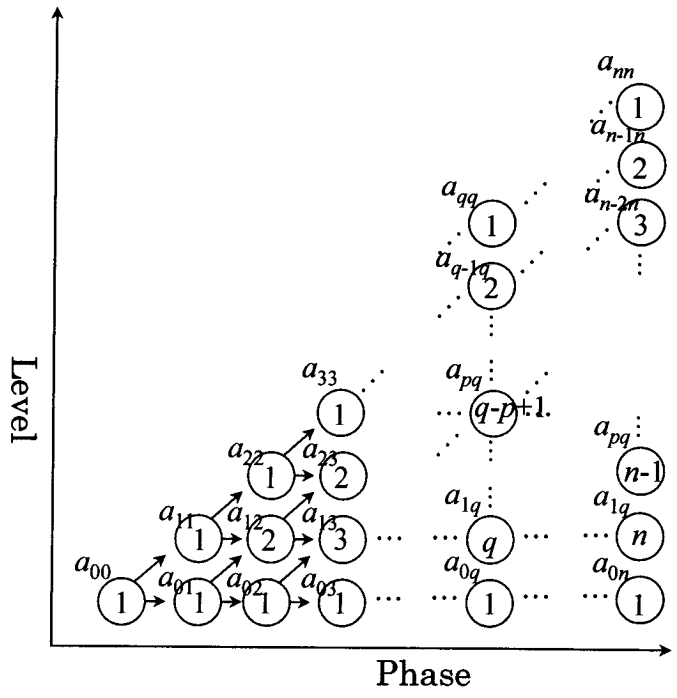

Figure 7: Maximum number of expanded partial schedule in each node

Thus, we obtained the following argorithm.

Step 1 - Initialization. Jobs are sorted in EDD rule as $J_1, J_2, \ldots, J_n$ $(d_{1m} \leq d_{2m} \leq \ldots \leq d_{nm})$. Node a_{00} is empty. Set Phase $q = 0$, Level $p = 0$, Loop $r = 0$.

Step 2 - Expansion. All partial schedules in node a_{pq} are expanded to node $a_{p(q+1)}$ that J_{q+1} is not processed just-in-time. If $\pi_R^{pq \to \mathrm{JIT}}$ exists, one partial schedule satisfying $\min_{\mathrm{R}} \left\{ C_{R1}^{pq \to \mathrm{JIT}} \right\}$ is expanded to node $a_{(p+1)(q+1)}$ that J_{q+1} is processed just-in-time. Set Phase $q = q + 1$, Level $p = p + 1$.

Step 3 - If $q = n$ then go to Step 4. Otherwise go to Step 2.

Step 4 - If a_{pq} is not empty then go to Step 5. Otherwise set Loop $r = r + 1$, Phase $q = r$, Level $p = 0$, and go to Step 2.

Step 5 - Termination. Optimal schedule is obtained in node a_{pq}.

Finally, we estimate its computational complexity. In the Step 1, sorting is required and it takes $O(n\log n)$. Number of expanded partial schedules in a_{pq} is $q - p + 1$ at most. Therefore computation order of our algorithm is as follows.

$$O\left(n\log n + \sum_{q=1}^{n} \left\{ 1 + \sum_{p=1}^{q} (q - p + 1) \right\}\right) = O\left(n^3\right)$$

4 Conclusions In this paper, we have proposed a polynomial time algorithm for the permutation flow-shop to maximize the number of just-in-time jobs. Our algorithm is based on complete binary tree search, but the number of the expanded partial schedules is

minimum. We has proposed a search algorithm with new network structure. We has shown that calculation order of our algorithm is $O(n^3)$. If most of the expansion schedules are not cut off, optimal schedule is found with an early stage.

REFERENCES

[1] K.R. Baker and G.D. Scudder, Sequencing with earliness and tardiness penalties: A review. *Operations Research* 38 (1990) 22-36

[2] J.S. Davis, J.J. Kanet, Single-machine scheduling with early and tardy completion costs, *Naval Research Logistics* 40 (1993) 85-101.

[3] A. Federgruen, G. Mosheiov, Simultaneous optimization of efficiency and performance balance measures in singlemachine scheduling problems, *Naval Research Logistics* 40 (1993) 951-970.

[4] Y.-D. Kim and C.A. Yano, Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates, *Naval Research Logistics* 41 (1994) 913-933.Japan. (2000)

[5] H. Kunihiko, L. Eugene and V. Milan. Scheduling of parallel identical machines to maximize the weighted number of just-in-time jobs. *Computers and Operations Research* 29 (2002) 841-848

[6] S.D. Liman, C.Y. Lee, Error bound of a heuristic for the common due date scheduling problem, *ORSA Journal on Computing* 5 (1993) 420-425.

[7] Chung-Lun Li, T.C.E. Cheng and Z.-L. Chen, A note on one-processor scheduling with asymmetric earliness and tardiness penalties, *Operations Research Letters* 13 (1993) 45-48.

[8] Chung-Lun Li, T.C.E. Cheng and Z.-L. Chen, Single-machine scheduling to minimize the weighted number of early and tardy aggreeable jobs, *Computers and Operations Research* 22 (1995) 205-219.

[9] Y. Sugimori, K. Kusunoki, F. Cho. and S. Uchikawa, Toyota production system and kanban system materialization of JIT and Respect-For-Human system. *International Journal of Production Research* 15 (1977) 553-564.

[a] Graduate School of Information Science and Technology, Osaka University, 2-1 Yamadaoka, Suita, 565-0871, Japan
e-mail: honda@ist.osaka-u.ac.jp, ishii@ist.osaka-u.ac.jp

[b] Faculty of Business Administration, Tezukayama University, 7-1-1 Tezukayama, Nara, 631-8501, Japan
e-mail: masuda@tezukayama-u.ac.jp