Int J Adv Manuf Technol (2012) 59:325–333
DOI 10.1007/s00170-011-3489-x

ORIGINAL ARTICLE

Due-date scheduling on parallel machines with job splitting and sequence-dependent major/minor setup times

Taeho Park · Taehyung Lee · Chang Ouk Kim

Received: 24 November 2010 / Accepted: 20 June 2011 / Published online: 10 June 2011

Abstract This paper addresses job scheduling problems with parallel machines. To satisfy customers better in a manufacturing company, meeting due dates has been an important performance metric. Besides the numerous other factors affecting due date satisfaction, the splitting of a job through parallel machines can contribute to the reduction of production lead time, resulting in less job tardiness against their due dates. Thus, this paper presents heuristic algorithms for minimizing total tardiness of jobs to meet their due dates in a manufacturing shop with identically functioning machines. The algorithms take into account job splitting and sequence-dependent major/minor setup times. The performance of the proposed heuristics is compared with that of past three algorithms in the literature.

Keywords Parallel machine scheduling · Due date · Sequence-dependent major/minor setup times · Total tardiness · Job splitting · Heuristics

1 Introduction

Scheduling problems on multiple parallel machines have been proved to be NP-hard [4]. Consequently, it is unlikely that a polynomial-time optimization scheduling algorithm for their solutions will be developed. In addition, job splitting greatly increases the complexity of scheduling problems. This paper will present heuristic algorithms that produce near-optimal schedules to minimize total job tardiness with identically functioning parallel machines in consideration of job splitting and major/minor setup times. A job is allowed to be split into sub-batches, and the sub-batches can be processed on different machines. However, all sub-batches of a job are constrained to be completed at the same time. The synchronization of sub-batch completion times on parallel machines is a reasonable production practice because allowing for the unequal completion times of sub-batches may incur carrying higher work-in-process inventories, longer flow times, and a more complicated shop floor control [10]. This operational policy is also employed on the ground that no matter how early sub-batches of a job are finished, meeting the job due date is determined by the finish time of a sub-batch to be completed last.

Interest in setup times has been evident in the literature that delineates applications diversified to different manufacturing industries [6, 9, 13, 14, 16, 18]. Two different treatments for handling setup times can be observed in the literature: sequence-dependent or independent setup times. Research associated with sequence-dependent setup times indicates that setup times vary, depending upon the sequence of jobs to be processed on a machine. On the other hand, research on sequence-independent setup times assumes that the same setup times are supposed to be incurred regardless of the sequence of jobs processed on a machine. So [16] and Wittrock [18] introduced the concept of minor and major setup times. While a minor setup time is incurred on a machine when switching a job to another in the same part family, a major setup time is required to process the job if its part family is different from the part family which the previous job on the machine belongs to.

As processing methods are improved through the efforts of continuous improvement and as more advanced automatic machines (e.g., numerically controlled machines) are used on the shop-floor, setup times could have been

T. Park · T. Lee · C. O. Kim (✉)
Yonsei University,
Seoul, South Korea
e-mail: kimco@yonsei.ac.kr

dramatically reduced. The reduced setup times allow for splitting a job into small sub-batches; the sub-batches can then be simultaneously processed on more than one machine. Consequently, job splitting becomes a promising production practice to meet the due dates of orders or to shorten production flow time wherever the job splitting can be justified. Wittrock [18] and Rajgopal and Bidanda [13] developed heuristic algorithms for splitting a job through identical machines to minimize makespan in consideration of both job splitting and major/minor setup times. Rajgopal and Bidanda also designed two other algorithms to account for minimizing the average flowtime of parts. They assume that major setup times are identical for all part families and minor setup times are the same for all part types. Xing and Zhang [19] investigated job splitting in parallel machine scheduling and presented a linear programming formulation for minimizing makespan with sequence-independent setup times. Yalaoui and Chu [20] reported a two-stage heuristic for minimizing makespan in the presence of job splitting and sequence-dependent setup times. Tahar et al. [17] improved the two-stage heuristic by incorporating a linear programming-based improvement at the second stage. Beraldi et al. [2] studied a parallel machine scheduling problem to minimize total setup cost with sequence-dependent setup costs and job splitting. They developed a new decomposition heuristics based on rolling horizon and fix-and-relax approaches.

Considering the current inclination in a manufacturing strategy towards just-in-time (JIT), the performance measure of total job tardiness in light of on-time delivery for better customer satisfaction is more appealing in the customer-focused competitive market. Balasubramanian et al. [1] attempted to minimize total weighted job tardiness in parallel machine environments in which several jobs can be processed on a machine at the same time but jobs of different families cannot be processed together. Radhakrishnan and Ventura [12] and Mason et al. [11] considered the sum of tardiness and earliness of jobs as a performance measure to realize JIT in parallel machine scheduling problems. Chen [3] developed a heuristic for minimizing total job tardiness with sequence- and machine-dependent setup times.

When job splitting was considered in the scheduling problems, few other researchers than Serafini [14], Kim et al. [7, 8], Logendran and Subur [10], and Shim and Kim [15] included the due date factor into the scheduling problems. Serafini dealt with a parallel machine scheduling problem with job splitting encountered in scheduling looms in a textile factory. He developed an efficient method using network flow techniques to minimize maximum weighted job tardiness in textile weaving operations with multiple uniform looms. He assumed that switching time for weaving from one article to another on the same loom is ignorable. Kim et al. [7] proposed a simulated annealing approach that minimizes total job tardiness with sequence-dependent setup times. The scheduling scheme proposed by Kim et al. [8] was designed for minimizing total job tardiness with sequence-independent setup times. Their scheme consists of two phases: the first for obtaining an initial solution without considering job splitting, and the second for improving the initial solution by splitting a job to sub-batches (i.e., sub-job) and rescheduling the sub-batches by using the combinations of job/sub-batch selection rules and machine selection rules. Logendran and Subur addressed the simultaneous completion of sub-batches of each job with dynamic job releases and machine availability considerations. However, a job is allowed to be split into two sub-batches in their model. They presented a heuristic that identifies an initial solution by composite dispatching rules and improves the solution using tabu search to find the best solution. Shim and Kim developed a branch and bound algorithm to minimize total job tardiness and demonstrated that the algorithm solves problems of moderate sizes with a reasonable computation burden.

2 Heuristics

2.1 Problem description

The multiple parallel machines scheduling problem discussed here have M identical machines and N ($>M$) jobs, each of which has small to medium batch sizes. While a minor setup time is needed on a machine to process jobs in the same part family as the previous one, a relatively significant amount of major setup time is required to switch from a job in a part family to another in a different part family. Furthermore, the major setup times are dependent upon the sequence of part families on machines. No setup time is necessary to process parts or workpieces in the same job. Although jobs do have different processing time requirements, it is assumed that processing time for parts of a job is the same over different machines.

The parallel machine scheduling algorithm presented in this research consists of two parts: (1) sequencing jobs to determine which job should be processed first and (2) splitting a job to sub-batches and assigning the sub-batches to machines. Section 2.2 will describe three different job sequencing algorithms. Then Section 2.3 will present a heuristic, called SPLIT, for splitting a job to sub-batches and then allocating the split sub-batches to machines with a rationale of saving setup times and balancing workload on machines. Algorithm SPLIT is embedded in the job sequencing algorithms because the start times and completion times of jobs predicted by SPLIT are used for sequencing jobs.

Algorithm SPLIT treats processing of discrete parts as a continuous process, which allows a fraction of a part on a machine. Consequently, the completion time of a job calculated by the algorithm might not be exactly the same as that in an actual production. Therefore, after jobs are scheduled on the machines with the job sequencing algorithms and SPLIT, a refinement task is necessary for switching the manufacturing process back to the original discrete process by removing a fraction (if any) of a part which might have been created by SPLIT. Section 2.4 will present the refinement algorithm.

The following is a list of notations used throughout this paper:

N	Number of batch jobs (part types).
M	Number of identical parallel machines.
F	Number of part families.
b_i	Batch size of job i.
t_i	Time required to process a workpiece of job i.
d_i	Due date of job i.
$F(i)$	Part family type of job i.
$S_{H',H}$	Major setup time required to switch part family H' to part family H.
s_i	Minor setup time required to switch to job i.
A	Set of unscheduled jobs.
$A(k)$	k-th job in set A.
$L(j)$	Last job assigned to machine j.
$\mathrm{ST}_{i,j}$	Starting time of job i on machine j.
ET_i	Ending time of job i.
J_i	Set of machines to process job i.

2.2 Job sequencing

2.2.1 Heuristic 1 (slack-based heuristic)

The procedure presented in this section sequences jobs based on their "slack times," i.e., the amount of time left from completion of jobs to their due dates. The predicted completion time of a job is calculated by using algorithm SPLIT which determines the allocation of a job onto machines and the completion time of the job. The algorithm of heuristic 1 is as follows:

Step 1. Set $A=\{1, \ldots, N\}$, $L(j)=0$ $(j=1,\ldots,M)$, $(j=1, \ldots, M)$, and $\mathrm{ET}_0=0$.
Step 2. Perform the allocation procedure, SPLIT, with a batch job $A(k)$ to determine machines to process $A(k)$ and to estimate the completion time of $A(k)$ for $k = 1, \ldots, |A|$.
Step 3. Calculate the slack time ($\mathrm{SLACK}_{A(k)}$) of $A(k)$ by $\mathrm{SLACK}_{A(k)} = d_{A(k)} - \mathrm{ET}_{A(k)}$ for $k = 1, \ldots, |A|$.
Step 4. Select job i^* with the least slack time and allocate it to machines as determined by the allocation procedure.
Step 5. Update $A=A-\{i^*\}$ and $L(j)=i^*$ for $j \in J_{i^*}$ and reset all information of jobs in A.
Step 6. Repeat steps 2–5 until A becomes ϕ.

2.2.2 Heuristic 2 (dynamic scheduling window-based heuristic)

This heuristic algorithm divides a scheduling horizon into small scheduling time buckets, or "scheduling windows," and then creates a schedule of jobs within each scheduling window. The division of the entire scheduling horizon into small scheduling windows attempts to obviate the possibility that jobs with early due dates will be processed late, or that jobs with late due dates will be processed too early. Since there will be significant setup time involved to switch from one part family to another, the algorithm will group jobs based on their part families and then determine a sequence of the part families. Once the part families are sequenced, jobs in each part family will be scheduled, based on the ratio of due date to workload.

Suppose that the entire scheduling horizon is divided into T number of windows, $\Sigma_r (r = 1, \ldots, T)$. The time span for each window varies and is determined by assigning a similar amount of work to all scheduling windows. A procedure for dividing a scheduling horizon to scheduling windows is described in Appendix 1. The algorithm of heuristic 2 is as follows:

Step 1. Divide the entire scheduling horizon into T scheduling windows by the procedure described in Appendix 1.
Step 2. Set $r=1$.
Step 3. Group jobs in Σ_r by their part families.
Step 4. For each part family K in Σ_r, calculate the sum of due date per workload (SDDW) of jobs in K by $\sum_{i \in r} d_i/(s_i + b_i \times t_i)$.
Step 5. Select a part family with the smallest SDDW among unassigned part families.
Step 6. Make a sequence of jobs in the part family in ascending order of due date per workload (i.e., $d_i/(s_i + b_i \times t_i)$ for job i).
Step 7. Assign the jobs using the allocation procedure, SPLIT, described in Section 2.2.
Step 8. Repeat steps 4–6 until all part families in Σ_r are scheduled.
Step 9. Repeat steps 2–7 by increasing r by one until all scheduling windows are considered.

2.2.3 Heuristic 3 (estimated latest starting time-based heuristic)

In this heuristic algorithm, jobs are sequenced, based on the estimated latest starting time calculated by the algorithm

SPLIT. The latest starting time of job i is estimated under assumptions that (1) $M/2$ (or $(M+1)/2$ if M is odd) number of machines will be assigned to process the job and (2) setup time required for the job on all assigned machines will be calculated by $\left(2 \times s_i + \left(\Sigma_K S_{K,F(i)}\right)/(F-1)\right)/2$, where Ks are part family types different from $F(i)$, and F is the total number of different part families. The algorithm of heuristic 3 is as follows:

Step 1. Estimate latest starting times (LST_i) of job i (i=1, …, N) by

$$d_i - \left(\frac{b_i \times t_i}{m} + \left(2 \times s_i + \frac{\sum_{K(K \neq F(i))} S_{K,F(i)}}{F-1}\right)/2\right)$$

where, m is $M/2$(or $(M+1)/2$) if M is even (odd).

Step 2. Sort jobs in ascending order of LST_i.

Step 3. Allocate the sorted jobs on machines using the SPLIT until all jobs are assigned.

2.3 Job splitting and assignment to machines (algorithm SPLIT)

From a rationale of saving setup times and balancing machine workload, algorithm SPLIT is developed in this research to split a job into sub-batches and then allocate them to machines. Sub-batches of a job that are split among machines will be finished at the same time. Two decision issues in the job allocation problem pertaining to trade-off between job splitting and setup time savings are: (1) the maximum total setup time (MST_i) allowed for job i and (2) an allowed ratio of setup time to total processing time (RST_i) on each machine. The first issue is concerned with the amount of total setup time to be spent on a job, compared to the total processing time needed for the job in the entire production. When job setup time is high, it is unlikely that the job will be split and processed on many machines, unless it is extremely urgent. The second issue deals with justifying the setup time required for a job on a machine, compared to the total processing time spent on that machine. Suppose that the setup time of a job and processing time of a part of that job on a machine are 20 and 3 min, respectively. In such a situation, it would not be advantageous to set up a machine to process just a few parts of the job. MST_i and RST_i for job i would be calculated as follows:

$$\mathrm{MST}_i = \max\left(\max_j \left(S_{F(L(j)),F(i)}\right) + s_i, \alpha \times b_i \times t_i\right) \quad (1)$$

$$\begin{aligned} \mathrm{RST}_i &= 1, \text{if } \max_j \left(S_{F(L(j)),F(i)}\right) + s_i \geq \beta \times b_i \times t_i \\ &= \beta, \text{otherwise} \end{aligned} \quad (2)$$

where, α and β are coefficients with a range of [0, 1].

In addition to reducing setup time requirements in job splitting, the algorithm SPLIT attempts to prevent a machine from being assigned to processing less than a part-worth work of a job. (It should be remembered that the SPLIT treats processing of discrete parts as a continuous process, allowing a fraction of a workpiece to be processed on a machine.) Thus, although it is allowed to process a fraction of a part on the machine at the first phase of scheduling, the total work of the job on the machine has to be at least a part (see step 10 of the SPLIT.)

While restricting setup time requirements within the limits described above, the SPLIT also tries to balance work among machines. First, it attempts to assign a new job i to a machine with the least workload (see step 2 of the SPLIT). If the machine (called machine j) requires a major setup time, the SPLIT will search for another machine which has completed a job in the same part family as that of the new job i. If it finds such a machine (called machine j*) and $S_{F(L(j)),F(i)} + ET_{L(j)} > ET_{L(j^*)}$, it will assign job i to machine j* instead of machine j. (See step 7 of the SPLIT.)

The algorithm SPLIT is as follows:

Step 1. Set $J_i = \phi$ and total setup time required for job i=0.

Step 2. [Balancing workload] Sort machines in the ascending order of $ET_{L(j)}$ for j=1, …, M, and create a set, B, which includes machines in the sorted order.

Step 3. Set r=0.

Step 4. Set r=r+1.

Step 5. Select machine j which is the rth machine in B.

Step 6. Set r_0 to be r and go to step 7 if the part family type of job i is not the same as that of $L(j)$. Otherwise, go to step 8.

Step 7. [Saving setup time and balancing workload] Search, by increasing r_0 by 1 at a time, for a machine such that the last job on the machine belongs to the same part family as job i. If such a machine is not found, reset r to be r_0 and go to step 8. Otherwise, let j* and r* be the machine number and the index of the order corresponding to the machine. If $S_{F(L(j)),F(i)} + ET_{L(j)} > ET_{L(j^*)}$ or no job has yet been assigned to machine j*, then reset r to r* and j to j*. Go to step 8.

Step 8. Decide setup time required for job i on machine j by

$$\begin{aligned} \mathrm{SETUP} &= s_i \text{ if job } i \text{ belongs to the same part family as } L(j), \\ &= s_i + S_{F(L(j)),F(i)} \text{ otherwise.} \end{aligned}$$

Step 9. Estimate total processing time (TPT) for job i on machine j by

$$\begin{aligned} \mathrm{TPT} &= b_i \times t_i \text{ if } J_i = \phi \\ &= |J_i| \times \left\{\mathrm{ET}_i - \left(\mathrm{ET}_{L(j)} + \mathrm{SETUP}\right)\right\}/ \\ &\quad (|J_i| + 1) \text{ otherwise.} \end{aligned}$$

The second equality shown above is to split job i currently allocated to $|J_i|$ number of machines to $|J_i|+1$ machines which include machine j. (Refer to Appendix 2 for details of the second equality.)

Step 10. [Validation of processing a job on a machine] Go to step 14 if TPT of job i is less than its unit processing time of t_i. Otherwise, go to step 11.

Step 11. Calculate $\mathrm{TST}' = \mathrm{TST} + \mathrm{SETUP}$.

Step 12. Calculate MST_i and RST_i using Eqs. 1 and 2 if $J_i=\phi$.

Step 13. [Checking appropriateness of job splitting in light of setup time requirements] If $\mathrm{TST}' \leq \mathrm{MST}_i$ and $\mathrm{SETUP} \leq \mathrm{RST}_i \times \mathrm{TPT}$, then (a) add machine j to J_i, (b) update $\mathrm{TST} = \mathrm{TST}'$ and $\mathrm{ET}_i = \mathrm{ET}_{L(j)} + \mathrm{TPT} + \mathrm{SETUP}$, (c) eliminate machine j from B, and (d) set $r=r-1$.

Step 14. Terminate this procedure if $B=\phi$.

Step 15. Go to step 4 if $r < |B|$.

Step 16. Go to step 3 if machine j was removed from B in step 13. Otherwise, terminate this procedure.

The above steps will determine (1) the allocation of job i to machines and (2) the start time and completion time of the job. This information will be fed into the job sequencing algorithms described previously.

2.4 Refinement algorithm

Since the SPLIT algorithm attempts to split the processing time of a job through calculation using the continuous process concept for discrete parts, it is most likely that a fraction of a part would be assigned to a machine. Thus, this refinement algorithm removes the fraction of a part of a sub-batch from the machine and augments it to the sub-batch on the other machine. The essence of this procedure is as follows:

1. If only a fraction of a part of a job is assigned to a machine, the job will be removed from the machine to save setup times.
2. Fractional parts of a job will be first rounded off; then, if the total number of parts is greater than the batch size of the job, the fractional parts will be trimmed from machines with heavy workloads one at a time. If it is less than the batch size, the fractional parts will become full part(s) and added to machines with small workloads.

Following is the procedure for the refinement algorithm.

Step 1. Sort jobs in ascending order of starting times: $\sigma_1, \ldots, \sigma_r, \ldots, \sigma_N$, where σ_r stands for rth job in that order.

Step 2. Set $r=0$.

Step 3. Set $r=r+1$ and i be a job index corresponding to σ_r.

Step 4. Calculate the number of workpieces ($W_{i,j}$) of job i to be processed on machine j ($j \in J_i$) by

$$W_{i,j} = \frac{\text{Finish time} - \text{Start time}}{\text{Processing time}}$$

Step 5. Perform a following fraction-rounding procedure for all machines in J_i:

If $W_{i,j}<1$, then remove machine j from J_i.

Otherwise, obtain a fraction of $W_{i,j}$: $f_{i,j} = W_{i,j} - \mathrm{Trunc}\,(W_{i,j})$, where $\mathrm{Trunc}(W_{i,j})$ stands for the largest integer no greater than $W_{i,j}$.

Step 6. Update the number of workpieces to be processed on machine j ($j \in J_i$) as follows: If $f_{i,j}<0.5$, then $W_{i,j'} = \mathrm{Trunc}\,(W_{i,j})$. Otherwise, $W_{i,j'} = \mathrm{Trunc}\,(W_{i,j}) + 1$.

Step 7. Adjust the total number of workpieces ($\Sigma_j W_{i,j'}$) of job i when it is greater or less than the batch size of b_i as follows with workload balancing in mind:

If $\Sigma_j W_{i,j'} > b_i$, then remove a workpiece one by one from heavily loaded machines as compared with the other machines in J_i until $\Sigma_j W_{i,j'}$ reaches b_i.

If $\Sigma_j W_{i,j'} < b_i$, then add a workpiece one by one to less-loaded machines as compared with the other machines in J_i until $\Sigma_j W_{i,j'}$ reaches b_i.

Step 8. Update starting and finishing times of job i and the other jobs to be processed later on the machines in J_i.

Step 9. Go to step 3 until r becomes N.

3 Analysis of performance

3.1 Statistical analyses and calibration

An empirical testing approach to evaluating the three proposed heuristic algorithms was employed, running the algorithms on 12 sample problems. The problems were selected by changing the number of families (5, 10, and 15) and the number of parallel machines (3, 5, 8, and 12). Five instances were created for each problem. The SPLIT job allocation algorithm involves two parameters, α and β, which are used for calculating the maximum total setup time allowed for jobs and an allowed ratio of setup time to processing time on each machine, respectively. The effects of the two parameters on the performance of the SPLIT algorithm were investigated by changing the values (0.05, 0.1, 0.15, 0.2, 0.3, 0.4, and 0.5). Another parameter indicating the number of scheduling windows involved in heuristic 2 (dynamic scheduling window-based heuristic) was also tested with five different window numbers (three to seven scheduling

windows). Thus, a full factorial design of experiments with 20,580 test runs was set up as follows:

Factors	Levels	Level values
1. Number of families	3	5, 10, 15
2. Number of parallel machines	4	3, 5, 8, 12
3. Parameter α	7	0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5
4. Parameter β	7	0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5
5. Heuristic algorithms	7	H1, H2 (3 W–7 W), H3

where, Hk means heuristic k, and kW stands for k number of windows.

To generate the number of jobs in each family, the batch size of jobs, the major and minor setup times, and the processing times of jobs for each instance, the following scheme was adapted from [15] with a slight modification:

Number of part types in a part family, uniformly distributed over [1, 10]
Batch size (number of workpieces) of a part type, uniformly distributed over [30, 90]
Major setup times, uniformly distributed over [20, 50]
Minor setup times, uniformly distributed over [1, 10]
Processing times, uniformly distributed over [2, 10].

The minute was used as a time unit. For the purpose of convenience, the time unit will be omitted throughout this paper. The due dates of jobs were generated from a due date generation scheme proposed by Dogramaci and Surkis [5]. The scheme distributes due dates uniformly between

$$P \times (1 - \tau - R/2) \text{ and } (1 - \tau + R/2),$$

where, τ is a mean tardiness factor, R is a due date range parameter, P is calculated by

$$\left(\sum_{i=1}^{N} b_i \times t_i + N \times M \times \sum_{t=1}^{F-1} \sum_{t'=t+1}^{F} S_{t,t'} / (2 \times F \times (F-1)) + M \times \sum_{i=1}^{N} s_i / 2 \right) / M$$

while a larger value of τ generates tighter due dates, a larger R spreads the due dates wider. In this experiment, R and τ were set to 1.0 and 0.5, respectively. The heuristic algorithms were programmed in Java and run on a personal computer with a Pentium CPU.

Analysis of variance for the numerical results obtained from the design of experiments revealed that all five factors shown above have significant effects on the performance measure of total job tardiness at the significance level of 0.01. In addition, Tukey's studentized range test was conducted to identify which algorithm performs better than the others. The result in Table 1 indicates that heuristics 1 and 3 perform better than heuristic 2, and the two algorithms are not significantly different at the significance level of 0.05. Although heuristic 2 divides the entire scheduling horizon into small scheduling windows to prevent jobs with early (late) due dates from being processed late (too early), it causes more major setup times required because part families in each scheduling window are scheduled separately from those in the other scheduling windows. Thus, the lengthened production flow times of jobs seem to result in worse performance in meeting due dates than the other two heuristic algorithms. To see how parameters α and β affect the performance of heuristics 1 and 3, mean total job tardiness was calculated for each parameter combination. The result is shown in Fig. 1. From the figure, it is recommended that the values in ranges of [0.05, 0.15] and [0.1, 0.2] be used for parameters α and β, respectively. This experiment suggests (α=0.1, β=0.15) as an optimal combination.

Table 1 Tukey's studentized range test for heuristic algorithms

Tukey grouping	Mean of total job tardiness	Heuristics (W)
A	18,936.7	H2 (3)
B	17,135.2	H2 (4)
C	15,776.3	H2 (5)
D	14,615.0	H2 (6)
E	14,234.2	H2 (7)
F	12,393.6	H1
F	12,107.4	H3

Heuristic algorithms with the same Tukey grouping letter are not significantly different at the significant level of 0.05

3.2 Performance comparison

We conducted two experiments for performance comparison. When job splitting and the synchronization of sub-batch completion times are considered in the scheduling problem, few other researchers than Logendran and Subur [10] included the due date factor into the scheduling problem. However, Logendran and Subur allow a job to be split into two sub-batches in their model, which will construct schedules with larger total job tardiness than those of the proposed heuristics that allow splitting a job into arbitrary number of small sub-batches. Therefore, the first comparison was conducted with two algorithms developed by Kim et al. [8] and Lee and Pinedo [9]. The two existing algorithms do not accommodate job synchronization con-

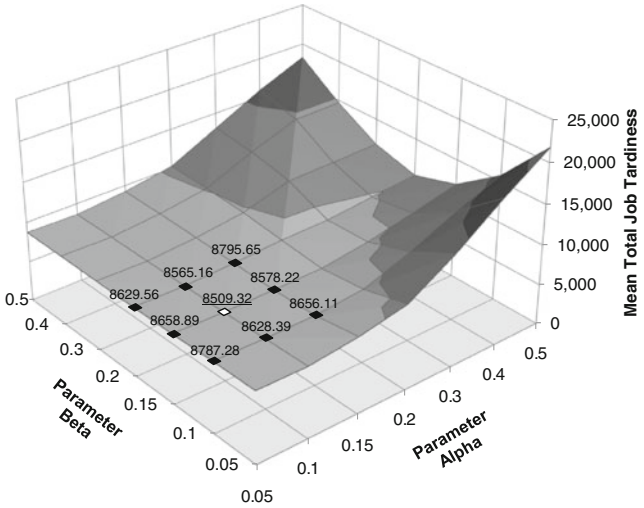

Fig. 1 Mean total job tardiness obtained from heuristics 1 and 3 over parameters α and β

straint. If our heuristics shows performance comparable to the two algorithms through the experiment, then the efficiency of our heuristics will be verified.

Kim et al. allow multiple job-splitting, but considers only sequence-independent setup times. On the other hand, the algorithm developed by Lee and Pinedo, which is called apparent tardiness cost with setup (ATCS), is designed to reflect sequence-dependent setup times in the scheduling, but it does not consider job splitting. Kim et al. presented a variant of ATCS, denoted as modified ATCS (MATCS) in their paper, which allows job splitting so as to demonstrate the performance efficiency of their algorithm. Among two dozens of dispatching rule combinations suggested by Kim et al., we chose two combinations: method1 with JS2 job/sub-job selection rule and MS3 machine selection rule, and method2 with JS1 job/sub-job selection rule and MS3 machine selection rule. The selected combinations demonstrated the best performance in their experiment (refer to p. 4539 of [8] for further information). Hereafter, we abbreviate the two dispatching combinations as method1-JS2-MS3 and method2-JS1-MS3. For the comparison experiment, we considered two cases: one with sequence-independent setup times and the other with sequence-dependent setup times. The former compares the performance of the proposed heuristic algorithms, method1-JS2-MS3, method2-JS1-MS3, and MATCS, while the latter tests the proposed heuristic algorithms and MATCS. For each case, 12 scheduling problems were considered by changing part family size (5, 10, and 15) and the number of machine size (3, 5, 8, and 12). Five instances were created for each problem. Thus, the total number of problem instances was 60 for each case. The computational times were within a few seconds, so we do not report the times in this paper. Based on the experiment result in Section 3.2, parameters α and β were set to 0.1 and 0.15, respectively.

Table 2 gives the experimental results. For each algorithm, the performance measures are the ratio of best solution to the total problem instances and the mean of relative deviations to the best solutions of all instances. As shown in Table 2, when sequence-independent setup times were considered, heuristic 3 demonstrated the best performance: it accounts for 56.67% of the best solutions, and its mean relative deviation is 13.95% for the entire instances. However, the mean relative deviation (19.31%) of heuristic 1 is not so different from that of heuristic 3 compared to the deviations of the other three algorithms. Method2-JS1-MS3 showed the worst performance in this case. Similar result can be observed for the case of sequence-dependent setup times. Heuristic 3 generated the best solutions for 60% of the 60 instances, while MATCS failed to generate any single best solution. The results of the two cases are interesting in that, even the constraint of job completion time synchronization is not imposed on method1-JS2-MS3, method2-JS1-MS3, and MATCS, the performance of the algorithms is not as satisfactory as that of the heuristics presented in this paper.

For the same problem instances in the previous experiment, we conducted the second experiment that compares the performance of the proposed heuristics with simulated annealing (SA) approach with job splitting [7]. We modified the objective function of the approach so that the sub-batch completion times of a job are as close to equal as possible. In specific, the modified objective function is a weighted sum of total job tardiness and penalty for the violation of the synchronization constraint. We considered two weights (0.5 and 0.9) for the penalty. The more weight is given to the penalty, the tighter the constraint is enforced. In the previous experiment, heuristic 3 showed the best performance. So, the performance measure of this experiment was defined the relative total job tardiness to heuristic 3. Also, computation time was measured. The results of the experiment are summarized in Table 3. In the table, the performance of SA is the average of results for the two penalty weights. As shown in the table, SA showed better performance only in the case of sequence-independent setup with 36.83% violation of the synchronization constraints. Furthermore, its execution time was 4.45 h on average while heuristic 3 took 1 ms. From the observation, we conclude that the proposed three heuristic algorithms show good performance results with short computation times.

4 Conclusions

Much attention has been paid to reducing setup times for the newly emerging manufacturing strategies of product flexibility and on-time delivery. Reducing setup time allows

Table 2 Performance comparison results (α=0.1, β=0.15)

	Optimum Ratio (mean relative deviation)	
	Sequence-independent setup (%)	Sequence-dependent setup (%)
Heuristic 1	25% (19.31)	33.33% (7.93)
Heuristic 2	3.33% (93.62)	6.67% (75.9)
Heuristic 3	56.67% (13.95)	60% (4.4)
Method1-JS2-MS3	10% (252.61)	N/A
Method2-JS1-MS3	1.67% (255 95)	N/A
MATCS	3.33% (944.36)	0% (864.88)

Each solution of heuristic 2 was obtained after five scheduling window sizes (3, 4, 5, 6, and 7) were tested

for splitting a production job into small batches and then processing a job on more than a machine simultaneously at a manufacturing workstation with multiple identically functioning machines. Thus, this research addressed job scheduling problems with identical machines considering job splitting, major/minor setup times, and due dates. Three heuristic algorithms (slack-based heuristic, dynamic scheduling window-based heuristic, and estimated latest starting time-based heuristic) were developed to minimize total job tardiness. Algorithm SPLIT which is embedded into the above three heuristics was also presented in this research to divide a job into sub-batches and then assign them to machines. The primary function of the algorithm is to split a job to more than one machine while limiting total setup times required and balancing the workloads of machines.

The performance of the heuristics was evaluated by varying the number of part families, the number of machines, and parameter values. The statistical analysis for the performance results showed that slack-based heuristic (heuristic 1) and estimated latest starting time-based heuristic (heuristic 3) are not significantly different in their performance. The two heuristics performed better than dynamic scheduling window-based heuristic (heuristic 2). The parameters, α and β, had significant effects on total job tardiness, and the values in [0.05, 0.15] and [0.1, 0.2] are recommended for α and β, respectively. Finally, throughout comparison experiments with three past algorithms, it was demonstrated that the performance of the proposed heuristics was better than that of the three algorithms.

Appendix 1. A procedure for dividing a scheduling horizon into scheduling windows

The following describes a procedure for dividing a scheduling horizon to T number of scheduling windows:

Step 1. Calculate the sum of minor setup and total processing requirements of jobs by using $\Sigma_i(s_i + b_i \times t_i)$.
Step 2. Calculate the average workload (AWL) in each scheduling window by dividing the sum by T.
Step 3. Set r=0.
Step 4. Set r=r+1.
Step 5. Assign a job with the earliest due date to Σ_r and repeat it until the total minor setup and processing time of jobs in Σ_r reaches AWL.
Step 6. Continue steps 4 and 5 until all jobs are assigned to one of the scheduling windows.

Appendix 2. Description of splitting a job onto machines

Supposing that job i has currently been split to $|J_i|$ number of machines as shown in Fig. 2a, algorithm SPLIT calculates the amount of work for job i to be processed on another machine, called machine j. When job i is processed on machine j, the work for job i after the time of $ET_{L(j)}$+SETUP that is currently allocated to $|J_i|$ number of machines needs to be split to $(|J_i| + 1)$

Table 3 Performance comparison between the proposed heuristics and simulated annealing (SA) approach

	Relative total job tardiness to heuristic 3 (mean computation time)		Ratio of synchronization constraint violation	
	Sequence-independent setup	Sequence-dependent setup	Sequence-dependent setup (%)	Sequence-independent setup (%)
Heuristic 1	106.46% (51 ms)	105.92% (113 ms)	0	0
Heuristic 2	134.08% (1 ms)	134.43% (1 ms)	0	0
Heuristic 3	100% (1 ms)	100% (1 ms)	0	0
SA	96.76% (4.45 h)	104.67% (4.67 h)	36.83	38.71

Each solution of heuristic 2 was obtained after five scheduling window sizes (3, 4, 5, 6, and 7) were tested

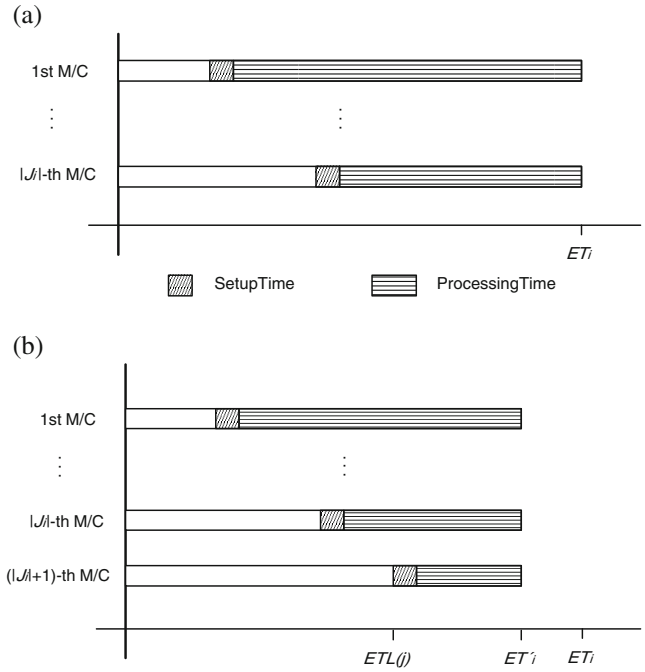

Fig. 2 Illustration of job split: **a** job i is split to $|J_i|$ number of machines and **b** job i split to $(|J_i|+1)$ number of machines. $\mathrm{ET}'_i$is a new ending time of job i, which is calculated by $\mathrm{ET}_{L(j)} + |J_i| \times \left\{\mathrm{ET}_i - \left(\mathrm{ET}_{L(j)} + \mathrm{SETUP}\right)\right\}/(|J_i|+1)$

machines. Thus, the TPT required for job i on machine j will be:

$$\mathrm{TPT} = |J_i| \times \left\{\mathrm{ET}_i - \left(\mathrm{ET}_{L(j)} + \mathrm{SETUP}\right)\right\}/(|J_i|+1)$$

Figure 2b illustrates the system status after job i is split to $(|J_i|+1)$ machines. It should be noted that TPT can have any sign (i.e., minus, zero, and plus). If TPT is less than the time for processing a workpiece of job i, it is not appropriate to process job i on machine j. Step 10 of the algorithm SPLIT tests the condition for the appropriateness of splitting job i to $(|J_i|+1)$ machines from $|J_i|$.

References

1. Balasubramanian H, Mönch L, Fowler J, Pfund M (2004) Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. Int J Prod Res 42(8):1621–1638
2. Beraldi P, Ghianib G, Griecob A, Guerrierob E (2008) Rolling-horizon and fix-and-relax heuristics for the parallel machine lot-sizing and scheduling problem with sequence-dependent set-up costs. Comput Oper Res 35:3644–3656
3. Chen J (2009) Scheduling on unrelated parallel machines with sequence- and machine-dependent setup times and due-date constraints. Int J Adv Manuf Technol 44:1204–1212
4. Coffman EG, Bruno JL (1976) Computer and job shop scheduling theory. Wiley, New York
5. Dogramaci A, Surkis J (1979) Evaluation of a heuristic for scheduling independent jobs on parallel identical processors. Manag Sci 25:1208–1216
6. Jia J, Mason SJ (2009) Semiconductor manufacturing scheduling of jobs containing multiple orders on identical parallel machines. Int J Prod Res 47(10):2565–2585
7. Kim DW, Kim KH, Jang WS, Chen FF (2002) Unrelated parallel machine scheduling with setup times using simulated annealing. Rob Comput Integr Manuf 18(3–4):223–231
8. Kim YD, Shim SO, Kim SB, Choi YC, Yoon HM (2004) Parallel machine scheduling considering a job-splitting property. Int J Prod Res 42(21):4531–4546
9. Lee YH, Pinedo M (1997) Scheduling jobs on parallel machines with sequence-dependent setup times. Eur J Oper Res 100:464–474
10. Logendran R, Subur F (2004) Unrelated parallel machine scheduling with job splitting. IIE Trans 36:359–372
11. Mason SJ, Jin S, Jampani J (2009) A moving block heuristic to minimise earliness and tardiness costs on parallel machines. Int J Prod Res 47(19):5377–5390
12. Radhakrishnan S, Ventura JA (2000) Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. Int J Prod Res 38(10):2233–2252
13. Rajgopal J, Bidanda B (1991) On scheduling parallel machines with two setup classes. Int J Prod Res 29(12):2443–2458
14. Serafini P (1996) Scheduling jobs on several machines with the job splitting property. Oper Res 44(4):617–628
15. Shim SO, Kim YD (2008) A branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property. Comput Oper Res 35(3):863–875
16. So KC (1990) Some heuristics for scheduling jobs on parallel machines with setups. Manag Sci 36(4):467–475
17. Tahar DN, Yalaoui F, Chu C, Amodeo L (2006) A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times. Int J Prod Econ 99:63–73
18. Wittrock RJ (1990) Scheduling parallel machines with major and minor setup times. Int J Flex Manuf Syst 2:329–341
19. Xing W, Zhang J (2000) Parallel machine scheduling with splitting jobs. Discret Appl Math 103:259–269
20. Yalaoui F, Chu C (2003) An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times. IIE Trans 35:183–190