Hindawi Publishing Corporation
ISRN Industrial Engineering
Volume 2013, Article ID 936814, 11 pages
http://dx.doi.org/10.1155/2013/936814

Research Article

A Dynamic Genetic Algorithm for Solving a Single Machine Scheduling Problem with Periodic Maintenance

Amir Ebrahimy Zade and Mohammad Bagher Fakhrzad

Department of Industrial Engineering, Yazd University, Yazd 7617659813, Iran

Correspondence should be addressed to Mohammad Bagher Fakhrzad; mfakhrzad@yazd.ac.ir

Received 7 October 2013; Accepted 27 November 2013

Academic Editors: T.-M. Choi and A. Gomez

The scheduling problem with nonresumable jobs and maintenance process is considered in order to minimize the makespan under two alternative strategies. The first strategy is to implement the maintenance process on the machine after a predetermined time period and the second one is to consider the maximum number of jobs that can be done with an especial tool. We propose a new mathematical formulation for the aforementioned problem which is included in the NP-Hard class of problems; in the second part of the paper, we propose a dynamic genetic algorithm so that the large- and medium-scale problems could be solvable in computationally reasonable time. Also we compare the performance of the proposed algorithm with existing methods in the literature. Experimental results showed that the proposed genetic algorithm is able to attain optimum solutions in most cases and also corroborate its better performance from the existing heuristic methods in the literature.

1. Introduction and Literature Review

In most scheduling problems, it is assumed that the machine is continuously and uninterruptedly available. However, in the real world problems this is not the case (Pinedo [1]). The possible machine breakdown and interruption would be unavoidable in case of working continuously and also this may affect the quality of the jobs being processed by the machine. Thus, the periodic preventive maintenance process is usually planned and implemented to avoid the aforementioned problem. In this paper the scheduling problem is mathematically formulated subject to the amount of time the machine is available between the two consecutive periods, at the end of which the maintenance process is implemented and the maximum number of jobs which could be processed by the machine over the operating time.

Jobs are done in two different ways when we have a limited time period in which the machine is available before the next maintenance process. In the first case, which is called preemptive jobs, it is assumed that operations of the in-process jobs could be continued after machine interruption. On the other hand, for nonpreemptive jobs, it is assumed that the incomplete jobs must be reprocessed after interruption. Also, for some manufacturing processes such as manufacturing of the electronic circuits, the number of jobs to process before changing the tools is a constraint. Considering Graham's three-field notation for symbolizing the scheduling problems, this case is specified with $1/nr\text{-}pm/c_{\max}$ (Graham et al. [2]).

The problem of machine availability has been widely considered in the literature due to machine breakdown, tool changing, and preventive maintenance process. In some of the researches, machine unavailability has been considered due to the stochastic breakdowns. For single machine scheduling problem with stochastic breakdowns, the optimal solution has been provided by Pinedo and Rammouz [3], Birge et al. [4], and Frostig [5]. Some heuristics have been also developed to minimize the number of tardy jobs or makespan (Adiri et al. [6, 7]). In addition, the constraint of the machine unavailability due to tool change has been also considered by Akturk et al. [8, 9], Ecker and Gupta [10], Chen [11], and Choi and Kim [12].

Machine unavailability due to preventive maintenance processes has been considered based on both flexible and periodic maintenance process. In the flexible maintenance process, the earliest and latest start time of the maintenance process is already specified and the machine shutdown is allowed in this range. Yang et al. [13] surveyed the problem of the single machine scheduling considering the only one flexible maintenance process. Regarding the actual time needed for maintenance process which is shorter than or equal to predetermined time, the flexible maintenance process is considered. They surveyed the problem to minimize the makespan and prove the NP-Hardness of the developed problem. Qi et al. [14] also considered the problem of scheduling several processes of maintenance as well as jobs processing on the machine. They applied heuristics and also methods on basis of the reputable branch and bound to determine the best jobs sequence and scheduling of the maintenance process in order to minimize the makespan. A mixed binary integer programming and heuristics to minimize the number of in-process jobs have been also proposed by Chen [15]. Finally, Low et al. [16, 17] surveyed the problem of single machine scheduling considering two alternative strategies, namely, machine unavailability after a fixed time period and also after processing a specific number of jobs to change the tool. They considered minimization of the makespan as their objective.

In context of the problems with periodic maintenance process, there are variants of research. Liao and Chen [18] developed an algorithm on the basis of the reputable branch and bound to minimize the maximum tardiness. Chen [19] suggested a heuristic algorithm to minimize the makespan. In addition, Chen [20] presented a method based on the reputable branch and bound as well as a heuristic to minimize the number of tardy jobs. Lee and Lee [21] developed a heuristic to minimize the makespan. Lau and Zhang [22] assumed that a fixed number of operations must be done between two successive interruptions. Liao et al. [23], Dell'Amico and Martello [24], and Yang et al. [25] assumed that the machine must stop working utmost after processing a fixed number of jobs due to the maintenance process. Finally, Hsu et al. [26] considered the single machine scheduling problem with periodic maintenance and the same strategies as the ones already considered by Low et al. [17]; their objective was to minimize $c_{\max}$. They developed a two-level integer programming for solving the problem optimally as well as two heuristics for solving the large-scale problems, namely, DBF and BBF. As Pinedo (2006) mentioned, the scheduling problems belong to strongly NP-Hard problems. Also in the single machine scheduling problems with machine unavailability, there are a lot of papers in which NP-Hardness of the problem is mentioned; for example, Lee and Liman [27] proved the NP-Hardness of the problems with deterministic maintenance times. In order to solve the problem in most cases heuristic algorithms are handled and to the best of our knowledge only Low et al. [17] use a modified version of particle swarm optimization algorithm for single machine scheduling problem with periodic maintenance. So in this paper, for solving the single machine scheduling problem, fixed time between two consecutive maintenance operations, and maximum number of jobs that can be done during this period due to the change of the tool, a dynamic genetic algorithm is proposed.

The rest of this paper is organized as follows. In the following section, we first propose a new mathematical model for solving the problem and following that we consider an efficient way of solving the problem in computationally reasonable time by proposing a genetic algorithm. At the third part, the results of applying both methods are presented and compared against each other. In this part we also compare the performance of the proposed GA with the two existing heuristic methods which are proposed by Hsu et al. [26]. Conclusions and some guidelines for future study are presented in the last part of this paper.

2. Proposed Methods

This section contains two parts in each of which a separate solution method to solve the aforementioned problem is provided. In the first part, a mixed integer programming model is proposed for the first time in the literature, whereas in the second part we propose a GA to solve the problem.

2.1. Mathematical Formulation. In this paper, a single machine scheduling problem is considered with two alternative strategies: (1) implementing the maintenance process after a predetermined time period and (2) the maximum number of jobs that can be done during this period in order to change the tool. The objective of the problem is to minimize the makespan and it is assumed that jobs are nonresumable which means if process of any job could not be completed till the next maintenance process, it must be repeated after maintenance time. Considering all the above, we can define the problem as follows: n jobs are planned to be processed on a machine and are nonresumable. The processing time of each job is indicated with p_i and it is assumed that all jobs are ready and available at the beginning. Time period between two successive maintenance processes is t and time needed for implementing any maintenance activity is specified with m.The jobs processed between two successive maintenance processes are called a batch and should not exceed k. Hence, if it is supposed to have l batches in a time horizon, the number of maintenance processes is l–1. This could be shown in the schematic view in Figure 1.

Considering the above-mentioned explanation, the problem notation is as follows:

t: time period between two successive maintenance processes;

m: time needed for implementing any maintenance activity;

k: maximum number of jobs to process in period t;

M: a large number;

n: number of jobs;

p_i: processing time of job $i, i = 1, 2, \ldots, n$;

x_{ij}: if job i belongs to batch j, it equals 1; Otherwise, it equals 0, $j = 1, 2, \ldots, l$;

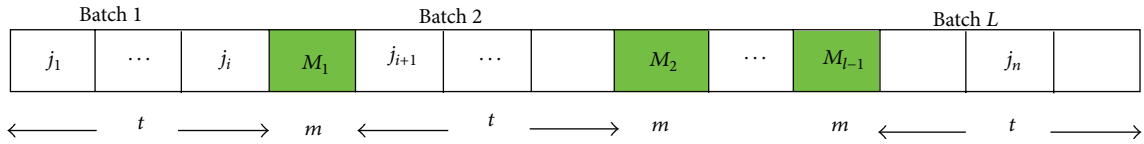

Figure 1: A schematic view of the problem.

y_j: if at least one job belongs to batch j, it equals 1; otherwise, this variable equals 0;

g_j: amount of gap for batch j;

w_j: the amount of gap for all batches except for empty batches and the last batch in which jobs take place.

In the research done by Hsu et al. [26], the problem of minimizing the makespan is solved by using two mathematical models; in the first one the minimum number of required batches to complete jobs is determined and in the second one, assuming the number of batches a priori, the jobs are sequenced and arranged in the batches such that the makespan is minimized. In this paper, we propose a new mathematical formulation for the problem which solves the problem in one stage.

Before representing the mathematical model, the general idea of solving the problem is to be explained. Regarding the optimal number of the batches which is not specified at the beginning, the maximum possible number of batches is considered initially. Hence, if there are n jobs, there could be n batches at most. However, after arranging the jobs considering minimization of the makespan, subject to predetermined maximum time period allowed and the maximum number of jobs in the batches, there would be batches with more than one job assigned and, as a result, we will have batches with no jobs assigned. The model omits these empty batches. Also the amount of gap for each batch, excluding the last batch, is calculated. At last, the sum of the jobs' processing times, gaps of the batches, and fixed times needed for maintenance processes constitute the amount of the makespan.

Following the explanation mentioned above, the mixed integer programming model is proposed as follows:

$$\min\left(\left(\sum_j y_j - 1\right)\cdot m + \sum_j w_j + \sum_i p_i\right), \tag{1}$$

$$\sum_j x_{ij} = 1 \quad \forall i, \tag{2}$$

$$\sum_i x_{ij} \le k \quad \forall j, \tag{3}$$

$$\sum_i x_{ij}\cdot p_i \le t \quad \forall j, \tag{4}$$

$$\sum_i x_{ij} \le M\cdot y_j \quad \forall j, \tag{5}$$

$$\sum_i x_{ij} > M\cdot\left(y_j - 1\right) \quad \forall j, \tag{6}$$

$$y_{j+1} \le y_j \quad \forall j, \tag{7}$$

$$g_j = t - \sum_i x_{ij}\cdot p_i \quad \forall j, \tag{8}$$

$$w_j = g_j.y_{j+1} \quad \forall j, \tag{9}$$

$$w_j, g_j \ge 0, \quad x_{ij}, y_i \in \{0,1\}, \quad y_{n+1} = 0 \quad \forall i, j. \tag{10}$$

The first expression (1) is objective function which, as described above, consists of three separate terms. The first term calculates the sum of the fixed times needed for maintenance processes, the second term calculates the sum of the gaps of the batches excluding the last one, and the third one calculates the sum of the jobs' processing times. Constraint (2) assures that each job is exactly assigned to one batch. Constraint (3) assures that the number of jobs in each batch does not exceed k. Constraint (4) assures that the sum of the jobs' processing times assigned to one batch could not exceed the maximum time period allowed t.

We considered n empty batches at the beginning and this may result in empty batches at last, so constraints (5) and (6) are regulated so that if any job is assigned to the kth batch, respective binary variable y takes the value 1 and otherwise it takes 0. Constraint (7) is regulated so that, until a batch is empty, all of its further batches will also be empty. Hence, this constraint helps the empty batches, whose corresponding binary variable y is 0, take place just after the filled batches. This characteristic, as will be explained later in the paper, is used to calculate the amount of gap. By using constraint (8), the initial value of gap is calculated. If we are going to calculate the amount of gap with (8) we will face a problem. In this situation an empty batch will have gap t and it will affect the objective function; in order to solve this problem we introduce constraint (9) to calculate the amount of variable w. For empty batches the amount of this variable will equal 0 also because the amount of gap for the last batch, in which jobs take place, is not a part of the makespan; the amount of this variable for the mentioned batch will equal 0. This variable for all other batches will be equal to gap for that batch. In order to solve the problem of nonlinearity of this constraint we propose the following constraints:

$$\begin{aligned} g_j - M\cdot y_{j+1} &\le w_j \quad \forall j, j \ne n, \\ g_j + M\cdot\left(1 - y_{j+1}\right) &\ge w_j \quad \forall j, j \ne n. \end{aligned} \tag{11}$$

In this constraints, if any job is assigned to the next batch, $w_j = G_j$, but if the next batch is empty, w_j is a free variable and, considering it only takes the positive values, it ranges from 0 to *infinite*. On the other hand, considering that this variable is a part of the objective function and

the minimization of the objective function is the case, this variable takes its least possible amount, say 0.

2.2. Proposed Genetic Algorithm. The genetic algorithm, first introduced by Holland [28], is an evolutionary algorithm on the basis of Darwinian's theory of evolution. One of the main differences of this algorithm from other metaheuristic methods, such as simulated annealing, is using a set of solutions (called population) instead of a single solution. As mentioned by Pinedo (2001), production planning problems belong to the NP-Hard class, and also because of the efficiency of GA for discrete optimization problems, as mentioned by Davis [29] and Goldberg [30], many researchers have used it for solving the scheduling problems.

The main steps of applying every genetic algorithm which must be taken into account are as follows:

(1) chromosome structure;

(2) initial population;

(3) fitness function;

(4) selection strategies;

(5) genetic operators;

(6) stopping criteria.

We will explain each of the above items in the following sections.

2.2.1. Chromosome Structure. Specifying the chromosome structure in every problem is one of the main steps of applying the genetic algorithm. The chromosome structure should be devised such that all of the problem's features could be fitted in that. In our problem, the chromosome structure indicates jobs sequence and their processing times. The rest of the required information for the problem can be derived from this structure.

2.2.2. Initial Population. Specifying the initial population size is of special importance in the genetic algorithm. As mentioned by Back et al. [31] if the population size is too small, we might not find a suitable solution or if the population size is too large, it might result in large computational time. We choose a population size of 200 from among other candidates such as 300, 100, and 250 because of its better performance on devised experiments.

Suppose that there are n jobs such that n_1 jobs require processing time t_1, n_2 jobs require processing time t_2, and n_3 jobs require processing time t_3 ($n_1 + n_2 + n_3 = n$). Hence, any permutation of these jobs is a possible solution for our problem. This strategy is used to make an initial population.

2.2.3. Fitness Function. Fitness function is calculated to specify how suitable a solution is. So fitness function should be devised such that different solutions could be evaluated according to its value. In order to calculate fitness of a chromosome we use the idea in the objective function and break it into three parts. The first part is related to the jobs' processing times, which remains the same for various job sequences. The second part is related to the fixed maintenance process at the end of a period. Considering that at the end of a fixed time period there is a maintenance activity, we should first find the number of batches and the number of maintenance processes is the *number of batches–1*. The third part of this function is equal to the sum of the gaps (except for the last nonempty batch). In order to calculate the amount of gap for each nonempty batch initially due to jobs sequence and their processing times we should find out the jobs in a batch, and then by subtracting sum of their processing times from the fixed time between two maintenance operations we will have the gap for a specified batch. Summing these three parts, we can calculate the makespan.

2.2.4. Selection Method. As introduced by Obitko [32], the selection methods are divided into four categories, namely, roulette wheel, rank, steady state, and elitism. Here we use the roulette wheel method. Goldberg [30] proposed this method. In this method, every population member has a chance to be selected although this probability is not the same for all members.

After all individuals in the population are ranked according to their fitness values, each of them is assigned a selection probability according to its' rank. At the next step, considering the weighed probabilities, we choose the parents amongst the population members.

2.2.5. Genetic Operators. There are two genetic operators, crossover and mutation. The crossover operator uses two chromosomes as the parents and combines them; it produces two offspring. Although there are other methods for crossover such as two-point crossover, and uniform crossover here we use one-point crossover which is the simplest crossover method. Using this method, a point is randomly selected from which both parents' chromosomes are cut off. Afterward, the first part of the first parent is connected to the second part of the second parent and the first offspring is produced. Following this approach, by combining the first part of the second parent and the second part of the first parent, the second offspring is produced. As specified in Figure 2, this method results in infeasible solutions. It is obvious that, in the first offspring produced, we do not have any job with processing time 10, whereas in the second offspring produced we have two jobs with processing time 10, and there are five jobs with processing time 5 in the first offspring and three jobs with processing time 5 in the second offspring.

To make over the aforementioned problem, there are multiple methods. One of these methods is the position-based crossover which was first proposed by Syswerda [33] for implementing uniform crossover that is used in other papers such as the paper by Hamidinia et al. [34] in which, with a little variation, this method is used for single-point crossover. However, it is to be noticed that in these kinds of problems, what causes the problem to be infeasible is the repetition of some jobs for two times and elimination of some others in the offspring. But in our problem the situation is different. Our objective in the problem ahead is to make the number of jobs

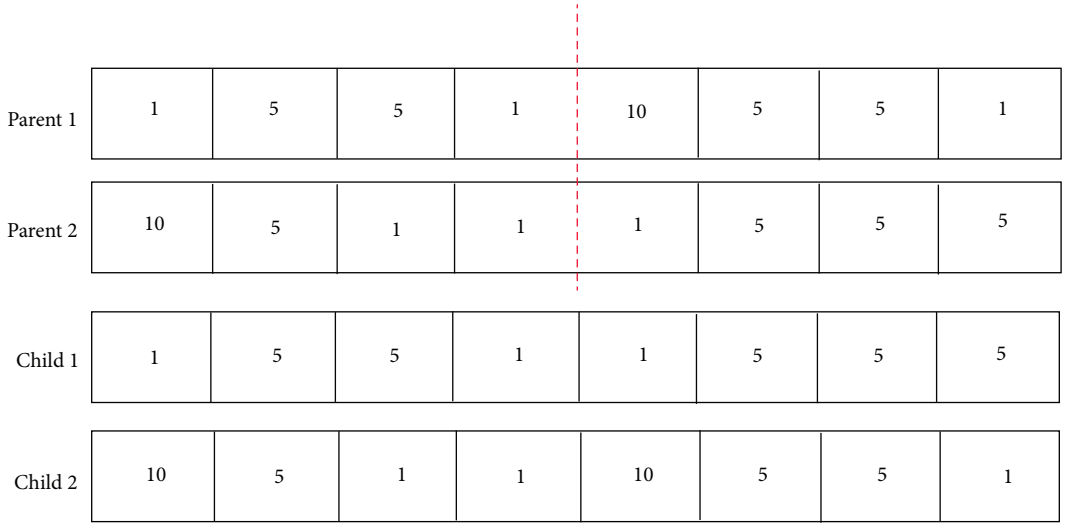

Figure 2: Crossover leads to infeasible solution.

of a specific kind fixed in the offspring's structures produced (in this problem, two jobs with the same processing time are the same in nature). Next, we are going to develop a method for crossover and we will explain it by its implementation on the example mentioned in Figure 2.

After cutting off the parents from a random point (as specified in Figure 2), we take into account the second parts of both parents. In each part, we determine the number of jobs with processing time 1, the number of jobs with processing time 5, and the number of jobs with processing time 10. Now for each type of jobs (regarding their processing times), we introduce an indicator in their corresponding offspring, the value of which equals the differentiation of the number of jobs with the same processing time in the second part of each parent. The results of the calculations for the first child which is produced from the first part of the first parent and the second part of the second parent could be seen in Table 2. In this table the amount of indicator for different kinds of jobs (indicator 10 − 1 means jobs with processing time 10 in the first offspring) is updated at each step in which a job is going to be transmitted to the corresponding offspring. We move the jobs assigned to the second part of the second parent to the first part of the first parent considering first the indicator of each job. If the indicator is less than or equals *zero*, it is moved without any changes. Otherwise, if the indicator is greater than zero, we randomly select a job, the value of which indicator is less than zero and transmit it except the original job. Then, we add 1 to the indicator of the moved job and subtract 1 from the main job's indicator. At the end of this process, the amount of the indicator of each job will equal *zero*. In Figure 3 you can see the second part of the second parent which is a candidate for sitting next to the first parent's first part and making the first offspring.

The first jobs processing time is 1 and according to Table 1 its indicator is 0, so it is transferred without any change. The second job has processing time 5 and its indicator at the beginning of this phase is greater than 0 (equals one) so we choose a job with an indicator less than zero randomly (here the only job with negative indicator is the job with processing time 10). At the end of this phase the indicator for jobs with processing time 10 will be increased one unit and will equal zero, and in the same way we decrease one unit from the indicator of the jobs with processing time 1. Now the amount of indicator for all three kinds of jobs equals 0; as a result, in further phases the jobs will transmit without any change.

The second part of the first parent is transmitted similarly. Finally we can see the produced offspring in Figure 4.

Mutation is the second genetic operator. This operator through random changes in the chromosomes enables better search in solution space. For consecutive iterations of the genetic algorithm, when best answer remains without changing, it means that the algorithm is trapped in a local optimum. In such cases, more searches in solution space, which can be done by mutation operator, can be instrumental in achieving optimal solution. In the literature, various approaches such as single-point mutations, two-point, multipoint, and swap are introduced. Here we introduce a dynamic multipoint swap mutation; experiments show that our method performs better than other static methods.

In the proposed method, the number of selected points for swapping (n) liaises to the number of iterations in which the best solution found by the algorithm remains the same without any changes. As the number of iterative solutions increases, the proposed mutation causes stronger search in the answer space. Size n is calculated from the following equation, in which t is the minimum number of jobs of a specific kind. As we have considered three types of jobs in this paper (as it is discussed in part 3, we have jobs with processing time 1, 5, and 10) then n will be min $\{n_1, n_2, n_3\}$. n_{Var} is the total number of jobs:

$$n = \begin{cases} \dfrac{t}{3} & \text{if } l < \dfrac{n_{\text{Var}}}{5} \\ \dfrac{t}{2} & \text{if } \dfrac{n_{\text{Var}}}{5} \le l < \dfrac{n_{\text{Var}}}{3} \\ t & \text{if } l \ge \dfrac{n_{\text{Var}}}{3}. \end{cases} \tag{12}$$

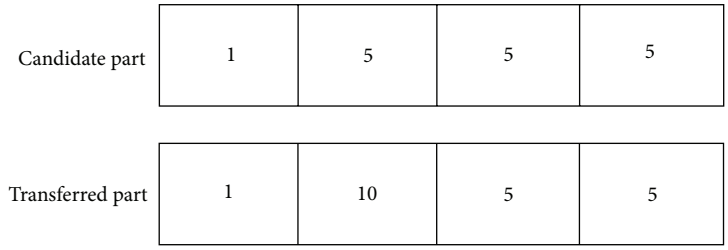

Figure 3: Candidate part versus the original part.

Table 1: Steps of the proposed method.

Phase 1	Phase 2	Phase 3	Phase 4
Indicator $10 - 1 = 0 - 1 = -1$	Indicator $10 - 1 = 0$	Indicator $10 - 1 = 0$	Indicator $10 - 1 = 0$
Indicator $5 - 1 = 3 - 2 = 1$	Indicator $5 - 1 = 0$	Indicator $5 - 1 = 0$	Indicator $5 - 1 = 0$
Indicator $1 - 1 = 1 - 1 = 0$	Indicator $1 - 1 = 0$	Indicator $1 - 1 = 0$	Indicator $1 - 1 = 0$

Two genes are randomly selected of each chromosome and their values will be swapped with each other; it is repeated n times for each chromosome.

2.2.6. Stopping Criteria. To terminate the algorithm, various criteria have been introduced in the literature; one of them, which is also very useful, is considering the maximum number of iterations for the algorithm of which the amount should be determined according to the problem. In this study according to the results of the experiments, 250 have been considered as the maximum number of generation (Max It). In some issues the algorithm will reach the optimum solution in early iterations and, until the last iteration, the solution remains unchanged. In this situation two possible modes may be arising: first, the algorithm being trapped in local optimum, and in this case more search is needed which is carried out by using the proposed dynamic mutation. Second, the algorithm reaches the optimal solution. In such cases, to reduce the time needed to solve the problem, it is appropriate to stop the algorithm before the last iteration. Due to the possibility of having a local optimum, algorithm is allowed Max $It/2$ iterations for further search and if, after this number of iterations, best solution did not change, the algorithm will stop. To improve the performance of the algorithm, the second stopping criterion will be used alongside the primary criterion.

2.2.7. Overview of Genetic Algorithm. Each genetic algorithm contains four stages: initialization, selection, reproduction, and termination. The pseudocode presented in Algorithm 1 shows the aforementioned stages in the proposed genetic algorithm.

3. Design of the Experiments and Experimental Results

In order to evaluate the proposed genetic algorithm, we require a number of experimental problems through which we can compare the results of the different methods against each other. For generating the random numbers and developing the problem, we use Hsu et al. [26]'s proposed approach. In this approach the problems are categorized into three sizes. Small-size problems are those with 20, 30, 40, 50, and 100 jobs. Medium-size problems are those with 250, 300, 350, 400, and 500 jobs. And in problems of large size, the number of jobs is considered 700, 1000, 2000, and 10000. Processing time of each job and the fixed time of the maintenance process are uniformly distributed over the discrete values of 1, 5, and 10. The time considered between two successive maintenance processes is calculated through the expression $t = \max\{\lceil a \cdot \sum_i p_i \rceil, \max p_i\}$ in which a is uniformly distributed over the discrete values of $\{1/3, 1/4, 1/5\}$. The maximum number of jobs to process in every production run with one tool is taken from $k = \lceil b, n \rceil$ in which b is uniformly distributed over the discrete values of $\{1, 1/2, 1/3, 1/4, 1/5\}$.

The proposed genetic algorithm has some parameters that should be adjusted so that the algorithm works satisfactorily. For the number of generations we choose 250, number of individuals in the population (n_{pop}) is 200, percentage of population which is chosen for crossover (P_c) is 0.9, and mutation percentage is 0.1.

The experiments were done on a computer with a 2.4 GHz Core 2 duo CPU and 3 GB of RAM. For achieving globally optimal solutions, we used the GAMS software. As a result, for small- and medium-size problems, the needed time is less than 24 hours. On the other hand, for solving the problems with more than 700 jobs (large-size problems), the needed processing time is not computationally reasonable and exceeds 24 hours. Table 2 shows the results of running the experiments. For each problem size there are nine sample problems and each of them was ran 4 times, so our proposed genetic algorithm was tested totally in 504 problems, and mean time to solve each problem along with the best solution found is presented in Table 2. As mentioned before there are also two heuristic methods, namely, butterfly order with best fit (BBF) and decreasing order with best fit (DBF), for this problem which are presented by Hsu et al. [26] In order to compare the performance of these algorithms with the proposed GA their results are also presented in Table 2.

Child 1	1	5	5	1	1	10	5	5
Child 2	10	5	1	1	5	5	5	1

Figure 4: Children produced using the proposed method.

Inputs: n_{Var} (Number of variables), n_{pop} (Number of initial individuals), P_c (Crossover percentage), P_m (Mutation percentage), Max It (Number of generations)
(1) Initialization:
(1-1) Create n_{pop} chromosomes calculate the fitness of each one and sort them due to their fitness
(2) Iterations: for $i = 1$ to Max It:
(2-1) Selection
(2-1-1) Select $\lceil n_{\text{pop}} \times p_c \rceil$ individuals using roulette wheel for crossover.
(2-1-2) Select $\lceil n_{\text{pop}} \times p_m \rceil$ individuals using roulette wheel for mutation.
(2-2) Reproductions
(2-2-1) Conduct the crossover operator on each pair of the selected parents to generate offspring
(2-2-2) conduct the mutation operator on each of the selected parents to generate an offspring.
(2-2-2-1) If the current solution is repeated less than (Max $It/5$) conduct $n_{\text{Var}}/3$ swapping and else go to (2-2-2-2)
(2-2-2-2) If the current solution is repeated less than (Max $It/3$) conduct $n_{\text{Var}}/2$ swapping and else go to (2-2-2-3)
(2-2-2-3) Conduct n_{Var} swapping.
(2-2-3) Merge mutants, Crossover offsprings and initial population; sort them according to fitness and choose the top n_{pop} members.
(2-3) If the current solution is repeated for (Max $It/2$) go to (3) and else continue.
(3) End of genetic algorithm.

Algorithm 1: Pseudocodes for proposed genetic algorithm.

Error percentage for proposed GA and the two heuristic methods in small- and medium-sized problems in which optimal solution was attainable less than 24 hours is calculated as follows:

$$\text{error} = \frac{\text{algorithm solution} - \text{optimum solution}}{\text{optimum solution}}. \quad (13)$$

For large-scale problems in which optimum solution was not attainable, due to better performance of the proposed GA compared with the heuristic methods in all cases, the amount of deviation of BBF and DBF algorithm from the attained solution by GA is mentioned as error:

$$\text{error} = \frac{\text{algorithm solution} - \text{GA solution}}{\text{GA solution}}. \quad (14)$$

Comparing the results of our proposed genetic algorithm with the optimal solution, found by GAMS, we can see that, from the 90 sample problems in which GAMS was able to find the optimal solution in a reasonable time and less than 24 hours (small-size and medium-size problems), only in seven sample problems our proposed algorithm did not achieve the optimal solution, in which maximum error was about 0.02. This shows that the proposed genetic algorithms performance is reliable on the aforementioned problem.

Also we can see that the in all cases proposed GA presents better results than the two heuristic methods. From the table we can see that for BBF algorithm maximum error is about 0.3 and for DBF algorithm maximum error is about 0.36 which shows their weaker performance in comparison with the proposed GA.

The other important point that should be noted is the time needed to solve a sample problem with the proposed GA compared to the time needed to solve the same problem optimally. As it is shown in Table 2 for the sample problems with less than 100 jobs the time needed to solve the problem optimally is a little less than the time needed to solve the problem with genetic algorithm but, as the problem size increases, the time needed to solve the problem optimally has very fast growth, so that, for the large-scale problems (700 jobs and more), even after 24 hours GAMS was not able to achieve the optimal solution but we can see that this growth in the time needed for the genetic algorithm is very low and our proposed genetic algorithm is able to solve the largest scale of the sample problems (problems with 2000 jobs) in a time of about 5 minutes. Another point that should be mentioned is the big range of runtimes for problems of the same size. For example, there is a problem with 72 seconds of runtime and also there is a problem with 101 seconds of runtime with the same scale. This is because of the dynamic nature of the proposed algorithm. Considering the nature of the problem and its convergence speed, number of iterations and runtime of the algorithm will be different. It should be noted that, in

Table 2: Computational results.

Number of jobs	Optimal solution (GAMS)		Proposed GA		DBF		BBF		
	Objective function	Computational time	Best solution	Computational time	Error	Solution	Error	Solution	Error
20	110	0.656	110	4.509	0	110	0.00	115	0.05
20	111	0.5	111	5.355	0	113	0.02	120	0.08
20	85	0.125	85	4.088	0	85	0.00	85	0.00
20	155	0.781	155	4.041	0	156	0.01	163	0.05
20	154	0.344	154	5.032	0	154	0.00	174	0.13
20	145	0.687	145	5.340	0	145	0.00	145	0.00
20	140	0.391	140	4.927	0	140	0.00	140	0.00
20	212	0.219	212	5.032	0	212	0.00	233	0.10
20	105	0.266	105	4.988	0	105	0.00	115	0.10
30	198	0.859	198	7.896	0	210	0.06	227	0.15
30	205	0.468	205	6.655	0	205	0.00	205	0.00
30	132	2.218	132	5.274	0	132	0.00	132	0.00
30	160	0.531	160	3.998	0	160	0.00	160	0.00
30	205	0.922	205	4.496	0	205	0.00	210	0.02
30	152	0.203	152	5.837	0	157	0.03	154	0.01
30	185	0.203	185	4.923	0	185	0.00	187	0.01
30	210	1.078	210	7.012	0	214	0.02	234	0.11
30	200	0.937	200	5.670	0	200	0.00	205	0.03
40	238	2.375	238	9.297	0	307	0.29	278	0.17
40	640	2.407	640	8.380	0	640	0.00	647	0.01
40	225	2.781	225	7.883	0	233	0.04	235	0.04
40	249	3.5	249	7.291	0	261	0.05	260	0.04
40	388	2.937	388	6.503	0	388	0.00	428	0.10
40	246	0.281	246	7.074	0	246	0.00	250	0.02
40	305	3.36	305	6.560	0	305	0.00	310	0.02
40	192	3.219	192	7.072	0	208	0.08	203	0.06
40	227	2.047	227	7.561	0	251	0.11	245	0.08
50	300	5.922	300	7.857	0	300	0.00	304	0.01
50	256	6.312	256	14.683	0	311	0.21	280	0.09
50	305	0.344	305	8.624	0	410	0.34	366	0.20
50	294	6.282	294	8.949	0	314	0.07	310	0.05
50	276	7.063	276	9.795	0	331	0.20	280	0.01
50	268	4.031	268	8.601	0	272	0.01	270	0.01
50	264	5.344	264	8.371	0	275	0.04	285	0.08
50	277	3.375	277	8.743	0	277	0.00	284	0.03
50	560	5.656	560	8.867	0	560	0.00	640	0.14
100	627	53.109	627	12.742	0	776	0.24	713	0.14
100	529	30.828	529	14.579	0	529	0.00	529	0.00
100	474	25.609	474	12.481	0	474	0.00	480	0.01
100	708	29.562	708	16.054	0	861	0.22	788	0.11
100	545	1.563	545	13.352	0	545	0.00	545	0.00
100	459	1.078	459	74.192	0	459	0.00	459	0.00
100	524	36.125	524	21.903	0	715	0.36	592	0.13

Table 2: Continued.

Number of jobs	Optimal solution (GAMS)		Proposed GA		DBF		BBF		
	Objective function	Computational time	Best solution	Computational time	Error	Solution	Error	Solution	Error
100	1608	24.125	1608	24.212	0	1608	0.00	1688	0.05
100	502	34.344	502	23.093	0	529	0.05	522	0.04
250	1832	357.969	1832	24.986	0	1832	0.00	2148	0.17
250	1400	455.625	1400	21.001	0	1400	0.00	1400	0.00
250	1331	128.265	1331	24.777	0	1331	0.00	1331	0.00
250	1310	245.922	1310	23.629	0	1334	0.02	1316	0.00
250	1415	1151.047	1415	22.888	0	1415	0.00	1415	0.00
250	1232	1797.672	1232	23.156	0	1232	0.00	1232	0.00
250	1386	1380.265	1386	26.281	0	1386	0.00	1386	0.00
250	1371	267.672	1371	28.730	0	1371	0.00	1382	0.01
250	1229	797.484	1229	25.392	0	1258	0.02	1229	0.00
300	1451	756.813	1451	27.272	0	1506	0.04	1456	0.00
300	1564	1307.328	1564	28.163	0	1650	0.05	1606	0.03
300	3333	794.718	3333	25.770	0	3336	0.00	3606	0.08
300	1567	1596.406	1567	25.006	0	1567	0.00	1567	0.00
300	1364	1364.328	1384	26.174	0.01	1864	0.37	1772	0.30
300	1617	1407.594	1617	39.732	0	1697	0.05	1650	0.02
300	1627	505.453	1627	25.355	0	1674	0.03	1627	0.00
300	1573	2046.297	1573	42.888	0	1573	0.00	1573	0.00
300	1587	932.031	1587	25.198	0	1587	0.00	1589	0.00
350	1844	2078.485	1844	26.676	0	2503	0.36	2106	0.14
350	1984	4810.844	1984	41.475	0	1984	0.00	2000	0.01
350	2674	2869.516	2735	25.839	0.02	3259	0.22	2954	0.10
350	1942	4259.25	1942	40.685	0	2610	0.34	2353	0.21
350	1920	5221.468	1920	27.500	0	1942	0.01	1944	0.01
350	1876	353.625	1876	27.440	0	2027	0.08	1955	0.04
350	1875	648.7	1876	95.350	0	2542	0.36	2153	0.15
350	5110	1688.516	5110	43.858	0	5110	0.00	5390	0.05
350	1937	2038.391	1937	29.710	0	2043	0.05	1949	0.01
400	2139	2250.391	2139	28.537	0	2139	0.00	2139	0.00
400	2112	7925.203	2116	38.906	0	2568	0.22	2160	0.02
400	4048	29977.14	4048	26.308	0	4048	0.00	4440	0.10
400	2156	3970.235	2156	39.168	0	2245	0.04	2157	0.00
400	2142	4132.75	2142	48.098	0	2264	0.06	2178	0.02
400	3016	4816.89	3016	101.472	0	3674	0.22	3336	0.11
400	2024	8860.75	2024	86.422	0	2024	0.00	2024	0.00
400	3210	2216.234	3234	75.483	0.01	3400	0.06	3746	0.17
400	5804	3560.797	5804	81.728	0	5804	0.00	6124	0.06
500	2616	8.7.9.25	2621	74.914	0	2621	0.00	2623	0.00
500	2901	3.10.35.859	2901	77.115	0	3052	0.05	2953	0.02
500	2687	9.34.20.062	2687	72.801	0	2687	0.00	2687	0.00
500	2680	5.0.47.578	2680	78.161	0	2793	0.04	2695	0.01
500	2657	8.57.48.344	2657	59.477	0	2657	0.00	2675	0.01

Table 2: Continued.

Number of jobs	Optimal solution (GAMS)		Proposed GA		DBF		BBF		
	Objective function	Computational time	Best solution	Computational time	Error	Solution	Error	Solution	Error
500	2782	5.58.37.266	2792	64.315	0	2895	0.04	2794	0.00
500	2861	3.15.48.468	2861	62.425	0	2861	0.00	2875	0.00
500	2552	4.2.48.437	2552	52.891	0	2628	0.03	2552	0.00
500	2842	6.16.25.563	2842	73.716	0	2917	0.03	2845	0.00
700	—	—	3717	126.470	—	3871	0.04	3745	0.01
700	—	—	3825	113.280	—	3984	0.04	3835	0.01
700	—	—	3650	92.464	—	3650	0.00	3656	0.00
700	—	—	3778	76.992	—	3778	0.00	3779	0.00
700	—	—	5593	107.256	—	6380	0.14	5796	0.04
700	—	—	3822	122.658	—	3968	0.04	3830	0.00
700	—	—	5537	154.857	—	6327	0.14	5752	0.04
700	—	—	3763	97.774	—	4567	0.21	3820	0.02
700	—	—	3859	104.507	—	3859	0.00	3863	0.00
1000	—	—	5331	116.954	—	6488	0.22	5450	0.02
1000	—	—	6301	151.998	—	7175	0.14	6536	0.04
1000	—	—	6299	139.229	—	7191	0.14	6548	0.04
1000	—	—	5165	143.560	—	5302	0.03	5165	0.00
1000	—	—	5481	159.823	—	6654	0.21	5590	0.02
1000	—	—	5876	224.859	—	7286	0.24	6215	0.06
1000	—	—	5185	283.944	—	5334	0.03	5200	0.00
1000	—	—	8099	159.319	—	9155	0.13	8312	0.03
1000	—	—	5289	224.859	—	5409	0.02	5289	0.00
2000	—	—	10569	283.944	—	11159	0.06	10840	0.03
2000	—	—	12593	159.823	—	14517	0.15	13201	0.05
2000	—	—	15943	250.156	—	17582	0.10	16248	0.02
2000	—	—	10589	204.176	—	10825	0.02	10589	0.00
2000	—	—	10746	255.996	—	10746	0.00	10757	0.00
2000	—	—	11613	248.525	—	14477	0.25	12276	0.06
2000	—	—	10781	195.705	—	10781	0.00	10783	0.00
2000	—	—	10934	236.549	—	11384	0.04	10949	0.00
2000	—	—	11086	197.456	—	11086	0.00	11087	0.00

Table 2 for the problems of size 500, the time needed to solve the problem optimally is written as hour.minute.second but all other times are written in seconds.

4. Conclusion

In this paper, a single machine scheduling problem with machine interruption due to the periodic maintenance process and constraint on the maximum number of jobs processed in each batch on the machine is considered. We proposed a new mixed integer programming formulation to solve this problem, and considering the problem belongs to the NP-Hard class, trying to solve the problem optimally may cause extremely long computational time, so we proposed a dynamic genetic algorithm with two stop criteria which, as computational results show, can achieve good solution (and in most cases the optimal solution) in considerably shorter time.

For the interested researchers in this field, as the future studies, it is suggested to compare the performance (due to time factor) of the proposed model in this paper with the model proposed by Hsu et al. in [26] or to compare the performance of other metaheuristic methods such as simulated annealing, tabu search, or PSO on this problem. Another interesting future study is to consider the jobs due dates and to formulate the problem such that the number of tardy jobs is minimized.

References

[1] M. Pinedo, *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, Upper Saddle River, NJ, USA, 2002.

[2] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[3] M. Pinedo and E. Rammouz, "A note on stochastic scheduling on a single machine subject to breakdown and repair," *Probability in the Engineering and Informational Sciences*, vol. 2, pp. 41–49, 1988.

[4] J. Birge, J. B. G. Frenk, J. Mittenthal, and A. H. G. R. Khan, "Single-machine scheduling subject to stochastic breakdowns," *Naval Research Logistics*, vol. 37, no. 5, pp. 661–677, 1990.

[5] E. Frostig, "A note on stochastic scheduling on a single machine subject to breakdown-the preemptive repeat model," *Probability in the Engineering and Informational Sciences*, vol. 5, pp. 349–354, 1991.

[6] I. Adiri, J. Bruno, E. Frostig, and A. H. G. R. Kan, "Single machine flow-time scheduling with a single breakdown," *Acta Informatica*, vol. 26, no. 7, pp. 679–696, 1989.

[7] I. Adiri, E. Frostig, and A. H. G. R. Kan, "Scheduling on a single machine with a single breakdown to minimize stochastically the number of tardy jobs," *Naval Research Logistics*, vol. 38, no. 2, pp. 261–271, 1991.

[8] M. S. Akturk, J. B. Ghosh, and E. D. Gunes, "Scheduling with tool changes to minimize total completion time: a study of heuristics and their performance," *Naval Research Logistics*, vol. 50, no. 1, pp. 15–30, 2003.

[9] M. S. Akturk, J. B. Ghosh, and E. D. Gunes, "Scheduling with tool changes to minimize total completion time: basic results and SPT performance," *European Journal of Operational Research*, vol. 157, no. 3, pp. 784–790, 2004.

[10] K. H. Ecker and J. N. D. Gupta, "Scheduling tasks on a flexible manufacturing machine to minimize tool change delays," *European Journal of Operational Research*, vol. 164, no. 3, pp. 627–638, 2005.

[11] J. S. Chen, "Optimization models for the tool change scheduling problem," *Omega*, vol. 36, no. 5, pp. 888–894, 2008.

[12] Y. C. Choi and Y. D. Kim, "Tool replacement policies for a machining centre producing multiple types of products with distinct due dates," *International Journal of Production Research*, vol. 39, no. 5, pp. 907–921, 2001.

[13] D. L. Yang, C. L. Hung, C. J. Hsu, and M. S. Chem, "Minimizing the makespan in a single machine scheduling problem with a flexible maintenance," *Journal of the Chinese Institute of Industrial Engineers*, vol. 19, no. 1, pp. 63–66, 2002.

[14] X. Qi, T. Chen, and F. Tu, "Scheduling the maintenance on a single machine," *Journal of the Operational Research Society*, vol. 50, no. 10, pp. 1071–1078, 1999.

[15] J. S. Chen, "Single-machine scheduling with flexible and periodic maintenance," *Journal of the Operational Research Society*, vol. 57, no. 6, pp. 703–710, 2006.

[16] C. Low, M. Ji, C. J. Hsu, and C. T. Su, "Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance," *Applied Mathematical Modelling*, vol. 34, no. 2, pp. 334–342, 2010.

[17] C. Low, C. J. Hsu, and C. T. Su, "A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance," *Expert Systems with Applications*, vol. 37, no. 9, pp. 6429–6434, 2010.

[18] C. J. Liao and W. J. Chen, "Single-machine scheduling with periodic maintenance and nonresumable jobs," *Computers and Operations Research*, vol. 30, no. 9, pp. 1335–1347, 2003.

[19] W. J. Chen, "Minimizing total flow time in the single-machine scheduling problem with periodic maintenance," *Journal of the Operational Research Society*, vol. 57, no. 4, pp. 410–415, 2006.

[20] W. J. Chen, "Minimizing number of tardy jobs on a single machine subject to periodic maintenance," *Omega*, vol. 37, no. 3, pp. 591–599, 2009.

[21] S. H. Lee and I. G. Lee, "Heuristic algorithm for the single machine scheduling with periodic maintenance," *Journal of the Korean Institute of Industrial Engineers*, vol. 34, pp. 318–327, 2008.

[22] H. C. Lau and C. Zhang, "Job scheduling with unfixed availability constraints," in *Proceedings of the 35th Meeting of the Decision Sciences Institute*, pp. 4401–4406, Boston, Mass, USA, 2004.

[23] C. J. Liao, C. M. Chen, and C. H. Lin, "Minimizing makespan for two parallel machines with job limit on each availability interval," *Journal of the Operational Research Society*, vol. 58, no. 7, pp. 938–947, 2007.

[24] M. Dell'Amico and S. Martello, "Bounds for the cardinality constrained P‖Cmax problem," *Journal of Scheduling*, vol. 4, no. 3, pp. 123–138, 2001.

[25] D. L. Yang, C. J. Hsu, and W. H. Kuo, "A two-machine flowshop scheduling problem with a separated maintenance constraint," *Computers and Operations Research*, vol. 35, no. 3, pp. 876–883, 2008.

[26] C. J. Hsu, C. Low, and C. T. Su, "A single-machine scheduling problem with maintenance activities to minimize makespan," *Applied Mathematics and Computation*, vol. 215, no. 11, pp. 3929–3935, 2010.

[27] C. Y. Lee and S. D. Liman, "Single machine flow-time scheduling with scheduled maintenance," *Acta Informatica*, vol. 29, no. 4, pp. 375–382, 1992.

[28] J. H. Holland, *Adaptation in Natural and Artificial System*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[29] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, NY, USA, 1991.

[30] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, New York, NY, USA, 1989.

[31] T. Back, D. Fogel, and Z. Michalawecz, *Handbook of Evolutionary Computation*, Oxford University Press, Oxford, UK, 1997.

[32] M. Obitko, "Introduction to genetic algorithms," 1998, http://www.obitko.com/tutorials/genetic-algorithms.

[33] G. Syswerda, "uniform crossover in genetic algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 2–9, 1989.

[34] A. Hamidinia, S. Khakabimamaghani, M. M. Mazdeh, and M. Jafari, "A genetic algorithm for minimizing total tardiness/earliness of weighted jobs in a batched delivery system," *Computers and Industrial Engineering*, vol. 62, no. 1, pp. 29–38, 2012.