Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 863–875
computers & operations research
www.elsevier.com/locate/cor

A branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property

Sang-Oh Shim, Yeong-Dae Kim*

Department of Industrial Engineering, Korea Advanced Institute of Science and Technology, Yusong-gu, Daejon 305-701, Korea

Available online 27 June 2006

Abstract

We focus on the problem of scheduling n independent jobs on m identical parallel machines with the objective of minimizing total tardiness of the jobs considering a job splitting property. In this problem, it is assumed that a job can be split into sub-jobs and these sub-jobs can be processed independently on parallel machines. We develop several dominance properties and lower bounds for the problem, and suggest a branch and bound algorithm using them. Computational experiments are performed on randomly generated test problems and results show that the suggested algorithm solves problems of moderate sizes in a reasonable amount of computation time.

Scope and purpose

In this paper, we consider an operations scheduling problem in a real manufacturing system that produces printed circuit boards (PCBs). In the PCB manufacturing system, production orders are specified by the product types, production quantities and due dates. In a bottleneck workstation of the manufacturing system, in which there are multiple parallel machines, a job required for a production order can be split into basic processing units, each of which can be processed on a machine. We present an algorithm that can give an optimal schedule for a scheduling problem in the workstation for the objective of minimizing total tardiness of production orders. Since meeting due dates of customers' orders become more and more important for the survival of a manufacturing company in highly competitive market environments of these days, the algorithm may provide the company with a competitive edge in the industry.

Keywords: Scheduling; Parallel machines; Job splitting; Total tardiness; Branch and bound

1. Introduction

This research focuses on a problem of scheduling n jobs on m identical parallel machines with the objective of minimizing total tardiness of the jobs, while considering a job splitting property of the jobs. In certain manufacturing shops with parallel machines, each job can be regarded as a production order to process a product type in a specified quantity by a given due date. In scheduling problems for those shops, jobs can be split into a number of sub-jobs that can be processed independently on two or more parallel machines at the same time. Such a problem is called the parallel machine scheduling problem with splitting jobs [1]. In this paper, a *job* is defined as a production order unit composed of a number of *unit-jobs*. Here, unit-jobs from a job are identical in that their processing times are the same

* Corresponding author. Tel.: +82 42 869 3120; fax +82 42 869 3110.
E-mail address: ydkim@kaist.ac.kr (Y.-D. Kim).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.04.006

and their due dates are the same, and a *sub-job* is defined as a set or a batch of unit-jobs from a job that is processed on a machine consecutively. In other words, a job can be split into one or more sub-jobs and these sub-jobs are processed on the parallel machines independently.

We develop an optimal solution algorithm for the problem described above. Here, tardiness of a job is defined as $T_i = \max\{0, C_i - d_i\}$, where C_i and d_i are the completion time and due date of job i, respectively, and the completion time of a job is the time when all sub-jobs from the job are completed. In the problem considered here, we assume:

(1) all jobs are available at time zero;
(2) each machine can process only one sub-job at a time;
(3) each sub-job can be processed on only one machine; and
(4) a setup operation is required before a sub-job is processed on a machine, if the job type of a sub-job to be processed is different from the job type of the sub-job just processed, and setup times are independent of sequences of the sub-jobs.

As noted in Kim et al. [2], this problem can be found in printed circuit board (PCB) manufacturing systems. In the PCB manufacturing systems, jobs are moved through and processed at workstations in batches, i.e., groups of panels called lots, in most part of the manufacturing process. In the drilling workstation, there are multiple parallel machines to drill holes on panels. In this workstation, each lot can be split into groups of smaller numbers of panels that are drilled together. These groups of panels can considered as unit-jobs in this study. After the drilling process, unit-jobs are combined into a lot. Unit-jobs split from a lot can be processed independently on the machines. Note that there may be other cases of problems with a job-splitting property in other types of manufacturing systems, in which a job consists of a large number of (say 100) work pieces that should be processed individually. In such cases, a job consists of a larger number of unit-jobs, and possibly a larger number of sub-jobs.

Although problems with due date-related performance measures were studied in many research articles, not much progress has been made for the objective of minimizing total tardiness in parallel machine scheduling problems except for special cases of identical parallel machines. Root [3] develops a constructive algorithm to minimize total tardiness when all jobs have the same due dates, and Elmaghraby and Park [4] develop a branch and bound (B&B) algorithm to minimize the penalty of tardiness when the due date is the same as the processing time for each job. Arkin and Roundy [5] present a dynamic programming algorithm and a heuristic algorithm for the problem of minimizing the sum of weighted tardiness on identical parallel machines where job weights are proportional to processing times. For problems of minimizing total tardiness on identical parallel machines, Azigzoglu and Kirca [6] and Yalaoui and Chu [7] give B&B algorithms for optimal solutions. On the other hand, Liaw et al. [8] and Shim and Kim [9] devise B&B algorithms to obtain optimal solutions in unrelated parallel machine scheduling problems.

Because of the difficulty of obtaining optimal solutions for problems of large sizes, heuristics have been developed and presented in many research articles. In most heuristics, jobs are prioritized according to methods developed for the single machine tardiness problem and then scheduled on the machines according to the priorities [10–13]. On the other hand, Kolumas [14] presents a decomposition heuristic by extending a decomposition principle for the single machine tardiness problem to a parallel-machine setting, and Kim [15] suggests a backward approach for the list scheduling algorithms for multi-machine tardiness problems. Also, Lee and Pinedo [16] propose a simulated annealing algorithm for a parallel machine tardiness problem with sequence-dependent setup times, and Park [17] suggests a genetic algorithm for a parallel-machine problem with the objective of minimizing total weighted tardiness.

There are very few research results on the parallel machine scheduling problems with job splitting properties. Blocher and Chhajed [18] show that the problem with the objective of minimizing total completion time is NP-hard and develop several heuristics and lower bounds, and Yang and Posner [19] develop heuristics and give worst-case bounds for these heuristics. Serafini [20] and Xing and Zhang [1] consider identical parallel machine problems with a job-splitting property for the objective of minimizing the maximum tardiness and minimizing makespan, respectively. In addition, Kim et al. [21] and Logendran and Surbur [22] present local search heuristics for the objective of minimizing total weighted tardiness on unrelated parallel machines. For identical parallel machine scheduling problems (IPMSPs) with the objective of minimizing total tardiness, Kim et al. [2] suggest a two-phase heuristic algorithm.

In this paper, we suggest a B&B algorithm for the IPMSP with the objective of minimizing total tardiness considering the job splitting property described earlier. We develop dominance properties and lower bounds and use them in the B&B algorithm. The remainder of this paper is organized as follows. In the next section, we introduce terminology and notation used in this paper, and we develop dominance properties that can be used in B&B algorithms in Section 3.

In Sections 4 and 5, we present a node representation scheme used to represent sub-problems (or partial solutions) in B&B algorithms for the problem and lower bounds for partial solutions, respectively. Section 6 presents a brief description of the overall procedure of the B&B algorithm suggested in this study. Performance of the algorithm is tested on randomly generated test problems and results are reported in Section 7. Finally, Section 8 concludes the paper with a short summary and discussions on possible extensions.

2. Terminology and notation

To describe the algorithm clearly, we first define terms that will be used in this paper.

job	a production order unit for a product type with order quantity and due date;
unit-job	unit of a minimum process batch for a job; unit-jobs from a job are identical, in that their processing times are the same and their due dates are the same;
sub-job	a set or a batch of unit-jobs from a job that are processed on a machine consecutively.

We also use the following terms for jobs classified by the status of sub-jobs of the jobs (in a partial schedule related to a sub-problem in the B&B algorithm).

completely scheduled jobs	jobs with all of their sub-jobs scheduled;
partially scheduled jobs	jobs with some of their sub-jobs scheduled but the rest are not;
un-scheduled jobs	jobs with none of their sub-jobs scheduled.

Notation used in this paper is given below

n number of jobs
m number of parallel machines
i index for jobs
j index for sub-jobs
k index for machines
u_i number of unit-jobs of job i
p_i processing time of each unit-job of job i
s_i (sequence-independent) setup time for job i
d_i due date of job i
$C_j(\sigma)$ completion time of sub-job j in (partial) schedule σ
$C_k^{\mathrm{M}}(\sigma)$ completion time on machine k in (partial) schedule σ, i.e., the time when all sub-jobs assigned to machine k are completed
S set of *completely* scheduled jobs (in schedule σ)
S' complement of S, i.e., set of partially scheduled or un-scheduled jobs.

In this study, a partial schedule, σ, is said to be *dominated* by another partial schedule, σ', if there is a complete schedule obtained from σ' that is better than complete schedules obtained from σ.

3. Dominance properties

In this section, we present several dominance properties that can be used in a B&B algorithm for the problem considered in this paper. The following proposition gives a dominance property related to sub-jobs assigned to the same machine.

Proposition 1. *Consider a schedule σ in which sub-job j_1 and sub-job j_2 of the same job are assigned to the same machine and sub-job j_1 is completed earlier than sub-job j_2. Then, σ is dominated by another schedule, σ', in which the sequence of sub-jobs is identical to that in σ except that sub-job j_1 is moved to the position immediately before sub-job j_2 and then these two sub-jobs are merged into one sub-job.*

Proof. Since the completion time of a job is the completion time of the last sub-job split from the job and setup is not required between processings of sub-jobs of the same job, the completion time of the job associated with the sub-jobs in σ' is not greater than that in σ. Also, since the completion times of all sub-jobs placed between sub-job j_1 and sub-job

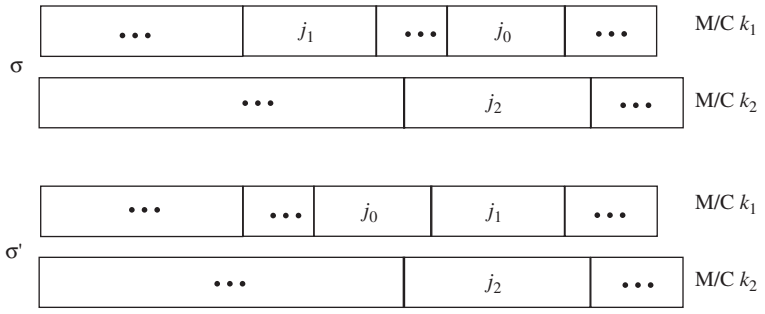

Fig. 1. Schedules σ and σ' in the proof for Proposition 2.

j_2 in schedule σ' are not greater than those in σ, tardiness of jobs related with these sub-jobs in σ' is not greater than that in σ. In addition, since the completion time of all sub-jobs included in σ (and σ') on each machine in σ' is not greater than that in σ, that is, $C_k^{\mathrm{M}}(\sigma') \leqslant C_k^{\mathrm{M}}(\sigma)$ for all k, the time each machine becomes available for sub-jobs not included in σ (and σ')for complete schedules associated with σ' is not greater than that for complete schedules associated with σ. Therefore, σ is dominated by σ'. □

Proposition 1 directly results in the following corollary.

Corollary 1. *There exists an optimal schedule in which the number of sub-jobs of each job is not greater than the number of machines.*

Proof. Using Proposition 1, we can easily show that a schedule in which two or more sub-jobs from the same job are assigned to the same machine is dominated by another schedule in which these sub-jobs are merged into one sub-job, and as a result, the corollary holds. □

In the following proposition, we give a dominance property related to sub-jobs assigned to different machines.

Proposition 2. *Consider a schedule σ in which two sub-jobs, say sub-jobs j_1 and j_2, of the same job are assigned to different machines and the completion time of sub-job j_1 is less than that of sub-job j_2. If there exists a sub-job of another job, say sub-job j_0, that is placed after sub-job j_1 on the machine to which sub-job j_1 is assigned and whose completion time is not greater than the completion time of sub-job j_2 in σ, then σ is dominated by a sequence, σ', in which sequences of sub-jobs on the machines are identical to those in σ except that sub-job j_1 is moved to the position immediately after sub-job j_0.*

Proof. Since the completion time of a job is determined by the completion time of a sub-job (of the job) that is completed last, the completion time of the job associated with sub-job j_1 in σ' is not greater than that in σ (see Fig. 1). In addition, since the completion times of sub-job j_0 as well as the completion times in σ' of sub-jobs that were placed (in σ) between sub-jobs j_1 and j_2 are not greater than those in σ, tardiness of jobs related with these sub-jobs in σ' is not greater than that in σ. Also, since $C_k^{\mathrm{M}}(\sigma') \leqslant C_k^{\mathrm{M}}(\sigma)$ for all k, the time each machine becomes available for sub-jobs not included inσ (and σ') for complete schedules associated with σ' is not greater than that for complete schedules associated with σ. Therefore, schedule σ is dominated by schedule σ'. □

The following proposition gives a dominance property related to the assignment of sub-jobs to the machines. Let schedule $\sigma j^{\wedge} k$ denote the (partial) schedule obtained by appending sub-job j at the end of σ on machine k.

Proposition 3. *Consider a schedule, say σ, in which one of the sub-jobs, say sub-job j_1, of a partially scheduled job, say job i, is already scheduled on machine k_1 and none of the sub-jobs of that job is assigned to machine k_2 $(k_1 \neq k_2)$. If $C_{j_2}(\sigma j_2^{\wedge} k_1) \leqslant C_{j_2}(\sigma j_2^{\wedge} k_2)$ and $C_{k_2}^{\mathrm{M}}(\sigma) \leqslant C_{k_1}^{\mathrm{M}}(\sigma)$, then $\sigma j_2^{\wedge} k_2$ is dominated by $\sigma j_2^{\wedge} k_1$.*

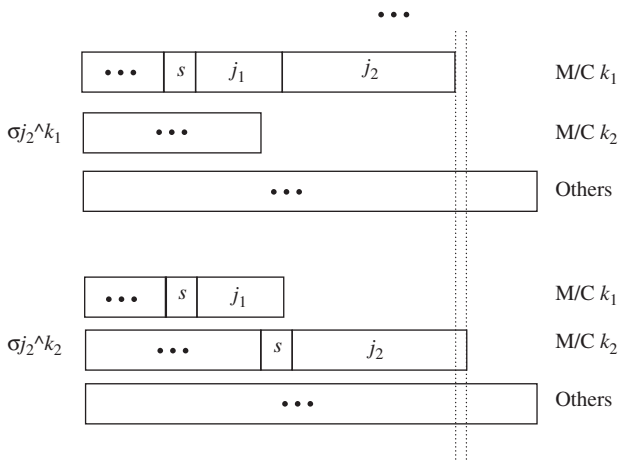

Fig. 2. Partial schedules $\sigma j_2\hat{}k_1$ and $\sigma j_2\hat{}k_2$ in the proof for Proposition 3.

Proof. Consider two partial schedules, $\sigma j_2\hat{}k_1$ and $\sigma j_2\hat{}k_2$ (Fig. 2). In $\sigma j_2\hat{}k_1$, setup is not needed before processing sub-job j_2. (Note that, if the job associated with the last sub-job of σ on machine k_1 is not the same as the job associated with sub-job j_2, we can move sub-jobs of that job just before sub-job j_2 without increasing tardiness using the results of Proposition 1.) Since the completion time of a job is determined as the completion time of the sub-job completed last, the completion time of job i in $\sigma j_2\hat{}k_1$ is not greater than that in $\sigma j_2\hat{}k_2$. Also, the completion time in $\sigma j_2\hat{}k_1$ of each of all sub-jobs (included in the two partial schedules) is not greater than that in $\sigma j_2\hat{}k_2$. In addition, we have $C^{\mathrm{M}}_{k_1}(\sigma j_2\hat{}k_1) \leqslant C^{\mathrm{M}}_{k_2}(\sigma j_2\hat{}k_2)$ and $C^{\mathrm{M}}_{k_2}(\sigma j_2\hat{}k_1) \leqslant C^{\mathrm{M}}_{k_1}(\sigma j_2\hat{}k_2)$ from $C^{\mathrm{M}}_{k_2}(\sigma) \leqslant C^{\mathrm{M}}_{k_1}(\sigma)$. This means sub-jobs that are not included in $\sigma j_2\hat{}k_1$ (or $\sigma j_2\hat{}k_2$) can be started earlier in a complete schedule that starts with $\sigma j_2\hat{}k_1$ than in a complete schedule that starts with $\sigma j_2\hat{}k_2$, since all machines are identical. Therefore, tardiness of jobs of a complete schedule that starts with $\sigma j_2\hat{}k_1$ is not greater than that of a complete schedule that starts with $\sigma j_2\hat{}k_2$. That is, $\sigma j_2\hat{}k_2$ is dominated by $\sigma j_2\hat{}k_1$. □

4. Node representation scheme

The problem considered in this paper can be considered as an IPMSP with Σu_i jobs and m machines, although it is different from a typical IPMSP. In a commonly used node representation scheme, or branching scheme, for the IPMSP, a complete schedule is represented with a permutation of n jobs. A complete schedule can be obtained easily from a permutation by scheduling each job (in the order in the permutation) on a machine on which the job can be started earliest [6,7,23], as in list scheduling algorithms. However, this node representation scheme can only be used for cases in which only active schedules need to be considered. (An active schedule is a schedule in which no job can be started earlier without delaying any other job, or equivalently, in which no global left-shift can be made.) In the problem considered in this study, active schedules are not dominant and we need to consider non-active schedules as well, as is done in the node representation scheme suggested by Brah and Hunsuncker [24] for an IPMSP. Therefore, based on the node representation scheme suggested by Brah and Hunsuncker, we devise a new representation scheme considering special characteristics of the problem.

In the node representation scheme developed here, the B&B tree is constructed as follows. In the B&B tree, there are two types of nodes, nodes with a square shape and nodes with a circular shape as shown in Fig. 3. Each node in the B&B tree represents a unit-job and the figure given in the node represents the index of the job from which the unit-job is split. A unit-job associated with a square-shape node (square node) is the first unit-job to be scheduled at the first position on a (new) machine, while a unit-job associated with a circular-shape node (circular node) is a unit-job to be assigned to the current machine, i.e., the machine to which the unit-job associated with the parent node of that node was assigned. Unit-jobs corresponding to nodes closer to the root node are scheduled earlier than those corresponding to nodes farther from the root node, if they are to be assigned to the same machine. Note that if a child node and its

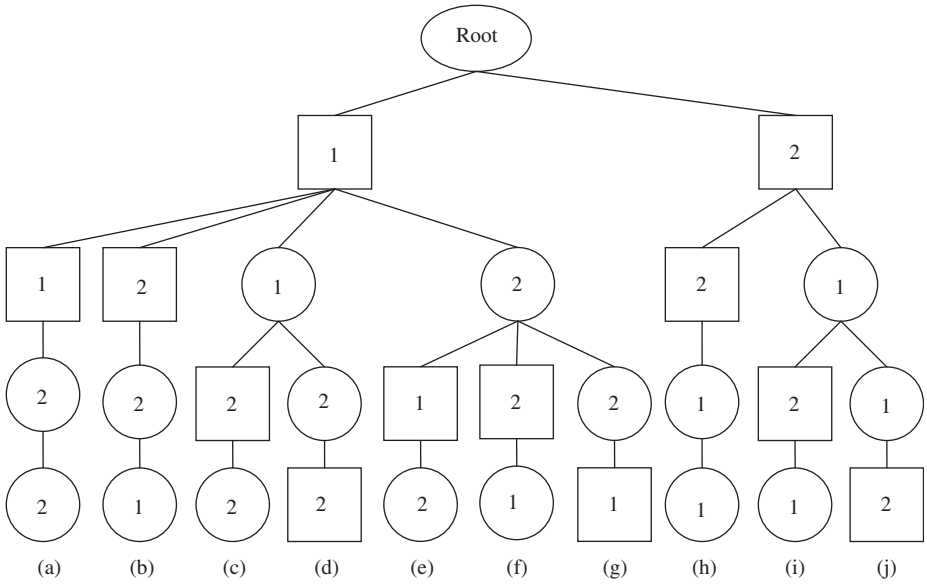

Fig. 3. A branch and bound tree obtained by applying the new scheme.

immediate parent node are associated with unit-jobs split from the same job, these two nodes represent a sub-job in the schedule.

To reduce the number of redundant schedules generated in the B&B procedure, we use the following rules when generating nodes in the B&B algorithm.

Rule 1: In level 1 of the B&B tree, no more than one square node can be generated for each job. The indices of the square nodes cannot be greater than $\min\{n, \Sigma u_i - m + 1\}$. Note that if $\Sigma u_i - m + 1 \leqslant 1$, only one square node can be generated.

Rule 2: From level 2 through level Σu_i, square nodes and circular nodes can be generated.

Rule 3: The job index of a unit-job associated with a certain square node should not be less than that of a unit-job associated with a square node that is on the path from the current node to the root node.

Rule 4: The number of square nodes in each path from the root node to a leaf node should not exceed the number of machines.

Rule 5: The unit-job associated with a circular node should be either a unit-job that is split from the job associated with its parent node or a unit-job that is split from another job, whose unit-jobs have not been scheduled on the current machine yet. Note that if the path from the node (to be generated) to the root node includes a node for a unit-job associated with this job (related to the node to be generated), the partial schedule associated with the node (to be generated) is dominated by another schedule, according to Proposition 1. By Proposition 1, the unit-job corresponding to the circular node to be generated can be merged with a unit-job or a sub-job that is scheduled earlier on the same machine, and the partial sequence obtained by the merge can be represented by another node in the B&B tree.

When a partial schedule is constructed for a node, setup time is inserted in front of unit-jobs corresponding to square nodes. Fig. 4 shows a Gantt chart for each leaf node of a B&B tree given in Fig. 3, for example, in which there are two machines and two jobs with two unit-jobs per each job, the processing time of each unit-job is 1, and setup times of job 1 and 2 are 1 and 2, respectively. In the schedule associated with leaf node *a*, one unit-job of job 1 is assigned to machine 1 and the other unit-job of job 1 and both unit-jobs of job 2 are assigned to machine 2, which means job 1 is split into two sub-jobs and there is only one sub-job for job 2. On the other hand, in the schedule associated with leaf node *c*, neither of the jobs is split.

In this example, it can be seen that the schedules associated with leaf nodes *a* and *g* are redundant, since we can get exactly identical schedules (note that the machines are identical). Also, the schedules associated with nodes *h* and *j* are redundant. It can be observed that redundant schedules are generated by the node representation (or branching)

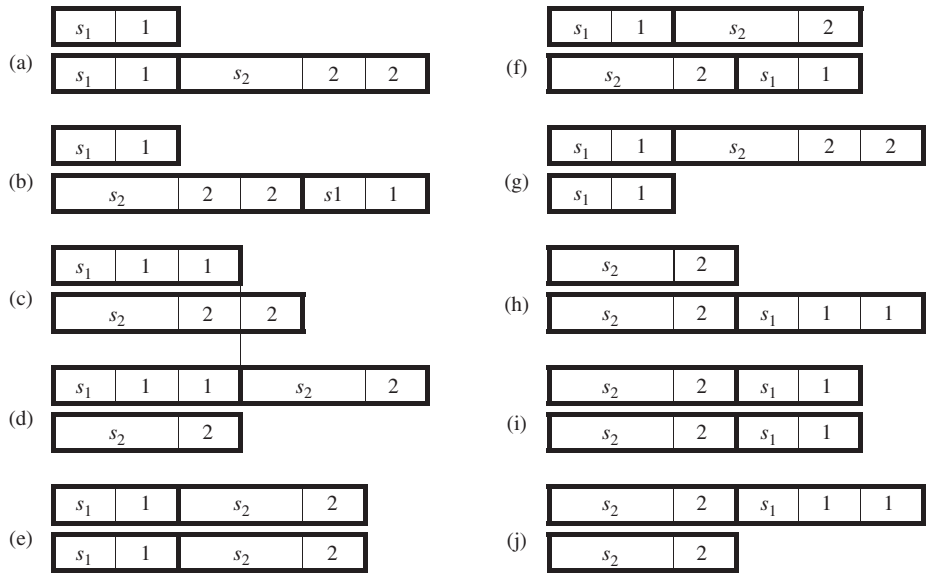

Fig. 4. Gantt Charts for schedules associated with leaf nodes of the B&B tree.

scheme suggested in this study, when more than two unit-jobs that are split from the same job are assigned to the first position on each machine. In the B&B algorithm suggested in this study, such a condition is checked when new nodes are generated, and if redundant schedules are found, one of these schedules is pruned.

5. Lower bounds

In the B&B algorithm, a lower bound is obtained for each node that cannot be eliminated by the dominance properties given in Section 3. In this study, we develop three lower bounds that correspond to partially scheduled jobs as well as un-scheduled jobs in the partial schedule associated with a node. Let σ denote such a partial schedule and let $u_i^{\mathrm{u}}(\sigma)$ denote the number of unit-jobs of (partially scheduled) job i that are not scheduled yet in σ.

The first lower bound, LB1, is obtained by computing the earliest possible completion times of jobs. That is, if the earliest possible completion time of a job can be obtained for partially scheduled jobs as well as un-scheduled jobs associated with σ, we can compute a lower bound on the tardiness of these jobs with those earliest possible completion times. Let e_i denote the earliest possible completion time of job i and let $[l]$ be the index of the machine (among $k = 1, 2, \ldots, m$) with the lth smallest value of $C_k^{\mathrm{M}}(\sigma)$. Then, e_i can be computed as,

$$e_i = \min_{1 \leqslant h \leqslant m} \left[\left\{ \sum_{l=1}^{h} C_{[l]}^{\mathrm{M}}(\sigma) + hs_i + u_i p_i \right\} \Big/ h \right], \quad \text{if job } i \text{ is an un-scheduled job,}$$

$$e_i = \min_{1 \leqslant h \leqslant m} \left[\left\{ \sum_{l=1}^{h} C_{[l]}^{\mathrm{M}}(\sigma) + p_i u_i^{\mathrm{u}}(\sigma) \right\} \Big/ h \right], \quad \text{if job } i \text{ is a partially scheduled job.}$$

The terms in the brackets, [ ], in the above equations denote the completion time of job i that can be obtained if the job was split into h sub-jobs, which are scheduled on h machines just after σ, and the workloads of the machines were exactly the same.

The above e_i for partially scheduled jobs can be modified since setup is needed only on machines to which no sub-job of job i is assigned (and scheduled) in partial schedule σ. Let $\hat{C}_k = C_k^{\mathrm{M}}(\sigma) + s_i$, if there is no sub-job of job i that has been assigned to machine k, and $\hat{C}_k = C_k^{\mathrm{M}}(\sigma)$, if there is at least one sub-job of job i assigned to machine k. If we let

$\{l\}$ be the index of the machine with the lth smallest value of $\hat{C}_k$, e_i can be computed as,

$$e_i = \min_{1 \leqslant h \leqslant m} \left[\left\{ \sum_{l=1}^{h} \hat{C}_{\{l\}} + u_i p_i \right\} \Big/ h \right], \quad \text{if job } i \text{ is an un-scheduled job,}$$

$$e_i = \min_{1 \leqslant h \leqslant m} \left[\left\{ \sum_{l=1}^{h} \hat{C}_{\{l\}} + p_i u_i^{\mathrm{u}}(\sigma) \right\} \Big/ h \right], \quad \text{if job } i \text{ is a partially scheduled job.}$$

Using the above, the first lower bound, LB1, can be written as

$$\mathrm{LB1} = \sum_{i \in S'} \max\{e_i - d_i, 0\}.$$

The second and third lower bounds, LB2 and LB3, are based on the following property, which is developed by Chu [25] and independently by Kondakci et al. [26] and Kim [27]. Here, $d_{(i)}$ denotes the due date of a job with the ith earliest due date among partially scheduled jobs and un-scheduled jobs associated with σ, the partial schedule corresponding to the current node.

Lemma 1 (Chu [25], Kondakci et al. [26], Kim [27]). *For nonnegative real number c_i and d_i corresponding to job i, if $c_1 \leqslant c_2 \leqslant \cdots \leqslant c_n$, then the following inequality holds*:

$$\sum_{i=1}^{n} \max(c_i - d_{(i)}, 0) \leqslant \sum_{i=1}^{n} \max(c_i - d_i, 0).$$

From Lemma 1, we can compute a lower bound on the total tardiness of jobs, if we can compute (lower bounds on) the completion times of the jobs. In the second and third lower bounds, LB2 and LB3, we use methods for computing lower bounds on the completion time of the job that is completed ith.

LB2 is obtained by relaxing the problem considered in this study into a single-machine tardiness problem in which processing time of job i is set to $t_i = p_i u_i^{\mathrm{u}}(\sigma)/m$ if job i is a partially scheduled job, and $t_i = (s_i + u_i p_i)/m$ if job i is an un-scheduled job. In this case, a lower bound on the completion time of the job that is completed ith ($i = 1, 2, \ldots, |S'|$) can be computed as

$$\min_{k \in M} C_k^{\mathrm{M}}(\sigma) + \sum_{l=1}^{i} t_{\langle l \rangle},$$

where $\langle l \rangle$ denotes the index of the job (among all jobs) with the lth smallest value of t_i. Therefore, a lower bound on the total tardiness for partially scheduled jobs and un-scheduled jobs associated with the partial schedule corresponding to the current node can be written as

$$\mathrm{LB2} = \sum_{i=1}^{|S'|} \max \left\{ \min C_k^{\mathrm{M}}(\sigma) + \sum_{l=1}^{i} t_{\langle l \rangle} - d_{(i)}, 0 \right\}.$$

The third lower bound, LB3, is obtained as follows. Let R_i denote the remaining processing time, which is set to $p_i u_i^{\mathrm{u}}(\sigma)$ if job i is partially scheduled, or $s_i + u_i p_i$ if job i is un-scheduled, and let $((l))$ denote the index of the job with the lth smallest value of R_i. Then, we can obtain a lower bound on the completion time of the job that is completed ith ($i = 1, 2, \ldots, |S'|$) as

$$\min_{1 \leqslant h \leqslant \min(i,m)} \left[\left\{ \sum_{l=1}^{h} C_{[l]}^{\mathrm{M}}(\sigma) + \sum_{l=1}^{i} R_{((l))} \right\} \Big/ h \right].$$

Consequently, the third lower bound, LB3, can be written as

$$\mathrm{LB3} = \sum_{i=1}^{|S'|} \max \left\{ \min_{1 \leqslant h \leqslant \min(i,m)} \left[\left\{ \sum_{l=1}^{h} C_{[l]}^{\mathrm{M}}(\sigma) + \sum_{l=1}^{i} R_{((l))} \right\} \Big/ h \right] - d_{(i)}, 0 \right\}.$$

In the B&B algorithm, a lower bound for each node is obtained by adding the largest of these three lower bounds to the total tardiness of completely scheduled jobs in the partial schedule associated with the node.

6. Branch and bound algorithm

In this section, we describe the B&B algorithm developed in this research. As described earlier in Section 4, each unit-job is considered as an independent job in the procedure of the B&B algorithm. For branching, i.e., for selecting a node to generate branches from, the depth-first rule is used. That is, a node with the most unit-jobs included in the associated partial schedule is selected for branching. In case of ties, a node with the lowest lower bound is selected. When child nodes are generated from a selected parent node, it is checked whether or not schedules associated with the child nodes are dominated by using dominance properties, that can be obtained from Propositions 1–3. For each child node that is not pruned, the lower bound given in Section 5 is computed. Nodes with lower bounds that are greater than or equal to the upper bound, i.e., the solution value of the current incumbent solution, are deleted from further consideration (fathomed).

We obtain an initial upper bound (incumbent solution) using the two-phase heuristic algorithm developed by Kim et al. [2] for the same problem as the one considered here. In the first phase of this heuristic algorithm, an initial schedule is obtained by the algorithm of Koulamas [14], which is a well-known heuristic for identical parallel machine tardiness problems, under the assumption that splitting of jobs is not allowed. In the second phase, schedules are improved iteratively by splitting jobs if necessary. In the B&B algorithm, an initial upper bound is obtained at the root node of the B&B tree using the heuristic algorithm. The upper bound is updated whenever a complete schedule that is better than the current incumbent solution is found.

7. Computational experiments

To evaluate performance of the suggested B&B algorithm, computational experiments were performed on randomly generated test problems. For the experiments, 1440 problems were generated, five problems for each of all combinations of two levels for the number of machines (2 and 3), four levels for the number of jobs (4, 6, 8 and 10), three levels for the number of unit-jobs of a job (4–6), three levels for the length of setup time short, medium and long and four different parameter sets used to generate due dates (see below). Other data were generated in such a way that the resulting problem instances reflected real situations relatively well, as follows.

(1) The processing time of a unit-job was generated from the discrete uniform distribution with a range [5,60].
(2) Setup time for a job was generated from the discrete uniform distribution with range [5,60], [60,120] and [120,180] for problems of short, medium and long setup time, respectively.
(3) As in Kim et al. [2], due date of job i, d_i, was generated from the discrete uniform distribution with range $[\alpha \cdot \sum_i (s_i + u_i p_i)/m, \beta \cdot \sum_i (s_i + u_i p_i)/m]$, where s_i, p_i, u_i and m denote the setup time of job i, processing time of a unit-job of job i, the number of unit-jobs associated with job i and the number of machines, respectively, and α and β are parameters used to control tightness (and range) of due dates. As stated above, four different pairs of values for α, β) were used, which were (0, 0.4), (0, 0.8), (0, 1.2) and (0.5, 1.0). For example, in problems with $(\alpha, \beta) = (0, 0.4)$, jobs have very tight due dates compared with problems with the other values for the pair.

First, to see the effectiveness of the dominance properties and the lower bounds given in this paper, we tested five B&B algorithms: BB1, in which only LB1 was used; BB2, in which only LB2 was used; BB3, in which only LB3 was used; BB4, in which all LBs were used without dominance properties and BB5, in which all dominance properties and all LB's were used. All the algorithms were coded in C, and computational experiments were performed on a personal computer with a Pentium 4 processor operating at 3.0 GHz clock speed. For each problem solved, we tallied the number of nodes generated in the B&B tree and the CPU time required to solve the problem.

Results of the test are shown in Table 1, which gives the average computation time required to solve a problem and the average percentage ratio of the number of subproblems generated in the algorithm to that of all subproblems to be generated if no node were fathomed. As can be seen from the results of BB1, BB2 and BB3, LB3 was slightly more effective than LB1 and LB2, although the three bounds resulted in similar performance. Also, BB4 outperformed BB1, BB2 and BB3, which implies that using all lower bounds is more effective than using only one lower bound, although

Table 1
Performance of the algorithms

Number of		BB1		BB2		BB3		BB4		BB5	
machines	jobs	PRSC[a]	CPUT[b]	PRSC	CPUT	PRSC	CPUT	PRSC	CPUT	PRSC	CPUT
2	4	8.7×10^{-2}	0.4	8.1×10^{-2}	0.4	4.5×10^{-2}	0.4	2.4×10^{-2}	0.3	3.9×10^{-3}	0.2
	6	1.1×10^{-2}	16.7	1.4×10^{-2}	19.4	1.1×10^{-2}	15.8	3.4×10^{-3}	11.3	3.0×10^{-5}	4.9
	8	8.7×10^{-5}	286.2	3.7×10^{-4}	326.2	7.6×10^{-4}	232.4	5.2×10^{-5}	200.6	3.9×10^{-7}	57.9
	10	3.9×10^{-5}	3126.7	4.1×10^{-5}	3840.1	1.7×10^{-5}	2981.4	2.6×10^{-6}	2531.7	6.2×10^{-10}	690.6
3	4	2.1×10^{-1}	0.4	2.2×10^{-1}	0.5	1.7×10^{-1}	0.4	4.5×10^{-2}	0.3	6.1×10^{-5}	0.2
	6	8.7×10^{-2}	29.2	0.4×10^{-2}	34.1	5.3×10^{-3}	21.8	0.6×10^{-5}	19.8	1.6×10^{-7}	5.1
	8	5.2×10^{-4}	381.4	6.3×10^{-4}	424.6	1.2×10^{-4}	351.5	3.5×10^{-6}	298.1	8.6×10^{-9}	59.0
	10	9.1×10^{-6}	4924.8	1.8×10^{-6}	5554.8	6.2×10^{-6}	4530.2	1.6×10^{-8}	4203.2	1.2×10^{-14}	1323.1

[a]Average percentage ratio of the number of subproblems considered to that of subproblems that should have been considered without the dominance conditions or lower bounds.
[b]Average CPU time (in seconds) required to solve a problem.

Table 2
Results of the comparison of the algorithms

Problem size[a]	Average ratio of CPU times				Average ratio of subproblems			
	T_{BB4}/T_{BB1}[b]	T_{BB4}/T_{BB2}	T_{BB4}/T_{BB3}	T_{BB5}/T_{BB4}	N_{BB1}/N_{BB4}[c]	N_{BB2}/N_{BB4}	N_{BB3}/N_{BB4}	N_{BB4}/N_{BB5}
(2, 4)	0.719	0.742	0.742	0.634	0.266	0.293	0.522	0.156
(2, 6)	0.667	0.568	0.693	0.417	0.299	0.238	0.299	0.008
(2, 8)	0.682	0.613	0.859	0.276	0.583	0.140	0.068	0.007
(2, 10)	0.793	0.634	0.847	0.265	0.066	0.061	0.145	0
(3, 4)	0.729	0.594	0.741	0.660	0.212	0.197	0.257	0.001
(3, 6)	0.651	0.562	0.894	0.246	0	0.001	0.001	0.026
(3, 8)	0.761	0.667	0.843	0.194	0.007	0.005	0.028	0.002
(3, 10)	0.838	0.737	0.901	0.308	0.002	0.009	0.003	0

[a](number of machines, number of jobs).
[b]T_X denotes the CPU time required to solve a problem using algorithm X.
[c]N_X denotes the number of subproblems considered for a problem in algorithm X.

computing all three lower bounds takes additional computation time. In addition, the dominance properties devised in this study can be considered effective since the number of subproblems considered in BB5 was significantly smaller than that of BB4 and BB5 required much shorter computation time than the other B&B algorithms.

The effectiveness of the lower bounds as well as that of the dominance rules developed in this study are more clearly shown in Table 2, which gives the ratios of the CPU times for the algorithms and the numbers of subproblems considered in the B&B algorithms. As mentioned earlier, using three lower bounds (BB4) was better in terms of the computation time than using only one lower bound. Also, it can be seen from the comparison of BB4 and BB5 that the dominance properties developed in this study were very effective in pruning unpromising nodes in the B&B algorithm. This means that it was better to check whether or not the dominance conditions were satisfied although such checking required additional time.

To show the effect of (the tightness of) due dates on the performance of the best performing algorithm (BB5), results of the tests are arranged for four different pairs of values for (α, β) in Table 3. As can be observed from the table, the performance of the algorithm was not much affected by the due-date tightness, although it took slightly longer time to solve a problem with $(\alpha, \beta) = (0.5, 1.0)$ than a problem with $(\alpha, \beta) = (0, 0.4)$. This may be because the dominance properties are not related with due dates. The dominance properties and lower bounds were effective regardless of due date tightness.

Table 4 shows the effect of the length of setup time on the performance of BB5. The table shows the average number of sub-jobs split from a job in the solutions as well the average CPU time and the average percentage ratio of subproblems generated. As in the case for the effect of due dates, the performance of the algorithm was not much affected by the length of setup time. A reason may be that the dominance properties given in this study are not related with setup time.

Table 3
Effects of the due-date tightness on the performance of BB5

Problem size[a]	$(\alpha, \beta) = (0, 0.4)$		$(\alpha, \beta) = (0, 0.8)$		$(\alpha, \beta) = (0, 1.2)$		$(\alpha, \beta) = (0.5, 1.0)$	
	PRSC[a]	CPUT[a]	PRSC	CPUT	PRSC	CPUT	PRSC	CPUT
(2, 4)	3.35×10^{-3}	0.2	3.64×10^{-3}	0.2	4.02×10^{-3}	0.2	4.25×10^{-3}	0.2
(2, 6)	2.61×10^{-5}	4.4	2.71×10^{-5}	4.6	3.03×10^{-5}	5.1	3.28×10^{-5}	5.4
(2, 8)	3.32×10^{-7}	52.9	3.65×10^{-7}	55.1	4.05×10^{-7}	60.3	4.25×10^{-7}	64.3
(2, 10)	5.51×10^{-10}	613.5	5.83×10^{-10}	653.6	6.37×10^{-10}	736.9	6.69×10^{-10}	754.9
(3, 4)	5.28×10^{-5}	0.2	5.53×10^{-5}	0.2	6.40×10^{-5}	0.2	6.47×10^{-5}	0.2
(3, 6)	1.38×10^{-7}	4.4	1.48×10^{-7}	4.8	1.65×10^{-7}	5.4	1.68×10^{-7}	5.7
(3, 8)	7.95×10^{-9}	55.6	8.22×10^{-9}	56.9	8.97×10^{-9}	59.6	9.01×10^{-9}	64.5
(3, 10)	1.06×10^{-14}	1198.6	1.13×10^{-14}	1238.0	1.21×10^{-14}	1379.6	1.45×10^{-14}	1472.1

[a]See the footnotes of Table 1.

Table 4
Effects of the length of setup time on the performance of BB5

Problem size[a]	Short setup time			Medium setup time			Long setup time		
	PRSC[a]	CPUT[a]	NSJ[b]	PRSC	CPUT	NSJ	PRSC	CPUT	NSJ
(2, 4)	3.96×10^{-3}	0.2	1.8	3.78×10^{-3}	0.2	1.6	3.57×10^{-3}	0.2	1.3
(2, 6)	3.01×10^{-5}	5.1	1.7	2.99×10^{-5}	4.9	1.6	2.83×10^{-5}	4.6	1.2
(2, 8)	4.07×10^{-7}	62.7	1.5	3.75×10^{-7}	55.9	1.4	3.68×10^{-7}	55.1	1.2
(2, 10)	6.38×10^{-10}	706.9	1.5	6.00×10^{-10}	690.8	1.3	5.66×10^{-10}	674.1	1.1
(3, 4)	6.37×10^{-5}	0.2	2.6	5.97×10^{-5}	0.2	2.4	5.78×10^{-5}	0.2	2.2
(3, 6)	1.64×10^{-7}	5.2	2.4	1.54×10^{-7}	5.0	2.3	1.51×10^{-7}	5.0	2.0
(3, 8)	8.84×10^{-9}	63.5	2.4	8.51×10^{-9}	57.9	2.2	7.92×10^{-9}	55.6	1.8
(3, 10)	1.25×10^{-14}	1352.1	2.3	1.18×10^{-14}	1318.6	2.1	1.13×10^{-14}	1298.5	1.7

[a]See the footnotes of Table 1.
[b]Average number of sub-jobs per job in the optimal schedules.

The number of sub-jobs split from a job increases as setup time decreases. Note that when a job is split into sub-jobs, at least one more setup is required compared with a case in which the job is not split into sub-jobs. Therefore, if setup time is long, it is not desirable to split jobs into sub-jobs.

To see the performance of BB5 on various problem sizes, we performed additional tests. For this series of tests, 1080 problems were randomly generated, 60 problems for each of all combinations of two levels for the number of machines 4 and 5, three levels for the number of jobs 4, 8, 12, and three levels for the number of unit-jobs of a job 4, 5 and 6. Other data were generated using the same method as the one used for the earlier experiments. The algorithm was terminated after 3600 s of CPU time to avoid excessive computation time required for this series of tests. Table 5 shows results of the tests. The algorithm gave optimal solutions for all problems except for the largest problems with $n = 12$ and $m = 5$, and $n = 12$ and $m = 4$. The computation time depends on the number of jobs as well as the number of machines.

To see computation time required for problems of larger sizes, we performed another series of tests (without a limit of CPU time for each problem). For this series of tests, 240 problems were randomly generated, five problems for each of all combinations of three levels for the number of machines 4, 8 and 12, four levels for the number of jobs 10, 15, 20 and 25 and four levels for the number of unit-jobs per each job 4, 6, 8 and 10. In these problems, due dates were generated with $(\alpha, \beta) = (0, 1.2)$ and setup time for a job was generated from the discrete uniform distribution with range [5,60]. Table 6 shows results of the tests. Computation time required for a problem increases quickly as the numbers of jobs and unit-jobs increase. From the results, it can be seen that the difficulty in solving the problems with the suggested B&B algorithm was affected by the numbers of jobs and unit-jobs slightly more than the number of machines.

Table 5
Results of tests on the performance of BB5 for different problem sizes

Problem size[a]	ACPUT[b]	MCPUT[c]	APRSC[d]	NPNS[e]
(4, 4, 16)	0.19	0.17	4.1×10^{-5}	0
(4, 4, 20)	0.24	0.23	9.2×10^{-5}	0
(4, 4, 24)	0.29	0.29	1.6×10^{-6}	0
(4, 8, 32)	83.24	80.75	2.6×10^{-9}	0
(4, 8, 40)	100.68	101.54	5.2×10^{-9}	0
(4, 8, 48)	146.38	156.22	8.4×10^{-9}	0
(4, 12, 48)	2150.67	1648.49	3.5×10^{-16}	5
(4, 12, 60)	2854.68	3485.12	2.7×10^{-17}	15
(4, 12, 72)	3468.57	3600	1.8×10^{-18}	45
(5, 4, 16)	0.22	0.13	7.7×10^{-6}	0
(5, 4, 20)	0.57	0.52	2.1×10^{-7}	0
(5, 4, 24)	1.08	1.06	5.3×10^{-7}	0
(5, 8, 32)	103.21	99.65	4.6×10^{-9}	0
(5, 8, 40)	184.39	180.67	2.4×10^{-10}	0
(5, 8, 48)	295.67	301.49	8.1×10^{-10}	0
(5, 12, 48)	2382.41	3515.06	8.2×10^{-19}	30
(5, 12, 60)	3521.58	3600	3.7×10^{-20}	45
(5, 12, 72)	3600	3600	8.3×10^{-20}	60

[a](number of machines, number of jobs, total number of unit-jobs).
[b]Average CPU time or lower bound on the average CPU time, which is computed assuming that the CPU time for a problem that has not been solved in 3600 s is 3600 s.
[c]Median of CPU times.
[d]Average (or median, in cases where the algorithm did not solve the problems to the optimality) percentage ratio of the number of subproblems considered to that of subproblems that should have been considered without the dominance conditions or lower bounds.
[e]Number of problems (among 60 problems) those were not solved in 3600 s.

Table 6
Results of tests on problems of larger sizes

Problem size[a]				
n	u_i	$m = 4$	$m = 8$	$m = 12$
10	4	1383.4[b]	1845.6	2153.6
	6	2256.7	2869.7	3983.6
	8	3459.6	4023.6	4825.7
	10	5236.8	6098.4	7324.9
15	4	2356.4	3185.3	4033.7
	6	4587.6	5834.8	8819.8
	8	6598.4	7846.6	9964.7
	10	8465.7	10 393.2	13 462.3
20	4	3368.9	5360.4	7244.5
	6	6782.0	8694.4	13 893.3
	8	9804.9	13 866.6	20 696.9
	10	13 007.1	17 577.9	24 136.3
25	4	6038.1	10 160.8	15 348.7
	6	9015.6	14 386.4	21 020.1
	8	14 078.8	20 971.3	28 821.1
	10	19 246.8	25 506.0	38 314.5

[a]m, n and u_i denote the number of machines, the number of jobs and the number of unit-jobs of a job, respectively.
[b]Average CPU time (seconds).

8. Concluding remarks

We considered the problem of scheduling jobs that are composed of multiple unit-jobs and can be split into subjobs, each with a number of unit jobs, on parallel machines for the objective of minimizing total tardiness of the

jobs. Considering special characteristics of the problem, we developed dominance properties and lower bounds on the tardiness of jobs for a partial schedule, and developed a B&B algorithm using them. Results of computational experiments showed that the suggested algorithm could find optimal solutions for problems with up to 4 machines and 12 jobs (and five machines and eight jobs) in a reasonable amount of CPU time.

The algorithms suggested in this paper may be useful for real PCB manufacturing systems from which this research is originated. However, further research is needed since actual problems in the real systems may be larger than what could be solved with the suggested B&B algorithm. For example, one needs to develop more effective dominance rules (related with due date and setup time) and tighter lower bounds. Alternatively, one can devise heuristic algorithms that give very good solutions for large problems within a reasonably short computation time.

References

[1] Xing W, Zhang J. Parallel machine scheduling with splitting jobs. Discrete Applied Mathematics 2000;103:259–69.
[2] Kim YD, Shim SO, Kim SB, Choi YC, Yoon HM. Parallel machine scheduling considering a job splitting property. International Journal of Production Research 2004;42:4531–46.
[3] Root JG. Scheduling with deadlines and loss functions on *k* parallel machines. Management Science 1965;11:460–75.
[4] Elmaghraby SE, Park SH. Scheduling jobs on a number of identical machines. AIIE Transactions 1974;6:1–13.
[5] Arkin EM, Roundy RO. Weighted-tardiness scheduling on parallel machines with proportional weights. Operations Research 1991;39:64–81.
[6] Azizoglu M, Kirca O. Tardiness minimization on parallel machines. International Journal of Production Economics 1998;55:163–8.
[7] Yalaoui F, Chu C. Parallel machine scheduling to minimize total tardiness. International Journal of Production Economics 2002;76:265–79.
[8] Liaw CF, Lin YK, Cheng CY, Chen M. Scheduling unrelated parallel machines to minimize total weighted tardiness. Computers and Operations Research 2003;30:1777–89.
[9] Shim SO, Kim YD. Minimizing total tardiness in an unrelated parallel-machine scheduling problem. Journal of the Operational Research Society 2005, to appear.
[10] Wilkerson LJ, Irwin JD. An improved algorithm for scheduling independent tasks. AIIE Transactions 1971;3:239–45.
[11] Dogramaci A, Surkis J. Evaluation of a heuristic for scheduling independent jobs on parallel identical processors. Management Science 1979;25:1208–16.
[12] Ho JC, Chang YL. Heuristics for minimizing mean tardiness for m parallel machines. Naval Research Logistics 1991;38:367–81.
[13] Alidaee B, Rosa D. Scheduling parallel machines to minimize total weighted and unweighted tardiness. Computers and Operations Research 1997;24:775–88.
[14] Koulamas C. Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem. Naval Research Logistics 1997;44:109–25.
[15] Kim YD. A backward approach in list scheduling algorithms for multi-machine tardiness problems. Computers and Operations Research 1995;22:307–19.
[16] Lee YH, Pinedo M. Scheduling jobs on parallel machines with sequence-dependent setup times. European Journal of Operational Research 1997;100:464–74.
[17] Park MW. A genetic algorithm for the parallel-machine total weighted tardiness problem. Journal of the Korean Institute of Industrial Engineers 2000;26:183–92.
[18] Blocher JD, Chhajed D. The customer order lead-time problem on parallel machines. Naval Research Logistics 1996;43:629–54.
[19] Yang J, Posner ME. Scheduling parallel machines for the customer order problem. Journal of Scheduling 2005;8:49–74.
[20] Serafini P. Scheduling jobs on several machines with the job splitting property. Operations Research 1996;44:617–28.
[21] Kim DW, Na DG, Chen FF. Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. Robotics and Computer Integrated Manufacturing 2003;19:173–81.
[22] Logendran R, Subur F. Unrelated parallel machine scheduling with job splitting. IIE Transactions 2004;36:359–72.
[23] Shim SO, Kim YD. Scheduling on parallel identical machines to minimize total tardiness. European Journal of Operational Research 2005, to appear.
[24] Brah SA, Hunsucker JL. Branch and bound algorithm for the flow shop with multiple processors. European Journal of Operational Research 1991;51:88–99.
[25] Chu C. A branch and bound algorithm to minimize total tardiness with different release dates. Naval Research Logistics 1992;39:265–83.
[26] Kondakci S, Azizoglu M, Kirca O. An efficient algorithm for the single machine tardiness problem. International Journal of Production Economics 1994;36:213–9.
[27] Kim YD. Minimizing total tardiness in permutation flowshops. European Journal of Operations Research 1995;85:541–55.

Sang-Oh Shim received both B.S. and M.S degrees from Korea Advanced Institute of Science and Technology (KAIST) in Industrial Engineering. He is a Ph.D. candidate in the Department of Industrial Engineering, KAIST. His research areas include operation scheduling and production control.

Yeong-Dae Kim is a professor at the Department of Industrial Engineering, KAIST. He received a B.S. degree from Seoul National University, an M.S. degree from KAIST in Industrial Engineering, and a Ph.D. degree from the University of Michigan in Industrial and Operations Engineering. His research areas include design and operation of manufacturing systems, operations scheduling and production planning and inventory management.