Hemmak Allaoua* and Bouderah Brahim

New Properties for Solving the Single-Machine Scheduling Problem with Early/Tardy Jobs

DOI 10.1515/jisys-2016-0063
Received May 23, 2016; previously published online July 12, 2016.

Abstract: This paper presents a mathematically enhanced genetic algorithm (MEGA) using the mathematical properties of the single-machine scheduling of multiple jobs with a common due date. The objective of the problem is to minimize the sum of earliness and tardiness penalty costs in order to encourage the completion time of each job as close as possible to the common due date. The importance of the problem is derived from its NP-hardness and its ideal modeling of just-in-time concept. This philosophy becomes very significant in modern manufacturing and service systems, where policy makers emphasize that a job should be completed as close as possible to its due date. That is to avoid inventory costs and loss of customer's goodwill. Five mathematical properties are identified and integrated into a genetic algorithm search process to avoid premature convergence, reduce computational effort, and produce high-quality solutions. The computational results demonstrate the significant impact of the introduced properties on the efficiency and effectiveness of MEGA and its competitiveness to state-of-the-art approaches.

Keywords: Genetic algorithm, single machine scheduling, early/tardy jobs, common due date, mathematical properties.

1 Introduction

The spectacular development in computational power in recent years demonstrated that using the generic search concept alone of any meta-heuristic (such as genetic algorithm, GA) for solving NP-hard combinatorial optimization problems is insufficient to obtain good results due to the generic nature of such methods. This observation led many researchers to embed domain-specific properties within the generic search concept to clearly improve GA effectiveness, avoid its premature convergence, and reduce its required processing time. The topic of this work is tracking single-machine early/tardy scheduling against a common due date with a variant of GA in which two new properties of this problem will be presented, proofed, then integrated.

Thus, a new variant of GA is proposed to solve the single-machine scheduling problem of independent jobs where the objective is to minimize the sum of earliness and tardiness penalties of jobs having a common due date (ETCDD). The theoretical importance of the single-machine ETCDD problem is due to its complexity as an NP-hard problem [4, 7, 20, 22] and due to its practical importance in just-in-time (JIT) modern production and service processing. In JIT, policy makers emphasize that a job should be completed as close as possible to its common due date to avoid inventory cost and loss of customer's goodwill. This concept is actually adopted by many economic and industrial companies around the world. The goal is to realize better results than existing works in the field and to reduce time processing that is required to compute results.

***Corresponding author: Hemmak Allaoua,** Computer Science Department, University of Bejaia, 06000 Bejaia, Algeria, e-mail: hem_all@yahoo.fr
Bouderah Brahim: Computer Science Department, University of M'sila, M'sila, Algeria

In scheduling on a single machine against a common due date, there are two classes of common due date problems that have proven to be NP-hard, namely the restrictive and non-restrictive common due date problem. As one job at most can be completed exactly at the due date, some of the jobs might be completed earlier than the common due date (d), while other jobs finish late. In this work, the treated problem is such as the restrictive case of the problem where the common due date is less than the sum of the processing times of all the jobs and each job possesses different earliness/tardiness penalties.

This problem has proven to be the most difficult problem in this area of research.

Reviewing the single-machine ETCDD literature, it was found that three different properties were embedded into various constructive heuristics to obtain good approximation results [4, 7, 8, 10, 14, 20]. However, to capitalize on the potential of such properties, two new additional propensities are proposed with supporting proofs. The set of all mathematical properties are integrated inside a GA search process to derive what is called a mathematically enhanced GA (MEGA) procedure. The quality of MEGA is demonstrated on a set of 200 benchmark instances available in Refs. [7, 8] and compared to the state-of-the-art algorithms for the problem in Refs. [7, 12, 11, 13, 20].

The remaining parts of the paper are organized as follows: Sections 2 to 4 provide background on the problem, including a statement, literature review, and mathematical properties. In Section 5, the implementation details of our proposed MEGA approach are presented. Computational results and discussions are provided in Section 6. Finally, we conclude with a summary of findings and perspectives on future work in Section 7.

2 Literature Review

In recent years, the single-machine ETD problem has received a lot of attention from researchers and practitioners. Its importance arises from its NP-hard complexity and its practical importance within the JIT context as in intelligent scheduling [21]. The large earliness and tardiness times of jobs are highly discouraged in order to minimize the inventory storage and late delivery costs to meet the JIT principle, which calls for the right amount of goods/services to be produced and delivered at exactly the right time and the right quantity.

A literature search on the single-machine ETD problem was conducted, and the following studies were found. They are arranged from the earliest to the latest as follows: Biskup [5], Gordon et al. [9], Feldmann and Biskup [8], Hino et al. [15], Lin et al. [19], Liao and Cheng [18], Nearchou [20], Ronconi and Kawamura [22], Alvarez-Valdes et al. [1], Lassig et al. [17], as well as Ying [25], Baker and Scudder [4], Vallée and Yiltizogli [24], and Cha et al. [6]. Lässig et al. has presented a study on the general case of the common due date scheduling problem. A linear algorithm for a given gob sequence was presented to optimize a given job sequence with a run-time complexity of $O(n)$, where n is the number of jobs [17].

Jafarnejad et al. elaborated a study to develop a goal function for optimizing single-machine scheduling problems, that combines both earliness and tardiness penalties. This study is also determined to minimize the earliness and tardiness penalties in this area [16]. Baker and Scudder developed a branch and bound algorithm to find optimal solutions to an ETCDD problem and reported the results of computational experiments. They also tested some heuristic procedures and found that surprisingly good performance can be achieved by a list schedule followed by an adjacent pairwise interchange procedure [3]. Shahriari et al. investigated a JIT single-machine scheduling problem with a periodic preventive maintenance. Also to maintain the quality of the products, there is a limitation on the maximum number of allowable jobs in each period. The proposed bi-objective mixed-integer model minimizes total earliness/tardiness and make span simultaneously [23].

Awasthi et al. considered the problem of scheduling jobs on single and parallel machines where all the jobs possess different processing times but a common due date. There was a penalty involved with each job if it is processed earlier or later than the due date. The objective of the problem was to find the assignment of jobs to machines, the processing sequence of jobs, and the time at which they are processed, which minimizes the total penalty incurred due to tardiness or earliness of the jobs. An exact polynomial algorithm was presented for optimizing a given job sequence for single and parallel machines with the run-time complexities of $O(n \log n)$ and $O(mn^2 \log n)$, respectively, where n is the number of jobs and m the number of machines [2].

3 Background of the Problem

The single-machine scheduling problem with early/tardy jobs around a common due date involves a set of jobs; each job has its own processing time requirements. All jobs must be processed on a single machine, and a penalty cost is incurred when a job is completed before (earliness) or after (tardiness) the common due date. The objective is to minimize the summation of earliness and tardiness penalty costs in order to encourage the completion time of each job to be as close as possible to the common due date. The following notations and statements are proposed to understand better the properties of the problem. The following notations define the problem statement:

- n: integer number of jobs to be scheduled;
- I: set of n jobs: $I = \{1, 2, \ldots, n\}$;
- d: common due date of all the n jobs;
- C_i: complete time of job i;
- p_i: processing time of job i;
- $E_i = \max\{d - C_i, 0\}$ (earliness of job i);
- $T_i = \max\{C_i - d, 0\}$ (tardiness of job i);
- α_i: penalty per unit time of earliness for job i;
- β_i: penalty per unit time of tardiness for job i;
- h: parameter of common due date tightness used as follows: $d = h * T$, where $T = \sum_{k=1}^{n} p_i$; $h \in \{0.2, 0.4, 0.6, 0.8\}$;
- B: the set of jobs to complete at or before the common due date d;
- A: the set of jobs to complete after d;

$$|A|+|B| = \begin{cases} n-1 \text{ if there are a straddled job} \\ n \text{ else} \end{cases};$$

A straddled job is a job that starts before d and completes after d. If, in a given sequence, there is a job that completes exactly at d, there will be no straddled job.

1. Each job has to be processed on the single machine without interruption.
2. Each job is available at time 0.
3. Each job must be processed just once.
4. For each job i, the processing time p_i, the cost per unit time of earliness α_i, and the cost per unit time of tardiness β_i are given and assumed integers. The objective function to be minimized can be expressed as the sum of weighted penalties of earliness and tardiness as follows: $\sum_{k=1}^{n} (\alpha_i E_i + \beta_i T_i)$. Any permutation of n jobs will be a feasible solution to the problem, and there are exponential numbers of such permutations ($O(n!)$). The optimal sequence is the permutation that has the minimum objective value among all permutations.

The optimal solution for the single-machine ETD problem satisfies the following three optimality properties:

Property 1: It does not contain any idle time between any consecutive jobs.

Property 2: It is V-shaped around the common due date: the jobs completing before or on the common due date are sorted in decreasing order of the ratios p_i/α_i, and the jobs starting on or after the common due date are sorted in increasing order of the ratios p_i/β_i.

Property 3: In an optimal schedule, either the first job starts at time zero or the completion time of one job coincides with the common due date.

The proofs of these properties are established using proof by contradiction in Baker and Scudder [3], Jafarnejad et al. [16], Gordon et al. [9], Feldmann and Biskup [8], Lin et al. [24], Biskup [23], and Hall and Posner [3].

4 New Mathematical Properties

The most difficult part of the ETCDD problem usually focuses on how to find the beginning time of optimal sequence, because that will multiply the time complexity of the problem by $d-1$ (where d is the common due date for all jobs). That means that the beginning time t_0 belongs to $[0, d-1]$.

The three problem properties presented above describe just the optimal sequence without specifying any details about the beginning time of the optimal sequence. That is happening often when $h=0.6$ or $h=0.8$ (in these cases, the common due date d is equal to 0.6∗sum_pi or 0.8∗sum_pi, so it is quite possible that scheduling starts at $t_0>0$). Therefore, in this work, two new properties were developed and then proofed for determining the beginning time t_0 of the optimal sequence. On one hand, that will considerably improve the solution quality, and, on the other hand, that will reduce clearly the time computing of the algorithm. These properties are called 4 and 5, respectively.

Property 4: There is an optimal schedule in which the arithmetic mean $\left[m=\frac{\left(\sum_{i\in B}\alpha_i C_i+\sum_{i\in A}\beta_i C_i\right)}{\left(\sum_{i\in B}\alpha_i+\sum_{i\in A}\beta_i\right)}\right]$ is equal to the common due date d.

where

- $\alpha_i, \beta_i, C_i, A$, and B are such as defined in the problem statement above;
- α_i: penalty per unit time of earliness for job i;
- β_i: penalty per unit time of tardiness for job i;
- C_i: complete time of job i (here, the completion times C_is are computed with starting time $t_0=0$ for any given sequence);
- B: the set of jobs to complete at or before the common due date d;
- A: the set of jobs to complete after d;
- m: the arithmetic mean of (C_i, μ_i);
- μ_i is defined such as $\mu_i=\alpha_i$ if $C_i\le d$ and $\mu_i=\beta_i$ if $C_i>d$, $\forall i\in I$;
- d: the common due date for all jobs.

Proof. The arithmetic mean is average of values (C_i, μ_i) is defined by

$$\left[m=\frac{\left(\sum_{i\in B}\alpha_i C_i+\sum_{i\in A}\beta_i C_i\right)}{\left(\sum_{i\in B}\alpha_i+\sum_{i\in A}\beta_i\right)}\right].$$

Then, the mean difference of the (C_i, μ_i) values is expressed by

$$\Delta=\frac{\left(\sum_{i\in B}\alpha_i|m-C_i|+\sum_{i\in A}\beta_i|C_i-m|\right)}{\left(\sum_{i\in B}\alpha_i+\sum_{i\in A}\beta_i\right)}.$$

Let consider the function G_p as follows:

$$G_p(x)=\frac{\left(\sum_{i\in B}\alpha_i|p-C_i|+\sum_{i\in A}\beta_i|C_i-p|\right)}{\left(\sum_{i\in B}\alpha_i+\sum_{i\in A}\beta_i\right)}.$$

From the arithmetic mean properties [the above ratio $G_p(x)$ reaches its minimum when $p=m$], we can deduce the relationship in Eq. (1) that

$$\text{Min}\{G_p(x)\}=G_m(x)=D. \tag{1}$$

Let consider the function F as follows:

$F(x)=\sum_{k=1}^{n}(\alpha_i E_i+\beta_i T_i)$, where x is a permutation sequence of jobs in set I.

As $E_i=d-C_i$ for $C_i \le d$ and $T_i=C_i-d$ for $C_i>d$, then $E_i=|d-C_i|$ and $T_i=|C_i-d|$; thus, replacing them in $F(x)$, we get

$F(x)=\left(\sum_{i\in B}\alpha_i\left|d-C_i\right|+\sum_{i\in A}\beta_i\left|C_i-d\right|\right)$, and consequently,

$$\frac{F(x)}{\left(\sum_{i\in B}\alpha_i+\sum_{i\in A}\beta_i\right)}=\frac{\left(\sum_{i\in B}\alpha_i\left|d-C_i\right|+\sum_{i\in A}\beta_i\left|C_i-d\right|\right)}{\left(\sum_{i\in B}\alpha_i+\sum_{i\in A}\beta_i\right)}=G_d(x). \tag{2}$$

As $\left(\sum_{i\in B}\alpha_i+\sum_{i\in A}\beta_i\right)$ is a constant value, we conclude from Eqs. (1) and (2) that $F(x)$ becomes minimal when $d=m$. It should be noted that when the above ratio reaches a minimum value, $F(x)$ also reaches its minimum value, because $F(x)$ and $G_d(x)$ are proportional and the denominator $\left(\sum_{i\in B}\alpha_i+\sum_{i\in A}\beta_i\right)$ is a constant value. □

Corollary:

1. If $d \le m$, then the optimal schedule may start at time $t_0=0$.
2. If $d>m$, then the optimal schedule may start at time $t_0=d-m$.

Consequence of using property 4: The main difficulty of the early/tardy scheduling problem is always to determine the starting time of jobs. Therefore, in this paper, an approach is proposed to solve this difficulty based on the above corollary as follows:

1. Compute the value of the above average m. If $d \le m$, just the sequences that start at time 0 may be sought; else, if $d>m$, only the sequences that start at $d-m$ will be examined.
2. Then, property 3 is applied, i.e. either of the jobs may complete at time d.

Therefore, it must choose the starting time of the schedule to be as close as possible to $d-m$. Note that m can be moved but never d, as d is constant and m is depending on C_i. That means that when $m<d$, the starting time t_0 can be delayed to obtain $m=d$, because the arithmetic mean is a linear function [i.e. mean (x_i+c) = mean $(x_i)+c$]. On the other hand, when $m \ge d$, m cannot coincide with d because of the time limit (0), as it is shown in Figure 1. It should be noted that m is computed with the starting time $t_0=0$; if $m<d$, the starting time can be taken as $t_0=d-m$ to have m coinciding with d (i.e. m is moved to d).

However, if $d \le m$, the stating time t_0 can be chosen to make m coincide with d; the best sequences may have a starting time $t_0=0$ (Figure 1).

Property 5: If $m<d$, then there is an optimal schedule in which $\sum_{i\in B}\alpha_i=\sum_{i\in A}\beta_i$.

Where

- d, α_i, β_i, A, B are such as defined in the problem statement above;
- m: the arithmetic mean of (C_i, μ_i);
- μ_i is defined such as $\mu_i=\alpha_i$ if $C_i \le d$ and $\mu_i=\beta_i$ if $C_i>d$, $\forall i \in I$.

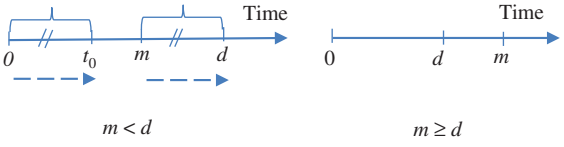


Figure 1: Possible Positions of d against m.

Proof. Let $F(x)=\sum_{k=1}^{n}(\alpha_i E_i+\beta_i T_i)$. As, $E_i=d-C_i$ for $C_i \le d$ and $T_i=C_i-d$ for $C_i>d$, replacing the terms E_i and T_i in $F(x)$, we would get $F(x)=d*\left(\sum_{i\in B}\alpha_i-\sum_{i\in A}\beta_i\right)+\left(\sum_{i\in A}\beta_i C_i-\sum_{i\in B}\alpha_i C_i\right)$.

Let $F'(x)=\sum_{i\in A}\beta_i C_i-\sum_{i\in B}\alpha_i C_i$; if $m \le d$, we can easily find two sets A and B such as $\sum_{i\in B}\alpha_i \ge \sum_{i\in A}\beta_i$. This can be done by setting initially $A=\Phi$ and $B=I$, then muting iteratively jobs from A to B until reaching $\sum_{i\in B}\alpha_i \ge \sum_{i\in A}\beta_i$. As consequence, $F(x)$ becomes the sum of two positive integers:

$d*\left(\sum_{i\in B}\alpha_i-\sum_{i\in A}\beta_i\right)$ and $\left(\sum_{i\in A}\beta_i C_i-\sum_{i\in B}\alpha_i C_i\right)$ (because $F(x)$ and $\sum_{i\in B}\alpha_i-\sum_{i\in A}\beta_i$) are positive.

As $F(x)\ge 0$ and $\sum_{i\in B}\alpha_i \ge \sum_{i\in A}\beta_i$, then $F'(x)\ge 0$; then $F(x)$ becomes minimal when $\sum_{i\in B}\alpha_i=\sum_{i\in A}\beta_i$. □

Corollary: To reduce the size of the space of solutions, it may just seek solutions having $\sum_{i\in B}\alpha_i \approx \sum_{i\in A}\beta_i$.

Consequence of using property 5: This type of problem becomes more difficult when d is large (i.e. often especially when d is given greatest). Therefore, this property is useful to find the starting time according the following algorithm (Algorithm 1):

Algorithm 1: Computing starting time using property 5.

1. For each sequence of jobs, $x=(1, 2, \ldots, n)$
2. $C_i=d$; $i=1$; sum_$\alpha_i=0$; sum_$\beta_i=\sum_{i=1}^{n}\beta_i$;
3. While (sum_α_i < sum_β_i and $C_i \ge 0$)
4. $C_i=C_i-p_i$;
5. sum_α_i = sum_$\alpha_i+\alpha_i$;
6. sum_β_i = sum_$\beta_i-\beta_i$;
7. $i=i+1$;
8. end while
9. starting_time = C_i.

This algorithm starts by making all jobs after d, then the jobs are moved before d one by one until realizing $\sum_{i\in B}\alpha_i \approx \sum_{i\in A}\beta_i$.

That will allow to choose the starting time as close as possible to the value C_i, which realizes $\sum_{i\in B}\alpha_i=\sum_{i\in A}\beta_i$.

Note that the two properties are more used in the most difficulty cases where $h=0.6$ and $h=0.8$, i.e. when there is a large margin to choose the starting time.

Example

Size $n=10$ jobs; instance $k=1$ (source: Biskup and Feldman benchmarks) [7].

Job i	1	2	3	4	5	6	7	8	9	10
p_i	20	6	13	13	12	12	12	3	12	13
α_i	4	1	5	2	7	9	5	6	6	10
β_i	5	15	13	13	6	8	15	1	8	1

Sum $p_i=T=\sum_{k=1}^{n}p_i=116$; $d=T*h$; $h\in\{0.2, 0.4, 0.6, 0.8\}$.

Results

h	d	t_0	Opt.	Optimal sequence
0.2	23	0	1936	4 2 7 3 9 6 5 8 1 10
Time =				0 d (straddled job 7)
0.4	46	0	1025	4 2 3 7 9 6 5 8 1 10
Time =				0 d (straddled job 9)
0.6	69	1	841	4 2 3 7 9 6 5 8 1 10
Time =				1 d (no straddled job)
0.8	92	16	818	4 2 1 3 7 6 9 5 8 10
Time =				16 d (no straddled job)

Straddled job means a job starts just before d and completes after d.
It is easy to verify that

1. There is no idle time in the optimal schedule. That is because $C_{i+1} = C_i + p_i$ (property 1).
2. Jobs in B are sorted by decreasing order of ratio p_i/α_i and jobs in set A are sorted by increasing order of ratio p_i/β_i (property 2) (V-shaped property), with B set of jobs completing at or before d and A set of jobs completing after d.
3. In an optimal job, there is a job that starts at 0 or a job completes at d (property 3).
4. If $d \leq$ mean, then the optimal time must start at 0 (as in $h = 0.2$ and $h = 0.4$ above). Else, the optimal schedule may start at $t_0 = d -$ mean (as in $h = 0.6$ and $h = 0.8$ above). Note that it is not always $t_0 = 0$ for $h = 0.2$, 0.4 and $t_0 > 0$ for $h = 0.6$, 0.8. However, it is depending on the instance data (property 4).
5. In each optimal sequence above, we have $\sum_{i \in B} \alpha_i \approx \sum_{i \in A} \beta_i$ (property 5).

5 MEGA

In this section, a variant implementation of GA will be presented that is used to solve the ETCDD problem. The five properties described above are integrated, as will be clarified later. However, implementing GA needs firstly to choose

- A good individual encoding process;
- A suitable fitness function for evaluating individuals;
- An adequate selection policy;
- Crossover and mutation probabilities;
- Empirically stopping criterion.

The GA algorithm process starts by generating a randomly initial population, then genetic operators (selection, crossover, mutation) [23, 24] are applied with certain probabilities to produce a new generation of offsprings. These offsprings are supposed to inherit the characteristics of their parents and evolve into better individuals. At each generation, the best individuals replace the weakest fit individuals. The GA process is iteratively repeated for a number of generations, and the best-found individual is declared as the final obtained solution. The GA process is generally described as follows (Algorithm 2):

Algorithm 2: GA for ETCDD.

```
1. Begin
2.   Population and parameters' initializations;
3.   Fitness evaluation;
4.   Repeat
5.     Pair selection of individuals;
6.     Crossover;
7.     Mutation;
8.     Fitness evaluation;
9.   Until (stopping criterion);
10. End.
```

However, the design of the GA algorithm depends on the designer's choices for the GA components to solve the problem at hand. Such components include individual's encoding, definition of fitness function for individuals' evaluation, selection of pairs of individuals, probability crossover, probability mutation, and criteria for stopping. The components' choices must be carefully defined and their parameter values must be carefully chosen, as they both considerably determine the solution quality of the proposed GA algorithm. Therefore, such components will be individually explained next.

5.1 Individual's Encoding

Traditionally, characteristics (also called genetic phenotypes) of an individual solution are represented using binary strings of 0s and 1s; however, other encodings are also possible. The whole abstract representation of a solution is often called chromosome (or genotypes of individuals). In our implementation, a chromosome is a feasible solution that consists of an array of $(n+1)$ integer numbers; the first entry represents the starting time to begin the processing of the sequence of jobs, and the other remaining n entries represent the sequence of jobs. Example: the following vector encodes a sequence of jobs (2, 6, 1, 4, 5, 3) that begins at time 4:

4	2	6	1	4	5	3

A population of m individuals will be represented by a two-dimensional matrix, $m*(n+1)$.

5.2 Individual's Fitness Function

The fitness function is defined over the genetic representation of a solution. It measures the quality of the generated solution in each chromosome. The fitness function is always problem dependent. An ideal fitness function should correlate closely with the algorithm's goal, and should be quickly computed. However, it should not favor solutions to avoid converging quickly toward a local solution. Many processes are possible to adjust the fitness function to prevent premature convergence or to diversify the population. The linearization and exponentiation function defines our fitness function:

$$\text{Fitness}(x)=h*(1-\text{obj}(x)/\text{sum\_obj})\text{^}0.1+1.$$

where h is the common due date rate, x is a chromosome, and obj(x) is the objective value solution/chromosome.

Note that obj(x) is inversely proportional to fitness(x), that is because obj(x) is to be minimized and fitness(x) is to be maximized.

The used factor h, the exponent 0.1, and the term +1 are used here to realize a compromise between the intensification and diversification processes generally used in meta-heuristics. Thus, this rule will provide different values and allow a chance to weak individuals. In short, this rule was tested and used in many related works and had justified its efficiency [12, 14].

5.3 Population's Initialization

The initial population is initialized using randomly generated individuals. Traditionally, a uniform law of probability must be used to cover all space of solutions (space search). The aim is to prevent premature convergence of the algorithm to local solutions. The size of the population depends on the nature of the problem. Occasionally, the population may be "seeded" with solutions in areas where optimal solutions are likely to be found. The population size is chosen sufficiency great to represent the evolutionary phenomenon as much as possible, and it is almost proportional to the problem size n (number of jobs). For reporting our computational results, the population size, Population_size = $50 * n$, is chosen.

5.4 Population's Evaluation

The evaluation of population at every generation computes the objective for each individual, then its fitness function. It also determines the minimum population value (best among all objective values) and saves the

corresponding solution. At the last generation, this step allows to determine the best individual in the population and to declare it as the best obtained solution during the whole search process.

5.5 Selection of Individuals

The selection process determines which individuals are to participate in the production of new individuals (off-springs) to include in the next generation. Individual solutions are selected based on a fitness-based process, where solutions with highest fitness values are more likely to be selected for reproduction. The selection process evaluates the fitness of each solution in the whole population (or a random sample of solutions in the population when time consuming) to select the best solutions for reproduction. In our implementation, a fortune wheel method is used to select individuals among those having great fitness. The random roulette function uses the following implementation (Algorithm 3):

Algorithm 3: Roulette implementation

```
1. Int Function roulette();
2.    Int i;
3.    Float r, s1;
4.    r = Rnd() * sum_fitness;
5.    s = 0;
6.    i = 0;
7.    While s < r
8.       i = i + 1;
9.       s = s + fitness(i);
10.   End While
11.   Return I;
12. End Function.
```

This random roulette process is applied twice whenever a pair of parents is needed for breeding.

5.6 Crossover Operator

The crossover operator allows the production of new off-springs (child solutions) from a couple of parent solutions. For each new couple of off-springs to be produced, a couple of solutions is selected for breeding from the population pool and the crossover operator is executed as shown below. Given two selected parents (parent 1 and parent 2), a two-point crossover is implemented. It randomly selects two positions, and exchanges the middle parts between parents to create two new children. In the proposed implementation, the first point is chosen before the common due date d, and the second point is chosen after d for better efficiency. In addition, the probability rate is set to call the crossover operator at $p_c = 0.75$, for reporting our experimental results and best solutions.

Parent 1	4	2	6	1	4	5	3
Parent 2	0	1	4	2	3	6	5

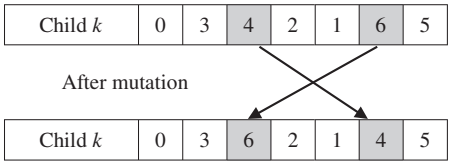
Crossover ↓

Child 1	4	6	4	2	3	5	1
Child 2	0	2	6	1	4	3	5

When the crossover operation generates a gene called "lethal gene" composed of repeated job, it is certain that another job is missing from the sequence. In such case, a repair mechanism is initiated to include the missing one and replaces the lethal job.

5.7 Mutation

The mutation genetic operator is normally introduced to produce small modifications to the obtained child by the crossover operator. The aim is to introduce into the population new characters that do not exist among the parents to assure the diversity of search into new solutions space, hopefully of better quality. In the proposed implementation, the mutation probability rate at which two random jobs will switch positions is set to $p_m = 0.01$ as follows:



It should be noted that the switched jobs are executed randomly in a similar way to the crossover operator; the first position is selected before the common due date d, and the second position is after d. Otherwise, the mutation operator will not have any effect on the solution because of property 2.

5.8 Mathematical Property Usage

The mathematical properties are used to evaluate the sequence at each generation whenever a new sequence is generated, for which the mean (m) is computed as indicated in Section 2.4 (property 4) to determine the starting time of the schedule. Then, the jobs in the sequence are sorted as described in property 2. Property 1 is always applied in the evaluation phase, which means there is no idle time between two jobs. Properties 3 and 5 are often used when there are opportunities to start the sequence at time >0 (i.e. the cases when d is large at $h = 0.6$ or $h = 0.8$). These mathematical properties are applied on each sequence at the following moments:

- After generating each sequence at the initialization phase;
- After generating each child at the crossover phase;
- After muting the individual at the mutation phase.

5.9 Termination Criterion

In the literature, many forms of termination criteria to stop an algorithm can be used, including execution of a fixed number of iterations; a prefixed computational effort is reached; the best solution is not improved after a given number of iterations; and the optimal solution is found (when known). For the current implementation, a fixed number of generations is invoked, with values proportional to the size of problem n. The computational results are reported when the number of iterations is reached, (number_of_iterations = n^2).

Note that the GA parameters (population size, crossover probability) were defined by Goldeberg, but in broad ranges for more flexibility of the algorithm when applied to a problem. These intervals are of course inspired by the natural phenomenon of the meta-heuristic. In addition, the values chosen here were tested and used in related works and had justified their efficiency [12, 14].

6 Computational Results

The set of benchmarks that was developed by Biskup and Feldman [7, 8] was employed to demonstrate the efficiency and effectiveness of the proposed MEGA approach. The benchmark set consisted of instances with variable sizes, $n = 10, 20, 50, 100, 200$ jobs. The produced results (MEGA) were compared to those obtained

by Nearchou [20]. It was found that Nearchou's results are the best results in the literature in most cases, and they were selected for comparison with the proposed algorithm. For each size of $n = 10$ to $n = 200$, there were 10 different instances with different values of rate $h = 0.2, 0.4, 0.6, 0.8$ to determine the common due date (i.e. there were 80 benchmarks for each size).

Table 1 summarizes the comparative results, starting from the problem's characteristics in column one followed by our CPU time, associated MEGA average solutions over the 10 instances per size, Nearchou's average solutions, the average of the differences per instance, and to the comments on the number of the new best solutions obtained by the proposed approach (B), the number of equal solutions (E), and the number of our worst solutions. The results were obtained on a PC with Intel Core 2 Duo 2.4 GHz CPU and 4 G RAM.

The summary results in Table 1 show clearly that for smaller-sized instances of $n = 10$ and $n = 20$, both our MEGA approach and that of Nearchou obtained the optimal solutions for such instances, which were validated using our approach. For the largest-sized instances, Nearchou's approach becomes ineffective for tight due dates with $h = 0.6$ and $h = 0.8$. For $n = 50$, $n = 100$, and $n = 200$, these results show 4 new best solutions, 18 new best solutions, versus 12 worse solutions, and 30 new best solutions versus 9 worse solutions, respectively.

In order to show the gap, the results are reported in Figure 2.

Table 2 summarizes the percentages of the obtained results compared to Nearchou's results. These results are then reported in Figure 3 to show the efficiency of our approach.

These results allow noting that

- For $n \leq 20$, both MEGA and Nearchou's approach give optimal solutions.
- In both approaches, the objective is inversely proportional to h. Indeed, when h increases, d increases, and we will have more opportunities to choose the starting time t_0.
- For $n = 50$, all results are better than those of Nearchou. As our algorithms, EDGA provides the same results as those of Nearchou except for the case $n = 50$, $h = 0.2$, $k = 3$, for which MEGA is better.
- From $n \geq 100$, Nearchou's approach is more efficient than that of Feldmann when $h = 0.2$ and $h = 0.4$, and the opposite when $h = 0.6$ and $h = 0.8$. However, the algorithm is EDGA in a uniform manner.

Table 1: MEGA Results Compared with Nearchou's Average of Best Results.

Size, Instance	CPU Time	MEGA	Nearchou	Gap	Comments
$n = 10$, $h = 0.2$	459.5 ms	1674.4	1674.4	0	
$n = 10$, $h = 0.4$	462.6 ms	973.1	973.1	0	
$n = 10$, $h = 0.6$	475.7 ms	734.3	734.3	0	
$n = 10$, $h = 0.8$	483.1 ms	715.3	715.3	0	All optimal
$n = 20$, $h = 0.2$	1.592 s	6178.3	6178.3	0	
$n = 20$, $h = 0.4$	1.555 s	3635	3635	0	
$n = 20$, $h = 0.6$	1.582 s	2811.4	2811.4	0	
$n = 20$, $h = 0.8$	1.606 s	2724.8	2724.8	0	All optimal
$n = 50$, $h = 0.2$	8.821 s	35,492.7	35,496.1	−3.4	(2B)
$n = 50$, $h = 0.4$	8.474 s	20,432.5	20,432.8	−0.3	(1B)
$n = 50$, $h = 0.6$	8.872 s	15,896.5	15,898.9	−2.4	(1B)
$n = 50$, $h = 0.8$	8.969 s	15,847.2	15,847.2	0	
$n = 100$, $h = 0.2$	1.325 min	132,417.4	132,435.3	−17.9	(9B, 1E)
$n = 100$, $h = 0.4$	1.301 min	78,112.4	78121	−8.6	(4B, 2E, 4W)
$n = 100$, $h = 0.6$	1.409 min	64,935.1	64,933	2.1	(2B, 2E, 6W)
$n = 100$, $h = 0.8$	1.481 min	64,904	64,906.5	−2.5	(3B, 6E, 1W)
$n = 200$, $h = 0.2$	4.302 min	512,282.9	512,319.7	−36.8	(3B, 1E, 6W)
$n = 200$, $h = 0.4$	4.335 min	303,374.6	303,503.1	−128.5	(7B, 3W)
$n = 200$, $h = 0.6$	4.528 min	256,356.3	257,208.7	−852.4	(10B)
$n = 200$, $h = 0.8$	4.445 min	256,255.4	257,152	−896.6	(10B)

B, MEGA finds a new best solution; E, equal solution values; W, MEGA is worse than Nearchou's.

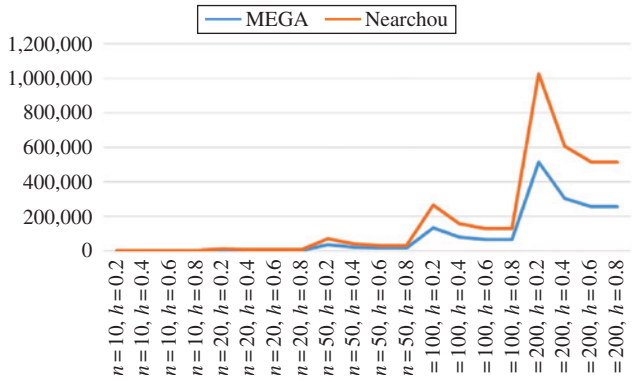


Figure 2: Graphical Representation of Results.

Table 2: Summary Results.

	Better	Equal	Worse
Number of instances	51	129	20
Percentages	25.5%	64.5%	10%

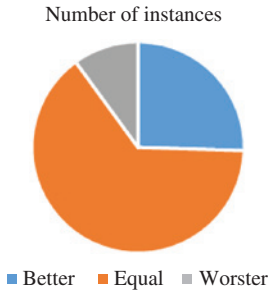


Figure 3: Graphical Representation of Results.

7 Conclusion and Perspectives

In this paper, a new mathematically enhanced variant of GA was proposed for solving single-machine scheduling problems with the objective of minimizing the sum of earliness and tardiness penalty costs for the deviation from a common due date. This type of problems are known and proofed as NP-hard. That is why an approximate method such as GA is necessary to adopt. Two new properties for the problem were identified, and their associated proofs were provided.

The already existing three properties might stay insufficient, because they only describe the optimal sequence structure without giving any details about the beginning time of the optimal sequence.

It should be noted that these new properties allow confirming that the sequence may start at time zero or be delayed to minimize the early/tardy penalties against the common due date. They also allow computing exactly the starting time of the optimal sequence. The experimental results show that these properties have a significant contribution in terms of solution quality and the time complexity of the approach. They are especially used to compute the beginning time of the optimal sequence. The results were compared to the best ones of the literature in the field of combinatorial optimization.

As for future perspectives and based on the encouraging results, it is suggested to study the impact of each property alone on the required computational effort and quality of solutions. Further, we will investigate

the new proposed approach to solutions of other related scheduling programs like single-machine scheduling with multiple due dates, as well as parallel machine scheduling problems.

Bibliography

[1] R. Alvarez-Valdes, E. Crespo, J. M. Tamarit and F. Villa, Minimizing weighted earliness-tardiness on a single machine with a common due date using quadratic models, *TOP* **20** (2012), 754–767.

[2] A. Awasthi, J. Lassig and O. Kramer, Common due-date problem: exact polynomial algorithms for a given job sequence, 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. Data Structures and Algorithms (cs.DS); Combinatorics (math.CO). Timisoara, Romania (2013).

[3] K. R. Baker and G. D. Scudder, Scheduling with earliness and tardiness penalties: a review, *Eur. J. Oper. Res.* **160** (2005), 190–201.

[4] K. R. Baker and G. D. Scudder, Minimizing earliness and tardiness costs in stochastic scheduling, *Eur. J. Oper. Res.* **236** (2013), 445–452.

[5] D. Biskup, Single-machine scheduling with learning considerations, *Eur. J. Oper. Res.* **115** (1999), 173–178.

[6] C. N. Cha, S. Lim and Y. K. Jeong, Single-machine job scheduling about a common due date with arbitrary earliness/tardiness penalties using a genetic algorithm, *Asia Pacific Management Review* **7** (2002), 239–254.

[7] M. Feldman and D. Biskup, Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates, *Comput. Ind. Eng.* **28** (2001), 787–801.

[8] M. Feldman and D. Biskup, Single-machine scheduling for minimizing earliness and tardiness penalties by meat-heuristic approaches, *Comput. Ind. Eng.* **44** (2003), 307–323.

[9] V. Gordon, J. M. Proth and C. Chu, Invited review: A survey of the state-of-the-art of common due date assignment and scheduling research, *Eur. J. Oper. Res.* **139** (2002), 1–25.

[10] R. Hassin and M. Shani, Machine scheduling with earliness and tardiness and non-execution penalties, *Comput. Ind. Eng.* **32** (2005), 683–705.

[11] A. Hemmak and B. Bouderah, Hybrid algorithm for optimization problems applied to single machine scheduling, *Int J. Comput. Appl.* **66** (2013), 7–11.

[12] A. Hemmak and B. Bouderah, Sieve algorithm – a new method for optimization problems, *Int. J. Advance. Soft Comput. Appl.* **5** (2013), 1–15.

[13] A. Hemmak and B. Bouderah, A mono crossover genetic algorithm for TSP, *Global J. Tech.* **7** (2015), 109–115.

[14] A. Hemmak and I. H. Osman, Variable parameters lengths genetic algorithm for minimizing earliness-tardiness penalties of single machine scheduling with a common due date, *Electron. Notes Discrete Math.* **36** (2010), 471–478.

[15] C. M. Hino, D. P. Ronconi and A. B. Mendes, Minimizing earliness and tardiness penalties in a single machine problem with a common due date, *Eur. J. Oper. Res.* **160** (2005), 190–201.

[16] A. Jafarnejad, S. M. Abtahi and S. M. R. Davoodi, Optimizing the earliness and tardiness penalties in the single-machine scheduling problems with focus on the just in time, *Int. J. Acad. Res. Bus. Soc. Sci.* **3** (2013), 315–322.

[17] J. Lassig, A. Awasthi and O. Kramer, Common due-date problem: linear algorithm for a given job sequence, in: *2014 IEEE 17th International Conference on Computational Science and Engineering*, University of Electronic Science and Technology of China, 2014.

[18] C. J. Liao and C. C. Cheng, A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date, *Comput. Ind. Eng.* **52** (2007), 404–413.

[19] S. W. Lin, S. Y. Chou and K. C. Ying, A sequential exchange approach for minimizing earliness-tardiness penalties of single machine scheduling with a common due date, *Eur. J. Oper. Res.* **177** (2007), 1294–1301.

[20] A. C. Nearchou, A differential evolution approach for the common due date early/tardy job scheduling problem, *Comput. Oper. Res.* **35** (2008), 1329–1343.

[21] Z. Ning, C. Tao and L. Fei, A hybrid heuristic algorithm for the intelligent transportation scheduling problem of the BRT system, *J. Intell. Syst.* **24** (2015), 437–448.

[22] D. P. Ronconi and M. S. Kawamura, The single machine earliness and tardiness scheduling problem: lower bounds and a branch-and-bound algorithm, *Comput. Appl. Math.* **29** (2010), 107–124.

[23] M. Shahriari, N. Shoja, A. E. Zade, S. Barak and M. Sharifi, JIT single machine scheduling problem with periodic preventive maintenance, *J. Ind. Eng. Int.* (2016), 1–12.

[24] T. Vallée and M. Yiltizogli, Présentation des algorithmes génétiques et leurs applications en économie, *Mai* **5** (2004), Rapport technique.

[25] K. C. Ying, Minimizing earliness-tardiness penalties for common due date single-machine scheduling problems by a recovering beam search algorithm, *Comput. Ind. Eng.* **55** (2008), 494–502.