IEMS Vol. 10, No. 4, pp. 247-254, December 2011.

Single-Machine Total Completion Time Scheduling with Position-Based Deterioration and Multiple Rate-Modifying Activities

Byung Soo Kim
Department of Systems Management and Engineering, Pukyong National University.
599-1 Daeyeon 3-Dong, Nam-Gu, Busan, 608-737, Korea
Tel: +82-51-629-6491, E-mail: iekbs@pknu.ac.kr

Cheol Min Joo†
Department of Systems and Management Engineering, Dongseo University
San 69-1, Jure-Dong, Sasang-Gu, Busan, Korea
Tel: +82-51-320-1715, E-mail: cmjoo@dongseo.ac.kr

Received, March 23, 2011; Revised, July 13, 2011; Accepted, September 7, 2011

Abstract. In this paper, we study a single-machine scheduling problem with deteriorating processing time of jobs and multiple rate-modifying activities which reset deteriorated processing time to the original processing time. In this situation, the objective function is to minimize total completion time. First, we formulate an integer programming model. Since the model is difficult to solve as the size of real problem being very large, we design an improved genetic algorithm called adaptive genetic algorithm (AGA) with spontaneously adjusting crossover and mutation rate depending upon the status of current population. Finally, we conduct some computational experiments to evaluate the performance of AGA with the conventional GAs with various combinations of crossover and mutation rates.

Keywords: Single-Machine Scheduling, Rate-Modifying Activity, Deterioration, Adaptive Genetic Algorithm

1. INTRODUCTION

In the classical scheduling problems, it is usually assumed that processing times for different jobs are constant within the planning horizon. However many practical have revealed that the processing time is an increasing function of the sequence of jobs or the starting time of jobs. It relates to many industrial applications such as scheduling maintenance or cleaning tasks, where any delay in starting to process a job increases its completion time (Kunnathur and Gupta, 1990; Mosheiov, 1995). This phenomenon known as deterioration, has been extensively studied in recent years. Gupta and Gupta (1988) and Brown and Yechiali (1990) independently initiated research on the scheduling problem with deteriorating jobs or time-dependent processing times. Kubiak and Vende (1998) studied the NP-hardness proof of makespan for single machine scheduling under deterioration. They developed a heuristic and branch-and-bound algorithm for the problem. Kovalyov and Kubiak (1998) 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 increasing order in the ratio of processing time to deterioration rate of jobs minimizes the makespan on single machine scheduling problem. Mosheiov (1991) considered the problem of minimizing total flow time of jobs with a linear deterioration of the processing time such as the processing time plus the deterioration rate multiplied by time consumed. He found that the optimal sequence of this problem is V-shaped. The V-shaped scheduling defines that 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. Cheng and Ding (2000) 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.* (2002) gave the NP-hardness proof of total weighted completion time for single machine scheduling in which the job processing times are decreasing linear functions dependent on their start times.

† : Corresponding Author

Recently, Wang *et al.* (2011) studied single-machine total completion time scheduling with a time-dependent deterioration. They proposed two heuristic algorithms utilizing the V-shaped property. Since the complexity of the problem remains open, they compared the worst case error bound using SPT with two proposed heuristics.

Meanwhile, rate-modifying activity (RMA) is any activity that alters the speed in which a resource performs jobs. The preventive maintenance activity is a good example of RMAs. Lee and Leon (2001) introduced the concept of (RMA) to the scheduling literature. 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 (i) the sequence in which the jobs should be performed and (ii) when to schedule the fixed length RMA so that the objective of the scheduling is optimized. They developed polynomial algorithms for solving minimizing both makespan and total completion time. Lee and Lin (2001) 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 non-resumable. 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 (1999) 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 a single maintenance activity can be performed during the planning horizon. Lee and Chen (2000) extended to parallel machines, but they are still limited on single maintenance activity allowed. Qi *et al.* (1999) considered a single machine problem with the possibility for multiple maintenance activities, but during sche-duling period they ignored the deterioration of processing time for jobs. Cassady and Kutanoglu (2005) 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.

This paper deals with a single-machine scheduling problem with position-based deterioration and multiple rate modifying activities to minimize total completion time. The position-based deterioration of jobs determines the actual processing time of a job depending upon the position of the job sequenced. In Section 2, we provide the mathematical formulation of the problem. In Section 3, heuristic algorithms for the problem are considered. Section 4 describes a new genetic algorithm called adaptive genetic algorithm (AGA). In Section 5, the results of computational experiments are presented to compare the performance of all proposed algorithms. Finally, we conclude the paper with a summary in Section 6.

2. MODEL FORMULATION

Before developing a mathematical formulation of the problem, the parameters and decision variables are introduced as follows:

Parameters

N	number of jobs
α	deterioration rate for jobs, where $0 < \alpha \le 1$
Q	RMA time
p_j	initial processing time of job j
p_{jm}	deteriorated processing time for job j if a RMA is executed at the end of m positions ahead from the current position assigned to job j, which is calculated by equation (1)

$$p_{jm} = (1+\alpha)^{m-1} p_j \text{, for } 0 < m \le N-1 \tag{1}$$

Decision variables

x_{ijk}	equal to 1, if job i is assigned to position j with the most recent RMA being executed at the beginning of k^{th} position, where $k \le j$, 0, otherwise;
y_k	equal to 1, if a RMA is executed at the beginning of k^{th} position, 0, otherwise

Depending variable

C_i	completion time of the job assigned in i^{th} position.

The integer programming (IP) can be formulated as follows:

$$\textit{Minimize } z = \sum_{j=1}^{N} C_j \tag{2}$$

subject to

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

$$\sum_{i=1}^{N}\sum_{k=1}^{j} x_{ijk} = 1, \qquad \text{for } j = 1, 2, \cdots, N, \tag{4}$$

$$x_{ijk} \le y_k, \qquad \text{for } i, k = 1, 2, \cdots, N, \quad j = k, \cdots, N, \tag{5}$$

$$C_1 = \sum_{i=1}^{N} p_{i1} x_{i11}, \tag{6}$$

$$C_i = C_{i-1} + \sum_{k=1}^{j}\sum_{i=1}^{N} p_{i(j-k+1)} x_{ijk} + Q y_i \text{, for } i = 2, \cdots, N, \tag{7}$$

$$C_i \ge 0, \qquad \text{for } i = 1, \cdots, N, \tag{8}$$

$$x_{ijk} \in \{0, 1\}, \quad \text{for } i, j = 1, 2, \cdots, N, \; k = 1, 2, \cdots, j, \tag{9}$$

$$y_1 = 1, \tag{10}$$

$$y_i \in \{0, 1\}, \quad \text{for } i = 2, \cdots, N. \tag{11}$$

Constraint (3) assures that each job is assigned to exactly one position. Constraint (4) guarantees that each position is scheduled for only one job. Constraint (5) assures the relation between x_{ijk} and y_k, in which y_k should be one if any job assigned to position *j* is executed in a RMA in position *k*. From constraints (6) and (7), the completion time for all assigned jobs can be calculated.

In the IP model, there are $\left(N^3 - N^2\right)/2 + N - 1$ binary variables and $N^3 - N^2 + 3N$ constraints. Although the model is quite solvable for problems of small sizes using CPLEX package, it is obvious that a heuristic alternative will be of great interest to provide a near-optimal solution within a limited computing time for large size problem in many applications. Thus, this paper is focused on developing effective heuristic algorithms instead.

3. TWO-PHASE HEURISTIC

A two-phase heuristic algorithm (TPHA) is proposed in this section. The number of RMAs is determined in the first phase of the algorithm, and then finds the job sequence in a bucket between consecutive RMAs using the shortest processing time job schedule (SPT) in the second phase. For classical $1 || \sum C_j$, the shortest processing time job schedule (SPT) on a single machine with zero release times are optimal schedule to total completion time. In order to solve the problem $1 | (1+\alpha)^{m-1} p_j, 0 < \alpha < 1 | \sum C_j$, the SPT provides also an optimal solution.

Theorem 1: *SPT minimizes* $1 | (1+\alpha)^{m-1} p_j, 0 < \alpha < 1 | \sum C_j$.

Proof: Suppose schedule *S* minimizes total completion time and is not in SPT order, there must be a pair of jobs in *S*, say job *i* and job *j*, and job *i* is immediately before job *j*, which are in position *m*-1 and position *m*. The processing times of job *i* and job *j* are $p_i > p_j$, where $i \neq j, 0 < i \leq N, 0 < j \leq N$. Now consider the schedule *S*, where *S* is the same as the schedule *S* except job *i* and *j* have been interchanged. Let *A* be the set of jobs after job *i* and *j* and *B* be the set of jobs before job *i* and *j*, *t* be the completion time of the last job in *B* and *n* be the position of the job. Assume that jobs in *A* and *B* are same position in both schedules *S* and *S*. Then $TC(A)$ is the total completion time of *A* and $TC(B)$ be the total completion time of *B*. The total completion time for *S* is

$$\begin{aligned} &TC(S) \\ &= TC(B) + \left(t + p_{(n-1)i}\right) + \left(t + p_{(n-1)i} + p_{nj}\right) + TC(A) \\ &= TC(B) + \left[t + (1+\alpha)^{n-2} p_i\right] \\ &\quad + \left[t + (1+\alpha)^{n-2} p_i + (1+\alpha)^{n-1} p_j\right] + TC(A), \end{aligned}$$

and the total completion time for *S* is

$$\begin{aligned} &TC\left(S'\right) \\ &= TC(B) + \left(t + p_{(n-1)j}\right) + \left(t + p_{(n-1)j} + p_{ni}\right) + TC(A) \\ &= TC(B) + \left[t + (1+\alpha)^{n-2} p_j\right] \\ &\quad + \left[t + (1+\alpha)^{n-2} p_j + (1+\alpha)^{n-1} p_i\right] + TC(A). \end{aligned}$$

By subtracting, we get the value of $TC(S) - TC(S')$ as $(1-\alpha)(\alpha+1)^{n-2}(p_i - p_j) \geq 0$. This implies the total completion time of S is smaller than or equal to *S*, which contradicts the assumption that S is optimal. Therefore, an optimal job schedule must be in SPT. □

In the first phase of the algorithm, we decide the number of buckets with deterioration ratio, $r = \pi/Q$, where π is the cumulative deterioration value of the job allocating order using largest processing time with deterioration ratio (LPTRT). The cumulative deterioration value means the sum of deteriorations from the position executing the previous RMA to the current position. If r is more than 1, a RMA is added and the number of buckets is increased by one. It reduces the total completion time, because the RMA time is smaller than the recovering time by less deterioration of jobs with the RMA. Since the SPT provides an optimal schedule on a single machine total completion time scheduling without RMAs, we also use the SPT to determine the job sequence in buckets in the second phase of the algorithm. The jobs sorted by the SPT is assigned to *k* buckets determined by phase I, in which each first *k* jobs sequentially is allocated into first available position in each bucket. The detail algorithm is as follows:

Algorithm: TPHA
(Phase I: Determination of the number of RMAs)

```
Step 1: Sort the jobs with LPTRT and make the sorted
        list.
        Set $\pi$ = 0, current position, $m$ = 1, and number
        of buckets, $k$ = 1
Step 2: If there are no remaining jobs in the sorted list,
        then stop.
Step 3: Select the first available job from the sorted list
        and calculate $r$.
Step 4: If $r > 1$, update $p_{jm} = p_j$ and set $\pi$ = 0, $m$ = 1,
        and $k = k + 1$.
        Otherwise, update $p_{jm} = (1+\alpha)^{m-1} p_j$ and $\pi = \pi$
        $+ \delta_j$, where $\delta_j = p_{jm} - p_j$, and set $m = m+1$.
Step 5: Remove the assigned job from the sorted list
        and go to Step 2.
```

(Phase II: Job sequencing in buckets)

Step 1: Sort the jobs with SPT and make the sorted list.

Step 2: Select the first *k* jobs in the sorted list

Step 3: Sequentially assign one of the *k* jobs to the first available position in each *k* bucket.

Step 4: If there are no remaining jobs in the sorted list, calculate the total completion time and stop, remove the first *k* jobs from the sorted list and go to Step 2, otherwise.

4. ADAPTIVE GENETIC ALGORITHM

The genetic algorithm (GA), which has been widely used variously for three decades, is stochastic search algorithms based on the mechanism of natural selection and the reproduction process of genetics. Being different from the conventional search techniques, GA starts 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 (Goldberg, 1989).

In the conventional GAs, we predetermine several parameters (i.e., the number of populations, the number of generations, crossover rate and mutation rate) before the experimental tests are executed. Among these parameters, the crossover and mutation rates are closely related with the performance of the solution by affecting the exploration of solution space and the convergence of a solution. If the crossover rate increases, GA intensively searches neighborhood solutions but it increases the more possibility to converge local optima. If the mutation rate increases, GA explores the solution space but it decreases the convergence of the solution. In this paper, we propose a new genetic algorithm called adaptive genetic algorithm (AGA), in which the crossover and mutation rates are spontaneously adjusted by the solution performance in each generation, during the algorithm.

4.1 Representation and Initialization

The proper representation of a solution plays a critical role in the development of a genetic algorithm. For the scheduling with preventive maintenances, Sortarakul *et al*. (2005) represented two separate chromosomes, one for the job sequence and the other for the existence of rate-modifying activities. We also adapt the representation of two chromosomes scheme. However, we use the random keys generation representation to generate the job sequence chromosome, in which *N* random numbers from [0, 1] are used as keys to represent a sequence of *N* integers in the range of [1, *N*] (Wang and Uzsoy, 2002). Since the random keys representation eliminates the infeasibility of the offspring chromosome as well as representing solutions in a soft manner, it is applicable to a wide variety of sequencing optimization problems such as machine scheduling, resource allocation, travel salesman problem, quadratic assignment, and vehicle routing, etc. (Gen and Cheng, 1997).

For representing a chromosome for the existence of the RMA, we use 0-1 binary number. We first generate total number of the rate-modifying activities (*k*) using the phase I in TPHA. Then we randomly select the *k* positions in the range of [1, *N*], where $1 \le k \le N$, to assign 1 to the selected positions and we assign 0 to the non-selected positions. Once we found the number of rate-modifying activities as $k = 3$, we randomly generate 3 positions from [1, 10]. If the randomly generated positions are 3, 5, and 8, a chromosome for RMA is described to (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 objective function. Since the problem under consideration is minimization problem, we have to convert the objective function values to fitness function value of maximization form. 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}) is as follows:

$$F_i = \left(Z_{\max} - Z_i\right) \Big/ \sum_{i=1}^{n} \left(Z_{\max} - Z_i\right), \tag{12}$$

where *n* is the number of chromosomes in a population.

4.3 Crossover and Mutation

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 keys representation for chromosome representing job sequence, we can use the one-cut-exchange crossover without violating feasibility. For binary representation for the existence of the RMA, we can also use the one-cut-exchange crossover without violating feasibility. Mutation produces spontaneous random changes in various chromosomes and serves the key role of replacing the genes lost from population during the selection process, or providing the genes that were not present in the initial population. In this paper, we use a simple mutation method in both chromosomes. For random key chromosome, a gene is selected with a small probability and replaced to another random number from [0, 1]. For the chromosome for the RMA, a gene is selected with a small probability and replaced to the other binary number.

The performance of GA is affected by values for the parameters as population size, generation size, crossover rate and mutation rate. Among the parameters, the crossover and mutation rates have greatly influence on the speed of convergence as well as on the success of the optimization. High crossover rate increases the potential premature convergence to a local optimum. The premature convergence can be overcome by raising mutation rate to increase the diversity of the population. But high mutation rate decreases the speed of convergence.

Instead of using fixed values for the crossover and mutation rates, the proposed AGA changes the parameter values spontaneously during a run according to the degree of population diversity. The purpose of changing the parameter values to keep the diversity of the population in solution progress and achieve a good convergence. Average fitness deviation of generation $g\,(AFD^g)$ is using to decide the crossover and mutation rates for each generation. AFD^g is the diversity level of potential pool of generation g and is calculated with the expression (13).

$$AFD^g = \sum_{i=1}^{n} \frac{|PF_i - C_{BEST}|}{C_{BEST}} \Big/ n, \tag{13}$$

where n is the number of chromosomes in the potential parent pool of generation g, PF_i is the fitness value of chromosome i, and C_{BEST} is the best fitness value until current generation. The large value of AFD^g means that the population is too diverse to achieve a good convergence. In this case, it is desirable to increase the crossover rate for good convergence. On the contrary, the small value of AFD^g means that the potential premature converge is high. In this case, it is desirable to increase the mutation rate for population variety. Thus, the crossover rate P_C^{g+1} and mutation rate P_M^{g+1} for the next generation g+1 in AGA are adjusted with the expression (14) and (15). Note that the initial values of P_C^1 and P_M^1 are fixed by extensive preliminary experimentations using a normal GA.

$$P_C^{g+1} = \frac{AFD^g - dL}{dH - dL}, \tag{14}$$

$$P_M^{g+1} = 1 - P_C^{g+1}, \tag{15}$$

where dH is the largest average fitness deviation until generation g and dH the smallest average fitness deviation until generation g.

4.4 Reproduction

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 the roulette wheel selection where each current string in the population has a roulette wheel slot sized in proportion to its fitness. Also, we use the elitist strategy, which is 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 RMA).

Step 2: Select two sets of chromosomes by roulette wheel method in generation g. If a random number from [0, 1] is less than a given crossover probability from equation (14), i.e., P_C^g, generate two sets of chromosomes by one-cut exchange crossover, or copy the two sets of chromosomes directly, otherwise. Repeat this step until g+1 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 from equation (15), i.e., P_M^g, then replace that the gene (random key) in the chromosome for job sequence with another random number from [0, 1] and replace that the gene in the chromosome for RMA with the other binary value.

5. COMPUTATIONAL EXPERIMENTS

Computational experiments are conducted to evaluate the effectiveness of TPHA and AGA with the optimal solutions obtained by the mathematical model using CPLEX 6.0.2 package in small size problems. Additional computational experiments are conducted for large size problems to evaluate the performance of AGA compared with conventional GAs. The heuristics, GAs and AGA are coded in VC++6.0 and the computational experiments are run on a 3.06 GHz, Intel Core 2 Duo CPU with 2GB of memory and Windows XP operating system. Both AGA and GAs are running with the population size of $2 \times N$ and the generation of 2,000 to equally compare each other.

In the case of small size problems, we first compare TPHA and AGA solutions to the optimal solutions. The test problems are generated as follows. Four different job sizes (N = 7, 8, 9, and 10) and three different deterioration rates (α = 0.01, 0.03, and 0.05) are examined and 10 replications are randomly generated for each condition. In the test problems, the processing time of job j, p_j is randomly generated from UNIFORM(10, 30) and Q is fixed as $\bar{p} = 0.5 \cdot Q$, where $\bar{p} = \sum_{j=1}^{N} p_j$. Table 1 shows the CPLEX CPU times for optimal solutions and the relative percent deviations (*RPD*s) of TPHA and AGA calculated by Equation (16)

$$RPD = 100 \times \frac{\sum_{i=1}^{n} (Heu_i - OPT)/OPT}{n}, \quad (16)$$

where Heu_i is the solution value of i^{th} replication obtained by AGA or GAs, n is the number of replications of the solution, and *OPT* is the optimal solution value.

The CPLEX CPU time for optimal solutions exponentially increases as the number of jobs increases, so the optimal solutions by the mathematical model cannot be obtained in a reasonable computing time. In this table, the mean *RPD*s of AGA vary between 0.24%~2.21%, but the mean *RPD*s of TPHA vary 7.31%~9.68%, and the maximum *RPD*s of AGA are no more than 4.52% in all instances. These results show AGA is very effective algorithm for the problem of this paper in comparison with TPHA.

For large size problems, we generated 20 problem sets generated with four different job sizes (N = 30, 60, 90, and 120) and three different deterioration rates (α = 0.01, 0.03, and 0.05), and 10 replications are randomly generated for each condition.

Since finding the optimal solution using the mathematical programming is not practical due to long CPU times, we compare the solutions of AGA and the conventional GAs having four different combination of the crossover and mutation probability with the best solution of all 50 replications of five heuristics (AGA, GA(0.6/0.4), GA(0.7/0.3), GA(0.8/0.2), and GA(0.9/0.1)) for each problem set. Thus, relative deviation with best solution (*RPD_Best*) calculated by Equation (17) is used to verify the performance of the algorithms instead of *RPD*.

$$RPD\_Best = 100 \times \frac{\sum_{i=1}^{n} (Heu_i - Best)/Best}{n}, \quad (17)$$

where *Best* is the best solution value of replications of AGA and GAs.

Mean absolute deviation (*MAD*) calculated by Equation (18) is used to verify the variance of the algorithm.

$$MAD = \sum_{i=1}^{n} \left| Heu_i - \overline{Heu} \right| / n, \quad (18)$$

where $\overline{Heu}$ the mean of the solution value obtained by AGA or GAs.

Table 2 shows the mean *RPD_Best* of 50 replications of AGA and four GAs, the mean absolute deviation (*MAD*) and CPU time for each test problem. The mean *RPD_Best* and mean *MAD* of AGA are 1.4942 and 0.0042% and, which are the smallest values over other GAs. The computing time of AGA is similar with other GAs. These results indicate that AGA is more effective and efficient algorithm than the conventional GAs with the various combinations of crossover and

Table 1. Comparison between Two Heuristics (TP and AGA) and the Optimal Solution.

N	α	CPLEX CPU times (Sec.)		*RPDs* of TPHA			*RPDs* of AGA		
		Mean	*Max*	*Mean*	*Max*	CPU time (Sec.)	*Mean*	*Max*	CPU time (Sec.)
7	0.01	0.3849	0.936	7.31	10.31	0.020	0.70	1.74	1.507
	0.03	1.9434	6.025	8.73	15.71	0.017	0.24	0.88	1.617
	0.05	1.7891	4.116	7.87	12.52	0.015	0.33	1.12	1.623
8	0.01	5.3361	15.16	8.39	20.51	0.032	1.51	4.21	1.610
	0.03	7.4566	19.16	9.68	17.61	0.026	1.10	1.61	1.651
	0.05	8.1861	16.015	9.25	15.94	0.025	1.22	2.23	1.647
9	0.01	53.234	150.15	8.06	10.85	0.041	1.27	2.33	1.804
	0.03	57.021	190.32	8.44	12.05	0.039	1.32	2.76	1.831
	0.05	119.271	300.31	8.59	12.09	0.040	1.16	2.31	1.879
10	0.01	237.065	1100.32	9.31	15.91	0.081	2.21	4.52	1.953
	0.03	453.328	1000.32	9.62	19.80	0.061	1.90	2.20	1.972
	0.05	1714.51	6700.16	9.00	16.94	0.055	1.14	3.10	1.983

Table 2. The Comparison Between GAs and AGA for Large Size Problems

N	α	AGA			GA(0.6/0.4)			GA(0.7/0.3)		
		RPD _Best	*MAD*	CPU time (Sec.)	*RPD _Best*	*MAD*	CPU time (Sec.)	*RPD _Best*	*MAD*	CPU time (Sec.)
30	0.03	0.9134	0.0046	5.20	2.6863	0.0084	6.30	3.2215	0.0049	6.30
	0.05	1.3644	0.0043	6.02	3.7476	0.0078	6.83	3.5236	0.0042	6.83
	0.07	1.2073	0.0053	6.30	3.7830	0.0070	6.88	2.9390	0.0066	6.78
60	0.03	1.6927	0.0034	43.40	3.1428	0.0060	40.80	2.3024	0.0051	40.80
	0.05	1.7140	0.0059	46.64	3.8010	0.0050	44.38	2.8276	0.0071	44.38
	0.07	1.4579	0.0003	46.16	4.7711	0.0060	45.14	3.9166	0.0034	45.04
90	0.03	2.1527	0.0068	131.80	3.2150	0.0037	131.20	2.4274	0.0067	131.80
	0.05	0.8896	0.0057	137.58	3.4928	0.0026	136.22	2.4926	0.0060	136.08
	0.07	3.4570	0.0042	134.26	7.9651	0.0041	136.72	6.4280	0.0051	136.61
120	0.03	1.3655	0.0022	287.60	2.8572	0.0024	292.10	2.0092	0.0056	289.50
	0.05	0.8420	0.0035	296.06	2.9354	0.0063	299.61	2.0769	0.0042	298.25
	0.07	0.8737	0.0039	298.71	4.0489	0.0052	298.16	3.0512	0.0024	297.23
Average		1.4942	0.0042	119.98	3.8705	0.0054	120.36	3.1013	0.0051	119.97

N	α	GA(0.8/0.2)			GA(0.9/0.1)			
		RPD _Best	*MAD*	CPU time (Sec.)	*RPD _Best*	*MAD*	CPU time (Sec.)	
30	0.03	2.7506	0.0084	6.40	2.2167	0.0078	6.51	
	0.05	2.2891	0.0081	7.24	2.1842	0.0091	7.36	
	0.07	2.5143	0.0070	7.32	1.9024	0.0039	7.25	
60	0.03	1.9269	0.0053	41.30	1.7017	0.0057	42.50	
	0.05	2.4622	0.0047	45.13	1.9308	0.0026	44.92	
	0.07	3.0661	0.0064	45.51	2.3202	0.0041	47.61	
90	0.03	2.2298	0.0048	131.20	1.8824	0.0036	132.30	
	0.05	1.7938	0.0035	136.12	1.5364	0.0043	131.12	
	0.07	5.7052	0.0035	136.71	4.7634	0.0054	137.81	
120	0.03	1.6799	0.0035	289.10	1.2690	0.0020	292.20	
	0.05	1.7597	0.0024	297.61	0.9801	0.0033	298.62	
	0.07	1.8337	0.0035	296.86	1.2942	0.0044	298.92	
Average		2.5009	0.0051	120.04	1.9985	0.0047	120.59	

mutation parameters.

6. CONCLUSIONS

In this paper, a single-machine scheduling problem with deteriorating processing time of jobs and multiple rate modifying activates is considered. First, we formulate an integer programming formulation. Since this problem is difficult to solve as the large size problems, we propose a new genetic algorithm, AGA with spontaneously adjusting crossover and mutation rate based on the status of solution quality of the current population. The maximum *RPD* of AGA is no more than 4.52% in comparison with the optimal solutions in small size problem sets. In large size problem sets, the mean *RPD_Best* and mean *MAD* of AGA provides the smallest values over other GAs. The computing time of AGA is similar with other GAs. Therefore, these results indicate that the proposed AGA is very effective and efficient algorithm for the scheduling problem in this paper.

REFERENCES

Bachman, A., Janiak, A., and Kovalyov, M. Y. (2002), Minimizing the total weighted completion time of deteriorating jobs, *Information Processing Letters*, **81**(2), 81-84.

Browne, S. and Yechiali, U. (1990), Scheduling deteriorating jobs on a single processor, *Operations Research*, **38**, 495-498.

Cheng, T. C. E. and Ding, Q. (2000), Single machine scheduling with deadlines and increasing rates of processing times, *Acta Informatica*, **36**, 673-692.

Gen, M. and Cheng, R. (1997), *Genetic Algorithms and Engineering Design*, Wiley, New York.

Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wes ley, New York.

Graves, G. H. and Lee, C. Y. (1999), Scheduling maintenance and semiresumable jobs on a single machine, *Naval Research Logistics*, **46**, 845-863.

Gupta, J. N. D. and Gupta, S. K. (1988), Single facility scheduling with nonlinear processing times, *Computers and Industrial Engineering*, **14**, 387-393.

Kovalyov, Y. M. and Kubiak, W. (1998), A fully polynomial approximation scheme for minimizing makespan of deteriorating jobs, *Journal of Heuristics*, **3**, 287-297.

Kubiak, W. and Velde, S. (1998), Scheduling deteriorating jobs to minimize makespan, *Naval Research Logistics*, **45**, 511-523.

Kunnathur, A. S. and Gupta, S. K. (1990), Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem, *European Journal of Operational Research*, **47**, 56-64.

Lee, C. Y. and Chen, Z. L. (2000), Scheduling of jobs and maintenance activities on parallel machines, *Naval Research Logistics*, **47**, 61-67.

Lee, C. Y. and Leon, V. J. (2001), Machine scheduling with a rate-modifying activity, *European Journal of Operational Research*, **128**, 119-128.

Lee, C. Y. and Lin, C. S. (2001), Single-machine scheduling with maintenance and repair rate-modifying activities, *European Journal of Operational Research*, **135**, 493-513.

Mosheiov, G. (1991), V-Shaped polices to scheduling deteriorating jobs, *Operation Research*, **39**, 979-991.

Qi, X., Chen, T., and Tu, F. (1999), Scheduling the maintenance on a single machine, *Journal of the Operational Research Society*, **50,** 1071-1078.

Sortrakul, N., Nachtmann, C. R., and Cassady, C. R. (2005), Genetic algorithms for integrated preventive maintenance planning and production scheduling for a single machine, *Computers In Industry*, **56**, 161-168.

Wang, J. L., Sun, L., and Sun, L. (2011), Single-machine total completion time scheduling with a time-dependent deterioration, *Applied Mathematical Modelling*, **35**, 1506-1511.

Wang, C. S. and Uzsoy, R. (2002), A genetic algorithm to minimize maximum lateness on a batch processing machine, *Computers and Operations Research*, **29**, 1621-1640.