Int J Adv Manuf Technol (2007) 31: 994–1000
DOI 10.1007/s00170-005-0261-0

ORIGINAL ARTICLE

Tung-I Tsai

A genetic algorithm for solving the single machine earliness/tardiness problem with distinct due dates and ready times

Received: 16 June 2004 / Accepted: 28 June 2005 / Published online: 12 July 2006

Abstract The genetic algorithm (GA) is a heuristics, and commonly used to solve combinational problems. It has been proven to have high effectiveness and efficiency in many application areas. However, a successful GA application requires adapting the algorithm to the characteristics of a problem. In the past decades, various articles on single machine earliness and tardiness (SET) problem using a heuristic approaching have been published. Yet, there are few research addressing the area of the SET problem with distinct due dates and ready times for jobs. In this paper, a designed GA, modified optimal timing procedure, which starts the searching with a feasible solution obtained by applying the EXP-ET rule developed by Ow and Morton is developed for the SET problem. Computational results in the provided experiment show that the designed GA improves the SET solution in both quality and efficiency.

Keywords Earliness and tardiness · Genetic algorithm · Ready time · Scheduling · Single machine

1 Introduction

Because of the exploration of the just-in-time (JIT) operations management philosophy, a lot of literature has been published on this topic in the last decade. Most of them, however, focus on problems with jobs of common due window or common due date, rather than on the problem of distinct due dates [1]. In addition, almost all the published literature assumed that all jobs are available in the beginning. Practically speaking, jobs often have distinct ready times for the process in the real world. This research thus aims at developing a scheduling procedure concerning the single machine earliness and tardiness (SET) problem which contains jobs with distinct due dates and ready times, and hopes that it can be systematically extended to multi-machine environments in future studies.

The SET problems with common due date have directed many researchers' attention to finding effective solving procedures. However, as pointed out by Baker and Scudder [1], the SET problems with distinct due dates demanded further extended works. Roughly speaking, there are two main problem types that the research efforts have been focused on in recent years. One allows the insertion of idle time in the system, and the other one does not. Ow and Morton [2] proposed the priority dispatching rules along with a filtered beam search algorithm for systems; Abdul-Razaq and Potts [3] obtained the lower bound by applying the dynamic programming state-space relaxation, and Liaw [4] proposed the Lagrangian relaxation based procedure and applied it to the branch and bound algorithm. All of the above researchers were not inserting idle time.

On inverse assertions, Baker and Scudder [1] figured out that the assumption of not inserting idle time is inconsistent with the objective function, and claimed that the procedures should comprise two parts for problems allowing idle time: an optimal sequencing procedure and an optimal timing procedure. Yano and Kim [5] proposed a branch and bound algorithm as the optimal sequencing and formulated the SET problem in a dynamic programming model for the optimal timing procedure. Lee and Choi [6] used a GA to sequence jobs and presented an optimal timing procedure for jobs with general penalty weights; Mazzini and Armentano [7] developed a moving heuristics for sequencing and timing; Wan and Yen [8] utilized a similar approach as Lee and Choi [6] for optimal timing but they adopted the tabu search (TS) approach for sequencing jobs.

As an extended work of the SET problems with distinct due dates, we consider different ready times should be included in the model. The model reflects the fact that many companies with supply chain management systems require their suppliers moving toward lean production systems or agile manufactories. Therefore, a ready time

T.-I. Tsai (✉)
Institute of Business and Management,
Tainan Woman's College of Arts & Technology,
529, Chung-Cheng Rd., Yung-Kung,
Tainan 710, Taiwan, Republic of China
e-mail: mj1786@anet.net.tw
Tel.: +886-6-3311495
Fax: +886-6-3310748

constraint must be considered in such a production environment.

Since SET are NP-hard problems [1, 9], this research tries to construct an algorithm which can handle large problems and also provide high solution quality. The principal strategy for achieving the objective is to employ the GA algorithm (provided with an efficient searching design) as the optimal sequencing procedure to obtain a job sequence, and subsequently to employ the optimal timing procedure for inserting idle times in the job sequence to obtain an schedule for the SET problems. We outline the approach in these steps as:

Step 1.
Use the priority dispatching rule proposed by Ow and Morton [2] to generate the initial sequence and go to step 3.

Step 2.
Utilize the designed GA to generate a job sequence.

Step 3.
Apply the optimal timing procedure to obtain a schedule and evaluate the schedule to check whether it satisfies the stopping criterion or not. If yes, then stop; otherwise go to step 2.

A detailed description of the optimal timing procedure, initial schedule generation, and the designed GA will be presented in Sect. 4. The remainder of this research will be organized as follows: Sect. 2 provides the problem definition, notions, objective formulations, and the adjacent condition; Sect. 3 presents the priority dispatching rules proposed by Ow and Morton [2]; Sect. 4 describes the optimal timing procedure and the GA procedure. The computational results and the comparisons between the proposed study and the branch and bound algorithm are presented in Sect. 5. Finally, the conclusions of this research are portrayed in Sect. 6.

2 Problem formulation

The SET problem dealt with in this paper focuses on scheduling n jobs with positive integer processing times p_j, and due dates d_j on a single machine. Each job j has a distinct ready time r_j and a finish time f_j. Some assumptions are also followed, such as that the setup time for each job is included in the processing time, and preemption is not allowed.

Lee and Choi [6] assumed all jobs are available at time zero. However, most companies will generally not prepare all materials for all jobs at time zero. Therefore, the objective of the SET problem approach here is minimizing total earliness and tardiness penalties with distinct due dates, distinct general early-tardy weights, and distinct ready times.

For future convenience, the notations used throughout this research are listed as follows:

- $N=\{1,2,\ldots,n\}$: the set of jobs to be processed on the machine.
- p_j: the processing time of job j.
- d_j: the due date of job j.
- α_j: the unit earliness penalty for job j.
- β_j: the unit tardiness penalty for job j.
- r_j: The ready time of job j.
- f_j: The completion time of job j.
- $t_j=\max(r_j, f_j)$: the starting time of job j given job i precedes immediately job j.
- $E_j=\max\{0, f_j-d_j\}$: the earliness of job j.
- $T_j=max\{0, f_j-d_j\}$: The tardiness of job j.
- $g_j=\alpha_j\times E_j\times+\beta_j\times T_j$: The earliness and tardiness penalties of job j, $\forall j\in N$.

Using the above notations, the objective function of the SET problem is formulated as:

$$Z = \min \sum_{j=1}^{n} g_j \tag{1}$$

Ow and Morton [2] presented an adjacency condition that interprets the condition necessary for all optimal schedules. This condition is also called dominance rule by Liaw [4] to produce a precedence relation between jobs. This precedence relation is the foundation of the priority dispatching rule mentioned in the next section.

2.1 Theorem of adjacency condition

All adjacent pairs of jobs in an optimal SET schedule must satisfy the following condition in an optimal schedule:

$$\beta_i p_j - \Omega_{ij}(\beta_i + \alpha_i) \geq \beta_j p_i - \Omega_{ji}(\beta_j + \alpha_j) \tag{2}$$

where job i precedes immediately job j, and Ω_{ij} and Ω_{ji} are defined as

$$\Omega_{ij} = \begin{cases} 0 & if\ s_i \leq 0 \\ s_i & if\ 0 \leq s_i \leq p_j \\ p_j & otherwise \end{cases} \tag{3}$$

where $s_i=d_i-(t+p_i)$ is the slack of job i.

3 Priority dispatching rule

It is well known that a good initial solution for a GA to start with has significant influence on its performance [6]. In this

section, the good initial solution is constructed by applying the priority dispatching rule. The algorithm proposed by Ow and Morton [2] called the exponential earliness/tardiness (EXP-ET) rule has exhibited good performance for most problem settings.

The EXP-ET rule utilizes the priority index expressed below at any time point t when the machine is available.

$$I_i(t) = \begin{cases} \frac{\beta_i}{p_i} & \text{if } s_i \le 0 \\ \frac{\beta_i}{p_i} \exp\left[-\frac{(\alpha_i+\beta_i)s_i}{\alpha_i \bar{p}}\right] & \text{if } 0 < s_i \le \left(\frac{\beta_i}{\alpha_i+\beta_i}\right) k\bar{p} \\ \alpha_i^{-2}\left[\frac{\beta_i}{p_i} - \frac{(\alpha_i+\beta_i)s_i}{kp_i\bar{p}}\right]^3 & \text{if } \left(\frac{\beta_i}{\alpha_i+\beta_i}\right) k\bar{p} < s_i \le k\bar{p} \\ \frac{-\alpha_i}{p_i} & \text{otherwise} \end{cases}$$

where $s_i=d_i-t-p_i$ is the slack of job i at time t, $\bar{p}$ is the average processing time of all remaining ready jobs and k is a look-ahead parameter which is determined through experiments, and should reflect the average clash number of jobs in the future when a sequencing decision is made. A clash of jobs at time t is defined as a need of sequencing more than one job at time t. Generally speaking, the tighter the due dates are, the more job clashes will be incurred, and a larger value of k should be used. Contrarily, when due dates are uniformly distributed, k should be small since few jobs will clash with another. For a single machine, according to Vepsalainen and Morton [10], the used value of k usually lies between 1 and 3. In this research, k is set equal to 2, according to our preliminary experiments. At the time a machine is available, the indices of all remaining ready jobs are calculated and the job with the highest index is chosen to be processed next. If there is no ready job at time t, then sets t equal to the earliest ready time of the unscheduled jobs, and calculates the indices of all remaining ready jobs and selects the job with the highest index to be processed next. Ow and Morton [2] have pointed out that the EXP-ET rule will focus on the tardiness penalty as the slack time of the job becomes small, and conversely, when the slack time is large the earliness cost will dominate.

The work done by Ow and Morton is integrated in our study to improve the GA procedure for SET problems. The integration contributes in finding a good initial starting point for the design GA.

4 The optimal timing procedure and the GA procedure for the SET problem

4.1 The optimal timing procedure

Lee and Choi [6], and Wan and Yen [8] presented similar optimal timing procedures to obtain the most valid completion time for jobs. They used job blocks as basic units for handling optimal timing procedure. Jobs consecutively scheduled without idle time form a job block, and a job block can be shifted toward the minimum point until one of the following three conditions are encountered:

1. The start time of the first job of a job block is zero.
2. The shift operation has reached the minimum point of a block.
3. The start time of the first job of a job block equals to the completion time of the last job of the preceding job block.

However, their researches have a common assumption: all jobs are available at time zero. This assumption isn't made in this research, as we assume a distinct ready time for each job. Consequently, while on processing the optimal timing procedure the start time of the first job of a block must be the ready time of that job. This means the block cannot be shifted backward and only shifting forward is valid in shifting operation. In stead of a job-by-job shifting operation in the procedure proposed by Lee and Choi [6] and Wan and Yen [8], we use "job run" (defined below) as the manipulation unit in the shifting procedure.

Before describing the details of the procedure, some definitions are given as:

Supposed there are l blocks in a schedule, let S_k be a partial sequence having k_n jobs, $S_k = \{[k_1], [k_2], \ldots, [k_n]\}$, and is the kth block in the schedule, k=1,2,…,l. We divide a block into several (may be only one at an extreme case) subunits called job runs. A tardy job run is defined as the consecutive tardy jobs being gathered backward from the last job of each block, denoted *TR*; and the consecutive early jobs form an early job run, denoted *ER*.

Four types of job runs in a block are shown in Fig. 1.

The detailed steps of the optimal timing procedure are described in the following steps:

Step 1.

Start from the last job block, and let k=l.

Step 2.

(forward operation)

Given a job block S_k, if the last job run is an *ER*, that is type 1 or type 3 job runs, shift this run forward until one of the following two conditions are encountered, and go to step 4. Otherwise, for type 2 job run go to step 5 (because there is no improvement can be obtained by shifting forward); for type 4 job run go to step 3.

1. The minimum point has been reached.

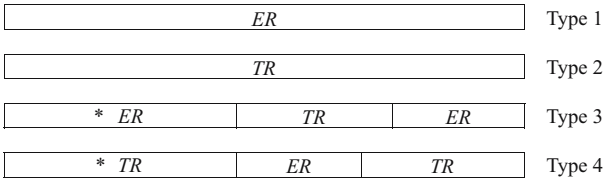

Fig. 1 Four types of job runs in a block; * means whatever

2. The completion time of the last job in this run equals the start time of the first job of the next block.

Step 3.
(pair operation)
If the total penalties of the *ER* is larger than the total penalties of the *TR* in the paired *TR* and *ER*, shifts this pair forward until one of the above mentioned two conditions are encountered; otherwise, keep the pair unchanged and go to step 4.

Step 4.
If the left jobs are all in an *ER*, go to step 2; if the left jobs are all in a *TR*, go to step 5 (because the *TR* can not be improved by a shift forward); if the left jobs are not in the above two cases, perform the same operation for the next pair of *TR* and *ER*, and repeat the step 3 and step 4 until all the runs in S_k are checked.

Step 5.
Let $k=k-1$ and repeat step 2 to step 4 until $k=1$.

4.2 The designed GA procedure

As mentioned in Sect. 3, the schedule obtained by EXP-ET priority rule is only a good one, and can not guarantee the global optimal one. Ow and Morton used the EXP-ET priority rule to evaluate a partial sequence generated by beam search and decide which node should be retained. They claimed that the beam search used is a non-backtracking algorithm; once a node is excluded from the beam, it will be lost and the chance to find an optimal solution is gone forever. Ow and Morton also reported that their work was accomplished with an average deviations of 3% from the optimal solutions. The results proved that the EXP-ET priority rule can provide a good initial sequence for the GA.

GA based heuristic procedures for scheduling problems were successful in many applications. The algorithm was initially proposed by John Holland in the 1970s [11]; it is an iterative improvement procedure with the ability to avoid from a local optimal solution. The whole process adapted for this research can be sketched briefly as follows:

Step 1.
Use the EXP-ET rule to obtain an initial solution x_0, and use it to generate an initial population by adjacent interchanging.

Step 2.
Reproduce a new string as the new offspring. Evaluate the new offspring and replace the old population.

Step 3.
Check whether the stopping criterion is satisfied. If yes, then stop; otherwise go to step 2. The stopping criterion can be either reaching the maximum number of generations or having found no improvement after a certain number of successive generations.

The structural elements of the designed GA procedure are described in detail in Sect. 4.2.1 to Sect. 4.2.8.

4.2.1 Encoding

Chromosomes are encoded typically in binary vectors in GA. This simple representation has a drawback of a long vector in large scaled scheduling problems for detracting from the efficiency. This flaw can, however, be resolved by considering that the jobs appear only one time in the work list and by adopting the order encoding method in establishing chromosomes. For example, assuming six jobs are scheduled with an encoding as: B, C, A, D, F, and E. It means that job B is produced first, job C is produced second, and so on. The sequence of chromosomes is set the same as the sequence of jobs.

4.2.2 Initial population

The initial solution population can be generated by random or heuristic generation. The most popular and traditional method is random generation, but this may cut down the efficiency of GA in large-scale scheduling problems. It is valid making use of the characteristics of the problem and adopting an appropriate heuristics for generating the initial population. For example, Lee and Choi [6] utilized the earliest due date (EDD) and earliest starting time (EST) rules in figuring out the initial solutions. Following that effort, a similar approach was made in our study. This study combines using the characteristics of a problem, accepting the EXP-ET algorithm, and extending the adjacent interchange to generate the population. The population size is set as the number of jobs. In this research, the size of population is maintained equal to the number of jobs on each step.

4.2.3 Fitness function

To determine which chromosomes will survive, the fitness function plays a similar role as nature does in its own way. The chromosome with a higher fitness value has higher probabilities to survive. In addition, after a few generations in the GA process, the fitness values of chromosomes will approximate each other, and it is hard to identify the difference between chromosomes. Hence, the fitness function in this study is purposely constructed as $fv_j = \left(g_j - g^*\right)^2$, where g^* is the biggest objective value in the population.

4.2.4 Reproduction/selection

The commonly used selection technique is the roulette wheel selection. It uses the total fitness values as the denominator and the individual value as the numerator. This implies that the one which has a higher fitness value has a higher probability to be selected. This technique requires extensive calculations in large-scale problems, as shown in the detailed steps in Goldberg [12] and Li et al.

[13]. We modified the tournament selection method to process this procedure, which is described as the following:

Step 1.
Pick two or more chromosomes randomly from the current population (Based on our computational experience, two chromosomes chosen randomly are suggested).

Step 2.
Compare the fitness values of the two chromosomes and keep the one with the higher fitness value.

Step 3.
Utilize the Boltzmann distribution to decide whether the one with lower fitness value should be discarded or not.

The Boltzmann distribution implements the following formula calculating the acceptance probability of the one with a lower fitness value:

$$p = \exp\left[\frac{fv_{low} - fv_{high}}{T}\right].$$

Where T is initially set at 10^N, N is the number of jobs, and has a decreasing rate of 0.9 from generation to generation. After calculating the value of p, we generate a random number from a uniform distribution between 0 and 1. If the random number is smaller than p, accept the one with a lower fitness value; otherwise, abandon it and go to step 4.

Step 4
Repeat the process until the population size is fulfilled.

4.2.5 Crossover

Crossover is the fundamental operator of GA. It recombines the parents and produces the offspring. In its process of exchanging genes, however, may lose some good features of parents and result in inappropriate solutions. To avoid the inappropriate solutions, various crossover techniques have been developed, such as the uniform order-based crossover proposed by Davis in 1991 [14], the partially matched crossover, the cycle crossover, and the order crossover seen in Goldberg [12]. The uniform order-based crossover seems to be the most suitable for scheduling problems except that the binary template used in the uniform order-based crossover is generated randomly; it cannot be guaranteed not to lose good features of the parents. In this research, a modified uniform order-based is proposed. The principal steps include:

Step 1.
Calculate the exchange number. This number equals to the population size times the crossover rate. Here, we set the crossover rate as 0.7.

Step 2.
Choose a pair of chromosomes randomly from the population obtained from the Selection procedure.

Step 3.
Generate the binary template.

a. Check parent 1 where the adjacent pair genes obey the Adjacency Condition, and mark the binary template 1 at the same position. Since we do not swap the adjacency pair of jobs, it is unconcerned with whether the start time will be changed or not.
b. Check parent 2 where the adjacent pair genes obey the adjacency condition, and mark the binary template 0 at the same position.
c. Generate 1 or 0 for the binary template step 4 used at the overlapped and unmarked position randomly.

Step 4.
Copy the genes from parent 1 (or parent 2) to offspring 1 (or offspring 2) whenever the binary template contains a "1" (or "0").

Step 5.
Fulfill the unfilled position of offspring 1 (or offspring 2) with genes which have not appeared in offspring 1 (or offspring 2) in the same order they appear on parent 2 (or parent 1).

Step 6.
Repeat steps 2–5 until the exchange number has been fulfilled.

4.2.6 Mutation

The purpose of mutation is departing from the local optimal solution toward the global optimal one. This unique feature makes GA different from other traditional search methods that sink sometimes in a local optimal position easily and cannot escape from it. GA can, nonetheless, depart from it through the mutation process. The principal steps of mutation in here are set as:

Step 1
Decide a mutation rate. If the mutation rate is set too high, GA will degenerate to random search; on the other hand, if it is too low, GA will become a traditional search method. Here, we choose a mutation rate equal to 0.1 according to our pilot experiments.

Step 2
Execute the position-based mutation. It is a simple random culling of two genes from the chosen chromosome and exchanging the two genes.

Step 3
Repeat step 2 to fulfill the candidate list, and pick up the one with the highest fitness value to replace the chosen chromosome. The length of the candidate list equals the size of population.

4.2.7 Representation

After the three operators of GA-selection, crossover and mutation-the parent population will produce an offspring

population and the offspring population will replace the parent population. There are two different directions to execute representation: whole representation and elitism representation. The whole representation replaces the parent population totally, no matter whether the offspring population is better or worse. It decreases the effectiveness and efficiency of a GA in large-scaled scheduling problems. Elitism representation is selected in this research and implemented by keeping chromosomes with the highest fitness value of the parent population and offspring population. This technique guarantees that the new population will never decline.

4.2.8 The stopping criterion

After the representation step, the minimum objective value (g_j) is recorded in order to conclude whether the GA procedure will stop or not. The GA procedure can be stopped on two conditions:

1. After a certain number of generations and no improvement on the objective value is obtained.
2. After a certain predetermined number of generations.

Based on the pilot experiments of this research, the procedure will stop after 500 iterations or after 30 generations and no improvement of objective value has been made.

5 Computational results

Four hundred and fifty generated problems were tested and the results were compared with the solutions obtained by the branch and bound algorithm proposed by Chang [15]. The GA procedure was executed with Microsoft Visual C++ software on a PC with Intel Pentium IV 1.6 GHz CPU and 512 MB DDR RAM.

Table 1 Experiment results when due date range R=0.5

					CPU seconds	
R	T	n	gap_{ET}(%)	gap_{GA}(%)	GA	Branch and bound
0.5	0.2	20	139.32	0.00	0.45	35.58
		30	139.72	0.00	1.02	436.8
		50	109.75	0.00	7.33	2070
		80	171.16	0.29	8.74	>6000
		100	162.99	0.36	34.60	>6000
	0.5	20	133.27	0.00	0.48	36.66
		30	136.84	0.00	1.10	420.56
		50	151.59	0.00	8.97	2206.4
		80	164.05	0.28	9.96	>6000
		100	162.07	0.39	35.94	>6000
	0.8	20	115.00	0.00	0.49	36.78
		30	134.28	0.00	1.05	399.76
		50	126.11	0.00	8.24	2331.8
		80	134.15	0.30	9.85	>6000
		100	167.16	0.35	36.03	>6000

Table 2 Experiment results when due date range R=1.0

					CPU seconds	
R	T	n	gap_{ET}(%)	gap_{GA}(%)	GA	Branch and bound
1.0	0.2	20	148.66	0.00	0.50	30.69
		30	137.82	0.00	1.48	411.76
		50	190.47	0.00	8.37	2046.8
		80	154.18	0.30	10.25	>6000
		100	128.52	0.36	36.26	>6000
	0.5	20	125.06	0.00	0.48	35.49
		30	127.33	0.00	1.86	416.96
		50	150.04	0.00	8.75	2226
		80	163.23	0.32	9.97	>6000
		100	154.18	0.40	35.81	>6000
	0.8	20	156.35	0.00	0.52	37.29
		30	135.04	0.00	1.54	424.32
		50	132.28	0.00	8.12	2127.8
		80	177.48	0.28	10.38	>6000
		100	133.69	0.37	35.79	>6000

5.1 Problem generation

The problems tested were generated using the method proposed by Potts and Van Wassenhove [16]. Researchers such as Liaw [4], Chang [15], Mazzini and Armentano [7], and Wan and Yen [8] also used this problem generating method in their work. The parameters required for testing in our study are generated as follows. The processing times p_j were uniformly and discretely distributed between 1 and 100. The unit tardiness penalties β_j were randomly generated from uniform distribution [1, 10] and the unit earliness penalties α_j were c times of β_j, where c was a uniform random variable from 0 to 1. The due dates were generated from $U\left[P\cdot\left(1-T-\frac{R}{2}\right), P\cdot\left(1-T+\frac{R}{2}\right)\right]$, where T is the tardiness factor, R is the due date range

Table 3 Experiment results when due date range R=1.5

					CPU seconds	
R	T	n	gap_{ET}(%)	gap_{GA}(%)	GA	Branch and bound
1.5	0.2	20	139.67	0.00	0.51	34.83
		30	147.79	0.00	1.13	412.4
		50	145.19	0.00	8.5	2048.4
		80	194.12	0.24	9.69	>6000
		100	118.21	0.37	36.18	>6000
	0.5	20	178.81	0.00	0.46	37.32
		30	148.99	0.00	1.2	421.84
		50	141.36	0.00	8.21	2153.4
		80	122.31	0.28	10.22	>6000
		100	131.83	0.36	35.5	>6000
	0.8	20	155.92	0.00	0.47	36.21
		30	140.46	0.00	1.15	423.84
		50	163.77	0.00	8.63	2060.8
		80	139.26	0.30	10.08	>6000
		100	141.93	0.35	36.05	>6000

and P is the sum of the processing times of all jobs. Three values were chosen for these two factors: T=0.2, 0.5, 0.8 and R=0.5, 1.0, 1.5. The ready time is distributed in a uniform distribution between 0 and P. The problem size n equals to 20, 30, 50, 80 and 100. Each combination of T, R and n generates 10 problem instances, i.e., a total of $3 \times 3 \times 5 \times 10$=450 problem instances were tested in the study.

5.2 Computational results

Computational results are presented in Tables 1, 2, 3. The gap value shown in column 4 and 5 in Table 1 through Table 3 is calculated using the following formulas.

$$gap_{ET} = \frac{g_{j(ET)} - g_{j(BB)}}{g_{j(BB)}} \times 100\%$$
$$gap_{GA} = \frac{g_{j(GA)} - g_{j(BB)}}{g_{j(BB)}} \times 100\%$$

where gap_{ET} and gap_{GA} are the gap values between the EXP-ET rule and the proposed GA procedure with the branch and bound procedure, respectively; $g_{j(GA)}$ is the total penalties obtained by the proposed GA procedure; $g_{j(BB)}$ is the total penalties obtained by the branch and bound procedure.

According to Tables 1, 2, 3, one can easily discover that both the branch and bound algorithm and the proposed GA procedure find the optimal solutions when the number of jobs is less than 50. For problems with more than 50 jobs, the proposed GA does not find optimal solutions. Because of the computational complexity, the branch and bound algorithm will be terminated if an optimal solution can not be reached in 6,000 s of CPU time. At the termination step, a lower bound was calculated using algorithm statement 1 given in Chang [15], and the lower bound was used as the final solution. According to the results, the gap_{ET} average is about 150%. The results shows that the proposed GA algorithm is very effective, even for problems with 100 jobs, the average gap being only about 0.4%. Besides, it also shows that the average gap reduces as the due date range increases. Note that the approach should be robust since the tables show that the results are independent of the tardiness factor and number of jobs.

The used CPU times in seconds for calculating the results are presented in column 6 for the GA, and in column 7 for the branch and bound of Tables 1, 2, 3. Because the used CPU time for the EXP-ET rule is almost zero, we do not list it in Tables 1, 2, 3. It is obvious that the proposed GA procedure is much more efficient than the branch and bound. The proposed GA procedure only takes less than 40 s to obtain a solution on average in problems with 100 jobs, whereas, the branch and bound needs more than 6,000 s. The number of jobs is the major factor of efficiency, and the due date range and tardiness factor have little influence on the solution quality and efficiency.

6 Conclusions

A designed GA procedure for enhancing the searching capability was developed in the paper for SET problems. The improvement was attributed to the integrating of a traditional sound heuristic search and the designed GA process.

The proposed GA has successfully generated the optimal solutions for problems with up to 50 jobs. In addition, the CPU time for obtaining an optimal solution of a problem with 100 jobs is less than 40 s. The results demonstrate that the proposed GA is an improvement over the branch and bound. Based on the intelligent capability of the GA in a flexible computing environment, the approach should be very promising in the industry applications when extending the study to a job shop environment with JIT practice.

References

1. Baker KR, Scudder GD (1990) Sequencing with earliness and tardiness penalties: a review. Oper Res 38:22–36
2. Ow P, Morton T (1989) The single machine early-tardy problem. Manage Sci 35:177–191
3. Abdul-Razaq T, Potts C (1988) Dynamic programming state-space relaxation for single-machine scheduling. J Oper Res Soc 39:141–152
4. Liaw CF (1999) A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. Comput Oper Res 26:679–693
5. Yano CA, Kim YD (1991) Algorithms for a class of single machine weighted tardiness and earliness problems. Eur J Oper Res 52:167–178
6. Lee CY, Choi JY (1995) A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. Comput Oper Res 22:857–869
7. Mazzini R, Armentano VA (2001) A heuristic for single machine scheduling with early and tardy costs. Eur J Oper Res 128:129–146
8. Wan G, Yen BP-C (2002) Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties. Eur J Oper Res 142:271–281
9. Garey MR, Tarjan RE, Wilfong GT (1988) One-processor scheduling with symmetric earliness and tardiness penalties. Math Oper Res 13:330–348
10. Vepsalainen A, Morton TE (1987) Priority rules for jobshops with weighted tardiness costs. Manage Sci 33:1035–1047
11. Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, MI
12. Goldberg DE (1989) Genetic algorithms in search, optimization & machine learning. Addison-Wesley, Boston
13. Li Y, Ip WH, Wang DW (1998) Genetic algorithm approach to earliness and tardiness production scheduling and planning problem. Int J Prod Econ 54:65–76
14. Davis L (1991) Handbook of genetic algorithms. Van Nostrand, Princeton
15. Chang PC (1999) A branch and bound approach for single machine scheduling with earliness and tardiness penalties. Comput Math Appl 37:133–144
16. Potts CN, Wassenhove LNV (1982) A decomposition algorithm for the single machine total tardiness problem. Oper Res Lett 1:177–181