Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2016), *30*, 289–299.
© Cambridge University Press 2015 0890-0604/15
doi:10.1017/S0890060415000323

Managing risk in production scheduling under uncertain disruption

RUHUL SARKER,[1] DARYL ESSAM,[1] S.M. KAMRUL HASAN,[1] AND A.N. MUSTAFIZUL KARIM[2]

[1]School of Engineering and Information Technology, University of New South Wales at Canberra, ADFA Campus, Canberra, Australia
[2]Department of Manufacturing and Materials Engineering, International Islamic University Malaysia, Gombak, Kuala Lumpur, Malaysia

(Received July 29, 2014; Accepted April 6, 2015)

Abstract

The job scheduling problem (JSP) is considered as one of the most complex combinatorial optimization problems. JSP is not an independent task, but is rather a part of a company business case. In this paper, we have studied JSPs under sudden machine breakdown scenarios that introduce a risk of not completing the jobs on time. We have first solved JSPs using an improved memetic algorithm and extended the algorithm to deal with the disruption situations, and then developed a simulation model to analyze the risk of using a job order and delivery scenario. This paper deals with job scheduling under ideal conditions and rescheduling under machine breakdown, and provides a risk analysis for a production business case. The extended algorithm provides better understanding and results than existing algorithms, the rescheduling shows a good way of recovering from disruptions, and the risk analysis shows an effective way of maximizing return under such situations.

Keywords: Disruption; Genetic Algorithm; Job Scheduling; Memetic Algorithm; Risk Analysis

1. INTRODUCTION

Job scheduling is a common task in the manufacturing industry. Over the last few decades, a considerable amount of research has been carried out on how to develop effective solution approaches for job scheduling problems (JSPs). However, no single algorithm is well accepted as an effective method for all kinds of JSPs (Hasan et al., 2011). Meeran and Moeshed (2012) pointed out that none of the techniques are sufficient on their own to solve these stubborn NP-hard problems. Although deterministic scheduling algorithms converge to an optimal solution, these algorithms are incapable of handling complex and large-scale problems (Qind-dao-er-ji & Wang, 2012). Stochastic and heuristic algorithms perform relatively better in those cases (Nasr & ElMekkawy, 2011). However, the existing genetic algorithms (GAs) for JSPs usually have slow convergence rates, and they can be trapped into local optima. Many of such algorithms can solve small problems optimally but have difficulty in solving large problems with optimal or near-optimal solutions within a reasonable time limit. In order to enhance the performance of these algorithms, researchers have been trying many different strategies, such as hybridizing with different local search algorithms (Qind-dao-er-ji & Wang, 2012). Although the deterministic scheduling algorithms converge to the optimal solution, these algorithms are incapable of handling complex and large-scale problems. Stochastic and heuristic algorithms perform relatively better in those cases (Nasr & ElMekkawy, 2011).

In this paper, we consider n jobs and m machines. The JSPs have several constraints, such as machine capacity and precedence requirements. It is considered that a machine can only perform a particular type of operation. Thus, a machine is able to execute just a single operation of a job. The operations are nonpreemptive; an operation can neither be paused nor resumed after it is started. The execution time for each of the operations is known. The setup time or cost is assumed to be negligible. An operation can only be started if the preceding operations of the same job are complete. This description is similar to a traditional job-shop scheduling problem (Hasan et al., 2009; Ghasem & Mehdi, 2011).

In almost all research on JSPs, the schedule was produced under ideal conditions, assuming there will be no disruptions of any type. However, machine unavailability is a common event on the shop floor due to both preventive and breakdown maintenance of machineries and their supporting equipment (Nasr & ElMekkawy, 2011). The inclusion of such unavailability with the JSPs makes the resulting problem not only more practical but also more complex and challenging. The preventive maintenance schedules, which are usually known in advance, can easily be incorporated in generating a job scheduling. However, in the case of sudden machine failure,

Reprint requests to: Ruhul Sarker, School of Engineering and Information Technology, University of New South Wales at Canberra, ADFA Campus, Canberra, Australia 2600. E-mail: r.sarker@adfa.edu.au

the tasks scheduled on the broken machine cannot be resumed until the machine is appropriately repaired. During and after the machine repair, the continued implementation of the schedule generated earlier would delay the completion of some jobs due to active precedence constraints. Such delays may create a contractual problem with customers. In order to minimize the delay, it is very important to reoptimize the remaining tasks at the time of machine breakdown. Even after reoptimization, it is expected that there will be some delay in the completion of some or all jobs. Any late completion may incur penalty costs for the existing jobs and may prevent the acceptance of new jobs. The situation becomes even more complex when the duration of delay is stochastic.

In this research, we have proposed an improved memetic algorithm (IMA) for solving JSPs, which combines a GA with a local search (/heuristic) technique. We have considered the makespan minimization as our objective. The total time between the starting of the first operation and the ending of the last operation is termed as the *makespan*. For a candidate solution, the heuristic technique identifies any gap left between any two consecutive operations (/jobs) on a machine. A job from the right of the gap can then be placed in it, if the gap is big enough for the operation without violating any precedence constraints. The job can also be placed in the gap, even if the gap is not big enough, but within a certain tolerance limit of the operation time, if it improves the overall fitness value. In this case, it may need to push (/shift) other jobs to the right. We have extended this algorithm to study JSPs under sudden machine breakdowns. In the case of machine breakdown, the breakdown information is known after the actual breakdown when the schedule is under implementation. In such a case, it is required to reoptimize the remaining operations by taking into account the machine downtime. The breakdowns will affect the shop capacity and the acceptance of new jobs that would ultimately affect the revenue earned. We have developed a mathematical model to study the revenue earned under different scenarios and have found the best parameter set for maximizing the revenue using a simulation model. With stochastic breakdown starting time and duration, the simulation model basically suggests a way of risk minimization in revenue generation.

To study the above approaches in a systematic manner, we have chosen a manufacturing shop with 10 machines. We have solved 20 benchmark test problems using the IMA and have experimented by varying the tolerance limit for placing the jobs in the gaps. We have shown that the inclusion of the heuristic (discussed earlier) not only improves the performance of the GAs but also reduces the overall computational requirements. We have observed that the solution quality gradually improves with the increase of the tolerance limit. For the condition of machine breakdowns, we have used different distributions to generate a number of breakdown scenarios. We have found that the revised solution is able to recover from most of the prescheduled breakdowns regardless of when it occurs. The simulation model is developed based on the job completion times under breakdowns. The model shows that the revenue can be maximized by carefully analyzing the different levels of risk.

The research contribution made in this paper can be summarized as follows. The main research problem considered in this paper is how to manage the financial risk in a job shop business unit that as time passes, receives different job orders from its customers and also faces frequent machine breakdowns. A new methodology has been proposed to study this problem. This methodology was developed, based on an approach for scheduling under machine breakdowns that is also proposed in this paper. The second approach was developed based on a scheduling algorithm under ideal situations, in that the base algorithm in this study is basically an improved version of our earlier algorithm. We have demonstrated the usefulness of these approaches by solving test problems under different scenarios.

The paper is organized as follows. After the Introduction, a brief outline of a standard JSP with and without disruption is given in Section 2. The GA to solve JSPs is discussed in Section 3. Section 4 provides an experimental study for JSPs with and without random machine breakdowns. Section 5 provides a new procedure for managing financial considerations under random disruption, along with results and discussions. Finally, conclusions are drawn in Section 6.

2. JOB SCHEDULING AND DISRUPTION PROBLEM

The standard JSP makes the following assumptions:

- Each job consists of a finite number of operations.
- The processing time for each operation in a particular machine is defined.
- There is a predefined sequence of operations that has to be maintained to complete each job.
- Delivery times of the products are not defined.
- There is no setup cost or tardiness cost.
- A machine can process only one job at a time.
- Each job visits each machine only once.
- No machine can deal with more than one type of task.
- The system cannot be interrupted until each operation of each job is finished.
- No machine can halt a job and start another job before finishing the previous one.
- Each and every machine has full efficiency.

The objective of the problem is to minimize the maximum time taken to complete each and every operation while satisfying the machining constraints and required operational sequence of each job.

In practice, the operations may be interrupted or delayed for many reasons, such as machine breakdown (including tools and fixtures), unexpected machine setting up, arrival of a new priority job, process time variation, change of job priorities, defective materials, delay in transfer line between machines, and order cancellation (Subramaniam et al., 2005; Fahmy et al., 2008). Machine breakdown is considered as one of the most frequent and challenging issues in production scheduling. The unavailability of machines, due to planned preventive maintenance, can be incorporated as a constraint when solving for an optimal schedule. In the case of sudden breakdowns, the easiest solution is to apply some dispatching rules that help to select a task immediately after the breakdown occurs (Blackstone et al., 1982).

Reactive scheduling was recently introduced to deal with the unexpected interruptions. Liu et al. (2005) have proposed dividing the scheduling process into two nonoverlapping parts: *predictive* before the breakdown occurs, and *reactive* when the machine is recovered after the breakdown. Fahmy et al. (2008) have suggested inserting dummy tasks to remove the affected tasks from the schedule and to later reschedule them. The duration of the dummy task is equal to the recovery time of the broken machine. Abumaizar and Svestka (1997) have proposed repairing the reactive schedules using a *right shifting* process. The process shifts each and every operation to its right after the machine breakdown. Wu et al. (1993) developed a GA with a pairwise-swapping heuristic and proposed using the right shifting technique to reoptimize. The drawback in this technique is the presence of uniform shifting for every operation, which increases the machine idle times between consecutive operations.

3. JOB SCHEDULING WITH A GA

In this paper, we consider the minimization of makespan as the objective of JSPs. According to the problem definition, the sequence of machines used (those are also the sequence of operations) by each job is given. That means each operation is linked to one particular machine. In this case, if we know either the starting or the finishing time of each operation, we can calculate the makespan for each job and generate the whole schedule. In JSPs, the main problem is finding the order of jobs to be operated on each machine that minimizes the overall makespan.

In solving JSPs using GAs, the chromosome of each individual usually comprises the schedule. Chromosomes can be represented by binary, integer, or real numbers. Some popular representations for solving JSPs are operation based, job based, preference-list based, priority-rule based, and job pair-relation based representations (Ponnambalam et al., 2001). We select the job pair-relation based genotype representation due to the flexibility of applying genetic operators (Nakano & Yamada, 1991; Paredis, 1992; Yamada & Nakano, 1997; Yamada, 2003). In this representation, a chromosome is symbolized by a binary string, where each bit stands for the order of a job pair (u,v) for a particular machine m.

For a chromosome p,

$$\mathrm{CP}_{uvm} = \begin{cases} 1 & \text{if the job } u \text{ leads the job } v \text{ on machine } m \\ 0 & \text{otherwise} \end{cases}.$$

A value of 1 means that, for the individual p, the job u must lead the job v on machine m. The job having the maximum number of 1s is the highest priority job for that machine. Further details on this representation can be found in Yamada (2003) and Hasan et al. (2009).

A GA starts with a set of randomly generated solutions, known as the initial population. As indicated earlier, we use the job pair relationship-based chromosome representation. The main advantage of this representation lies with the organization of the genes. In this representation, two consecutive genes represent the relationship between two job pairs in two different machines. When a simple exchange crossover or swap mutation is applied, it affects multiple machines. Thus, this representation helps to generate diverse solutions.

It is well known that randomly generated solutions may not be feasible in JSPs. For example, on a given machine, the order of three jobs may be generated as follows: $j_1 \rightarrow j_2, j_2 \rightarrow j_3$, and $j_3 \rightarrow j_1$. Unless we change the order from $j_3 \rightarrow j_1$ to $j_1 \rightarrow j_3$, this looks like an infeasible assignment. We repair such infeasibility before applying the search operators. The mapped phenotypes contain a set of randomly assigned tasks where the constraints may be violated. For example, two different jobs may be assigned to a single machine at the same time. Therefore, this again requires repair operations. Nanko and Yamada (1991) have referred to such repair processes as harmonization. We apply similar repair mechanisms in this research.

We have proposed a heuristic in this paper, introduced as the job rearranging method for improving the performance of GA. The job rearranging method process is applied to the phenotypes to improve the solution, and thus ensure feasibility. The heuristic can also be applied as a part of the reactive scheduling.

3.1. Rearranging method

It is very common in practice to leave some gaps (i.e., machine idle time) between the consecutive tasks scheduled on a machine. In some cases, these are absolutely necessary to satisfy the precedence constraints. In other cases, this is due to the generation of suboptimal solutions for implementation. For the later cases, it is possible to improve the solution by filling in the gaps with suitable tasks. In our proposed rearranging method (RM), a gap will be filled in with a task if the gap is good enough to accommodate it without creating infeasibility. In addition, a gap can also be filled in if it is smaller than the task duration by a certain tolerance limit by shifting the tasks to the right of the gap. Here, let us assume that $g_{(\text{time})}$ is a gap where we wish to fit a task with operation time t_{ij}. By (i, j) we mean the task of job j that will be processed in machine i. We further assume that α is the tolerance limit, which varies between 0 and 1. The necessary condition of applying RM can be defined as follows: if $g_{\text{time}} \geq (1-\alpha)t_{ij}$, the job is placed in the gap. Further, a gap may be removed or reduced by simply moving a job to its adjacent gap at the left (left shifting). This process could be used to develop a compact schedule, starting from the left, and continuing up to the last job for each machine. If we assume $\alpha = 0$, the rearranging procedure would be similar to the gap reduction technique used in the memetic algorithm recently reported by Hasan et al. (2009). For convenience of applying RM, it is done on the phenotype, just after the evaluation of individuals (see below Step 2C).

3.2. IMA

The IMA can be described as follows:

```
1. Initialize $P(t)$ as a random population $P(t = 0)$ of size $K$.
2. Repeat until the stopping criteria is met.
    A. Set $t := t - 1$.
    B. Evaluate $P(t-1)$ as follows:
        i. Decode each individual $p$.
        ii. Repair if the individual is infeasible.
        iii. Generate schedule and calculate makespan.
        iv. Go to step i, if every individual is not evaluated.
        v. Rank the individuals according to the fitness values.
        vi. Apply elitism. Assume $P'(t-1)$ are the nonelite individuals.
    C. Apply RM to all individuals in $P'(t-1)$.
    D. Go to Step 3 if the stopping criteria is met.
    E. Modify $P'(t\text{-}1)$ using the following steps:
        i. Select individuals for reproduction.
        ii. Apply crossover.
        iii. Apply mutation.
        iv. Assign $P(t) =$ Modified $P'(t-1) +$ the preserved elite individuals $(K - P'(t-1))$.
    [End of Step 2 Loop]
3. Save the best solution among all of the feasible solutions.
   [End of Algorithm]
```

GA is similar to the above IMA except that it does not use Step 2C.

4. EXPERIMENTAL STUDY

We have implemented IMA for solving JSPs. We have started with a randomly generated population. Each individual is represented by the job-pair relation based representation. The parameters used in this study are similar to Hasan et al. (2009): the population size is 2500, the probability of crossover is 0.65, the probability of mutation is 0.30, and the number of generations is 1000. In JSPs, a large population is usually used because the feasible space is too small compared to the entire search space. As an example, Pezzella et al. (2008) used a population size of 5000 merely to solve a 10×10 JSP.

We apply elitism in each generation to pass a few of the best individuals unaltered to the next generation. We use a tournament selection that chooses one individual from the elite class of the individual (i.e., the top 15%) and two individuals from the rest. This selection then plays a tournament between the last two and performs crossover between the winner and the elite one. We use 2-point crossovers, and a bit-flip mutation. We rank the individuals on the basis of their fitness values. A high selection pressure on the better individuals may contribute to premature convergence. If the most or all of the elite class have the same solution, then their offspring will be quite similar after some generations. In such cases, a higher mutation rate would help to diversify the population. We use the tolerance level $\alpha = 0.20$ for applying RM. We have run IMA by varying the tolerance level α from 0.00 to 0.25 with an increment of 0.05 and decided upon the best α by analyzing the best and the average fitness values, as well as the computational times.

To test the performance of our proposed algorithms, we solve 20 "la" series (10-machine) benchmark problems (*la*16–*la*35) that were proposed by Lawrence (1985) and compare with several existing algorithms. Based on the number of operations, these problems can be grouped into four different sizes (10, 15, 20, or 30 operations) with five problems in each group.

The results for the benchmark problems are obtained by executing the algorithms on a personal computer, and the summary of results is tabulated in Table 1. From the results, it is clear that the performance of both MA (where $\alpha = 0$) and IMA is overwhelmingly better than GA. Out of 20 test problems, IMA obtained optimal solutions in 13 problems compared to 11 for MA and 3 for GA. The average percentage deviation of fitness values from the known optimal is much lower for IMA (1.047 compared to 1.129 for MA and 4.161 for GA). The average percentage deviation is calculated using the best of the best fitness values in each of the 30 independent runs for each problem. We must mention here that *la*16–*la*35 test problems are much harder than *la*01–*la*15. For the first 15 *la* series test problems, GA was able to obtain optimal solutions in 13 problems, whereas both MA and IMA successfully achieved optimal solutions in all 15 problems. Both IMA and MA are computationally more attractive than GA. This means that the proposed hybridization of GA not only improved the quality of solutions but also reduced the requirement of the number of fitness evaluations and the computational time (see Table 1).

Table 1. *Comparing the performance of IMA with GA and MA*

Algorithm	Optimal Found	Average Deviation (%)	Fitness Evaluation (10^3)	Average Comp. Time (s)
GA	3	4.161	992.38	291.76
MA	11	1.129	525.75	176.68
IMA	13	1.047	521.69	175.21

Note: GA, Genetic algorithm; MA, memetic algorithm; IMA, improved memetic algorithm.

The job shop scheduling problems have been solved by using many different algorithms, such as exact algorithms, heuristics, and population based stochastic search algorithms. Although the exact algorithms produce one single solution, the population based algorithms must be executed multiple times, and the reported results must include at least the best of all runs, the average of best objective values in all runs, and their standard deviations. This is because an algorithm may obtain the optimal solution for a given problem, due to the many randomized parameters used in the algorithm, even though the average performance is bad. For such algorithms, we need to compare not only the best objective value but also the average performance, using statistical significant testing. It is interesting that it has not become a norm in job shop problem solving using evolutionary algorithms (GA and others), so we cannot make meaningful comparisons (e.g., Qing-dao-er-ji & Wang, 2012). In addition, many researchers solved nonstandard test problems and some others solved only selected problems from the test sets (see Qiu & Lau, 2014). Many others studied different variants of job shop scheduling problems, for example, flexible, stochastic, or multiobjective job shop (Lei, 2011; Zhang et al., 2013; Demir & Isleyen, 2014). For the above reasons, we are unable to appropriately compare our approach with the existing algorithms.

We have compared our base algorithm with three different well-known approaches that did solve all the test problems we considered in our study (see Table 2). These algorithms are widely cited in the literature of conventional job shop scheduling. The algorithms are GRASP (Bianto et al., 2000), Shifting Bottleneck (Adams et al., 1994), and Local Search 1 & 2 (Aarts et al., 1988). GRASP (Bianto et al., 2000) is a metaheuristic for combinatorial optimization that is basically a greedy randomized adaptive search algorithm. The algorithm produces a reasonably good solution within a reasonable computational effort. The Shifting Bottleneck procedure developed by Adams et al. (1994) is an approximation approach where a local reoptimization method is applied when a bottleneck is identified in the scheduling process. The algorithm showed reasonably good performance in solving the complex test problems. Aarts et al. (1988) proposed two local search algorithms that were used with other metaheuristic algorithms, including a GA. According of Table 2, IMA outperforms all other algorithms compared.

Table 2. *Comparing our algorithm with other related algorithms*

Authors	Algorithm	Optimal Found	Average Deviation (%)	Fitness Deviation SD
Our algorithm	IMA	13	1.0471	1.6548
Hasan et al.	MA	11	1.1290	1.6659
Bianto et al.	GRASP	8	2.5631	3.2270
Adams et al.	SB	5	4.7644	4.1633
Aarts et al.	GLS1	3	2.6245	2.7781
Aarts et al.	GLS2	3	2.2739	2.2998

Note: MA, Memetic algorithm; IMA, improved memetic algorithm.

4.1. Rescheduling under machine breakdown

In this study, we consider disruptions caused by sudden machine breakdowns in the classical JSPs. In the case of sudden breakdown, it is required to reoptimize the affected tasks for the remaining operations from the time of breakdown. The situation becomes complex when there are multiple breakdowns.

The most obvious issue of a breakdown is that the broken machine cannot be used until it is either repaired or replaced. In the JSPs, the tasks are interrelated. Thus, if any task is incomplete due to a broken machine, the task must wait for a certain period of time. In reactive scheduling, the right-shifting strategy is normally applied to the affected tasks for the generation of a revised schedule. We instead apply the RM that minimizes the possible gaps left before pushing the tasks toward the right. Here, we make the following assumptions and definitions.

- Any task that needs to be relocated due to the interruption is classified as *affected*. The set of affected tasks is generated based on the precedence relationships of the tasks. For any instance, if the shifting of a task (because of breakdown) does not affect its successor task, then the successor task is not considered as affected.
- Reactive schedules consist of the affected tasks that begin from a revised starting time of each machine. The starting times are calculated from the finishing time of the completed tasks.

To analyze the reschedules under machine breakdown, we define a machine breakdown scenario by $\lambda(i, t, r)$, which indicates that a machine i that breaks at time t needs r units of time to be recovered. In this study, a breakdown instance is generated randomly. More specifically, we use a uniform distribution to identify the machine i, a Poisson distribution for t, and exponential distribution for r. For multiple breakdowns, for convenience of analysis, we divide the time line into several segments and generate the breakdown instances, one in each segment.

We solve the JSPs using IMA under ideal conditions. We generate a number of interruption scenarios λ as discussed earlier. We introduce these breakdowns to only the best solution obtained from IMA, and apply the RM alone to reoptimize the affected tasks.

4.2. Experimental study with breakdown scenarios

In this section, for ease of explanation, we define the following terms:

- Problem instance: each test problem (from 20 benchmark problems)
- Breakdown frequency: one, two, three, or four breakdowns

- Breakdown scenario: representing the parameters for λ(*i*, *t*, *r*) for a given breakdown frequency of a problem instance
- Approach: RM

Two sets of experimentation were performed. The purpose of the first set was to analyze the effect of the number of breakdowns on the overall makespan. The second set was designed to see the combined effect of breakdowns.

4.2.1. *Experiment 1: Different number of breakdowns*

In this experiment, analysis was performed for a fixed number of breakdown frequencies that varied from one to four. The steps of the first set of experimentation were the following:

1. Select a problem instance.
2. Select a breakdown frequency.
3. Generate 100 different breakdown scenarios.
    - Choose a machine or machines.
    - Choose start time for breakdown(s).
    - Choose breakdown duration(s).

For one breakdown, we choose λ(*i*, *t*, *r*): a machine, a downtime start time, and a repair duration using the distribution discussed earlier. For the two breakdowns, we choose two different machines, two nonoverlapping downtime start times (one for each machine), and their durations. Similarly, three and four nonoverlapping downtimes are generated for three and four machines, respectively. The length of machine breakdown for one downtime is assumed to be higher than each machine's downtime for the two downtimes. The downtime for each machine for the two downtimes is higher than the average downtime of each machine for the three downtimes case. We can make similar statements between the three and four downtime cases. In our experiments, we assume that the total downtime duration is nearly equal in all cases of the breakdowns, for convenience of judging the algorithm performance in terms of individual breakdown duration and frequency.

In the experiments, we pick the best solution from IMA, introduce breakdowns, and produce the reactive schedule. In the part of reactive scheduling, we apply RM to reoptimize the affected subsolution. For multiple breakdowns, we introduce the first breakdown and reoptimize the solution. Then we introduce the second breakdown, and so on. In this section, we have presented the results of four test problems, the last test problem from each of four groups, in Figure 1. In the figure the *x*-axis shows the number of breakdowns, and the *y*-axis represents the duration (the sum of breakdowns and the increase in makespan). The increase in makespan is equal to the difference between the makespan after rescheduling and the same under ideal conditions. These figures show that the rescheduled makespan decreases with the number of breakdowns, in situations where the total length of breakdowns is approximately equal for all breakdown frequencies. The revised (/rescheduling) makespan is clearly lower than the sum of the original makespan under ideal conditions and the total breakdown duration.

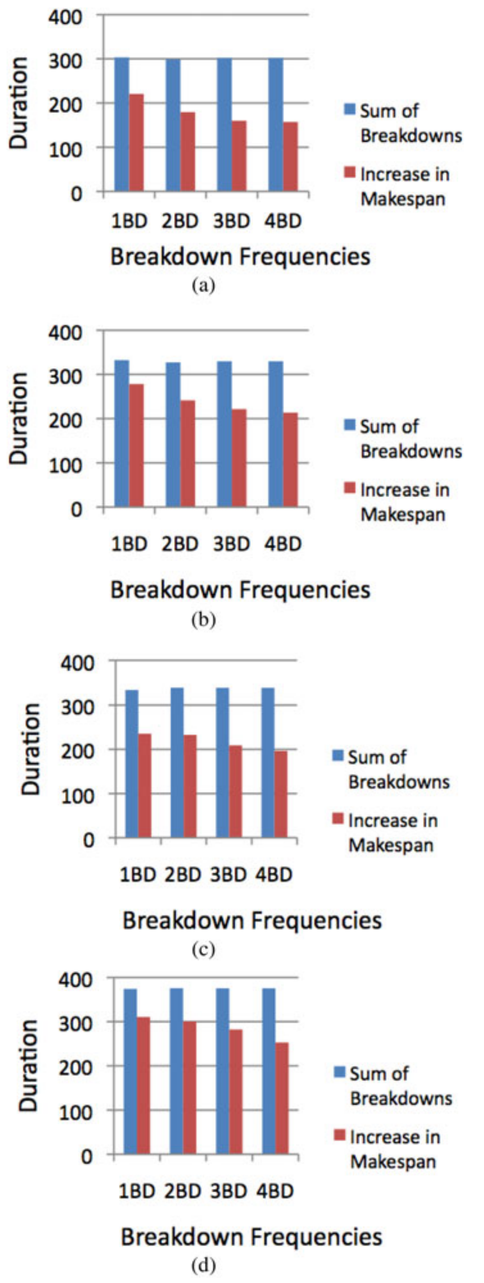

Fig. 1. (a) Problem La20, (b) problem La25, (c) problem La30, and (d) problem La35.

As discussed earlier, the total length of breakdowns for any reschedule is designed to be approximately equal. This means that the length of downtime in a one-breakdown case is approximately equal to the sum of all of the breakdowns in a four-breakdown case. It is interesting that the solution quality has a strong relationship with the frequency of breakdowns when the duration of the total breakdown is fixed. As evident in the above figures, multiple smaller breakdowns are better than one long breakdown, in terms of minimizing the loss due to breakdowns.

From our experiments, we have observed that the location of breakdowns within the scheduling duration influences the makespan of the reoptimized schedule. If a breakdown occurs during the early stage of a solution, the effected tasks could be rescheduled with only a small increase in makespan. This is because an early breakdown has more alternatives to reschedule the remaining jobs than does a later breakdown. To study this aspect, we have divided the makespan duration of each job into four equal time windows. In the study, one single breakdown is introduced in the first half of the first time window randomly, with an average downtime of 7% of the makespan, and the rescheduled makespan is recorded. The process is repeated 100 times for each time window. The average deviations from makespan for four test problems, considered for the breakdown frequencies above, are presented in Figure 2. The observed average deviations are 1.78%, 2.94%, 5.31%, and 6.67% from the makespan for time windows 1, 2, 3, and 4, respectively.

4.2.2. *Experiment 2: Combined effect of breakdowns*

In this experiment, the mix of the number of breakdowns is considered. This is because a practical situation may face different numbers of breakdowns at different time periods. The steps of the second set of experimentation were the following:

1. Select a problem instance.
2. Generate 100 different breakdown scenarios as follows.
3. For each scenario:
    - Choose a breakdown frequency randomly;
    - Choose a machine(s);
    - Choose start time for breakdown(s); and
    - Choose breakdown duration(s).

Other conditions are the same as Experiment 1. The results of four test problems (as selected in Experiment 1), in terms of average makespan, are presented in Table 3. Although, as expected, the average makespans are higher than the makespan under ideal conditions, they are a little lower than the average of the makespan calculated from the results of Experiment 1, and are much lower than the same obtained from the Right Shifting method. From the last column of Table 3, it is clear

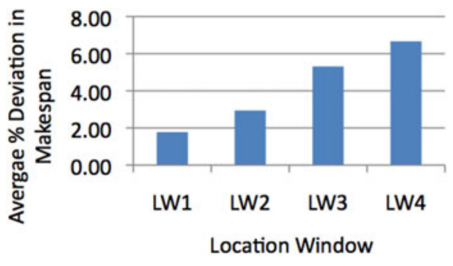

Fig. 2. The influence of the location of breakdowns.

Table 3. *Makespans after rescheduling*

Average Makespan				
Test Problem Number	Makespan Under Ideal Conditions	After Rescheduling With Proposed Approach (A)	After Rescheduling With Right Shifting Method (B)	% Difference = 100 × (A – B)/A
la20	902	979.34	1087.557	11.05
la25	977	1124.13	1275.325	13.45
la30	1355	1482.68	1690.848	14.04
la35	1888	1981.46	2318.11	24.49

that the percentage difference from the right-shifting techniques increases with the increase in size of the problems.

4.3. Summary of rescheduling approach

In Section 4, we have proposed a methodology for rescheduling jobs under machine breakdowns, and provided experimental studies with different machine breakdown scenarios and their analysis. Although machine breakdown is common in practice, for judging the performance of our approach, there is no test problem or practical data available in the literature. To generate the breakdown scenarios, we have introduced random breakdowns to the existing benchmark problems. The experiments were conducted to analyze the effect of a fixed number of breakdowns, which varied from one to four, on the makespan, as well as the combined effect of different numbers of breakdowns in the test set. For a fixed number of breakdowns, the revised makespan depends on the frequency of breakdowns, the duration of breakdowns, and the timing of breakdowns. From the experimental result analysis, it is found that an early breakdown has a lower effect on the makespan than a breakdown of similar duration at a later stage of the schedule, and multiple smaller breakdowns will have a lower effect than one long breakdown. The effect for mixed cases is somewhat an average of the fixed number of breakdowns.

5. MANAGING FINANCIAL RISK

In the previous two sections, we discussed the effect of sudden disruptions on different JSPs. In this section, we propose a procedure to manage the risk effectively in a manufacturing shop that receives different job orders from its customers and faces frequent machine disruptions. We assume, under ideal conditions, that the shop would accept N numbers of job orders. If the shop experiences any disruptions, it may not be able to accept all N job orders due to shop capacity and customer assigned due dates. Hence, it needs to develop a methodology for accepting or rejecting job orders that will maximize the benefit to the company or minimize the risk of losing revenue. We assume the company has the right to accept or reject any external job order.

A new job order would be accepted by the shop if it has the capacity to process it within the agreed delivery time. However, because of the stochastic nature of the disruptions, we assume the manufacturing shops accept a new job order, as a rule of thumb, if the estimated processing time [$=(1+\delta)\times$ (*average processing time of the job*)] is available before the delivery time. If the tolerance δ is too high, we would accept a lower number of job orders than expected, which would hence reduce the shop revenue. In contrast, a small δ may force some jobs to be tardy (late completion). We assume that any late completion would incur a penalty cost. Therefore, a question arises of what is the appropriate value of δ that would maximize the benefit to the shop under uncertain disruptions. To answer this question and analyze the relevant factors, we develop a simulation model and implement it using the data and results generated in the previous sections. The simulation model is briefly discussed below.

A manufacturing shop usually contains a fixed number of machines and receives a known variety of job orders. Initially, we made limited experimentation with 5, 10, 15, and 20 machines to observe their behavior. We have then fixed the number of machines to 10 (a reasonable number for practice) for a complete set of experiments because we have observed similar behavior with different numbers of machines. The 20 test problems studied earlier will be used as the mix of job orders. The job orders may be given to a manufacturing shop either randomly or following any particular pattern.

5.1. A simulation model

We have developed a simulation model to analyze the effect of δ on the revenue (or profit) of the shop. In other words, it will determine the value of the appropriate δ that would minimize the financial risk. In the simulation model, the external job orders (arriving based on a certain arrival pattern) join the queue for order acceptance and processing. The current order will be accepted if the estimated delivery time is earlier than or equal to the required delivery time as shown below:

$$\begin{aligned}&\mathrm{ED}t_{ij} \le \mathrm{RD}t_{ij},\\ &\mathrm{ED}t_{ij} = \max_k \mathrm{C}t_k + (1+\delta)\times \mathrm{AP}t_j,\end{aligned}$$

where $k \in \mathrm{AJ}$ and $k \le (i-1)$, $\mathrm{ED}t_{ij}$ is the estimated delivery time of the ith job of type j, $\mathrm{RD}t_{ij}$ is the required delivery time of the ith job of type j, $\mathrm{AP}t_j$ is the average processing time of job type j, $\mathrm{C}t_k$ is the completion time of job k (here $i=k$), δ is the allowable delivery tolerance, and AJ is a set of accepted jobs for processing.

We assume an order will be assigned for processing immediately after its acceptance (i.e., the earliest possible start time), if the shop is free. Otherwise, the accepted orders will be placed in a queue. The average processing time, for each job type, can be obtained from the shop's historical data. The actual processing time ($\mathrm{P}t_i$) for each job type is generated using a certain distribution that represents the shop's historical pattern (including disruption information). The actual delivery time ($\mathrm{AD}t_i$) is calculated and compared with the required delivery time ($\mathrm{RD}t_i$). If $\mathrm{AD}t_i > \mathrm{RD}t_i$, we also assume there will be a penalty cost (PC_i) based on the duration of delay.

$$\mathrm{PC}_i = \begin{cases} (\mathrm{AD}t_i - \mathrm{RD}t_i)\times \mathrm{UPC}_i, & \text{if } \mathrm{AD}t_i > \mathrm{RD}t_i \\ 0 & \text{otherwise}\end{cases}$$

Under this situation, the total revenue (TR) earned by the shop in a given time period can be calculated as follows:

$$\mathrm{TR} = \sum_i \{R_i - \mathrm{PC}_i\} - \sum_j \mathrm{OL}_j,$$

where UPC_i is the penalty cost per unit time, R_i is the earning (\$) for job i, OL_j is the opportunity loss (\$) for job j, and

$$R_i = \begin{cases} R_i, & \text{if job } i \text{ is accepted for processing} \\ 0 & \text{otherwise}\end{cases},$$

$$\mathrm{OL}_j = \begin{cases} \mathrm{OL}_j, & \text{if job } j \text{ is rejected for processing} \\ 0 & \text{otherwise}\end{cases}.$$

The TR equals the revenue from all accepted and delivered orders minus the penalty cost for delayed delivery minus the opportunity loss for orders rejected due to shortage of capacity. Although the opportunity loss component can be excluded from the total earning equation, its inclusion puts more pressure to accept more jobs by increasing the value of δ specifically for the scenarios with relatively lower penalty cost. Based on the organizational goal, one can define R_i as either profit or revenue.

Assume that **ShopSchedule** is a routine that receives the orders from a customer, calculates the feasibility, and processes the accepted job using its **Execute** subroutine. In addition, arrival of a job creates an event ARRIVE, while completion of a job creates an event FINISH. Finally, the program generates statistics depending on the status of the job and calculates overall revenue. The steps of the simulation model are briefly stated below.

```
function ShopSchedule

1. Set job number $i = 0$.
2. Get the current time $T$ from the system.
3. If $event$ = ARRIVE, then
    a. Set $i = i + 1$, get the job type $j$, order time $ORt_{ij}$, and
       required delivery time $RDt_{ij}$ of the arrived job order.
    b. Set the estimated delivery time $EDt_{ij}$.
    c. Calculate the actual processing time $Pt_{ij}$ using a nor-
       mal distribution (parameters shown in Appendix A)
       of the job type $j$.
    d. If $RDt_{ij} < ORt_{ij}$, then
        i. Set $status$ := DECLINED.
        ii. Go to Step 2.
    e. Else if $shopstat$ = FREE, then
        i. Start executing the job order immediately.
```

```
      f. Else
            i. Put the job order in the Queue for delayed processing.
         [End of Step 3.d If]
   4. Else if $event$ = FINISH, then
      a. Set $shopstat$ := FREE.
      b. Set the first job order in the queue as the current job order.
      c. Start executing the current job order.
      d. Delete the current job order from the Queue.
         [End of Step 2 If]
   5. Repeat Steps 1 to 4 until all the arrival job orders have been processed.
[End of function ShopSchedule]
```

function Execute (*Current, T*)

```
   1. Set the starting time of the current job as $T$.
   2. Set the actual delivery $ADt_i := T + Pt_i$.
   3. Calculate the penalty duration $PDt_i := ADT_i - RDt_i$.
   4. If $PDt_i > 0$, then
      a. Set $status$ := PENALTY.
   5. Else
      a. Set $status$ := NOPENALTY.
   [End of Step 4 If]
[End of function Execute]
```

In the above algorithm, "Shopstat := Free," means that the shop is free to start processing a job order.

5.2. Experimental study

The simulation model was coded in C++ and run on a PC. We have run the simulation model for 1000 job orders with 10 different δ (from 0.05 to 0.50 with an increment of 0.05). For the experiments, the job order arrivals, the type of job, and the required delivery times were generated randomly. The average job processing times were taken from previous experiments as reported in Appendix A. The actual processing times, for each job type, were generated using a normal distribution with average and standard deviation as shown in Appendix A. We found that the distribution of job processing times with machine disruption (as discussed in the previous section) approximately follows a normal distribution.

In a job shop business, some of the arriving jobs can be processed within their given due dates. Some other jobs can be delivered late with an agreed penalty per unit time delayed. If the delay is too long, the penalty will be higher, and that will reduce the overall revenue. Based on a given job arrival pattern, we have studied the level of delayed jobs that can be accepted for maximizing the company revenue. The level of delay is represented by the parameter δ in this study. We have experimented with different values of δ for different job arrival scenarios, and analyzed the job orders acceptance and rejection numbers and the overall revenues. For different δ values, the percentage of job orders accepted (with and without penalty) and job orders declined are presented in Table 4. The results show that the acceptance of job orders decreases with the increase of the δ value. This is because the penalty is higher with a higher value of δ, and that reduces the overall revenue. Although this behavior is expected, in this experiment, we are interested finding an appropriate value of δ that maximizes the earning of the company or minimizes the financial risk. To determine the best value of δ, we have assumed that an average earning from a successful delivery is \$500, an opportunity lost cost for a job order is \$50, and a penalty cost is \$0.50 per unit time per job order. The resulting TR values are plotted against δ values in Figure 3. From the experimental results, it shows that δ = 0.15 is the most appropriate value to minimize the financial loss, for a given arrival pattern, under uncertain disruptions. With a lower or higher value of δ = 0.15, the revenue will be reduced for the given capacity.

Table 4. *Accepted and declined jobs with varying δ*

	Average Percentage of Jobs			
δ	Total Accepted	Accepted With Penalty	Accepted Without Penalty	Declined
0.05	61	22.4	38.6	39
0.10	58	22.2	35.8	42
0.15	54.9	21.3	33.6	45.1
0.20	52.2	20.8	31.4	47.8
0.25	50.2	19.9	30.3	49.8
0.30	48.4	19.5	28.9	51.6
0.35	46.1	18.3	27.8	53.9
0.40	44	17.8	26.2	56
0.45	42.8	18.1	24.7	57.2
0.50	41.9	17.8	24.1	58.1

5.3. Summary of risk management

In Section 5, we have proposed a procedure that makes the acceptance/rejection decision of the arriving jobs as time passes, and at the same time, schedules all the accepted jobs, while considering that the system may be disrupted. In our approach, we intend to maximize the earning of the production system, which is basically minimizing the risk of losing revenue. In making the acceptance/rejection decision, an arriving job will be accepted if there is enough capacity within its tolerance limit. We have presented a numerical example to demonstrate the use of the proposed approach and experimented with different tolerance levels. It is interesting to report that a small tolerance is more appropriate than having either no or higher tolerance. We are not aware of any such research in the literature.

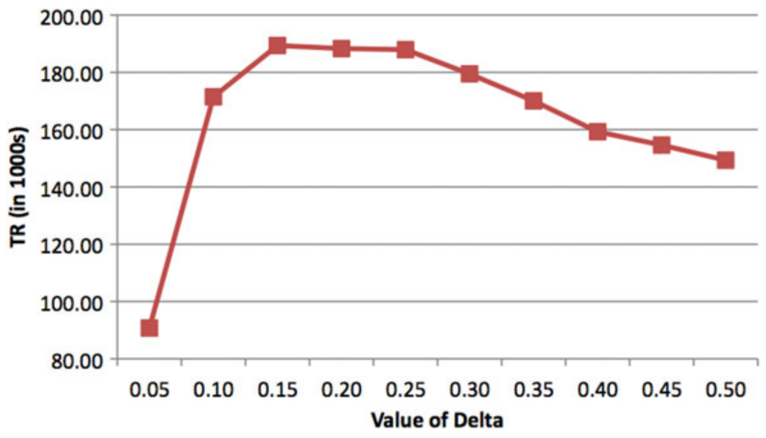

Fig. 3. Total revenue versus the values of delta.

6. CONCLUSIONS

The JSP is a well-known combinatorial optimization problem. A considerable amount of research has already been carried out to improve the algorithms for solving JSPs. Some of these algorithms are designed for specific cases of these problems, but still no algorithm guarantees optimality for all JSPs. In this paper, we have introduced an IMA that provides better solutions for JSPs than many existing algorithms. As compared to GA alone, we have shown that the IMA not only improves the performance of GA but also reduces the overall computational requirements.

In almost all research on JSPs, the schedules were produced under ideal conditions, assuming that there will be no interruptions of any type. We have used IMA as a base algorithm to study JSPs under sudden machine breakdowns, where the breakdown information is known after the actual breakdown takes place. We have solved and experimented with 20 benchmark test problems. For the condition of machine breakdowns, we have used different distributions to generate a number of breakdown scenarios. In this study, the experiments were conducted to analyze the effect of a fixed number of breakdowns on makespan. From the experimental result analysis, it is found that an early breakdown has a lower effect on makespan than a breakdown of similar duration at a later stage of the schedule, and multiple smaller breakdowns will have a lower effect than one long breakdown.

We have considered JSPs as a business case and developed a mathematical model to study the revenue earned under different scenarios and have also implemented a simulation model to find the best parameter set for maximizing the revenue. The simulation model is developed based on the job completion times under breakdowns. The model shows that the revenue can be maximized by carefully analyzing the different levels of risk. In this study, an arriving job is accepted if the processing time of the job is within a certain tolerance limit of the capacity. From the experiments with different tolerance levels, it shows that a small tolerance is more appropriate than having either no or a higher tolerance for minimizing financial risk. We believe this approach is unique and new in the literature.

REFERENCES

Aarts, E.H.L., Van Laarhoven, P.J.M., Lenstra, J.K., & Ulder, N.L.J. (1994). A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing 6*, 118–125.

Abumaizar, R.J., & Svestka, J.A. (1997). Rescheduling job shops under random disruptions. *International Journal of Production Research 35(7)*, 2065–2082.

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science 34(3)*, 391–401.

Binato, S., Hery, W., Loewenstern, D., & Resende, M. (2000). *A GRASP for Job Shop Scheduling*. Dordrecht: Kluwer Academic.

Blackstone, J.H., Jr, Phillips, D.T., & Hogg, G.L. (1982). A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research 20(1)*, 27–45.

Demir, Y., & Isleyen, S.K. (2014). An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations. *International Journal of Production Research 52(13)*, 3905–3921.

Fahmy, S.A., Balakrishnan, A., & ElMekkawy, T.Y. (2008). A generic deadlock-free reactive scheduling approach. *International Journal of Production Research 46(1)*, 1–20.

Ghasem, M., & Mehdi, M. (2011). A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. *International Journal of Production Economics 129(1)*, 14–22.

Hasan, S.M.K., Sarker, R., & Essam, D. (2011). Genetic algorithm for job-shop scheduling with machine unavailability and breakdown. *International Journal of Production Research 49(16)*, 4999–5015.

Hasan, S.M.K., Sarker, R., Essam, D., & Cornforth, D. (2009). Memetic algorithms for solving job-shop scheduling problems. *Memetic Computing 1(1)*, 69–83.

Lawrence, S. (1985). *Job Shop Scheduling with Genetic Algorithms: First International Conference on Genetic Algorithms*, pp. 136–140. Mahwah, NJ: Erlbaum.

Lei, D. (2011). Simplified multi-objective genetic algorithms for stochastic job shop scheduling. *Applied Soft Computing 11(8)*, 4991–4996.

Liu, S.Q., Ong, H.L., & Ng, K.M. (2005). Metaheuristics for minimizing the makespan of the dynamic shop scheduling problem. *Advances in Engineering Software 36(3)*, 199–205.

Meeran, S., & Morshed, M.S. (2012). A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. *Journal of Intelligent Manufacturing 23(4)*, 1063–1078.

Nakano, R., & Yamada, T. (1991). Conventional genetic algorithm for job shop problems. *Proc. Fourth Int. Conf. Genetic Algorithms*, pp. 474–479. San Mateo, CA: Kaufmann.

Nasr, A.-H., & ElMekkawy, T.Y. (2011). Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. *International Journal of Production Economics 132(2)*, 279–291.

Paredis, J. (1992). Exploiting constraints as background knowledge for genetic algorithms: a case-study for scheduling. In *Parallel Problem Solving from Nature*, Vol. 2, pp. 229–238. Amsterdam: North-Holland.

Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research 35(10)*, 3202–3212.

Ponnambalam, S.G., Aravindan, P., & Rao, P.S. (2001). Comparative evaluation of genetic algorithms for job-shop scheduling. *Production Planning & Control 12(6)*, 560–674.

Qing-dao-er-ji, R., & Wang, Y. (2012). A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research 39(10)*, 2291–2299.

Qiu, X., & Lau, H.Y.K. (2014). An AIS-based algorithm for static job shop scheduling problem. *Journal of Intelligent Manufacturing 25*, 489–503.

Subramaniam, V., Raheja, A.S., & Reddy, K.R.B. (2005). Reactive repair tool for job shop schedules. *International Journal of Production Research 43(1)*, 1–23.

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

Yamada, T. (2003). *Studies on metaheuristics for jobshop and flowshop scheduling problems.* PhD Thesis, Kyoto University, Department of Applied Mathematics and Physics, pp. 1–120.

Yamada, T., & Nakano, R. (1997). Genetic algorithms for job-shop scheduling problems. In *Modern Heuristic for Decision Support*, pp. 67–81. UNICOM seminar, London.

Zhang, L., Gao, L., & Li, X. (2013). A hybrid genetic algorithm and tabu search for a multi-objective dynamic job shop scheduling problem. *International Journal of Production Research 51(12)*, 3516–3531.

Ruhul Sarker is an Associate Professor in the School of Engineering and Information Technology, University of New South Wales. He received his PhD in operations research from the Department of Industrial Engineering, Dalhousie University, Halifax, Canada. Dr. Sarker has published more than 230 papers in high-impact journals, edited books, and international conference proceedings. His research interests include applied operations research and evolutionary optimization.

Daryl Essam is a Senior Lecturer in the School of Engineering and Information Technology, University of New South Wales. He obtained his PhD in computer science from the University of New South Wales. Dr. Essam's research covers design and analysis of evolutionary algorithms and optimization problem solving using evolutionary algorithms. He has published in numerous international high-impact journals and conference proceedings.

S.M. Kamrul Hasan currently works as a Software Consultant with RungePincockMinarco, Brisbane, Australia. He completed his PhD in computer science at the University of New South Wales, where he then worked in the School of Engineering and Information Technology. His research mainly covers scheduling using evolutionary algorithms. Dr. Hasan has published five journal papers and several papers in IEEE conference proceedings.

A.N. Mustafizul Karim is a Professor in the Department of Manufacturing and Materials Engineering, International Islamic University Malaysia. He received his PhD in manufacturing from Dublin City University, Ireland. His research interests include production scheduling, supply chain optimization, and engineering economics. Dr. Karim obtained two large competitive research grants from the Ministry of Education, Government of Malaysia, for conducting research on real-world industrial problems. He has been widely published in the industrial and manufacturing engineering field.

APPENDIX A

Experimental results with random breakdown scenarios

Problems	Optimal	Average Solution	Standard Deviation
la16	945	1027.35	35.062
la17	784	884.90	30.776
la18	848	961.00	34.131
la19	842	975.75	36.963
la20	902	979.34	28.804
la21	1046	1212.91	37.077
la22	927	1111.35	23.984
la23	1032	1162.98	25.556
la24	935	1128.60	27.958
la25	977	1124.13	25.353
la26	1218	1406.39	26.364
la27	1235	1433.55	26.594
la28	1216	1420.29	32.242
la29	1157	1378.33	25.824
la30	1355	1482.68	31.444
la31	1784	1874.68	30.164
la32	1850	2003.95	26.221
la33	1719	1812.67	28.489
la34	1721	1863.48	32.370
la35	1888	1981.46	25.162