A Multi-Population Genetic Algorithm Approach for PID Controller Auto-Tuning

Claudio Fabiano Motta Toledo
Institute of Mathematics and Computer Science
São Paulo University, São Carlos, Brazil
claudio@icmc.usp.br

João Miguel G. Lima
Department of Electronic Engineering and Informatics
University of Algarve, Faro, Portugal
jlima@ualg.pt

Márcio da Silva Arantes
Institute of Mathematics and Computer Science
São Paulo University, São Carlos, Brazil
marcio@icmc.usp.br

Abstract

The present paper applies a multi-population genetic algorithm (MPGA) to the Proportional, Integral and Derivative (PID) controller tuning problem. Two control criteria were optimized, the integral of the time multiplied by the absolute error (ITAE), and the integral of the time multiplied by the absolute output (ITAY). The MPGAl is compared with a standard genetic algorithm (SGA) already applied to the same control model. The control criteria are supplied by neural networks (NN) previously trained for this purpose. The control tuning and the corresponding responses were obtained using the MATLAB/SIMULINK environment. The computational results show a superior performance of the MPGAl even when compared with the exact values found by dynamic simulation using gradient techniques.

1. Introduction

Proportional, Integral and Derivative (PID) controllers are widely used in a several kind of applications due to its simple structure (only three terms to tune) and robust performance over a large range of operation conditions. The popularity of this controller justifies the great amount of literature about it, among them, [1] and [8] constitute some examples.

The PID control module is a building block that provides the regulation and disturbance rejection for several control topologies; a single loop control with a first order plant with time delay was used in this work.

In the robotics field, the PID controller is being used to control many components of a system, such as, the position of cameras, manipulators [21], wheels [18] etc. Recently, the PID controller was used to control the gain of a CMOS image sensor [19].

Since 1942, with the work of Ziegler and Nichols [22], several methods of PID controllers auto-tuning have been proposed in the literature. Recently, a technique [9] was proposed in which neural networks (NN) trained off-line are used to supply on-line parameters for the PID controller, optimised for a given control criterion. The tuning procedure is an optimization task for a given set of control criteria. Thus, the NN was used for modelling purpose and a standard genetic algorithm (SGA) [4] was applied for optimization purpose.

However, the SGA demands a significant amount of solution evaluation as well as the best solution obtained can still be improved. Taking into account our goal of a real time implementation in a digital hardware like field programmable gate array (FPGA), a more efficient approach is desirable.

In this paper a multi-population genetic algorithm (MPGA) is proposed for PID tuning. This kind of MPGAl presents individuals hierarchically structured in tree. This type of genetic algorithm (GA) is used in [6] to solve the total tardiness single machine scheduling problem, where a memetic algorithm (MA) outperforms a standard GA.

A MA with population hierarchically structured is proposed by [3] to solve instances of the travelling salesman problem. This method returns better results than six other approaches from the literature for 27 problem instances. A MA with 13 individuals hierarchically structured in a ternary tree is applied by [17] to the gene ordering problem, returning better results for two microarray data sets. The two-level soft drink planning problem is solved in [20] by a MPGAl using three populations with 13 individuals each, hierarchically disposed in ternary trees.

In this paper we present and discuss the optimization results obtained with both GAs (SGA and MPGAl) as well as the corresponding PID controller tunings.
The paper is structured as follows: section 2 is dedicated to problem description; in this section the control system is presented as well as the modelling and optimization blocks. In section 3 both optimization methods are detailed. The results obtained comparing the GAs and the corresponding effects in terms of PID tuning are reported in section 4. The paper ends with section 5 where some conclusions are presented, and suggestions of future works are pointed out.

2. Problem description

An accurate PID tuning aims at producing an adequate behavior for a controlled system [1] (bottom of Figure 1; represented in terms of Laplace transform (s variable)).

![Figure 1. Control system + tuning block.](image)

In the bottom of Figure 1, R(s) is the input reference, Y(s) is the output response and G(s) is the plant transfer function. PI(s) and D(s) are proportional + integral and derivative blocks, respectively represented by transfer functions (1) and (2).

\[ PI(s) = k_p \left( 1 + \frac{1}{st_f} \right) \]  

\[ D(s) = \frac{1 + st_d}{1 + st_f}, \quad t_r = \frac{t_f}{10} \]  

The derivative block is associated with a low-pass filter with time constant \( t_f \). In the remaining text, the PID parameters are referred as a vector: \( PID = [k_p, t_i, t_d] \).

In general we deal with a time variant plant G(s); however, for the objective of the present work, the plant was considered time invariant, so, we choose as an example the plant (3) that is an industrial processes representative of a first order with time delay.

\[ G(s) = \frac{e^{-s}}{s + 1} \]  

As already indicated, the main goal of PID auto-tuning is to deliver the optimized PID parameters in real time for a given set of control criteria. So, we conclude that during a normal operation of a control loop (Figure 1) some optimizing procedures should take place. These procedures, no matter which optimization method has been employed, intermediate solutions appear before obtaining the optimal solution. Those intermediate solutions, depending on the real application used, could cause great damage to the control system during its online operation. In order to overcome this problem it was decided to build a model of the plant and use it instead of the original plant [11], for tuning purposes. Thus, starting from the standard closed loop control system (Figure 1, in the bottom), our complete system coupled with the tuning block (Figure 1, in the top).

The contribution of this work is inserted into the optimizer block (GA) which uses the model (NN) to evaluate and supply optimal PID parameters to the control system (Figure 1). In spite of this approach allowing the optimization of any control criterion, for testing purpose, it was considered two objectives for PID tuning:

1. reference tracking,
2. output disturbance rejection.

In relation to the objective 1, the minimization of the integral of the time multiplied by the absolute error \( ITAE \) criterion (4)) to a reference step \( R(s) = \frac{1}{s} \) is employed.

\[ ITAE = \int t|e(t)|dt, \quad e(t) = y(t) - r(t) \]  

As for the objective 2, considering a null reference \( R(s) = 0 \) and a unit step added to the G(s) output, the criterion used to minimise is the integral of the time multiplied by the absolute output \( y(t) \) (\( ITAY \) criterion).

\[ ITAY = \int t|y(t)|dt \]  

In this work, three single objective minimizations were considered: \( ITAE, ITAY \) and the weighted sum of both (6).

\[ ITAE = 0.5ITAE + 0.5ITAY \]  

What is concerned the GA minimization, the fitness was evaluated by propagation of the PID parameters through the corresponding NN. Several types of artificial neural networks were already tested for modelling purpose (see details in [10]). For this work it was chosen artificial NN of the type multi-layer perceptrons (MLP) with a sigmoid activation function. It was trained off-line
two MLP, one for ITAE mapping and other for ITAY mapping (Figure 2).

Figure 2 Mappings between PID parameters and the control criterion (in this case, ITAE or ITAY).

Several experiments took place and the best generalisation performance was reached for a topology with 2 hidden layers MLP with 9 and 5 neurons in the 1st and 2nd hidden layers, respectively: [3 9 5 1]; this subject was detailed in [9].

For comparison purposes, an alternative optimization procedure which uses a gradient method implemented in MATLAB optimization toolbox [13], took place. To accomplish it, the control loop was modelled in SIMULINK and the criteria to minimize ((4), (5) or (6)) were coded in MATLAB [14].

Previous research has shown that unstable solutions are obtained before the final solution more frequently using gradient methods than GA [11]. We can conclude that optimization based on gradient methods can be unfeasible for a real time implementation. Therefore, it is only used in the present work for comparison purpose, appearing in the text (section 4) as “Reference Value”.

3. Genetic algorithms

This section is dedicated to GAs which are the optimization tools used in the optimizer block of the Figure 1. First, the pseudocodes of the GA are described. Next, the representation of individual, fitness calculation and genetic operators are detailed. The SGA previously applied by [10] to the PID auto-tuning is presented aiming to become the text self-contained. The multi-population genetic algorithm (MPGA) proposed is fully described.

3.1. Standard Genetic Algorithm

Algorithm 1 describes the standard genetic algorithm (SGA) that is presented here for comparison purpose. The SGA was coded using the GA MATLAB Toolbox [14]. There is only one real coded population that is created and evaluated by procedures initialize() and evaluate(), respectively. These procedures will be detailed in the next section as they are the same for Algorithms 1 and 2. In lines 5-10, the new population is obtained from population in the last generation. In the actual application a linear ranking is used with a selective pressure of 2. The selectedParents() performs selection of individuals from the actual population, where the selection function used is the selected universal sampling.

Algorithm 1: Standard Genetic Algorithm
1. initialize(pop);
2. evaluate(pop);
3. repeat
4. matingPool←Ø;
5. for j←1 tonumberOfIndividuals do
6. (ind1,ind2). selectedParents(pop);
7. newInd←crossover(ind1,ind2);
8. newInd.mutation(newInd);
9. evaluate(newInd)
10. matingPool←matingPool∪newInd;
11.pop←matingPool;
12. until numberOfGenerations;

The newInd is created by crossover() using a uniform crossover where a discrete recombination between pairs of individuals is performed. In this operator, the parent who contributes to the offspring for each variable is chosen randomly with equal probability. Next, the population is changed by the mutation() with mutation rate of 0.7/Nvar (Nvar=3 is the number of PID parameters). The objective function is evaluated in line 9 for the actual offspring. Finally, the offspring replaces least fit parents where the rate of reinsertion is 90%. At this point the next generation starts. The stop criterion is defined by the number of generations. In this case it was chosen 60 generations.

3.2. Multi-Population Genetic Algorithm

Algorithm 2 describes the proposed multi-population genetic algorithm (MPGA), where several overlapping populations evolve until the convergence criterion has been reached. This algorithm is adapted from the MPGA presented in [20].

Algorithm 2: Multi-Population GeneticAlgorithm
1. repeat
2. for i←1 tonumberOfPopulations do
3. initialize(pop(i));
4. evaluate (pop(i));
5. structure(pop(i));
6. repeat
7. for j←1 to nCrossovers do
8. (ind1,ind2). selectedParents(pop(i));
9. newInd←crossover(ind1,ind2);10. if executedMutation then11. newInd←mutation(newInd);12. evaluate(newInd);
13. insert(newInd, pop(i));14. structure (pop(i));15. executeMigration(pop(i));16. until convergence(pop(i));17. until numberOfEvaluations;
As it happens for SGA, each populations is first generated by the procedure initialize() and next the individuals are evaluated by evaluate() procedure. For each population the individuals are structured in a binary tree with 4 clusters as illustrated by the Figure 3.

Figure 3. Population hierarchically structured in a binary tree.

The individuals are represented by nodes and the values depicted on them are their fitness in a minimization problem. The clusters are composed by 3 individuals disposed in 2 levels, where the best fitted is named cluster “leader”, and it is displayed in the upper level. The other 2 individuals (located in the lower level) are the supporters. This hierarchical population structure is performed by the procedure structure().

To better explain the method, we present in Figure 3 a population composed by 7 individuals hierarchically structured in binary trees. The best individual of the whole population is the root node (node with fitness 20).

Regarding the Algorithm 2, after the procedure structure(), it follows 2 nested loops. According to the inner loop, the genetic operators (crossover(), mutation()) are applied and a new hierarchical population is produced; the outer loop runs the inner loop until the convergence criterion is reached. The genetic operator crossover() takes two individuals randomly chosen from a cluster, where one of them is always the leader and the other is a supporter. Next, the new individuals created may be changed by mutation(). These operators create a number of new individuals defined by parameter (7).

\[ nCrossover = \text{crossoverRate} \times \text{populationSize} \quad (7) \]

The crossoverRate defines the amount of new individuals from the populationSize. The new individuals (newInd) are inserted (insert()) into the population, if better than the selected supporters (worst parent). After the nCrossovers be evaluated, the hierarchical structure is updated by structure(). The procedure rearranges individuals in each cluster so that the fittest individual will be the leader of its cluster.

The new individual replaces the supporter, but the cluster needs to be rearranged to keep the hierarchy.

The convergence is reached when no new individuals are inserted after nCrossovers attempts. At this point, a migration operator sends the best individual from this population to the next. Thus, a ring migration occurs once the best individual from population \( a-l \) is sent to population \( a \), and the best individual of the last population is sent to the first one.

After the convergence is reached, the next population will evolve following the same steps already described. However, the following executions of initialize() will keep the best individual and the migrated individual. This elitist re-initialization pretends preserving the best individual found so far as well as keeping the information that comes from the migrated individuals. The method stops when a defined number of evaluations are reached.

3.3. Individual, Fitness and Operators

This section gives details about the representation of individual, the determination of fitness value as well as the genetic operators used by the proposed MPGA. The representation of individual and the calculus of fitness are the same already used in SGA case. However, the genetic operators for the proposed MPGA are different from the ones used by the SGA once a random choice among three crossovers and three mutations is executed.

The individual is represented as a vector, \( \text{PID} = [k, t, t] \), whose PID parameters were already explained in section 2. The initialize() procedure (common to both algorithms) creates each individual randomly assigning values within the boundaries of each parameter. These boundaries are the maximum and minimum of each parameter within the learning sets used for training and test the MLPs [9], \( k \in [0.20 \quad 0.94], t \in [0.23 \quad 1.32] \) and, \( t \in [0.05 \quad 0.33] \).

The fitness of each individual is determined by its propagation through the corresponding multilayer perceptrons neural networks (MLPs). As explained in section 2, the MLPs approximate the mappings between the PID parameters and the criteria ITAE and ITAY. Therefore, the fitness value is determined by equation (8):

\[ \text{Fitness} = w_1 \text{MLP}^{\text{ITAE}} + w_2 \text{MLP}^{\text{ITAY}} \quad (8) \]

Parameters \( w_1 \) and \( w_2 \) represent weights used to define which control criterion we want to optimize. For example, if we want minimize the ITAE we set \( (w_1,w_2)=(1,0) \), but if we want minimize the weighted sum of ITAE and ITAY we use \( (w_1,w_2)=(0.5,0.5) \), as defined by expression (6).

Three crossovers for real-coded individuals were adapted for the proposed individual representation: arithmetic, geometrical and blend. In the arithmetic and geometric crossover [15], each one of the three...
parameters in the new individual is, respectively, an arithmetic mean and a geometric mean from the same parameter in the parents as presented by equations (9) and (10) with \( \text{param} \in \{k_i, t_i, t_d\} \).

\[
\text{newInd}_{\text{new}} = 0.5(\text{ind}1_{\text{new}} + \text{ind}2_{\text{new}}) \quad (9)
\]

\[
\text{newInd}_{\text{new}} = \sqrt{\text{ind}1_{\text{new}} \cdot \text{ind}2_{\text{new}}} \quad (10)
\]

The blend crossover (BLX) [5] restricts the new value within the hyper-rectangular defined by the respective values in the parents:

\[
\text{newInd}_{\text{new}} = \text{ind}1_{\text{new}} + \mathcal{U}(-\alpha,1) \cdot \left| \text{ind}1_{\text{new}} - \text{ind}2_{\text{new}} \right| \quad (11)
\]

where \( \alpha \in \{0.0, 0.5, 1.0\} \) was randomly selected at each blend crossover execution.

The mutation operators always randomly select one of the three parameters from the individual representation to be changed. Three types of mutation were defined:

- **Replace**: Take the selected parameter and replace its value by another randomly determined.
- **Limit**: Take the selected parameter and redefine its value as the minimum or maximum values in its domains. The domain limit assigned is randomly chosen.
- **Invert**: Take the selected parameter and change its value by the opposite within the domain: new\( \text{Value} = \text{max} + \text{min} - \text{oldValue} \) considering a domain interval as \([\text{min}, \text{max}]\).

During the crossover or mutation execution (lines 9-11 in Algorithm 2), one of the three crossovers and one of the three mutations previously described are randomly chosen to be performed.

### 4. Results

Our computational results are presented in two subsections. In section 4.1, a comparison between the performance of the SGA and MPGA will take place. Next, the PID solutions obtained with both algorithms are used to tune the control system. The SIMULINK model is then used to produce the corresponding outputs in section 4.2.

#### 4.1. Genetic Algorithm Performance

As explained in section 3.3, each component of the individual is randomly generated between the maximum and minimum values of each parameter domain. The population in SGA is represented by a matrix with dimensions 20\( \times \)3, where the individuals are disposed in the rows and the columns are their genes \( \{k_i, t_i, t_d\} \). For MPGA there are 2 populations with 7 individuals and 1 population with 6 individuals. These individuals are hierarchically structured in binary trees.

It was randomly generated a set of 1000 initial populations with 20 individuals for testing both genetic algorithms. The tests aim to evaluate the GAs when used for minimizing the control criteria ITAE, ITAY and ITAE-ITAY (expressions (4), (5) and (6)). The stop criterion is set as 1200 evaluations in Algorithms 1 and 2; this means that, in the case of SGA, 60 generations took place. The MPGA is coded in Java programming language and set with 3 populations, where the crossoverRate is set 1.0 and the mutation rate is set 0.9.

Table 1 presents the frequency distribution of the minimum fitness values returned. There are 16 classes of frequency distributions, where each class has 0.05 of amplitude. These classes are represented by their open upper limits (first column). The range of classes where no null frequencies occur for at least one GA is highlighted. Based on this table, we can see that in the first row the class \([1.15;1.20]\) is represented by 1.20 which is its open upper limit. MPGA and SGA found a total of 770 and 703 fitness minima, respectively, within this class and when optimizing the ITAY criterion.

What ITAE minimization is concerned, MPGA found 946 fitness minima while SGA found 766 minima within class \([1.65;1.70]\) (row 11, Table 1). The frequency of fitness minimum is concentrated in class \([1.45;1.50]\) when a weighted sum of both criteria (ITAE and ITAY) are minimized (equation(8) with \(w_1=w_2=0.5\)). This fact is supported by the results from row 7, Table 1 (873 for MPGA and 660 for SGA).

The total number of individuals evaluated until the minimum of fitness is reached is presented in Table 2. This table follows the same structure of Table 1, but the amplitude of classes is 100 individuals. Looking at the classes with highest frequencies for ITAE minimization, we realize that MPGA need evaluate a number of individuals within the interval \([1,000; 1,100]\) to reach a minimum, for 169 experiments, (row 11,Table 2). On the other hand, SGA reached 992 minima fitness spending a number of evaluations within the interval \([1,100; 1,200]\) (last row,Table 2).

The average and the standard deviation, for the minimum fitness and for the number of individuals evaluated until the minimum is reached, are shown in Table 3 and Table 4, respectively. In the last row of both tables, the result of the two-tailed Mann-Whitney U test [12] is also presented taking into account 1% of significance level; based on it, we will decide which GA is preferable. In the Table 3, the average fitness minimum is compared with the corresponding reference value.

The average values were obtained taking into account 1000 experiments performed by the genetic algorithms (GAs) for each criterion (ITAE, ITAY and ITAE-ITAY). The reference values are the minima obtained by the minimization gradient method implemented in the MATLAB minimization toolbox[13] as explained in section 2.

Considering the statistics for minimum fitness (Table 3) average values (AV) show that MPGA is always better than SGA and presents a smaller standard deviation (SD). According to the statistical test (U-Test),
we can conclude that MPGA is preferable, except for the minimization of ITAY, for which both methods are tied. Both genetic algorithms present also a small positive deviation from the reference values.

Table 1. Frequency distribution of the minimum fitness.

<table>
<thead>
<tr>
<th>Class</th>
<th>ITAE</th>
<th>ITAY</th>
<th>ITAE_ITAY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPGA</td>
<td>SGA</td>
<td>MPGA</td>
</tr>
<tr>
<td>1.20</td>
<td>0</td>
<td>770</td>
<td>0</td>
</tr>
<tr>
<td>1.25</td>
<td>0</td>
<td>203</td>
<td>0</td>
</tr>
<tr>
<td>1.30</td>
<td>0</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>1.35</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1.40</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1.45</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.50</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.55</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.60</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.65</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.70</td>
<td>946</td>
<td>766</td>
<td>0</td>
</tr>
<tr>
<td>1.75</td>
<td>50</td>
<td>169</td>
<td>0</td>
</tr>
<tr>
<td>1.80</td>
<td>5</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>1.85</td>
<td>5</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>1.95</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2. Frequency distribution of the minimum fitness.

<table>
<thead>
<tr>
<th>Class</th>
<th>ITAE</th>
<th>ITAY</th>
<th>ITAE_ITAY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPGA</td>
<td>SGA</td>
<td>MPGA</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>200</td>
<td>11</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>300</td>
<td>22</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>400</td>
<td>31</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td>500</td>
<td>57</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>600</td>
<td>64</td>
<td>0</td>
<td>58</td>
</tr>
<tr>
<td>700</td>
<td>83</td>
<td>0</td>
<td>66</td>
</tr>
<tr>
<td>800</td>
<td>128</td>
<td>0</td>
<td>94</td>
</tr>
<tr>
<td>900</td>
<td>119</td>
<td>0</td>
<td>107</td>
</tr>
<tr>
<td>1000</td>
<td>153</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>1100</td>
<td>169</td>
<td>8</td>
<td>191</td>
</tr>
<tr>
<td>1200</td>
<td>161</td>
<td>992</td>
<td>213</td>
</tr>
</tbody>
</table>

Table 3. Average and standard deviation of minimum fitness.

<table>
<thead>
<tr>
<th></th>
<th>ITAE</th>
<th>ITAY</th>
<th>ITAE_ITAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPGA</td>
<td>AV.</td>
<td>1.678</td>
<td>1.192</td>
</tr>
<tr>
<td>SD.</td>
<td>0.012</td>
<td>0.022</td>
<td>0.016</td>
</tr>
<tr>
<td>SGA</td>
<td>AV.</td>
<td>1.691</td>
<td>1.201</td>
</tr>
<tr>
<td>SD.</td>
<td>0.034</td>
<td>0.049</td>
<td>0.042</td>
</tr>
<tr>
<td>Reference value</td>
<td>1.658</td>
<td>1.150</td>
<td>1.429</td>
</tr>
<tr>
<td>U-TEST</td>
<td>MPGA</td>
<td>TIE</td>
<td>MPGA</td>
</tr>
</tbody>
</table>

Table 4: Average and standard deviation of number of evaluations.

<table>
<thead>
<tr>
<th></th>
<th>ITAE</th>
<th>ITAY</th>
<th>ITAE_ITAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPGA</td>
<td>AV.</td>
<td>838.9</td>
<td>886.5</td>
</tr>
<tr>
<td>SD.</td>
<td>255.0</td>
<td>247.9</td>
<td>267.5</td>
</tr>
<tr>
<td>SGA</td>
<td>AV.</td>
<td>1180.6</td>
<td>1179.8</td>
</tr>
<tr>
<td>SD.</td>
<td>20.3</td>
<td>20.0</td>
<td>19.0</td>
</tr>
<tr>
<td>ΔAV. (%)</td>
<td>40.7</td>
<td>33.1</td>
<td>40.9</td>
</tr>
</tbody>
</table>

In relation to the statistics presented in Table 4, when MPGA is used, the minima of the fitness is reached with a small number of individuals evaluation compared with the number spent when SGA is applied. For all the criteria, the number of evaluations needed to reach the minima using SGA is greater than 30% compared with the MPGA. Looking at the 2<sup>nd</sup> and 4<sup>th</sup> rows (Table 4), MPGA is more unstable as shown by a standard deviation higher than the one obtained by SGA. However, the statistical test indicates that MPGA outperforms SGA with a smaller number of evaluations for all criteria.

4.2. PID Controller Genetic Tuning

Taking the results from all the 1000 experiments, it is determined six PID vectors of solutions computing the average values \([k, t_i, t_d]\) of the minima obtained by each approach (SGA and MPGA), when optimizing each control criterion (ITAE, ITAY and ITAE_ITAY), when optimizing each control criterion (ITAE, ITAY and ITAE_ITAY). Looking at Table 5, these six PID vectors (3<sup>rd</sup> to 5<sup>th</sup> columns) correspond to the 1<sup>st</sup>, 2<sup>nd</sup>, 4<sup>th</sup>, 5<sup>th</sup>, and 7<sup>th</sup> 8<sup>th</sup> rows. All the criteria (Table 5, last column) were evaluated using the exact control system SIMULINK model instead of NN models used during the optimization procedures. Because of this, all the values in the last column (except the ones written in boldface) are different from the corresponding in the Table 3. E.g. the value of ITAE when the optimization was performed by SGA is 1.663 (Table 5) using the SIMULINK model. However, during the optimization procedure it was reached an average value of 1.691 (Table 3, 3<sup>rd</sup> row, 3<sup>rd</sup> column).

It is also presented the reference PID values evaluated by SIMULINK model using the minimization gradient method to find \([k, t_i, t_d]\) (Table 5, bold rows); these reference values for the criteria are the same already presented in Table 3. Looking at this column (last column, Table 5) we realise that reference values in boldface were almost reached by both GAs, for every criterion. The difference between the reference value and those obtained with SGA or MPGA is negligible. The consequence of this, no matter the PID controller is tuned with GAs or with the exact values (reference
values), the corresponding controlled system responses are very close.

**Table 5: PID solutions and corresponding criteria values.**

<table>
<thead>
<tr>
<th></th>
<th>$k_c$</th>
<th>$t_1$</th>
<th>$t_d$</th>
<th>criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITAE</td>
<td>SGA</td>
<td>0.749</td>
<td>0.937</td>
<td>0.447</td>
</tr>
<tr>
<td></td>
<td>MPG</td>
<td>0.754</td>
<td>0.943</td>
<td>0.442</td>
</tr>
<tr>
<td></td>
<td>Reference value</td>
<td>0.749</td>
<td>0.963</td>
<td>0.434</td>
</tr>
<tr>
<td>ITAY</td>
<td>SGA</td>
<td>0.667</td>
<td>0.898</td>
<td>0.487</td>
</tr>
<tr>
<td></td>
<td>MPG</td>
<td>0.671</td>
<td>0.905</td>
<td>0.481</td>
</tr>
<tr>
<td></td>
<td>Reference value</td>
<td>0.664</td>
<td>0.903</td>
<td>0.479</td>
</tr>
<tr>
<td>ITAE_ITAY</td>
<td>SGA</td>
<td>0.716</td>
<td>0.933</td>
<td>0.448</td>
</tr>
<tr>
<td></td>
<td>MPG</td>
<td>0.723</td>
<td>0.941</td>
<td>0.444</td>
</tr>
<tr>
<td></td>
<td>Reference value</td>
<td>0.714</td>
<td>0.964</td>
<td>0.424</td>
</tr>
</tbody>
</table>

Using the data from Table 5, we will illustrate this fact for 3 control criteria: ITAE, ITAY and ITAE_ITAY. The proximity of the responses when we optimise the ITAE is illustrated by using the PID parameter from 2nd and 3rd rows. Let’s define $y_1(t)$ and $y_2(t)$ the steps outputs vectors, when the tuning is performed by reference values (3rd row), and using MPG (2nd row), respectively. For a time simulation of 15s and defining a relative error norm measure by the expression (12), we obtain $E = 0.5\%$.

$$E = 100 \frac{\|y_2(t) - y_1(t)\|}{\|y_1(t)\|} \quad (12)$$

The curves of $y_1(t)$ and $y_2(t)$ are almost superimposed, so we decide only represent $y_1(t)$ in Figure 4.

Now, let’s compare the ITAY optimization using the SGA (4th row, Table 5) with the reference (6th row, Table 5). In this case $y_1(t)$ and $y_2(t)$ are the system output when an output disturbance is added, and the reference input is null ($R(s) = 0$, Figure 1). The responses $y_1(t)$ and $y_2(t)$ are produced when used the 6th and 4th rows PID parameters, respectively. Considering the previous error measure (12) we obtain $E = 1.2\%$. The magnitude of the error norm is not enough to distinguish both curves superimposed, so it was also decided to represent only the reference $y_1(t)$ in Figure 5.

Finally, let's considering the tuning when we optimize the weighted sum of both criteria (expression (8), $w_1 = w_2 = 0.5$). In this case we will compare the tuning obtained using the MPG with the tuning using the reference values (PID parameters displayed in the 8th and 9th rows).

The step output and the system output with null reference and a step disturbance added to output are represented in Figure 6, for the tuning performed with reference value (9th row, Table 5). Using the same error measure (12) with the definitions previously presented we obtain $E = 3\%$ and $E = 0.5\%$ for the disturbance rejection (ITAY) and reference following (ITAE), respectively.

**Figure 4. Step output, when the PID is tuned with reference parameters (3rd row, Table 5).**

**Figure 5. System output when the PID is tuned with reference parameters (6th row, Table 5).**

**Figure 6. PID tuned with reference parameters (9th row, Table 5). Top: Step output. Bottom: System output for null reference and a step disturbance added to the output.**

5. Conclusions

In this paper we compare and evaluate two approaches of genetic algorithms (GA): a standard genetic algorithm (SGA) and a multi-population genetic algorithm (MPGA). Both GAs, combined with artificial neural networks (NN), provide an efficient tool for solving the problem of automatic tuning of PID controllers for a time-invariant plant. The use of an optimizer like a GA together with a NN acting as a model, avoids the need of...
perturbing the control loop operation, which is a key factor considering a real-time application.

Also important for a real-time application is the accuracy of the models as well as the accuracy and efficiency of the GA. For the case under study, comparing the minima obtained using a gradient method with the minima obtained by GA, where the system is modelled by NN, we can conclude that NN generalize well enough and GA is an efficient optimizer for control tuning problems.

The MPGA outperforms the SGA in terms of solution quality as well as in the number of function evaluations. MPGA was able to find better solutions spending less number of function evaluations that is an important issue for a digital hardware implementation. Also the response obtained by the controlled system, when tuned with the average of the best values for PID parameters found by MPGA, presents a value very close to the reference for all criteria. In a real time situation, where time restrictions are present, the number of evaluation must be reduced. Based on Table 2, it is possible to conclude that with less than 1000 evaluations only the MPGA was able to return minimum values for all criteria.

Our current research is targeted in two main directions, both crucial for PID controller auto-tuning: modelling and optimization. As for the modelling procedures, our goal is to accommodate time-variant plants. In terms of optimization, a large set of control criteria should be taken into account when a robust control response is desirable; the extension for a multi-objective optimization will be natural. In this context, multi-populations can be used to optimize the conflicting objectives. Finally, the proposed system will be implemented in real time using digital hardware like FPGA.

Acknowledgments

João Lima wishes to acknowledge the Institute of Mathematic Sciences and Computation, São Carlos, of São Paulo University, Brazil, in particular to Prof. Dr. Eduardo Marques for receiving him during the sabbatical year. He also wishes to acknowledge the support for sabbatical year of ‘Fundação para a Ciência e Tecnologia’, SFRH/BSAB/1114/2011.

References