An Adaptive Genetic Algorithm to the Single Machine Scheduling Problem with Earliness and Tardiness Penalties

Fábio Fernandes Ribeiro¹, Marcone Jamilson Freitas Souza², and Sérgio Ricardo de Souza¹

¹ Centro Federal de Educação Tecnológica de Minas Gerais - CEFET/MG
Programa de Pós-Graduação em Modelagem Matemática e Computacional,
Avenida Amazonas 7675, Belo Horizonte, CEP: 30510-000, Minas Gerais, Brazil
fabiobh@gmail.com, sergio@dppg.cefetmg.br
http://www.mmc.cefetmg.br

² Universidade Federal de Ouro Preto - UFOP, Departamento de Computação, ICEB
Campus Universitário (UFOP), Ouro Preto, CEP: 35400-000, Minas Gerais, Brazil
marcone@iceb.ufop.br
http://www.iceb.ufop.br

Abstract. This paper deals with the Single Machine Scheduling Problem with Earliness and Tardiness Penalties, considering distinct due windows and sequence-dependent setup time. Due to its complexity, an adaptive genetic algorithm is proposed for solving it. Many search operators are used to explore the solution space where the choice probability for each operator depends on the success in a previous search. The initial population is generated by the combination between construct methods based on greedy, random and GRASP techniques. For each job sequence generated, a polynomial time algorithm is used for determining the processing initial optimal date to each job. During the evaluation process, the best individuals produced are added to a special group, called elite group. The individuals of this group are submitted to refinement, aiming to improve their quality. Three variations of this algorithm are submitted to computational test. The results show the effectiveness of the proposed algorithm.

1 Introduction

Scheduling problems are one of the most studied problems in combinatorial optimization [1]. It occurs mainly by two aspects: the first one concerns their practical importance, with various applications in several industrial fields. The second aspect is about the difficulty for solving the majority problems of this class. This paper deals with the Single Machine Scheduling Problem with Earliness and Tardiness Penalties (SMSPETP) with distinct due windows and sequence-dependent setup time. To our knowledge, this problem has not been still object of great attention of the scientific community, as it could seen in the recent survey [1].

The criteria to penalize the tardiness and earliness production goes to the Just-in-Time philosophy goal, that is, the production is done just when necessary.
The existence of a due window for each job, according to [2], is because of an uncertainly situation or tolerance related to due date. It is accepted that this time interval operations can be finalized without costs. On the other hand, in most industrial processes the machines should first be prepared for doing new jobs, including the time to obtain tools, positioning materials that will be used in the process, cleaning process, preparing process, tools adjustment and materials inspection. The necessary time to this preparation is known by setup time. Many production scheduling researches disregard this time or include it in the operation processing time. This act simplifies the analysis but affect the solution quality when the setup time has a relevant variability in function of the job sequence in machine. This work considers that the setup times are dependents from the production scheduling. Since it was showed in [3] that a simplified version of this problem is NP-Hard, the application of metaheuristics for solving this problem is justified.

In order to solve this scheduling problem with the presented characteristics, an Adaptive Genetic Algorithm, so-called AGA, is proposed here. To generate different individuals having good quality, the initial population is generated by a construction method based on GRASP [5], which uses five dispatch rules to form the individuals. During the evolution process, the population passes through mutation and crossover conventional process. However, the crossover uses criteria based on solution quality generated by each crossover operator to choose which operator will be used. By the way, according to how well an operator performs, the probabilities it be chosen is increased or decreased during the evolution. A local search is applied in the best offspring produced for each operator, to refine it. The survival population is composed by individuals chosen by elitism technique. Mutation process is then applied to a slice of the surviving population for diversifying it. Periodically, a Path Relinking module is applied taking the best one so far generated by the algorithm as base individual and each one of the five best individuals generated by each crossover operator as guide individual. The population improvement occurs until the stop criteria is reached.

The remaining of this work is organized as follows: section 2 details the studied problem; section 3 presents the adaptive algorithm for solving SMSETP; section 4 shows and discuss the results; finally, section 5 ends this work.

2 Problem Description

This work studies the single machine scheduling problem, with earliness and tardiness penalties, distinct due windows and sequence-dependent setup time. In this problem, one machine must process a set of \( n \) jobs. Each job \( i \) has processing time \( P_i \), initial date \( E_i \) and final date \( T_i \), desired for ending the processing. The machine executes one job per time and, if a job processing is started, it must be finished, since processing interruptions are not allowed. All jobs are available for processing in the instant 0. When a job \( j \) is sequenced immediately after a job \( i \), for setting the machine is necessary a setup time \( S_{ij} \). Setup times equal 0 mean products of the same family. The initial setup times are considered, i.e., the setup
time to the first job in the sequence is 0. The idle time between the execution with two consecutive jobs is allowed. The jobs must be finalized inside the time interval \([E_i, T_i]\), called due window. In case of job finalization before \(E_i\), there is a cost to earliness. Case the job are finalized after \(T_i\), a cost will be generated for tardiness. For jobs completed within due windows, none cost is incurred. The costs to earliness and tardiness of production depend on jobs. Each job \(i\) have a earliness cost \(\alpha_i\) and a tardiness cost \(\beta_i\). Finally, the objective of the problem is to minimize the summation of the earliness and tardiness penalties.

This scheduling problem is described by the mixed integer programming model (MIP) shown below, based on [6]:

\[
\begin{align*}
\text{min} \quad Z &= \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \quad (1) \\
\text{s.t.} \quad s_j - s_i - y_{ij}(M + S_{ij}) &\geq P_i - M \quad \forall \ i = 0, \ldots, n; \quad j = 0, \ldots, n; \quad i \neq j \quad (2) \\
\sum_{j=0, j \neq i}^{n} y_{ij} &= 1 \quad \forall \ i = 0, \ldots, n \quad (3) \\
\sum_{i=0, i \neq j}^{n} y_{ij} &= 1 \quad \forall \ j = 0, \ldots, n \quad (4) \\
s_i + P_i + e_i &\geq E_i \quad \forall \ i = 1, \ldots, n \quad (5) \\
s_i + P_i - t_i &\leq T_i \quad \forall \ i = 1, \ldots, n \quad (6) \\
s_i &\geq 0 \quad \forall \ i = 0, \ldots, n \quad (7) \\
e_i &\geq 0 \quad \forall \ i = 1, \ldots, n \quad (8) \\
t_i &\geq 0 \quad \forall \ i = 1, \ldots, n \quad (9) \\
y_{ij} &\in \{0, 1\} \quad \forall \ i, j = 0, \ldots, n \quad (10)
\end{align*}
\]

where:

- \(s_i\): the starting time of job \(i\);
- \(C_i\): the completion time of job \(i\);
- \(y_{ij}\): binary variable that assumes value 1 if job \(j\) is processed immediately after job \(i\) and 0, otherwise;
- \(e_i\): the earliness of job \(i\), that is, \(e_i = \max\{0, E_i - C_i\}\);
- \(t_i\): the tardiness of job \(i\), that is, \(t_i = \max\{0, C_i - T_i\}\);
- \(M\): a sufficiently large number;
- \(0\): a fictitious job, which precedes and succeeds all other jobs;

It also assumes that \(P_0 = 0, S_{0i} = S_{i0} = 0 \ \forall i \in \{1, 2, \ldots, n\}\)

The objective function (1) expresses the total earliness and tardiness cost. The constraints (2) establish that job \(j\) can be processed when job \(i\) is finished and the machine is prepared to processes it. The constraints (3), (4) and (10) guarantee that the variable \(y_{ij}\) assumes value 1 if and only if job \(j\) is processed immediately after job \(i\). The constraints (5) and (6) define, respectively, the tardiness and earliness values according of the due window. The constraints (7) to (10) define the type of the variables.
3 Heuristic Framework

In this section, the adaptive genetic algorithm framework proposed for solving the problem is described.

**Individual representation:** An individual (i.e., a solution) is represented by a vector \( v \) of \( n \) genes (jobs). The production sequence of each job is given by the \( i \) position in the vector. For example, in the sequence \( v = \{5, 2, 7, 4, 6, 3, 1\} \), the job 5 is the first to be processed and the job 1 is the last.

**Evaluation of individuals:** All of individuals are evaluated by the same objective function, given by expression (1) of MIP model (Mixed Integer Programming), where the individual which obtained the shortest value to the objective function is considered the most adapted.

**Initial population construction:** The initial population of the proposed adaptive genetic algorithm is generated by GRASP construction phase ([5]), having five dispatch rules (EDD, TDD, SPT, WSPT and LPT) as guide function. For each construction (GRASP + dispatch rule), 200 individuals are generated. In the sequel, the individuals are ordered according to evaluation function, from the best one to the worst one. The initial population is composed by the 100 best generated individuals.

**GRASP construction procedure:** In this construction procedure, an offspring is formed by genes that are inserted one by one. The offspring is constructed according with a partially greedy selection criteria. To estimate the insertion benefit of each gene, dispatch rules EDD, TDD, SPT, WSPT and LPT are used. Each variant gives a different construction. In Figure 1 the GRASP construction phase is showed. In this figure, \( g_{\text{min}} \) represents the best evaluation according to the selected dispatch rule and \( g_{\text{max}} \), the worst one.

```
procedure Construction(g(.), γ, v);
1   v ← ∅;
2   Initialize a set \( C \) of candidate genes;
3   while \( (C ≠ ∅) \) do
4       \( g_{\text{min}} = \min\{g(t) \mid t ∈ C\}; \)
5       \( g_{\text{max}} = \max\{g(t) \mid t ∈ C\}; \)
6       \( RCL = \{t ∈ C \mid g(t) ≤ g_{\text{min}} + γ \times (g_{\text{max}} - g_{\text{min}})\}; \)
7       Select, randomly, a gene \( t ∈ RCL; \)
8       v ← v ∪ \{t\};
9       Update C;
10  end-while;
11  Return v;
end Construction;
```

Fig. 1. Procedure to build an individual
An Adaptive Genetic Algorithm to the SMSPETP

Algorithm AGA(germax, nind, probcross, probmut, freq);
1 \( t \leftarrow 0; \)
2 Generate Initial Population \( P(t); \)
3 Evaluate \( P(t); \)
4 while \( (t < \text{germax}) \) do
5 \( t \leftarrow t + 1; \)
6 \( P(t) \leftarrow P(t - 1); \)
7 \( i \leftarrow 0; \) \{ number of new individuals \}
8 while \( (i < \text{nind}) \) do
9 Select two individuals from \( P(t - 1); \)
10 \( \text{cross} \leftarrow \) Randomly number from 1 to 100;
11 if \( (\text{cross} \leq \text{probcross}) \) then
12 Choose a crossover operator \( O_k; \)
13 Apply the chosen crossover operator;
14 \( i \leftarrow i + 2; \)
15 Incorporate the new individuals to \( P(t); \)
16 end-if;
17 end-while;
18 Evaluate \( P(t); \)
19 Define \( \text{nind} \) survivors;
20 Apply mutation with probability \( \text{probmut} \) in all members of population \( P(t) \)
21 if \( (t \mod \text{freq} = 0) \) then
22 Update the probability of selecting each crossover operator \( (p(O_k)); \)
23 Execute Local Search;
24 Apply Path Relinking;
25 end-if;
26 end-while;
end AGA;

Fig. 2. Pseudocode of the proposed Adaptive Genetic Algorithm

3.1 Adaptive Genetic Algorithm Applied to SMSETP

Figure 2 shows the pseudocode of the proposed Adaptive Genetic Algorithm (AGA). The algorithm phases are described in the following.

Individual selection method: The individuals are selected to reproduction (line 9 of Figure 2) by the tournament selection. In the implemented strategy, only one tournament involving all the population is realized. The winner of the tournament, that is, the one with the best fitness, is selected for crossover with each individual from the population. Therefore, all the individuals are selected to reproduction.

Crossover: The crossover process uses the following operators: (i) One Point Crossover (OX), (ii) Similar Job Order Crossover (SJOX), (iii) Relative Job Order Crossover (RRX), (iv) Based Order Uniform Crossover (BOUX) and (v) Partially Mapped Crossover (PMX). This choice was taken by the fact of this
operators being the most common operators to solve problems like this by genetic algorithm [4]. The choice probability of crossover operators modifies according to the quality of individuals produced by the operators in the past generations. More specifically, let \( O_k \), with \( k = 1, \cdots, 5 \), be the five crossover operators. Initially, each crossover operator \( O_k \) has the same probability to be chosen, that means, \( p(O_k) = 1/5 \). Let \( f(s^*) \) be the best individual so far and \( A_k \) the average value found for each operator \( O_k \) since the last update. Case the operator was not chosen in last five generations, make \( A_k = 0 \). Then, calculate \( q_k = f(s^*)/A_k \) and \( p(O_k) = q_k / \sum_{j=1}^{5} q_j \) for all \( k = 1, \cdots, 5 \). Observe that how much better the individual is, more high is the value of \( q_k \) and, consequently, the probability of choosing the \( O_k \) operator is increased. Therefore, during the algorithm evolution, the best operator have its chance of choice increased. This procedure is inspired in Reactive GRASP algorithm, proposed by [7].

**Local search:** Like said previously, at each five generations, a local search is applied to the best individual generated by each crossover operator. The local search used method is Random descent. This method uses two kinds of movement to explore the search space: the change of two jobs of the the sequence and the job relocation to another production sequence. The method works as follows: to an individual, two jobs are selected randomly and the positions are exchanged. If the new individual is better than the current one, according to the evaluation function, it is accepted and becomes the current solution; otherwise, another movement are randomly chosen. If during \( MRD_{\text{max}} \) iterations any solution better than the current one are generated, then relocate movements are used. If there is any improvement in this phase, the method returns to use exchange movements; otherwise, the local search is ended up after \( MRD_{\text{max}} \) iterations without improvement.

**Path Relinking:** During the evolutive process, a group with the best five individuals generated by each crossover operator are built. At each five generations, Path Relinking (PR) procedure is applied, taking as base solution the best individual generated by the method and as guide individual each one of the five best individuals generated by each crossover operator. The PR is interrupted when 75% of the guide individual is added to the base solution. Besides it, the base solution is better than the guide solution. So, it is so-called Truncated Backward Path Relinking. A job position of production sequence is considered as an attribute. For each job candidate to insertion, a local search method like described previously is applied, and a movement of a fixed job is not allowed.

**Surviving individuals:** The surviving individuals are chosen by the elitism technique. So, the individuals more adapted survive.

**Stop criteria:** The maximum number of generations is used as a stop criteria of the adaptive genetic algorithm.

### 3.2 Variants of Proposed Algorithm

In this work, three variants of AGA are shown. In Variant 1, called AGA1, the refresh of crossover operator selection rate \( (freq = 5 \) in Fig. [2]) happens at
each five generations. After that, all elite group members are submitted to local search (see local search paragraph) and, in the sequel, they are submitted to path relinking procedure. In this variant, the elite group are composed by the best individuals produced by each crossover operator in last five generations.

Variant 2, called AGA2, differs to AGA1 variant by elite group composition. In variant 2, elite group are composed by the best individual produced by each crossover operators globally and not just at last five generations.

In Variant 3, called AGA3, the refresh of crossover operator selection rate, the elite group submission to local search and the submission to path relinking procedure happens at each ten generations ($freq = 10$ in Fig. 2). The elite group are composed by the best three individuals globally produced, by the best solution produced at past ten generation, if this individual have diversity index upper than 30% for another individuals in elite group. If the individual does not fit in this criteria, the second best solution are analyzed and so on until the one of them satisfied this criteria is found. The fifth element is chosen by selecting at random an individual of a set of the best ten individuals produced over past ten generations. To compute the diversity index of two individuals, the number of different genes in a same position is summed and this value is divided by the total number of genes of the individual.

4 Computational Experiments

The proposed algorithm was developed in C++ language, using Borland C++ Builder 5.0 compiler. The used parameters were obtained experimentally and they are presented in Table 1.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter $\gamma$ GRASP construction phase</td>
<td>0.20</td>
</tr>
<tr>
<td>Maximum iterations of local search ($MRD_{max}$)</td>
<td>$7 \times n$</td>
</tr>
<tr>
<td>Maximum generations of AGA ($ger_{max}$)</td>
<td>100</td>
</tr>
<tr>
<td>Crossover probability ($prob_{cross}$)</td>
<td>80%</td>
</tr>
<tr>
<td>Mutation rate ($prob_{mut}$)</td>
<td>5%</td>
</tr>
</tbody>
</table>

Two instances were used to test the three variants of AGA. The first one is the same of [6]. These authors generated instances randomly with job number equal to 6, 7, 8, 9, 10, 11, 12, 15, 20, 25, 30, 35, 40, 50 and 75, using the same parameters of [2], [8] and [9]. The second one was generated by [11] with jobs numbers equal to 6, 7, 8, 9, 10, 11, 12, 15, 20, 25, 30, 35, 40, 50, 75 and 100. These instances are available for download at http://www.iceb.ufop.br/decom/prof/marcone/projects/scheduling/instances.htm.

All experiments were realized in a Pentium Core 2 Duo 2.1 GHz computer with 4 GB RAM and Windows Vista operational system. Two sets of experiments were realized. The description, details and results of each one of these experiments are described in the following.
Table 2. Results of the first set of experiments - BAT 1

<table>
<thead>
<tr>
<th># Jobs</th>
<th>Average result deviation</th>
<th>Best result deviation</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AGA 1</td>
<td>AGA 2</td>
<td>AGA 3</td>
</tr>
<tr>
<td>8</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>9</td>
<td>0.15%</td>
<td>0.16%</td>
<td>0.24%</td>
</tr>
<tr>
<td>10</td>
<td>0.24%</td>
<td>0.25%</td>
<td>0.41%</td>
</tr>
<tr>
<td>11</td>
<td>0.03%</td>
<td>0.05%</td>
<td>0.10%</td>
</tr>
<tr>
<td>12</td>
<td>0.07%</td>
<td>0.08%</td>
<td>0.21%</td>
</tr>
<tr>
<td>15</td>
<td>0.76%</td>
<td>0.80%</td>
<td>1.16%</td>
</tr>
<tr>
<td>20</td>
<td>0.73%</td>
<td>0.75%</td>
<td>0.85%</td>
</tr>
<tr>
<td>25</td>
<td>1.02%</td>
<td>1.08%</td>
<td>1.42%</td>
</tr>
<tr>
<td>30</td>
<td>1.60%</td>
<td>1.82%</td>
<td>2.64%</td>
</tr>
<tr>
<td>40</td>
<td>2.33%</td>
<td>2.54%</td>
<td>3.56%</td>
</tr>
<tr>
<td>50</td>
<td>4.06%</td>
<td>4.37%</td>
<td>6.32%</td>
</tr>
<tr>
<td>75</td>
<td>6.52%</td>
<td>9.48%</td>
<td>11.86%</td>
</tr>
<tr>
<td>Avg</td>
<td>1.46%</td>
<td>1.77%</td>
<td>2.40%</td>
</tr>
</tbody>
</table>

Table 3. Comparing AGA 1 × GTPRS, proposed by [12]

<table>
<thead>
<tr>
<th># Jobs</th>
<th>Average result deviation</th>
<th>Best result deviation</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AGA 1</td>
<td>GTPRS</td>
<td>% Improv.</td>
</tr>
<tr>
<td>8</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>9</td>
<td>0.15%</td>
<td>0.00%</td>
<td>-15.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.24%</td>
<td>0.00%</td>
<td>-24.00%</td>
</tr>
<tr>
<td>11</td>
<td>0.03%</td>
<td>0.00%</td>
<td>-3.00%</td>
</tr>
<tr>
<td>12</td>
<td>0.07%</td>
<td>0.00%</td>
<td>-7.00%</td>
</tr>
<tr>
<td>15</td>
<td>0.76%</td>
<td>1.25%</td>
<td>64.08%</td>
</tr>
<tr>
<td>20</td>
<td>0.73%</td>
<td>1.11%</td>
<td>51.87%</td>
</tr>
<tr>
<td>25</td>
<td>1.02%</td>
<td>1.60%</td>
<td>56.53%</td>
</tr>
<tr>
<td>30</td>
<td>1.60%</td>
<td>2.57%</td>
<td>60.61%</td>
</tr>
<tr>
<td>40</td>
<td>2.33%</td>
<td>3.77%</td>
<td>61.84%</td>
</tr>
<tr>
<td>50</td>
<td>4.06%</td>
<td>5.58%</td>
<td>37.64%</td>
</tr>
<tr>
<td>75</td>
<td>6.52%</td>
<td>9.48%</td>
<td>11.86%</td>
</tr>
<tr>
<td>Avg</td>
<td>1.46%</td>
<td>2.01%</td>
<td>2.27%</td>
</tr>
</tbody>
</table>

4.1 The First Set of Experiments - BAT 1

The first set of experiments uses the first instance of problems. Each set of problems was tested 30 times for AGA1, AGA2 and AGA3, the variants of AGA method. Table 2 shows the results reached in this set of experiments. The first column shows the number of jobs; the second, third and fourth columns show how much the average of solutions of each variant are diverted from the best solution known. In the fifth, sixth and seventh columns are showed how much the best generated solutions are diverted from the best solution known. In the eighth, ninth and tenth columns the computational time average of applying the variants of AGA are showed.

Table 3 compares AGA1 with the algorithm GTSPR, proposed by [12]. In this table, “% Improv.” indicates how much AGA1 improves the solutions produced by GTSPR with relation to the average deviation (or to the best deviation).

4.2 The Second Set of Experiments - BAT 2

The second set of experiments uses the second instance of problems. Each set of problems was tested 30 times for each variant of AGA method. Table 4 shows the...
An Adaptive Genetic Algorithm to the SMSPETP

Table 4. Results of the second set of experiments - BAT 2

<table>
<thead>
<tr>
<th># Jobs</th>
<th>Average result deviation</th>
<th>Best result Deviation</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AGA 1  AGA 2  AGA 3</td>
<td>AGA 1  AGA 2  AGA 3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.00%  0.00%  0.00%</td>
<td>0.00%  0.00%  0.00%</td>
<td>0.76</td>
</tr>
<tr>
<td>7</td>
<td>0.00%  0.00%  0.00%</td>
<td>0.00%  0.00%  0.00%</td>
<td>0.76</td>
</tr>
<tr>
<td>8</td>
<td>0.00%  0.13%  0.20%</td>
<td>0.00%  0.13%  0.20%</td>
<td>1.03</td>
</tr>
<tr>
<td>9</td>
<td>0.00%  0.03%  0.06%</td>
<td>0.00%  0.03%  0.06%</td>
<td>1.61</td>
</tr>
<tr>
<td>10</td>
<td>0.00%  0.11%  0.17%</td>
<td>5.97%  0.11%  6.16%</td>
<td>2.20</td>
</tr>
<tr>
<td>11</td>
<td>0.00%  0.01%  0.11%</td>
<td>0.00%  0.01%  0.11%</td>
<td>3.32</td>
</tr>
<tr>
<td>12</td>
<td>0.00%  0.14%  0.37%</td>
<td>0.01%  0.14%  0.37%</td>
<td>7.49</td>
</tr>
<tr>
<td>15</td>
<td>0.30%  0.64%  0.81%</td>
<td>0.30%  0.64%  0.81%</td>
<td>23.88</td>
</tr>
<tr>
<td>20</td>
<td>1.20%  1.19%  1.16%</td>
<td>1.36%  1.51%  1.50%</td>
<td>172.51</td>
</tr>
<tr>
<td>30</td>
<td>0.98%  1.21%  1.26%</td>
<td>1.36%  1.42%  2.34%</td>
<td>801.67</td>
</tr>
<tr>
<td>50</td>
<td>1.14%  1.46%  1.14%</td>
<td>1.26%  2.24%  2.58%</td>
<td>1575.11</td>
</tr>
<tr>
<td>75</td>
<td>0.00%  1.26%  2.36%</td>
<td>0.09%  1.67%  3.60%</td>
<td>4978.02</td>
</tr>
<tr>
<td>100</td>
<td>0.25%  2.50%  1.10%</td>
<td>0.25%  3.14%  2.41%</td>
<td>18107.72</td>
</tr>
<tr>
<td>Avg</td>
<td>0.28%  0.62%  0.62%</td>
<td>0.73%  0.79%  1.44%</td>
<td>1834.10</td>
</tr>
</tbody>
</table>

results reached in the first set of experiments. The first column shows the number of jobs. The second, third and fourth columns show how much the average of solutions of each variant are diverted from the best solution known. In the fifth, sixth and seventh columns are showed how much the best generated solutions are diverted from the best solution known. In the eighth, ninth and tenth columns the computational time average of applying the variants of AGA are showed.

5 Conclusions

This paper dealt with the single machine scheduling problem with earliness and tardiness penalties, considering distinct due windows and sequence-dependent setup time. To solve this problem, an adaptive genetic algorithm, named AGA, was proposed. The initial population was generated by a GRASP procedure using dispatch rules as guide functions. During the evolution process, population is submitted to selection, crossover and mutation processes. In crossover process, five operators are used, being that the best solutions produced by each operator are submitted to local search and path relinking. The path relinking procedure connects the best solution reached so far with the best solutions produced by each operator.

By the end, two set of instances were used to test the proposed algorithm, and three variants of AGA were developed. The results of each instance were compared with another algorithm from the literature. In these experiments, the proposed algorithm presented high quality solutions with lower gap, always reaching the best known value. The developed algorithm presented solutions better than the best solutions found in the literature, besides presenting a minor variability of final solutions.

Acknowledgment

The authors would like to thank CEFET/MG, CAPES, FAPEMIG and FAPERJ for the support to development of this work.
## References