European Journal of Operational Research 112 (1999) 81–97

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Match-up scheduling under a machine breakdown

M. Selim Akturk *, Elif Gorgulu

Department of Industrial Engineering, Faculty of Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey

Received 1 May 1995; accepted 1 September 1997

Abstract

When a machine breakdown forces a modified flow shop (MFS) out of the prescribed state, the proposed strategy reschedules part of the initial schedule to match up with the preschedule at some point. The objective is to create a new schedule that is consistent with the other production planning decisions like material flow, tooling and purchasing by utilizing the time critical decision making concept. We propose a new rescheduling strategy and a match-up point determination procedure through a feedback mechanism to increase both the schedule quality and stability. The proposed approach is compared with alternative reactive scheduling methods under different experimental settings. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Scheduling; Heuristics; Branch and bound

1. Introduction

Traditional scheduling procedures consider static and deterministic future conditions even though this may not be the case in actual scheduling problems. After a disruption, the preplanned schedule can become inapplicable to the new conditions. As Graves [9] stated, there is no scheduling problem but rather a rescheduling problem. Responding to such dynamic factors immediately as they occur is called real-time scheduling. Given an initial schedule and a perturbing event, the system is rescheduled to cope with these new conditions. This can also be called a time critical decision making process since the shop waits to receive the new schedule. There are primarily two distinct strategies to deal with the disruption effects on the preschedule, which are just-in-case efforts and a reactive scheduling. In the first one, the possibility of future disruptions is considered within the initial schedule by allocating some slack time to prevent an activity that may or may not occur during the given time period. In reactive scheduling, an operational solution is suggested to compensate the disruption effects when it occurs.

An on-line simulation methodology is proposed by Davis and Jones [6] to analyze several scheduling rules in a stochastic job-shop. Nof and Grant [19] analyze various simulation experiments to

* Corresponding author. Fax: 90 312 266 4126; e-mail: akturk@bilkent.edu.tr

compare three types of automatic recovery procedures, such as rerouting, splitting orders and rescheduling, when disruption events occur. Their experiments indicate that the rescheduling policy is better than the others when there is a machine breakdown. A reactive scheduling approach is developed by Smith et al. [20], which use different knowledge sources and aim to make decisions faster with less emphasis on optimality. It utilizes the 'opportunistic reasoning' idea which decides on one of their strategic schedule revision alternatives according to the current conditions of the system. For the knowledge-based systems, the most difficult operation is to decide which knowledge source has to be activated. A discussion on the knowledge-based reactive scheduling systems can be found in Blazewicz et al. [4] and Szelke and Kerr [21]. Bean et al. [3] propose a 'match-up' heuristic method for scheduling problems with disruptions. They show that assuming enough idle time is present in the original schedule and disruptions are sufficiently spaced over time, the optimal rescheduling strategy is to match-up with the preschedule at some time in the future. They search the time horizon with equal increments and a unique match-up point, and then reschedule from the disruption to the match-up time using the best of six single machine priority ordering rules. If the best single machine solution results in excessive tardiness costs, then their procedure proceeds to the multimachine lot assignment rule to redistribute lot-to-machine assignments. Leon et al. [16] develop robustness measures and robust scheduling methods to deal with machine breakdowns and processing time variability where a right-shift control policy is used in case of a disruption to minimize the expected makespan. Wu et al. [22], on the other hand, propose a different reactive scheduling approach by utilizing various local search heuristics based on genetic algorithms for a single machine problem.

The remainder of this paper is organized as follows. In Section 2, we discuss the underlying assumptions and define the problem. We present the proposed reactive hierarchical scheduling approach in Section 3. Computational analysis of the proposed approach along with a comparison of alternative reactive scheduling methods is reported in Section 4. Finally, some concluding remarks are provided in Section 5.

2. Problem statement

We propose a new approach to reschedule the preschedule in the case of a machine breakdown in a modified flow shop (MFS), which is a physical arrangement of machines in a cellular manufacturing system (CMS). There is a direct relationship between scheduling models and shop floor configurations. Job shops represent the most versatile, most flexible and the most general of all operating systems. In a job shop, work flow is not uni-directional, that is, a part can enter the job shop at any machine and leave the shop from any machine. The flow shop is a simplified job shop where all the jobs follow the same routing. For a pure flow shop, parts can enter at only the first machine, can only follow a single path through the shop, and can exit only at the last machine. Each cell within a CMS is considered an MFS, which falls between a job shop and flow shop. In an MFS, parts can enter the cell at one of several machines, can progress through the cell by a limited number of paths, and can exit the cell at one of several machines. In an MFS, parts follow a uni-directional flow, i.e. the backtracking is not allowed, but they do not have the same routing. A simple example of an MFS with three jobs and six machines is given in Fig. 1, where the numbers correspond to the jobs. A more detailed discussion on the design of MFS can be found in Akturk and Balkose [2]. Furthermore, the research on total tardiness in multimachine configurations is quite limited. To our knowledge there are few exact approaches for $F2||T$ problem as discussed by Koulamas [10] but there is no published research on the MFS problem.

We assume that the breakdown time is not known a priori, but immediately after the event occurs the down duration can be determined. There are no alternative machines in the cell; therefore an operation on a machine cannot be swapped to another machine. Furthermore, a job that is preempted due to a machine breakdown resumes its processing from the point at which the

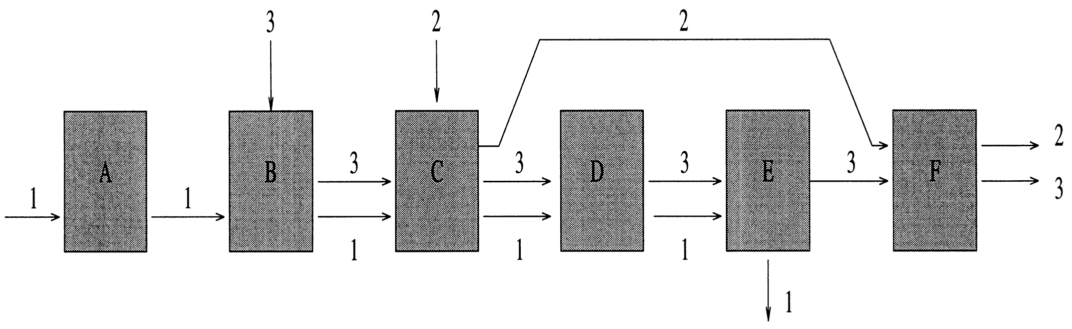

Fig. 1. An example of a modified flow shop.

interruption occurred, and otherwise preemptions are not allowed. A similar problem definition with a single disruption due to either a machine breakdown or preventive maintenance during the scheduling period is formulated as a machine scheduling problem with an availability constraint in the literature. Adiri et al. [1] consider the single machine scheduling problem with the objective of minimizing the total completion time where the machine is not available in some periods due to a machine breakdown. They assume that a job that is preempted due to a breakdown must be restarted. They studied both a stochastic case where the breakdowns occur at random times and the times for repair are random, and the deterministic case where the time for a single breakdown is known before scheduling begins. They show that the deterministic single machine scheduling problem with a single breakdown is NP-hard. Lee and Liman [14] investigate the deterministic problem of Adiri et al., except that a single disruption is now due to a preventive maintenance, and provide a simpler proof of NP-hardness. Lee and Liman [15] study the two-parallel-machines scheduling problem of minimizing the total completion time where one machine is not available for a specified period of time. They prove that the problem is NP-hard and provide a pseudo-polynomial dynamic programming algorithm to solve it.

Lee [12] studies the deterministic machine availability constraint problem with different performance measures, including makespan, total weighted completion time, maximum lateness and number of tardy jobs, for single machine and parallel machines situations. In each case, he either provides a polynomial exact algorithm to solve the problem, or proves that the problem is NP-hard. He also demonstrates that the rules which are shown to be effective for a certain scheduling problem, such as the longest processing time rule for $Pm||C_{\max}$ problem, may not work as well for the same problem with a machine availability constraint, i.e. $Pm|r-a|C_{\max}$. In this notation, $r-a$ in the second field denotes a resumable activity constraint that means if a job cannot finish before the machine breakdown then it can continue after the machine is available again. On the other hand, $nr-a$ denotes a nonresumable activity constraint such that job must be restarted after the machine breakdown. Lee [13] shows that the two-machine flow shop scheduling problem with an availability constraint to minimize the makespan, either $F2|r-a(M_1)|C_{\max}$ or $F2|r-a(M_2)|C_{\max}$, is NP-hard in the ordinary sense, although the traditional two-machine flow shop problem, $F2||C_{\max}$, can be solved optimally in polynomial time. Since it is assumed that one machine is always available, he only imposes availability constraint in one machine, i.e. M_1 or M_2, during the scheduling horizon. He also provides heuristic and pseudo-polynomial algorithms to solve this problem.

The proposed strategy in this research utilizes the match-up idea, where the state reached by the revised schedule is the same as that reached by the initial schedule, and the preschedule can be followed if no disruption occurs. Furthermore, the match-up values of each machine can be different as opposed to a single match-up point for all machines since they all have different conditions in terms of flexibilities they incur. Therefore, a feedback mechanism is developed to determine the next match-up value on each machine instead of searching the time horizon with equal increments and a unique match-up point as suggested by Bean et al. [3]. Moreover, a reactive hierarchical scheduling approach is proposed, which considers the additional constraints of the rescheduling problem. There are other production planning decisions like material flow, tooling and purchasing in the decision making hierarchy which are contingent to the results of the initial schedule. Although it might be impossible to eliminate the rescheduling problem totally for other decision levels in the hierarchy, our objective is to create a new schedule that is as consistent as possible with the consequent production planning decisions.

After a machine breakdown, a match-up point for each machine is determined and a part of the initial schedule that covers the time interval between the disruption and match-up time is rescheduled. There are two decisions that are closely related:

1. Choosing a match-up point for each machine which means determining a part of the initial schedule to be rescheduled.
2. Finding a schedule for the chosen time interval that the state of the reschedule at the match-up point is the same as that reached by the initial schedule.

We will approach to this problem heuristically because of the computational complexity of solving for both decisions simultaneously. In the proposed heuristic approach, there are two objectives:

1. The deviation from the existing schedule should be as small as possible due to the restrictions caused by other decision levels in the hierarchy, which are contingent to the results of the initial schedule. There are two types of deviations that are both undesirable in the reschedule:

- earliness: new beginning time can be earlier than the one at the initial schedule,
- tardiness: new completion time can be later than the one at the initial schedule.

In the proposed heuristic, we do not allow earliness because the new schedule should be consistent with the material flow plans, otherwise material may not be ready at the new scheduled operation starting times. However we try to minimize the tardiness since some jobs could be tardy anyway due to the down duration.

2. We utilize a time critical decision making concept because of the real-time nature of the environment. Therefore the computation should be completed in a reasonable amount of time since the shop waits to receive the new schedule. The match-up point is an important factor on the computation time, and it should be minimized to reduce the problem size, which also supports the previous objective of minimizing the deviation.

As a result, the beginning time, which is equal to the end of the machine breakdown, and match-up time of the reschedule define the time interval to be rescheduled. They also become hard constraints in addition to the non-interference and precedence constraints of an MFS, such that each machine can handle at most one job at a time and each job has a specified processing order through the machines. The beginning time constraint states that, since disruption is not foreseen, jobs processed previously cannot be rescheduled. The match-up time constraint brings the system to a state at which the initial schedule is applicable again.

In summary, we assume that a production schedule is produced off-line and in advance of execution. This preschedule then serves as the basis for the production planning decisions of other shop floor activities. We propose a reactive scheduling approach for the problem of rescheduling an MFS. The approach is based on the idea of match-up scheduling which revises the preschedule after a machine breakdown occurs in the system. The match-up schedule is constructed such that at some time after the disruption, the state of the system returned to its planned trajectory as specified by the preschedule. There are two objectives that guide the proposed heuristic, which are minimization of job tardiness and the match-up point, i.e. the length of the rescheduling period. The rescheduling attempt begins with the determination of a match-up point on each machine in order to determine the operation pool, which will be discussed in the following section. If we refer to the literature on machine scheduling problem with an availability constraint, our problem can be formally defined as solving a deterministic MFS problem with a set of resumable jobs to minimize total tardiness of all jobs for a given match-up point on each machine under the assumption that one machine is not available for a certain period of time due to a machine breakdown. Based on the computational complexity results of machine scheduling with an availability constraint, we can easily conjecture that our problem is also NP-hard.

3. Reactive hierarchical scheduling

The proposed reactive hierarchical scheduling approach (RHSA) initially selects a match-up point for each machine, then reschedules the specified job and machine pool up to the match-up point. A flow chart of the proposed approach is given in Fig. 2.

3.1. Initial pool determination

When a machine is broken down, the down time may cover the processing time of some jobs on the initial schedule. To insert this disruption duration into the initial schedule, idle times that already exist in the preschedule of down machine are utilized. Therefore jobs on the disrupted machine could be rescheduled in such a way that the down time of the machine is compensated for by the idle times in the preschedule. This, however, changes the start and finish times of the rescheduled jobs and renders the schedules for the other machines infeasible. As a result, jobs on the machines which are affected by the brokendown machine must also be rescheduled. The initial pool determination procedure starts by setting a match-up point for the brokendown machine. Next, the rescheduling pool for the brokendown machine is determined. This is followed by a match-up point determination for each of the other machines and

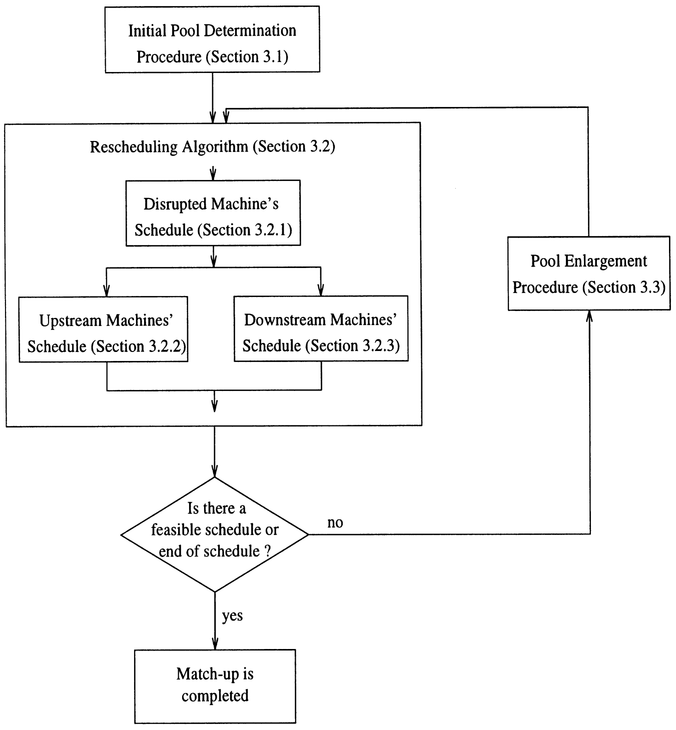
Initial Pool Determination Procedure (Section 3.1)
Rescheduling Algorithm (Section 3.2)
Disrupted Machine's Schedule (Section 3.2.1)
Upstream Machines' Schedule (Section 3.2.2)
Downstream Machines' Schedule (Section 3.2.3)
Pool Enlargement Procedure (Section 3.3)
Is there a feasible schedule or end of schedule ?
no
yes
Match-up is completed

Fig. 2. Flow-chart representation of the algorithm's framework.

augmenting the rescheduling pool based on the affected jobs on those machines.

An algorithmic description of the initial pool determination procedure can be outlined as follows, where the notation used throughout the paper is given in Table 1.

S1 Jobs to be rescheduled on k_d are collected in the set $\mathscr{J}_{k_d}$ as follows:

S1.0 Jobs following the breakdown at the down machine are sequenced in a chronological order $O_{[1]}$, $O_{[2]}$, $O_{[3]}, \ldots$ such that $O_{[1]} = \min_{\forall i} (X_{i,k_d} \mid X_{i,k_d} \geqslant t_d)$. We use subscript $[i]$ to denote the job scheduled in ith position after the breakdown.

S1.1 Find the minimum np that satisfies $t_u + \sum_{i=1}^{np} p_{[i],k_d} - X_{[np+1],k_d} \leqslant 0$. Set $T_{k_d}^{\mathrm{M}} = X_{[np+1],k_d}$ and $\mathscr{J}_{k_d} = \{O_{[1]}, O_{[2]}, \ldots, O_{[np]}\}$.

S2 To determine $\mathscr{J}_k$ for any machine other than k_d, set $T_k^{\mathrm{M}} = \max_{i \in \mathscr{J}_{k_d}} X_{i,k} + p_{i,k}$. For any i in the initial schedule that satisfies $t_d \leqslant X_{i,k} \leqslant T_k^{\mathrm{M}}$ for $k \neq k_d$, then add i to $\mathscr{J}_k$. Set $\mathscr{J} = \mathscr{J}_1 \cup \mathscr{J}_2 \cup \cdots \cup \mathscr{J}_m$ and $n = |\mathscr{J}|$.

In the first step, new jobs are added to the brokendown machine's pool until the total collected idle time exceeds the length of the breakdown duration. In the second step, all the jobs in the brokendown machine's pool are included by other machines' pools to ensure the continuity of time horizon. Furthermore, the jobs that are not processed on the brokendown machine, but scheduled in between two jobs of set $\mathscr{J}$ on any of the machines in the initial schedule, are also added to the job pool. Therefore, all operations of a job and consequently all machines processing it are included into the pool to increase job flexibility. Since the additional requirements due to the precedence and non-interference constraints are not considered in the first step, we may not find a

Table 1
Notation summary

n	number of jobs in the match-up pool
m	number of machines in the cell
$p_{i,k}$	processing time of job i on machine k
k_d	the brokendown machine
T_k^{B}	beginning time of the schedule for machine k in the initial schedule
T_k^{M}	match-up time of the schedule for machine k in the new schedule
$T_{\max}$	maximum match-up point (end point of initial schedule)
k_i^{f}	first machine that processes job i in the cell
k_i^{l}	last machine that processes job i in the cell
$X_{i,k}$	planned operation starting time of job i on machine k in the initial schedule
$\mathrm{ES}_{i,k}$	possible earliest start time of operation (i,k)
$\mathrm{LF}_{i,k}$	possible latest finish time of operation (i,k) without any tardiness
$Y_{i,k}$	starting time of job i on machine k in the new schedule
t_d	the beginning time of disruption
t_u	the ending time of disruption
$O_{[i]}$	the ith job after the disruption in the initial sequence
UNSCH	set of unscheduled jobs
$\mathscr{J}_k$	job pool for machine k and $\mathscr{J} = \bigcup_{k=1}^{m} \mathscr{J}_k$
$\mathscr{M}_i$	set of machines that process job i
$\mathscr{I}$	job set of upstream machines
$\mathscr{I}\mathscr{I}$	job set of the brokendown machine
$\mathscr{I}\mathscr{I}\mathscr{I}$	job set of downstream machines
$C_{j,\cdot}$	completion time of job j on machine group $(\cdot)$
$B_{j,\cdot}$	beginning time of job j on machine group $(\cdot)$
$\mathscr{F}_{\cdot}$	set of infeasible jobs on pair $(\cdot)$
$e_{\cdot}(j)$	amount of infeasibility for job j on pair $(\cdot)$
$e\_\max_{\mathrm{b}}$	amount of enlargement on the brokendown machine
$e\_\max_{\mathrm{d}}$	amount of enlargement on the downstream machines

feasible schedule. But the infeasible solution will still give a feedback about the amount of enlargement that is expected to be required in the next iteration.

We will now discuss the difficulty of rescheduling a set of jobs in $\mathscr{J}$. As stated earlier, a job that is preempted due to a machine breakdown resumes its processing from the point at which the interruption occurred. Therefore, the beginning time of the match-up schedule, $T^{\mathrm{B}}_{k_d}$, is equal to the sum of t_u and the remaining processing time of the preempted job, if there is any. If we keep the same sequence on the brokendown machine, k_d, then

$$\begin{aligned}
&Y_{[1],k_d} = \max\{T^{\mathrm{B}}_{k_d}\ ,\ X_{[1],k_d}\},\\
&Y_{[2],k_d} = \max\{Y_{[1],k_d} + p_{[1],k_d}\ ,\ X_{[2],k_d}\},\\
&\qquad\vdots\\
&Y_{[np],k_d} = \max\{Y_{[np-1],k_d} + p_{[np-1],k_d}\ ,\ X_{[np],k_d}\}.
\end{aligned}$$

In the initial pool determination procedure, we have collected enough idle time to cover the down duration of $(t_u - t_d)$, such that $\sum_{i=1}^{np}(X_{[i+1],k_d} - X_{[i],k_d} - p_{[i],k_d}) \geqslant t_u - t_d$. Therefore,

$$\begin{aligned}
&T^{\mathrm{B}}_{k_d} > X_{[1],k_d} \Rightarrow Y_{[1],k_d} = T^{\mathrm{B}}_{k_d},\\
&Y_{[1],k_d} + p_{[1],k_d} > X_{[2],k_d} \Rightarrow Y_{[2],k_d} = Y_{[1],k_d} + p_{[1],k_d},\\
&\qquad\vdots\\
&Y_{[np-1],k_d} + p_{[np-1],k_d} > X_{[np],k_d} \Rightarrow Y_{[np],k_d}\\
&\quad = Y_{[np-1],k_d} + p_{[np-1],k_d}.
\end{aligned}$$

Let us extend these results to a regular flow shop problem. If $Y_{[i],k_d} + p_{[i],k_d} > X_{[i],k_d+1}$ for any $i = 1,\ldots,np$, which could happen since $Y_{[i],k_d} > X_{[i],k_d}\ \forall\, i \in \mathscr{J}_{k_d}$, then the new schedule becomes infeasible due to the precedence relationships. In that case, we can still retain the same sequence and set $Y_{[i],k_d+1} = Y_{[i],k_d} + p_{[1],k_d}$ and update the starting times of the other jobs for $k = k_d + 1,\ldots,m$, accordingly. This strategy is called the right-shift procedure as discussed in Section 4, which might result in a large deviation from the completion times in the initial schedule. Another possibility would be to calculate the earliest possible start times, r_i, and the latest possible completion times, or deadlines, $\bar{d}_i$, for each job on machine k_d, such that $r_{i,k_d} = \max\{T^{\mathrm{B}}_{k_d}\ ,\ X_{i,k_d-1} + p_{i,k_d-1}\}$ and $\bar{d}_{i,k_d} = X_{i,k_d+1}$. Therefore, our aim will be to find a feasible schedule for the $1|r_i,\bar{d}_i|$ problem. Du and Leung [7] establish NP-hardness of $1|pmtn, r_i, \bar{d}_i|\sum C_i$ as stated in Lawler et al. [11]. Furthermore, we may not even find a feasible schedule for this problem without changing either r_i or $\bar{d}_i$ values, or both, that means solving the rescheduling problem for both the upstream and downstream machines. After discussing the computational complexity of the rescheduling problem, it is justifiable to develop a heuristic method as discussed below to solve the match-up scheduling problem in a reasonable computation time.

3.2. Rescheduling within the pool

After deciding on a pool size, the domain set for rescheduling problem is formed. The disrupted machine is in a nonworking condition for some time though it has previously scheduled jobs to be processed on that period. So it becomes a bottleneck because of the delayed jobs and a high utilization is required to cover this disruption duration. Therefore, the problem is decomposed into three distinct parts that are the bottleneck machine, machines in the upstream direction of it and the ones in the downstream direction, according to the sequence of machines in the MFS. First, the earliest start and latest finishing times of each job in the rescheduling pool are calculated for the first and last machine visited by that job, respectively. The earliest start and latest finishing times for each of these jobs on the remaining machines are calculated next. The scheduling attempt begins with the most critical resource, which is the brokendown machine. After this the latest finishing times are updated for the upstream machines and the earliest start times are updated for the downstream machines. If a feasible solution is found after both of these updates then the match-up is completed. Otherwise, the job pool is enlarged if the end of the preschedule is not reached.

An outline of the proposed approach is given below, in which (i, k_i^{prec}) is the immediate predecessor machine such that job i visited before coming to machine k, while (i, k_i^{suc}) as the immediate successor machine of job i on machine k, i.e.

operation (i,k). Each job has $|\mathscr{M}_i|$ operations in a predetermined sequence on the machines. In step 1, we ensure that the beginning time and the match-up time of the rescheduling pool become constraints of the rescheduling problem in addition to the precedence constraints such that $Y_{i,k} \geqslant T_k^{\mathrm{B}}$ and $Y_{i,k} + p_{i,k} \leqslant T_k^{\mathrm{M}}$ for $\forall i \in \mathscr{J}$ and $\forall k \in \mathscr{M}_i$.

S1 Calculate $\mathrm{ES}_{i,k}, \mathrm{LF}_{i,k}$ for $\forall i \in \mathscr{J}$ and $\forall k \in \mathscr{M}_i$ as follows:

S1.1 $\mathrm{ES}_{i,k_i^{\mathrm{f}}} = \max(X_{i,k_i^{\mathrm{f}}}, T_{k_i^{\mathrm{f}}}^{\mathrm{B}})$ and $\mathrm{LF}_{i,k_i^{\mathrm{l}}} = \min(X_{i,k_i^{\mathrm{l}}} + p_{i,k_i^{\mathrm{l}}}, T_{k_i^{\mathrm{l}}}^{\mathrm{M}})$ for $\forall i \in \mathscr{J}$.

S1.2

$$\mathrm{ES}_{i,k} = \begin{cases} \max(\mathrm{ES}_{i,k_i^{\mathrm{prec}}} + p_{i,k_i^{\mathrm{prec}}}, T_k^{\mathrm{B}}) & \text{if } i \in \mathscr{J}_{k_i^{\mathrm{prec}}} \\ \max(X_{i,k_i^{\mathrm{prec}}} + p_{i,k_i^{\mathrm{prec}}}, T_k^{\mathrm{B}}) & \text{if } i \notin \mathscr{J}_{k_i^{\mathrm{prec}}} \end{cases}$$

for $\forall i \in \mathscr{J}$ and $\forall k \in \mathscr{M}_i, k \neq k_i^{\mathrm{f}}$.

$$\mathrm{LF}_{i,k} = \begin{cases} \min(\mathrm{LF}_{i,k_i^{\mathrm{suc}}} - p_{i,k_i^{\mathrm{suc}}}, T_k^{\mathrm{M}}) & \text{if } i \in \mathscr{J}_{k_i^{\mathrm{suc}}} \\ \min(X_{i,k_i^{\mathrm{suc}}}, T_k^{\mathrm{M}}) & \text{if } i \notin \mathscr{J}_{k_i^{\mathrm{suc}}} \end{cases}$$

for $\forall i \in \mathscr{J}$ and $\forall k \in \mathscr{M}_i, k \neq k_i^{\mathrm{l}}$.

S2 Schedule the brokendown machine, k_d, using the algorithm discussed in Section 3.2.1.

S3 Update LF bounds for operations on the upstream machines,

$$\mathrm{LF}_{i,k} = \begin{cases} Y_{i,k_d} & \text{if } k_i^{\mathrm{suc}} = k_d, \\ \mathrm{LF}_{i,k_i^{\mathrm{suc}}} - p_{i,k_i^{\mathrm{suc}}} & \text{otherwise} \end{cases}$$

for $k = 1, \ldots, k_d - 1$ and $\forall i \in \mathscr{J}_k$. Schedule upstream machines using the algorithm discussed in Section 3.2.2.

S4 Update ES bounds for operations on the downstream machines,

$$\mathrm{ES}_{i,k} = \begin{cases} Y_{i,k_d} + p_{i,k_d} & \text{if } k_i^{\mathrm{prec}} = k_d, \\ \mathrm{ES}_{i,k_i^{\mathrm{prec}}} + p_{i,k_i^{\mathrm{prec}}} & \text{otherwise} \end{cases}$$

for $k = k_d + 1, \ldots, m$ and $\forall i \in \mathscr{J}_k$. Schedule downstream machines using the algorithm discussed in Section 3.2.3.

S5 If a feasible solution exists in both Steps 3 and 4, then STOP, match-up is completed. Else go to Step 6.

S6 If $T_{\max}$ is not reached, then enlarge the pool by adding new jobs using the proposed enlargement procedure discussed in Section 3.3 and go to Step 1. Else STOP, a complete rescheduling is performed.

3.2.1. Scheduling the brokendown machine

We propose a new dominance rule embedded in a branch and bound algorithm (BB) to schedule a set of jobs on the brokendown machine to minimize the deviation from the match-up point, which is equivalent to minimizing total tardiness with unequal ready times problem, $1|r_j|\sum T_j$. Du and Leung [8] have shown that the total tardiness problem, $1|\,|\sum T_j$, is NP-hard in the ordinary sense, whereas unequal release dates problem, $1|r_j|\sum T_j$ is strongly NP-hard because the alternatives of inserting machine idle times need to be considered as stated by Lawler et al. [11]. Although customer orders may not arrive simultaneously in real-life problems, to our knowledge, there is only one exact approach in the literature to solve the $1|r_j|\sum T_j$ problem by Chu [5]. To simplify the notation, we let $\mathrm{ES}_{j,k_d} = r_j$ and $\mathrm{LF}_{j,k_d} = d_j$ for $\forall j \in \mathscr{J}_{k_d}$, and dropped the machine index k_d for clarity.

The proposed dominance rule can be used not only to eliminate nodes of a search tree in a BB algorithm, but also to provide an initial upper bound on the minimum value of the total tardiness. The dominance rule stated in Theorem 1 is a sufficient condition for some job i to precede some other job j at the current time t^{c}; it does not imply that an optimal sequence actually exists in which i precedes j. Proposition 1 and the consequent definition of a global dominance, on the other hand, state conditions for the existence of an optimal sequence in which job i precedes job j is guaranteed. Finally, Theorem 3 extends these results and states conditions for the existence of a conditional global dominance of job i over job j at any time t. If such a condition holds, then no sequences in which job j precedes job i need to be considered. Thus, the number of sequences that have to be considered in an implicit enumeration technique can be reduced. In the following theorems, p_j, r_j and d_j are the processing time, ready time and duedate of job j on the machine k_d, respectively, and t^{c}

is the current time. Detailed proofs of these theorems are not included here due to space limitations but can be obtained from the author.

Theorem 1. *For any two jobs i and j with $p_i \leqslant p_j$, assuming both ready at the current time t^c. the following rules minimize the total tardiness.*

(i) for $p_i \neq p_j$

$$\begin{cases} \text{use the earliest due date (EDD) rule} \\ \text{if } d_i - p_j \geqslant t^c, \\ \text{use the shortest processing time (SPT) rule} \\ \text{otherwise.} \end{cases}$$

(ii) for $p_i = p_j$ use EDD.

Theorem 2. *Let $i \prec j$ denote job i precedes job j according to the dominance rule stated in Theorem 1. For any three jobs i, j and k ready at the current time, the transitivity property holds for the dominance rule, i.e., if $i \prec j$ and $j \prec k$, then $i \prec k$.*

The proposed dominance rule provides a breakpoint of precedence, b_1, which is defined as $d_i - p_j$ for each pair of jobs i and j, where $p_i \leqslant p_j$. Furthermore, the order of jobs may change in two sides of that breakpoint. Due to transitivity condition stated in Theorem 2, there is only one job that satisfies the dominance rule at time t^c. When unequal release dates are introduced, another breakpoint, b_2, appears as $\max\{r_i, r_j\}$. In the time region where only one of the two jobs is ready, the first-in first-out (FIFO) rule is applied until both jobs become available. There are at most three different possible orderings between each pair of jobs, which is illustrated in Fig. 3 for jobs i and j, where $p_i < p_j$, $d_j < d_i$ and $r_i < r_j$. A breakpoint could be inactive, such that the order of jobs is

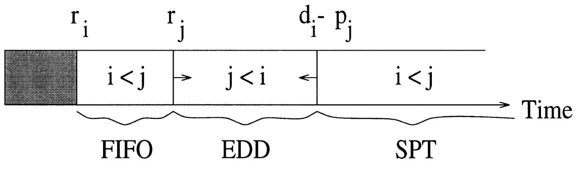

Fig. 3. An illustration of the dominance rule.

same in two sides of that breakpoint, as defined in Proposition 1.

Proposition 1. *The first breakpoint, b_1, becomes inactive in the following cases:*

(i) If $d_i \leqslant d_j$, then EDD $\equiv$ SPT.

(ii) If $\max\{r_i, r_j\} < d_i - p_j < t^c$, then always SPT.

(iii) If $d_i - p_j < \max\{r_i, r_j\}$, which means $b_1 < b_2$,

(a) If $r_i \leqslant r_j$, then always SPT.

(b) If $r_i > r_j$, then FIFO until r_i and SPT afterwards.

On the other hand, the second breakpoint, b_2, becomes inactive

(i) If $\max\{r_i, r_j\} \leqslant t^c$.

(ii) If $\max\{r_i, r_j\} \leqslant d_i - p_j$ and FIFO $\equiv$ EDD.

(iii) If $\max\{r_i, r_j\} > d_i - p_j$ and FIFO $\equiv$ SPT.

The following definition of a global dominance is derived from Proposition 1, which is very important for reducing the number of nodes in a BB algorithm since the ordering of each pair of jobs is fixed when both breakpoints are inactive.

Definition 1 (*Global dominance*)**.** If job i dominates job j for every $t \geqslant t^c$, i.e., both breakpoints are inactive, then this unconditional ordering is called a global dominance and denoted by $i \rightarrow j$.

Based on Theorem 1, which states a sufficient condition for local optimality, the following theorem is proved to establish the branching condition.

Theorem 3. *Let j be the dominant job at time t^c, if either $j \rightarrow i$ or $t^c + p_j \leqslant d_i - p_j$ or $r_i \geqslant t^c + \min\{p_j, \Delta I\}$, where ΔI gives the remaining idleness on the machine k_d, for any job i, then for the rest of the problem job j precedes job i in an optimal schedule.*

Proof. The argument is trivial when $j \rightarrow i$. The last condition $r_i \geqslant t^c + \min\{p_j, \Delta I\}$ is necessary both to have an active schedule and to satisfy the match-up constraint. Furthermore, if job j

dominates job i at time t^c and $p_i \geqslant p_j$ then it remains more desirable than job i regardless of any breakpoint. So let us look at the case where job j dominates job i at time t^c due to EDD criterion. Then the following conditions must hold $d_j < d_i$, $p_i < p_j$ and $d_i - p_j \geqslant t^c + p_j$. Let us consider a schedule S in which jobs i and j satisfy the above conditions and job j precedes job i. We construct a new schedule S' by interchanging the positions of jobs i and j. Obviously, this interchange will not affect the completion times of other jobs. Moreover, for jobs i and j, the tardiness of job j after choosing job i is denoted as $T^j_{i<j}$ and total tardiness of i and j after this decision is denoted as $T_{i<j}$ where

$$T_{i<j} = T^i_{i<j} + T^j_{i<j} = \max(0\,,\ t^c + p_i - d_i) + \max(0\,,\ t^c + p_i + p_j - d_j),$$

$$T_{j<i} = T^j_{j<i} + T^i_{j<i} = \max(0\,,\ t^c + p_j - d_j) + \max(0\,,\ t^c + p_j + p_i - d_i).$$

Furthermore $(t^c + p_i - d_i) < 0$ and $(t + p_j + p_i - d_i) < 0$, because $p_i < p_j$ and $d_i \geqslant t^c + 2p_j$. Therefore $T^i_{i<j} = T^i_{j<i} = 0$. On the other hand, $(t^c + p_i + p_j - d_j) > (t^c + p_j - d_j)$, which means $T^j_{i<j} \geqslant T^j_{j<i}$. We thus conclude that $T_{j<i} \leqslant T_{i<j}$. Hence we conclude that scheduling job i as a first job of the remaining sequence cannot improve total tardiness. □

Corollary 1. *Let j be the dominant job at time* t^c. *If there is an unscheduled job i such that job i is not globally dominated by job j,* $t^c + p_j > d_i - p_j$ *and* $r_i < t^c + \min\{p_j, \Delta I\}$, *then the current schedule may be improved by scheduling job i before job j.*

We now present a BB algorithm based upon the dominance theorems, where each node represents a partial schedule. There are primarily two alternative decisions at each node either choosing a job among the available and unscheduled ones using the dominance rule, or selecting a job that satisfies the branching condition stated in Corollary 1.

S0 [*Initialization*] Set seq $\leftarrow 0$, ub $\leftarrow \infty$, $t^c \leftarrow r_{(1)}$, and $\mathscr{S} \leftarrow \emptyset$.
S1 [*Selecting dominant*] Let j be the dominant job at t^c.
S2 [*Global dominance*] If $j \to i$ or $t^c + p_j \leqslant d_i - p_j$ or $r_i \geqslant t^c + \min\{p_j, \Delta I\}$ for every $i \in \mathscr{S}$ and $i \neq j$ then goto Step 4.
S3 [*Selecting subproblem*] Calculate lb, if $lb < ub$ then insert every unscheduled job i dominating j at $t \in (t^c, t^c + p_j]$ such that $r_i < t^c + \min\{p_j, \Delta I\}$, to the active stack, AS, then store $t_i \leftarrow \max\{t^c, r_i\}$ and seq$_i \leftarrow$ seq. Else goto Step 5.
S4 [*Upper bounding*] Set $\mathscr{S}_{[\text{seq}]} \leftarrow j$, $t^c \leftarrow \max\{t^c + p_j, r_{\min}\}$ and seq $\leftarrow$ seq + 1. If seq $< n$ then goto Step 1. Else if $T_{\mathscr{S}} <$ ub, set $\bar{\mathscr{S}} \leftarrow \mathscr{S}$ and ub $\leftarrow T_{\mathscr{S}}$.
S5 [*Branching*] If AS $\neq \emptyset$ pick job i from AS (LIFO) and set $t^c \leftarrow t_i + p_i$, $\mathscr{S}_{(\text{seq}_i)} \leftarrow i$ and seq $\leftarrow$ seq$_i + 1$ then goto Step 1.
S6 [*Report optimum*] Else report $\mathscr{S}_{\text{opt}} \leftarrow \bar{\mathscr{S}}$.

The first step finds the dominant job at t^c as a result of the proposed dominance rule. If job j satisfies the conditions of Theorem 3 for all the remaining unscheduled jobs then we increment the current time t^c, either to the maximum of the completion time of job j or to the earliest release date of the unscheduled jobs $r_{\min}$, and select the next dominant job at new t^c. Otherwise, a lower bound is calculated and if it is greater than or equal to the upper bound then this node is eliminated. In Step 3, the set of unscheduled jobs which are available and dominate job j in the interval $(t^c, t^c + p_j]$ are selected for branching at the same level, as explained in Corollary 1. These jobs are kept in the active stack, AS. The starting time and the sequence of them are also recorded. In Step 4, if tardiness of the current schedule $T_{\mathscr{S}}$ is strictly less than the upper bound, $\mathscr{S}$ is kept as the incumbent schedule. If the current schedule is completed and the AS is not empty then we pick the last inserted job i, last-in first-out (LIFO), in a depth-first enumeration scheme (DFES) and schedule it with sequence seq$_i$ in Step 5 and a new subtree is grown by returning to Step 1, otherwise the incumbent schedule $\bar{\mathscr{S}}$ is reported as optimal. We have implemented the LIFO rule in a DFES as a search strategy, because the complete schedule occurs at the bottom level of the search tree and the DFES provides an initial upper bound due to the proposed dominance rule.

3.2.2. Upstream machines' schedule

After an optimum solution is found for the disrupted machine, its scheduled jobs' beginning times specify the due dates for the upstream machines. This is a constraint-directed problem with no objectives but strict bounds. The aim is to find a feasible solution, even though it may not always exist. But the violation of due-dates caused by the schedule of the disrupted machine are allowed in order to provide a feedback information for the enlargement procedure. In the proposed approach, the scheduling attempt begins with the most restricted jobs. These jobs have no slack time and should be scheduled at their ready times since any tardiness corresponds to the violation of the precedence constraints. After scheduling a job, the earliest start and the latest finish times of other jobs' operations processed by the same machine are updated. For each of the unscheduled jobs a one-step look ahead feasibility check is performed to find candidate jobs at a given time. At the beginning of the procedure, a job-oriented scheduling is performed to place the critical jobs, and for the remaining jobs a machine-oriented scheduling is applied, though it shifts back to the job-oriented approach as any job becomes critical. An outline of the proposed approach is given below.

S0 Calculate slack values, s_i, of each job as, $s_i = \mathrm{ES}_{i,f} - \mathrm{LF}_{i,k_d-1} - \sum_{k=1}^{k_d-1} p_{i,k}$. If there exists any critical job, i.e. $s_i = 0$, schedule it through all machines. Set $k = 1$.
S1 Update $\mathrm{ES}_{i,k}$ and $\mathrm{LF}_{i,k}$ based on the scheduled jobs for $k = 1, \ldots, k_d - 1$.
S2 Set $t = \min_{i \in \mathrm{UNSCH}} \mathrm{ES}_{i,k}$. For every $i \in$ UNSCH, apply a one-step look ahead procedure such that if job i is scheduled at time t, either do any of the remaining jobs could become infeasible, or does it overlap with any of the previously scheduled jobs. If the number of feasible alternatives is more than one, then choose the most critical job, with minimum s_i value, to be scheduled at time t, call i^*. Otherwise, choose the one causing minimum tardiness on other unscheduled jobs.
S3 Update $\mathrm{ES}_{i,k}$ and $\mathrm{LF}_{i,k}$ based on the scheduled job. If any job becomes critical when all other jobs have positive slack then schedule it through all upstream machines. If there are unscheduled jobs on the current machine k, then set $t = t + p_{i^*,k}$ and go to Step 1. Else if machine k's schedule is completed, then set $k = k + 1$. If $k < k_d$ then go to Step 1.
S4 Check the feasibility of the schedule. If at least one job is scheduled after its due-date then the schedule is infeasible, so apply the Pool Enlargement Procedure, which is described in Section 3.3.

3.2.3. Downstream machines' schedule

An operation-oriented scheduling scheme, which is a hybrid approach of machine and job-oriented applications, is performed to minimize the total tardiness. The proposed backward scheduling heuristic for the downstream machines begins scheduling from the match-up points, where the due-date is considered as a hard constraint if it is equal to the match-up time. In this scheme, on each machine, a candidate job is looked for by the dominance rule stated in Theorem 1, while ready times are taken as due-dates and due-dates as ready times. The candidate job is not scheduled before the beginning time of the last job scheduled at the succeeding machine. Such a restriction avoids making a decision for a time interval on some machine before the same time interval is being scheduled for the succeeding machines. Furthermore, the latest finish time of job i on machine k either corresponds to the due-date if k is the last machine of the job or to the beginning time of the succeeding operation of the same job that is already scheduled. For the first case, the latest finish time is a soft constraint which means that a job can be tardy. However, in the second case it is a hard constraint since the latest finish time represents the precedence relations. An outline of the proposed approach can be summarized as follows:

S0 Set $t_k = T_k^{\mathrm{M}}$, for $k = k_d + 1, \ldots, m$.
S1 Set $i^* = \max_{i \in \mathrm{UNSCH}} \{\mathrm{LF}_{i,m}\}$, and $Y_{i^*,m} = t_m - p_{i^*,m}$.
S2 Set $t_m = t_m - p_{i^*,m}$ and $k = m - 1$.

S3 If $k = k_d$ then go to Step 6, otherwise recalculate $\mathrm{ES}_{j,l}$ and $\mathrm{LF}_{j,l}$ for $l = k_d + 1, \ldots, m$ based on the scheduled job.
S4 Let j^* be the dominant job at time t_k on machine k due to Theorem 1.
S5 If $\mathrm{LF}_{j^*,k} \geqslant t_k$ or k is the last machine of j^*, then
  if $t_k - p_{j^*,k} \geqslant t_{k+1}$, then set $Y_{j^*,k} = t_k - p_{j^*,k}$ and $t_k = t_k - p_{j^*,k}$.
  Otherwise, set $k = k - 1$ and go to Step 3.
else
  if $\mathrm{LF}_{j^*,k} - p_{j^*,k} \geqslant t_{k+1}$, then set $Y_{j^*,k} = \mathrm{LF}_{j^*,k} - p_{j^*,k}$ and $t_k = \mathrm{LF}_{j^*,k} - p_{j^*,k}$.
  Otherwise, set $k = k - 1$ and go to Step 3.
S6 Check the feasibility of the schedule. If at least one job is scheduled before its ready time then schedule is infeasible. If it is feasible then apply left shift procedure to minimize the tardiness, otherwise apply the Pool Enlargement Procedure.

3.3. Enlarging the pool size

In this hierarchical scheme, the overall rescheduling problem is decomposed into three parts by relaxing the precedence constraints among them, and also by utilizing a spatial decomposition due to the MFS assumption. Therefore, the resulting schedule may not be feasible since the completion time of a job's operation might be greater than the beginning time of its succeeding operation only if they are performed in two different machine groups, although it is penalized in the algorithm. Consequently, the current pool should be enlarged to increase both the amount of idle time on each machine and the number of possible sequences that a job can be scheduled. There is a trade-off since the enlargement in the pool should give the required amount of flexibility by adding a minimum number of jobs to the pool due to our second objective of time critical decision making. Because, a large match-up value can increase both complexity and nervousness of the system.

There are three possible pairs that can have infeasibilities between each other which are the upstream machines' group and the brokendown machine, the brokendown machine and the downstream machines' group, and the upstream and downstream machines' groups. The amounts of infeasibilities are checked for jobs using both of the machine groups in the pair. The maximum amount of infeasibility is accepted as the required enlargement quantity, which is the amount of idle time that should be added to the pool to get a feasible schedule for the corresponding pair. Initially, an enlargement is performed by the downstream machine group according to the other group in the pair. New jobs are added to its job pool till the amount of collected idle time is equal to the demanded quantity. Consequently, these newly contained jobs are also added to the job pools of the other two machine groups to ensure continuity. Furthermore, the earliest start and latest finish times of the infeasible jobs are updated according to the current degree of infeasibility between the machine groups. An outline of the pool enlargement procedure is given below.

S1 Check the feasibility of the current schedule:
  S1.1 Check the feasibility between upstream machines and the brokendown machine. Set $\mathscr{F}_1 = \emptyset$. For every $j \in \mathscr{I} \cap \mathscr{I}\mathscr{I}$, if $C_{j,\mathscr{I}\mathscr{I}} > B_{j,\mathscr{I}}$ then job j causes infeasibility, so calculate $e_1(j) = C_{j,\mathscr{I}\mathscr{I}} - B_{j,\mathscr{I}}$ and job j is added to the set $\mathscr{F}_1$.
  S1.2 Check the feasibility between downstream machines and the brokendown machine in a similar way. Calculate $e_2(j)$ values and determine the set $\mathscr{F}_2$.
  S1.3 Check the feasibility between upstream machines and downstream machines in a similar way. Calculate $e_3(j)$ values and determine the set $\mathscr{F}_3$.
S2 If $e_1(j) = 0\ \forall j \in \mathscr{F}_1$, $e_2(j) = 0\ \forall j \in \mathscr{F}_2$ and $e_3(j) = 0\ \forall j \in \mathscr{F}_3$ then STOP, schedule is feasible.
S3 Let $e\_\mathrm{max}_\mathrm{b}$ and $e\_\mathrm{max}_\mathrm{d}$ be $\max\{e_1(j)\}$ and $\max\{e_2(j)\ ,\ e_3(j)\}$, respectively.
S4 If $e\_\mathrm{max}_\mathrm{b} > 0$ and $e\_\mathrm{max}_\mathrm{d} = 0$, then,
  S4.1 Enlarge the brokendown machine's pool to add $e\_\mathrm{max}_\mathrm{b}$ amount of idleness,
  S4.2 Enlarge the pools of the other machines to include the new jobs added to the brokendown machine's pool, STOP.

Else if $e\_\max_b \geqslant 0$ and $e\_\max_d > 0$, then,

S4.1 Enlarge the pool of the every downstream machine to add $e\_\max_d$ amount of idleness,

S4.2 Enlarge each of the downstream machines' job pool to make sure that any job in one of these machines' is also included by other downstream machines' pools,

S4.3 Enlarge the pools of upstream machines and the brokendown machine to include the new jobs added to the downstream machines' pools. If this indirect enlargement of the brokendown machine allows at least $e\_\max_b$ amount of idleness then STOP. Otherwise, enlarge the brokendown machine's pool to complete $e\_\max_b$ amount of idleness, and include these new jobs to the pools of the other machines, STOP.

4. Experimental results

There are two major input variables that can affect the efficiency of the proposed approach, which are the initial schedule and the duration of the breakdown. Though the down duration is a measurable quantity, this is not easy for the initial schedule. That causes a need for decomposing the factors that affect the initial schedule. These are classified as the idle time percentage in the schedule, the variance of processing times, ready times and due-dates. They can have a significant impact on the amount, distribution and frequency of idle times in the planning horizon. A summary of the fixed parameters and the five experimental factors are given in Tables 2 and 3, respectively, where $U \sim [a, b]$ represents a uniformly distributed random variable in interval $[a, b]$. In each complete trial or replication of the experiment, all possible combinations of the levels of the factors are investigated. It is a $4 \cdot 2^4$ full-factorial design, which corresponds to 64 treatment combinations. The number of replications of each combination is taken as five, that gives 320 different randomly generated runs.

In an MFS the number of operations per job is not constant as discussed earlier. In the proposed experimental setting, there are 10 machines and each job might visit each machine with a 75% probability indicated as the precedence relationships, hence the average number of operations per job is 7.5. Furthermore, the first factor in the experimental design is the idle time percentage in the initial schedule. Consequently, the idle times are selected randomly from the interval $U \sim [1, 3]$ and the number of idle time occurrences on each machine, *nidle*, is determined as follows:

$$nidle = \left\lfloor \left(\frac{\text{Idle time }\%}{1 - \text{Idle time}\%} \cdot 300 \cdot 4 \right) \Big/ 2 \right\rfloor,$$

where 300, 4 and 2 correspond to the number of jobs, mean processing time and mean idle time, respectively. Moreover, an expected makespan is required to determine the range of ready times as the third experimental factor, which is found as follows:

$$\text{Expected makespan} = 1.1[(0.75 \cdot 4 \cdot 300) + (2\,nidle)].$$

Finally, the distribution for the length of the breakdown is taken as the last experimental factor whereas the beginning time of the breakdown is equal to the starting time of the fifth job on the fifth machine in the initial schedule.

An initial schedule for each run is generated by using the best of four dispatching rules which are the Shortest Processing Time (SPT), Earliest Due Date (EDD), Apparent Urgency (AU) heuristic proposed by Morton et al. [17], and the modified AU heuristic for the inserted idleness consideration [18]. The AU heuristic is a composite dispatching rule that combines the SPT rule and the minimum slack (MS) rule. Under the AU rule jobs are scheduled one at a time; that is, every time the machine becomes free, a ranking index is computed for each remaining job *i*. The job with the

Table 2
Fixed variables

Number of machines	10
Number of jobs	300
Brokendown machine	5th machine
Precedence relationships	75%
Mean processing time	4 units
Mean idle time	2 units

Table 3
Experimental Factors

Factors	# Levels	Values
Idle time percentage	4	0.05, 0.10, 0.20, 0.40
Processing time variability	2	U ∼ [3, 5], U ∼ [1, 7]
Ready time	2	U ∼ [0, makespan], U ∼ [0, (0.8 ∗ makespan)]
Due-date coefficient	2	U ∼ [2, 3], U ∼ [5, 6]
Breakdown duration	2	U ∼ [12, 16], U ∼ [20, 24]

highest ranking index is then selected to be processed next. The ranking index, $\pi_i(k,t)$ is a function of the time t and defined as

$$\pi_i(k,t) = 1/p_{ik} \exp\left(- \max\left(d_i - t - p_{ik} - \sum_{q=k+1}^{m} p_{iq}, 0 \right) \bigg/ (b\bar{p}) \right),$$

where we set the look-ahead parameter b at 2 as suggested in [17], and $\bar{p}$ is the average processing time. The AU rule is modified for the inserted idleness case and a revised priority function for any job i on machine k, $I_i(k,t)$, is calculated as follows [18]:

$$I_i(k,t) = \pi_i(k,t)(1 - b \max(\mathrm{ES}_{ik} - t, 0) \,/\, \bar{p}).$$

We first find the set of jobs that will be processed on the first machine and a sequence is found by applying these rules independently. We then define the job pool on the second machine meanwhile we update the earliest start times of the jobs that were already processed on the first machine, and proceed on. The schedule that minimizes the sum of tardiness and earliness is selected as the initial schedule.

There are two main steps in the proposed approach, which are finding a match-up point and rescheduling up to it. The match-up point and consequently the rescheduling pool are determined with individual match-up points for each machine, while different amounts of increases are allowed at each iteration, (variable ΔT_k), by using a feedback mechanism from the previous iteration. The rescheduling pool is decomposed into three partitions and a new solution methodology is proposed to each one by utilizing a hierarchical scheme. There can be alternative approaches to each step. The match-up point could be equal for all machines, determined with fixed increments on the time horizon without any feedback, (fixed ΔT), as suggested by Bean et al. [3]. The pool can be rescheduled with the best of four dispatching rules of SPT, EDD, AU and the modified AU heuristic as discussed above. Alternative approaches are created by combining the different aspects of these two steps as shown in Table 4. In addition, a static pushback strategy, also called the right-shift procedure, is included in the experimental analysis. In this strategy, when a machine breakdown occurs, the job sequences on each machine are kept same, only the starting times are shifted to the right (in a Gantt chart representation) as far as necessary to accommodate the disruption. Furthermore, in the RHSA and the right shift, earliness is not allowed as discussed earlier, while the forward dispatching rules might deviate from the initial starting times,

Table 4
Alternative approaches for comparison

Alternatives	Rescheduling method	Match-up point
RHSA	Hierarchical approach	Variable ΔT_k
Alt 1	Best of dispatching rules	Variable ΔT_k
Alt 2	Best of dispatching rules	Fixed $\Delta T = T_{\max}/10$
Alt 3	Best of dispatching rules	Fixed $\Delta T = T_{\max}/20$
Alt 4	Best of dispatching rules	Fixed $\Delta T = T_{\max}/40$
Right shift	Same job sequence	Variable

i.e. earliness is allowed. All of these alternatives, including RHSA, are coded in C language and implemented on the Sun Sparc system.

There are primarily two main objectives, which are the schedule quality and schedule stability. The schedule quality is quantified in terms of two performance measures that are the earliness and tardiness. Both terms measure the amount of change between the preschedule and the new schedule. The schedule stability is measured by the computation time and the match-up point. The results of the experimental analysis over 320 randomly generated runs are summarized in Table 5 along with the minimum and maximum of all runs to indicate the range of values. When we compare the proposed RHSA with the other match-up alternatives in terms of the total deviation that is the sum of tardiness and earliness, the RHSA performs better than others at 0.1% significance level due to a paired t-test. The match-up point is also an important measure, that a large match-up value is not desirable due to our second objective. The RHSA again outperforms the others and the amount of improvement is notable at 0.1% significance level. The match-up time is proportional to the computation time in the RHSA. The reason of our heuristic's low computation time is the low match-up value which decreases the size of the pool to be rescheduled and reduces the number of iterations. Furthermore, the breakdown machine scheduling corresponds to a relatively easy instance of $1|r_j|\sum T_j$ problem. Therefore an initial upper bound of the BB algorithm generated by the proposed dominance rule gives the optimum solution in almost all of the runs, which significantly reduces the computation time of the BB algorithm. Furthermore, the range of values can be very helpful to analyze which problem characteristics are easy or hard on each performance measure. For each factor, low and high levels are represented by 0 and 1, respectively. In general, the low idle time percentage and high breakdown duration create the most difficult instance for all alternatives, namely 00001. On the other hand, the easy instances correspond to the high idle time percentage and low breakdown duration, i.e. 11010 and 11110, as expected.

Since the objective functions, or performance measures, are not expressed in commensurable terms, a unique measure is desired that includes all of the four performances in it in order to allow an overall evaluation. The following eigenvalue normalization procedure is used to have a common unit of measure for each objective, since it is less sensitive both to the range for the actual values and to the number of data points than the (0-1)

Table 5
Comparison of the match-up alternatives

		Tard.	Earl.	Comp. (s.)	Match-up point	Overall perform.
	Minimum	0	0	0.468	41	
RHSA	Average	83.5	0	0.745	119.1	0.183
	Maximum	365	0	2.22	210	
	Minimum	0	0	0.24	41	
Alt 1	Average	297.7	572.5	1.67	187.8	0.664
	Maximum	3249	6073	13.924	861	
	Minimum	0	4	0.25	144	
Alt 2	Average	101.4	105.4	1.174	253.9	0.347
	Maximum	595	471	10.514	480	
	Minimum	0	18	0.264	100	
Alt 3	Average	86.1	199.4	1.098	193.7	0.333
	Maximum	507	744	7.942	423	
	Minimum	0	4	0.328	70	
Alt 4	Average	78.9	172.3	1.81	163.7	0.359
	Maximum	498	599	25.254	434	
	Minimum	0	0	0.37	82	
Right Shift	Average	141.9	0	0.561	143.4	0.221
	Maximum	629	0	0.77	377	

scaling function, where 0 indicates the best value and 1 indicates the worst value among the alternatives for each objective.

$$N_{ij} = \frac{A_{ij}}{\sqrt{\sum_{j\in\rho} A_{ij}^2}},$$

where A_{ij} stands for the value of ith objective in jth alternative, ρ is the set of alternatives and N_{ij} is the normalized value of the A_{ij} value. A global measure is found by using an eigenvector normalization with equal weights as shown in the last column of Table 5, which also indicates the effectiveness of the proposed approach. On the other hand, the right-shift procedure is the second best heuristic when there is a low idle time percentage in the initial schedule due to its low computation time. Whereas Alternative 1, which is a mixture of the proposed enlargement procedure and the dispatching rules, is the second best heuristic when there is a high idle time percentage. Among alternatives of 2, 3 and 4, Alternative 3 is better than the others in the overall measure, although Alternative 4 performs better than Alternative 3 in terms of the total deviation because of its smaller ΔT value. It allows determining the match up point earlier, which improves the schedule quality but increases the computation time.

Finally, a two-way analysis of variance (ANOVA) test is applied on three measures of the tardiness, computation time and match-up point to test the equality of observed responses from the different treatments of the chosen factors. A summary of ANOVA tables is given in Table 6 with the corresponding significance levels, p, for each factor. The percentage of idle time in the initial schedule, factor A, the ready time interval, factor C, job slack times, factor D, and the breakdown duration, factor E, are significant on the match-up point. The factor A directly represents the amount of idle time in the initial schedule, while the factor C indirectly affects the distribution of idle time on the Gantt Chart. Therefore, both the total amount and the frequency of idle time occurrences in the initial schedule in conjunction with the job criticality and the breakdown duration are important factors in the case of disruptions.

5. Conclusions

This paper presents a reactive scheduling approach to compensate the disruption effect caused by a machine breakdown on the initial schedule. We propose a new rescheduling strategy by using a spatial decomposition due to the CMSs along with a variable match-up time approach with a feedback mechanism for the enlargement procedure. In order to schedule the jobs on the brokendown machine, we have proved a new dominance rule embedded in a BB algorithm and the consequent theorems to provide the sufficient condition for local optimality as a function of time. Furthermore, our objective is to create a new schedule that is consistent with the other production planning decisions like material flow, tooling and purchasing by utilizing the time critical decision making concept. The proposed approach, RHSA, is compared with several match-up alternatives and the static pushback strategy under different experimental settings. The experiments indicate that RHSA results in less deviation with a smaller match-up point and computation time. Therefore we can conjecture that the proposed RHSA improved the schedule quality and stability, and also reduced the computation time. Furthermore, we

Table 6
Summary of ANOVA results

Factors	Tardiness	Comp. time	Match-up
Idle time percent.	$p = 0.000$	$p = 0.000$	$p = 0.000$
Processing time var.	$p = 0.000$	$p = 0.774$	$p = 0.245$
Ready time	$p = 0.079$	$p = 0.544$	$p = 0.003$
Due-date coef.	$p = 0.779$	$p = 0.000$	$p = 0.002$
Breakdown duration	$p = 0.000$	$p = 0.046$	$p = 0.000$

also show that the initial schedule has an important effect on the rescheduling problem. Consequently, an initial schedule should not be evaluated only by regular performance measures, but also by its inherent flexibility and robustness.

Acknowledgements

The authors would like to thank the three anonymous referees for their helpful comments and suggestions on improving the format and contents of the paper.

References

[1] I. Adiri, J. Bruno, E. Frostig, A.H.G. Rinnooy Kan, Single machine flow-time scheduling with a single breakdown, Acta Informatica 26 (7) (1989) 676–696.
[2] M.S. Akturk, H.O. Balkose, Part-machine grouping using a multi-objective cluster analysis, International Journal of Production Research 34 (8) (1996) 2299–2315.
[3] J.C. Bean, J.R. Birge, J. Mittenehal, C.E. Noon, Match-up scheduling with multiple resources, release dates and disruption, Operations Research 39 (3) (1991) 470–483.
[4] J. Blazewicz, K. Ecker, G. Schmidt, J. Weglarz, Scheduling in Computer and Manufacturing Systems, Springer, Berlin, 1993.
[5] C. Chu, A branch-and-bound algorithm to minimize total tardiness with different release dates, Naval Research Logistics Quarterly 39 (2) (1992) 265–283.
[6] W.J. Davis, A.T. Jones, A real-time production scheduler for a stochastic manufacturing environment, International Journal of Computer Integrated Manufacturing 1 (2) (1988) 101–112.
[7] J. Du, J.Y. Leung, Minimizing mean flow time with release time and deadline constraints, Technical Report, Computer Science Program, University of Texas, Dallas, 1988.
[8] J. Du, J.Y. Leung, Minimizing total tardiness on one machine is NP-hard, Mathematics of Operations Research 15 (1990) 483–495.
[9] S.C. Graves, A review of production scheduling, Operations Research 29 (4) (1981) 646–675.
[10] C. Koulamas, The total tardiness problem: review and extensions, Operations Research 42 (6) (1994) 1025–1041.
[11] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, in: S.C. Graves, A.H.G. Rinnooy Kan, P.H. Zipkin (Eds.), Handbooks in OR&MS, vol. 4, Elsevier, Amsterdam, 1993.
[12] C.-Y. Lee, Machine scheduling with an availability constraint, Journal of Global Optimization, 9 (1996) 395–416.
[13] C.-Y. Lee, Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint, in: Operations Research Letters, 20 (1997) 129–139.
[14] C.-Y. Lee, S.D. Liman, Single machine flow-time scheduling with scheduled maintenance, Acta Informatica 29 (4) (1992) 375–382.
[15] C.-Y. Lee, S.D. Liman, Capacitated two-parallel machine scheduling to minimize sum of job completion times, Discrete Applied Mathematics 41 (1993) 211–222.
[16] V.J. Leon, S.D. Wu, R.H. Storer, Robustness measures and robust scheduling for job shops, IIE Transactions 26 (5) (1994) 32–43.
[17] T.E. Morton, R.V. Rachamadugu, A.P.J. Vepsalainen, Accurate myopic heuristics for tardiness scheduling, Working Paper 36-83-84, GSIA, Carnegie Mellon University, 1984.
[18] T.E. Morton, D.W. Pentico, Heuristic Scheduling Systems: With Applications to Production Systems and Project Management, Wiley, New York, 1993.
[19] S.Y. Nof, F.H. Grant, Adaptive/predictive scheduling; review and a general framework, Production Planning and Control 2 (4) (1991) 298–312.
[20] S.F. Smith, P.S. Ow, N. Muscettola, J. Potvin, D.C. Matthy, An integrated framework for generating and revising factory schedules, Journal of Operational Research Society 41 (6) (1990) 539–552.
[21] E. Szelke, R.M. Kerr, Knowledge-based reactive scheduling, Production Planning and Control 5 (2) (1994) 124–145.
[22] S.D. Wu, R.H. Storer, P. Chang, One-machine rescheduling heuristics with efficiency and stability as criteria, Computers and Operations Research 20 (1) (1993) 1–14.