A Genetic Algorithm for Single Machine Total Weighted Tardiness Scheduling Problem

Ning LIU*, M. ABDELRAHMAN**, Srini RAMASWAMY +

Abstract— **This paper presents a new genetic algorithm for the single machine total weighted tardiness scheduling problem, which is a strong NP-hard problem. The developmental focus has been on techniques to speed up execution in order to solve large-size problems. This genetic algorithm uses the natural permutation representation of a chromosome for encoding simplicity. Heuristic dispatching rules combined with a random method are used to create the initial population for improving (decreasing) the search space, consequently improving searching simplicity. Position-based crossover and order-based mutation operators are used for operator simplicity. The best members of the population during generations are used for searching simplicity, too. Extensive computational results for randomly generated problems with up to 500 jobs show the good performance and the efficiency of the developed algorithm.**

Keywords*:* Scheduling, Genetic algorithm, Local search

1. Introduction

The single machine total weighted tardiness problem is to schedule n jobs on a single machine to minimize the sum of the weighted tardiness of all the jobs. For arbitrary positive weights, it is strongly NP-hard [12]. Many scheduling problems, which do not have efficient, so-called polynomial time, optimal algorithms, are the so-called NP-hard problems; efficient optimal algorithms are unlikely to exist for these problems. An algorithm is referred to as a polynomial time algorithm when the number of iterations in the algorithm is polynomial in the size (n, the number of jobs) of the problem. Consider the various permutations of the n jobs of the total weighted tardiness problem to find the optimal schedule. Even for a modest-sized problem, complete enumeration is not computationally feasible since the complete enumeration requires the evaluation of n! sequences (e.g., a 20-job problem requires the evaluation of more than $2.4 * 10^{18}$ sequences) [22].

As single machine scheduling problems can provide help and insight into resolving, understanding, managing, and modeling more complex multi-machine scheduling problems, the single machine total weighted tardiness problem has received much attention in literature. The single machine total weighted tardiness problem has been tackled by enumerative algorithms: branch and bound algorithms [10, 17, 19, 22] and dynamic programming algorithms [21] to generate exact solutions that are guaranteed to be optimal. But the branch and bound algorithms are limited by computational times and the dynamic programming algorithms are limited by computer storage requirements, especially when the number of jobs is more than about 50 [4]. Thereafter, the problem has been extensively studied by heuristics -- solution procedures that generate good or even optimal solutions, but do not guarantee optimality.

These heuristics include heuristic dispatching rules [2, 11, 26] and local search heuristics. As there is no single best dispatching rule for all problem environments, in other words, dispatching rules do not consistently provide good quality solutions, in recent years, much attention has been devoted to local search heuristics [9, 14, 24]. These local search heuristics mainly include neighborhood search methods, such as descent methods, simulated annealing, threshold accepting, and tabu search [4, 6, 15, 19]; and genetic algorithms (GA) [3, 4, 13].

Crauwels et al. [4] compare the performance of a number of local search heuristics that have the binary representation, namely, descent methods, simulated annealing, threshold accepting, tabu search, and GA, for total weighted tardiness problems with 40, 50, and 100 jobs. The paper indicates that its binary encoded GA performs very well and requires comparatively little computation time; this binary encoded GA is also a viable alternative to other heuristic methods, especially in view of its small maximum relative deviations and modest computation time. Madureira et al. [13] also obtain good solutions in a short time by a natural permutation encoded GA for problems with 40 and 50 jobs. Avci et al. [3] propose a GA for problems with 200 jobs that use global and local dominance rules to improve the neighborhood structure of the search space and obtain almost the same results as those of Crauwels et al. [4].

In this paper, we present a new genetic algorithm for solving the single machine total weighted tardiness problem. Our algorithm is different from the algorithms in [3, 4, 13], and in chromosome representation, initial creation of the population, genetic operators, parameters selection, and population replacement. Technique simplicity is employed as guidance throughout the development of this genetic algorithm to speed up its running in order to solve large-size problems. The algorithm is illustrated using 500-job problems.

This genetic algorithm uses the natural permutation representation of a chromosome for encoding simplicity.

Manuscript received ___ Revised Dec 8th, 2005. This work was supported, in part, by the Center for Manufacturing Research, Tennessee Technological University.

* Ning Liu is a Ph.D. student at the Department of Electrical and Computer Engineering, Tennessee Technological University (email: ningliu2001@yahoo.com)

** Dr. Abdelrahman is an associate professor at the Department of Electrical and Computer Engineering, Tennessee Tech University (email: mabdelrahman@tntech.edu)

+ Srinivasan Ramaswamy is currently professor and chairperson of the Computer Science Department at University of Arkansas at Little Rock. (e-mail: srini@ieee.org / srini@acm.org)

Heuristic dispatching rules combined with random methods are used to create the initial population for improving (decreasing) the search space, thereby improving searching simplicity. Position-based crossover and order-based mutation operators are used for operator simplicity. The best members of the population during generations are used for searching simplicity, too. The algorithm has been applied to two 7-job problems, two 10-job problems, and one 25-job problem to show optimal solutions obtained. The algorithm has also been applied to solve randomly generated problems with 50, 100, 200, and 500 jobs. Their computational results are compared with those of other scheduling methods to show the improvement from the better of two heuristic dispatching rules, Earliest Due Date (EDD) and Weighted Shortest Processing Time (WSPT), whose definitions are given in Section 3.

The paper is organized as follows. Section 2 presents the definition of the single machine total weighted tardiness problem. Section 3 describes the genetic algorithm for solving this problem. Section 4 reports the computational results of the algorithm. Section 5 summarizes the main conclusions.

2. Problem Definition

The single machine total weighted tardiness problem is defined as follows. Consider n jobs to be processed without interruption on a single machine that can handle only one job at a time. Each job j, available for processing at time zero, has a positive processing time p_j, a positive weight w_j, and a positive due date d_j. For a given sequence of jobs, the tardiness of job j is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of job j. The objective of the total weighted tardiness problem is to find a processing order of all the jobs; this order is a schedule that minimizes the sum of the weighted tardiness $\sum_{j=1}^{n} w_j T_j$ of all jobs. Thus, the problem is to schedule n jobs on a single machine to minimize the sum of the weighted tardiness of all the jobs.

3. Genetic Algorithm

Genetic Algorithms (GAs) were originally proposed by John H. Holland [7]. They are search algorithms that explore a solution space and mimic the biological evolution process. There are many GA implementations successfully applied to a great variety of problems [5]. The main components of a genetic algorithm are as follows [1]:

1. **Solution encoding**: A chromosomal representation of solutions.
2. **Initial population:** Creation of an initial population of chromosomes.
3. **Fitness:** Measurement of chromosome fitness based on the objective function.
4. **Selection:** Natural selection of some chromosomes (parents) in the population for generating new members (children) in the population.
5. **Genetic operators**: Genetic operators applied to these chromosomes whose role is to create new members (children) in the population by crossing the genes of two chromosomes (crossover operators) or by modifying the genes of one chromosome (mutation operators).
6. **Replacement**: Natural selection of the members of the population who will survive.
7. **Parameter selection:** Natural convergence of the whole population that is globally improved at each step of the algorithm.

The performance of a GA depends largely on the design of the above components and the choice of parameters such as population size, probabilities of genetic operators (crossover rate and mutation rate), and number of generations.

3.1. Solution encoding

For the single machine total weighted tardiness problem, the natural permutation representation of a solution is a permutation of the integers 1,…,n, which defines the processing order of n jobs. Each chromosome is represented by such a scheduling solution, i.e., the natural permutation representation of a solution, so as to simplify solution encoding. For example, for a 10-job problem,

A scheduling solution:	2	1	4	3	6	10	8	9	7	5
A chromosome:	2	1	4	3	6	10	8	9	7	5

3.2. Initial population

In order to approximate an optimal solution most accurately, the initial population of chromosomes is created by heuristic dispatching rules, combined with the random method that generates chromosomes randomly. In this way, the search space is decreased - hence guiding towards a quick optimal solution by simplifying the solution space.

The dispatching rules are as follows:

1. EDD: The next job scheduled is a job that has the earliest due date among the jobs that are not scheduled yet. The resulting sequence is the same as the sequence of jobs arranged in the ascending order of their due dates.
2. WSPT: Calculate ratio $S_j = p_j / w_j$. Jobs are scheduled in the ascending order of the ratios.
3. SPT (Shortest Processing Time): Jobs are scheduled in the ascending order of their processing times.
4. BWF (Biggest Weight First): Jobs are scheduled in the descending order of their weights.
5. AU (Apparent Urgency) [16]: Calculate apparent urgency priority AU_j. Jobs are arranged in the descending order of their apparent urgency priorities.

$$AU_j = \left(\frac{w_j}{p_j}\right)\exp\left(\frac{-\max\{0, d_j - t - p_j\}}{k \cdot \overline{p}}\right)$$

Here, k represents the look-ahead parameter and is set according to the tightness of the due date; $\overline{p}$ is the average processing time; t is the current time, but for static problems, here t = 0. This heuristic has the same time complexity as EDD and WSPT.

3.3. Fitness

When a population is generated, each chromosome is evaluated and its fitness is calculated as follows. The total weighted tardiness is computed for each chromosome. Then chromosomes are sorted by the increasing values of their total weighted tardiness. Finally, the first chromosome is assigned its fitness with the value of population size minus one. The fitness of a chromosome is assigned the value of the fitness of its forward adjacent chromosome minus one.

3.4. Selection

By selection methods, chromosomes (parents) are selected from the population for combining to produce new chromosomes (children), for applying genetic operators. Here we use a selection method that selects parents randomly due to its simplicity.

3.5. Genetic operators

1) Crossover: The role of a crossover operator is to combine elements from two parent chromosomes to generate one or more child chromosomes. Here we use position-based crossover. Position-based crossover randomly chooses 0.5*n (n is the number of jobs) positions, and the characters (genes) in these positions are kept unchanged in the offspring. For example, for a 10-job problem, with selected crossover positions 2, 4, 5, 7 and 9:

For crossover

Positions:	0	1	***2***	3	***4***	***5***	6	***7***	8	***9***
Parent 1:	1	2	**3**	4	**5**	**6**	7	**8**	9	***10***
Parent 2:	3	5	***9***	6	***1***	***10***	8	***4***	2	***7***
Child 1:	2	3	9	5	1	10	6	4	8	7
Child 2:	9	1	3	4	5	6	2	8	7	10

2) Mutation: The role of a mutation operator is to provide and maintain diversity in a population so that other operators can continue to work. Here we use order-based mutation. Two positions are selected randomly, and two characters (genes) in these positions are interchanged. For example, with selected positions 2 and 5:

For crossover

Positions:	0	1	***2***	3	4	***5***	6	7	8	9
Parent:	1	2	**3**	4	5	**6**	7	8	9	10
Child:	1	2	6	4	5	3	7	8	9	10

We choose position-based crossover and order-based mutation due to their good performance [25] and simplicities.

3.6. Replacement

This selection is based on elitism [5] due to its simplicity. That is to keep the best chromosomes of the current population and their offspring. These best chromosomes form a new population to survive into the next generation. Our genetic algorithm is summarized by the pseudo-code in Fig. 1

3.7. Parameter selection

For choosing suitable values of parameters such as population size, crossover rate, and mutation rate, we use De Jong's guidelines, as cited by [8], which are still widely followed, namely, to start with:

1. A relatively high crossover probability (0.6-0.7);
2. A relatively low mutation probability (typically set to $1/\lambda$ for chromosomes of length λ);
3. A moderately sized (50-500) population.

Therefore, we apply the position-based crossover operator to N/2 pairs of chromosomes selected randomly, where N is the population size. The mutation probability, which determines the mutation rate, is set to $1/\lambda$, where λ is the length of a chromosome. Population size and generation size are dependent on the problem size.

4. Computational results

Two 7-job problems, two 10-job problems, and one 25-job problem are selected as computational examples to show optimal solutions obtained when applying the genetic algorithm. The genetic algorithm is also applied to solve randomly generated problems with 50, 100, 200, and 500 jobs to show the performance obtained and the efficiency of the proposed algorithm.

4.1. Computational examples

Example 1: Table 1 presents a 7-job example taken from [17] that has an optimal solution 2-1-4-5-3-6-7 with total weighted tardiness 454. The computational results of the genetic algorithm are 4-2-1-5-3-6-7 with total weighted tardiness 454.

Table 1: 7-Job Problem of Example 1

j	*1*	*2*	*3*	*4*	*5*	*6*	*7*
p_j	12	13	14	16	26	31	32
d_j	42	33	51	48	63	88	146
w_j	7	9	5	14	10	11	8

Example 2: Table 2 presents another 7-job example taken from [18]. Our computational results are 1-4-2-3-7-6-5 with total weighted tardiness 4.

Table 2: 7-Job Problem of Example 2

j	1	2	3	4	5	6	7
p_i	6	18	12	10	10	17	16
d_i	1	5	2	4	1	4	2
w_j	8	42	44	24	90	85	68

Example 3: Table 3 presents a 10-job example taken from [22] that has an optimal solution 1-2-3-5-4-6-8-9-7-10 with total weighted tardiness 27. Our computational results are the same.

Table 3: 10-Job Problem of Example 3

j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
p_i	4	1	2	4	1	4	2	2	3	2
d_i	3	4	7	8	11	15	16	20	20	25
w_j	3	1	4	2	3	5	1	5	3	10

Example 4: Table 4 presents another 10-job example taken from [23]. The total weighted tardiness of heuristic dispatching rules EDD, WSPT, SPT, BWF, and AU (k = 2) are 496, 383, 535, 355, and 230 respectively. Our computational results are 3-1-8-4-5-9-7-6-10-2 with total weighted tardiness 218.

Table 4: 10-Job Problem of Example 4

j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
p_i	8	12	6	10	3	11	9	11	13	7
d_i	26	28	32	35	38	48	50	51	53	64
w_j	4	1	6	5	1	4	5	9	8	1

Notation
N is the population size. BESTSEQ is the better-found sequence. BEST is the better-found total weighted tardiness.
T(σ) is a function that calculates the total weighted tardiness of the sequence σ.
RANDSEQ is a function that creates a permutation of 1, 2, …, n randomly.
MBEST(σ) is a function that updates BEST and BESTSEQ if the sequence σ is better.
SORT($\sigma_1, \sigma_2, \ldots, \sigma_N$) is a function that sorts the sequences $\sigma_1, \sigma_2, \ldots, \sigma_N$ in the increasing order of their total weighted tardiness.
INSERT($\sigma_1, \sigma_2, \ldots, \sigma_N$, л) is a function that inserts the sequence л into the sequences $\sigma_1, \sigma_2, \ldots, \sigma_N$ if it is better and deletes the last sequence. Actually, "π" kicks the last sequence out of the line if "π" is better
K is the generation size.
RANDOM(N) is a function that gives an integer between 1 and N randomly.
CROSSOVER(σ_1, σ_2) is a function that generates two new sequences by applying the position-based crossover operator to two sequences σ_1 and σ_2.
M is the mutation size that equals [(1/ λ)*N], where λ is the length of a chromosome.
MUTATION(σ) is a function that generates a new sequence by applying the order-based mutation operator to the sequence σ.

```
// Initialization phase
BESTSEQ := EDD, WSPT, SPT, BWF, or AU;
BEST := T(BESTSEQ);
for i increasing from 1 by 1 until N do $\sigma_i$ := RANDSEQ and MBEST($\sigma_i$);
SORT($\sigma_1, \sigma_2, \ldots, \sigma_N$);
INSERT($\sigma_1, \sigma_2, \ldots, \sigma_N$, EDD|WSPT|SPT|BWF|AU);
// generation phase
for k increasing from 1 by 1 until K do
begin
        // computation of the fitness for the population
        FT($\sigma_1$) := N - 1;
        for i increasing from 2 by 1 until N do FT($\sigma_i$) := FT($\sigma_{i-1}$) – 1;
        // crossover phase
        for i increasing from 1 by 1 until N/2 do
        begin
                // selection of two parents
                repeat
                        $\mu_{i,1}$ := RANDOM(N); $\mu_{i,2}$ := RANDOM(N);
                until $\mu_{i,1} \neq \mu_{i,2}$;
                // application of crossover operator
                ($\pi_{2i-1}, \pi_{2i}$) := CROSSOVER($\sigma_{\mu i,1}$, $\sigma_{\mu i,2}$);
                MBEST($\pi_{2i-1}$); MBEST($\pi_{2i}$);
        end
        // mutation phase
        for i increasing from 1 by 1 until M do
        begin
                // selection of one parent
                h := RANDOM(N);
                $\omega_i$ := MUTATION($\pi_h$); MBEST($\omega_i$);
        end
        // replacement phase
        for i increasing from 1 by 1 until N do INSERT($\sigma_1, \sigma_2, \ldots, \sigma_N, \pi_i$);
        for i increasing from 1 by 1 until M do INSERT($\sigma_1, \sigma_2, \ldots, \sigma_N, \omega_i$);
end;
print BESTSEQ, BEST;
```

Fig. 1. Pseudo-Code of Genetic Algorithm

Example 5: Table 5 presents one 25-job example taken from [9] that has the results 7-17-10-4-6-21-2-12-24-13-1-9-25-3-5-23-8-18-14-15-22-16-19-11-20 with the total weighted tardiness 14,930. Our computation results are 5-9-17-10-4-6-21-2-12-24-13-1-25-7-3-23-8-18-14-15-22-16-19-11-20 with the total weighted tardiness 14,410.

Table 5: 25-Job Problem of Example 5

Job number	Process times	Due dates	Weights
1	71	595	7
2	8	477	7
3	24	768	7
4	35	259	10
5	25	171	2
6	76	321	7
7	92	157	6
8	63	898	3
9	71	201	6
10	56	218	6
11	69	730	1
12	97	484	9
13	25	567	10
14	85	1055	9
15	93	1058	7
16	78	559	2
17	60	197	7
18	21	965	3
19	84	490	2
20	94	334	1
21	58	352	7
22	92	428	4
23	97	686	6
24	26	493	5
25	56	670	4

4.2. General problems

The genetic algorithm is also extensively tested on a set of randomly generated problems with 50, 100, 200 and 500 jobs, namely, general problems, which are created using the methods described in [4, 9, 19-20]. The general problems are generated as follows:

1. Each job j is assigned an integer processing time p_j in the uniform distribution (1, 100), an integer weight w_j in the uniform distribution (1, 10).
2. The difficulty of each problem depends on the range of due dates (RDD) and on the tightness factor of due dates (TF). RDD and TF values are selected from the set {0.2, 0.4, 0.6, 0.8, 1.0} respectively.
3. The range of due dates is defined as

$$\text{RDD} = \frac{d_{\max} - d_{\min}}{C_{\max}}$$

4. A high value of RDD indicates a wide range of due dates, whereas a low value indicates a narrow range of due dates.
5. The tightness factor of due dates is defined as

$$\text{TF} = 1 - \frac{\sum d_j}{nC_{\max}}$$

6. Values of TF close to 1 indicate that the due dates are tight and values close to 0 indicate that the due dates are loose.
7. Each job j is assigned an integer due date d_j in the uniform distribution (P*(1-TF-RDD/2), P*(1-TF+RDD/2)), where $P = \sum_{j=1}^{n} p_j$. When P*(1-TF-RDD/2) is less than or equal to 0, d_j is in the uniform distribution (1, P*(1-TF+RDD/2)).
8. For each value of n, i.e., 50, 100, 200, and 500, five problems have been generated for each of the 25 pairs of values of RDD and TF. Thus a total of 500 problems have been generated.

As these problems have more than 50 jobs, the optimal solutions are not available. Therefore, computational results have been compared with the computational results of descent methods, i.e., DES and DESO, and two heuristic dispatching rules, EDD and WSPT, to show their improvement from the better of these two heuristic dispatching rules.

Descent Methods: These are pairwise interchange methods, including DES and DESO [4, 9, 20]. They begin with the sequence obtained by applying the AU dispatching rule and exchange jobs (u, v) from (1, 2) to (1, 3), (1, 4), …, and (n-1, n), where (u, v) means to exchange the job in the uth position with the job in the vth position.

DES: This is the strict descent method in which only pairwise job exchanges yielding a decrease in objective

Table 6: Computational results for general problems with 50 jobs

RDD	TF	DES %Imp (%)	DESO %Imp (%)	GA %Imp (%)
0.2	0.2	39.36	39.22	49.10
	0.4	12.62	13.60	27.29
	0.6	3.89	3.81	12.25
	0.8	-1.22	-1.24	6.89
	1.0	-1.60	-1.61	1.06
0.4	0.2	19.96	22.93	22.46
	0.4	44.30	43.94	60.23
	0.6	19.41	19.80	34.52
	0.8	0.207	0.207	14.10
	1.0	-1.19	-1.19	4.13
0.6	0.2	-	-	-
	0.4	52.23	52.67	69.64
	0.6	34.80	34.26	56.18
	0.8	12.58	12.59	28.47
	1.0	0.68	0.64	7.99
0.8	0.2	-	-	-
	0.4	46.21	47.77	57.01
	0.6	41.53	41.74	59.09
	0.8	20.64	20.59	42.72
	1.0	3.45	3.47	15.88
1.0	0.2	-	-	-
	0.4	-	-	-
	0.6	52.16	51.54	66.84
	0.8	26.92	27.42	44.78
	1.0	9.41	9.18	26.75

Table 7: Computational results for general problems with 100 jobs

RDD	TF	DES %Imp (%)	DESO %Imp (%)	GA %Imp (%)
0.2	0.2	49.83	46.55	59.73
	0.4	5.54	6.85	26.29
	0.6	0.40	1.05	12.17
	0.8	-2.39	-2.38	6.76
	1.0	-3.71	-3.70	1.06
0.4	0.2	17.64	17.64	17.64
	0.4	45.79	47.48	60.89
	0.6	13.79	13.07	33.20
	0.8	0.31	0.32	12.93
	1.0	-3.97	-4.04	4.21
0.6	0.2	-	-	-
	0.4	49.14	49.70	69.85
	0.6	29.13	30.10	55.39
	0.8	6.12	6.13	21.82
	1.0	-2.91	-2.90	8.30
0.8	0.2	-	-	-
	0.4	57.06	50.03	56.32
	0.6	44.99	45.24	65.30
	0.8	15.29	15.42	35.61
	1.0	1.29	1.28	15.07
1.0	0.2	-	-	-
	0.4	-	-	-
	0.6	56.43	57.22	74.12
	0.8	27.70	27.59	49.10
	1.0	6.28	6.30	22.21
Average		19.70	19.47	33.71

function, here the total weighted tardiness, are accepted.

DESO (Descent Method with Zero Interchanges): In this method, pairwise job exchanges yielding no change in objective function are also accepted, in addition to exchanges yielding a decrease in objective function. Accepting an exchange yielding no change in objective function is because both jobs may be early and remain so even after the exchange.

Percentage Improvement: The percentage improvement is calculated by the following equation [9]:

$$\%\mathrm{Imp} = \frac{\min(T_{EDD}, T_{WSPT}) - T_H}{\min(T_{EDD}, T_{WSPT})} * 100\%$$

where T_{EDD} is the total weighted tardiness obtained after applying EDD rule, T_{WSPT} is the total weighted tardiness obtained after applying WSPT rule, and T_H is the total weighted tardiness obtained after applying a certain heuristic method. Here, the heuristic methods are DES, DESO, and the genetic algorithm. Hence, their computational results are compared with the better of two dispatching rules, EDD and WSPT.

Tables 6-9 present the computational results for the problems with 50, 100, 200, and 500 jobs respectively when applying the GA, which are compared with these of DES and DESO, where "-" means the total weighted tardiness of the problem is zero when applying EDD, WSPT, AU, DES, DESO, or GA respectively. For AU, the look ahead parameter is k = 0.5, 0.9, 2.0, 2.0, and 2.0 for TF = 0.2, 0.4, 0.6, 0.8, and 1.0 respectively. The GA is coded in C language and the computational tests are carried out on a DELL Dimension 8200 PC. Here, for the same problem size, the population and generation sizes of the GA are the same. The analysis of the computational results is as follows:

1. For the same problem size, different problems have different percentage improvements and computation times. The GA has the best improvements, but needs more computation times. For the worst case, the computation times for problems with 50, 100, 200, and 500 jobs are 20 seconds, 45 seconds, 2 minutes 30 seconds, and 12 minutes respectively. The computation times are a reasonable price to pay for the improvement in performance.
2. When the problem size increases, the percentage improvements of descent methods decline. However, the percentage improvements of the GA can be stable or better if we enlarge the population size and/or generation size of the GA. However, there is a trade-off between the improvement in performance and the computation times.

5. Conclusions

In this paper, we propose a new genetic algorithm for solving the single machine total weighted tardiness scheduling problem. Technique simplicity is employed as guidance throughout the development of this genetic algorithm to speed

Table 8: Computational results for general problems with 200 jobs

RDD	TF	DES %Imp (%)	DESO %Imp (%)	GA %Imp (%)
0.2	0.2	38.91	42.61	57.17
	0.4	6.38	6.77	27.31
	0.6	-1.07	-1.04	13.51
	0.8	-3.65	-3.67	5.79
	1.0	-5.55	-5.56	0.98
0.4	0.2	27.54	28.24	30.92
	0.4	33.67	34.85	58.04
	0.6	13.11	13.31	33.63
	0.8	0.50	0.49	13.37
	1.0	-4.12	-4.12	3.56
0.6	0.2	-	-	-
	0.4	51.68	54.92	76.04
	0.6	24.15	24.09	48.48
	0.8	5.10	5.04	22.58
	1.0	-3.23	-3.23	7.38
0.8	0.2	-	-	-
	0.4	55.06	49.87	59.31
	0.6	36.30	36.57	61.42
	0.8	13.63	13.54	33.75
	1.0	-0.43	-0.43	13.88
1.0	0.2	-	-	-
	0.4	-	-	-
	0.6	50.90	51.57	70.64
	0.8	24.17	24.34	48.47
	1.0	3.56	3.55	21.79
Average		17.46	17.70	33.72

up its running in order to solve large-size problems. The genetic algorithm is based on the natural permutation representation of a chromosome for encoding simplicity, the combination of dispatching rules and random method to create the initial population for improving searching space, consequently searching simplicity, position-based crossover and order-based mutation for operator simplicity, and elitism for searching simplicity as well.

Table 9: Computational results for general problems with 500 jobs

RDD	TF	DES %Imp (%)	DESO %Imp (%)	GA %Imp (%)
0.2	0.2	49.16	49.55	58.26
	0.4	4.66	3.92	22.63
	0.6	-1.72	-1.84	11.07
	0.8	-2.94	-3.55	5.90
	1.0	-5.09	-5.09	0.90
0.4	0.2	21.74	14.86	43.44
	0.4	35.89	35.39	60.25
	0.6	11.12	10.77	31.49
	0.8	0.01	0.01	11.52
	1.0	-5.21	-5.21	3.06
0.6	0.2	-	-	-
	0.4	50.10	55.54	76.23
	0.6	19.81	19.73	44.84
	0.8	3.33	3.33	17.77
	1.0	-3.71	-3.71	6.49
0.8	0.2	-	-	-
	0.4	39.84	45.60	68.73
	0.6	25.32	25.33	46.50
	0.8	15.18	11.25	29.95
	1.0	-0.68	-0.68	11.64
1.0	0.2	-	-	-
	0.4	-	-	-
	0.6	38.52	38.43	61.26
	0.8	15.35	15.32	37.04
	1.0	2.11	2.11	18.03
Average		14.89	14.81	31.76

Computational results show that the solutions of the GA are better than those of heuristic dispatching rules and descent methods, but the GA needs more computation times. However, the computation time of the GA is reasonable for good solutions. In addition, the GA is speeded up on solving problems with up to 500 jobs. Therefore the GA can obtain good solutions in an efficient way. Future work will be carried out on the comparison of our computational results with other published GAs' by running the GA on their problems in literature.

REFERENCES

1. Alexandre, F., Cardeira, C., Charpillet, F., Mammeri, Z., and Portmann, M.–C. "Compu-search methodologies II: Scheduling Using Genetic Algorithms and Artificial Neural Networks", in A. Artiba, and S.E. Elmaghraby, eds., *The Planning and Scheduling of Production Systems*, London: Chapman and Hall. pp. 300-335, 1997
2. Akturk, M.S., "A new dominance rule for the total weighted tardiness problem", *Production Planning and Control*, 10(2), pp. 138-149, 1999.
3. Avci S., Akturk, M.S., and Storer, R.H., "A Problem Space Algorithm for Single Machine Weighted Tardiness Problems", *IIE Transactions*, 35, 479-486, 2003
4. Crauwels, H.A.J., Potts, C.N., and Wassenhove, L.N.V., "Local Search Heuristics for the Single Machine Total Weighted Tardiness Scheduling Problem" *INFORMS Journal on Computing*, 10(3), pp, 341-350, 1998.
5. Goldberg, D.E., "*Genetic Algorithms in Search, Optimization and Machine Learning*", Massachusetts: Addison-Wesley. 1989
6. Grosso, A., Croce, F.D, and Tadei, A., "An Enhanced Dynasearch Neighborhood for the Single Machine Total Weighted Tardiness Scheduling Problem", *Operations Research Letters*, 32(1), pp. 68-72, 2004.
7. Holland, J.H., "*Adaptation in Natural and Artificial Systems*", University of Michigan Press, Ann Arbor, 1975
8. Hopgood, A.A., "Genetic Algorithms" In: A.A. Hopgood, *Intelligent Systems for Engineers and Scientists*, CRC Press, Florida, pp. 173-194 2001.
9. Huegler, P.A., and Vasko, F.J., "A Performance Comparison of Heuristics for the Total Weighted Tardiness Problem", *Computers and Industrial Engineering*, 32(4), pp. 753-767, 1997
10. Jouglet, A., Baptiste, P., and Carlier, J., "Exact Procedure for Single Machine Total Cost Scheduling", *IEEE International Conference on Systems, Man and Cybernetics*, Oct 6-9, 2002
11. Kan, A.H.G.R., Lageweg, B.J., and Lenstra, J.K., "Minimizing Total Costs in One Machine Scheduling", *Operations Research*, 23, pp. 908-927, 1975
12. Lenstra, J.K., Kan, A.H.G.K., and Brucker, P., "Complexity of Machine Scheduling Problems", *Annals of Discrete Mathematics*, 1, pp. 343-362, 1977.
13. Madureira, A., Ramos, C., and Silva, S.d.C., "A GA based Scheduling System for Dynamic Single Machine Problem", *Proceedings of the 4th IEEE International Symposium on Assembly and Task Planning,* Soft Research Park, Fukuoka, Japan, pp. 262-267, 2001
14. Maheswaran, R., and Ponnambalam, S.G., "An Investigation on Single Machine Total Weighted Tardiness Scheduling Problems", *Intl. Jour. of Advanced Manufacturing Technology*, 22(3-4), pp. 243-248, 2003
15. Matsuo, H., Suh, C.J., and Sullivan, R.S., "A Controlled Search Simulated Annealing Method for the Single Machine Weighted Tardiness Problem", *Annals of Operations Research*, 21, 85-108, 1989
16. Morton, T.E., Rachamadugu, R.M., and Vepsalainen, A., "Accurate Myopic Heuristics for Tardiness Scheduling", *GSIA Working Paper No. 36-83-84*, Carnegie-Mellon University, PA, 1984
17. Picard, J.C., and Queyranne, M., "The Time-Dependent Traveling Salesman Problem and its Application to the Tardiness Problem in One-Machine Scheduling", *Operations Research*, 26(1), pp. 86-110, 1978.
18. Pinedo, M., "*Scheduling: Theory, Algorithms, and Systems*", Prentice Hall, 2001.
19. Potts, C.N., and Wassenhove, L.N.V., "A Branch and Bound Algorithm for the Total Weighted Tardiness Problems", *Operations Research*, 33(2), pp. 363-377, 1985
20. Potts, C.N., and Wassenhove, L.N.V., "Single Machine Tardiness Sequencing Heuristics", *IIE Transactions*, 23(4), 346-354. 1991
21. Schrage, L., and Baker, K.R., "Dynamic Programming Solution of Sequencing Problem with Precedence Constraints", *Operations Research*, 26, pp. 444-449, 1978
22. Shwimer, J., "On the n-job, One-machine, Sequence-independent Scheduling Problem with Tardiness Penalties: A Branch-Bound Solution", *Management Science*, 18(6), B-301-B-313, 1972.
23. Sipper, D., and Bulfin, Jr., R.L.. "*Production: Planning, Control, and Integration*", p. 399, McGraw-Hill, 1997.
24. Sule, D.R., "Heuristic Method for a Single Machine Scheduling Problem", *Intl. Jour. of Industrial Engineering*, 1(2), pp. 167-174, 1994
25. Syswerda, G., "Schedule Optimization Using Genetic Algorithms", in L. Davis, eds., *Handbook of Genetic Algorithms*, pp. 332-349), Van Nostrand Reinhold., 1991
26. Ying, K.C., and Liao C.J., "An Ant Colony System Approach for Scheduling Problems', *Production Planning and Control*, 14(1), 68-75, 2003.

Ms. Ning Liu is currently a Ph.D. candidate of Electrical and Computer Engineering Department at Tennessee Technological University. She earned her M.S and B.S. degrees from Computer Science and Engineering Department at Zhejiang University, China. Her research interests are in artificial intelligence, planning and scheduling

Dr. Abdelrahman joined the Electrical and Computer Engineering (ECE) Department, Tennessee Technological University (TTU) in 1997. He is currently an associate professor in the ECE department and an associate faculty with the Center for Manufacturing Research, TTU. He earned a Ph.D. in Nuclear Engineering with Emphasis on Reactor Control in 1996 and an MS in Measurement and Control Engineering in 1994 from Idaho State University. He also earned an MS in Engineering Physics and a BS in Electrical Engineering from Cairo University, Egypt in 1992 and 1988, respectively. His main research interests include intelligent systems, measurement systems, mechatronics and integration of sensing and control with special focus on industrial applications. Dr. Abdelrahman has been the principal investigator on several major research projects aiming at improving the operational efficiency of the metal casting industry through intelligent sensing and control.

Dr. Ramaswamy is currently the chairperson of Department of Computer Science at University of Arkansas at Little Rock. Earlier he was the Chairperson of the Computer Science Department at Tennessee Tech University. His research interests are on intelligent and flexible control systems, behavior modeling, analysis and simulation, software stability and scalability; particularly in the design and development of better software systems for real-time control issues in manufacturing and other distributed, real-time applications. He has actively consulted on the design, development and implementation of a knowledge capture, classification and mining project with eFutureKnowledge, Inc. and on a NIST sponsored ATP project with InRAD, LLC. During the summers of 2003 and 2004, he was a visiting research professor of Computer Science in the Institute of Software Integrated Systems (ISIS) at Vanderbilt University as part of a NSF ITR project - Foundations of Hybrid and Embedded Software Systems. In 1994-1995, and subsequently during the summer months of 1996 and 1998, he was a post-doctoral research fellow / visiting scientist in the Laboratory for Intelligent Processes and Systems (LIPS) at the University of Texas at Austin where he helped with research efforts on Sensible Agents. Dr. Ramaswamy earned his Ph.D. degree in Computer Science in 1994 from the Center for Advanced Computer Studies (CACS) at the University of Southwestern Louisiana, Lafayette, LA, USA (now University of Louisiana at Lafayette). He is member of the ACM, Society for Computer Simulation International, Computing Professionals for Social Responsibility and a senior member of the IEEE.