Proactive Release Procedures for Just-in-Time Job Shop Environments, Subject to Machine Failures

Ramesh Bollapragada,[1,2,]* Norman M. Sadeh[3]

[1] *College of Business, San Francisco State University, San Francisco, California 94132*

[2] *Bell Laboratories, Lucent Technologies, Holmdel, New Jersey 07733*

[3] *The School of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213*

Received 4 April 2001; revised 25 April 2004; accepted 18 May 2004
DOI 10.1002/nav.20036
Published online 16 August 2004 in Wiley InterScience (www.interscience.wiley.com).

Abstract: In this paper, we consider just-in-time job shop environments (job shop problems with an objective of minimizing the sum of tardiness and inventory costs), subject to uncertainty due to machine failures. We present techniques for proactive uncertainty management that exploit prior knowledge of uncertainty to build competitive release dates, whose execution improves performance. These techniques determine the release dates of different jobs based on measures of shop load, statistical data of machine failures, and repairs with a tradeoff between inventory and tardiness costs. Empirical results show that our methodology is very promising in comparison with simulated annealing and the best of 39 combinations of dispatch rules & release policies, under different frequencies of breakdowns. We observe that the performance of the proactive technique compared to the other two approaches improves in schedule quality (maximizing delivery performance while minimizing costs) with increase in frequency of breakdowns. The proactive technique presented here is also computationally less expensive than the other two approaches. © 2004 Wiley Periodicals, Inc. Naval Research Logistics 51: 1018–1044, 2004.

Keywords: job shop scheduling; proactive release policies; machine breakdowns; reoptimizing heuristics

1. INTRODUCTION AND LITERATURE REVIEW

Job shop scheduling problems have been shown to be NP-hard (Garey and Johnson [10], Lawrence, Lenstra, and Rinnooy [15]) for a number of objectives (e.g., makespan, minimizing sum of tardiness, and inventory costs). There is an extensive literature on job shop problems for varying objectives. The general practice in industry to address these problems has been to rely on priority dispatch rules (Baker [3], Morton and Pentico [18], Panwalkar and Iskander [24]).

* *Present address:* 451 Gateway Drive, Apt. 119, Pacifica, CA 94044
Correspondence to: R. Bollapragada (rbollapragada@yahoo.com)

Though computationally inexpensive, the schedules produced by these greedy rules are typically of poor quality. Mathematical programming techniques proposed for a job shop domain (Nemhauser and Wosley [21]), build high quality schedules, though often at intense computational expense. With the advent of powerful computers, simulated annealing (Van Laarhoven, Aarts, and Lenstra [34], Matsuo, Suh, and Sullivan [17]), improved variations exploiting tabu search principles (Glover and Laguna [12], Taillard [33]), and genetic algorithms (Della Croce, Tadei, and Volta [8], Holland [14]) have attracted the attention of a growing number of researchers. These procedures, while computationally expensive, can yield near-optimal solutions, if run for long enough. More comprehensive reviews of the job shop scheduling literature can be found in Baker [3], Pinedo [25], Morton and Pentico [18], and Applegate and Cook [2].

A significant amount of research in this area has focused on bottleneck based procedures. These include, the OPT factory scheduling system by Goldratt and Cox [13] and the shifting bottleneck algorithm by Adams, Balas, and Zawack [1]. While minimizing makespan has been the main objective in the above techniques, the sched-star scheduling module by Morton et al. [19], and the microboss scheduling system by Sadeh [27], address predictive job shop problems subject to tardiness and inventory costs.

In this paper, we address stochastic job shop problems, simultaneously subject to tardiness and inventory costs. This is compatible with the just-in-time objective of meeting customer demand in a timely, yet cost-effective manner. In particular, we focus on uncerainty due to machine failures. There are two general approaches to deal with uncertainty of this nature: (1) The notion of reactive scheduling, which involves revising or reoptimizing a schedule, after a contingency occurs; (2) the idea of proactive scheduling, which constructs schedules that account up-front for statistical knowledge of uncertainty. Ideally, the above two approaches could be combined to deal with contingencies such as the above.

The sheer size of most manufacturing scheduling problems generally precludes the construction of a brand new schedule each time an unanticipated event, such as a machine breakdown occurs. For this reason, schedules have traditionally been used to provide overall tactical guidance, while simple control rules are relied upon at the operational level to decide which operation to perform next on a given machine. This approach is typical of the scheduling engines in most MRP/MRP-II systems (Orlicky [22]) where schedules are generated to account for contingencies like machine failures through standard safety leadtimes, while local decision rules (FIFO, EDD, etc.) help control schedule execution. In contrast, it is shown (Bean et al. [5], Baker [3], Pinedo [25], Smith et al. [32]) that reoptimizing when contingencies occur from a more global perspective than would be allowed by local dispatch mechanisms, can result in significant improvements. The paper on match-up scheduling (Bean et al. [5]) addresses a multiresource scheduling problem with disruptions. An overall strategy to reconstruct the schedule, to match up with the preschedule at some future time is adopted in this work. The domain and the objective are different from that considered in this paper. A knowledge based methodology for reactively revising schedules in response to unexpected events is presented by Sadeh, Otsuka, and Schnelbach [30] and Smith et al. [32]. This approach is based on recognition of constraint conflicts in the schedule, and identifying the most appropriate reaction in a given situation. These and the other approaches in the literature follow a reactive scheduling strategy. The work by Glassey and Resende [11] attempts to consider the knowledge of uncertainty, while releasing jobs to the shop floor based on a control mechanism (starvation avoidance) that adapts reorder point concepts of inventory management. Lee, Piramuthu, and Tsai [16] use machine learning and genetic algorithms to integrate release, sequencing, and control decisions. The objective considered in these studies is different from the one dealt with in this paper. Thus, the problem considered here has not been addressed in the literature before.

In this work, we present a novel methodology, namely, proactive uncertainty management, which is built on the ideas of both safety leadtimes and reactive scheduling, to determine the release and sequencing decisions in a stochastic job shop environment. In a way, this methodology attempts to integrate release, sequencing, and control decisions.

In a job shop domain subject to machine failures, the release dates used for execution play an important role in determining the quality of the solutions obtained. The traditional approach followed to obtain good release dates (that are used for releasing jobs during schedule execution) has been through building near-optimal schedules, based on average durations of the operations. Instead, we show in this work that good initial schedules may not be sufficient to obtain the best release dates. The proactive uncertainty management techniques presented here attempt to obtain near-optimal release dates, by building initial schedules that consider the statistical data of uncertainty.

The main contribution of this work is to show that the proactive procedures presented here build better release dates, compared to those obtained by techniques that produce near-optimal initial schedules (these initial schedules are built based on average duration of the operations). This is empirically validated, by comparing the proactive technique with solutions derived from multiple runs of simulated annealing. The simulated annealing procedure implemented here is similar to the one described in Sadeh and Nakakuki [29]. The release dates from the initial schedules obtained by these two approaches are executed in simulation by a common control policy. We show that though simulated annealing builds better initial schedules compared to the proactive technique (the performance metric being average schedule cost), the proactive technique does better than simulated annealing after schedule execution (the performance metric being average cost of executed schedules, evaluated through simulation). Experimentation presented at the end of this paper also shows that the release dates built by simulated annealing (SA), when executed with SA policy (i.e., at each breakdown, the remaining operations are rescheduled with simulated annealing procedure) only marginally outperform the proactive technique (the release dates from the initial schedules built by the proactive technique are executed with a proactive based control policy).

Some other key contributions of this research include: (1) Illustration from a thorough experimentation that the proactive technique clearly outperforms the solutions obtained from the best of 39 combinations of dispatch rules and release policies; (2) empirically show that the solutions obtained by the proactive technique are less expensive computationally, compared to simulated annealing and the best of 39 combinations of dispatch rules & release policies; and (3) the presentation of a novel modeling procedure for handling uncertainty due to machine failures (through this modeling, we provide a solution procedure for addressing a stochastic single machine early/tardy problem with piecewise linear penalties).

The rest of the paper is organized as follows. Section 2 provides a formal definition of the job shop problem considered in this study. Section 3 and 4 describe the proactive uncertainty management approach, in detail. Experimental design is discussed in Section 5, while Section 6 presents a detailed discussion of computational results.

2. THE STOCHASTIC JOB SHOP SCHEDULING PROBLEM WITH EARLINESS AND TARDINESS

The model considered here is similar to the one proposed by Sadeh [27, 28]. We envision an environment where a finite set of jobs, $J = \{j_1, j_2, \cdots, j_n\}$, have to be scheduled on a finite set of resources, $R = \{R_1, R_2, \cdots, R_m\}$. The jobs are assumed to be known ahead of time. The resources (the term resource and machine are used interchangeably in our paper) are subject to random breakdowns and hence may not be available over the entire scheduling horizon. Each job j_l requires performing a set of manufacturing operations $O_l = \{O_{1l}, O_{2l}, \cdots, O_{n_l l}\}$ and, ideally, should be completed by a specific due date, dd_l, for delivery to the customer. Precedence constraints specify a complete order in which operations in each job have to be performed. By convention, we assume that operation O_{il} has to be completed before operation $O_{(i+1)l}$ can start ($i = 1, 2, \cdots, n_l - 1$).

Each job j_l has an earliest acceptable release date, erd_l, before which it cannot start, e.g., because the raw materials or components required by this job cannot be delivered before that date. Each job also has a latest acceptable completion date (deadline), lcd_l, by which it should absolutely be completed, e.g., because the customer would otherwise refuse delivery of the order. For each job, we assume that $erd_l + processingtime \leq dd_l \leq lcd_l$. The lcd_l, is assumed to be *very large* to avoid the possibility of infeasible schedules (in our experimentation, we have not encountered a single infeasible schedule).

We consider here, problems in which each operation O_{il} requires a single resource $R_{il} \in R$ and the order in which a job visits different resources varies from one job to another. In addition, we assume that each operation of the same job is processed on a different resource. Each resource can only process one operation at a time and is nonpreemptable. The duration du_{il} of each operation O_{il} is assumed to be known.

The problem requires finding release dates which when used to process all the operations minimizes the expected sum of tardiness and inventory costs over all the jobs.

Specifically, each job j_l incurs a positive marginal tardiness cost $tard_l$ for each unit of time that it finishes past its due date dd_l. Marginal tardiness costs generally correspond to tardiness penalties, interests on lost profits, loss of customer goodwill, etc. The total tardiness cost of job j_l, in a given schedule, is measured as:

$$TARD_l = tard_l \cdot \text{MAX}(0, C_l - dd_l),$$

where C_l is the completion date of job j_l (this corresponds to the completion of last operation of this job).

Inventory costs on the other hand can be introduced at the level of any operation in a job. In our model, each operation O_{il} can have its own nonnegative marginal inventory cost, in_{il}. This is the marginal cost that is incurred for each unit of time that spans between the start time of this operation and either the completion date of the job or its due date, whichever is larger. In other words, the total inventory cost introduced by an operation O_{il} in a given schedule is

$$INV_{il} = in_{il} \cdot (\text{MAX}(C_l, dd_l) - st_{il}),$$

where st_{il} is the start time of operation O_{il}. Typically, the first operation in a job introduces marginal inventory costs that correspond to interests on the costs of raw materials, interests on processing costs (for that first operation), and marginal holding costs. Following operations introduce additional inventory costs such as interests on processing costs, interests on the costs of additional raw materials or components required by these operations, etc.

The average cost across all possible executions of the release dates is:

$$E\left(\sum_{j_l \in J} TARD_l + \sum_{j_l \in J} \sum_{i=1}^{n_l} INV_{il}\right).$$

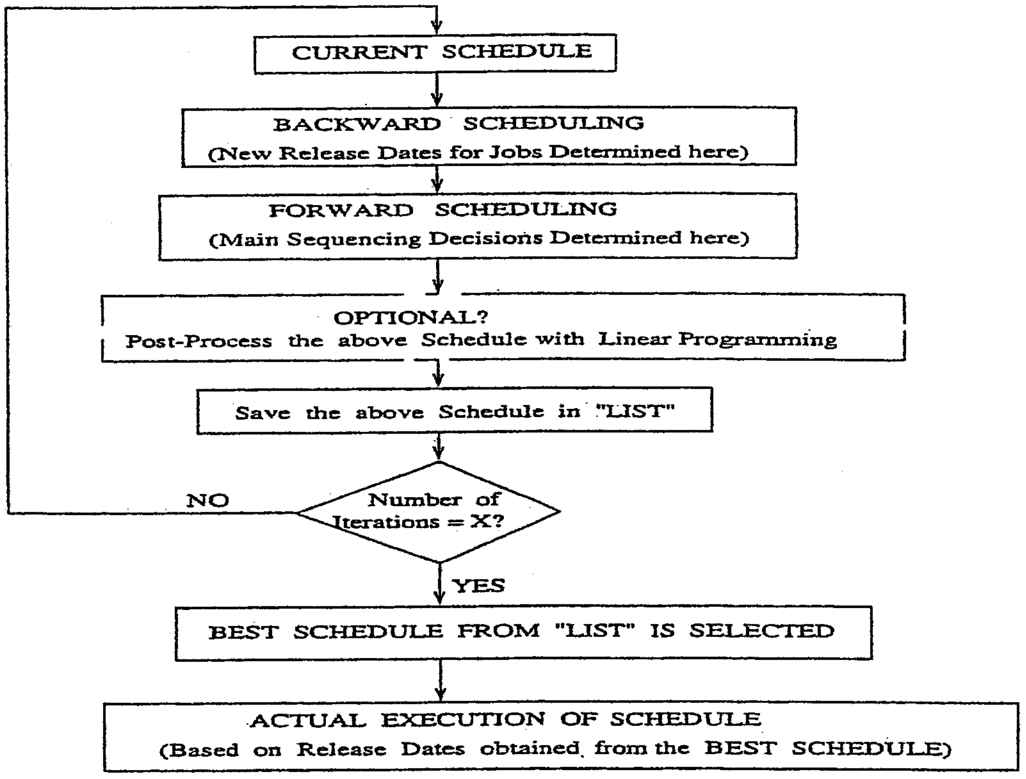
CURRENT SCHEDULE
BACKWARD SCHEDULING
(New Release Dates for Jobs Determined here)
FORWARD SCHEDULING
(Main Sequencing Decisions Determined here)
OPTIONAL?
Post-Process the above Schedule with Linear Programming
Save the above Schedule in "LIST"
NO
Number of Iterations = X?
YES
BEST SCHEDULE FROM "LIST" IS SELECTED
ACTUAL EXECUTION OF SCHEDULE
(Based on Release Dates obtained from the BEST SCHEDULE)

Figure 1. Proactive technique.

The main goal in this paper has been to compare the goodness of the release dates obtained by the techniques presented for proactive uncertainty management, with (1) simulated annealing and (2) best of 39 combinations of dispatch rules and release policies. The initial release dates generated by the above three approaches are executed in simulation by a common control policy. The comparison between the different approaches is based on the average cost of the executed schedules, evaluated through simulation.

The next section describes the proactive technique in detail.

3. PROACTIVE TECHNIQUE

The approach focuses on building near-optimal release dates, with an attempt to result in better executed schedules. The procedure presented here attempts to determine the release decisions for different operations of each job based on measures of shop load, statistical data of machine failures, and repairs with a tradeoff between inventory and tardiness costs.

- The main decisions to be made are release and sequencing, while incorporating the statistical data of uncertainty. These are determined by the *backward* and *forward* scheduling modules described later in this section. The overall procedure, as described in flow chart of Figure 1, consists in postprocessing an original schedule (obtained from a simple dispatch rule such as WSPT), in an attempt to build good release dates that are used in schedule execution. The "current schedule" in first iteration of the flow chart refers to the schedule generated by the dispatch rule. It briefly consists of the following steps:

1. Build initial release dates by applying a backward scheduling procedure. The jobs are then released to the shop at these times.
2. Apply a forward scheduling procedure to determine the sequencing decisions (note that the start time of the first operation of each job may not necessarily correspond to the release date obtained from the backward scheduling procedure, as operations of two different jobs competing for the same resource might have windows of overlap). The procedure here attempts to consider the statistical data of machine failures and repair, while building the schedule.
3. If the "resulting schedule" from (2) is better than the schedules obtained so far, it is saved in a *LIST.* (We also present results in our computational section where this schedule was postprocessed with linear programming to obtain optimal release dates, for this sequence of operations. The problem of inserting idle time optimally in a schedule, given completely specified sequences of operations on each machine, can be formulated as a linear programming problem. Refer to Sadeh and Nakakuki [29] for more details on this formulation.)
4. The procedure in steps 1 and 2 is repeated, with the above "resulting schedule" as "current schedule." Applying the backward and forward scheduling procedures iteratively, on different schedules, is a diversification strategy. A good value for the "number of iterations" X, was determined empirically, to be 10. The tradeoff here is between computational expense and the quality of the schedules produced. As the value of "X" is increased, the quality of solution tends to improve, but at the cost of more CPU time. Refer to the experimental section for a cost analysis based on different values of "X."
5. The release dates from the beset of the schedules (based on the average schedule cost) in the stored *LIST* are used in actual schedule execution, when machine breakdowns are simulated.

- A reoptimizing procedure described in Section 4 determines the sequencing decisions in the forward scheduling module of the above procedure. This involves solving a stochastic single machine early/tardy problem with piecewise linear penalties when a decision of sequencing from a queue of operations has to be made. (We show, in our cost analysis, that the total penalty cost, which includes both inventory and tardiness costs, has this behavior. In our single machine model, we allow different inventory holding and tardiness *rates* for different operations of the same job. A detailed discussion on the computation of these holding and tardiness rates is provided in Section 4. Also, note that these are *different* from the inventory and tardiness *penalties* discussed in Section 2. The penalties discussed in Section 2 are used in judging the quality of the final schedule obtained.)

The *backward scheduling procedure* involves sorting the jobs in a given schedule according to their contribution to the tardiness cost in the total cost function. As this procedure is applied iteratively (see Fig. 1) on different schedules, it helps shift late jobs in one schedule to be early in another. The job that contributes most to the tardiness in the total cost function is scheduled around its due date. In other words, the operations of this job are scheduled just-in-time. The procedure is then repeated for the other jobs in the sorted list. In the event of conflicts due to overlapping with earlier reservations, the operations are scheduled at the latest time compatible with the due date or the earliest possible completion time. The start time of the first operation of each job in the resulting feasible schedule determines the job's new release date. (We tried other procedures to obtain the release dates. Interested readers are referred to Bollapragada [7] for a detailed comparison between these different procedures.)

The *forward scheduling* module then determines the sequencing decisions for the operations of each job. The main procedure here involves solving a stochastic one machine early/tardy problem with piecewise linear penalties at each scheduling decision instant. Literature on various versions of a single machine early-tardy problems, (deterministic as well as stochastic), can be found in Baker and Scudder [4], Mittenthal and Raghavachari [20], and Pinedo [26]. To our knowledge, a early/tardy problem with piecewise linear penalties has not been addressed before.

The reoptimizing procedure that solves a stochastic single machine early/tardy problem to determine the sequencing decisions in the forward scheduling module is presented in detail in the following section.

4. REOPTIMIZING PROCEDURE

The reoptimizing heuristics presented in this section are priority-based rules, which use cost analysis from a global level, to select an operation to be scheduled next. We take a snapshot of the queue of operations in front of a resource at a certain time instant t. We then formulate a stochastic one machine early/tardy model for these operations and derive lower and upper bounds on the expected cost function. A number of new heuristics are then presented. The best of these policies that produced solutions close to the lower bound was used to determine the sequencing of these operations. (We observed that on an average, one of these heuristics consistently performed better than the others. This was based on the computational testing of different heuristics on a wide range of problem sets.) In our experimentation, we compared the performance of the proactive technique with other approaches (simulated annealing, etc.) based on this heuristic (a comparison of the performance of the proactive technique based on the other heuristics is also discussed in the computational section).

Consider a queue of operations in front of a resource at a certain time instant t. The stochasticity due to machine breakdowns is modeled by assuming processing times to be random. Though this modeling appears a little naive, the structural properties obtained through such a procedure (as an example, the above modeling assumption was helpful in obtaining a closed form expression on the cost function) were helpful in developing a good heuristic. In our model, the processing times are normally distributed random variables, whose distribution is obtained from the knowledge of deterministic processing time and the statistical data of time to repair (TTR) and time between failure (TBF). Essentially, the mean of the distribution is obtained from the deterministic processing time, MTTR (mean time to repair) and MTBF (mean time between failure) information. The standard deviation of the distribution is obtained from the knowledge of second moments to TTR and TBF distributions. We assume that TTR and TBF are normally distributed, which has been justified previously by Blau [6] and Sarin, Erel, and Steiner [31].

The processing time of an operation in position i, $du_{[i]}$ is a random variable with marginal probability density function $f_{[i]}(\cdot)$. We assume that the processing times $du_{[i]}$, $i = 1, 2, \ldots, n$ are independent and normally distributed.

While performing the cost analysis to determine the sequencing decisions in the *forward scheduling* procedure, we model varying inventory and tardiness rates for different operations of the same job. These rates are *different* from the inventory and tardiness penalties discussed in Section 2 (note that the penalties described in Section 2 are used to compute the actual cost of a schedule).

The intuition for associating varying inventory and tardiness rates is based on the fact that the merits of different possible start times can dramatically change as the schedule is being constructed. It is thus critical for the scheduler to constantly keep track of these changes in order to provide a good compromise between the cost incurred by the various operations that are competing. The tradeoff in these costs can be indirectly captured in the tardiness and inventory rates for each operation, as the schedule is constructed. The method we used to revise these inventory and tardiness rates was originally proposed and discussed in detail by Sadeh [27]. Operation i of job j is used synonymously with previous definition of operation O_{ij} of job j_l, here and in the remainder of the paper, where appropriate.

We define the just-in-time start time of an operation i of job j, $jist_{ij}$, as the start time that minimizes the expected sum of holding and tardiness costs for this operation. It is the start time that minimizes the sum of inventory and tardiness costs of job j, if operation i were to be scheduled by itself on its resource and within the time windows specified by its predecessors and successors. The uncertainty due to breakdowns is captured here by considering average durations for the operations (i.e., only the mean time to repair and mean time between failure information is considered). A procedure to compute the just-in-time start times for different operations of each job is presented in the Appendix.

We define expected-min-cost start time for operation i of job j, $emcst_{ij}$, as the start time that represents the best tradeoff between inventory holding and tardiness rates for this operation, given the randomness due to machine failures (the uncertainty due to breakdowns is based on the time to repair and time between failure distributions). It is thus the start time that minimizes the expected sum of inventory and tardiness costs of job j, if operation i were to be scheduled by itself on its resource and within the time windows specified by predecessors and successors, while considering the statistical data of machine failure and repair.

We now define the tail leadtime of an operation as the empirical estimate of the leadtime for this operation and its successors to go through the shop [as an example, a method to compute the tail leadtime for operation i of job j (op_{ij}) is to sum the average processing time of op_{ij} and its successors]. The empirical method that performed the best in a job shop setting as defined in Morton and Pentico [18], was used to compute this measure. The just-in-time start times for op_{ij} are computed heuristically from the knowledge of tail leadtime for this operation, and the due date of this job.

The balance of this section is organized as follows:

- Illustration of piecewise linearity of the total penalty cost contribution from each operation (refer to Section A.1 of the Appendix)
- Derivation of the expected cost function for a stochastic single machine early/tardy model (refer to Section A.2 of the Appendix)
- Presentation of lower and upper bounding methods (refer to Section A.3 of the Appendix)
- Presentation of heuristics that determine the operation to be sequenced next from a queue

The next section presents a number of heuristics, whose goal is to determine the expected-min-cost start times, $emcst_{ij}$, for operations in the queue of a resource.

4.1. Reoptimizing Heuristics

We present four new heuristics (PBH1, PBH2, PBH3, and PBH4, where PBH stands for proactive based reoptimizing heuristic) that determine the sequencing decisions for operations waiting in front of a resource. These heuristics exploit the properties of the cost function derived in Section A.3 of the Appendix. The best of these heuristics that produced solutions close to the lower bound was used to determine the sequencing of operations at each reoptimizing instant. We present here, in detail, the reoptimizing heuristic that compared the best among the four. The results reported in the experimental section, provide a comparison of the proactive technique with PBH1 as the reoptimizing heuristic, against (1) simulated annealing and (2) best of 39 combinations of dispatch rules and release policies. Refer to the Appendix for a discussion on the other three heuristics.

PBH1: The basic idea in this heuristic is to minimize the expected early/tardy cost for a single machine problem. We formulate a mixed integer linear programming model, where the objective is to minimize the lower bound cost function described in Section A.3 of the Appendix. For simplicity, we refer to each operation by a single subscript. This is fine for our domain, as we assume a linear process plan. The inventory holding and tardiness rates for each operation, correspond to those defined in Section A.1 of the Appendix.

$$\text{Minimize} \sum_{i=1}^{i=n} (tard_i \tau_i + inv_i \epsilon_i)$$

subject to

$$ct_i \geq t + \overline{du_i}, \qquad i = 1, 2, \ldots, n, \tag{1}$$

$$ct_i - tard_i + inv_i = jist_i + \overline{du_i}, \qquad i = 1, 2, \ldots, n, \tag{2}$$

$$ct_j - ct_i + M(1 - x_{ij}) \geq \overline{du_j}, \qquad i < j = 1, 2, \ldots, n, \tag{3}$$

$$ct_i - ct_j + Mx_{ij} \geq \overline{du_i}, \qquad i < j = 1, 2, \ldots, n, \tag{4}$$

$ct_i, tard_i, inv_i \geq 0, \qquad i = 1, 2, \ldots, n, \qquad x_{ij} \in (0, 1),$

$i < j = 1, 2, \ldots, n, x_{ij} = 1$ if i precedes j, and 0 otherwise,

where

t = current time,
τ_i = tardness rate of operation i,
ϵ_i = inventory holding rate of operation i,
$\overline{du_i}$ = expected processing time of operation i,
$jist_i$ = just-in-time start time of operation i,
ct_i = expected completion time of operation i,
$tard_i$ = expected tardiness of operation i,
inv_i = expected earliness of operation i.

The expected processing time, $\overline{du_i}$, is obtained by assigning different probabilities for each realization of the processing time. This was estimated based on assigning (a) certain probability of realization for the deterministic duration, (b) certain probability of realization for the sum of deterministic duration and the mean time to repair, and (c) certain probability of realization based on the knowledge of deterministic duration and the standard deviation of the processing time, which is dependent on time to repair and time between failure distributions (the best fit probabilities of realization for cases a, b, c discussed here were determined empirically through extensive experimentation).

The first constraint ensures that the operation's completion time is at least as large as the expected processing time plus the current time. The right-hand side of constraint (2) specifies a local due date for each operation, and thus we define expected tardiness and expected earliness relative to this due date. The local due date of an operation is defined as the sum of just-in-time start time (a method to compute the just-in-time start times is presented in the Appendix) plus the expected duration of this operation. Through the definition of x_{ij} variables and M, we specify the rule for precedence relations among the operations. In our experiments, the value of M was set to be the maximum of the latest completion time of all jobs in the order book.

The reoptimizing heuristic thus solves an integer linear program for a single machine problem, whenever sequencing decisions have to be made at each machine. The number of jobs (operations) to be sequenced at a single machine at any time instant is thus a small subset of the total number of operations, and thus the reoptimizing heuristic is not computationally expensive.

5. PERFORMANCE EVALUATION

Experimental results with the proactive technique are reported for eights sets of scheduling problems. Each set consists of 10 problems, and is obtained by adjusting three parameters (see table 1) to cover a wide range of scheduling conditions. The three parameters are: an average due date parameter (tight versus loose average due date), a due date range parameter (narrow versus wide range of due dates), and a parameter controlling the number of major bottlenecks (one or two). The due date range parameter determines the variance in due date. The above data set generation was originally proposed by Sadeh [27, 29]. The time between failures is drawn from a Poisson distribution, while the time to repair is obtained from a normal distribution, with a small coefficient of variation. For details on the design of this test data set, refer to the Appendix. The scheduling problems were simulated with three sets of breakdowns (0%, 5%, 10%). The results reported for a set were over an average of 10 scheduling problems. Each problem requires scheduling 20 jobs on five resources for a total of 100 operations. Marginal tardiness costs in these problems were set to be on the average, five times larger than marginal inventory costs to model a situation where tardiness costs dominate but inventory costs are nonnegligible. (We also experimented with varying ratios of marginal tardiness to marginal inventory costs. In fact, we experimented with marginal tardiness to inventory ratios of 1, 3, 5, 10, 15 and have observed that our proactive heuristic is robust in its performance across all these ratios with an average error of only 2–3%.) The results presented in the computational section are only for a marginal tardiness to inventory ratio of 5. A factory simulator was implemented in C++ to simulate the breakdowns and thus model the execution of the schedules.

Our main experimentation here has been to compare the goodness of the release dates obtained by the proactive technique with the following two approaches:

1. Simulated annealing
2. Best of 39 combinations of dispatch rules and release policies

Table 1. Characteristics of the eight problem sets.

Problem set	Number of bottlenecks	Avg. due date	Due date range
1	1	Loose	Wide
2	1	Loose	Narrow
3	1	Tight	Wide
4	1	Tight	Narrow
5	2	Loose	Wide
6	2	Loose	Narrow
7	2	Tight	Wide
8	2	Tight	Narrow

The performance metric considered is the average cost of executed schedules evaluated through simulation and the average CPU time. We report performance based on other metrics also: average weighted tardiness, average weighted flowtime, and average weighted in-system time.

The release dates from the schedules generated by the above three approaches are executed by an R&M dispatch rule for 5% and 10% frequency of breakdowns. (We also experimented with a number of other dispatch rules, WCOVERT, WSPT, EDD, SRPT, ATC, for schedule execution and the results based on them can be obtained from Bollapragada [7]. The performance of the above three approaches by taking the best of executed schedules from all of the dispatch control policies is also presented there. We observed that the results obtained has similar trend to the one reported with the R&M control policy.) R&M is a slack based heuristic, which has compared well against other priority dispatch rules in the literature. Refer to Morton and Pentico [18] for more details on this heuristic.

We also present results where the release dates from schedules generated by the above three approaches are executed by their respective policies. (For example, the release dates from the initial schedule built by simulated annealing (SA) are executed by an SA policy. This involved rescheduling the remaining unscheduled operations at each breakdown, with multiple runs of simulated annealing.)

As described earlier, the problem of optimally inserting idle time in an existing job shop schedule (i.e., given completely defined operation sequences on each resource) can be formulated as a linear program. We also present results where the schedules generated by the three approaches we are comparing are postprocessed by the linear program, before being executed.

We describe next the different procedures used for comparison against the proactive technique.

5.1. Dispatch Rules & Release Policies

We considered various combinations of dispatch and release policies that are promising in the domain we are addressing. The priority dispatch rules include WSPT, EDD, slack per remaining processing time (SRPT), weighted covert (WCOVERT), and apparent tardiness cost rule (ATC). The two release policies used are: (a) an immediate release policy (IM-REL) that allows each job to be released immediately and (b) a parametric release policy termed AQT (average queue time release policy) that computes a job's release date by offsetting the job's due date by the sum of its total duration and its estimated queueing time. An exponential version of the parametric early/tardy dispatch rule termed as EXP-ET (see Morton and Pentico [18]) is yet another dispatch rule used in our study. This has an intrinsic release policy that allows job to be released when the priority for the job becomes positive. The 39 combinations are thus as follows:

EXP-ET and its intrinsic release policy (times four parameter settings), EXP-ET/IM-REL (times four parameter settings), EDD/AQT (times four parameter settings), EDD/IM-REL, WSPT/AQT (times four parameter settings), WSPT/IM-REL, SRPT/AQT (times four parameter settings), SRPT/IM-REL, WCOVERT/IM-REL (times four parameter settings), WCOVERT/AQT (times four parameter settings), ATC/IM-REL (times four parameter settings), and ATC/AQT (times four parameter settings).

On each problem, the best of the 39 schedules produced by these combinations was recorded for comparison against the proactive technique. These same 39 combinations of dispatch rules and release policies were used before for comparison against the microboss scheduling system (Sadeh [27]) and simulated annealing procedures (Sadeh and Nakakuki [29]).

5.2. Simulated Annealing

The simulated annealing procedure implemented here is similar to the one described in Sadeh and Nakakuki [29]. The simulated annealing algorithm used for comparison against the proactive the technique is also described in detail in Sadeh and Nakakuki [29].

The SA procedure was run six times on each problem. For each problem, we recorded the best solution it found over these runs for comparisons with the proactive technique. In each run, the initial temperature, T_0, was set to 700, temperature T_1 was 6.25, and the cooling rate α was 0.85. The value of β was selected such that all precedence constraints were satisfied at the end of each run. The procedure was run on an average for 6000 s on each experiment.

The next section describes the empirical evaluation of the different techniques considered in this paper.

6. DISCUSSION AND COMPUTATIONAL RESULTS

We report the performance of our technique on the basis of average expected costs (tardiness penalties, work in process, and finished goods inventory levels) and average CPU time (for a definition of the notation used below, refer to Section 2).

- *Average schedule cost:* $\mathrm{E}(\sum_{l \in J} TARD_l + \sum_{l \in J} \sum_{i=1}^{n_l} INV_{il})$

Additionally, performance with respect to the following criteria was also measured:

- *Average weighted tardiness:* $\mathrm{E}\left(\dfrac{\sum_l tard_l \cdot \mathrm{MAX}(0, C_l - dd_l)}{\sum_l tard_l}\right)$
- *Average weighted flowtime:* $\mathrm{E}\left(\dfrac{\sum_l inv_{1l} \cdot (C_l - st_{1l})}{\sum_l inv_{1l}}\right)$

The “average weighted flowtime” measure corresponds to the average in-process or work-in-process inventory of the schedule:

- *Average weighted in-system time:* $\mathrm{E}\left(\dfrac{\sum_l inv_{1l} \cdot [\mathrm{MAX}(dd_l, C_l) - st_{1l}]}{\sum_l inv_{1l}}\right)$

The “average weighted in-system time” measure accounts for both job flowtime (i.e., in-process inventory) and job earliness (i.e., finished goods inventory). It corresponds to the average weighted time spent by a job in the system. It is thus a measure of the total inventory in the system.

The experiments were conducted on a SparcStation 5. The proactive technique is compared against the best of 39 combinations of dispatch & release policies and the best of simulated annealing runs. Graphically, we report the experimental results for an average over 80 scheduling problems (8 sets of 10 problems each). The experimental runs for each individual problem, set on all the above objectives and for different frequencies of breakdowns, can be obtained in detail from Bollapragada [7]. The initial schedule with the proactive technique is obtained by postprocessing a deterministic schedule from a standard dispatch rule with immediate release policy. The dispatch rule used in our experiments was WSPT. The release dates from the initial schedules generated by the proactive technique, multiple runs of simulated annealing (SA), and 39 combinations of dispatch rules & release policies were executed through an R&M dispatch policy, when the frequency of breakdowns was 5% and 10%. The results reported were over an average of 10 simulation runs, for each of the above 80 experiments. (On each of these 10 simulation runs, we used a different random seed and thus simulated machine breakdowns at varying time instances. At the same time, we made sure to use the same seed while comparing the three different approaches on an identical simulation run, for each of the experiments.)

The simulated annealing procedure was run for a total of 6000 s (this corresponds to six runs of the SA procedure) on each experiment. The CPU time reported for the "best of 39 dispatch & release policies" is the sum of the CPU time taken by all 39 combinations of dispatch rules & release policies. As the performance in schedule quality was reported by considering the best of these policies in each experiment, we felt it reasonable to report the total CPU time taken. We also note that 19 of these 39 combinations performed best on at least one of the 80 problems considered in this study. These 19 combinations included all dispatch rules and all 3 release policies.

One of the main results of this paper is presented in Figures 3 and 4. A comparison between the three different approaches, when the machines are not available on an average for 5% and 10% of the time, is presented in Figure 3. The schedules compared here are built based on average durations for the operations (a method to compute the average duration of an operation is described in the Section 4.1 on reoptimizing heuristic, PBH1). We draw the following inferences from these experiments:

- When the machines are not available on an average for 5% of the time, the simulated annealing procedure reduced average schedule cost over 80 experiments by 5.144% compared to proactive technique with reoptimizing heuristic, PBH1.
- When the machines are not available on an average for 10% of the time, the simulated annealing procedure reduced average schedule cost over 80 experiments by 3.88% compared to proactive technique with reoptimizing heuristic, PBH1.

The *release dates* from the above schedules are executed by a common control policy, with breakdowns simulated. The time between failures is drawn from a Poisson distribution, while the time to repair is randomly drawn from a normal distribution, with a small coefficient of variation (for more details on the generation of the MTTR and MTBF data, refer to the Appendix). The original durations of the operations are used while executing these release dates. Figure 4 compares the performance of the different approaches, where a total of 10 simulations are run on each experiment.

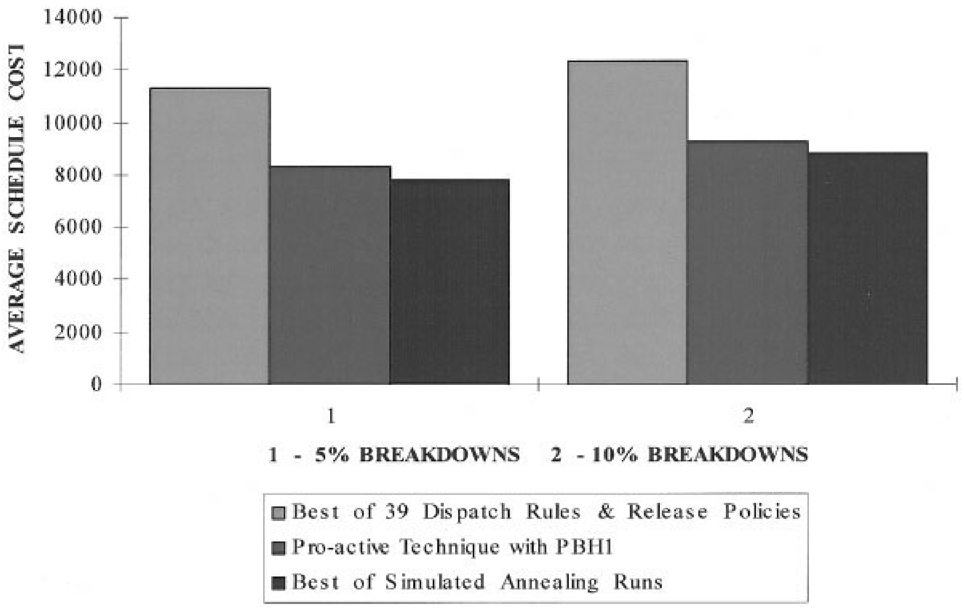

Figure 3. Average schedule cost before execution.

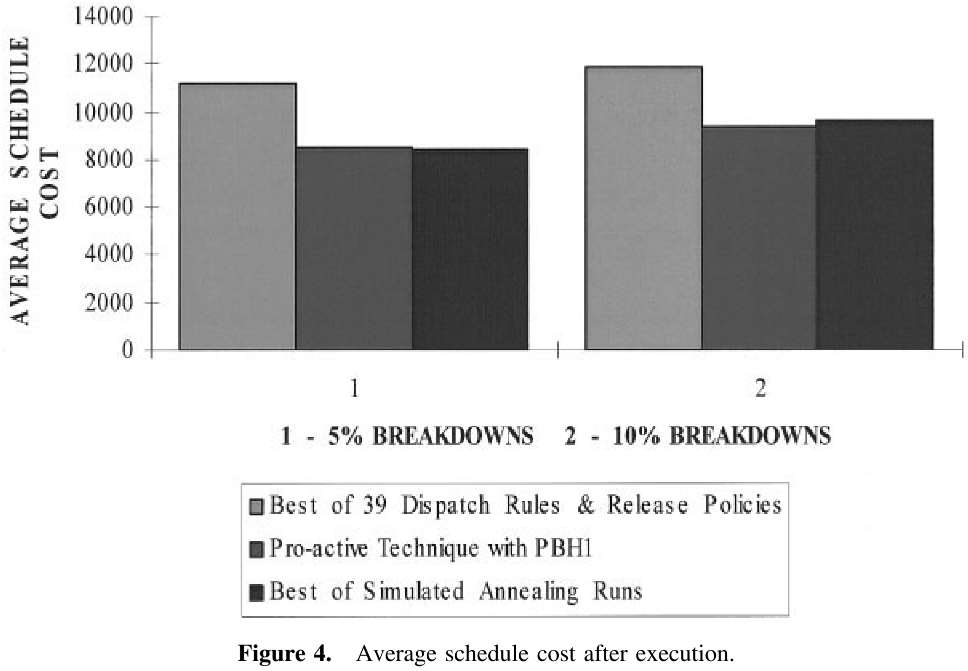

Figure 4. Average schedule cost after execution.

- 5% Breakdowns

—The simulated annealing (SA) procedure reduced the average cost of executed schedules over 80 experiments by 0.42% compared to proactive technique with PBH1.

- 10% Breakdowns

—The proactive technique with PBH1 reduced the average cost of executed schedules over 80 experiments by 2.49% compared to the simulated annealing procedure.

The performance of the proactive technique with the best of PBH1 through PBH4 reoptimizing heuristics was even better. On 5% and 10% breakdowns, it compared 1.68% and 5.27% better than the SA procedure, respectively. We thus show that the proactive procedures presented here build better release dates compared to those obtained from schedules that are believed to be near-optimal. Note that the simulated annealing is computationally very expensive compared to the proactive technique (refer to the comparison table on the computational times in the next few pages).

We did experimentation on a smaller set of problems, where the release dates from the initial schedules (based on average durations), built by multiple runs of simulated annealing (SA) and proactive technique with PBH1, are executed by their respective scheduling policies. Essentially, the schedule built by simulated annealing procedure was repaired by an SA policy, i.e., at each breakdown, the operations still remaining to be scheduled are rescheduled with simulated annealing procedure (this involved running simulated annealing six times whenever a contingency occurred), while *reoptimizing heuristic PBH1* was used as a control policy while executing the release dates from the initial schedule built by the proactive technique. A total of 24 experiments were conducted, with three experiments from each of the eight sets described earlier. In terms of average cost of executed schedules over 24 experiments, the SA procedure compared *only 2.85%* better than the proactive technique, when the frequency of breakdowns was 5%. It compared *only 1.21%* better than the proactive technique, when the frequency of breakdowns was 10%. The simulated annealing took several hours while the computational expense of the proactive technique was within minutes.

The remaining results are presented in detail in the below tables. We report the inferences based on an average over 80 experiments for the different performance metrics. A total of 10 simulations were performed on each experiment. A more detailed analysis based on the individual eight problem sets for all these measures is discussed extensively in Bollapragada [7]. Note that the results reported in these figures are *without the idle time optimization* (the results with "idle time optimization" are presented later in this section). The following conclusions can be drawn through these experiments, where linear program optimization was not used.

The results in Table 2 can be interpreted as follows. Specifically, we illustrate it in the context of two performance measures.

- 0% Breakdowns

—The proactive approach with PBH1 reduced the average cost of executed schedules over 80 experiments by 27.41% compared to the best of 39 dispatch & release policies, while the simulated annealing procedure was better than this version of proactive approach by 12.6%.

Table 2. Performance evaluation.

Performance metric	Comparison of "proactive technique with PBH1," against	Frequency of breakdowns		
		0%	5%	10%
Average cost of executed schedules	Simulated annealing	12.6%	−0.42%	2.48%
	Best of 39 dispatch rules and release policies	27.41%	29.79%	26.78%
Average weighted tardiness	Simulated annealing	−7.1%	6.07%	6.35%
	Best of 39 dispatch rules and release policies	4.4%	9.26%	14.73%
Average weighted flowtime	Simulated annealing	−9.66%	13.08%	15.94%
	Best of 39 dispatch rules and release policies	58.78%	82.13%	80.33%
Average weighted-in-system time	Simulated annealing	−17.68%	−7.67%	−8.10%
	Best of 39 dispatch rules and release policies	40.11%	46.72%	42.51%

—In terms of average weighted tardiness, the proactive approach with PBH1 was better than the best of 39 dispatch & release policies by 4.4%, while the SA procedure reduced this measure by 7.1% in comparison with this version of proactive approach.

• 5% Breakdowns

—The proactive approach with PBH1 reduced average cost of executed schedules over 80 experiments by 29.79% compared to the best of 39 dispatch & release policies, while the simulated annealing procedure was better than this version of the proactive approach by 0.42%.
—In terms of average weighted tardiness, the proactive approach with PBH1 was better than the best of 39 dispatch & release policies by 9.26%, while it reduced this measure by 6.07% in comparison with the SA procedure.

The next table (Table 3) is analogous to the previous one (Table 2), except that it presents the

Table 3. Performance evaluation.

Performance metric	Comparison of "proactive technique with the best of PBH1 through PBH4," against	Frequency of breakdowns		
		0%	5%	10%
Average cost of executed schedules	Simulated annealing	−10.55%	1.68%	5.27%
	Best of 39 dispatch rules and release policies	29.70%	32.54%	30.25%
Average weighted tardiness	Simulated annealing	−2.89%	11.14%	13.63%
	Best of 39 dispatch rules and release policies	8.65%	14.51%	19.06%
Average weighted flowtime	Simulated annealing	−8.74%	0.79%	6.59%
	Best of 39 dispatch rules and release policies	60.14%	62.33%	65.78%
Average weighted-in-system time	Simulated annealing	−16.83%	−13.59%	−10.56%
	Best of 39 dispatch rules and release policies	41.08%	39.03%	39.29%

Table 4. Performance evaluation.

Performance metric	Comparison of the different approaches	Frequency of breakdowns		
		0%	5%	10%
Average cost of executed schedules (in units of mesaure)	Simulated annealing	6842	8459	9599
	Proactive technique with best of PBH1 through PBH4	7564	8247	9005
	Proactive technique with PBH1	7704	8495	9367
	Best of 39 dispatch rules and release policies	9816	11026	11875
Average CPU time (in seconds)	Simulated annealing	6000	6000	6000
	Proactive technique with best of PBH1 through PBH4	44.23	47.97	58.12
	Proactive technique with PBH1	10.19	10.52	12.34
	Best of 39 dispatch rules and release policies	15.21	15.60	17.16

results obtained from the “Proactive technique with the best of PBH1 through PBH4.” Table 4 provides a comparison of the of the different approaches in terms of actual units of measure. The performance measures discussed here are “Schedule Cost” and “Computational Time.” The reported results for the “best of 39 dispatch rules & release policies” in all the above tables were based on the executed schedule with the best schedule quality (average schedule cost) over all these 39 combinations. Each combination involved building an original schedule based on average durations (with their respective release policy & dispatch rule) and executing the resulting schedule with R&M control policy.

The comparison with the “best of 39 dispatch rules & release policies” illustrated the robust performance of the proactive technique against heuristics that have proven to perform well for different objectives. The comparison with simulated annealing has shown that the proactive technique is capable of producing near-optimal solutions.

We intended to experiment with other methods such as match-up scheduling (Bean et al. [5]) and the shifting bottleneck procedure (Adams, Balas, and Zawack [1]); but the comparison would involve modifying those procedures to tailor to our problem, and hence we deemed it inappropriate.

We next provide a comparison of the various versions of proactive technique (the metric compared is the “average schedule cost over 80 experiments”). The results presented here are for 0% frequency of breakdowns, but similar trends have been observed for 5% and 10% breakdowns. The proactive technique with PBH1 outperformed the others in schedule quality. It compared 1.29% better than proactive technique with PBH2, 4.11% better than proactive technique with PBH3, and 9.27% better than proactive technique with PBH4.

By selecting a specific reoptimizing heuristic, PBH1, we experimented with various versions of the proactive technique. This involved obtaining the release dates before invoking the *forward scheduling module,* by a variety of techniques. Essentially, we experimented with different backward scheduling procedures compared to that proposed in Section 3. (Specifically, the release dates are obtained by a “bottleneck centered approach” and a “conflict propagation based procedure.” Interested readers are referred to Bollapragada [7] for a detailed description of these two approaches. These two approaches are compared with the proactive technique outlined in Section 3, on the 80 experiments discussed before. Experimental results on the average schedule cost metric for different frequencies of breakdowns indicate that the proactive approach with reoptimizing heuristic PBH1 compared 14.46% and 6.87% better than these two

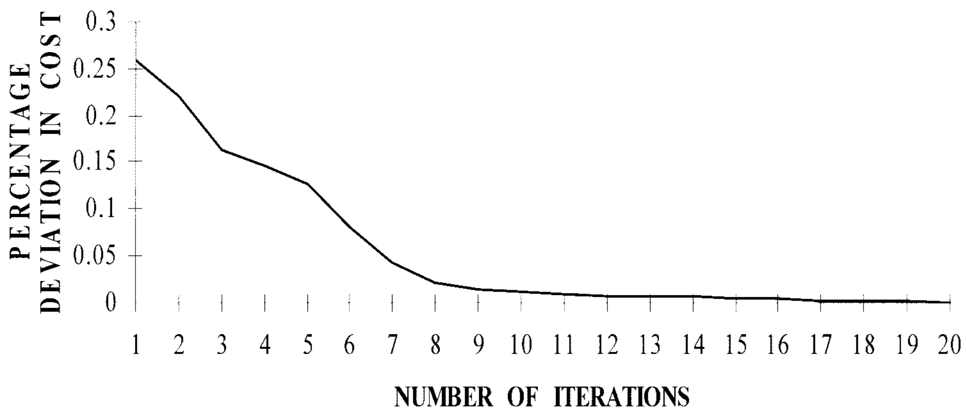

Figure 5. Changes of costs with varying iterations in proactive technique with PBH1 (average over 80 experiments).

approaches.) This indicates that release dates, and hence the procedure to obtain the start time of jobs, play an important role in determining the quality of an executed schedule.

As stated in Section 3, the number of iterations "*X*" in the proactive technique, were empirically determined to be 10. This tradeoff here is between schedule quality and computational expense. We present in Figure 5 the trend in average cost, with a varying number of iterations. This was based on an average over 80 experiments for 0% breakdowns. The cost at each iteration is normalized with reference to the cost at 20th iteration. We infer that the extra payoff in schedule cost after 10 iterations is not worth the additional CPU time.

We also compared the performance of simulated annealing, best of 39 dispatch & release policies, and the "proactive technique with PBH1" on the standard 80 problems, with *idle time optimization.* The problem of inserting idle time optimally in a schedule, given completely specified sequences of operations on each machine, can be formulated as a linear programming problem (refer to Sadeh and Nakakuki [29]). The idle time insertion was performed on the schedules obtained by each of the above techniques, before actual execution of their respective release dates (refer to Fig. 1). This is done in two different ways: (a) considering expected durations for the operations in the LP formulation and (b) sampling the durations for the operations before solving the linear program. In the discussion to follow, we refer to these two procedures as (a) and (b) (more discussion on this is referred to Section A.4 of the Appendix). We draw the following conclusions, through our experimentation on the average schedule cost metric over 80 experiments, for different frequencies of breakdowns.

- The idle time optimization procedure (a), applied to the best of 39 dispatch & release policies improved the performance (in comparison with no idle time insertion) by approximately 4%, while the idle time optimization procedure (b), compared 5% better.
- The idle time optimization procedure (a), applied to the simulated annealing procedure improved the performance (in comparison with no idle time insertion) by 0.9%, while the idle time optimization procedure (b), compared 1.1% better.
- The idle time optimization procedure (a), applied to the proactive approach with reoptimizing heuristic PBH1, improved the performance (in comparison with no idle time insertion) by 1.2%, while the idle time optimization procedure (b), compared 1.7% better.

7. SUMMARY DISCUSSION

In this paper, we address stochastic job shop problems, simultaneously subject to both inventory and tardiness costs. The uncertainty is due to machine failures. We present proactive techniques that focus on building competitive release dates, to result in better executed schedules. These techniques exploit the ideas of both safety leadtime and reactive scheduling, and attempt to integrate release, sequencing, and control decisions. We show that, in terms of average cost of executed schedules (evaluated through simulation), the proactive technique (1) clearly outperforms the best of 39 combinations of dispatch rules and release policies, (2) compares better than the simulated annealing procedure for higher frequencies of breakdowns, (3) its performance compared to other techniques improves with increase in frequency of breakdowns, and (4) is computationally less expensive than the other two approaches.

The current work presents the empirical evaluation of the techniques for proactive uncertainty management when the stochasticity is due to machine failures. The flexibility in our modeling (incorporating machine failure and repair information through the processing times of the jobs) allows us to extend this methodology to handle other types of uncertainty such as random yield in this environment. Another interesting area for future research would be to handle uncertainty due to demand fluctuation (order arrival/cancellation) through this model. (In our experimental design, the duration of operations are correlated to the size of its job. This allows us to work in terms of processing time per unit of a job. This helps us model internal uncertainties such as yield in the manufacturing process and external uncertainties such as demand variations, through the processing time of the operations.)

APPENDIX

A.1. Illustration of Piecewise Linearity of the Total Penalty Cost Contribution from Each Operation

Operation i of job j is used synonymously with previous definition of operation O_{ij} of job j_l, defined in Section 2 of the paper.

Define

$$\begin{aligned} h_{ij} &= \text{inventory holding rate for operation } i \text{ of job } j,\\ p_{ij} &= \text{tardiness rate for operation } i \text{ of job } j,\\ h_{(i+1)j} &= \text{holding rate for operation } i+1 \text{ of job } j,\\ p_{(i+1)j} &= \text{tardiness rate for operation } i+1 \text{ of job } j. \end{aligned}$$

Define

$$\begin{aligned} t &= \text{current time},\\ jist_{ij} &= \text{just-in-time start time of operation } i \text{ of job } j,\\ y_{ij} &= \text{start time of operation } i \text{ of job } j \text{ (variable)},\\ du_{ij} &= \text{processing time of operation } i \text{ of job } j \text{ (random variable)}. \end{aligned}$$

The rest of the section presents an analysis of the total penalty costs associated with the start time of an operation. More generally, we show that these costs are piecewise linear.

We take a snapshot of the queue of operations in front of a resource at a certain time instant t. As discussed earlier, we associate varying inventory and tardiness rates for different operations of the same job. While performing the cost analysis, we restrict attention to the operations is the queue of a resource and their immediate successors. (A cost analysis based on a more global view of the problem was also experimented. For instance, this involves considering all the downstream operations of each job in the queue, while computing the associated penalties. The incremental benefits obtained didn't warrant pursuing this approach further.)

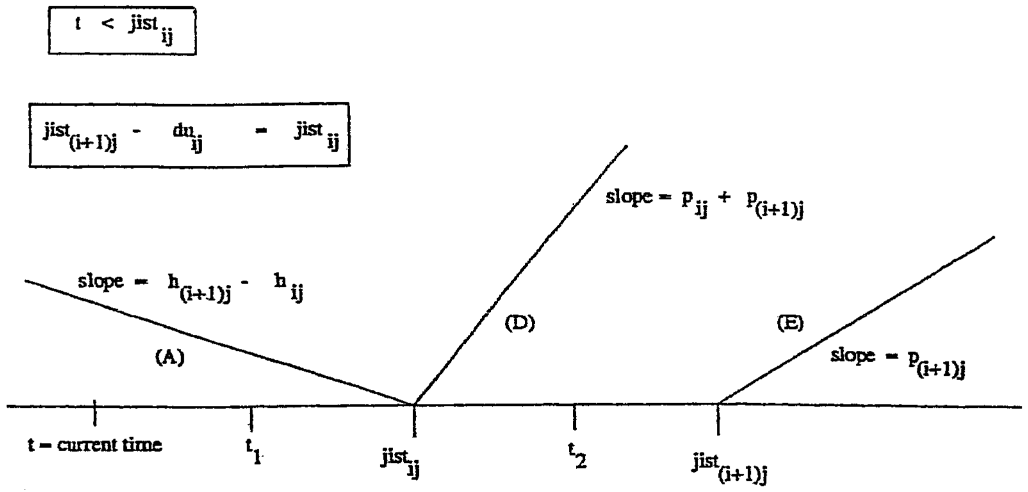

Figure 2.

The total penalty costs of y_{ij}, penalty-cost(y_{ij}), representing the sum of holding and tardiness cost of this operation and its successor for different scenarios, is presented in detail in Bollapragada [7]. (Specifically here, the possible regions where y_{ij} can fall and the associated costs for having the start time of the operation at different time instances is illustrated theoretically.)

For illustrative purposes, we present the intuition in Figure 2 below and show the piecewise linearity of the cost function. Clearly, the total cost associated with the start time y_{ij} (corresponds to the Y-axis) at time instances t_1 and t_2 is not linear in the inventory holding and tardiness rates, as these time instances fall under different cost regions. This proves the piecewise linearity of the cost function.

A.2. Derivation of Expected Cost Function

A closed form expression on the expected cost of a sequence of operations in the queue of a resource, is derived here. (The quality of the solutions generated by the heuristics proposed in the following section, can thus be obtained directly from this expression.)

Consider n jobs (more specifically, they correspond to different operations of these jobs) in the queue of a resource. Let [1][2][3][4]. . .[n] be the sequence of jobs for a schedule, where [i] indicates ith position in the sequence.

Define

t = current time,
$jist_{[i]}$ = just-in-time start time for operation in position i,
$emcst_{[i]}$ = expected-min-cost start time for operation in position i,
$du_{[i]}$ = processing time of operation in position i (random variable),
$ct_{[i]}$ = completion time of operation in position i,
C_S = total expected cost for a sequence of n operations in the queue of a resource,
$C_{[i]}$ = cost contribution from operation in position i of a sequence of operations.

Without loss of generality, we assume that operation i of job j is in position i of the sequence of n operations waiting to be processed, where, $i = 1, 2, \ldots, n$.

Hence, $du_{ij} = du_{[i]}$ and $ct_{ij} = ct_{[i]}$, where

$$ct_{[i]} = t + \sum_{k=1}^{i} du_{[k]}.$$

As stated earlier in Section 4, the processing time of operation in position i, $du_{[i]}$, is a random variable with marginal probability density function $f_{[i]}(\cdot)$. We assume that the processing times $du_{[i]}$, $i = 1, 2, \ldots, n$, are independent and normally distributed. The completion time of an operation in $(i - 1)$th position, $ct_{[i-1]}$, is thus normally distributed with mean $\mu_{[i-1]}$ and standard deviation $\sigma_{[i-1]}$. This is equivalent to stating that the start time of operation in position i of the sequence is normally distributed with mean $\mu_{[i-1]}$ and standard deviation $\sigma_{[i-1]}$.

We define $K_{1[i]}$, $K_{2[i]}$, $K_{3[i]}$ as positive constants, whose values are dependent on the holding and tardiness rates for the operation in position i of the sequence:

$\mu_{[i-1]}$ = mean start time of operation in position i of a sequence of operations,
$\sigma_{[i-1]}$ = standard deviation of the start time for operation in position i.

Define

$$v_{[i]} = \frac{jist_{[i]} - \mu_{[i-1]}}{\sigma_{[i-1]}},$$

$\phi(v_{[i]})$ = cumulative standard normal distribution function evaluated at $v_{[i]}$,
$\varphi(v_{[i]})$ = standard normal probability density function evaluated at $v_{[i]}$.

We show that the expected cost contribution of an operation in ith position of a sequence has the form below. A detailed analysis of the cost contribution by an operation for the six possible different scenarios, see Bollapragada [7]:

$$C_{[i]} = K_{1[i]} + (K_{2[i]} + K_{3[i]})[\sigma_{[i-1]}\varphi(v_{[i]}) + (jist_{ij} - \mu_{[i-1]})\phi(v_{[i]})] - K_{3[i]}(jist_{ij} - \mu_{[i-1]}).$$

More generally, for each of the six cases, the total expected cost for a sequence of n operations in a queue, has the following structure:

$$C_S = \sum_{i=1}^{n} K_{1[i]} + (K_{2[i]} + K_{3[i]})[\sigma_{[i-1]}\varphi(v_{[i]}) + (jist_{ij} - \mu_{[i-1]})\phi(v_{[i]})] - K_{3[i]}(jist_{ij} - \mu_{[i-1]}).$$

The values of $K_{1[i]}$, $K_{2[i]}$, $K_{3[i]}$ can be different for each scenario. The computation of these values is referred to Bollapragada [7].

We now concentrate on the above expected cost function and derive lower and upper bounds on the cost function. We present four new heuristics in the subsequent section. The best of these policies that produced solutions close to the lower bound was used to determine the sequencing of operations at each reoptimizing instant.

A.3. Lower and Upper Bounding Methods

Denote

$C_{[1i]}$ = cost contribution from expected start time of the operation in position i,
$C_{[2i]}$ = cost contribution from standard deviation of start time of operation in position i.

We show below analytically that the cost contribution from operation i of job j, $C_{[i]}$, has the following bounds:

$$C_{[1i]} \leq C_{[i]} \leq C_{[1i]} + C_{[2i]},$$

where

$$C_{[i]} = K_{1[i]} + (K_{2[i]} + K_{3[i]})[\sigma_{[i-1]}\varphi(v_{[i]}) + (jist_{ij} - \mu_{[i-1]})\phi(v_{[i]})] - K_{3[i]}(jist_{ij} - \mu_{[i-1]}).$$

If the mean of the start time of operation i of job j (which is assumed to be in position i of a sequence of n jobs), $\mu_{[i-1]}$, is equal to $jist_{ij}$, the cost contribution from this operation, $C_{2[i]}$, is

$$C_{[2i]} = K_{1[i]} + (K_{2[i]} + K_{3[i]})\sigma_{[i-1]} \frac{1}{\sqrt{2\pi}}.$$

If $\phi(v_{[i]}) = \Pr(ct_{[i-1]} < jist_{ij}) = 1$, $C_{[i]} = K_{1[i]} + K_{2[i]}(jist_{ij} - \mu_{[i-1]})$. While $\phi(v_{[i]}) = \Pr(ct_{[i-1]} > jist_{ij}) = 1$, $C_{[i]} = K_{1[i]} + K_{3[i]}(\mu_{[i-1]} - jist_{ij})$.

Summarizing the above cases,

$$C_{[1i]} = K_{1[i]} + K_{2[i]} \max[(jist_{ij} - \mu_{[i-1]}), 0] + K_{3[i]} \max[(\mu_{[i-1]} - jist_{ij}), 0].$$

PROPOSITION 1: The cost $C_{[i]}$ has an upper bound of $C_{[1i]} + C_{[2i]}$.

PROOF:

$$C_{[i]} = K_{1[i]} + (K_{2[i]} + K_{3[i]})[\sigma_{[i-1]}\varphi(v_{[i]}) + (jist_{ij} - \mu_{[i-1]})\phi(v_{[i]})] - K_{3[i]}(jist_{ij} - \mu_{[i-1]}).$$

If $(jist_{ij} - \mu_{[i-1]})$ is ≤ 0, $C_{[i]}$ is maximum when $\phi(v_{[i]}) = 0$ and $\varphi(v_{[i]}) = 1/\sqrt{2\pi}$.

This implies

$$C_{[i]} = K_{1[i]} + (K_{2[i]} + K_{3[i]})\sigma_{[i-1]} \frac{1}{\sqrt{2\pi}} - K_{3[i]}(jist_{ij} - \mu_{[i-1]}).$$

If $(jist_{ij} - \mu_{[i-1]})$ is ≤ 0, $C_{[i]}$ is maximum when $\phi(v_{[i]}) = 1$ and $\varphi(v_{[i]}) = 1/\sqrt{2\pi}$.

Thus

$$C_{[i]} = K_{1[i]} + (K_{2[i]} + K_{3[i]})\sigma_{[i-1]} \frac{1}{\sqrt{2\pi}} + K_{2[i]}(jist_{ij} - \mu_{[i-1]}).$$

Generalizing the above two cases, we conclude that $C_{[i]}$ has an upper bound of $C_{[1i]} + C_{[2i]}$. □

PROPOSITION 2: The cost $C_{[i]}$ has a lower bound of $C_{[1i]}$.

PROOF: We prove that the expected cost of the operation in ith position $C_{[i]}$ is convex in $\mu_{[i-1]}$ and $\sigma_{[i-1]}$, by showing that the Hessian is positive semidefinite.

The gradient of $C_{[i]}(\mu_{[i-1]}, \sigma_{[i-1]})$ is

$$\begin{pmatrix} \dfrac{\partial C_{[i]}}{\partial \mu_{[i-1]}} \\ \dfrac{\partial C_{[i]}}{\partial \sigma_{[i-1]}} \end{pmatrix},$$

which is equivalent to

$$\begin{pmatrix} -(K_{2[i]} + K_{3[i]})\phi(v_{[i]}) + K_{3[i]} \\ (K_{2[i]} + K_{3[i]})\varphi(v_{[i]}) \end{pmatrix}.$$

The Hessian of $C_{[i]}(\mu_{[i-1]}, \sigma_{[i-1]})$ is

$$\begin{pmatrix} (K_{2[i]} + K_{3[i]}) \dfrac{1}{\sigma_{[i-1]}} \varphi(v_{[i]}) & (K_{2[i]} + K_{3[i]}) \dfrac{jist_{ij} - \mu_{[i-1]}}{\sigma^2_{[i-1]}} \varphi(v_{[i]}) \\ (K_{2[i]} + K_{3[i]}) \dfrac{jist_{ij} - \mu_{[i-1]}}{\sigma^2_{[i-1]}} \varphi(v_{[i]}) & (K_{2[i]} + K_{3[i]}) \dfrac{jist_{ij} - \mu_{[i-1]}}{\sigma^2_{[i-1]}} \varphi(v_{[i]}) \end{pmatrix}.$$

As $\sigma_{[i-1]} \geq 0$, we clearly observe that the determinant of first principle minor of Hessian is positive and the determinant of second principle minor is zero, thus proving the Hessian to be positive semidefinite. The convexity of $C_{[i]}$ in $\sigma_{[i-1]}$ is thus proved. Also note that, $C_{[i]}$ achieves its minimum at $\sigma_{[i-1]} = 0$, illustrating that the expected cost $C_{[i]}$ has a lower bound at $C_{[1i]}$. □

We also observe that $\partial C_{[i]}/\partial \sigma_{[i-1]}$ is always positive for $\sigma_{[i-1]} > 0$ and hence infer that $C_{[i]}$ is an increasing nonlinear function of $\sigma_{[i-1]}$.

A.4. Idle Time Insertion as a Linear Program

The problem of optimally inserting idle time in an existing job shop schedule (i.e., given completely defined operation sequences on each resource) can be formulated as a linear program, as detailed below. This formulation was originally proposed in Sadeh and Nakakuki [29].

We experimented with two different ways of obtaining the processing time for each operation. In the computational testing section, we report the results based on both these approaches.

1. The duration of an operation was set equal to the expected processing time of the operation.
2. The duration of an operation was obtained by a sampling procedure. This involved assigning different probabilities for each realization of the processing time. The duration of an operation was estimated based on assigning (a) a certain probability of realization for the deterministic duration, (b) a certain probability of realization for the sum of deterministic duration and the mean time to repair, and (c) a certain probability of realization based on the knowledge of deterministic processing time and the standard deviation of the processing time, which is dependent on time to repair and time between failure distributions (the best fit probabilities of realization for cases a, b, c discussed here were determined empirically through extensive experimentation).

We also tried solving an extensive form of stochastic linear program, but did not pursue it after preliminary testing on some of the problem sets. The computational expenditure of this procedure was not worth the little improvements in schedule quality obtained over the above two procedures.

A.5. A Method to Compute Just-in-Time Start Times

We present here a method to compute the just-in-time start time for an operation of a job. Apart from the standard notation in the paper, we define

$\overline{remdu_{ij}}$ = expected remaining processing time for operation i of job j

$\overline{resdu_{ij}}$ = expected resource processing time for operation i of job j

The expected remaining processing time of an operation i is defined as the sum of the expected duration of this operation and its successors in the job. The expected resource processing time of an operation i is defined as the average processing time of all the operations (belonging to different jobs) that visit this resource. Note that $\overline{resdu_{ij}}$ is identical for all operations that are processed on the same resource:

$$jist_{ij} = dd_l - \overline{remdu_{ij}} - k\overline{resdu_{ij}}\ \ln\ \frac{inv_{ij} + tard_l}{inv_{il}}.$$

The value of k can be computed empirically. A good value of k was determined to be around 1.5–2.5 for our experiments. In our experimentation, the above method was used to compute the value of just-in-time start times. Yet, another heuristic to compute $jist_{ij}$ is

$$jist_{ij} = dd_l - \overline{remdu_{ij}} - k\overline{resdu_{ij}}\frac{inv_{il} + tard_l}{inv_{il}}.$$

The tail leadtime estimation ideas based on EXP-ET and LIN-ET policies in the literature (see Morton and Pentico [18]) are used in defining the above just-in-time start times.

A.6. Reoptimizing Heuristics

We present here the remaining three reoptimizing heuristics that determine the sequencing decisions for operations waiting in front of a resource. The goal of these heuristics is to determine the expected-min-cost start times, $emcst_{ij}$, for operations in the queue of this resource. In our experimental section, we provide a comparison between these three heuristics and the reoptimizing heuristic, PBH1, presented in Section A.1.

PBH2: This heuristic is essentially similar to PBH1, except that the tardiness (τ_i) and inventory (ϵ_i) rates are computed differently. The inventory holding and tardiness rates for each operation correspond to that defined in Section 2.

The values of τ_i and ϵ_i are set, respectively, to the tardiness and earliness rates of each operation. The tardiness rate of each operation is set to the tardy rate of the job and the holding rate of each operation to that of the marginal inventory cost introduced by this operation, and the sum of the marginal inventory costs introduced by the upstream operations belonging to this job.

Except the way the inventory and tardiness rates for each operation are computed, *the rest of the formulation is identical to that of PBH1* (*the objective as well as the constraints*).

PBH3: In the previous two heuristics proposed, local due dates of the operation were used as a measure in solving the single machine problem. This heuristic considers the global due date for the operation (essentially the job's due date to which this operation belongs), while sequencing the operations.

Apart from the notation defined in PBH1, define

$\overline{remdu_i}$ = expected remaining processing time of operation i,
dd_i = due date of the job to which operation i belongs.

The expected remaining processing time of an operation i is defined as the sum of the expected processing time of the operation and its successors in the job. A method to compute the expected processing time of an operation, $\overline{du_i}$, is described in the section on reoptimizing heuristic PBH1.

The formulation here is different from the formulation for PBH1 only in constraints 1–4, which are presented here:

$$ct_i \geq t + \overline{remdu_i}, \qquad i = 1, 2, \ldots, n, \tag{1'}$$

$$ct_i - tard_i + inv_i = dd_i, \qquad i = 1, 2, \ldots, n, \tag{2'}$$

$$ct_j - ct_i + M(1 - x_{ij}) \geq \overline{du_i} + \overline{remdu_j} - \overline{remdu_i}, \qquad i < j = 1, 2, \ldots, n, \tag{3'}$$

$$ct_i - ct_j + Mx_{ij} \geq \overline{du_j} + \overline{remdu_i} - \overline{remdu_j}, \qquad i < j = 1, 2, \ldots, n. \tag{4'}$$

The tardiness (τ_i) and inventory rates (ϵ_i) for an operation are similar to those defined under PBH1.

PBH4: The intuition behind the newsboy problem is used in developing this heuristic. The expected-min-cost start time, $emcst_{ij}$, for each operation is obtained by a tradeoff between inventory, tardiness costs, and the knowledge of the randomness due to machine failures. The just-in-time start time, $jist_{ij}$, here is analogous to the mean demand in the standard newsboy model. The holding cost for having the start time of this operation before its $jist_{ij}$ and tardiness penalty for starting the operation after its $jist_{ij}$ are analogous to inventory and shortage penalties in a standard newsboy problem. A parallel can be drawn between the stochasticity of demand in the newsboy model and the randomness due to machine breakdowns in our domain. As stated earlier, the time between failure (TBF) and time to repair (TTR) are assumed to be normally distributed. In this heuristic analysis, the holding and tardiness rates for each operation are contributed both from the job to which it belongs (job-centered perspective) and the resource where the operation is currently present (resource-centered perspective). The holding and tardiness penalties contribution for an operation from the resource is based on the pricing ideas presented in Morton and Pentico [18] and Morton et al. [19]. These are computed based on the average load of each resource at the current time, and the current stage of completion of each job. The resource price could thus be different for operations in front of the same resource.

The holding and tardiness rates for each operation (h_{ij} and p_{ij}) are similar to those defined in Section A.1.

Apart from the notation introduced before, we define

h_{i_m} = holding cost per unit time for operation i in front of machine m,
p_{i_m} = tardiness cost per unit time for operation i in front of machine m,
$f_{f_m}(\cdot)$ = probability density function of failure time of machine i,
$f_{r_m}(\cdot)$ = probability density function of repair time of machine i,
$\Phi_m(\cdot)$ = joint probability density function of failure and repair time of machine i,
$F_m(\cdot)$ = cumulative distribution function of failure and repair time of machine i,
var_x = variance of the joint probability density function.

The expected-min-cost start time, $emcst_{ij}$, is obtained by minimizing the cost function:

$$f(y_{ij}) = (h_{ij} + h_{i_m}) \int_{du_{ij}}^{y_{ij}} (y_{ij} - x)\Phi_i(x)dx + (p_{ij} + p_{i_m}) \int_{y_{ij}}^{\infty} (x - y_{ij})\Phi_i(x)\, dx.$$

The resulting value of $emcst_{ij}$ is obtained by solving the above expression using Liebnitz's theorem.

$$emcst_{ij} = jist_{ij} - \sqrt{\mathrm{var}_x}\, \mathrm{F}_i^{-1} \frac{(\mathrm{p}_{ij}+\mathrm{p}_{i_m})}{(\mathrm{h}_{ij}+\mathrm{p}_{ij}+\mathrm{h}_{i_m}+\mathrm{p}_{i_m})}.$$

The intuition here is to obtain expected-min-cost start times for each operation of a job by appropriately considering the variance due to randomness created by machine failures and repairs apart from the associated tradeoff between inventory and tardiness costs.

A.7. Design of the Test Data

The material in this subsection can be used as a reference aid for the experimental section of the paper. It describes the problem set generation outlined in Section 5 of the paper (see Table 1) in detail here.

A set of 80 scheduling problems was randomly generated to cover a wide range of scheduling conditions. Scheduling conditions were varied by adjusting a set of three parameters [these parameters or similar ones are commonly used in the scheduling literature to study different shop floor situations (Fisher [10], Ow [23], Morton et al. [19])]: a parameter controlling the average due date of the jobs to be scheduled (tardy factor), a parameter controlling the variance of the job due dates (due date range), and a parameter controlling the number of major bottleneck machines.

These three scheduling parameters were set as follows:

- Tardy factor (τ): This factor controlled the average tightness of job due dates in the experiments. Given an estimate of the expected makespan of the problem, say M, the average job due date was set to $(1 - \tau)M$, where τ is the tardy factor ($M = (n - 1)\overline{du_{R_{btnk}}} + \Sigma_{R=R_1}^{R_m} \overline{du_R}$, where n is the number of jobs, m the number of resources, R_{btnk} the main bottleneck resource, and $\overline{du_{R_i}}$ denotes the average duration of the operation requiring resource R_i. This estimate was first proposed by Ow [23]). If $\tau = 0$, it might be possible to complete all jobs on time. As τ increases, the proportion of jobs that can still be completed on time decreases. In the scheduling problems that were generated, two values of the tardy factor were used: $\tau = 0.2$ (loose average due date) and $\tau = 0.4$ (tight average due date).
- Due date range (R): Job due dates were randomly drawn from a uniform distribution $(1 - \tau)MU(1 - R/2, 1 + R/2)$, where $U(a, b)$ represents a uniform distribution between a and b, and R is the due date range. Two due date ranges were used: 1.2 (wide due date range) and 0.6 (tight due date range).
- Number of bottlenecks: all problems involved five resources. In half of the problems, one out of the five resources was selected to be a major bottleneck, while the other half of the problems had two major bottlenecks.

A set of 10 scheduling problems was randomly generated for each parameter combination (see Table 1), resulting in a total of 80 scheduling problems (10 problems $\times$ 2 tardy factors $\times$ 2 due date ranges $\times$ 2 bottleneck configurations). All experiments involved 20 jobs and 5 resources, for a total of 100 operations. Each job had a linear process routing, and had to go through each machine exactly once. The order in which a job would go through the machines was randomly generated for each job, except for the bottleneck machines, which were always visited after a fixed number of operations (in order to further increase resource contention). In the case with one bottleneck resource, the bottleneck operation corresponded to the fourth operation (out of five); in the case with two bottlenecks, the bottlenecks correspond to the second and fifth operations of each job.

- Job sizes (S_l): The size S_l of job j_l (i.e., the number of parts to be produced) was randomly drawn from a uniform distribution $U(1, 7)$.
- Operation durations (du_i^l): Operation durations were correlated with job sizes. The duration of an operation O_i^l on a nonbottleneck resource was randomly drawn from a uniform distribution $S_lU(0.5, 1.5)$. In problems with a single bottleneck, the duration of an operation O_i^l requiring that bottleneck was set to $du_i^l = 2S_l$. In problems with two bottlenecks, the duration of the operation requiring the first bottleneck was set to $1.8S_l$ and that of the operation requiring the second bottleneck was set to $2S_l$.

- Tardiness costs ($tard_l$): the marginal tardiness cost of job j_l, $tard_l$, was randomly drawn from a uniform distribution $5U(1, 2S_l)$.
- Inventory costs (inv_i^l): Inventory costs were only introduced by the first operation in each job, namely, operation O_1^l. inv_1^l can be interpreted as the sum of the marginal holding costs and the interests on raw material for job j_l. These costs are typically proportional to the size of the job. In these experiments, inv_1^l was simply set to S_l. Given that the mean of the marginal tardiness cost distribution is slightly larger than $5S_l$, this corresponds to a ratio of marginal tardiness cost over marginal inventory cost with a mean slightly larger than 5.
- MTTR and MTBF data: The time to repair is drawn from a Poisson distribution, with the mean parameter defined as the average of the duration of all operations (100) in the scheduling problem considered. The time between failures is drawn from a normal distribution with a coefficient of variation of 0.2–0.3. The mean parameter here is defined in terms of the mean parameter for time to repair and the percentage of time the machines are not available. (As an example, if mean time to repair is 5 units of time, and machines are not available for 10% of the time, the mean parameter for time between failures is 45 units of time.)
- Earliest acceptable release dates/latest acceptable completion date (erd_l, lcd_l): In order to increase contention, all jobs were given the same earliest acceptable release date, $erd_l = 0$. Since these experiments were designed to study the quality of the schedules produced by the different techniques discussed in the paper, all jobs were given a nonconstraining latest acceptable completion date, which was set to $lcd_l = 3M$ (where M is the makespan estimate described earlier).

ACKNOWLEDGMENTS

This work is part of the first author's Doctoral Dissertation at GSIA, Carnegie Mellon University. He would like to thank the Business School and the Computer Science department at CMU for providing a creative atmosphere to conduct this research. Further, the first author would like to dedicate this paper in loving memory of his mother, Mrs. Mangatayaru Bollapragada. Her tremendous love, constant support, patience, and dedication have been a great source of inspiration in the first author's entire academic career.

REFERENCES

[1] J. Adams, E. Balas, and D. Zawack, The shifting bottleneck procedure for job shop scheduling, Management Sci 34(3) (1988), 391–401.
[2] D. Applegate and W. Cook, A computational study of the job shop scheduling problem, ORSA J Comput 3(2) (1991), 149–156.
[3] K.R. Baker, Introduction to sequencing and scheduling, Wiley, New York, 1974.
[4] K.R. Baker and G.D. Scudder, Sequencing with earliness and tardiness penalties: A review, Oper Res 38(1) (1990), 22–36.
[5] J.C. Bean, J.R. Birge, J. Mittenthal, and C.E. Noon, Matchup scheduling with multiple resources, release dates and disruptions, Oper Res 39(3) (1991), 470–483.
[6] R.A. Blau, *N*-job, one machine sequencing problems under uncertainty, Management Sci 20 (1973), 101–109.
[7] R. Bollapragada, Integration of capacitated lotsizing, scheduling and control decisions, Doctoral Thesis, Graduate School of Industrial Administration and The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.
[8] F. Della Croce, R. Tadei, and G. Volta, A genetic algorithm for the job shop problem, Technical Report, D.A.I. Politecnico di Torino, Italy, 1992.
[9] M.L. Fisher, A dual algorithm for the one machine sequencing problem, Math Program 11 (1976), 229–251.
[10] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, Freeman, New York, 1979.
[11] C.R. Glassey and M.G.C. Resende, Closed-loop job release control for VLSI circuit manufacturing, IEEE Trans Semiconductor Manufacturing 1(1) (1988), 36–46.

[12] F. Glover and M. Laguna, "Tabu search," Modern heuristic techniques for combinatorial problems, Colin Reeves (Editor), Blackwell Scientific, Oxford, 1992, pp. 70–150.
[13] E.M. Goldratt and J. Cox, The goal: A process of ongoing improvement, North River Press, Great Barrington, MA, 1986.
[14] J. Holland, Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, MI, 1975.
[15] E.L. Lawrence, J.K. Lenstra, and A.H.G. Rinnooy, Recent developments in deterministic sequencing and scheduling: A survey, Deterministic and stochastic scheduling, Riedel, Dordrecht, 1982.
[16] C.Y. Lee, S. Piramuthu, and Y.K. Tsai, Job shop scheduling with a genetic algorithm and machine learning, Int J Prod Res 35 (1997), 1171–1191.
[17] H. Matsuo, C.J. Suh, and R.S. Sullivan, A controlled search simulated annealing method for the general job shop scheduling problem, Technical report, Department of Management, University of Texas at Austin, Austin, TX, 1988.
[18] T.E. Morton and D.W. Pentico, Heuristic scheduling systems, Wiley Series in Engineering and Technology Management, Wiley, New York, 1993.
[19] T.E. Morton, S.R. Lawrence, S. Rajagopalan, and S. Kekre, SCHED-STAR a price-based shop scheduling module, J Manuf Oper Management 1(2) (1988), 131–181.
[20] J. Mittenthal and M. Raghavachari, Stochastic single machine scheduling with quadratic early-tardy penalties, Oper Res 41(4) (1993), 786–796.
[21] G.L. Nemhauser and L.A. Wosley, Integer and combinatorial optimization, Wiley, New York, 1988.
[22] J. Orlicky, Material requirements planning, McGraw-Hill, New York, 1975.
[23] P.S. Ow, Focused scheduling in proportionate flowshops, Management Sci 31 (1985), 852–869.
[24] S.S. Panwalkar and W. Iskander, A survey of scheduling rules, Oper Res 25(1) (1977), 45–61.
[25] M. Pinedo, Scheduling: Theory, algorithms, and systems, Prentice Hall International Series in Industrial and Systems Engineering, Prentice Hall, Englewood Cliffs, NJ, 1995.
[26] M. Pinedo, Stochastic scheduling with release dates and due dates, Oper Res 31(3) (1983), 559–572.
[27] N. Sadeh, Look-ahead techniques for micro-opportunistic job shop scheduling, Doctoral Thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1991.
[28] N. Sadeh, "Micro-opportunistic scheduling: The micro-boss factory scheduler," Intelligent scheduling, M. Zweben and M. Fox (Editors), Morgan Kaufmann, San Mateo, CA, 1994, Chap. 4.
[29] N. Sadeh and Y. Nakakuki, Focused simulated annealing search: An application to job shop scheduling, Ann Oper Res 63 (1996), 77–103.
[30] N. Sadeh, S. Otsuka, and R. Schnelbach, Predictive and reactive scheduling with the micro-boss production scheduling and control system, IJCAI-93 Workshop, Chambery, France, 1993, pp. 10–18.
[31] S.C. Sarin, E. Erel, and G. Steiner, Sequencing jobs on a single machine with a common due date and stochastic processing times, European J Oper Research 51 (1991), 188–198.
[32] S.F. Smith, P.S. Ow, C. LePape, and N. Muscettola, Reactive management of factory schedules, ISL Working Paper, Carnegie Mellon University, Pittsburgh, PA, 1986.
[33] E. Taillard, Parallel taboo search technique for the job shop scheduling problem, ORSA J Comput 6 (1994), 108–117.
[34] P.J. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra, Job shop scheduling by simulated annealing, Oper Res 40(1) (1992), 113–125.