

IIE Transactions

ISSN: 0740-817X (Print) 1545-8830 (Online) Journal homepage: https://www.tandfonline.com/loi/uiie20

Robustness and stability measures for scheduling: single-machine environment

Selcuk Goren & Ihsan Sabuncuoglu

To cite this article: Selcuk Goren & Ihsan Sabuncuoglu (2008) Robustness and stability measures for scheduling: single-machine environment, IIE Transactions, 40:1, 66-83, DOI: 10.1080/07408170701283198

To link to this article: https://doi.org/10.1080/07408170701283198

Published online: 06 Nov 2007.

Submit your article to this journal

Article views: 1492

View related articles

Citing articles: 69 View citing articles

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=uiie21

IIE Transactions (2008) **40**, 66–83
Copyright © "IIE"
ISSN: 0740-817X print / 1545-8830 online
DOI: 10.1080/07408170701283198

Robustness and stability measures for scheduling: single-machine environment

SELCUK GOREN and IHSAN SABUNCUOGLU*

Department of Industrial Engineering, Bilkent University, Ankara, Turkey
E-mail: {goren,sabun}@bilkent.edu.tr

Received November 2003 and accepted September 2005

This paper addresses the issue of finding robust and stable schedules with respect to random disruptions. Specifically, two surrogate measures for robustness and stability are developed. The proposed surrogate measures, which consider both busy and repair time distributions, are embedded in a tabu-search-based scheduling algorithm, which generates schedules in a single-machine environment subject to machine breakdowns. The performance of the proposed scheduling algorithm and the surrogate measures are tested under a wide range of experimental conditions. The results indicate that one of the proposed surrogate measures performs better than existing methods for the total tardiness and total flowtime criteria in a periodic scheduling environment. A comprehensive bibliography is also presented.

Keywords: Robustness, stability, proactive scheduling, tabu search

1. Introduction

Scheduling is a decision-making process concerned with the allocation of limited resources (machines, material handling equipment, operators, tools, etc.) to competing tasks (operations of jobs) over time with the goal of optimizing one or more objectives (Pinedo, 1995). The output of this process is time/machine/operation assignments. In the scheduling literature, the performance of a schedule is usually measured by using *regular measures*, which are nondecreasing in completion time. In theoretical investigations, a schedule is generated with the objective of optimizing one or more of these regular measures. The main emphasis in these studies is to generate optimal or near-optimal schedules, leaving the implementation process to practitioners.

In practice, however, due to unexpected interruptions (e.g., breakdowns, new order arrivals, order cancellations, or due date changes), schedules rapidly become infeasible and there is a need for appropriate modifications. Indeed, an inability to cope with these inevitable disruptions can be viewed as a major contributor to the gap between scheduling theory and industry practice. In industry practice, the scheduling process is generally as follows. An *initial* schedule is generated for a certain period of time to guide the shop floor activities. In the face of disruptions, this schedule is partly or completely revised to accommodate the disruptions and maintain the schedule's feasibility. The schedule that is actually executed on the shop floor is called the *realized schedule*. This schedule may differ substantially from the initial one, depending on the level of disruption and the type of action used to restore production. It is beneficial from the practitioner's viewpoint for the realized schedule to have a good performance and not to significantly deviate from the initial schedule.

In the recent literature, two new criteria have been brought to the attention of researchers for their consideration: *robustness* and *stability*. A schedule whose performance does not significantly degrade in the face of disruption is called *robust*. By definition, the performance of a robust schedule should be insensitive to disruption. In practice, when evaluating a scheduling system, the actual performance (i.e., the performance of the realized schedule) is more important than the planned or estimated performance of the initial schedule. A schedule whose realized events do not deviate from the original schedule in the face of disruption is called *stable*. It is important to point out that in the scheduling process not only are limited resources allocated to competing jobs, but also a plan is prepared for other production activities, such as setting delivery dates, release times of orders to suppliers, determining planning requirements for secondary resources such as tools, fixtures, etc. Any deviation from the planned schedule can easily disrupt these secondary plans and create *system nervousness*. In the literature, there are several studies to generate robust and stable schedules including Leon *et al*. (1994) and

*Corresponding author

0740-817X © 2008 "IIE"

Mehta and Uzsoy (1998). These will be reviewed in the next section.

In this paper, we propose two new surrogate measures for robustness and stability. These new surrogate measures are embedded into a Tabu Search (TS)-based scheduling algorithm to generate schedules in a single-machine environment with stochastic machine breakdowns. Extensive computational experiments indicate that one of the proposed surrogate measures performs better than existing methods for the total tardiness and total flowtime criteria.

The rest of the paper is organized as follows. The next section is devoted to a review of the literature. In Section 3, we consider the single-machine scheduling environment subject to random machine breakdowns, and develop a TS-based scheduling algorithm that utilizes two surrogate measures. In Section 4, we conduct extensive simulation experiments to measure the performance of the proposed algorithm and the surrogate measures. Concluding remarks and future research directions are given in Section 5.

2. Literature review

Aytug *et al.* (2005) review the existing literature on scheduling in the face of uncertainties. The authors give a four-dimensional taxonomy of the uncertainty faced in scheduling environments: *cause, context, impact* and *inclusion* of disruptions. *Cause* can be viewed as *object* (e.g., machine) and *state* (e.g., not ready). *Context* refers to the environmental situation, that is, the factors that can alter expectations for processing time, yield or any other performance measure. *Impact* refers to the result of the disruption. *Inclusion* refers to the type of action used to handle the disruptions; if disruptions are considered during schedule generation then the *inclusion* is *predictive*. On the other hand, if the disruptions are responded to after they occur, the *inclusion* is *reactive*. On reviewing the existing literature it is possible to make the following observations: the *cause* is often machine availability, and the *context* is ignored. The *inclusion* is generally predictive/reactive. The reconfiguration cost of the system after a disruption is not included. The authors suggest that the inclusion of *impact* and *context* of uncertainty as well as estimation of reconfiguration costs be the subjects of future research. They also suggest that using all the available information on the nature of a disruption, such as using distributions of machine failures, could be a potentially fruitful topic for further research.

Davenport and Beck (2000) review scheduling techniques containing uncertainty. They identify two approaches to dealing with uncertainty in a scheduling environment: *proactive* and *reactive* scheduling. *Reactive scheduling* does not consider the uncertainty in generating the initial schedules, and this schedule is revised when an unexpected event occurs. On the other hand, *robust (proactive) scheduling* takes future disruptions into consideration during the generation of the initial schedule. In conjunction with these approaches, the following scheduling techniques, each of which models the uncertainty differently, can be used: *stochastic scheduling, contingent scheduling* and *off-line/on-line approaches.*

Herroelen and Leus (2005) review project scheduling under uncertainty. Their treatment is very close to that of Davenport and Beck (2000) but it is in the domain of project scheduling rather than machine scheduling. The authors also include *fuzzy scheduling* and *sensitivity* analysis among the techniques that are used to cope with uncertainty. *Fuzzy scheduling* is based on the claim that probability distributions cannot be estimated correctly most of the time and therefore it models the imprecision of input data by using fuzzy numbers rather than probability distributions. *Sensitivity analysis* is a post-solution analysis that tries to answer several "what-if" questions on the optimality of the generated schedule in the face of changes in input parameters.

In the rest of this section, we present our review of the robust scheduling literature. Table 1 summarizes the details of these studies using the classification framework proposed in Sabuncuoglu and Goren (2005).

Wu *et al.* (1993) consider a single-machine environment in which the machine is rescheduled after every failure and they place an emphasis on stability along with makespan. The authors obtain a set of non-dominated schedules. Their computational experiments indicate that it is possible to improve the stability substantially with slight increases in makespan.

Leon *et al.* (1994) generate an initial schedule for a job shop that minimizes the weighted average of the expected makespan and the performance degradation under random machine breakdowns. They use the average system slack as a surrogate measure to estimate the expected performance degradation. Their computational experiments indicate that the average slack surrogate measure performs well under high processing time variability.

Daniels and Kouvelis (1995) generate a robust job sequence for a single machine under processing time variability such that the degradation of the performance measure under the worst possible scenario is minimized (i.e., minimize maximum regret). The computational experiments with the flowtime measure indicate that the sequences found by their algorithms are more robust than priority dispatching rules.

The work by Mehta and Uzsoy (1998, 1999) is on stability. Specifically, they insert additional idle times when generating schedules with their OSMH heuristic. Their computational results indicate that inserting additional idleness achieves stability without significantly degrading the maximum lateness.

O'Donovan *et al.* (1999) also work on generating stable schedules. Unlike the study of Mehta and Uzsoy (1998), they use the total tardiness as the performance measure. They also test rescheduling policies (right shift, some ATC derivatives, etc.) in response to machine breakdowns.

Table 1. Classification of relevant studies

	Environment			*Schedule generation*				
							How to	
Author	*Shop floor*	*Static/ dynamic*	*Stochastic/ deterministic*	*Method*	*Objective*	*When to*	*Scheme*	*Respond*
Wu *et al.* (1993)	Single machine	Static	Stochastic (machine breakdown)	GA Pairwise swapping methods	Minimize deviation between start times or between sequences (stability) Minimize makespan	Continuous after every machine breakdown	Off line	Reschedule (same method)
Leon *et al.* (1994)	Job shop	Static	Stochastic (machine breakdown, processing time variability)	GA	Minimize expected makespan and expected deviation from original makespan using surrogate measures (robustness)	Periodic	Off line	Right shift
Daniels *et al.* (1995)	Single machine	Static	Stochastic (processing time variability)	B&B, heuristics	Minimize absolute worst-case total flow time difference (robustness)	Periodic	Off line	Do nothing (left or right shift)
Mehta and Uzsoy (1998, 1999)	Job shop Single machine	Static with non-zero ready times	Stochastic (machine breakdown)	OSMH/LP	Minimize deviations between completion times while keeping L_{MAX} low using surrogate measures (stability)	Periodic	Off line	Right shift
O'Donovan *et al.* (1999)	Single machine	Static with non-zero ready times	Stochastic (machine breakdown, processing time variability)	OSMH and ATC derivatives in combination	Minimize deviations between completion times while keeping total tardiness low (stability)	Continuous after every machine breakdowns	Off line	Right shift ATC derivatives
Wu *et al.* (1999)	Job shop	Static	Stochastic (processing time variability)	B&B, ATC	Minimize expected weighted total tardiness using surrogate measures (robustness)	Periodic	Quasi on-line	Right shift

GA = genetic algorithm, B&B = branch-and-band algorithm, LP = linear programming approach.

Finally, Wu *et al.* (1999) consider the job shop scheduling problem with processing time variability. Their approach (the Process First Schedule Later approach) combines the global viewpoint of off-line methods and adaptability of on-line methods. Their computational results indicate that the proposed method is superior to the traditional off-line and on-line methods for the robustness measure (i.e., expected weighted total tardiness).

Sotskov *et al*. (1997) discuss another viewpoint for stability. They handle the uncertainty in a job shop environment by an *a posteriori* analysis of the existing optimal schedule with the objective of determining the maximum variation in the processing times of the operations that allows the optimal schedule at hand to still remain optimal. Such a maximum variation is called "the stability radius" of the schedule. This notion of stability, obtained by sensitivity analysis, can be considered as a measure of "solution robustness" in the terms discussed by Herroelen and Leus (2005). Although this type of post-optimality analysis may provide valuable insights about the impacts of the uncertainty, it does have associated drawbacks. If the "stability radius" of the optimal schedule is large enough to accommodate all possible changes in the processing times then the optimal schedule at hand can safely be used, but if it is not that large, the question of what course of action to take still remains to be answered. Hence, in this paper, we are after a proactive viewpoint and incorporate uncertainty into the scheduling processes. We concentrate on optimizing the "quality robustness" rather than the "solution robustness".

From this literature review, we identify the following research points for further investigation.

1. The probability distributions of both the busy time and repair duration are generally available to practitioners. However, existing surrogate measures do not make use of this information. As stated in Mehta and Uzsoy (1998), one can develop better surrogate measures for stability and robustness using this information.
2. In the majority of existing studies, makespan is used in robustness studies and maximum lateness is used in stability studies as the primary performance measure. As also pointed out by Leon *et al.* (1994), one can extend the existing studies for other performance measures.
3. Whether practitioners should consider robustness, stability, or both is an open research question. What is the trade-off between these performance metrics? Along the same lines, is it possible to develop a bicriterion approach that considers both the stability and robustness simultaneously?
4. Can a minimax regret type of robustness (such as the one used by Daniels and Kouvelis (1995)) be applied to disruption types other than processing time variability?

In this paper, we focus on developing two new surrogate measures for robustness and stability (i.e., the first research question) where the two scheduling criteria, total flowtime and total tardiness, are discussed separately. The details of the proposed surrogate measures can be found in the next section. Our computational experiments indicate that the proposed method performs better than the average slack method, which is commonly used in the literature as a surrogate measure. Our study also provides a partial answer to the third research question as explained later.

In our treatment, we assume that some information about the nature of the uncertainty (machine breakdowns in our case) is available in the form of probability density functions. It may be the case that the decision maker has inadequate knowledge. In such a case, a "robustness approach" that hedges against the worst contingency that may arise can be used. The uncertain data is generally modeled by interval estimations and no knowledge of probability densities is assumed. We refer the reader to Kouvelis and Yu (1997), who apply this approach to various problems such as linear programming, assignment problem, shortest path problem, etc. as well as scheduling. An example of such an approach in a scheduling context is the study of Daniels and Kouvelis (1995).

3. Methodology

3.1. *Problem formulation*

In this paper, we focus on the problem of periodic scheduling of a single machine subject to random machine breakdowns. Consider a set of n jobs to be processed on the machine. The release times (r_i), due dates (d_i) and processing times (p_i) of the jobs are deterministic and known *a priori*. Release times are not necessarily all zero. We also assume that busy times and repair times of the machine are independent and identically distributed continuous random variables and their probability density functions ($h(.)$ and $g(.)$, respectively) are known in advance. We focus on generating robust or stable initial schedules and assume the right-shift rescheduling scheme, i.e., when the machine fails the jobs are shifted to the right on the Gannt chart a sufficient amount to just accommodate the repair duration. The work already performed on the affected job is not lost and processing resumes from where it left off. Let $c_i(s)$ denote the completion time of job i in the initial schedule s and $c_i^{\mathrm{r}}(s)$ denote the corresponding realized completion time. We consider the scheduling criteria of makespan, total flow time and total tardiness.

From the viewpoint of robustness, what really matters is the realized performance of the schedule rather than the expected or planned performance (i.e., performance of the initial schedule). Since the $c_i^{\mathrm{r}}(s)$ are random variables, so is the performance of the realized schedule. Although there are several stochastic dominance notions that are used in stochastic optimization, we follow the common practice in the robust scheduling literature and try to optimize in expectation. That is, the robustness measure is the

Table 2. Robustness measures.

Performance measure: $f(s)$	*Robustness measure: $E[f^r(s)]$*
Makespan: $\max_i c_i(s)$	$E[\max_i c_i^r(s)]$
Total flow time: $\sum_{i=1}^{n}(c_i(s) - r_i)$	$E[\sum_{i=1}^{n}(c_i^r(s) - r_i)]$
Total tardiness: $\sum_{i=1}^{n}\max(c_i(s) - d_i, 0)$	$E[\sum_{i=1}^{n}\max(c_i^r(s) - d_i, 0)]$

expected realized performance of the system according to the scheduling criterion in use. Table 2 shows the robustness measures.

As for stability, a realized schedule should deviate only minimally from the initial (or planned) schedule. We use the expected sum of absolute deviations in job completion times (i.e., $E[\sum_{i=1}^{n}|c_i(s) - c_i^r(s)|]$) as our stability measure. Let S be the set of all possible permutations of the n jobs. Then, when generating robust schedules we try to find a schedule s^* such that:

$$s^* \in \arg\min_{s\in S} E[f^r(s)],$$

where the definition of $f^r(s)$ depends on the performance measure under consideration as given in Table 2.

Similarly we try to find a schedule s^{**} such that

$$s^{**} \in \arg\min_{s\in S} E\left[\sum_{i=1}^{n}\left|c_i(s) - c_i^r(s)\right|\right],$$

when generating stable schedules.

Even though the analytical calculation of these robustness and stability measures is extremely hard for general busy time and repair time distributions (we refer the reader to Pinedo (1995) for some analytical results in the case of an exponential distribution), one can predict or estimate these values by using simulation and/or employing Surrogate Measures (SMs). In the simulation case, one can employ iterative-simulation mechanisms (e.g., Kutanoglu and Sabuncuoglu (2001)). Since simulation experiments require substantial computational times, most of the robustness and stability studies in the literature focus on the development of SMs.

The average slack method is usually used in SM applications (Leon *et al.*, 1994) to generate robust schedules. The expected realized performance of a schedule S is calculated by $E[f^r(s)] = f(s) + E[\delta(s)]$ where $f(s)$ is the initial or planned performance measure of s and $E[\delta(s)]$ is the degradation in the performance measure due to random disruptions. This is estimated by using the average system slack SM. For each operation the difference between the earliest and latest start times is calculated as the *slack* of that operation. The average system slack value is calculated by averaging the slack values of all operations. Experimental studies performed by Leon *et al.* (1994) indicate that there is a high correlation between the robustness and the average slack value of schedules. The performance measure in Leon *et al.* (1994) is the makespan but we consider total tardiness and total flowtime measures in addition to makespan.

The average slack method is also used to generate stable schedules as well as robustness. In the stability case, the schedule with the largest average slack value is selected. Mehta and Uzsoy (1998, 1999) show that there is a high correlation between the stability and the average slack value of schedules.

In this paper, we propose two new SMs for a single-machine system with random machine breakdowns. In addition, we develop a scheduling algorithm that uses tabu search methodology to efficiently evaluate sequences in a large search space.

3.2. *Structure of the proposed algorithm*

The proposed algorithm is designed to generate robust schedules by considering three different performance measures (makespan, total tardiness or total flowtime) or to generate stable schedules. We use a TS algorithm to generate these schedules. The proposed algorithm consists of two parts: a sequence generator and a sequence evaluator. We start with an initial job sequence and use the TS algorithm to scan the solution space efficiently (sequence generator). At each iteration of the TS algorithm, we assess the quality of the generated sequences using the evaluator and adopt the best sequence as the new input to the generator. The generator creates new sequences (the neighborhood of the new input), which in turn are re-evaluated. The process goes on until the stopping criterion is satisfied. The details of the neighborhood generated are given in the next section.

Using the same sequence generator, we use four different evaluators and compare their performances. As will be discussed in Section 3.4 in more detail, the first two evaluators (called method 1 and method 2) generate the realization of the job sequence (they create a breakdown/repair pattern given a sequence, which comes from the generator part of the algorithm). These realizations are then used to calculate the values of the selected SMs. We also consider the average slack method and the classical approach (which minimizes the performance measure of the initial schedule without taking breakdowns into account) as the other evaluators for benchmark purposes. These four evaluators are compared with each other in a simulation-based experimental study in Section 4. Note that the average slack method for robustness was originally developed for the job shop scheduling problem with a makespan criterion. We adapt this method to the single-machine scheduling problem by calculating the slacks of the jobs (rather than the slacks of operations) and taking the average. The reason for this modification (at the risk of deteriorating the performance of the original method) is that there is no robustness procedure that is especially designed for the single-machine problem with total flowtime or total tardiness measures (to the best of our knowledge).

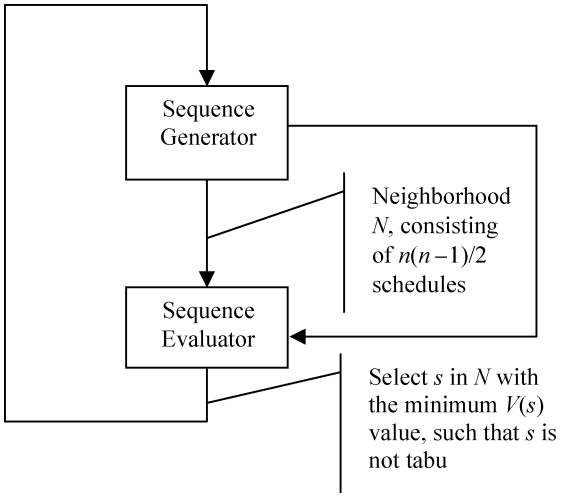


Fig. 1. Schematic representation of the proposed algorithm.

The details of the estimation methods are presented in Section 3.4. Figure 1 illustrates the basic idea behind the proposed algorithm. In the figure, $V(s)$ denotes the value of the SM calculated by the sequence evaluator for a given sequence s. Table 3 presents the formulations of $V(s)$ using the notation developed in Section 3.4.

3.3. *Sequence (neighborhood) generation*

We use swap moves to generate the neighborhood of a given schedule. The quality of the sequences in the neighborhood is evaluated in terms of the SMs discussed in the next section. The tabu list consists of triplets (x, y, z), where x is the job identification number, y is the position in the sequence and z is the remaining tabu tenure. If (x, y, z) is in the tabu list, job x cannot move to position y during the next z iterations. A swap is identified as the tabu if one of the corresponding moves is in the tabu list. For example suppose the current sequence is "*a b c d e*...". The swap of job *a* with job *e* is tabu if one of (*a*, 5, .) or (*e*, 1, .) is in the tabu list. If the swap is tabu but the resulting sequence performs better than the best sequence found so far, it is still executed due to the aspiration criterion. If all possible swaps are tabu, the swap that yields the best quality schedule in the neighborhood is executed. After execution of a swap, the corresponding two moves are added into the tabu list, if they are not already in it. For example, assume that the current candidate is "*a b c d e*...". If swap of job *a* with job *c* is executed, then (*a*, 1, .) and (*c*, 3, .) entries are added to the tabu list.

Table 3. Formulation of $V(s)$

Evaluator	*Purpose of algorithm*	$V(s)$
Method 1	Robustness	$f(s')$
Method 1	Stability	$\sum_{i=1}^{n} \lvert c_i(s) - c_i(s')\rvert$
Method 2	Robustness	$\sum_{i=1}^{n} a_i f(s_i'')$
Method 2	Stability	$\sum_{i=1}^{n} a_i(\sum_{j=1}^{n} \lvert c_j(s) - c_j(s_i'')\rvert)$
Average slack method	Robustness	$f(s)$–average slack of s
Average slack method	Stability	- average slack of s

3.4. *Sequence evaluation*

The sequences (neighbors of the current solution) generated in the previous step (Section 3.3) are now evaluated using the SMs. As stated earlier, we use one of two SMs: method 1 or method 2. These are explained in the following sections.

3.4.1. *Method 1*

In our opinion, the well known and commonly used average slack method fails to incorporate the information that can be inferred from the busy time and repair time distributions. In contrast, the first method proposed in this study, method 1, explicitly considers probability distribution functions. During estimation, multiple breakdowns are considered in a given scheduling period.

The essence of method 1 is to quickly estimate the performance of a realized schedule. Specifically, it generates a realization of the schedule (generated via the sequence generator) by considering the machine breakdown and repair events. We obtain an approximate realized schedule by inserting constant downtimes into the initial schedule after every constant busy time period. We assume that the machine fails after every busy time for a period of length $\lambda L + (1 - \lambda)U$, where λ is a real number between zero and one, and L and U are points on the left and right tails of the distribution $g(t)$, respectively. As shown in Fig. 2, the probability that a machine breakdown will occur between L and U is 0.95 (i.e, $\alpha = 0.05$). That is, L and U are the 25th and 975th tiles of the 1000-tile busy time distribution. The parameter λ is determined from a correlation study, in which a number of pilot schedules are estimated using λ values of 0.2, 0.4, 0.6 and 0.8 along with the alternative of using $E[g(t)]$ instead of $\lambda L + (1 - \lambda)U$ as the busy time period. We further assume that the repair activity lasts $E[h(t)]$ time units (i.e., the expectation of the repair time distribution).

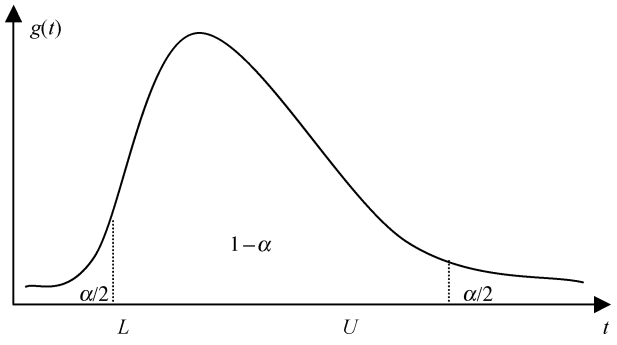


Fig. 2. Parameters of busy time distribution for method 1.

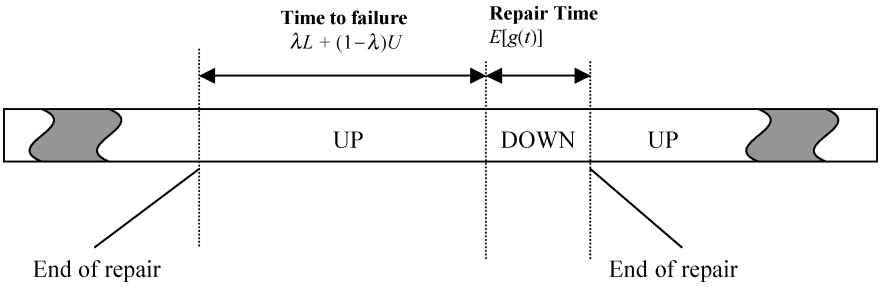


Fig. 3. Realization of the current schedule under method 1.

We find it quite satisfactory to consider only the expectation instead of incorporating more information that can be inferred from the repair time distribution because the study carried out by Mehta and Uzsoy (1999) indicates that this makes little difference. Figure 3 illustrates the estimation of the realized schedule.

In our experimental study, we use a gamma distribution with a scale parameter of 0.7 for the busy time distribution and set λ equal to 0.6 after a correlation study, whose details are given in Section 4.2. Note that the method can be applied to virtually any continuous distribution.

After the realization of the input schedule is generated, the SM for robustness or stability is calculated. We use the performance of the estimated realization as the first SM for robustness. Let s be the input sequence to the evaluator. Method 1 inserts the repair durations as explained above and let s' be the estimated realization. We use $f(s')$ as a surrogate for $E[f^{r}(s)]$, which is our robustness measure, where $f(.)$ stands for makespan, total flowtime, or total tardiness depending on the performance measure under consideration. Similarly, the sum of the absolute deviations of the job completion times between the initial sequence s and the estimated realization s' is used as the first SM for stability. That is, we use $\sum_{i=1}^{n} |c_i(s) - c_i(s')|$ as a surrogate for $E[\sum_{i=1}^{n} |c_i(s) - c_i^{r}(s)|]$.

This SM for stability is very similar to the one that is used in Mehta and Uzsoy (1999). In fact, both measures will insert the same amount of total repair duration into initial schedules if we use $E[g(t)]$ instead of a convex combination of 0.025- and 0.975-quantiles of the distribution when we estimate the busy times. Mehta and Uzsoy, however, distribute the total amount of repair duration as additional inserted idle times to jobs (proportional to their processing times) without considering the shape of the busy time distribution whereas we estimate the times of breakdown events (by using the parameters of the busy time distribution) and insert the repair duration only at this point. Another difference arises from the different interpretations of stability: Mehta and Uzsoy first solve the problem without considering the breakdowns and then insert additional idle times (aforementioned repair durations) after each job to obtain the initial schedule. Their stability measure is the sum of the absolute deviations of the job completion times between this initial schedule, which contains additional idle times, and the realized schedule. On the other hand, we confine ourselves to the domain of non-delay schedules. Hence, our stability measure is as in Wu *et al.* (1993) rather than in Mehta and Uzsoy (1999).

3.4.2. *Method 2*

Method 2 assumes that there is only one machine failure during a scheduling period. Because of this restrictive assumption, this approach can be used within a *continuous rescheduling scheme*, where the system is rescheduled from scratch after every machine breakdown (see Sabuncuoglu and Goren (2005)). The performance of the realization of a schedule s (the second SM for robustness) is estimated in three steps:

Step 1. For each job, calculate the probability that the machine fails during the processing of that job. As seen in Fig. 4, a_i is the probability that the machine fails during the processing of job i.

Step 2. For each job, determine the realized sequence assuming that the machine failure materializes during the processing of that job. Use $E[h(t)]$ as the repair duration. Let s_i'' be this realized schedule.

Step 3. Calculate the estimated realized performance measure of the sequence s as $\sum_{i=1}^{n} a_i f(s_i'')$, which is the second SM for robustness.

Similarly, $\sum_{i=1}^{n} a_i(\sum_{j=1}^{n} |c_j(s) - c_j(s_i'')|)$ is used as the second SM for stability.

The following numerical example is presented to further explain the execution of the algorithm for the robustness measure. To make things clear, all four evaluators are explained. However, only one of them would be used in an actual implementation, which the user would choose.

3.4.3. *A numerical example*

Consider the single-machine scheduling problem with three jobs, whose arrival times, processing times and due dates are given in Table 4.

Assume that the busy time distribution is Uniform [0, 3] and the repair time distribution is also Uniform [0, 2]. We use the total tardiness criterion as the performance measure (i.e., we want to generate a schedule whose expected total

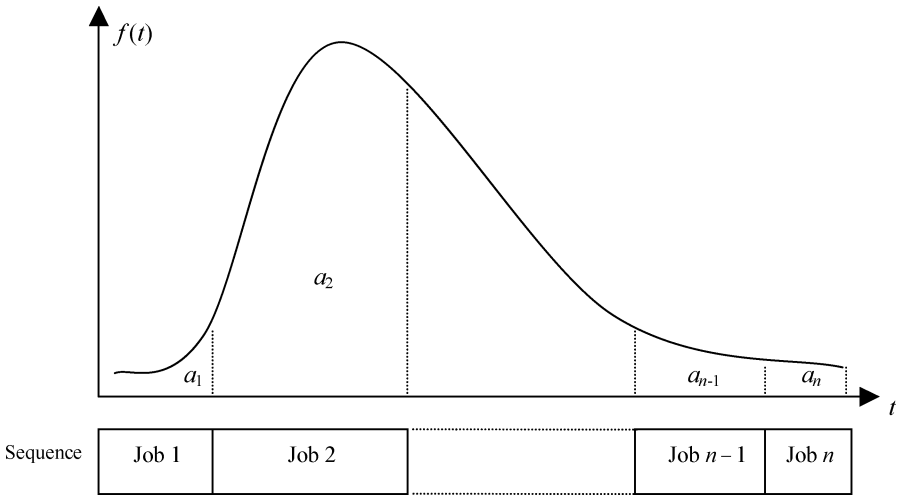


Fig. 4. Parameters of method 2.

tardiness is minimized). Now, suppose that our algorithm begins with the initial schedule "J1-J2-J3". The algorithm first generates the neighborhood of this schedule by all-pair wise swapping (there are three schedules in the neighborhood of the schedule). These are: "J2-J1-J3", "J3-J2-J1" and "J1-J3-J2". Let us call them S1, S2 and S3, respectively. Next, the algorithm evaluates the quality of S1, S2 and S3 by using the SMs. Recall that the classical approach is also used as a benchmark.

Classical approach: The algorithm calculates the initial total tardiness of each candidate schedule (i.e., S1, S2 and S3). The initial schedules are given in Fig. 5(a). The results are given in the second column of Table 5.

Average slack method: As explained at the beginning of Section 3, this method employs (initial tardiness – average system slack) as the SM for robustness. The third column of Table 5 gives the values of this measure. The calculation of average system slack can be found in Table 6.

Method 1: Method 1 calculates the total tardiness of realized versions of each candidate schedule (i.e., S1, S2 and S3). The L parameter of method 1 is 0.075 and the value of U is 2.925 if we take $\alpha = 0.05$ (see Fig. 6). Assume that the value of λ is 0.5. As explained in Section 3.4.1, when estimating realization according to method 1, the algorithm inserts $E[h(t)] = 1$ hour of idle time as the repair duration after every $0.5 \times 0.075 + 0.5 \times 2.925 = 1.5$ hours of busy time period. The estimated realized schedules can be seen in Fig. 5(b). The fourth column of Table 5 displays the total tardiness values of these realizations.

Method 2: As explained in Section 3.4.2, this procedure assumes that the machine breaks down once and calculates the expected total tardiness of the estimated realization of each candidate schedule. Note that the probability of machine failure during the processing of any job (J1, J2 or J3) is one-third. Method 2 first calculates the total tardiness of the realized schedule assuming that the breakdown occurs during the processing of J1, then J2 and finally J3. These total tardiness values can be found in Table 7. The weighted average of these values (weights being the corresponding probabilities, all one-third in this case) gives us the SM of method 2. The values of the SM are given in the fifth column of Table 5. Figure 5(c) illustrates the above calculation for candidate S1 as an example.

Next, the algorithm selects the best schedule in the neighborhood. As seen in Table 5, S2 (J3-J2-J1) happens to be the best solution in the neighborhood for all evaluation methods. Therefore, S2 will be used as the new current schedule. This move involves the swap of J3 with J1. Assume that the tabu tenure is 12. Then, (J1, 1, 12) and (J3, 3, 12) entries are

Table 4. Job parameters for the numerical example

Job	*Arrival time (hour)*	*Processing time (hour)*	*Due date (hour)*
J1	5	1	6
J2	2	1	5
J3	1	1	4

Table 5. Evaluation of the neighborhood

	Evaluated candidate qualities			
Schedule in the neighbourhood	*Classical approach*	*Average slack method*	*Method 1*	*Method 2*
J2-J1-J3	3	2.33	5	4
J3-J2-J1	0	−1.33	0	0.33
J1-J3-J2	6	6	8	8

Table 6. Calculation of the average system slack level

	S1			*S2*			*S3*		
Job	*Earliest*	*Latest*	*Slack*	*Earliest*	*Latest*	*Slack*	*Earliest*	*Latest*	*Slack*
J1	5	5	0	5	5	0	5	5	0
J2	2	4	2	2	4	2	7	7	0
J3	6	6	0	1	3	2	6	6	0
	Average slack		0.67	Average slack		1.33	Average slack		0

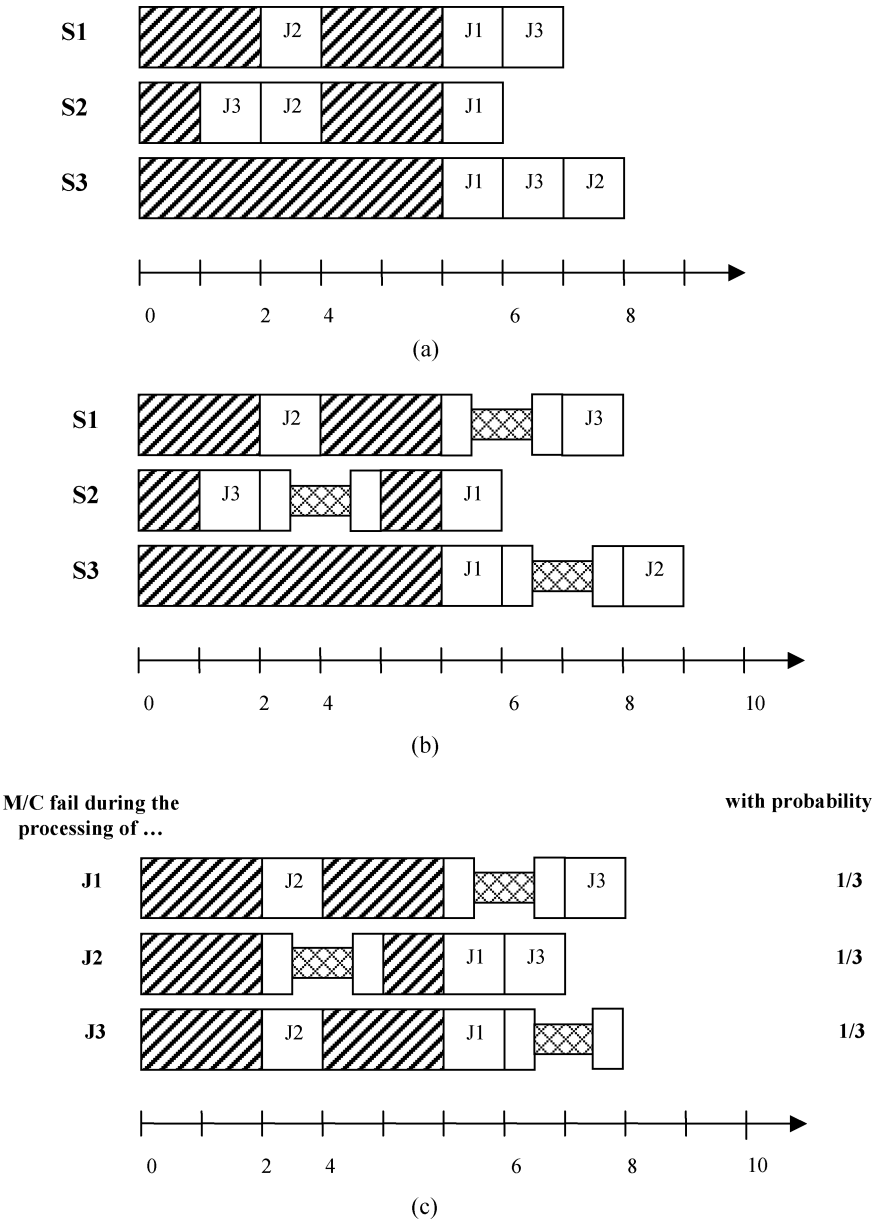


Fig. 5. Schematics of the numerical example: (a) candidate schedules for S1, S2 and S3; (b) estimated realizations (S1, S2 and S3) under method 1; and (c) calculation of expectation for S1 under method 2.

Table 7. Calculation of SM for robustness (method 2)

	Total tardiness of realization if machine fails during processing of . . .			
Schedule	*J1*	*J2*	*J3*	*Expectation*
S1	5	3	4	4
S2	1	0	0	0.33
S3	9	7	8	8

added to the tabu list (which was empty). The algorithm continues until the stopping criterion is met.

4. Computational results

We conducted extensive computational experiments in order to tune-up the parameters and evaluate the performance of the proposed algorithm. We solved a broad range of test problems, whose details are given in the following section. In general, we assumed a periodic scheduling scheme i.e., scheduling decisions are made periodically (Sabuncuoglu and Goren, 2005). Thus, when a breakdown event occurs the jobs in the schedule are right shifted by the length of the repair time. For the second SM, however, we used a continuous scheduling scheme in which the system is scheduled from scratch, due to the assumption of a single breakdown in any scheduling period. Hence, both the continuous and periodic schedulings were implemented in the experiments. The proposed scheduling system, programmed in the C language, reads the problem parameters from an input file and generates the desired schedules. The computational environment and CPU times are given in Section 4.5.

4.1. *Experimental environment*

4.1.1. *Problem parameters*

We used the data generation scheme previously proposed by Mehta and Uzsoy (1999) to create test problems.

Number of jobs (n): We considered five levels of the number of jobs (n = 10, 30, 50, 70 and 90). In general, the number of jobs designates the size of the problem. As it increases, the computational burden and the amount of time needed to find the optimal solution also increase.

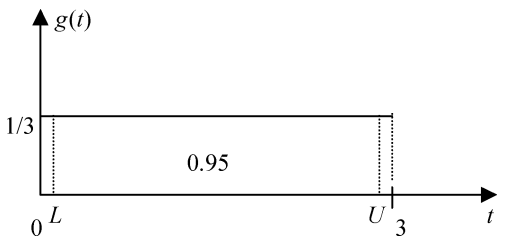


Fig. 6. The L and U parameters of method 1.

Processing time (p_i): Job processing times were generated from discrete uniform distributions. Two such distributions were used – Uniform [1, 11] and Uniform [4, 8], which are referred to as P1 and P2, respectively. P1 and P2 have the same mean, but the variance of P1 is higher than that of P2. Hence, we also tested the performance of the algorithm as a function of variability in the system.

Arrival times (r_i): Job arrival times were generated from a discrete uniform distribution between zero and $\alpha n E[p_i]$, where $E[p_i]$ is the expected job processing time (= 6 time units). Therefore, $nE[p_i]$ is the expected makespan of the schedule. A low level for α makes the jobs arrive in a shorter time horizon. Note that $\alpha = 0$ means that all jobs are ready at time 0.

Job due dates (d_i): Job due dates were generated as $d_i = r_i + \gamma p_i$, with γ being generated from a continuous uniform distribution with the parameters of a and b. Different a and b levels determine the tightness and the range of the due dates. Four levels of (a, b) were considered as shown in Table 8. D1 and D2 have tighter mean due dates than D3 and D4. The range of due dates are greater for D1 and D4 and smaller for D2 and D3. Note that we have 200 experimental design points from the combination of the above parameters for test problems (See Table 8).

4.1.2. *Breakdown parameters*

In the absence of real data, Law and Kelton (2000) recommend the use of a gamma distribution for the busy time distribution with a shape parameter of 0.7, and a scale parameter to be specified. They also state that a gamma distribution with a shape parameter of 1.4 is appropriate to

Table 8. Test problem parameters

Parameter	*Values used in experimentation*	*Total values*
Number of jobs (n)	10, 30, 50, 70, 90	5
Processing times (p_i)	Uniform [1, 11] (P1) Uniform [4, 8] (P2)	2
Arrival times (r_i)	a_i = Uniform (0, $a_{\max}$) where $a_{\max} = \alpha \mathrm{E}[C_{\max}]$ α = 0.25, 0.5, 0.75, 1.25, 1.75	5
Job due date (d_i)	$d_i = r_i + \gamma p_i$ where γ = uniform $[a, b]$ where values of (a, b) are taken as (−1, 3) (D1) (0, 2) (D2) (2, 4) (D3) (1, 5) (D4)	4

Table 9. Breakdown parameters

Parameter	*Values used in experimentation*	*Total values*
Busy time	Gamma distribution with a shape parameter of 0.7 and mean of $\theta E[p_i]$ $\theta = 10.0$ (long busy time) (B1, B2) $\theta = 3.0$ (short busy time) (B3, B4)	2
Downtime	Gamma distribution with a shape parameter of 1.4 and mean of $\beta E[p_i]$ $\beta = 1.5$ (long repair time) (B1, B3) $\beta = 0.5$ (short repair time) (B2, B4)	2

describe the downtime distribution. The scale parameter of the busy time distribution was arranged so that the mean was $\theta E[p_i]$. We considered θ values of ten and three. Mehta and Uzsoy (1999) use the same scheme except that an exponential distribution is used instead of a gamma distribution. The scale parameter of the downtime distribution was set such that the mean was $\beta E[p_i]$. We considered β values of 1.5 and 0.5. Consequently, we have 200 problem combinations (Table 8) and four breakdown combinations (Table 9) giving 800 problem *classes* as a whole.

4.2. *Fine-tuning of algorithm parameters*

We conducted pilot runs to set the parameters of the scheduling algorithm. These are:

Stopping criterion: This parameter determines when the scheduling algorithm should cease searching the solution space.

Tabu tenure: This parameter dictates the number of iterations for which a move should remain in the tabu list after inclusion. We considered five levels of tabu tenure in our pilot runs that were conducted for fine-tuning purposes: 1, 5, 7, 10 and 15.

Objective functions: We considered five objective functions in our pilot runs. Total tardiness, total flowtime, and total system slack of the initial schedule, total tardiness and total flowtime estimated by method 1. Note that method 1 employs the breakdown distributions. Instead of using the breakdown parameters presented above, which would lead to four method 1 variants, we used an average breakdown scheme to keep the pilot runs simple: we set $\theta = 6$ and $\beta = 1$. Similarly we used the expectation ($\theta E[p_i] = 6 \times 6 = 36$) instead of $\lambda L + (1 - \lambda)U$ as the busy time period, because the optimal level for λ has not yet been determined.

In order to fine tune the parameters of the algorithm (stopping criterion, tabu tenure, λ) we generated 5000 problems created from 200 problem combinations, five tabu tenure levels and five objective functions. We ran the algorithm on each problem for 1000 iterations. We recorded the number of iterations between the improvements of the best solution updates of the TS. We also recorded the best objective value attained for each problem. We observed that for 96.4% of the test problems, it is possible to stop the algorithm when there is no improvement for 20 iterations. We deliberately did not include method 2 in our pilot runs because the numerical integration of the density function of the gamma distribution (needed to calculate relevant probabilities of method 2) and rescheduling the system from scratch after every breakdown requires excessive computer time. Based on the results, we also used random tabu tenures that use the discrete uniform distribution with parameters of 10 and 15.

Determining the best λ value: We generated five instances for each of the 800 possible parameter combinations (Tables 8 and 9) to set the λ value of method 1. There were two objective functions in the TS: total tardiness and total flowtime of the estimated realizations (SM for robustness). We used 0.2, 0.4, 0.6 and 0.8 as candidate λ values. We also used $E[g(t)]$ instead of $\lambda L + (1 - \lambda)U$ as the busy time period of the machine. We simulated the "best so far" schedule five times, and recorded the average objective function values as well as the average estimate of method 1 for these values. We also conducted a simple linear regression analysis using 800 data points (corresponding to the total number of factor combinations). The dependent variable was the expected realized performance, that is, the robustness measure. This was taken as the average of the 25 simulation runs (five replications $\times$ five instances). The independent variable was the average method 1 estimates of the performance measures (SM for robustness). The regression analysis was repeated for each λ level. The R^2 (coefficient of determination) values were also recorded. Since the coefficients of the determinations are very close to each other (and they are almost equal to unity), the choice of λ is not important with respect to R^2, hence, we determined the value of λ by considering the expected performance of the realized schedule (i.e., the λ value which generates the most robust schedules). After examining the simulation results, $\lambda = 0.6$ was chosen as the overall best. The R^2 values for this selection were 0.9744 and 0.9683 when the total flowtime and total tardiness criteria were used, respectively.

4.3. *Evaluation of method 1*

4.3.1. *Robustness*

We compare the performance of method 1 with the average slack approach. The classical approach, which minimizes the planned (initial) performance measure, is also used as a benchmark. Leon *et al.* (1994) estimate the realized performance measure as given below:

$$E[f(S)] = f(S_0) + E[\delta(S)],$$

Table 10. Summary table for robustness results

Evaluation method	*Makespan*	*Total tardiness*	*Total flow time*
Robustness			
Classical approach	400.25	3801.64^{+}	3915.99
Slack method	399.76*	3792.81	3915.99
Method 1	400.14	3755.19*	3873.46*
Stability			
Classical approach	1381.04^{+}	1159.66	1160.56
Slack method	1337.55*	1158.46	1160.44
Method 1	1358.24	815.17*	815.26*

where $E[f(S)]$ is the expectation of the realized performance measure (our robustness measure); $f(S_0)$ is the initial (or planned) performance measure of S, and $E[\delta(S)]$ is the degradation in the performance measure because of the disruptions, which is estimated by using –(average system slack) as the SM.

We generated five instances for each of 800 possible problem classes (see Tables 8 and 9), leading to 4000 experimental design points. We ran the TS algorithm with three different evaluators (the classical approach, the average slack method and method 1) for three performance measures (makespan, total tardiness and total flowtime). This yielded $4000 \times 3 \times 3 = 36\,000$ instances. We simulated the solutions to obtain the expected realized performances, that is, the value of robustness measures (taken as the average of five simulations that yields $36\,000 \times 5 = 180\,000$ simulation runs), and stability measures (total absolute job completion time differences). Table 10 presents the averages of the results. Note that the stability values are also given even though the primary objective is to optimize the robustness. The figures marked with "+" sign are the worst in their group, whereas the figures marked with a "*" sign are the best according to paired-t tests with $\alpha = 0.05$.

After a careful examination of these results, we can make the following observations: In terms of robustness, the classical approach is very poor. It displays the worst performance of all the criteria but the difference is statistically significant only for the total tardiness criterion. In contrast, the proposed method (method 1) is statistically better than both the average slack method and the classical approach for the total tardiness and total flowtime criteria. It is also very competitive for the makespan criterion. Note that the numerical difference from the average slack method is very small (practically insignificant), yet it is statistically significant.

In terms of stability, the classical approach is again the worst, but the difference is significant only for the makespan criterion. In general, method 1 and the average slack method are very competitive. The average slack method is better than method 1 for the makespan criterion. However, method 1 is better than the average slack method for the other two criteria (total tardiness and total flowtime). A slightly improved performance of the average slack method over method 1 for makespan in the stability case can be attributed to the following fact: as stated before, the average slack method estimates the expected makespan (the robustness measure) as $E[f(S)] = f(S_0) + E[\delta(S)]$. The initial makespan of any non-delay schedule (i.e., a schedule that does not keep a machine idle when there is a job waiting in the queue) is constant. Therefore, minimizing $E[f(S)] = f(S_0) + E[\delta(S)]$ is equivalent to minimizing $E[\delta(S)]$, which is estimated by the average system slack level. As a result, the average system slack method optimizes the robustness by maximizing the average system slack level for the makespan criterion, which is exactly the same approach used to optimize the stability (this argument is only valid for non-delay schedules). The average slack method inherently optimizes the stability while minimizing the expected makespan (robustness measure), whereas method 1 does not consider stability when it is used to optimize the robustness. However, the proposed method is better than the average slack method when the other two performance measures are considered.

In summary, to optimize the robustness, the average slack method is the appropriate choice for the makespan criterion (after all this is the criterion for which it is originally developed) whereas method 1 should be recommended for the total tardiness or total flowtime criteria. Note that the use of the makespan criterion does not make sense in dynamic problems since there is no fixed job population at any point in time in the system due to new job arrivals. Instead, practitioners often use flowtime and tardiness-related performance measures in these environments.

4.3.2. *Stability*

Unlike the robustness, the stability of schedules does not depend on the regular performance metrics such as flowtime and tardiness since it measures the deviation of the realized schedule in terms of completion times. For that reason, we solved 12 000 instances instead of 36 000 instances. Table 11 presents the averages of the results. Again, the figures marked with a "+" sign are the worst in their group, whereas the figures marked with a "*" sign are the best according to paired-t tests with $\alpha = 0.05$.

We make the following observations from the results. In terms of stability, which is the primary goal in these

Table 11. Summary results for stability

Evaluation method	*Makespan*	*Total tardiness*	*Total flow time*
Stability			
Classical approach	1381.04^{+}	1159.66^{+}	1160.56^{+}
Slack method	904.96	904.96	904.96
Method 1	755.52*	755.52*	755.52*
Robustness			
Classical approach	400.25*	3801.64*	3915.99*
Slack method	495.24^{+}	7891.78^{+}	8000.26^{+}
Method 1	425.74	5065.84	5174.91

experiments, the proposed method (method 1) outperforms the other methods since it yields a significantly better performance than the average slack method and the classical approach. Hence, it is clearly the winner. When we consider the secondary goal (robustness), however, the classical approach is significantly better than the other two methods (the average slack method and method 1). This can be explained as follows: the robustness value of the average slack method and method 1 deteriorate when the primary goal is changed to optimize stability. However, the classical approach does not suffer from this deterioration, since it does not consider stability at all. As a result, it yields a good robustness performance. These observations also indicate a trade-off between robustness and stability. This trade-off is further discussed in Section 4.3.3.

4.3.3. *Bicriterion approach*

When considering two performance measures simultaneously, we minimized the objective function of "$y = w \times$ *robustness measure* $+ (1 - w) \times$ *stability measure*," where w is a real number between zero and one that represents the weight given to a particular measure. Recall that w is equal to one (pure robustness) in Section 4.3.1 and it is equal to zero (pure stability) in Section 4.3.2. We applied the linear combination approach used in Leon *et al.* (1994) to see how the proposed SM works under their conditions. We designed the experiments to see if it is possible to maintain a good realized schedule performance (robustness) while significantly increasing schedule stability when w is close to one. We took $w = 0.85$ and used 0.85 *robustness measure* + 0.15 *stability measure* as the neighborhood evaluation function of the TS. Again, we included the classical approach as a benchmark.

Table 12 presents a summary of the results (for 36 000 instances). Table 13 gives the 0.85 *robustness* + 0.15 *stability* values.

Now we can make the following observations. In terms of robustness, the average slack method is statistically the worst for the makespan criterion and method 1 is the worst for the total tardiness and total flowtime criteria. The classical approach performs the best for the makespan criterion. For the other criteria (total tardiness and total flowtime) the average slack method and the classical approach are very competitive and the differences between them are not statistically significant.

Table 12. Summary table for bicriterion approach results

Evaluation method	*Makespan*	*Total tardiness*	*Total flow time*
Robustness			
Classical approach	400.25*	3801.64	3915.99
Slack method	404.36$^+$	3796.77	3917.34
Method 1	402.28	3823.97$^+$	3934.74$^+$
Stability			
Classical approach	1381.04$^+$	1159.66	1160.56
Slack method	1329.49	1159.41	1161.07
Method 1	853.89*	805.17*	805.40*

Table 13. Values obtained using the composite objective function 0.85 *robustness* + 0.15 *stability*

Evaluation method	*Makespan*	*Total tardiness*	*Total flow time*
Classical approach	547.37$^+$	3405.34	3502.67
Slack method	543.13	3401.17	3503.90
Method 1	470.02*	3371.15*	3465.34*

In terms of stability, method 1 performs the best. In general, the classical approach and the average slack method are competitive. Even though the average slack method performs better than the classical approach for the makespan and total tardiness criteria, the difference is statistically significant only for the makespan criterion.

In short when a composite objective function (which is the primary objective) is considered the proposed method (method 1) performs statistically better than the average slack and the classical approach for all three performance measures. Both the classical approach and the average slack method are competitive. The average slack method is better than the classical approach for the makespan and total tardiness criteria, even though the classical approach is better for the total flowtime criterion. The difference in their performances is slight and it is significant for only the makespan criterion.

The behavior of method 1 when both criteria are considered can be explained as follows. The robustness measure is not highly dependent on the sequence. An initial sequence with a high traditional performance measure is likely to have a high expected realized performance. This can be confirmed by comparing the robustness measures in Table 10. The independence from sequence is also stated by Mehta and Uzsoy (1999) for the maximum lateness measure. Although the robustness values of method 1 and the classical approach are very close to each other, we observe that there is a significant difference between the stability values. The difference in stability values suggests that the solutions of the two methods are significantly different, but despite this difference their expected realized performances are close. Intuitively, there is a set of "elite" solutions with high expected realized performance values, but their stabilities differ drastically. From Tables 10–13 we can see that the classical approach is inclined to give better robustness values whereas method 1 is inclined to give better stability values no matter what the primary objective. When a composite objective is considered, the classical approach sticks to the schedules with the best performance ignoring stability (after all, this is what it is designed for) and the average slack method is clearly better able to generate stable schedules rather than robust ones (see Table 11); both methods lose their competitive advantages. On the other hand,

Table 14. Robustness under method 1 (summary)

Evaluation method	*Robustness values*		
	$w = 0$	$w = 0.85$	$w = 1$
Makespan	425.74	402.28	400.14
Total tardiness	5065.84	3823.97	3755.19
Total flowtime	5174.91	3934.74	3873.46

Table 15. Stability values under method 1 (summary)

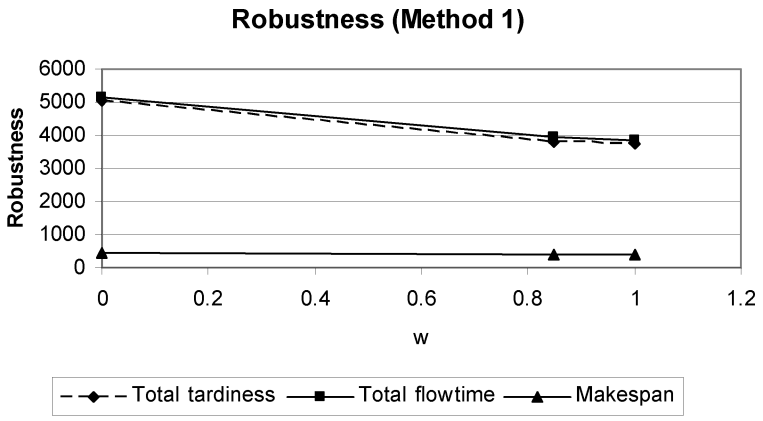
Evaluation method	*Stability values*		
	$w = 0$	$w = 0.85$	$w = 1.0$
Makespan	755.52	853.89	1358.24
Total tardiness	755.52	805.17	815.17
Total flowtime	755.52	805.40	815.26

method 1 has enough flexibility to scan the set of "elite" solutions and find the most stable schedule with a good robustness value. Tables 12 and 13 confirm this intuition.

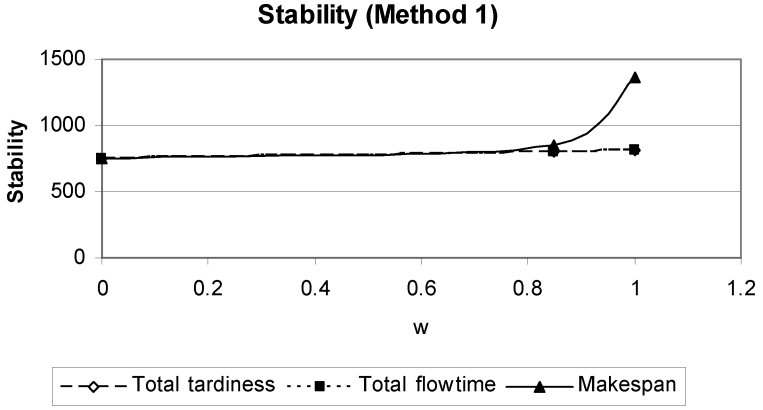
We do not, however, claim that our proposed algorithm is very good at handling robustness and stability together in a multi-criterion setting. We are aware of the fact that using the linear combinations of conflicting objective functions is not a good approach. Previous findings of Leon *et al.* (1994) and Mehta and Uzsoy (1999) suggest that robustness and stability are conflicting objectives. The aim is only to shed some light on the question of whether it is possible to maintain an acceptable level of robustness while achieving high stability values, and to understand the nature of the conflict. Our results show that it is possible to achieve high stability measures without sacrificing much from the robustness level. A method which uses the estimation logic of method 1 and is capable of generating a set of non-dominated solutions instead of using a linear combination may be designed to give better results.

We further elaborate on the performance of method 1 for varying values of w. Our experimental results indicate that method 1 performs generally better than the other methods when the concern is to minimize ($w \times$ *robustness* $+ (1 - w) \times$ *stability*) measure, for $w = 0$, $w = 0.85$ and $w = 1.0$ (Sections 4.3.1, 4.3.2 and 4.3.3). Tables 14 and 15, and Fig. 7 summarize the experimental results.

As seen in Fig. 7(a) for the makespan criterion, the realized performance (robustness) is fairly constant, (i.e., the



(a)



(b)

Fig. 7. (a) Robustness values; and (b) stability values.

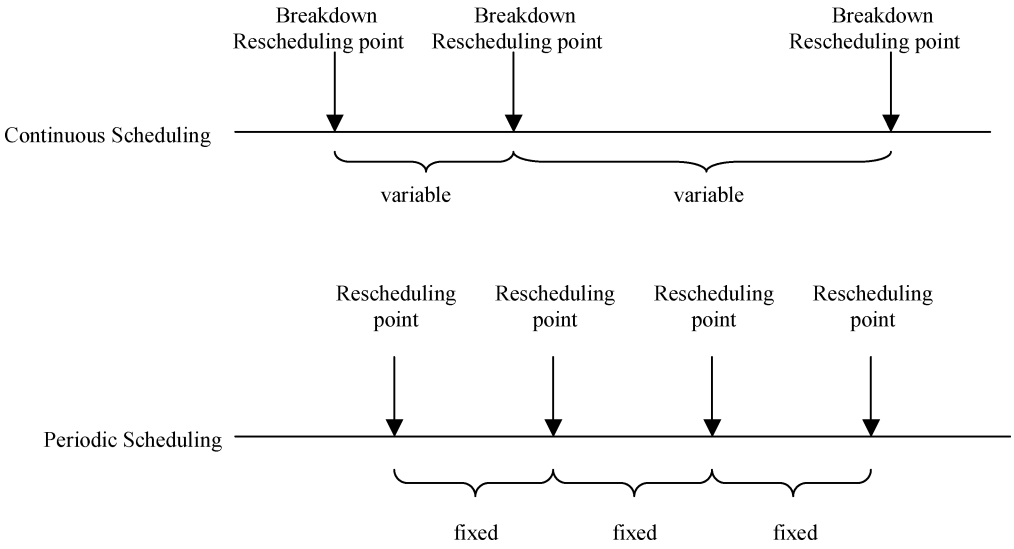


Fig. 8. Continuous and periodic scheduling (see Sabuncuoglu and Goren, 2005).

robustness plot is nearly horizontal). From Fig. 7(b) it can be seen that the stability plot is linear with a slight slope up to $w = 0.85$, after which it abruptly increases. For the other two performance measures (total tardiness and total flowtime), we see that the plot for both robustness and stability are nearly linear. That is, as w increases, the robustness improves gradually, whereas the stability gradually deteriorates. The absolute value of the slope of the robustness line ($\approx$1300) is much greater than the slope of the stability line ($\approx$60). We see that the robustness is more sensitive to changes in w. Moreover, the lines are nearly parallel to each other. This indicates that the trade-off between the robustness and the stability is independent of the performance measure in use.

From the above two observations, we conclude that practitioners should place emphasis on the stability, rather than the robustness for the makespan criterion. For the other two performance measures (i.e., total tardiness and total flowtime) there is a linear trade-off (e.g., if we increase w by Δw, we gain $1300\Delta w$ units of the robustness, whereas we lose $60\Delta w$ units of the stability for our problem set). We can say that practitioners should pay more attention to robustness than stability when the performance measure is the total tardiness or total flowtime, on seeing too small an improvement in stability associated with a substantial sacrifice from robustness.

4.4. *Evaluation of method 2*

We compare the performance of method 2, from the viewpoint of the robustness, to the average system slack and method 1. We also use the classical approach as a benchmark. As stated in Section 3.4.2, method 2 assumes that the machine experiences a single breakdown in a given scheduling period. Because of this restrictive assumption, we use a continuous scheduling scheme under which the entire system is rescheduled from scratch after every disruption (machine breakdown in our case), rather than a periodic scheduling scheme in which the system is rescheduled periodically. In Fig. 8, the heads of the arrows denote the scheduling points in these two different policies.

We excluded 70-job and 90-job problems due to the high computational burden of the continuous rescheduling approach. We generated five instances for each of the remaining 480 possible problem classes (see Tables 8 and 9), leading to 2400 experimental design points. We ran our TS algorithm with four different evaluation functions (the classical approach, the slack method, method 1 and method 2) for three performance measures (makespan, total tardiness and total flow time). This resulted in $2400 \times 4 \times 3 = 28\,800$ runs. After a machine breakdown, the remaining jobs were rescheduled from scratch. We recorded the estimated performance measures, the realized performance measures, and the stability measure (total absolute job completion time difference). Table 16 presents a summary of

Table 16. Summary table for continuous scheduling results (robustness)

Evaluation method	*Makespan*	*Total tardiness*	*Total flow time*
Robustness			
Classical approach	237.89	1250.31	1315.24
Slack method	237.29*	1249.75	1315.05*
Method 1	239.90	1339.10^{+}	1404.70
Method 2	242.47^{+}	1291.99	1324.86
Stability			
Classical approach	374.65	378.99	378.31
Slack method	363.14	372.91*	375.76*
Method 1	476.08	432.52	436.64^{+}
Method 2	1188.80^{+}	412.95	378.44

the results. Note that the stability values are also given even though our objective is to optimize the robustness. We compared the best two methods with each other for each group of Table 16. If the difference is statistically significant according to a paired-t test with $\alpha = 0.05$, the figure that summarizes the performance of that method is marked with a "*" sign. Similarly, if the difference between the worst two methods is statistically significant, we mark the corresponding figure with a "+" sign.

After careful analysis of the presented results we can make the following observations. In terms of robustness, the numerical values are very close to each other but the differences are not statistically significant for the makespan criterion. Method 2 yields the numerically worst results. The average slack method performs the best. This is somewhat counterintuitive because one expects a continuous rescheduling scheme to yield the best results in terms of robustness, since it is able to make changes after each breakdown. We also observe similar results for the total tardiness and total flowtime criteria. The good performance of method 1 also deteriorates significantly in a continuous scheduling scheme and it yields the numerically worst results for the total tardiness and total flowtime criteria, where the difference is also statistically significant for the total tardiness case. The inferior performance of method 1 can be explained as follows: method 1 considers the possibility of future breakdowns and generates the initial schedule accordingly. However, according to the continuous scheduling scheme we stop the execution of this schedule as soon as the first breakdown event occurs. This premature interruption of the schedule adversely affects the performance of method 1. We conclude that method 1 is suitable for a periodic scheduling scheme. Our results also indicate that method 2 does not perform well in the continuous scheduling case due to the single breakdown assumption. Leon *et al*. (1994) have already reported a similar result for the makespan criterion under a job shop environment in a periodic scheduling context. Our results indicate that this result is also valid for total tardiness and total flowtime criteria, even if a continuous scheduling scheme is used.

In terms of stability, the average slack method yields the best results for all three criteria. The differences are statistically significant for the total tardiness and total flowtime criteria.

In conclusion, the average slack method is suitable for a continuous scheduling approach. On the other hand, scheduling after each machine breakdown creates system nervousness. Thus, there is a need to balance robustness gains and stability loses in the continuous scheduling scheme. This trade-off should be investigated in future research studies. Note that, as Church and Uzsoy (1992) discuss, the benefit of extra scheduling diminishes rapidly. We believe that in the adaptive scheduling approach (where a scheduling process is triggered after a certain amount of deviation from the initial schedule occurs), the proposed methods, especially method 1, can maintain its high performance. This topic deserves further investigation.

Table 17. Average CPU time required by evaluators (periodic scheduling)

Number of jobs	*Evaluator*	*Average CPU time (seconds)*
10	Classical	0.017
	Average slack	0.040
	Method 1	0.033
30	Classical	0.415
	Average slack	1.168
	Method 1	0.824
50	Classical	2.301
	Average slack	7.175
	Method 1	4.316
70	Classical	7.623
	Average slack	26.883
	Method 1	13.714
90	Classical	18.271
	Average slack	75.270
	Method 1	32.329

4.5. *Computational times*

We recorded the solution times in terms of CPU time in seconds for each problem. These measurements were performed by utilizing the `clock()` function of the standard C library. The computational experiments were performed on a Sun4u Sparc Ultra-Enterprise Sun Workstation running under SunOS 5.7 with a 3 GB memory. The machine is, however, not solely dedicated to our computational tests. In fact, the average CPU capacity dedicated to our tests was about 1% throughout the experiment. Tables 17 and 18 and Figs. 9 and 10 present the averages of the solution times.

It can be observed that the smallest computational effort is required by the classical approach and the greatest

Table 18. Average CPU time required by evaluators (continuous scheduling)

Number of jobs	*Evaluator*	*Average CPU time (seconds)*
10	Classical	0.008
	Average slack	0.016
	Method 1	0.029
	Method 2	0.090
30	Classical	0.394
	Average slack	1.175
	Method 1	0.976
	Method 2	7.337
50	Classical	2.613
	Average slack	10.213
	Method 1	5.759
	Method 2	90.339

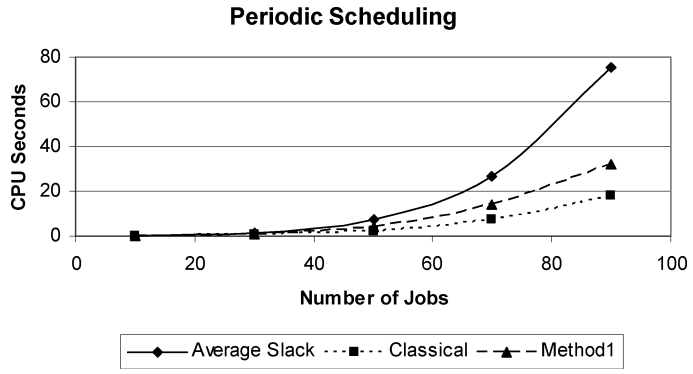


Fig. 9. Average CPU time of evaluators (periodic scheduling).

burden is incurred by the average slack method among the periodical scheduling methods. For the continuous scheduling schemes, method 2 requires the most amount of time whereas the classical approach is associated with the smallest computational effort.

5. Concluding remarks

In this paper, we studied robustness and stability in a scheduling context. Specifically, we developed two new SMs. We also embed these SMs into a TS algorithm so as to be able to generate robust and stable schedules for a single machine subject to random machine breakdowns.

We compared method 1 with the average slack method and the classical approach in a periodic scheduling scheme where the response to machine breakdowns is to right shift the remaining jobs. We first compared the alternative approaches from the viewpoint of robustness with stability being considered to be a secondary goal. The results indicated that the classical approach is not very good, and method 1 is better than the average slack method for the total tardiness and total flowtime criteria. As for the makespen criterion, although the difference between method 1 and the average slack method is slight, it is, however, statistically significant.

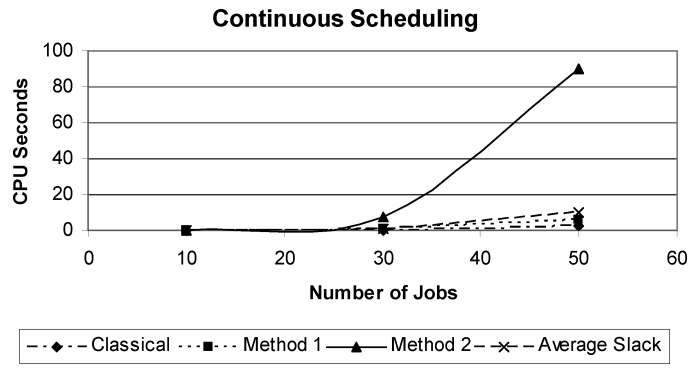


Fig. 10. Average CPU time of evaluators (continuous scheduling).

Method 1 also yields significantly improved stable schedules compared to the average slack method for the total tardiness and total flowtime criteria.

Next, we compared the methods from the viewpoint of stability, considering robustness as a secondary objective. We observed that method 1 generates more stable schedules than others for all three performance measures. We also observed that the classical approach does not perform well in terms of stability (statistically the worst alternative), however, it is statistically the best alternative in terms of robustness.

Then, we examined whether it is possible to maintain high robustness levels and improve stability when we use a bicriterion approach. We used 0.85 *robustness* + 0.15 *stability* as the neighborhood evaluation function in the proposed algorithm. We observed that the classical approach is generally better in terms of robustness and the average slack method is generally better in terms of stability although method 1 is competitive for both robustness and stability. However, we observed that when we evaluated the composite objection function, which considers both robustness and stability simultaneously, method 1 is better. In addition we found that it is possible to maintain high robustness levels while significantly improving stability by means of this bicriterion approach, even when the weight of the stability is as small as 0.15.

Then we analyzed the performance of method 1 with varying weights (w) given to the robustness. We observed that for the makespan criterion, practitioners should place more emphasis on stability because robustness is insensitive to the value of w. We also observed that there is a linear trade-off between robustness and stability (i.e., improving one deteriorates the other). Our experimental results also show that stability is less sensitive to changes in w than robustness (i.e., the absolute value of the slope of the line that plots stability values against varying values of w is less than that of robustness).

Finally, we compared the performance of method 2 with the other alternatives (the classical approach, the average slack method and method 1) from the viewpoint of robustness, taking stability as the secondary goal. Since method 2 assumes that the machine only breaks down once during a scheduling period, we used it in a continuous scheduling environment where the whole system is rescheduled from scratch after a machine breakdown. The average slack method performed well in such an environment.

We completed this study under a static environment with non-zero job arrival times (can be called semi-dynamic). Actually, we do not know the performance of the proposed methods in a true dynamic environment. Also, we do not know the stability performance of the proposed methods in a continuous scheduling environment. These topics should be investigated in future studies. Furthermore, the use of the linear combination approach for multi-objective problems leaves out a lot of non-dominated solutions when dealing with a scheduling problem. Hence, a more thorough

study can be conducted when considering stability and robustness together, such as applying an epsilon-constraint approach.

References

Aytug, H., Lawley, M., McKay, K., Mohan, S. and Uzsoy, R. (2005) Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, **161**(1), 86–110.

Church, L.K. and Uzsoy, R. (1992) Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, **5**, 153–163.

Daniels, R.L. and Kouvelis, P. (1995) Robust scheduling to hedge against processing time uncertainty in single-stage production. *Management Science*, **41**(2), 363–376.

Davenport, A.J. and Beck, J.C. (2000) A survey of techniques for scheduling with uncertainty. available at http://eil.utoronto.ca/profiles/chris/chris.papers.html, accessed September 2003.

Herroelen, W. and Leus E. (2005) Project scheduling under uncertainty: survey and research potentials. *European Journal of Operational Research*, **165**(2), 289–306.

Kouvelis, P. and Yu, G. (1997) *Robust Discrete Optimization and its Applications*, Kluwer, Boston, MA.

Kutanoglu, E. and Sabuncuoglu, I. (2001) Experimental investigation of iterative simulation-based scheduling in a dynamic and stochastic job shop. *Journal of Manufacturing Systems*, **20**(4), 264–279.

Law, A.M. and Kelton, W.D. (2000) *Simulation Modeling and Analysis*, 3rd ed., McGraw-Hill, Singapore.

Leon, V.J., Wu, S.D. and Storer, R.H. (1994) Robustness measures and robust scheduling for job shops. *IIE Transactions*, **26**(5), 32–43.

Mehta, S.V. and Uzsoy, R. (1998) Predictable scheduling of a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation*, **14**(3), 365–378.

Mehta, S.V. and Uzsoy, R. (1999) Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, **12**(1), 15–38.

O'Donovan, R., Uzsoy, R. and McKay, K.N. (1999) Predictable scheduling of a single machine with breakdowns and sensitive jobs. *International Journal of Production Research*, **37**(18), 4217–4233.

Pinedo, M. (1995). *Scheduling: Theory, Algorithms, and Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Sabuncuoglu, I. and Goren, S. (2005) A review of reactive scheduling research: proactive scheduling and new robustness and stability measures. Technical Report, IE/OR 2005-02, Department of Industrial Engineering, Bilkent University, Ankara.

Sotskov, Y., Sotskova, N.Y. and Werner, F. (1997) Stability of an optimal schedule in a job shop. *Omega: the International Journal of Management Science*, **25**(4) 397–414.

Wu, S.D., Byeon, E. and Storer, R.H. (1999) A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, **47**(1), 113–124.

Wu, S.D., Storer, R.N. and Chang, P. (1993) One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research*, **20**(1), 1–14.

Biographies

Selcuk Goren is a Ph.D. candidate in the Department of Industrial Engineering at Bilkent University. He received his B.S. and M.S. degrees from Bilkent University. His primary research interest is scheduling under uncertainty.

Ihsan Sabuncuoglu is a Professor of Industrial Engineering at Bilkent University. He received B.S. and M.S. degrees in Industrial Engineering from the Middle East Technical University and a Ph.D. degree in Industrial Engineering from Wichita State University. Dr. Sabuncuoglu teaches and conducts research in the areas of scheduling, production management, simulation, and manufacturing systems. He has published papers in *IIE Transactions*, *Decision Sciences*, *Simulation*, *International Journal of Production Research*, *International Journal of Flexible Manufacturing Systems*, *International Journal of Computer Integrated Manufacturing*, *European Journal of Operational Research*, *Production Planning and Control*, *Journal of the Operational Research Society*, *Computers and Operations Research*, *Computers and Industrial Engineering*, *OMEGA*, *Journal of Intelligent Manufacturing, and International Journal of Production Economics*. He is on the Editorial board of *International Journal of Operations and Quantitative Management* and also the *Journal of Operations Management*. He is an associate member of Institute of Industrial Engineering, Institute of Simulation, and Institute of Operations Research and Management Science.