Int J Adv Manuf Technol (2013) 67:1127–1137
DOI 10.1007/s00170-012-4553-x

ORIGINAL ARTICLE

Scheduling a single machine with multiple preventive maintenance activities and position-based deteriorations using genetic algorithms

Byung Soo Kim · Yucel Ozturkoglu

Received: 8 February 2012 / Accepted: 2 October 2012 / Published online: 13 November 2012

Abstract In this paper, we study a single machine scheduling problem with deteriorating processing time of jobs and multiple preventive maintenances which reset deteriorated processing time to the original processing time. In this situation, we consider three kinds of problems whose performance measures are makespan, total completion time, and total weighted completion time. First, we formulate integer programming formulations, and using the formulations, one can find optimal solutions for small problems. Since these problems are known to be NP-hard and the size of real problem is very large, we propose a number of heuristics and design genetic algorithms for the problems. Finally, we conduct some computational experiments to evaluate the performance of the proposed algorithms.

Keywords Scheduling · Preventive maintenance · Deterioration · Genetic algorithms · Discrete optimization

1 Introduction

In several decades, a number of researchers have considered problems with task processing time being known parameters in deterministic scheduling problems. However, the task processing time may change due to various activities (i.e., human or machine' fatigue, learning effects, etc.) in real manufacturing area. In this case, the processing times of tasks are variable depending on starting time of the task or position of the task. Several scheduling studies with nonconstant processing time have received increasing attention in recent years. In the nonconstant processing time, Gupta and Gupta [10] and Brown and Yechiali [2] first introduced scheduling problem with task deterioration. In this study, the actual processing time of tasks is modeled as an increasing function of its starting time due to deterioration effects. In order to make the analysis possible, most research models the actual processing time of a task as a linear or piecewise linear increasing function of its starting time. Gupta and Gupta [10] introduced a scheduling model with variable processing time of a job which is a polynomial function of its initial processing time. Browne and Yechiali [2] discussed deteriorating jobs and their processing times increase while they await service. They assumed that the actual processing time is p_i+a_it, where p_i is the original processing time, a_i is the growth rate of the processing time, and t is the start time of job i, and the processing time of the job increases linearly with its starting time t. This problem is called scheduling with *time-dependent deterioration*. It reflects a variety of real-life situations such as steel production, resource allocation, firefighting, maintenance or cleaning, etc. (see Kunnathur and Gupta, [13, 19]) in which any delay in processing a job may result in an increasing effort to accomplish the job. Kubiak and Vende [12] investigated the computational complexity of makespan under deterioration. They developed a heuristic and branch-and-bound algorithm for the problem. Kovalyov and Kubiak [11] presented a fully polynomial approximation scheme for a single machine scheduling problem to minimize makespan of deteriorating jobs. They showed that sequencing the jobs in an increasing order of p_i/a_i minimizes the makespan on single machine scheduling problem. Mosheiov [18] considered the problem of minimizing total flow time of jobs with the actual processing time p_i+a_it. He found that the optimal sequence of this problem is V-shaped. The V-shaped scheduling indicates that jobs are arranged in descending order of

B. S. Kim (✉)
Graduate School of Management of Technology,
Pukyong National University,
Busan 608-737, South Korea
e-mail: iekbs@pknu.ac.kr

Y. Ozturkoglu (✉)
International Logistics Management, Yasar University,
Izmir, Turkey
e-mail: yucel.ozturkoglu@yasar.edu.tr

growth rate if they are placed before the minimal growth rate job and in ascending order if placed after it. Cheng and Ding [4] studied a single machine to minimize makespan with deadlines and increasing rates of processing times. They found that both problems are solvable by a dynamic programming algorithm. Bachman et al. [1] proved that total weighted completion time is non-deterministic polynomial-time (NP)-hard for single machine scheduling in which the job processing times are decreasing linear functions dependent on their start times.

Scheduling problems have less attention to include the possibility of inserting preventive maintenance activity into the job sequence. Lee and Leon [16] introduced the concept of *rate-modifying activity* to the scheduling literature. A rate-modifying activity (RMA) is any activity that alters the speed in which a resource performs tasks. Therefore, the preventive maintenance activity is a good example of RMAs. They assume that one rate-modifying activity which changes the production rate of equipment is included during the planning horizon. Given a set of jobs to be performed by a resource and an RMA of fixed length that will recover the processing rate of the resource, they determine (a) the sequence in which the jobs should be performed and (b) when to schedule the fixed-length RMA so that the objective of the scheduling is optimized. In this study, they do not include the possibility of machine breakdown. They developed polynomial algorithms for solving minimizing both makespan and total completion time. Lee and Lin [15] considered single machine scheduling problems involving *repair* and *maintenance* activities. They derived optimal policies for scheduling fixed-length RMAs and job sequencing in an environment by machine breakdowns. In particular, the machine breakdown is modeled by the stochastic process. They focused on two types of processing cases, *resumable* and *nonresumable*. If the RMA is scheduled before a breakdown, then job processing times are reduced. If breakdown occurs, a repair activity whose duration is a random variable is immediately applied, and the resource's normal processing time is resumed for the remainder of the planning horizon. The objective functions were minimizing the expected makespan, total expected completion time, maximum expected lateness, and expected maximum lateness, respectively. Grave and Lee [9] presented a single machine scheduling problem where the objective was to minimize the total weighted completion time of jobs. However, this study is limited in that only one maintenance activity can be performed during the planning horizon. Lee and Chen [14] extended to parallel machines, but they are still limited on single maintenance activity allowed. Qi et al. [22] considered a single machine problem with the possibility for multiple maintenance activities, but during scheduling period, they ignore the deterioration of processing time for jobs. Cassady and Kutanoglu [3] developed an integrated stochastic model for a single machine problem with total weighted expected completion time as the objective function. Their model allows multiple maintenance activities and explicitly captures the risk of not performing maintenance. Sortrakul et al. [23] developed a genetic algorithm for the integrated stochastic optimization model for production scheduling and multiple preventive maintenance activities by Cassady and Kutanoglu [3].

Despite the trade-offs between the two activities such as deterioration of processing time for jobs and preventive maintenance, most studies are typically planned independently or neglected in real production scheduling even if productivity can be improved by optimizing both production scheduling and preventive maintenance decisions simultaneously. Lodree and Geiger [17] integrated time-dependent processing times and a maintenance activity by inserting a rate-modifying activity into the job sequence. They presented that a single maintenance must be inserted in the middle of the optimal job sequence to minimize makespan.

This paper deals with a deterministic scheduling problem which is proposed by Ozturkoglu and Bulfin [20]. They integrated multiple preventive maintenance activities and position-based deterioration of jobs, where position-based deterioration of jobs determines the actual processing time of a job depending on the position of the job sequenced. We propose several heuristics including hybrid genetic algorithm to solve the problem of Ozturkoglu and Bulfin [20] in three objective functions: makespan, total completion time, and total weighted completion time.

The following section contains an integer programming formulation of the problem and a simple example problem. In Section 4, heuristic algorithms for the problems are considered, including genetic algorithm. In Section 5, we report the results of computational experiments to compare the performance of algorithms proposed. Finally, we conclude the paper with a summary and some directions for future research in Section 6.

2 Integer programming formulation

In this paper, we model single machine problem with multiple preventive maintenance activities and position-based deteriorations. First, we formulate the problem minimizing makespan or minimizing total weighted completion time as an integer linear programming. Before we introduce the formulation, the assumptions we make about this system are as follows:

1. There is only one worker in single machine.
2. The deterioration of a job depends on its job sequence.
3. Preemptive jobs are not permitted.

4. After a preventive maintenance, the worker recovers completely, which means the processing time of jobs returns to an original processing time.
5. Deterioration process is the same after a preventive maintenance is completed.

The definition of index and notation for the formulation are as follows:

N	Number of jobs
i	Index for ith position from the previous maintenance
j	Index for job
k	Index for position for the most recent preventive maintenance
α	Deterioration rate for jobs, where $0<\alpha\leq 1$ when delayed by one position
t	Preventive maintenance period
A	The set of jobs after job i and j
B	The set of jobs before job i and j
p_j	Initial processing time of job j
p_{ij}	Deteriorated processing time for job j assigned in ith position if a maintenance is executed in just before initial position, where, $k\leq i$
w_j	Weight of job j
C_i	Completion time of the job assigned in ith position

$$p_{ij} = (1+\alpha)^{i-1} p_j$$

and the integer programming uses the following variables

$$x_{ijk} = \begin{cases} 1, & \text{if job j is assigned to ith position under the most recent maintenance is executed in} \\ & k\text{th position, where } k \leq i \\ 0, & \text{otherwise} \end{cases}$$

$$y_i = \begin{cases} 1, & \text{if a preventive maintenance is performed before ith position,} \\ 0, & \text{otherwise} \end{cases}$$

Using the above parameters and variables, we can represent our problem of minimizing makespan or minimizing total completion time to an integer linear programming formulation as follows:

$$\text{Min } \mathrm{C_n} \text{ or } \sum_{\mathrm{i=1}}^{\mathrm{N}} \mathrm{C_i}, \tag{1}$$

subject to

$$\sum_{i=1}^{N}\sum_{k=1}^{i} x_{ijk} = 1 \qquad j = 1, 2, \ldots, N \tag{2}$$

$$\sum_{j=1}^{N}\sum_{k=1}^{i} x_{ijk} = 1 \qquad i = 1, 2, \ldots, N \tag{3}$$

$$x_{ijk} \leq y_k \qquad j,k = 1,\ 2, \ldots, N \quad i = k, \ldots, N, \tag{4}$$

$$C_1 = \sum_{j=1}^{N} p_{1j} x_{1j1} \tag{5}$$

$$C_i = C_{i-1} + \sum_{k=1}^{i}\sum_{j=1}^{N} p_{(i-k+1)j} x_{ijk} + t y_i, \quad i = 2, \ldots, N, \tag{6}$$

$$x_{ijk}, y_i \in \{0,1\} \qquad i,j = 1,\ 2, \ldots, N \qquad k = 1,\ 2, \ldots, i \tag{7}$$

Equation (1) means that this problem is to minimize makespan or minimize total weighted completion time. Equation (2) assures that each job is assigned to exactly one position. Equation (3) guarantees that each position is scheduled for only one job. Equation (4) assures that if any job j is assigned to any kth position under the most recent maintenance is executed in ith position, a preventive maintenance must be performed in ith position. By Eqs. (5) and (6), the completion time for all the positions can be calculated. Equation (7) shows that the variables are 0–1 binary integers. Cheng and Ding [4] proved that single machine scheduling problem with step-deteriorating processing time and identical deterioration date by reductions from partition, which is known as strongly NP-hard [5]. According to Graham [8], the problem they studied can be represented by $1/p_i = a_i$ or $p_i = a_i + b_i$, $d_i = d/\gamma$, where a_i, b_i, and d_i are normal processing time, deterioration penalty, and the deteriorating due date of job i, respectively, and the objective function γ is $C_{\max}$, $\sum C_i$ or $\sum w_iC_i$. We assume that $a_{[i]}$, $b_{[i]}$, and $d_{[i]}$ are normal processing time, deterioration penalty, and the deteriorating due date of ith job. If $d=0$ and $p_{[i]}=a_{[i]}$ if $s_{[i]}\le d$ or $p_{[i]}=a_{[i-1]}+b_{[i]}$, otherwise, where $b_{[i]}=\alpha a_{[i-1]}$, the problem is equivalent to our problem in this paper except considering a number of preventive maintenances. Thus, the problems are even more difficult due to the existence of the multiple numbers of preventive maintenances during planning horizon. Therefore, our problem is NP-hard, too.

3 Heuristics

In this section, we propose several heuristics for single machine problem with multiple preventive maintenance activity and position-based deteriorations. Three performance measures are used for our system which consists of makespan, total completion time, and total weighted completion time. According to Graham [8], the problem studied in this paper can be represented by $1\left|p_{ij}=(1+\alpha)^{i-1}p_j,\mathrm{pm}\right|\gamma$ which the symbol α represents positive deteriorating rate, p_{ij} means a deteriorated processing time for job j assigned in ith position if the maintenance is executed just before initial position, pm describes preventive maintenances, and γ represents an objective function among $C_{\max}$, $\sum C_i$, or $\sum w_iC_i$.

3.1 Makespan problem

In this section, we consider minimizing makespan on single machine problem with multiple preventive maintenance and exponential job deterioration. Since this problem is NP-hard [4], an efficient optimal solution procedure unlikely exists and one needs to search for effective heuristic procedures. In this section, we propose several heuristics for minimizing makespan, i.e., $1\Big/p_{ij}=(1+\alpha)^{i-1}p_j,\mathrm{pm}/C_{\max}$. If there is no maintenance, i.e., $1\Big/p_{ij}=(1+\alpha)^{i-1}p_j/C_{\max}$, the following theorem can be introduced.

Proposition 3.1: Longest processing time (LPT) sequencing minimizes the makespan on single machine problem with exponential job deterioration.

Proof: Since $p_{nj}=(1+\alpha)^{n-1}p_j$ and suppose schedule S minimizes makespan and is not in LPT order, there must be a pair of jobs in S, say job i and job j, and job i is immediately after job j they are in position $n-1$ and position n, and $p_i<p_j$, where $i\ne j$, $0<i\le N$, $0<j\le N$, and $0<n\le N$. Let TC(B) be the total completion time of the jobs before job i and j and TC(A) be the total completion time of the jobs after job i and j. Now, consider the schedule S', where S' is the same as schedule S except job i and j have been interchanged and the sets of jobs A and B are in the same position in both schedules. The total completion time for S is

$$\begin{aligned}\mathrm{TC(S)}&=\mathrm{TC(B)}+\mathrm{p}_{(\mathrm{n}-1)\mathrm{i}}+\mathrm{p}_{\mathrm{nj}}+\mathrm{TC(A)}\\&=\mathrm{TC(B)}+(1+\alpha)^{\mathrm{n}-2}\mathrm{p_i}+(1+\alpha)^{\mathrm{n}-1}\mathrm{p_j}+\mathrm{TC(A)},\end{aligned}$$

and the total completion time for S' is

$$\begin{aligned}\mathrm{TC}\left(\mathrm{S}'\right)&=\mathrm{TC(B)}+\mathrm{p}_{(\mathrm{n}-1)\mathrm{j}}+\mathrm{p}_{\mathrm{ni}}+\mathrm{TC(A)}\\&=\mathrm{TC(B)}+(1+\alpha)^{\mathrm{n}-2}\mathrm{p_j}+(1+\alpha)^{\mathrm{n}-1}\mathrm{p_i}+\mathrm{TC(A)}.\end{aligned}$$

Subtracting, we get

$$\mathrm{TC(S)}-\mathrm{TC}\left(\mathrm{S}'\right)=\alpha(1+\alpha)^{\mathrm{n}-2}\left(\mathrm{p_j}-\mathrm{p_i}\right)>0.$$

This implies that the makespan of S' is smaller than S, which contradicts the assumption that S was optimal. Therefore, an optimal solution must be in LPT.

Since LPT sequencing minimizes the makespan on single machine problem with exponentially growing position-based job deterioration, we propose two local search heuristics using LPT sequencing. We called the first heuristic as largest processing time and preventive maintenance ratio (LPTRT) and we called the second heuristic as largest processing time and bucket assignment (LPTBK). First, we propose a basic heuristic which generates random schedule with random preventive maintenance (RSRM) to compare the performance of the heuristics and genetic algorithm.

Algorithm RSRM

```
Step 1: Sort the jobs in a random order.
Step 2: If there are no remaining jobs in the sorted list to be placed in the current position, stop.
Step 3: Place the first job of the sorted list to the position.
Step 4: Generate a binary random number from [0, 1] for the existence of a preventive maintenance is in the current position.
Step 5: If the number is 1, set $p_{ij}=p_j$, or set $p_{ij}=(1+\alpha)^{i-1}p_j$, otherwise. Calculate the completion time of the job $j$ using $p_{ij}$. Update $i=1$, if the number is 1, or update $i=i+1$, otherwise.
Step 6: Remove the job from the sorted list and go to step 2.
```

RSRM is based as a benchmark for comparison with the heuristics developed in this paper. Based on the RSRM, we propose the second heuristic, LPTRT heuristic. In this heuristic, the decision to perform a preventive maintenance to current position is determined by deterioration ratio $r = \pi / t_{\mathrm{m}}$, where π is the cumulative deterioration and t is the maintenance period. The cumulative deterioration means the sum of deterioration periods of the succeeding jobs assigned between the position of the previous maintenance and current position. If the deterioration ratio (r) is more than 1, a preventive maintenance needs to be in front of the current job because it is more time efficient to include a preventive maintenance and recover original processing time of jobs when r is more than 1. The detail heuristic procedure is as follows:

Algorithm LPTRT

```
Step 1: Sort the jobs in LPT order and set $\pi$=0.
Step 2: If there are no remaining jobs in the sorted list to be placed in the current position, stop.
Step 3: Place the first job from the sorted list.
Step 4: If $r$>1, include a preventive maintenance in front of the current position and $p_{ij}$=$p_j$ or set $p_{ij} = (1+\alpha)^{i-1} p_j$, otherwise. Calculate the completion time of the job $j$ using $p_{ij}$.
Step 5: Calculate $\delta_j = p_{ij} - p_j$ and update $\pi$ by using equation $\pi = \pi + \delta_{ij}$.
Step 6: Remove the assigned job from the sorted list and go to step 2.
```

LPTRT constructively determines the positions of the preventive maintenance by comparing the cumulative deterioration to the preventive maintenance period. In this case, it is difficult to place more jobs with large processing time to the front location in the preventive maintenance positions. To assign jobs with larger processing time to front location in each maintenance position, we propose LPTBK. In this heuristic, we first find the required number of preventive maintenances, N using the deterioration ratio r in LPTRT. We generate N buckets to contain a number of positions to place jobs in each bucket. A preventive maintenance is executed in front of the first position in each bucket. The number of positions for the jobs is evenly divided in each bucket. Then, we select the first N jobs in the list of LPT order and place the first position in each N bucket. Then, we select next N jobs in LPT order and place the second position in each N bucket until all the jobs in the list are assigned to appropriate slots in N buckets. The detail heuristic procedure is as follows:

Algorithm LPTBK

```
Step 1: Sort the jobs in LPT order.
Step 2: Find bucket size $N$ by calculating $r$ in LPTRT, where $N$ is increased by 1, whenever $r$>1.
Step 3: Select the first $N$ jobs in the sorted list and place the first available position in each bucket.
Step 4: If there are no remaining jobs in the sorted list to be placed in the current position, stop.
Step 5: Remove the first $N$ jobs from the sorted list and go to step 3.
Step 6: Calculate the completion time of jobs assigned in $N$ bucket.
```

3.2 Total completion time and total weighted completion time problem

The total completion time is a special case of the total weighted completion time, e.g., if we set all weights in the total weighted completion time case to one, it reduces to the total completion time problem. Therefore, we study these two problems in the current section together. Since these problems with no maintenance were proved to be NP-hard [4], the current problem is NP-hard too. Hence, it is needed to search for effective heuristic procedure.

In contrast to the makespan problem, it is known that shortest processing time (SPT) and weighted SPT rules work better than LPT in the total completion time problem and the total weighted completion time problem, respectively. In single machine problem with multiple preventive maintenances and exponential position-based deterioration, however, we have to consider not only the actual processing time of jobs considering position-based job deterioration but also the positions of preventive maintenances. Using this basic concept, we propose a simple greedy heuristic.

Algorithm GRD

```
Step 1: Sort the jobs in $p_j / w_j$ order and set $\pi$=0.
Step 2: Use steps 2, 3, 4, 5, and 6 of LPTRT.
```

Mosheiov [18] considers the problem of minimizing total completion time in single machine that the actual processing time of each job grows linearly with its starting time. In this case, a different rate of growth is associated with each job. He shows that an optimal sequence to minimize total completion time is V-shaped: jobs are arranged in descending order of growth rate if they are placed before the minimal growth rate job and in ascending order if placed after it. Since the growth rate exponentially depends on the processing time of each job in this paper, we follow the general heuristic procedure using the processing time and the weight instead of the growth rate proposed by Mosheiov [18].

Algorithm VGRD

```
Step 1: Use step 1 of greedy heuristic (GRD).
Step 2: Use step 2 of LPTBK.
Step 3: Assign $\lfloor n/N \rfloor$ number of slots in each bucket and if $k = n - N \times \lfloor n/N \rfloor$ exists, assign an additional slot from the first to $k$th bucket.
Step 4: Open an available bucket. Place first job in the sorted list into the first slot in the bucket, place next job
```

in the sorted list into the last slot in the bucket, place next job in the sorted list into the second slot in the bucket, and place next job in the sorted list into the next to the last slot in the bucket. Repeat this procedure until jobs are assigned in the every slot in the bucket.

Step 6: If there are buckets to be scheduled, go to *step 4*, otherwise stop.

Although these procedures are made for total weighted completion time, it can easily be applied to total completion time by setting all weights to one, i.e., $w_j=1$.

4 Genetic algorithms

The genetic algorithms, which have been widely used variously for three decades, are stochastic search algorithms based on the mechanism of natural selection and natural reproduction process of genetics. Being different from the conventional search techniques, they start with an initial set of (random) solutions called a population. Each individual in the population is called a chromosome, representing a solution to the problem at hand. The chromosomes evolve through successive iterations, called generations. During each generation, the chromosomes are evaluated using some measures of fitness. Generally speaking, the genetic algorithm is applied to spaces that are too large to be exhaustively searched [7].

4.1 Representation and initialization

The proper representation of a solution plays a critical role in the development of a genetic algorithm. Sortrakul et al. [23] generate two separate chromosomes: one for the job sequence and the other for the existence of preventive maintenance. Following Sortrakul et al. [23], we also use the two separate chromosome representation. However, we use the random key generation representation to generate the job sequence chromosome, in which K random numbers from [0, 1] are used as keys to represent a sequence of K integers in the range of (1, K) [24]. Since the random key representation eliminates the infeasibility of the offspring chromosome as well as representing solutions in a soft manner, this representation is applicable to a wide variety of sequencing optimization problems (i.e., machine scheduling, resource allocation, travel salesman problem, quadratic assignment, vehicle routing, etc.) [6]. For the initial preventive maintenance, we first generate total number of the preventive maintenances (N) using the ratio of total deterioration period to preventive maintenance period in LPTBK in Section 3.1. Then, we randomly select the N positions in the range of [1, K], where $0 \le N \le K$. Thus, our initial solution set can be obtained by simple random number generation from [0, 1] for the job sequence and 0–1 binary number generation for preventive maintenance existence by adding 1 into N position between first position and Kth position. In our example, stated in earlier sections, there are ten jobs. The sets of processing times of each job are (5.01, 7.82, 5.97, 9.04, 7.93, 7.4, 6.75, 9.48, 9.11, and 8.73) from [0, 10]. Preventive maintenance period and deterioration rate for jobs are $t=6.0$ and $\alpha=0.2$. Under this condition, suppose that we have a chromosome generated from ten random key numbers from [0, 1], it can be a chromosome representing for the job sequence, e.g., (0.174, 0.859, 0.711, 0.514, 0.304, 0.015, 0.0914, 0.364, 0.147, and 0.166). From this chromosome, we sort the random key numbers in ascending order. Based on the sequence of the sorted genes, we can generate job index in each position, which results in (5, 10, 9, 8, 6, 1, 2, 7, 3, and 4). Based on the job index using random key generation, we can find the processing time of each job, e.g., the processing time of job 5 is 5.01, the processing time of job 10 is 7.82, and the processing time of job 4 is 8.73. For chromosome initial representation for preventive maintenance, we first sort processing time of jobs in LPT order. The sorted jobs and their processing times of sorted job are (7, 3, 8, 4, 6, 10, 1, 2, 9, 5) and (9.48, 9.11, 9.04, 8.73, 7.93, 7.82, 7.40, 6.75, 5.97, 5.01), respectively. Using the LPTBK heuristic procedure, we calculate N. Table 1 presents the procedure for generating N.

Once we found the number of preventive maintenances as $N=3$, we randomly generate three positions from [1, 10]. If the randomly generated positions are 3, 5, and 8, we generate a chromosome for preventive maintenance as follows: (0, 0, 1, 0, 1, 0, 0, 1, 0, and 0).

4.2 Objective and fitness function

The flexibility of objective function is one of the most powerful characteristics in genetic algorithm. In the genetic algorithms, whether the objective function is linear or not is not important. So, we can use any equations for makespan or total completion time for objective function. Since all the problems

Table 1 Generation procedure for N

j	p_j	δ_j	π	N
7	9.48	0.00	0.00	0
3	9.11	1.82	1.82	0
8	9.04	3.98	5.80	0
4	8.73	0.00	0.00	1
6	7.93	1.59	1.59	1
10	7.82	3.44	5.03	1
1	7.40	0.00	0.00	2
2	6.75	1.35	1.35	2
9	5.97	2.63	3.98	2
5	5.01	0.00	0.00	3

under consideration are minimization problems, however, we have to convert the objective function values to fitness function value of maximization form. To get the fitness value for a chromosome (i.e., F_i) from the objective function value of the chromosome (i.e., Z_i) and the maximum objective function value among the population (i.e., $Z_{\max}$) as follows:

$$F_i = (Z_{\max} - Z_i) \Bigg/ \sum_{i=1}^{n} (Z_{\max} - Z_i). \tag{8}$$

4.3 Crossover and mutation and reproduction

A simple genetic algorithm that yields good results in many practical problems is composed of three essential operators: crossover, mutation, and reproduction. The crossover operator takes two chromosomes and swaps a part of genes containing their genetic information to produce new chromosomes. The easiest and the most classical method for crossover is to choose a random cut-point and generate the offspring by combining the segment of one parent to the left of the cut-point with the segment of the other parent to the right of the cut-point. When some representations, like the permutation representation, are used in the sequencing problem, this one-cut-exchange crossover can hardly be applied since the offspring from the crossover may be illegal. Due to using the random key representation for chromosome representing job sequence, we can use the one-cut-exchange crossover without violating feasibility. For binary representation for preventive representation, we can also use the one-cut-exchange crossover without violating feasibility.

Mutation produces spontaneous random changes in various chromosomes. This genetic operation serves the crucial role of replacing the genes lost from population during the selection process so that they can be tried in a new context or providing the genes that were not present in the initial population. In our algorithm, we also use the simplest method, in which a gene is selected by a very small probability and replaced with another random number from [0, 1] for chromosome representing job sequence and with other binary number for chromosome representing preventive maintenance.

Reproduction is a process in which individual chromosomes are proportionally copied from their fitness function value. We use the most popular method that is referred to as roulette wheel reproduction where each current string in the population has a roulette wheel slot sized in proportion to its fitness. Adopting the elitist strategy, the two best chromosomes are directly copied to the next generation. The detailed procedure to generate the next generation is as follows:

Step 1: Copy two best sets of chromosomes (a chromosome for job sequence and a chromosome for preventive maintenance)

Step 2: Select two sets of chromosomes by roulette wheel method. If a random number from [0, 1] is less than a given crossover probability, i.e., p_c, generate two sets of chromosomes by one-cut-exchange crossover or copy the two sets of chromosomes directly, otherwise. Repeat this step until the next generation is constructed.

Step 3: For every gene in the set of chromosomes generated at step 2 excluding the best set of chromosomes, if a random number from [0, 1] is less than a given mutation probability, i.e., p_m, then replace that gene in the chromosome for job sequence with another random number from [0, 1] and replace that gene in the chromosome for preventive maintenance with the other binary value.

5 Computational experiments

To evaluate the performance of heuristic algorithms presented in this paper, computational experiments have been conducted using randomly generated problems. Since the complexity of a problem depends on the number of jobs (N), preventive maintenance period (t), and deterioration rate (α), we control these parameters to several levels, in which n can be one of three values (50, 100, 150, and 200), t can be one of three values ($5\overline{p}$, $10\overline{p}$, and $15\overline{p}$), where $\overline{p} = \sum_{j=1}^{n} p_j$, and α can be one of four values (0.01, 0.03, 0.05, and 0.07). For each combination of out of these 36 combinations, we randomly generate 20 problems. Here, problem generation means the generation of p_j and w_j for each job. The value of each p_j and w_j is U (10, 30) and U (0, 1), where U (a, b) means a random real number generated from interval (a, b).

All solution approaches have been coded in C++ and run on 2.17 GHz, Intel Core 2 Duo CPU with 4 GB of memory and Windows Vista operating system. Under these conditions, the CPU time to get a solution from a heuristic algorithm excluding genetic algorithm is shorter than 1 ms. But computing times for genetic algorithms are rather long and depend highly on the number of jobs (i.e., N). Table 2 shows the computation time s when we set the population size to 20 and the maximum generation is 1,000. The value in of each cell in this table represents the average CPU time (in seconds) from 360 problems (30 random problems in three levels of t and four levels of α).

Table 2 CPU time (in seconds) for GA

n	50.00	100.00	150.00	200.00
$C_{\max}$	0.88	2.77	5.66	9.56
SUMC	0.97	3.16	6.68	11.41
SUMWC	0.98	3.18	6.70	11.48

SUMC sum of total completion time; *SUMWC* sum of total weighted completion time

In the comparison with other heuristics, the computation times for genetic algorithm are very long, which vary from 0.88 to 11.48 s in average. But, from the practical viewpoint, 11.48 is not very long time to schedule the jobs for a week or a month. So, we think that the genetic algorithm can be also a reasonable alternative to get a schedule provided that it produces a good result.

5.1 Results for makespan

Table 3 shows the results for makespan, which include the results for LPTRT and LPTBK heuristics and the genetic algorithm (GA). In small deterioration rate, LPTRT and LPTBK heuristics give reasonably good solutions compared to GA. Both LPTRT and LPTBK heuristics are focused on an appropriate number of preventive maintenances and their positions rather than job sequences. Since the number of preventive maintenances increases, LPTRT and LPTBK heuristics give a good performance. However, GA is better than LPTRT and LPTBK heuristics as the deterioration rate increases and the number of jobs becomes larger. The reason for this is there are more chances to have large number of jobs between preventive maintenances. In other words, the number of maintenances decreases as the number of jobs and the deterioration rate increases. In this case, there are many alternatives to construct jobs between preventive maintenances. Therefore, the sequence of jobs between preventive maintenances is relatively more critical than the decision of the position of maintenances for solution performance. As a result, GA performs better than other heuristics because the problem becomes a larger solution space as the number of jobs between preventive maintenances.

5.2 Results for total completion time

The results of GRD, V-shaped greedy heuristic (VGRD), and GA for minimizing total completion time are presented in Table 4. In this table, VGRD heuristic is better than GRD and GA for minimizing total completion time. The reason for this is that we provide V-shaped job sequence by Mosheiov [18] between preventive maintenances. In the case of a single machine problem, Mosheiov [18] proved that the actual processing time of each job grows linearly with its starting time to minimize total completion time and there is an optimal sequence for minimizing total completion time. In this problem, since the jobs between preventive

Table 3 Performance comparison for makespan

t	α		n=50			n=100			n=150			n=200		
			LPTRT	LPTBK	GA	LPTRT	LPTBK	GA	LPTRT	LPTBK	GA	LPTRT	LPTBK	GA
$5\overline{p}$	0.01	Mean	26	26	26	25	25	24	25	25	24	25	25	24
		SD	0.539	0.537	0.532	0.442	0.44	0.448	0.313	0.321	0.332	0.284	0.285	0.291
	0.03	Mean	20	20	19	20	20	17	20	20	17	19	19	16
		SD	0.634	0.633	0.612	0.401	0.403	0.393	0.363	0.367	0.378	0.276	0.279	0.301
	0.05	Mean	17	17	16	17	16	14	17	16	13	16	16	12
		SD	0.539	0.54	0.536	0.383	0.394	0.408	0.369	0.37	0.379	0.32	0.317	0.354
	0.07	Mean	15	15	13	15	14	11	15	14	10	14	14	9
		SD	0.441	0.445	0.483	0.337	0.343	0.323	0.291	0.296	0.373	0.313	0.309	0.387
$10\overline{p}$	0.01	Mean	51	53	53	52	53	52	51	51	50	51	51	50
		SD	0.761	0.701	0.702	0.457	0.454	0.468	0.498	0.494	0.496	0.314	0.31	0.332
	0.03	Mean	46	46	46	44	44	43	43	44	41	44	44	41
		SD	0.73	0.724	0.749	0.578	0.573	0.59	0.527	0.529	0.569	0.484	0.462	0.463
	0.05	Mean	41	41	40	39	39	36	41	40	37	40	40	35
		SD	0.786	0.784	0.799	0.563	0.58	0.624	0.459	0.465	0.538	0.378	0.389	0.479
	0.07	Mean	39	38	37	37	37	34	36	36	32	36	36	30
		SD	0.749	0.759	0.757	0.737	0.762	0.753	0.435	0.426	0.504	0.402	0.402	0.495
$15\overline{p}$	0.01	Mean	66	67	67	64	65	64	64	64	63	63	63	62
		SD	0.728	0.72	0.716	0.422	0.419	0.426	0.383	0.371	0.386	0.284	0.281	0.3
	0.03	Mean	60	60	60	57	57	56	57	57	55	57	57	54
		SD	0.711	0.713	0.717	0.506	0.509	0.549	0.484	0.49	0.541	0.336	0.334	0.36
	0.05	Mean	56	56	55	53	53	51	52	52	49	51	51	47
		SD	0.731	0.728	0.773	0.559	0.556	0.575	0.417	0.422	0.48	0.451	0.453	0.489
	0.07	Mean	50	51	49	48	48	45	48	48	44	49	48	43
		SD	0.75	0.773	0.821	0.649	0.637	0.71	0.549	0.55	0.619	0.309	0.314	0.38

Table 4 Performance comparison for total completion time

t	α		n=50			n=100			n=150			n=200		
			GRD	VGRD	GA	GRD	VGRD	GA	GRD	VGRD	GA	GRD	VGRD	GA
$5\overline{p}$	0.01	Mean	25	26	29	23	25	26	23	24	25	23	24	24
		SD	0.7927	0.7821	0.7296	0.528	0.4847	0.441	0.3811	0.4887	0.3637	0.3824	0.3624	0.3106
	0.03	Mean	19	20	23	19	20	20	18	18	18	18	19	18
		SD	0.9712	0.949	0.8452	0.5303	0.5499	0.5524	0.4871	0.4086	0.3665	0.4175	0.3824	0.36
	0.05	Mean	15	17	18	15	16	16	15	16	14	14	16	13
		SD	0.6915	0.6825	0.6191	0.5121	0.5371	0.4963	0.3672	0.421	0.4289	0.3205	0.3388	0.3531
	0.07	Mean	13	14	16	12	14	13	12	13	11	13	14	10
		SD	0.7514	0.8731	0.6619	0.5679	0.5482	0.4714	0.4232	0.3986	0.3744	0.3582	0.3918	0.4116
$10\overline{p}$	0.01	Mean	51	53	54	52	52	53	50	51	51	50	51	50
		SD	0.912	0.9073	0.833	0.6376	0.6559	0.6598	0.4878	0.4661	0.501	0.4778	0.4439	0.4736
	0.03	Mean	45	47	48	43	43	43	43	43	42	43	44	42
		SD	1.243	1.165	1.226	0.6073	0.6322	0.625	0.5825	0.5704	0.5388	0.4106	0.4474	0.4256
	0.05	Mean	40	41	42	39	40	38	39	40	37	38	39	35
		SD	1.054	0.9548	0.9492	0.633	0.6194	0.6231	0.6621	0.6124	0.6039	0.5281	0.5445	0.5884
	0.07	Mean	38	40	40	37	39	36	35	36	33	35	35	31
		SD	0.8175	0.7683	0.7378	0.6185	0.6078	0.6261	0.4771	0.448	0.496	0.4822	0.5268	0.5143
$15\overline{p}$	0.01	Mean	64	65	66	63	64	64	64	64	64	63	63	63
		SD	0.7898	0.6928	0.6932	0.5831	0.5519	0.5612	0.5339	0.4908	0.5109	0.4187	0.4195	0.3927
	0.03	Mean	59	60	61	57	58	57	56	57	56	57	57	55
		SD	0.672	0.5632	0.5891	0.5876	0.5431	0.5449	0.5644	0.5498	0.5418	0.5076	0.5219	0.5336
	0.05	Mean	53	55	55	52	53	51	51	52	49	51	52	48
		SD	1.167	1.089	1.088	0.5822	0.5941	0.5896	0.52	0.5256	0.4703	0.4449	0.434	0.4854
	0.07	Mean	52	53	53	47	49	47	48	49	45	47	48	44
		SD	1.019	1.003	1.018	0.8698	0.8747	0.8954	0.629	0.642	0.6976	0.409	0.3804	0.4058

maintenances are deteriorated and the objective function in this problem is minimizing total completion time, the job sequences between the preventive maintenances in VGRD heuristic give good performance. In the table, the GA provides the worst results among the heuristics in spite of the long CPU time. The differences of mean values of VGRD and GA in this table vary from 0 to 8 %. If this machine is a bottleneck of the production line, 8 % of difference seems to be very significant one. For example, if we are to schedule jobs for 30 days, we can save 2.1 days. From the viewpoint of theory of constraints, this saving results in a 0 to 8 % improvement of the plant capacity without any investment.

5.3 Results for total weighted completion time

The results for minimizing total completion time are presented in Table 5. In this table, we summarize the results of GRD, VGRD, and GA. In fact, this problem generalizes the problem in minimizing total completion time. In this problem, VGRD heuristic provides relatively poor performance unlike the problems for minimizing total completion time. This table shows that GRD and GA are better than VGRD. But one of the GRD and GA does not dominate the other. If the number of jobs becomes smaller, the GRD heuristic becomes better than GA. As the number of jobs becomes larger, GA performs better than GRD. This means that GA performs well in large solution space. Furthermore, GA gives a lower standard deviation of percent improvement compared to GRD and VGRD. This result provides that GA provides robust performance in various control parameters. The mean percent improvement values of GA vary from 15 to 70 %, which means that if we use the GA heuristic during 30 days of scheduling to minimize total weighted completion time, we can save 4.5 to 21 days of the time in comparison with RSRM heuristic.

6 Conclusions

Deterioration of processing times of jobs and preventive maintenance planning are in common and significant problems in real production scheduling areas. Ozturkoglu and Bulfin [21] showed that human or machine fatigue results in the deterioration of processing time of the jobs. The preventive maintenance recovers the machine's fatigue and prevents the potential risk for machine failure. However, excessive

Table 5 Performance comparison for total weighted completion time

t	α		n=50			n=100			n=150			n=200		
			GRD	VGRD	GA	GRD	VGRD	GA	GRD	VGRD	GA	GRD	VGRD	GA
$5\overline{p}$	0.01	Mean	24	24	32	23	23	29	23	23	28	25	24	28
		SD	1.569	1.446	0.9887	1.089	0.7069	0.6676	0.6327	0.7419	0.4286	0.6697	0.7263	0.5867
	0.03	Mean	20	21	28	18	19	23	20	19	22	20	19	22
		SD	1.159	1.248	0.7835	1.24	1.311	0.9359	0.9149	0.9273	0.6987	0.6639	0.7859	0.572
	0.05	Mean	17	16	25	16	15	21	16	16	19	15	15	17
		SD	1.795	1.251	1.192	1.202	1.282	0.8644	0.839	1.023	0.6902	0.7823	0.792	0.6257
	0.07	Mean	16	15	22	12	12	16	16	16	17	14	14	15
		SD	1.326	1.239	0.9352	0.867	0.9519	0.6357	0.8705	0.8599	0.6258	0.6069	0.6838	0.4964
$10\overline{p}$	0.01	Mean	53	50	58	50	51	55	50	49	53	51	51	53
		SD	0.9847	1.221	0.8424	0.7654	0.8974	0.6003	0.6054	0.7213	0.5144	0.472	0.5309	0.3847
	0.03	Mean	43	42	49	44	43	46	43	42	45	44	43	44
		SD	1.345	1.352	0.9782	0.9549	1.031	0.8692	0.9951	0.7956	0.8535	0.6203	0.5323	0.4915
	0.05	Mean	40	38	46	40	39	43	38	38	39	37	37	38
		SD	1.514	1.445	1.231	0.8667	1.008	0.7482	0.9089	0.9031	0.7335	0.7656	0.7282	0.6753
	0.07	Mean	38	36	44	37	36	40	36	35	36	34	33	33
		SD	1.423	1.274	1.212	1.108	0.9051	0.8922	0.8524	0.9269	0.7963	0.776	0.6495	0.5736
$15\overline{p}$	0.01	Mean	66	63	70	62	61	66	63	63	65	62	62	64
		SD	0.9778	1.349	0.9179	0.8537	0.8272	0.6968	0.6614	0.6764	0.5126	0.4218	0.4174	0.3668
	0.03	Mean	59	58	65	57	55	60	57	56	58	56	55	57
		SD	1.243	1.368	0.7887	0.8044	1.01	0.5782	0.8603	0.8981	0.6636	0.6477	0.7195	0.5427
	0.05	Mean	53	51	60	50	48	53	49	48	51	51	50	51
		SD	1.351	1.355	0.9762	1.269	1.436	1.17	0.7481	0.9906	0.5913	0.6705	0.6892	0.6094
	0.07	Mean	49	47	55	48	47	51	48	46	47	49	48	49
		SD	1.347	1.703	1.226	0.9162	1.222	0.7393	0.8391	0.6933	0.7793	0.5795	0.6353	0.5233

preventive maintenance results in too much maintenance time. In this paper, we studied scheduling of jobs and the planning of the preventive maintenance under the job scheduling. We presented a simple integer programming formulation to minimize makespan, total completion time, or total weighted completion time. Since the problem is NP-hard and the size of a real problem is very large, we proposed a number of heuristic algorithms and a genetic algorithm to solve large-scale problems in a reasonable computational time.

The results of computational experiments show that the heuristics have different performances for the problems and some heuristics provide very good solutions for particular problems in reasonable CPU time. From the results, we highly recommend that one should use GA for makespan compared to LPTRT and LPTBK heuristics as the deterioration rate increases and the number of jobs becomes large. In the case of the total completion time, VGRD heuristic is better than GRD and GA. However, VGRD heuristic does not perform well for the total weighted completion time and one of the heuristics cannot dominate the other. In this case, GRD heuristic works better than GA, as the number of jobs becomes smaller. If the number of jobs becomes larger, GA performs better than GRD.

Several directions for future research are apparent from this study. In this study, we have not reflected the machine failure time yet. In general, the risk of machine failures increases if preventive maintenances do not execute appropriately. Since the machine failure happens stochastically, it is difficult to model to express job deterioration. If we include machine failure in the current model, more realistic study can be expected. Secondly, this study can be extended in the scheduling framework with human learning curve and preventive maintenance. In general, task processing time of workers is decreased by the human learning effect. If the worker does not perform preventive maintenance appropriately, the risk of failure increases so that more failures happen stochastically during scheduling. As a result, the task processing time increases because the worker frequently takes breaks from the task and learning effect is diminished. Using the trade-off between learning curve, preventive maintenance, and risk of failure, we can propose an optimal schedule. Finally, the current study can be extended to research for the parallel machine problem with identical machines.

References

1. Bachman A, Janiak A, Kovalyov MY (2002) Minimizing the total weighted completion time of deteriorating jobs. Inf Process Lett 81 (2):81–84
2. Browne S, Yechiali U (1990) Scheduling deteriorating jobs on a single processor. Oper Res 38:495–498
3. Cassady CR, Kutanoglu E (2005) Integrating preventive maintenance planning and production scheduling for a single machine. IEEE Trans Reliab 54(2):304–309
4. Cheng TCE, Ding Q (2000) Single machine scheduling with deadlines and increasing rates of processing times. Acta Informatica 36:673–692
5. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
6. Gen M, Cheng R (1997) Genetic algorithms and engineering design. Wiley, New York
7. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, New York
8. Graham RL, Lawler EL, Lenstra JK, Rinnooy KAHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326
9. Graves GH, Lee CY (1999) Scheduling maintenance and semi-resumable jobs on a single machine. Nav Res Logist 46:845–863
10. Gupta JND, Gupta SK (1988) Single facility scheduling with nonlinear processing times. Comput Ind Eng 14:387–393
11. Kovalyov YM, Kubiak W (1998) A fully polynomial approximation scheme for minimizing makespan of deteriorating jobs. Journal of Heuristics 3:287–297
12. Kubiak W, Vende S (1998) Scheduling deteriorating jobs to minimize makespan. Nav Res Logist 45:511–523
13. Kunnathur AS, Gupta SK (1990) Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. Eur J Oper Res 47:56–64
14. Lee CY, Chen ZL (2000) Scheduling of jobs and maintenance activities on parallel machines. Nav Res Logist 47:61–67
15. Lee CY, Lin CS (2001) Single-machine scheduling with maintenance and repair rate-modifying activities. Eur J Oper Res 135:493–513
16. Lee CY, Leon VJ (2001) Machine scheduling with a rate-modifying activity. Eur J Oper Res 128:119–128
17. Lodree EJ, Geiger CD (2009) A note on the optimal sequence position for a rate-modifying activity under simple linear deterioration. Eur J Oper Res 201(2):644–648
18. Mosheiov G (1991) V-shaped polices to scheduling deteriorating jobs. Operation Research 39:979–991
19. Mosheiov G (1995) Scheduling jobs with step-deterioration: minimizing makespan on a single- and multi-machine. Comput Ind Eng 28:869–879
20. Ozturkoglu Y, Bulfin R (2011) A unique integer mathematical model for scheduling deteriorating jobs with rate-modifying activities on a single machine. Int J Adv Manuf Technol 57:753–762
21. Ozturkoglu Y, Bulfin R (2012) Scheduling jobs to consider physiological factors. Human factors and ergonomics in manufacturing & service industries. Wiley, New York, pp 113–120, 22 (2)
22. Qi X, Chen T, Tu F (1999) Scheduling the maintenance on a single machine. J Oper Res Soc 50:1071–1078
23. Sortrakul N, Nachtmann CR, Cassady CR (2005) Genetic algorithms for integrated preventive maintenance planning and production scheduling for a single machine. Computers In Industry 56:161–168
24. Wang CS, Uzsoy R (2002) A genetic algorithm to minimize maximum lateness on a batch processing machine. Comput Oper Res 29:1621–1640