Journal of the Operations Research Society of Japan
Vol. 44, No. 2, June 2001

A SHIFTING BOTTLENECK APPROACH FOR A PARALLEL-MACHINE FLOWSHOP SCHEDULING PROBLEM

Jinliang Cheng Yoshiyuki Karuno Hiroshi Kise
Kyoto Institute of Technology

(Received April 10, 2000; Revised December 13, 2000)

Abstract This paper considers a scheduling problem of minimizing the maximum lateness for a parallel-machine flowshop with m stages, each of which consists of one or more identical parallel machines. We propose a heuristic algorithm being based on a shifting bottleneck approach which decomposes the parallel-machine flowshop problem into m parallel-machine scheduling problems to be solved one by one. Each parallel-machine problem is approximately solved by applying a property of its reversibility in the proposed heuristic. To evaluate performance of the proposed heuristic, it is numerically compared with Wittrock's algorithm for a real production line, and with Santos *et al.*'s global lower bound for test problem instances randomly generated. The results obtained strongly suggest that the proposed heuristic produces optimal or quite near optimal solutions within short computational time with high probability.

1. Introducton

This paper considers a scheduling problem for a parallel-machine flowshop (denoted by PMFS) consisting of m stages, each of which is composed of one or more identical machines. A flowshop with multiple processors (e.g., Brah and Hunsucker [4], Brah and Loo [5], Rajendran and Chaudhuri [25], Santos *et al.* [28]), a hybrid flowshop (e.g., Aghezzaf and Artiba [2], Guinet and Solomon [10], Gupta [11], Gupta *et al.* [12], Gupta and Tunc [13], Lee and Vairaktarakis [19], Riane *et al.* [26]) and a flexible flow line (e.g., Kochhar and Morris [18], Leon and Ramamoothy [21], Sriskandarajah and Sethi [30], Wittrock [31, 32]) belong to the same class of the PMFS.

The PMFS is a generalization of the traditional flowshop model with only one machine at each stage and a generalization of a parallel-machine shop with a single stage, and thus scheduling problems for the PMFS are more intractable. In fact, the problem of minimizing the makespan for the PMFS is NP-hard even in special cases such as a two-stage case with a single machine at a stage (see Gupta [11]) and a two-stage with preemption allowed (see Hoogeveen *et al.* [14]).

One of the earliest papers for the PMFS is Arthanary and Ramaswarmy [3] where a two-stage problem was discussed. Since then, there have been a lot of papers for two- or three-stage problems (e.g., Chen [8], Deal and Hunsucker [9], Gupta [11], Gupta *et al.* [12], Gupta and Tunc [13], Hoogeveen *et al.* [14], Riane *et al.* [26], Seetharama [29], Sriskandarajah and Sethi [30]).

Wittrock [31, 32] has developed heuristic algorithms for general m-stage problems of minimizing the makespan and reducing work-in-processes (WIP's) for real production lines in IBM. Kochhar and Morris [18] discussed problems of minimizing the effect of setup

time, blocking with finite buffers and down time. Hunsucker and Shah [15] developed a dynamic simulation model for a PMFS to evaluate priority rules for some measures such as the makespan, the mean flow time, the maximum flow time, the mean tardiness and the number of tardy jobs. Santos *et al.* [28] developed a global lower bound for a general makespan problem to assess the quality of heuristic solutions when the optimal one is unknown. In fact, branch-and-bound algorithms developed so far could solve only small size problem instances with typically two-stage and up to ten jobs (e.g., Brah and Loo [5], Deal and Hunsucker [9], Gupta and Tunc [13], Rajendran and Chaudhuri [24, 25]). Leon and Ramamoothy [21] proposed a problem-space-based neighborhood for local seach which does not mean to perturb the current solution, but to perturb problem data for generating new solutions. Brah and Loo [5] investigated five better performing heuristics for their performance of the makespan and the mean flow time criteria using regression analysis. There have been reports on practical applications to problems of the PMFS (e.g., Iima and Sannomiya [16], Luss and Rosenwein [22], Paul [23], Wittrock [31, 32]).

We discuss the parallel-machine flowshop scheduling problem of minimizing the maximum lateness (denoted by PMFSP), and propose an efficient heuristic algorithm for the PMFSP. It adopts the basic idea of the so-called *shifting bottleneck procedure* (SBP, for short) that was originally developed for the classical jobshop scheduling problem by Adams *et al.* [1]. The SBP sequences the machines one by one, successively, at each time taking the machine identified as a bottleneck among the ones not yet sequenced. For this purpose and getting feasible solutions the SBP optimally solves a lot of one-machine scheduling problems by a quite effective branch-and-bound algorithm. But this is not the case for the parallel-machine scheduling problem (denoted by PMSP) with which we have to be confronted in the PMFSP, because there is no such an effective exact algorithm. Thus, our heuristic utilizes effective lower bounds instead of optimal values for the bottleneck identification.

For each PMSP, our heuristic utilizes the reversibility of the problem. That is, each PMSP has the corresponding reverse PMSP that is different from the original one. They have the same optimal value in sequences reversed each other. This means that a heuristic applied to the original and its reverse PMSP yields different approximate solutions, thus we can easily choose the better solution. This reversibilty generalizes that for the one-machine scheduling problem (see Kise *et al.* [17]).

The remainder of this paper is organized as follows. Section 2 formulates the PMFSP through a disjunctive graph. Section 3 presents a heuristic algorithm for the PMFSP, being based on a shifting bottleneck approach that decomposes the PMFSP into m PMSP's, and describes the reversibility of the PMSP. The proposed heuristic consists of two phases, getting a feasible solution in Phase One and reoptimizing in Phase Two. Section 4 shows the results of numerical experiments executed to evaluate the performance of the proposed heuristic. It is compared with Wittrock's algorithm for his bench mark problem instances (see Wittrock [32]) and with Santos *et al.*'s global lower bound for their test problem instances randomly generated (see Santos *et al.* [28]). The results obtained strongly suggest that the proposed heuristic produces optimal or quite near optimal solutions within short computational time with high probability.

2. Problem Formulation

2.1. Description of the model

The parallel-machine flowshop scheduling problem (PMFSP) considered here can be described as follows:

(1) n parts are processed by a m-stage production system. Each stage consists of one or more identical parallel machines and has sufficient capacity of buffer storage for work-in-processes (WIP's).

(2) Each part is processed by at most one machine in each stage, i.e., each part consists of at most m tasks and the tasks of a part are processed at different stages each other. All the parts visit the m stages in the same order of them, but some parts are allowed to skip some stages, if necessary.

(3) The processing times of tasks are known, and does not depend on machines to be used. The setup time is independent of the part sequence, and therefore is included in the processing time.

(4) No machine can process more than one part at a time, and no preemption is permitted.

(5) Parts are continuously transported from one stage to another (e.g., by belt conveyer), and the transportation time from one machine to another is simply a function of the two stages to which the two machines to be used belong.

(6) Each part has a known ready time, i.e., its processing cannot be started before this time. The ready time may represent the part arrival time in the system.

(7) Each part has a known due date, and the minimization of the maximum lateness is sought. When the maximum lateness takes a positive value, then it is tardiness; when the due date is zero, then the maximum lateness is identical to the maximum completion time (i.e., the makespan).

An example of the parallel-machine flowshop with three machine stages and four parts is illustrated in Figure 1. Part 1 and part 2 can skip stage 2 and stage 3, respectively, in this example.

2.2. Notations

The following notations are used to formulate problem PMFSP:

(1) Input parameters:

- $J = \{1, 2, \ldots, n\}$: the set of n parts;
- $M = \{0, 1, \ldots, m, m+1\}$: the set of $m+2$ stages including the input buffer stage 0 and the output buffer stage $m+1$ (see Figure 1);
- $M^k = \{m_{kl} \mid l = 1, 2, \ldots, h(k), k = 1, 2, \ldots, m\}$: the set of $h(k)$ identical machines composing stage k, where m_{kl} is the l-th machine at stage k with buffer B_{kl};
- J_j: the set of tasks composing part $j \in J$;
- J^k: the set of tasks to be processed at stage k ($k = 1, 2, \ldots, m$). Note that $|J_j \cap J^k| \leq 1$ holds;
- $s(u)$: the stage at which task $u \in \cup_{j=1}^{n} J_j$ is to be processed;
- r_j, d_j: ready time and due date of part $j \in J$, respectively;
- $p(u)$: processing time of task $u \in \cup_{j=1}^{n} J_j$;

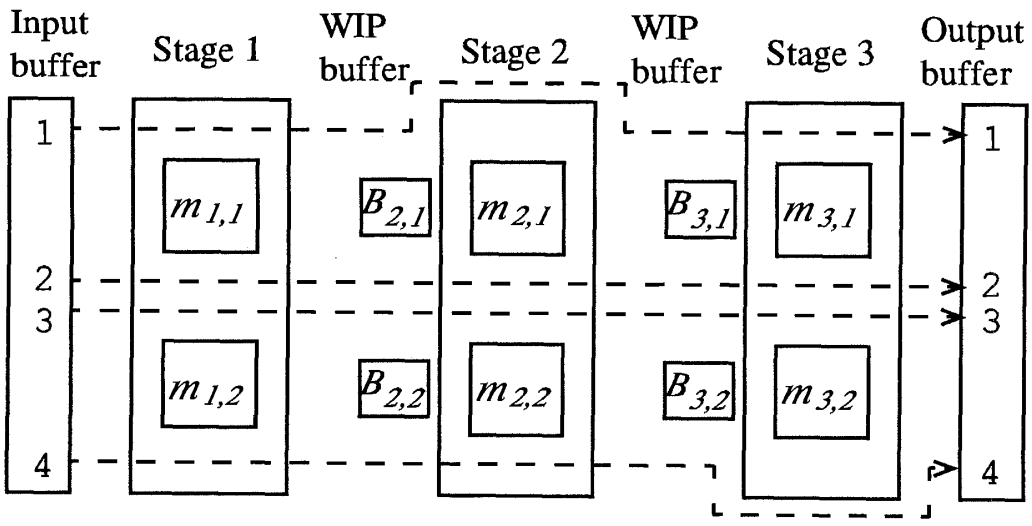


Figure 1: An example of the parallel-machine flowshop.

- $t_{kk'}$: transportation time of a part from stage k to stage k' ($k, k' \in M$ and $k < k'$).

(2) Parts routing

- $\Omega = \Omega^1 \times \Omega^2 \times \cdots \times \Omega^m$: a set of parts routings, each of which assigns each task u of the parts to one of machines at stage $s(u)$, where Ω^k stands for allocating tasks to be processed at stage k to $h(k)$ machines, i.e., let $\omega^k \in \Omega^k$ and $\omega \in \Omega$, then

$$\omega^k = \omega^{k1} \times \omega^{k2} \times \cdots \times \omega^{k,h(k)},$$

$$\cup_{l=1}^{h(k)} \omega^{kl} = J^k,$$

$$\omega^{kl} \cap \omega^{kl'} = \emptyset, \quad l, l'(\neq l) = 1, 2, \ldots, h(k),$$

$$\omega = \omega^1 \times \omega^2 \times \cdots \times \omega^m.$$

(3) Disjunctive graph

- $G = (N, A, D, P, T)$: disjunctive graph representing a scheduling problem under a parts routing ($\omega \in \Omega$);
- $N_j = \{n_j^0, n_j^0 + 1, \ldots, n_j^*\}$: the set of nodes, each representing a task of part j where n_j^0 and n_j^* represent the ready time and the due date of the part, respectively. The ready time node (number) and the due date node (number) of part j are given by $n_j^0 = \sum_{j'=1}^{j-1}(|J_{j'}| + 2) + 1$ and $n_j^* = \sum_{j'=1}^{j}(|J_{j'}| + 2)$, respectively;
- $N = \{0, *\} \cup \{N_j \mid j \in J\}$: the set of nodes, each representing a task except that source 0 and sink $*$ represent the start and the finish of the schedule, respectively;
- $A_j = \{(u, u+1) \mid n_j^0 \leq u < n_j^*\}$: the arc set representing task's order of part j;
- $A = \{A_j \mid j = 1, 2, \ldots, n\}$: the set of conjunctive arcs;
- $D^{kl} = \{(u, v), (v, u) \mid u, v(\neq u) \in \omega^{kl}\}$: the set of disjunctive arc pairs where each disjunctive arc represents the processing order between the tasks on the l-th machine at stage k;

- $D^k = \{D^{kl} \mid l = 1, 2, \ldots, h(k)\}$: the set of disjunctive arc pairs at stage k;
- $D = \{D^k \mid k = 1, 2, \ldots, m\}$: the set of disjunctive arc pairs;
- $P = \{p(u) \mid u \in \cup_{j=1}^{n}(N_j - \{n_j^0, n_j^*\}) - \{0, *\}\} \cup \{p(0) = 0, p(*) = 0\} \cup \{p(n_j^0) = 0, p(n_j^*) = 0 \mid j \in J\}$: the set of node weights representing the processing times of tasks;
- $T = \{t(u,v) \mid (u,v) \in A\} \cup \{t(u,v) = 0 \mid (u,v) \in D\}$: the set of arc weights where $t(0, n_j^0) = r_j$, $t(n_j^*, *) = -d_j$ and $t(u,v) = t_{s(u)s(v)}$.

(4) Schedule

- $S_l^k = \{(u,v) \mid u, v$ are nodes which correspond to two tasks assigned to machine $m_{kl}\}$: a subsequence for machine m_{kl}, i.e., a set of disjunctive arcs selected for scheduling, meaning that its counterpart is discarded;
- $S^k = \{S_l^k \mid l = 1, 2, \ldots, h(k)\}$; a subschedule for stage k $(k = 1, 2, \ldots, m)$;
- $S = \{S^k \mid k = 1, 2, \ldots, m\}$: a (whole) schedule of m stages;
- Π: the set of schedules S under a parts routing $(\omega \in \Omega)$;
- $G_S = (N, A \cup S, \emptyset, P, T)$: a conjunctive graph, called *scheduled graph*, corresponding a schedule $S \in \Pi$, where $\emptyset$ means that no disjunctive arc pair exists;
- $L(u,v)$: the longest path length to node v from node u in scheduled graph G_S;
- $E(v) \triangleq L(0,v)$: the earliest start time of task v.

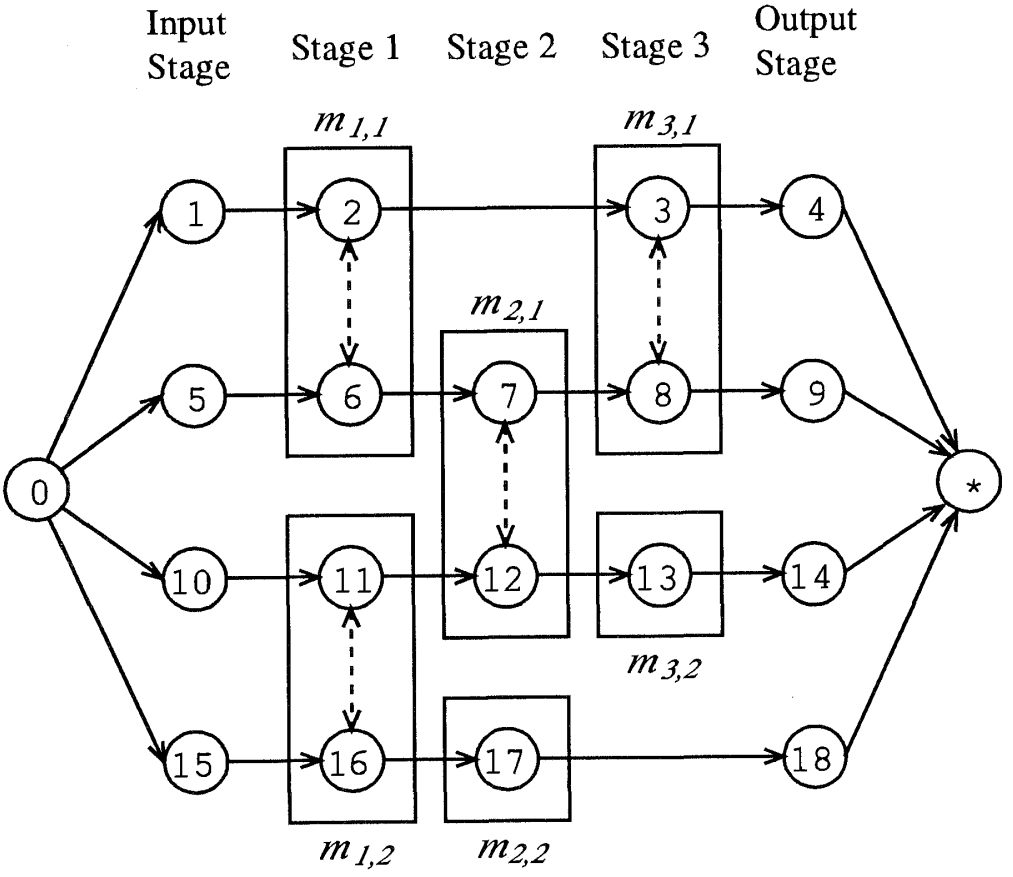


Figure 2: An example of the disjunctive graph.

2.3. Formulation of the scheduling problem

Suppose that a parts routing $\omega \in \Omega$ is given, then $G = (N, A, D, P, T)$ is the corresponding disjunctive graph defined in Subsection 2.2. Figure 2 illustrates a disjunctive graph for a 3-stages, 4-parts problem (see Figure 1, also), where solid and dotted arcs represent the conjunctive and the disjunctive arcs, respectively. Obviously, a sequence of tasks assigned to each machine is determined by selecting one arc from each disjunctive arc pair, and discarding the other. Figure 3 illustrates an example of the scheduled graph, which is generated from Figure 2.

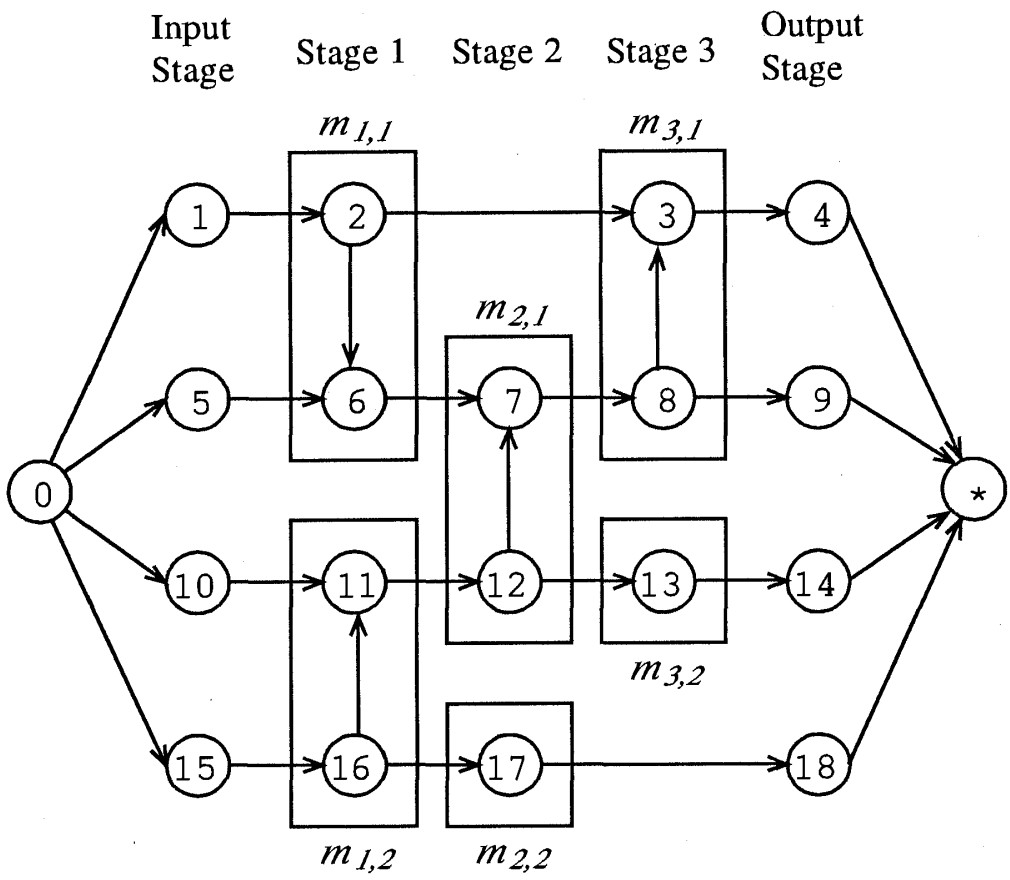


Figure 3: An example of the scheduled graph.

Given scheduled graph G_S, since $E(v) \triangleq L(0, v)$ is the longest path length to node v from source 0, it is defined by

$$E(v) = \max_{u \in B(v)} \{E(u) + p(u) + t(u, v)\}, \tag{1}$$

where $B(v) = \{u | (u, v) \in A \cup S\}$ and $E(0) = 0$. As defined in Subsection 2.2, $E(v)$ represents the earliest start time of task v. Especially, $E(n_j^0) = 0$; $E(n_j^0 + 1)$ and $E(n_j^*)$ are earliest starting and the earliest finishing times of processing part j, respectively. The lateness of part j is defined by

$$L_j = E(n_j^*) + p(n_j^*) + t(n_j^*, *) = E(n_j^*) - d_j,$$

and the maximum lateness of schedule S is by

$$L_{max} = E(*) = \max_{j \in J} L_j.$$

Thus the parallel-machine flowshop scheduling problem can be expressed by

$$\text{PMFSP}: \quad \min_{w \in \Omega, S \in \Pi} L_{max} \quad \text{s.t. the equation of (1).} \tag{2}$$

3. A Shifting Bottleneck Based Heuristic

The shifting bottleneck approach we employ is based on the idea of decomposition of the PMFSP into m subproblems to be solved one by one. The subproblem at each stage becomes a parallel-machine scheduling problem with ready times and due dates. The bottleneck concept plays an important role in such a heuristic approach, and a good solution on the bottleneck stage is expected to result in a good schedule on the whole. Adams *et al.* [1] first developed a shifting bottleneck procedure for the jobshop problem. Since then, it has been generalized to solve various types of scheduling problems (e.g., see Rumudhin and Marier [27]). Its key point to score a success is to exploit a quite efficient branch-and-bound procedure developed by Carlier [6] to solve decomposed one-machine scheduling problems exactly and to use their optimal velues as a measure of bottleneck quality associated with each machine.

Unfortunately, there has not been such a highly efficient exact algorithm for our parallel-machine scheduling problem so far. Thus, we solve it approximately and utilize a lower bound for the bottleneck quality.

The proposed heuristic algorithm consists of two phases. Phase One schedules m stages one by one successively in descending order of lower bounds of the parallel-machine scheduling problems to obtain an initial schedule. Phase Two locally reoptimizes each stage, being based on schedules of the remaining $(m-1)$ stages that have been the best ones obtained so far. This procedure is repeated until no improvement is possible.

3.1. Parallel-machine problem

Assume that a proper subset K of stages already has schedule $S^K = \{S^{k'} | k' \in K\}$, then a parallel-machine scheduling problem for task set J^k at stage $k \notin K$ (denoted by PMSP$(k|K)$ is defined as follows:

$$\text{PMSP}(k|K): \quad \{G = (N, A_K, D^k, P, T), \text{ for all possible } \Omega_k\}, \tag{3}$$

where $A_K = A \cup \{S^{k'} | k' \in K\}$. Let $G_{S^K} = (N, A \cup S^K, \emptyset, P, T)$ be the conjunctive graph defined by S^K, and let $L^K(u, v)$ be the longest path length from node u to node v in graph G_{S^K}, then PMSP$(k|K)$ is reduced to a parallel-machine scheduling problem of minimizing the maximum lateness, i.e., $P|r_j|L_{max}$, according to the standard classification scheme (e.g., see Lenstra *et al.*, 1977), of which ready time, processing time and due date of task $u \in J^k$ are respectively defined by

$$\begin{cases} r_k(u) = L^K(0, u), \\ p_k(u) = p(u), \\ d_k(u) = L^K(0, *) - L^K(u, *) + p(u). \end{cases} \tag{4}$$

Moreover, let

$$q_k(u) = L^K(0, *) - d_k(u), \tag{5}$$

then the PMSP is also equivalent to a parallel-machine scheduling problem with head $r_k(u)$ and tail $q_k(u)$ (which correspond to ready time and delivery time, respectively).

Obviously, a feasible solution of a PMSP($k|M - \{k\}$) leads to a feasible one of PMFSP with the same objective value.

Define a parallel-machine scheduling problem for stage k by graph $G = (N, A, \emptyset, P, T)$ obtained by removing all the disjunctive arcs, i.e., all stages are not yet scheduled, then calculate ready times $r_k(u)$, due dates $d_k(u)$, and delivery times $q_k(u)$ by using equations (4) and (5), and arrange $r_k(u)$, $q_k(u)$ in ascending order and $d_k(u)$ in descending order. Let SR be the sum of the first $h(k)$ ready times in the ascending order, SQ the sum of the first $h(k)$ delivery times in the ascending order, and SD the sum of the first $h(k)$ due dates in the descending order, then Carlier's result is stated as follows:

Lemma 1 [7] *A lower bound on the optimal makespan for the delivery time form of the parallel-machine scheduling problem with ready times $r_k(u)$ and delivery times $q_k(u)$ is given by*

$$LB(k) = \frac{SR + \sum_{u \in J^k} p(u) + SQ}{h(k)}. \tag{6}$$

The following is straightforward from Lemma 1 and (5):

Lemma 2 *A lower bound on the optimal maximum lateness for the parallel-machine scheduling problem with release times $r_k(u)$ and due dates $d_k(u)$ is given by*

$$LB(k) = L^{\emptyset}(0, *) + \frac{SR + \sum_{u \in J^k} p(u) - SD}{h(k)}. \tag{7}$$

Suppose that a schedule S and the corresponding scheduled graph $G = (N, A \cup S, \emptyset, P, T)$ are given. We say that the stage k is critical with respect to S if stage k has some tasks on a critical path. In order to identify a bottleneck stage in graph $G = (N, A \cup S, \emptyset, P, T)$ from a different view of bottleneck concept, we measure the amount of contribution $CT(k)$ of stage k to the maximum lateness.

Now, let C_{kl} be the set of tasks on the longest paths that are processed on machine m_{kl} of stage k, and $T(k, l) = \sum_{u \in C_{kl}} p(u)$. Then, we define the maximum sum of $T(k, l)$ among various machines of stage k as $CT(k)$, i.e.,

$$CT(k) = \max_{1 \le l \le h(k)} T(k, l). \tag{8}$$

In the local reoptimization of Phase Two in the propose heuristic which will appear in Subsection 3.3, we reoptimize the sequence of each critical stage in descending order of $CT(k)$.

3.2. Parallel-machine scheduling

It is well known that the parallel-machine scheduling problem of minimizing the maximum lateness with ready times and due dates is NP-hard in the strong sense (e.g., see Lenstra *et al.* [20]). This implies that a polynomially bounded algorithm to optimally solve the parallel-machine problem is probably impossible. Thus, we propose an approximate algorithm which is based on the following algorithm H and on the reversibility of the problem:

Algorithm H: At each time point (t), a machine becomes idle again, select a task with the smallest due date from among all the tasks u that are available $(r(u) \leq t)$, but not yet scheduled, breaking ties by preferring longer processing times (breaking ties arbitrarily, if the processing times are also the same).

Reversibility of the PMSP: We here omit the indexes k and K for simplicity unless confusion occurs. Let a parallel-machine scheduling problem be denoted by PMSP $= (\mathcal{R}, \mathcal{P}, \mathcal{D})$, where $\mathcal{R}, \mathcal{P}$ and $\mathcal{D}$ be respectively ordered sets of ready times, processing times and due dates, then problem $\overline{\text{PMSP}} = (-\mathcal{D}, \mathcal{P}, -\mathcal{R})$ is referred to as the reverse of PMSP, where $-\mathcal{D} = (-d(1), -d(2), \ldots, -d(n))$ and $-\mathcal{R} = (-r(1), -r(2), \ldots, -r(n))$ be respectively ordered sets of ready times and due dates. Furthermore, let $\pi = (\pi_1, \pi_2, \ldots, \pi_g)$ be a sequence of g tasks to be processed on a machine, where π_i stands for the i-th task, then $\overline{\pi} = (\pi_g, \pi_{g-1}, \ldots, \pi_1)$ is referred to as the reverse of π, and let $\mathcal{S}$ be a set of sequences for all machines which is called schedule, then $\overline{\mathcal{S}}$ is referred to as the reverse of schedule $\mathcal{S}$, if $\overline{\mathcal{S}}$ consists of the reverse sequences of $\mathcal{S}$.

Theorem 1 *The parallel-machine scheduling problem* PMSP *and its reverse* $\overline{\text{PMSP}}$ *have the same minimum value of the maximum lateness, and their optimal schedules are reverse each other.*

Proof: Let $S_{kl} = (u_l(1), u_l(2), \cdots, u_l(q_l))$ be a schedule on machine m_{kl} of stage k, and let $S_k = \bigcup_{l=1}^{h(k)} S_{kl}$ be a schedule at stage k. Then, the finishing time $f(u_l(i))$ of task $u_l(i)$ is given as follows:

$$\begin{aligned} f(u_l(1)) &= r(u_l(1)) + p(u_l(1)), \\ f(u_l(i)) &= \max\{f(u_l(i-1), r(u_l(i))\} + p(u_l(i)) \\ &= \max_{1 \leq j \leq i}\{r(u_l(j)) + \sum_{h=j}^{i} p(u_l(h))\}. \end{aligned} \tag{9}$$

Let the maximum lateness on machine m_{kl} be defined by

$$L(S_{kl}, \text{PMSP}) = \max_{1 \leq i \leq q_l}\{f(u_l(i) - d(u_l(i))\}, \quad l = 1, 2, \cdots, h(k), \tag{10}$$

then the maximum lateness of PMSP is given by

$$L(S_k, \text{PMSP}) = \max_{1 \leq l \leq h(k)} L(S_{kl}, \text{PMSP}). \tag{11}$$

On the other hand, consider the reverse problem $\overline{\text{PMSP}}$. For $S_{kl} = (u_l(1), u_l(2), \cdots, u_l(q_l))$, $\overline{S}_{kl} = (\overline{u}_l(1), \overline{u}_l(2), \cdots, \overline{u}_l(q_l)) = (u_l(q_l), u_l(q_l - 1), \cdots, u_l(1))$ is called the reverse of

S_{kl}, and $\overline{S}_k = \cup_{l=1}^{h(k)} \overline{S}_{kl}$ is called the reverse of S_k. Then, we have

$$\begin{aligned} L(S_{kl}, \overline{\text{PMSP}}) &= \max_{1\le i\le q_l}\Big\{\max_{1\le j\le i}[-d(u_l(j)) + \sum_{h=j}^{i} p(u_l(h))] + r(u_l(i))\Big\} \\ &= \max_{1\le i\le q_l}\max_{1\le j\le i}\{r(\overline{u}_l(q_l - i + 1)) + \sum_{h=j}^{i} p(\overline{u}_l(q_l - h + 1)) - d(\overline{u}_l(q_l - j + 1))\} \\ &= \max_{1\le j\le q_l}\max_{j\le i\le q_l}\{r(\overline{u}_l(j)) + \sum_{h=i}^{j} p(\overline{u}_l(h)) - d(\overline{u}_l(i))\} \\ &= L(\overline{S}_{kl}, \text{PMSP}). \end{aligned} \tag{12}$$

Then,

$$L(S_k, \overline{\text{PMSP}}) = L(\overline{S}_k, \text{PMSP}). \tag{13}$$

The above equation of (13) means that both the latenesses of the schedule S_k for the reverse problem $\overline{\text{PMSP}}$ and the reverse schedule $\overline{S}_k$ for PMSP are identical.

Assume $\overline{S}_k$ is an optimal schedule for PMSP and S_k is not an optimal schedule for $\overline{\text{PMSP}}$. Then, there is a different optimal schedule, say S'_k for $\overline{\text{PMSP}}$, i.e., $L(S'_k, \overline{\text{PMSP}}) < L(S_k, \overline{\text{PMSP}})$. It can be seen that the PMSP and the reverse problem $\overline{\text{PMSP}}$ are one to one from the definition, so $L(\overline{S}'_k, \text{PMSP}) < L(S_k, \overline{\text{PMSP}}) = L(\overline{S}_k, \text{PMSP})$. This results in contradiction, i.e., S_k is an optimal schedule for $\overline{\text{PMSP}}$. □

As shown in the equation of (13), it should be noted that the PMSP and the $\overline{\text{PMSP}}$ have the identical maximum lateness if their schedules are reverse each other, but it does not mean that they yield the identical maximum lateness when an approximate algorithm is applied to them.

Obviously, the reverse of a schedule obtained from $\overline{\text{PMSP}}$ is a feasible one for PMSP. Thus, the following approximate algorithm is significant:

Algorithm $\overline{H}$: Apply algorithm H to the reverse problem $\overline{\text{PMSP}}$, and then reverse the schedule obtained.

Table 1: An example of the parallel-machine scheduling problem and its reverse problem.

(a) PMSP

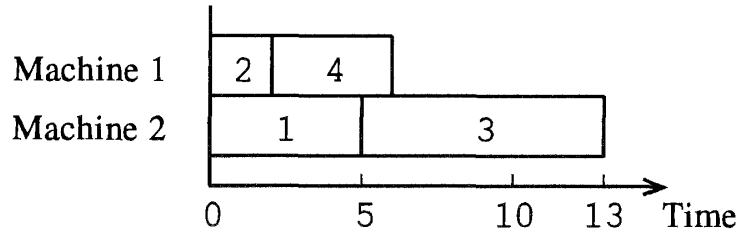
u	1	2	3	4
$p(u)$	5	2	8	4
$r(u)$	0	0	3	1
$d(u)$	10	9	10	8

$p(u)$: processing time;
$r(u)$: ready time;
$d(u)$: due date.

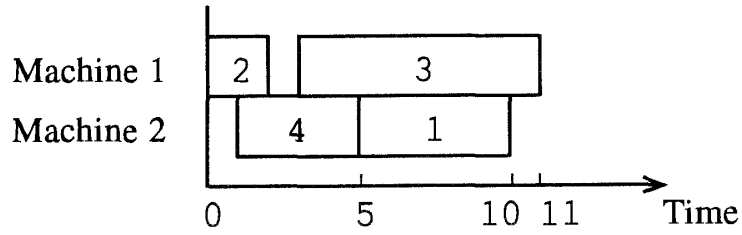
(b) Reverse problem $\overline{\text{PMSP}}$

u	1	2	3	4
$P(u)$	5	2	8	4
$r(u)$	-10	-9	-10	-8
$d(u)$	0	0	-3	-1

$p(u)$: processing time;
$r(u)$: ready time;
$d(u)$: due date.



(a) Schedule (2,4) and (1,3)



(b) Schedule (2,3) and (4,1)

Figure 4: Gantt chart of two approximate schedules.

As an illustration of this observation, consider a two-machine four-task PMSP and its reverse provided in Table 1. Two schedules obtained by algorithms H and $\overline{H}$ are respectively shown in Figure 4. Consequently, algorithm $\overline{H}$ produces a better schedule with $L_{max} = 1$ than a schedule with $L_{max} = 3$ produced by algorithm H.

On the basis of these facts, the approximate algorithm HA can be described as follows:

Approximate algorithm HA: The better one of the schedules obtained by applying the algorithms H and $\overline{H}$ to a given problem instance is to be selected.

3.3. A heuristic algorithm for the PMFSP

Algorithm A**:**

(Phase One: Initial scheduling)

Step 1 Convert the given PMFSP into a graph $G = (N, A, \emptyset, P, T)$.

Step 2 Compute the lower bound $LB(k)$ by the equation of (7) on optimal maximum lateness for PMSP$(k|\emptyset)$, $k = 1, 2, \ldots, m$.

Step 3 Solve PMSP$(k|K)$ (K is the set of stages which have already been scheduled) in descending order of lower bounds $LB(k)$ for $k \in \{1, 2, \cdots, m\}$, using approximate algorithm HA. An initial schedule is obtained.

(Phase Two: Improvement of initial schedule)

Step 4 Find critical paths and compute $CT(k)$ ($k \in M_{CS}$) by the equation of (8), where M_{CS} is the set of the critical stages.

Step 5 $k' \leftarrow \text{Arg} \max_{k \in M_{CS}} CT(k)$. Solve PMSP$(k'|K)$ ($K = \{1, 2, \cdots, k'-1, k'+1, \cdots, m\}$) associated with stage k', using approximate algorithm HA. $M_{CS} \leftarrow M_{CS} - \{k'\}$. If

the schedule is improved, return to Step 4.

Step 6 If $M_{CS} \neq \emptyset$, return to Step 5. Otherwise, stop. □

It should be noted that algorithm A concurrently makes the three decisions, i.e., routing, sequencing and timing.

We discuss the complexity of algorithm A. Step 1 requires $O(mn)$ time. Step 2 computes the longest path lengths of mn nodes for obtaining m lower bounds by the equation of (7), which takes $O(m^2n^2)$ time as the total. Step 3 requires $O(mn \log mn)$ time to solve m parallel-machine scheduling problems. Step 4 takes $O(m^2n^2)$ time per iteration. Step 5 requires $O(mn \log mn)$ time per iteration. Thus, let I be the number of iterations required in Phase Two, then the total computation time of Phase Two is $O(Im^2n^2)$ time which dominates the overall time complexity of algorithm A.

4. Numerical Experiments

In order to examine the performance of the proposed heuristic algorithm A, two kinds of numerical experiments were conducted. The first is a comparison with Wittrock's heuristic algorithm (see Wittrock [32]) through the same problem data as that in his experiments.

The second is a comparison with the global lower bound presented by Santos *et al.* [28] for the same problem types as that employed by them. This means that the ready times, the due dates and the transportation times are set to be zero, and the objective function reduces to the makespan.

The program was coded in FORTRAN and run on DEC 3000 (35MFLOPS) EWS.

4.1. Comparison with Wittrock's algorithm

Wittrock's algorithm is decomposed into three parts: machine allocation, sequencing and timing. For the machine allocation, the LPT (largest processing time) dispatching rule is employed stage by stage. For sequencing on each machine, a heuristic algorithm (called workload approximation) based on an approximate DP (dynamic programming) is developed. For timing, job ready times are determined so as to minimize the maximum queue on each stage without increasing the makespan. This does not mean the earliest start time of each task for a given load sequence.

Table 2 shows a comparison of the makespan between algorithm A and Wittrock's one. It also shows values of the Santos *et al.*'s global lower bound, denoted by LB_{max} (which will

Table 2: A comparison with Wittrock's algorithm ($m = 3, h(1) = 3, h(2) = 2, h(3) = 3$).

Problem Number	n	Makespan		LB_{max}
		algorithm A (Phase One only)	Wittrock	
1	51	764 (764)	784	731
2	38	773 (784)	789	744
3	38	764 (775)	785	743
4	36	789 (789)	796	741
5	40	963 (963)	964	963
6	30	669 (669)	686	651

be described in the next subsection).

It is obvious from this comparison that the proposed heuristic produces smaller makespan than Wittrock's algorithm does.

4.2. Comparison with Santos *et al.*'s global lower bound

Santos *et al.* [28] developed the following stage-based lower bound for the PMFSP:

$$LB(k) = \frac{1}{h(k)}\Big\{\sum_{y=1}^{h(k)} LSA(y,k) + \sum_{u\in J^k} p(u) + \sum_{y=1}^{h(k)} RSA(y,k)\Big\}, \tag{14}$$

where $LSA(y,k)$ is the y-th smallest value of job-based total processing times before stage k and $RSA(y,k)$ is the y-th smallest value of job-based total processing times after stage k. Obviously, the best possible lower bound is given by

$$LB_{max} = \max\{LB(0), max_k LB(k)\}, \tag{15}$$

where $LB(0)$ stands for the maximum value of the job-based total processing times over all stages.

They calculated the mean relative errors over 30 instances for each of 21 problem types. These mean relative errors were utilized to more pertinently estimate the performance of our algorithm as discussed below.

Let RE_{AO} = (APP − OPT)/OPT and RE_{OL} = (OPT − LOW)/OPT be respectively the relative errors of an approximate value (APP) and a lower bound value (LOW) to the optimum value (OPT), and RD_{AL} = (APP − LOW)/APP be the relative deviation between the approximate and the lower bound values. Then,

$$RE_{AO} = (\mathrm{APP}/\mathrm{LOW})(RD_{AL} - RE_{OL}). \tag{16}$$

Let $\overline{x}$ stands for the mean value of variable x, then we approximately have

$$\overline{RE}_{AO} = (\overline{\mathrm{APP}}/\overline{\mathrm{LOW}})(\overline{RD}_{AL} - \overline{RE}_{OL}). \tag{17}$$

The results obtained are shown in Table 3. The first three columns represent problem types tested. The processing times were generated by random integer numbers from 1 to 10 as in Santos *et al.* [28]. The fourth column represents the mean relative error of algorithm A which is estimated by the equation of (17). They ranges from 0.9[%] to 6.2[%] with average 3.1[%]. On the other hand, $\overline{RD}_{AL}$ in the sixth column ranges from 4.2[%] to 11.8[%] with average 7.6[%].

In order to examine the effect of the reversibility (discussed in Subsection 3.2), we examined the performance of algorithm A', which is the same as algorithm A except that algorithm H is used for each parallel-machine problem, instead of algorithm HA. The results obtained are given in the fifth column. It suggests that the makespan can be reduced as much as 2[%] by utilizeing the property of the reversibility.

The last column FR_{OPT} represents the fraction of the problem instances tested for which algorithm A produced the same value as the lower bound, meaning that it optimally solved the problem instances.

Table 3: Estimated mean relative error and fraction of optimal solutions.

Problem Type			$\overline{RE}_{AO}$ [%]			
Parts	Stages	Machines/Stage	A	A'	$\overline{RD}_{AL}$ [%]	FR_{OPT} [%]
4	2	2	2.9	5.9	7.5	27
	3	2	3.4	6.3	8.3	23
	4	2	2.4	2.2	6.6	33
	5	2	4.1	5.7	5.7	23
5	2	2	1.6	3.4	6.0	40
	3	2	3.9	6.4	11.4	0
	4	2	4.0	5.4	10.3	7
	5	2	5.8	7.2	10.0	3
6	2	2	4.2	6.3	7.0	43
		3	2.8	6.9	5.6	37
	3	2	3.8	4.8	10.9	13
		3	2.7	4.8	6.1	43
	4	2	6.2	9.6	11.8	3
		3	1.9	4.3	5.4	37
7	2	2	2.5	5.1	4.8	33
		3	0.9	4.1	7.1	33
	3	2	4.8	5.7	10.0	10
		3	2.7	5.5	7.4	20
8	2	2	2.1	5.3	4.2	50
		3	2.3	5.4	7.2	17
		4	1.2	2.3	4.9	53
Average			3.1	5.4	7.6	26

Table 4: Mean relative deviation $\overline{RD}_{AL}$ [%] for medium and large sized problem instances ($h(k) = 2$ for all stages $k = 1, 2, \ldots, m$).

	Stages				
Parts	2	3	4	5	6
10	2.6	8.6	14.4	15.4	21.0
20	0.5	4.8	8.3	11.6	14.3
30	0.3	3.1	6.5	9.9	12.9
40	0.5	2.5	4.6	8.1	11.9
50	0.3	2.6	5.4	7.2	10.5
60	0.1	2.5	4.3	8.2	10.4
70	0.1	1.1	3.9	5.8	8.7
80	0.1	0.9	2.7	7.1	8.3
90	0.1	0.9	3.1	5.6	7.9
100	0.1	1.2	3.1	5.2	7.2
Average	0.5	2.8	5.6	8.4	11.3

Comparison with the global lower bound was also executed for mideum and lager size problem instances. The mean relative deviations RD_{AL} obtained are shown in Table 4, in which each stage consists of 2 machines for every instance. The results suggest that algorithm A have much better performance for smaller number of stages and larger number of parts.

The running time of algorithm A is short. It took 0.16 [CPU sec.] on average for the problem instances in Table 2 and about 2 [CPU min.] for ones with 100 parts and 6 stages in Table 4. This fact means that the iteration number I of the Phase Two in algorithm A was quite limited for the 2156 problem instances tested.

5. Conclusion

This paper discussed a scheduling problem for a parallel-machine flowshop with m stages, each stage having one or more identical parallel machines, and developed a shifting bottleneck based heuristic algorithm, which decomposes the problem into m marallel-machine scheduling problems to be solved sequentially. For this purpose, a new approximate algorithm utilizing the property of the reversibility for the latter problem was developed. Extensive numerical experiments including comparisons with Wittrock's algorithm and Santos *et al.*'s global lower bound were executed. The results obtained strongly suggested that the proposed heuristic algorithm produces optimal or quite near optimal schedules within short computational time with high probability. This conclusion encourages extensions of the shifting bottleneck based approach to more general sysytems such as parallel-machine flowshops with non-identical parallel machines, precedence constraints between tasks, setup times and/or finite capacity of buffer storage between stages.

Acknowledgement

This research was partially supported a Scientific Grant in Aid from the Ministry of Education, Science, Sports and Culture of Japan.

References

[1] J. Adams, E. Balas and D. Zawaku: The shifting bottleneck procedure for job shop scheduling. *Management Science*, **34** (1988) 391–401.

[2] E. H. Aghezzaf and A. Artiba: Aggregate planning in hybrid flowshops. *International Journal of Production Research*, **36** (1998) 2463–2477.

[3] T. S. Arthanary and K. G. Ramaswarmy: An extension of two machine sequencing problem. *Opsearch*, **8** (1971) 10–22.

[4] S. A. Brah and J. L. Hunsucker: Branch-and-bound algorithm for the flowshop with multiple processors. *European Journal of Operational Research*, **51** (1991) 88–99.

[5] S. A. Brah and L. L. Loo: Heuristics for scheduling in a flow shop with multiple processors. *European Journal of Operational Research*, **113** (1999) 113–122.

[6] J. Carlier: One machine sequencing problem. *European Journal of Operational Research*, **11** (1982) 42–47.

[7] J. Carlier: Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *European Journal of Operational Research*, **29** (1987) 298–306.

[8] B. Chen: Analysis of classes of heuristics for scheduling a two-stage flowshop with parallel machines at one stage. *Operations Research*, **46** (1995) 234–244.

[9] D. Deal and J. Hunsucker: The two-stage flow-shop problem with m machines at each stage. *Journal of Information and Optimization Science*, **12** (1991) 407–417.

[10] A. Guinet and M. Solomon: Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. *International Journal of Production Research*, **34** (1996) 1643–1654.

[11] J. N. D. Gupta: Two-stage hybrid flow shop scheduling problem. *Operations Research*, **39** (1988) 359–364.

[12] J. N. D. Gupta, A. M. A. Hariri and C. N. Potts: Scheduling a two-stage hybrid flowshop with parallel-machine at the first stage. *Annals of Operations Research*, **69** (1997) 171–191.

[13] J. N. D. Gupta and E. Tunc: Scheduling for a two-stage hybrid flowshop with parallel-machine at the second stage. *International Journal of Production Research*, **29** (1991) 1489–1502.

[14] J. A. Hoogeveen, J. A. Lenstra and J. K. Veltman: Minimizing makespan in a multiprocessor flowshop is strongly NP-hard. *European Journal of Operational Research*, **89** (1993) 172–175.

[15] J. Hunsucker and J. Shah: Performance of priority rules in a due date flow shop. *OMEGA*, **20** (1992) 73–89.

[16] H. Iima and N. Sannomiya: Genetic algorithm for a scheduling problem in an electric wire production system with three subprocesses. *Proceedings of 1999 IFAC 14th Triennial World Congress, Beijing, P.R. China*, (1999) 279–284.

[17] H. Kise, T. Ibaraki and H. Mine: Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times. *Journal of the Operations Research Society of Japan*, **22** (1979) 205–224.

[18] S. Kochhar and R. J. T. Morris: Heuristic methods for flexible flow line scheduling. *Journal of Manufacturing Systems*, **6** (1987) 299–314.

[19] C. Lee and G. Vairaktarakis: Minimizing makespan in hybrid flow-shops. *Operations Research Letters*, **16** (1994) 149–151.

[20] J. K. Lenstra, A. H. G. Rinnooy Kan and P. Brucker: Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, **1** (1977) 343-362.

[21] V. J. Leon and B. Ramamoothy: An adaptable problem-space-based search method for flexible flow line scheduling. *IIE Transactions*, **29** (1997) 115–125.

[22] H. Luss and M. Rosenwein: A due date assignment algorithm for multiproduct manufacturing facilities. *European Journal of Operational Research*, **65** (1993) 187–198.

[23] R. J. Paul: A production scheduling problem in the glass-container industriy. *Operations Research*, **22** (1979) 290–312.

[24] C. Rajendran and D. Chaudhuri: A multi-stage parallel-processor flowshop problem with minimum flow time. *European Journal of Operational Research*, **57** (1992) 111–122.

[25] C. Rajendran and D. Chaudhuri: Scheduling in n-job, m-stage flow-shop with parallel-processor to minimize makespan. *International Journal of Production Research*, **27** (1992) 137–143.

[26] F. Riane, A. Artiba and S. E. Elmaghraby: A hybrid three-stage flowshop problem: efficient heuristics to minimize makespan. *European Journal of Operational Research*, **109** (1998) 321–329.

[27] A. Rumudhin and P. Marier: The generalized shifting bottleneck procedure. *European Journal of Operational Research*, **93** (1996) 34–48.

[28] D. L. Santos, J. L. Hunssucker and D. E. Deal: Global lower bounds for flowshops with multiple processors. *European Journal of Operational Research*, **80** (1995) 112–120.

[29] L. Seetharama: Scheduling in a two-stage manufacturing process. *International Journal of Production Research*, **22** (1984) 555-564.

[30] C. Sriskandarajah and S. P. Sethi: Scheduling algorithms for flexible flow shops: worst and average case performance. *European Journal of Operational Research*, **43** (1989) 143–160.

[31] R. Wittrock: Scheduling algorithms for flexible flow line. *IBM Journal of Research and Development*, **29** (1985) 401–412.

[32] R. Wittrock: An adaptable scheduling algorithm for flexible flow lines. *Operations Research*, **36** (1988) 445–453.

Hiroshi Kise
Department of Mechanical and System Engineering
Faculty of Engineering and Design
Kyoto Institute of Technology
Matsugasaki, Sakyo
Kyoto 606-8585, Japan
E-mail: kise@ipc.kit.ac.jp