Optimization of Schedule Stability and Efficiency Under Processing Time Variability and Random Machine Breakdowns in a Job Shop Environment

Selcuk Goren,[1] Ihsan Sabuncuoglu,[2] Utku Koc[2]

[1] *Department of Industrial Engineering, Istanbul Kemerburgaz University, Istanbul, Turkey*

[2] *Department of Industrial Engineering, Bilkent University, Ankara, Turkey*

Received 15 February 2010; revised 26 September 2011; accepted 4 October 2011
DOI 10.1002/nav.20488
Published online 8 November 2011 in Wiley Online Library (wileyonlinelibrary.com).

Abstract: The ability to cope with uncertainty in dynamic scheduling environments is becoming an increasingly important issue. In such environments, any disruption in the production schedule will translate into a disturbance of the plans for several external activities as well. Hence, from a practical point of view, deviations between the planned and realized schedules are to be avoided as much as possible. The term stability refers to this concern. We propose a proactive approach to generate efficient and stable schedules for a job shop subject to processing time variability and random machine breakdowns. In our approach, efficiency is measured by the makespan, and the stability measure is the sum of the variances of the realized completion times. Because the calculation of the original measure is mathematically intractable, we develop a surrogate stability measure. The version of the problem with the surrogate stability measure is proven to be NP-hard, even without machine breakdowns; a branch-and-bound algorithm is developed for this problem variant. A tabu search algorithm is proposed to handle larger instances of the problem with machine breakdowns. The results of extensive computational experiments indicate that the proposed algorithms are quite promising in performance.
 Naval Research Logistics 59: 26–38, 2012

Keywords: proactive scheduling; job-shop scheduling; uncertainty; stability

1. INTRODUCTION

Uncertainty and disruptions have been a problem since the beginning of systemized manufacturing and remain so today [4]. In many manufacturing environments the scheduling process is generally as follows: An initial schedule is generated for a certain period of time into the future (i.e., the planning horizon). This schedule, which is sent to the shop floor at the beginning of the planning horizon, is modified and revised (i.e., the system is rescheduled) in response to unforeseen disruptions such as machine breakdowns, variations in processing times, arrivals of rush orders, order cancellations, etc. The extent of the revisions may range from minor changes to a complete rescheduling of the system. At the end of the planning horizon, we refer to the schedule that was actually executed on the shop floor as the realized schedule.

There is a vast body of literature on production scheduling, with original formulations dating back to the 1950s. However, the majority of the models used in scholarly studies are deterministic and static, in contrast to the stochastic and dynamic nature of shop floors in practice. Indeed, the failure of classical scheduling theory to adequately address uncertainties encountered in the practical environment is often cited as one of the major reasons for the gap between the theory and practice [e.g., 8, 19, 23, 28]. Especially during the last two decades, researchers have tried to bridge this gap. The suggested approaches in the relevant literature can be broadly categorized into two classes: reactive scheduling, where the objective is to develop efficient and effective methods to revise schedules in response to disruptions after they happen, and proactive scheduling, where initial schedules are generated in anticipation of disruptions.

Schedules also guide activities such as the purchase and delivery of raw materials, the (re)negotiation of due dates with customers, and the assignment of tools and operators. Disruptions in the schedule will disturb the programs of such external activities. From a practical point of view, deviations between the planned and realized schedules are to be avoided as much as possible. The term stability refers to this concern.

Most studies on schedule stability focus on rescheduling activities such that deviations between the original and new schedules are minimal. One of the earliest studies in this area is [6], where the system is rescheduled in response to machine

Correspondence to: S. Goren (selcuk.goren@kemerburgaz.edu.tr)

breakdowns to match up with the original schedule in the future. Another match-up procedure for a modified flow shop subject to machine breakdowns is developed in [3]. In another line of research, explicit stability measures are developed and minimized alongside traditional performance measures [e.g., 1, 7, 26, 35]. A detailed review of such reactive studies can be found in [23, 25, 34].

The approach in this article is proactive, i.e., it introduces stability into the initial schedule. The literature is rather sparse on this subject. The usual approach consists of a two-stage procedure: In the first stage, an efficient initial schedule is generated, where efficiency is generally measured in terms of a regular performance measure such as makespan or maximum lateness. In the second stage, additional idle times are inserted into the schedule to protect it against the negative effects of possible disruptions or variations. Examples of such studies include [17, 20–22, 32]. We refer the reader to [4, 14, 27] for detailed reviews of the studies that endeavor to cope with unforeseen disruptions and uncertainties.

In this article, we consider a job shop environment with random machine breakdowns and processing time variability. Our objective is to generate efficient and stable initial schedules. Efficiency is measured in terms of makespan. We assume that the decision-maker is willing to sacrifice from the efficiency as long as the degradation is accompanied by a significant gain in terms of schedule stability. Specifically, we optimize schedule stability subject to an upper bound T on the makespan. The threshold T denotes the maximum value of the makespan of an efficient schedule according to the decision-maker. To the best of our knowledge, this problem is not considered in the proactive scheduling literature. The formulation we use is, in a sense, dual to the models considered in the literature: instead of first optimizing efficiency and then sacrificing from it to gain stability (e.g., by inserting additional idle times), we first determine the acceptable maximum amount of efficiency degradation, and then optimize stability, respecting the constraint on efficiency. The stability measure that we consider is the sum of the variances of the job completion times in the realized schedule. This stability measure is studied in a single machine environment in [12]. Our study can be seen as an extension of [12], to the job shop environment, but we consider schedule efficiency in addition to stability. A similar bi-criteria approach is used in [11] in a single machine environment, where the trade-off between stability (measured by the total absolute difference of job completion times between the initial and the realized schedules) and efficiency (expected total tardiness/flow time of the realized schedule) is handled by means of a weighted average of separate objectives. In this article, on the other hand, stability is the main objective and efficiency is only considered to avoid an unacceptable performance of the generated schedules. In other words, the nature of the trade-off between the stability and the efficiency of the generated schedules, or the shape of the Pareto frontier, is beyond the scope of this study. We refer the reader to [31] for a comprehensive review of how the trade-off between several performance measures can be analyzed in a multi-criteria setting.

Since optimizing the actual stability measure is mathematically intractable, we propose a surrogate measure. As has been pointed out [20, 21], existing surrogate measures do not make enough use of available information about the uncertainties. Generally, the probability distributions of the uncertain parameters are available to or can be estimated by the decision-maker, but the literature merely only considers their expectations. The surrogate measure we propose makes use of the expectations and the variances of the processing times, as well as machine up-times (when the machine is in operation) and down-times (when the machine is not in operation due to breakdown). We show that the version of the problem with the surrogate stability measure is NP-hard, even without machine breakdowns, and develop a branch-and-bound algorithm for this case. We propose a tabu search to handle larger instances of the problem with machine breakdowns. The results of our computational experiments indicate that the proposed algorithms are quite effective.

The rest of the article is organized as follows: In Section 2, we define the problem and present the stability measure on a disjunctive graph model. In Section 3, we explain our branch-and-bound algorithm and the tabu search algorithm In Section 4, we discuss the computational experiments and the results. Finally, in Section 5, we present our concluding remarks and discuss several future research directions.

2. PROBLEM DEFINITION

Consider the job shop scheduling problem with n jobs and m machines. Each job consists of at most m operations. Each of the operations associated with a job must be carried out in sequence and each operation is associated with a machine. The operation associated with machine i and job j is called operation (i, j). The processing time of operation (i, j) is denoted by a random variable X_{ij} with a general cumulative distribution function $H_{ij}(t)$. Let $a_{ij} = \mathrm{E}[X_{ij}]$ and $b_{ij} = \mathrm{V}[X_{ij}]$, where E and V are the expectation and variance operators, respectively. The up-times for machine i have independent and identical general distributions $G_{i1}(t)$. Similarly, the down-times are independent and identically distributed according to a general distribution $G_{i2}(t)$. Let $U_{i1}, U_{i2} \ldots$ be the sequence of up-times and $D_{i1}, D_{i2} \ldots$ be the sequence of down-times for machine i. That is, the machine is operational from time 0 until U_{i1}, when the first breakdown occurs. The machine then takes time D_{i1} to be repaired and is again available for processing from time $U_{i1} + D_{i1}$ until time $U_{i1} + D_{i1} + U_{i2}$, and so on. Let $C_j(\phi)$ denote the time that job j completes its last operation in a schedule ϕ. We assume that all n jobs are available for processing at $t = 0$.

As pointed out in [24, Chapter 9], in stochastic scheduling, certain conventions must be applied that are not needed in deterministic scheduling. We refer to the set of sequences of operations on machines as a solution throughout this article. One needs to adapt a policy to transform a solution into a schedule, which is defined as the set of completion times of operations. Therefore, a schedule is a multivariate stochastic variable, parameterized by a policy. In this study, we assume a nonpreemptive static list policy [24]. Specifically, a sequence of operations for each machine (i.e., a solution) is generated in a proactive manner at the beginning of the planning horizon (i.e., at time $t = 0$). During the execution of the generated solution, the sequence of operations is not altered; whenever a machine becomes free, the next operation in the sequence begins processing. In other words, we assume that inserted idle times are not allowed. The main motivation behind this assumption is to maintain schedule efficiency. Moreover, in certain environments, inserted idle times cannot be tolerated due to the nature of the manufacturing process (e.g., in steel rolling, a hot slab of steel can not wait too long to be rolled because it would cool down). Similarly, preemptions are not allowed (unless they are inevitable due to a machine breakdown) because they are expensive to implement in practice. All machines are subject to random breakdowns. In a breakdown, the machine is unavailable until it is repaired. The times for repair are random and independent of each other and of the breakdown process. An operation is preempted in the case of a breakdown and is processed for its remaining processing time after the machine is available again (i.e., a preempt-resume policy is assumed).

In applying a nonpreemptive static list policy, we also differentiate between the initial and the realized schedule. In the initial schedule (which will guide production and other external activities), the completion times of operations are taken as the expected realized completion times, calculated in accordance with the nonpreemptive static list policy using $H_{ij}(t)$, $G_{i1}(t)$, and $G_{i2}(t)$. Note that the job completion times are also random variables because processing times and machine up- and down-times are random. One can use their means as point estimators. In other words, it is reasonable to base the plans for production and other external activities on expected completion times. In the realized schedule, completion times are determined as the actual processing times and machine up- and down-times unfold; thus the values of these random variables materialize as time passes along the schedule horizon.

The objective is to optimize schedule stability (i.e., to minimize deviations between the initial and the realized schedule) subject to an upper bound T on the expected makespan. The threshold T denotes the maximum value of the expected makespan a schedule can have and still be considered efficient by the decision-maker. We use the sum of the variances of the realized completion times as the stability measure (SM). We refer to the problem of minimizing SM subject to the expected makespan being lower than a threshold T in a job shop environment with random machine breakdowns and processing time variability as problem Π. As discussed later in this section, Π is mathematically intractable. Hence, a surrogate stability measure (SSM) is developed and the efficiency constraint is relaxed (the expected makespan is replaced by a lower bound) to manage the problem. This version of the problem is called Π'. The version of problem Π' without machine breakdowns is called problem Π''. It is proven in this section that Π'' (and therefore Π') is NP-hard. An exact solution procedure (a branch-and-bound algorithm) is proposed for Π''. As shown later, calculation of even the surrogate measure SSM is not possible for Π' due to random machine breakdowns. Therefore, a tabu search algorithm is proposed to handle Π' as well as large instances of Π''.

2.1. Disjunctive Graph Model

The $J_m//C_{max}$ problem can be represented with a disjunctive graph $G = (N, A, E)$, as shown in [5]. With minor changes, this representation can also be used for problems where the completion time of each job must be calculated individually, rather than just the maximum [24]. The set of nodes N contains one source node S, one node for each operation (i, j), and n sink nodes (one for each job). The source node S denotes the start of the schedule and the sink node V_j represents the completion of job j. The set of conjunctive arcs A contains the arcs that connect the nodes representing each pair of consecutive operations, say, (i, j) and (k, j) of job j (i.e., $(i, j) \rightarrow (k, j) \in A$). Each arc $(i, j) \rightarrow (k, j)$ has a length X_{ij} and represents the constraint that operation (k, j) may be started no fewer than X_{ij} time units after operation (i, j) has been started. Note that X_{ij} is a random variable. The node that represents the final operation of job j, say (h, j), has an arc of length X_{hj} incident to V_j. The source node S has n outgoing arcs, each one incident to the first operation of job j, $j = 1, \ldots, n$, with lengths equal to 0. Let N_i denote the set of nodes corresponding to the operations processed on machine i. The set of disjunctive arcs E has two arcs going in opposite directions for every pair of nodes (i, j) and (i, k) in N_i (i.e., $(i, j) \leftrightarrow (i, k) \in E$). The arc $(i, j) \rightarrow (i, k)$ has length X_{ij} and the arc $(i, k) \rightarrow (i, j)$ has length X_{ik}. Each pair of disjunctive arcs represents the fact that two operations cannot be processed on the same machine simultaneously. Orienting a disjunctive arc pair in one direction or the other corresponds to a decision as to which operation comes first. For instance, fixing arc $(i, j) \rightarrow (i, k)$ implies that operation (i, k) is processed after operation (i, j). Figure 1 presents a four-machine, three-job example.

Let $\sigma(E)$ denote a selection of disjunctive arcs from E. Any feasible schedule ϕ is equivalent to some $\sigma(E)$, having

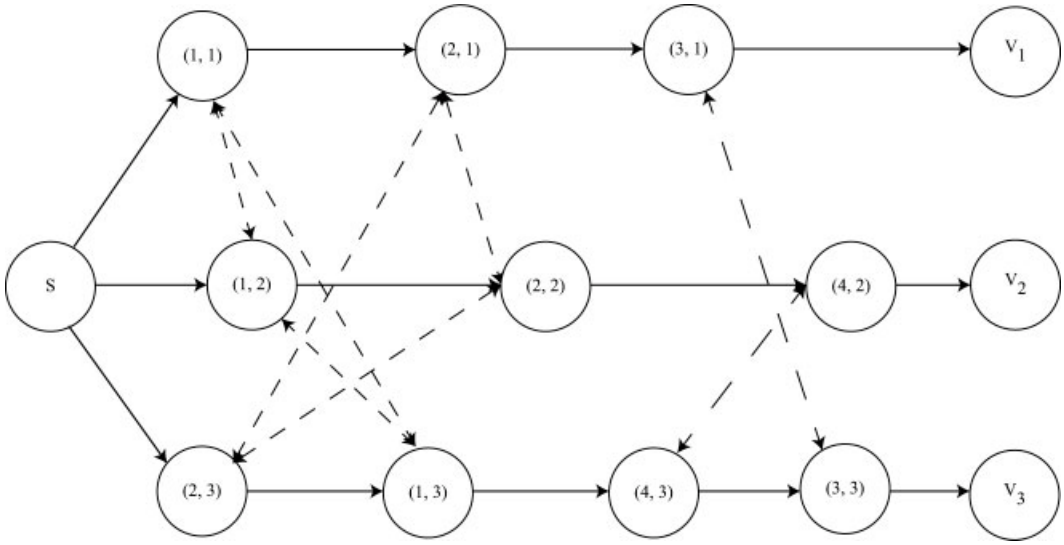

Figure 1. Disjunctive graph representation.

exactly one arc from every disjunctive pair $(i, j) \leftrightarrow (i, k)$, such that the resulting graph $G(N, A, \sigma(E))$ is acyclic. Conversely, any selection $\sigma(E)$ satisfying the above properties corresponds to a feasible schedule ϕ. Let $L_{\sigma(E)}(O, O')$ denote the distance from node O to node O' in the graph $G(N, A, \sigma(E))$. If there is no path from O to O', then $L_{\sigma(E)}(O, O')$ is not defined. Note that $L_{\sigma(E)}(O, O')$ is not associated with a single concrete path from O to O' because the arc lengths are random variables; any path from O to O' can be a critical path with some positive probability. Rather, $L_{\sigma(E)}(O, O')$ is a random variable that represents the minimum amount of time required for O' to be able to start after operation O begins its processing. The completion time $C_j(\phi)$ of job j is equal to $L_{\sigma(E)}(S, V_j)$. Therefore, $C_j(\phi)$ is also a random variable. Finally, we define a mean-critical path from O to O' as a path whose expected length $L'_{\sigma(E)}(O, O')$ is the longest among all paths from O to O'.

2.2. Stability Measure

In the literature, schedule stability is generally measured in terms of deviations in job completion times. A nondecreasing function of the differences between job completion times in the initial and realized schedules was usually employed as a stability measure. The most frequently used function is the sum of the absolute differences. The sum of the squared differences is also considered in [12]. In this article, we use the latter measure. Job completion times are random variables. It is reasonable for a decision maker to base his/her plans on their expected values. In other words, we minimize the deviations between mean (i.e., planned) completion times and realized completion times. Specifically, the stability measure we consider is

$$\mathrm{SM} = \sum_j \mathrm{E}[(\mathrm{E}[C_j(\varphi)] - C_j(\varphi))^2] = \sum_j \mathrm{V}[C_j(\varphi)]. \quad (1)$$

Note that SM is not a regular performance measure; an increase in C_j may decrease $\mathrm{V}[C_j]$. We can now define problem Π as

$$\min_{\varphi} \left\{ \mathrm{SM} = \sum_j \mathrm{V}[C_j(\varphi)] : \mathrm{E}[C_j(\varphi)] \le T, j = 1, \ldots, n \right\}. \quad (2)$$

For stochastic project networks, assuming discrete processing times, Hagstrom [13] argues that the expected project time cannot be computed efficiently. As pointed out in [18], it seems unlikely that an efficient algorithm exists for more complicated distributions of the processing times as well, because the mean cannot be computed in a time polynomial with the number of possible values of the makespan, unless P = NP [13]. The difficulty arises from the fact that even if the arc lengths are statistically independent from each other, the lengths of the paths from the source to the sinks are correlated. The same difficulty makes the optimization of SM computationally intractable.

Surrogate Stability Measure: The variance of the realized completion time $C_j(\varphi)$ of job j is estimated as the sum of the variances of the arc lengths (b_{ij}'s) that lie on the mean-critical path from S to V_j (If there are more than one mean-critical paths, the one with the maximum variance is selected).

Note that SSM is not a regular performance measure.

The approximation of SM with SSM is similar to the approximation of the variance of the project duration in PERT networks. Recall that the calculation of $E[C_j(\varphi)]$ is also mathematically intractable. The efficiency constraint is therefore modified in a similar fashion. Specifically, $E[C_j(\varphi)]$ is estimated by the length of a mean-critical path from S to V_j, $L'_{\sigma(E)}(S, V_j)$. We now formulate problem Π' as

$$\min_{\sigma(E)} \left\{ \text{SSM} : L'_{\sigma(\text{E})}(S, V_j) \leq T, j = 1, \ldots, n \right\}. \quad (3)$$

Recall that the version of problem Π' without machine breakdowns is called problem Π''. Thus, Π' is at least as hard as Π'', which is proven to be NP-hard in the following theorem:

THEOREM 1: Problem Π'' is NP-hard.

PROOF: Consider an instance of Problem Π'' where variances of operation processing times are a constant multiple of their expectations. Any mean-critical path is also the longest in terms of variances. Hence, the objective function can be computed as the sum of the "variance lengths" of mean-critical paths. This makes the stochastic instance equivalent to a deterministic $J_m//\sum C_j$ instance, in which the processing times are taken as the processing time variances in the Π'' instance. $J_m//\sum C_j$ is already known to be NP-hard [9], which completes the proof. □

3. SOLUTION METHODOLOGY

Since the problems Π' and Π'' are NP-hard, no polynomial time algorithm that solves those problems exists unless P = NP. Hence, any solution approach will either resort to an implicit enumeration procedure or to a heuristic method. In this study, we propose a branch-and-bound algorithm to exactly solve the relatively easier problem Π''. We also propose a tabu search algorithm to handle machine breakdowns (problem Π'), and large instances of problem Π''. The branch-and-bound algorithm will restrict itself to the class of so-called active schedules and the tabu search algorithm will consider so-called semiactive schedules, as well as active schedules [24]. We start with the branch-and-bound algorithm.

3.1. Branch-and-Bound Algorithm

A schedule is called active if no other schedule can be constructed to have at least one operation finishing earlier without delaying any other operation, by changing the order of processing on the machines. A schedule is called semiactive if no operation can be completed earlier without changing the processing order on any machine [24]. Note that an off-line schedule for problem Π'' cannot be identified as active or semiactive without knowing the processing times in advance. In this study, schedules are said to be active or semiactive with respect to the mean processing times of the operations.

It can be easily proven that there exists at least one active schedule that is optimal for problems with regular performance measures. Since SSM is not a regular performance measure and an optimal schedule that is active may not exist, the search space for problem Π'' should be the set of semiactive schedules, as inserted idle times not allowed. Unfortunately, our pilot computational tests indicate that a branch-and-bound algorithm that implicitly enumerates all semiactive schedules requires too much computational time to be of practical value. Therefore, the search space is restricted to the class of active schedules, which is explored using the branching scheme in [10].

A node in the proposed branch-and-bound tree consists of a partial schedule and its disjunctive graph representation. The graph in the root node includes only conjunctive arcs (precedence constraints imposed by job routings). Operations are scheduled one at a time. Nodes that are deeper in the branching tree include more precedence constraints, imposed by the disjunctive arcs whose orientations are decided. The partial schedule constituted by those precedence constraints develops into a complete feasible schedule at the leaf nodes.

Although the lower bound itself depends on the objective function and the implemented branching scheme, one property inherently holds in any branch-and-bound algorithm: the lower bound of a child node is greater than or equal to the lower bound of its parent node. Lower bounds that are used in the job shop scheduling literature generally are based on the objective function value of the partial schedule or the partial graph at a node. For problem Π'', conventional lower bounds used in the literature do not have the aforementioned property. Inserting a new operation into a parent node's partial graph to create its children may result in longer mean-critical paths, but these new paths may have lower total variance values. Hence, the objective function value of a partial schedule is not a lower bound for SSM. This is because SSM is not a regular performance measure.

To calculate a lower bound of a partial schedule with a newly oriented disjunctive arc, we first examine the clique that belongs to the machine on which the inserted operation is processed. The arcs that identify the sequence of the scheduled operations on the machine are kept and the remaining (redundant) disjunctive arcs are permanently excluded, as they cannot lie on a mean-critical path. Figure 2 gives an example with five operations.

In the figure, the clique corresponding to machine i is examined. Since the order of operations on that machine is (1-2-3-4-5), the arcs that identify this order are kept (solid arcs), and the redundant ones (dashed arcs) are excluded because they cannot lie on a mean-critical path.

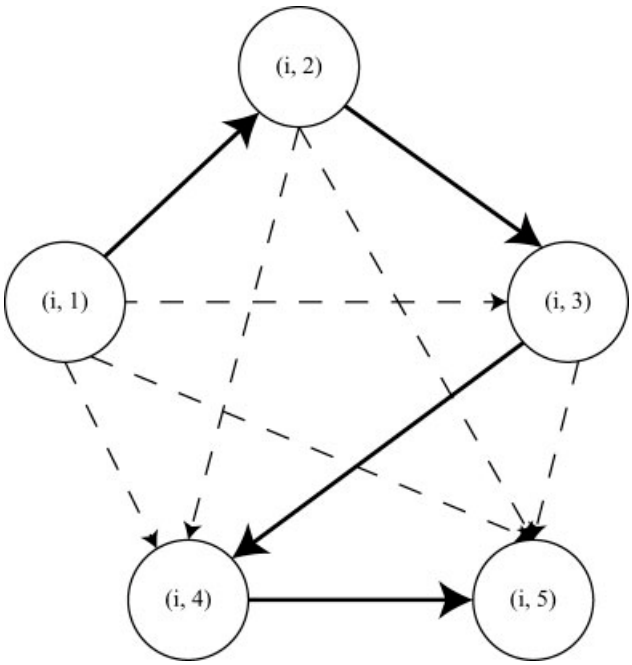

Figure 2. Examining disjunctive arcs on a machine clique.

Moreover, the partial schedule is further examined to identify additional redundant arcs. For any arc, say $(i, j) \rightarrow (k, l)$, if there exists a longer path from (i, j) to (k, l), the arc is permanently removed from the disjunctive graph as it cannot be in any mean-critical path.

After removing all redundant arcs, we temporarily insert, in both directions, all disjunctive arcs $(i, j) \leftrightarrow (i, k)$ that are not yet oriented [i.e., both $(i, j) \rightarrow (i, k)$ and $(i, k) \rightarrow (i, j)$ are inserted]. The shortest variance paths from S to V_j for all j, are identified in the augmented graph using Dijkstra's algorithm. The sum of the path variances from S to V_j for all j is a lower bound to SSM.

The performance of a branch-and-bound algorithm depends on the branching order. The search strategies that can be used when exploring the branch-and-bound tree can be categorized into two classes: depth-first search and breadth-first search. A depth-first search starts at the root node and explores the tree as far as possible along each branch before backtracking. A breadth-first search explores all the nodes on a level before advancing to the next level. In a breadth-first search, if the created children are stored in a priority queue in accordance with their lower bounds, the resulting strategy is called a best-first search. In a given amount of computational time, a depth-first search will reach a greater number of leaf nodes (faster upper bound improvement) while a best-first search will explore a wider part of the solution landscape (faster lower bound improvement). A depth-first search requires much less storage space (i.e., memory) than a best-first search. In our implementation, we use a hybrid search to combine the advantages of both strategies to improve the lower and the upper bounds as fast as possible so we can obtain better optimality gaps at the end of the allowed computational time period. Specifically, a best-first search (the node with the best lower bound value is branched first) strategy is used as long as the memory used to store unexplored nodes of the branch-and-bound tree is below a threshold value M_1. The search strategy switches to depth-first search when the memory requirement surpasses M_1 and continues until it reaches a threshold value M_2, where $M_2 < M_1$. After this point in time, the best-first search is back in use and the exploration of the nodes continues in that fashion until an optimal solution is found or the time limit is up.

3.2. Tabu Search Algorithm

The proposed branch-and-bound algorithm becomes increasingly expensive in terms of computational time as the problem size gets larger. In this section, we propose a tabu search algorithm that can be used to solve large problem instances.

We begin with a seed schedule, which is generated using the shifting bottleneck heuristic proposed in [2]. At each iteration, we generate the neighborhood of the current seed and evaluate the objective function value of the schedules in the neighborhood to select the best schedule to be the new seed (if it satisfies the efficiency constraint on the makespan and the move to generate it is not tabu). The generator creates new schedules, which in turn are evaluated again. The search continues in this fashion until the stopping criterion is met.

The neighborhood generator reverses the orientation of a disjunctive arc on a mean-critical path to obtain a neighbor. Note that the neighbors generated with this move cannot be infeasible (otherwise the reversed arc would not be a part of a mean-critical path, see [33, Lemma 2]). Moreover, the neighbors need not be active, even if the seed is an active schedule. Therefore, the search space for the proposed tabu search algorithm is not restricted within the class of active schedules, unlike the branch-and-bound algorithm. This neighborhood generation mechanism is first proposed in [33] to schedule a job shop using simulated annealing. The reversed arc is added to the tabu list to prevent immediate backtracking. If the best neighbor performs better than the current best solution so far, it is taken as the new seed, even if the move needed to generate it is tabu (aspiration criterion).

The proposed tabu search algorithm can also be used for problem Π', where there is a machine breakdown/repair process to be considered. Recall that the surrogate measure SSM estimates stability as the sum of arc variances on the mean-critical paths. In the presence of a breakdown/repair process, however, it is difficult to calculate even SSM analytically because one does not know which operations will be interrupted in advance. We use the common approach of

inflating the processing times of the operations appropriately to account for the effects of breakdowns [e.g., 20]. Specifically, we preprocess the problem instance and modify the means and variances of operation durations as follows:

$$\mu_{ij} = a_{ij} \times \left(1 + \frac{\mathrm{E}[D_i]}{\mathrm{E}[U_i]}\right), \tag{4}$$

$$\sigma_{ij}^2 = b_{ij} \times \left(\frac{a_{ij}}{\mathrm{E}[U_i]} \times \mathrm{V}[D_i]\right), \tag{5}$$

where D_i and U_i are the independent and identically distributed random variables denoting down- and up-times for machine i, respectively. The mean and the variance of the processing time of operation (i, j) is taken as μ_{ij} and σ_{ij}^2. The tabu search algorithm works as if no breakdowns occur, except that input mean and variance values are inflated as explained earlier. One could consider inflating processing times' means and variances and using the proposed branch-and-bound algorithm to handle machine breakdowns; however, the objective function values calculated using modified mean and variance values are only estimates of the actual SSM values because the breakdown process is approximated. Such an approximation would therefore demote the proposed branch-and-bound algorithm to a heuristic method, which would be unacceptable because of the excessive amount of computational effort a heuristic requires.

4. COMPUTATIONAL EXPERIMENTS

To assess the quality of the proposed branch-and-bound and tabu search algorithms, several input problems are solved. Since the objective function under study is the total variance on a mean-critical path, long arcs with small variances are likely to be included in the critical paths of an optimal solution. It is expected to take longer to solve the problem optimally if fewer such arcs exist. In other words, the difficulty of the problem instances is conjectured to be dependent on the ratio of expectations and variances of the arcs. To investigate this conjecture, a computational test bed is prepared to include three levels of the coefficient of variation (CV) for the processing times of the operations. For this reason, the processing times are taken to have Erlang distributions because of the relative easiness of altering the CV for that distribution. The three levels of the CV of the processing times considered are: low (*CV1*), medium (*CV2*), and high (*CV3*). The shape parameter k of the processing time distribution is uniformly sampled from U[100, 400], U[9, 25], and U[2, 6] for *CV1*, *CV2*, and *CV3*, respectively. Therefore, the CVs are in the ranges [0.05, 0.1], [0.2, 0.33], and [0.4, 0.7], respectively. The scale parameter θ of the processing times is selected such that all the mean processing times are in the interval [100, 800]. In other words, the scale parameter θ of the processing time distribution is uniformly sampled from U[1, 2], U[11, 32], and U[50, 133] for *CV1*, *CV2*, and *CV3*, respectively.

The effect of shop configuration is also examined. All jobs have operations on all machines. Three levels of machine routing are considered. On one extreme, all jobs are taken to visit the machines in the same order (flow shop or fixed routing). On the other extreme, all routings are randomly generated (job shop or random routing). The third level considers a semirandom routing, in which the set of machines is partitioned into two sets. All jobs must visit all machines in the first set before visiting any machine in the second set.

In this article, a problem instance is called to be of size $n \times m$, if the number of jobs is n and the number of machines is m. In our experiments, four levels of problem size are considered: 5×5, 5×10, 10×5, and 10×10.

To sum up, the computational environment consists of 36 problem classes (4 levels of size $\times 3$ levels of coefficient of variation $\times 3$ levels of machine routing). Ten instances of each problem class are generated, resulting in a test bed of 360 instances.

The threshold value T for the makespan to ensure efficiency is 10% higher than the makespan value obtained by using the shifting bottleneck heuristic on a deterministic job shop scheduling instance, where the processing times are taken as the means of the actual random processing times.

The algorithms are coded in the C++ language and run on a Linux box with eight GBs of physical memory, running Debian Lenny (5.0.7) on eight Intel Xeon E5430 processors at 2.66 GHz.

4.1. Problem Π''

The computational experiments in this section consider problem Π''. We first investigate the effect of the test problem type (i.e., coefficient of variation, routing, size of the problem instance) on the relative difficulty of the problem. Next, we assess the performances of the proposed algorithms with respect to the well-known shifting bottleneck algorithm and 12 dispatching rules, given in Table 1. We begin with the branch-and-bound algorithm.

4.1.1. Branch-and-Bound

The proposed branch-and-bound algorithm is allowed to run for a maximum of two hours of computational time (7200 CPU seconds) for each instance. The threshold values M_1 and M_2 are set to approximately 12 and 10% of the available physical memory, respectively, as we have eight CPUs each solving a different problem instance and we do not want to exhaust the memory. The optimality gaps and solution times in CPU seconds are summarized in Table 2, where levels of machine routing are shown in columns and the rows list the

Table 1. Dispatching rules.

Rule	Explanation
SVPT	Select operation for the job with the smallest varience of processing time
LEPT	Select operation for the job with the largest expected processing time
SEPT	Select operation for the job with the smallest expected processing time
CV	Select operation for the job with the highest coefficient of varience
MEWKR	Select operation for the job that has most expected work remaining
MEWKR-P	Select operation for the job that has most expected work remaining on operations subsequent to the "schedulable" operation
MEWKR/P	Select operation for the job that has greatest ratio between the expected work remaining and the processing time of the "schedulable" operation
LEWKR	Select operation for the job that has least expected work remaining
LEWKR-P	Select operation for the job that has least expected work remaining on operations subsequent to the "schedulable" operation
MOPNR	Select operation for the job that has the most number of operations remaining to be processed
LOPNR	Select operation for the job that has the least number of operations remaining to be processed
RAND	Select operation at random

levels of the CV. Each number in Table 2 is the average of 10 instances of the corresponding experimental point. The first number in each cell is the average optimality gap, which is calculated as

$$\text{Gap}\% = \frac{\text{UB} - \text{LB}}{\text{LB}} \times 100\%, \tag{6}$$

where UB is the *SSM* value of the best solution found so far, and LB is the lower bound obtained within 7200 CPU seconds. The second number is the average solution time in CPU seconds.

On examining Table 2, we observe that the impact of an increase in the number of jobs is more than the impact of an increase in the number of machines on the computational time and on the optimality gap.

For five-job problems, less computational time is needed to solve the instances with random machine routings than the instances with fixed or semi-random routings. Similar results are also reported in the literature for regular performance measures [15, 29, 30]. For 10-job problems, two hours of CPU time is not enough to arrive at optimality but it can still be said that the solution quality improves in terms of average optimality gap as the shop configuration gets closer to flow shop. This is because the procedures explained in Section 3.1 that remove redundant arcs work better in flow shops. Consequently, the lower bounds for flow shop problems are significantly tighter than those for the problems with random machine routing for 10-job problems.

We note that the optimality gaps generally increase with increasing levels of the CV. This can be better observed by comparing gaps for *CV1* with *CV3*. This observation could be explained by the intuition that for smaller values of the CV, the ranges of the arc variances are relatively narrower and therefore the lower bounds are tighter on the average.

We next compare the performance of the proposed branch-and-bound algorithm (B&B) with the shifting bottleneck algorithm (SB) and the "best dispatching rule" method (BDR). The BDR method runs all dispatching rules given in Table 1 sequentially and returns a schedule with the minimum objective function value that respects the efficiency constraint. Note that the dispatching rules may not yield a schedule with a makespan value that is less than or equal to T. If all 12 dispatching rules fail to find such a schedule, the BDR method does not generate a solution. All 360 instances are solved with the three algorithms (B&B, SB, and BDR).

Table 2. Results for branch-and-bound algorithm.

	5×5			5×10		
	Fixed	Semi	Random	Fixed	Semi	Random
CV1						
	0.00%	0.00%	0.00%	9.09%	30.32%	0.00%
	31.01	162.65	3.60	5353.90	4520.98	60.18
CV2						
	0.00%	0.00%	0.00%	6.28%	51.05%	0.00%
	58.39	104.46	1.86	4136.20	5387.18	27.30
CV3						
	0.00%	0.00%	0.00%	16.69%	27.33%	0.00%
	103.39	69.98	2.42	5907.42	3444.32	161.41
	10×5			10×10		
CV1						
	161.80%	282.04%	398.95%	82.38%	364.41%	553.58%
	7200.00	7200.00	7200.00	7200.00	7200.00	7200.00
CV2						
	182.33%	276.48%	411.18%	124.49%	437.98%	492.62%
	7200.00	7200.00	7200.00	7200.00	7200.00	7200.00
CV3						
	194.28%	342.60%	440.70%	131.91%	408.24%	617.51%
	7200.00	7200.00	7200.00	7200.00	7200.00	7200.00

Table 3. Performance comparison for branch-and-bound algorithm.

Size	Routing	CV	B&B (%)	SB (%)	BDR (%)	BDR (No. of solved)
5 × 5	Flow	CV1	0.00	10.80	5.06	6
5 × 5	Flow	CV2	0.00	18.05	15.64	9
5 × 5	Flow	CV3	0.00	12.70	13.86	8
5 × 5	Jobs	CV1	0.00	12.46	11.72	10
5 × 5	Jobs	CV2	0.00	13.33	20.52	9
5 × 5	Jobs	CV3	0.00	17.67	13.16	10
5 × 5	Semi	CV1	0.00	21.30	19.81	9
5 × 5	Semi	CV2	0.00	27.68	32.62	9
5 × 5	Semi	CV3	0.00	25.62	19.39	10
5 × 10	Flow	CV1	0.00	12.37	9.08	9
5 × 10	Flow	CV2	0.00	12.31	8.80	8
5 × 10	Flow	CV3	0.00	18.05	9.25	10
5 × 10	Jobs	CV1	0.00	11.08	9.69	10
5 × 10	Jobs	CV2	0.00	10.01	9.23	10
5 × 10	Jobs	CV3	0.00	15.00	13.28	10
5 × 10	Semi	CV1	0.00	11.54	16.77	8
5 × 10	Semi	CV2	0.00	19.19	28.76	9
5 × 10	Semi	CV3	0.00	18.16	21.56	8
10 × 5	Flow	CV1	2.82	4.41	3.64	10
10 × 5	Flow	CV2	0.82	5.96	5.84	7
10 × 5	Flow	CV3	0.59	0.59	4.70	6
10 × 5	Jobs	CV1	0.33	2.86	3.74	6
10 × 5	Jobs	CV2	0.00	5.58	15.32	7
10 × 5	Jobs	CV3	0.02	6.93	15.18	9
10 × 5	Semi	CV1	0.93	0.93	8.26	4
10 × 5	Semi	CV2	0.00	0.81	11.50	2
10 × 5	Semi	CV3	0.00	3.95	13.44	5
10 × 10	Flow	CV1	1.10	2.59	3.73	8
10 × 10	Flow	CV2	3.02	6.02	5.87	6
10 × 10	Flow	CV3	3.30	8.32	5.21	10
10 × 10	Jobs	CV1	0.89	1.18	6.38	10
10 × 10	Jobs	CV2	1.21	3.23	5.23	7
10 × 10	Jobs	CV3	0.00	0.50	5.90	8
10 × 10	Semi	CV1	0.00	2.93	15.80	3
10 × 10	Semi	CV2	0.00	4.31	16.67	1
10 × 10	Semi	CV3	0.00	3.28	16.07	3

The results are summarized in Table 3, where each number is the average of 10 instances of the corresponding problem class determined by the first three columns. The next three columns give the average percentage deviation in performance from the best solution obtained. The deviation is calculated as

$$\text{Deviation}(a)\% = \frac{\text{SSM}(a) - \min\{\text{SSM(B\&B)}, \text{SSM(SB)}, \text{SSM(BDR)}\}}{\min\{\text{SSM(B\&B)}, \text{SSM(SB)}, \text{SSM(BDR)}\}} \times 100\%. \quad (7)$$

In (7), SSM(a) is the objective function value obtained using algorithm a, where a is B&B, SB, or BDR. The last column shows the number of instances (out of 10) that BDR can generate an efficient schedule. Note that BDR fails to generate an efficient schedule in 86 instances out of 360.

Examining Table 3, we observe that B&B performs best for five-job problems. For 10-job problems, even though 7200 CPU seconds is not enough to reach optimality, B&B still outperforms SB and BDR.

4.1.2. Tabu Search

In this section, we test the performance of the proposed tabu search (TS) algorithm. After some pilot experimentation, we decided to allow a maximum of 5000 (20,000) iterations and to use a tabu list of length 10 (40) for five- (10-) job problems. We compare the performance of the proposed tabu search algorithm with the SB and BDR methods. The last column displays the number of instances (out of 10) for which the tabu search generates a semi-active schedule that is not active. The results are summarized in Table 4.

One can observe in Table 4 that the proposed tabu search algorithm performs significantly better than the SB and BDR methods on each type of test problems. The difference is more visible for 10-job problems or when the machine routing is semirandom. One can also observe that for more than 90% of the instances, the generated schedule is not active. This result illustrates the merit of including semiactive schedules in the search space.

We conclude that the proposed branch-and-bound algorithm can be used to solve small instances of problem Π'' to optimality. The proposed tabu search algorithm performs very well in general and should be employed to solve especially large instances of Π''.

4.2. Problem Π'

In this section, we consider machine breakdowns. We assess the performance of the proposed tabu search algorithm under mild and heavy breakdowns. We use Gamma distribution as a busy-time distribution with a shape parameter of 0.7, and a scale parameter arranged so that the mean busy-time is 4000 for mild breakdowns and 2000 for heavy breakdowns, respectively. We use Gamma distribution with a shape parameter of 1.4 for the down-time distribution, as recommended in [16]. The scale parameter of the down-time distribution is arranged to have a mean repair duration of 500.

The benchmark algorithms are the same as before, namely, SB and BDR. All the algorithms work as in the previous section, except that the mean and variance values of the processing times are inflated according to (4) and (5), respectively. Since the SSM values used by the algorithm now become estimates, we compare the performance of the mentioned algorithms by simulating the generated schedules to approximate total completion time variances (SM itself). Generated schedules are simulated 100 times. During the

Table 4. Performance comparison for tabu search algorithm.

Size	Routing	CV	TS (%)	SB (%)	BDR (%)	TS (No. of semiactive)
5 × 5	Flow	CV1	1.06	15.62	12.46	8
5 × 5	Flow	CV2	0.00	38.84	35.71	7
5 × 5	Flow	CV3	0.73	24.48	22.93	6
5 × 5	Jobs	CV1	0.00	16.67	15.92	8
5 × 5	Jobs	CV2	0.00	22.44	27.93	9
5 × 5	Jobs	CV3	1.18	23.41	19.04	8
5 × 5	Semi	CV1	0.00	40.23	36.63	10
5 × 5	Semi	CV2	0.00	45.63	49.80	9
5 × 5	Semi	CV3	0.73	35.34	28.72	9
5 × 10	Flow	CV1	1.62	13.74	11.12	10
5 × 10	Flow	CV2	0.33	14.09	7.36	8
5 × 10	Flow	CV3	1.42	23.21	13.74	8
5 × 10	Jobs	CV1	0.00	9.68	8.36	7
5 × 10	Jobs	CV2	0.00	10.90	10.03	6
5 × 10	Jobs	CV3	0.12	15.14	13.40	10
5 × 10	Semi	CV1	0.00	15.94	22.34	10
5 × 10	Semi	CV2	0.00	24.34	33.54	9
5 × 10	Semi	CV3	0.00	18.73	21.87	8
10 × 5	Flow	CV1	0.00	98.55	97.28	10
10 × 5	Flow	CV2	0.00	94.06	92.38	10
10 × 5	Flow	CV3	0.00	89.21	95.45	10
10 × 5	Jobs	CV1	0.00	79.73	81.46	10
10 × 5	Jobs	CV2	0.00	81.89	94.01	10
10 × 5	Jobs	CV3	0.00	85.29	99.88	10
10 × 5	Semi	CV1	0.00	97.11	111.43	10
10 × 5	Semi	CV2	0.00	103.58	135.35	10
10 × 5	Semi	CV3	0.00	108.98	131.92	10
10 × 10	Flow	CV1	0.00	54.82	55.11	10
10 × 10	Flow	CV2	0.00	65.68	62.06	10
10 × 10	Flow	CV3	0.00	67.25	63.07	10
10 × 10	Jobs	CV1	0.00	42.04	49.33	10
10 × 10	Jobs	CV2	0.00	47.79	49.00	10
10 × 10	Jobs	CV3	0.00	45.28	53.93	10
10 × 10	Semi	CV1	0.00	63.39	86.87	10
10 × 10	Semi	CV2	0.00	72.98	93.26	10
10 × 10	Semi	CV3	0.00	67.98	91.31	10

simulations, first, the processing times of operation (i, j) are sampled from an Erlang distribution with mean a_{ij} and variance b_{ij}. Then, breakdown times and repair durations are inserted into the schedule. Finally, job completion times are recorded and their variances are calculated. The sum of the completion time variances is taken as the performance measure (SM). The results are summarized in Tables 5 and 6, for mild and heavy breakdowns, respectively.

One can observe in Tables 5 and 6 that the proposed tabu search algorithm outperforms SB and BDR significantly. We also observe that SB performs generally better than BDR. The difference in the performances of the proposed tabu search algorithm and the benchmark algorithms is magnified when

- The number of jobs increases,
- The severity of the breakdowns increases, and/or
- The shop configuration is semirandom.

Note that these three conditions increase the difficulty of the problem in general. Therefore, it is expected that the performance of the benchmark algorithms (SB and BDR) will deteriorate under these conditions. Our computational experiments indicate that the performance of the proposed TS algorithm does not deteriorate as much as the benchmark algorithms. In other words, the benefit of using the proposed tabu search algorithm becomes greater as the problem gets harder, which is an encouraging result.

In this article, we assume that the decision-maker is willing to sacrifice from the efficiency as long as there is a significant gain in terms of the stability of the schedules. In Table 7, we quantify the percentage gain in terms of stability, which accompanies a 10% degradation in the makespan. We assume that the decision-maker would solve the problem using the shifting bottleneck algorithm to minimize the makespan if stability were not considered. The numbers in Table 7 are

Table 5. Performance comparison for tabu search under mild breakdowns.

Size	Routing	CV	TS (%)	SB (%)	BDR (%)
5 × 5	Flow	CV1	0.00	23.69	55.28
5 × 5	Flow	CV2	0.00	35.69	42.28
5 × 5	Flow	CV3	1.67	27.17	35.84
5 × 5	Jobs	CV1	0.02	37.83	27.41
5 × 5	Jobs	CV2	6.16	18.43	13.15
5 × 5	Jobs	CV3	5.24	29.17	19.40
5 × 5	Semi	CV1	0.00	73.56	51.43
5 × 5	Semi	CV2	0.00	59.60	71.89
5 × 5	Semi	CV3	0.08	36.04	35.76
5 × 10	Flow	CV1	3.75	16.87	62.73
5 × 10	Flow	CV2	7.98	12.35	25.24
5 × 10	Flow	CV3	4.49	22.31	24.10
5 × 10	Jobs	CV1	4.35	20.81	18.60
5 × 10	Jobs	CV2	4.49	22.14	18.49
5 × 10	Jobs	CV3	2.52	14.11	15.43
5 × 10	Semi	CV1	2.63	21.91	28.97
5 × 10	Semi	CV2	1.35	36.23	36.03
5 × 10	Semi	CV3	1.48	23.42	26.72
10 × 5	Flow	CV1	0.00	124.34	122.74
10 × 5	Flow	CV2	0.00	111.35	132.77
10 × 5	Flow	CV3	0.00	96.27	121.03
10 × 5	Jobs	CV1	0.00	135.88	113.10
10 × 5	Jobs	CV2	0.00	108.14	109.43
10 × 5	Jobs	CV3	0.00	95.44	99.35
10 × 5	Semi	CV1	0.00	160.21	154.82
10 × 5	Semi	CV2	0.00	128.75	160.94
10 × 5	Semi	CV3	0.00	122.00	185.93
10 × 10	Flow	CV1	0.00	120.52	141.11
10 × 10	Flow	CV2	0.00	71.86	80.93
10 × 10	Flow	CV3	0.00	62.21	85.05
10 × 10	Jobs	CV1	0.00	99.32	82.94
10 × 10	Jobs	CV2	0.00	72.97	46.00
10 × 10	Jobs	CV3	0.00	50.92	49.40
10 × 10	Semi	CV1	0.00	168.72	142.61
10 × 10	Semi	CV2	0.00	133.39	224.49
10 × 10	Semi	CV3	0.00	96.52	101.28

Table 6. Performance comparison for tabu search under heavy breakdowns.

Size	Routing	CV	TS (%)	SB (%)	BDR (%)
5 × 5	Flow	CV1	1.00	20.89	35.25
5 × 5	Flow	CV2	0.00	39.07	54.99
5 × 5	Flow	CV3	1.33	33.51	35.56
5 × 5	Jobs	CV1	0.00	43.90	38.55
5 × 5	Jobs	CV2	2.91	30.52	22.19
5 × 5	Jobs	CV3	2.70	22.73	16.13
5 × 5	Semi	CV1	0.00	65.77	62.07
5 × 5	Semi	CV2	0.00	65.66	67.29
5 × 5	Semi	CV3	0.80	40.63	34.84
5 × 10	Flow	CV1	1.71	15.29	37.53
5 × 10	Flow	CV2	7.07	7.82	14.62
5 × 10	Flow	CV3	2.75	13.82	16.11
5 × 10	Jobs	CV1	0.71	18.07	19.09
5 × 10	Jobs	CV2	3.75	16.73	13.59
5 × 10	Jobs	CV3	0.60	16.35	20.94
5 × 10	Semi	CV1	0.09	19.02	28.48
5 × 10	Semi	CV2	0.86	35.56	24.30
5 × 10	Semi	CV3	0.45	37.84	36.61
10 × 5	Flow	CV1	0.00	145.39	142.22
10 × 5	Flow	CV2	0.00	146.25	149.12
10 × 5	Flow	CV3	0.00	124.58	141.99
10 × 5	Jobs	CV1	0.00	133.61	127.62
10 × 5	Jobs	CV2	0.00	114.14	118.94
10 × 5	Jobs	CV3	0.00	103.94	118.28
10 × 5	Semi	CV1	0.00	172.54	176.08
10 × 5	Semi	CV2	0.00	155.06	162.81
10 × 5	Semi	CV3	0.00	137.49	169.62
10 × 10	Flow	CV1	0.00	123.60	128.24
10 × 10	Flow	CV2	0.00	89.96	104.84
10 × 10	Flow	CV3	0.00	83.80	81.57
10 × 10	Jobs	CV1	0.00	106.11	94.69
10 × 10	Jobs	CV2	0.00	77.74	69.29
10 × 10	Jobs	CV3	0.00	70.01	72.51
10 × 10	Semi	CV1	0.00	155.06	175.77
10 × 10	Semi	CV2	0.00	131.47	133.15
10 × 10	Semi	CV3	0.00	120.46	130.11

percentage improvements in stability if the proposed tabu search algorithm is employed under the condition that the makespan is within 10% of the makespan value of the shifting bottleneck schedule. For each instance, the percentage improvement is calculated as

$$\text{Improvement}\% = \frac{f(\text{SB}) - f(\text{TS})}{f(\text{SB})} \times 100\%, \qquad (8)$$

where f (SB) and f (TS) are the stability values for the schedules generated by the shifting bottleneck algorithm and the proposed tabu search algorithm, respectively. In other words, $f(.)$ is the simulated SM in the presence of breakdowns, and SSM otherwise. The three numbers in each cell correspond to no breakdowns, mild breakdowns, and heavy breakdowns, respectively.

On examining Table 7, we first observe that by sacrificing 10% from efficiency, it is possible to gain significant amounts of stability (up to 60%). The percentage improvement in stability increases significantly with increasing job number. Increasing the number of machines, however, has an effect to the contrary. The percentage improvement in stability is magnified when the job configuration is semirandom. As the severity of the breakdowns increases, the gain in stability also increases. In practice, stability is an important measure, especially for shop floors experiencing significant uncertainty. It is convenient that the improvement is more when it is needed in the face of severe breakdowns. Finally, we observe that improvement does not depend on the level of the CV of processing times in general.

5. CONCLUDING REMARKS AND FUTURE RESEARCH DIRECTIONS

In this article, we consider a job shop environment with random machine breakdowns and processing time variability. Our objective is to generate efficient and stable initial schedules. Efficiency is measured in terms of makespan. We optimize schedule stability under the condition that the makespan is within a certain percentage of its best known value. This problem is not considered in the proactive scheduling

Table 7. Percentage improvement in stability.

5 × 5			5 × 10		
Fixed (%)	Semi (%)	Random (%)	Fixed (%)	Semi (%)	Random (%)
CV1					
10.68	27.76	13.78	9.73	13.36	8.75
16.37	39.40	22.97	8.12	13.14	10.28
12.34	36.42	26.99	9.43	13.59	13.43
CV2					
26.80	29.90	17.05	11.01	19.19	9.34
25.57	33.61	9.23	0.90	19.68	12.79
26.31	35.25	17.04	−0.77	21.65	9.59
CV3					
17.06	24.01	17.21	16.39	14.97	12.60
17.73	24.23	16.94	13.59	14.75	9.30
20.89	25.87	14.63	8.32	23.52	12.25
10 × 5			10 × 10		
CV1					
49.51	49.07	44.24	35.22	38.61	29.31
53.69	60.03	55.90	52.32	61.79	48.52
58.37	62.43	56.41	54.55	60.22	50.94
CV2					
48.25	50.57	44.78	39.20	41.93	32.11
51.58	54.75	50.36	39.60	55.83	39.28
59.12	59.89	52.36	46.35	55.99	42.38
CV3					
46.79	51.89	45.94	39.93	40.19	30.98
46.68	53.10	47.80	37.48	46.53	32.94
54.94	57.21	50.27	43.22	54.29	38.50

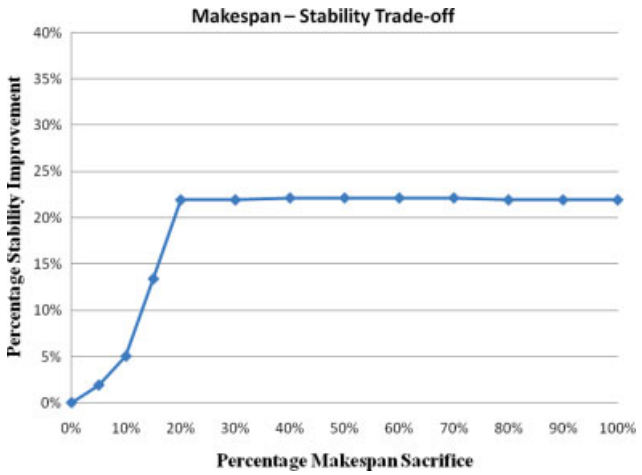

Figure 3. Stability-makespan trade-off curves. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

literature to the best of our knowledge. A surrogate stability measure is used to generate stable schedules since calculating the stability measure analytically is impractical. We show that minimizing even the surrogate stability measure is NP-hard. We develop a branch-and-bound algorithm that optimizes the surrogate stability measure in the class of active schedules. We also propose a tabu search algorithm to handle large problems with machine breakdown/repair. Our computational experiments indicate that the proposed algorithms perform quite well. The results also demonstrate that it is possible to significantly improve schedule stability with only a small compromise in efficiency.

We suggest several further research directions. First, the proposed algorithms can be specialized to flow shop environments. Our computational experiments provide evidence that machine routings may affect the solution quality. Algorithms that are customized for a flow shop environment may perform better than the general job shop algorithms developed in this article. In addition, the job population in this study is fixed and all jobs are available at time zero. Including nonzero ready times and dynamic job arrivals would likely be a useful extension.

Second, our computational results provide evidence that efficiency and stability may be conflicting objectives. Figure 3 demonstrates the stability-makespan trade-off curves for an example instances. The x-axis in Fig. 3 correspond to the maximum allowable percentage degradation in makespan and the y-axis depict percentage improvement in stability. We believe that a thorough analysis of the nature of the trade-off between these two objectives would be of practical and academic value.

Furthermore, as robustness and stability are important performance measures for practitioners, similar algorithms to generate robust job shop schedules can be developed. Moreover, a bi-criteria algorithm that can handle both measures is of practical importance. The relationship and the trade-off between robustness and stability can also be analyzed.

Finally, different stability measures and better surrogates can also be developed. Even though our computational results indicate that there is a high positive correlation between the proposed stability measure and its surrogate, employing simulation in order to estimate the stability performance of the schedules (in contrast to employing a surrogate measure) may be beneficial.

REFERENCES

[1] R.J. Abumaizar and J.A. Svetska, Rescheduling job shops under random disruptions, Int J Prod Res 35 (1997), 2065–2082.
[2] J. Adams, E. Balas, and D. Zawack, The shifting bottleneck procedure for job shop scheduling, Manage Sci 34 (1988), 391–401.
[3] M.S. Akturk and E. Gorgulu, Match-up scheduling under a machine breakdown, Eur J Oper Res 112 (1999), 81–97.
[4] H. Aytug, M.A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, Executing production schedules in the face of uncertainties: A review and some future directions, Eur J Oper Res 161 (2005), 86–110.
[5] E. Balas, Machine sequencing via disjunctive graphs: An implicit enumeration algorithm, Oper Res 17 (1969), 941–957.
[6] J.C. Bean, J.R. Birge, J. Mittenthal, and C.E. Noon, Match up scheduling with multiple resources release dates and disruptions, Oper Res 39 (1991), 471–483.
[7] L.K. Church and R. Uzsoy, Analysis of periodic and event-driven rescheduling policies in dynamic shops, Int J Comput Integrated Manuf 5 (1992), 153–163.
[8] P.I. Cowling and M. Johansson, Using real-time information for effective dynamic scheduling, Eur J Oper Res 139 (2002), 230–244.
[9] M.R. Garey, D.S. Johnson, and R. Sethi, The complexity of flowshop and jobshop scheduling, Math Oper Res 1 (1976), 117–129.
[10] B. Giffler and G.L. Thompson, Algorithms for solving production-scheduling problems, Oper Res 8 (1960), 487–503.
[11] S. Goren and I. Sabuncuoglu, Robustness and stability measures for scheduling: Single-machine environment, IIE Trans 40 (2008), 66–83.
[12] S. Goren and I. Sabuncuoglu, Optimization of schedule robustness and stability under random machine breakdowns and processing time variability, IIE Trans 42 (2010), 203–220.
[13] J.N. Hagstrom, Computational complexity of PERT problems, Networks 18 (1988), 139–147.
[14] W. Herroelen and R. Leus, Project scheduling under uncertainty: Survey and research potentials, Eur J Oper Res 165 (2005), 289–306.
[15] H.H. Holtsclaw and R. Uzsoy, Machine criticality measures and subproblem solution procedures in shifting bottleneck methods: A computational study, J Oper Res Soc 47 (1996), 666–677.
[16] A.M. Law and W.D. Kelton, Simulation modeling and analysis, McGraw-Hill, Singapore, 2000.
[17] R. Leus and W. Herroelen, The complexity of machine scheduling for stability with a single disrupted job, Oper Res Lett 33 (2005), 151–156.

[18] A. Ludwig, R.H. Möhring, and F. Stork, A computational study on bounding the makespan distribution in stochastic project networks, Annals Oper Res 49 (2001), 49–64.
[19] B.L. MacCarthy and J. Liu, Addressing the gap in scheduling research: A review of optimization and heuristic methods in production scheduling, Int J Prod Res 31 (1993), 59–79.
[20] S.V. Mehta and R. Uzsoy, Predictable scheduling of a job shop subject to breakdowns, IEEE Trans Robot Autom 14 (1998), 365–378.
[21] S.V. Mehta and R. Uzsoy, Predictable scheduling of a single machine subject to breakdowns, Int J Comput Integrated Manuf 12 (1999), 15–38.
[22] R. O'Donovan, R. Uzsoy, and K.N. McKay, Predictable scheduling of a single machine with breakdowns and sensitive jobs, Int J Prod Res 37 (1999), 4217–4233.
[23] D. Ouelhadj and S. Petrovic, A survey of dynamic scheduling in manufacturing systems, J Scheduling 12 (2009), 417–431.
[24] M. Pinedo, Scheduling: Theory, algorithms, and systems, Prentice Hall, New Jersey, 2002.
[25] A.S. Raheja and V. Subramaniam, Reactive recovery of job shop schedules—A review, Int J Adv Manuf Technol 19 (2002), 756–763.
[26] R. Rangsaritratsamee, Ferrell W. G. Jr., and M.B. Kurz, Dynamic rescheduling that simultaneously considers efficiency and stability, Comput Ind Eng 46 (2004), 1–15.
[27] I. Sabuncuoglu and S. Goren, Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research, Int J Comput Integrated Manuf 22 (2009), 138–157.
[28] C.S. Shukla and F.F. Chen, The state of the art in intelligent real-time FMS control: A comprehensive survey, J Intelligent Manuf 7 (1996), 441–455.
[29] M. Singer and M. Pinedo, A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops, IIE Trans 30 (1998), 109–118.
[30] R.H. Storer, S.D. Wu, and R. Vaccari, New search spaces for sequencing problems with application to job shop scheduling, Manage Sci 38 (1992), 1495–1509.
[31] V. T'kindt and J.-C. Billaut, Multicriteria scheduling: Theory, models and algorithms, Springer, Berlin, Germany, 2006.
[32] S. Van de Vonder, E. Demeulemeester, and W. Herroelen, Proactive heuristic procedures for robust project scheduling: An experimental analysis, Eur J Oper Res 189 (2008), 723–733.
[33] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra, Job shop scheduling by simulated annealing, Oper Res 40 (1992), 113–125.
[34] G.E. Vieira, J.W. Herrmann, and E. Lin, Rescheduling manufacturing systems: A framework of strategies, policies, and methods, J Scheduling 6 (2003), 39–62.
[35] S.D. Wu, R.H. Storer, and P. Chang, One-machine rescheduling heuristics with efficiency and stability as criteria, Comput Oper Res 20 (1993), 1–14.