Self-organizing genetic algorithm based tuning of PID controllers

Zhang Jinhua*, Zhuang Jian, Du Haifeng, Wang Sun'an

School of Mechanical Engineering, Xi'an Jiaotong University, No. 28, Xianning West Road, Xi'an, Shaanxi Province 710049, People's Republic of China

Article info

Article history:
Received 14 July 2007
Received in revised form 17 November 2008
Accepted 19 November 2008

Keywords:
Genetic algorithm
Cyclic mutation
Dominant selection
Self-organizing
PID controller

Abstract

This paper proposes a self-organizing genetic algorithm (SOGA) with good global search properties and a high convergence speed. First, we introduce a new dominant selection operator that enhances the action of the dominant individuals, along with a cyclical mutation operator that periodically varies the mutation probability in accordance with evolution generation found in biological evolutionary processes. Next, the SOGA is constructed using the two operators mentioned above. The results of a nonlinear regression analysis demonstrate that the self-organizing genetic algorithm is able to avoid premature convergence with a higher convergence speed, and also indicate that it possesses self-organization properties. Finally, the new algorithm is used to optimize Proportional Integral Derivative (PID) controller parameters. Our simulation results indicate that a suitable set of PID parameters can be calculated by the proposed SOGA.

1. Introduction

Proportional Integral Derivative (PID) controllers have the advantage of simple structure, good stability, and high reliability. Accordingly, PID controllers are widely used to control system outputs, especially for systems with accurate mathematical models. The key issue for PID controllers is the accurate and efficient tuning of parameters. In practice, controlled systems usually have some features, such as nonlinearity, time-variability, and time delay, which make controller parameter tuning more complex. Moreover, in some cases, system parameters and even system structure can vary with time and environment. As a result, the traditional PID parameter tuning methods are not suitable for these difficult calculations. With the aid of Genetic Algorithms (GAs), Artificial Neural Networks and Fuzzy Logic, many researchers have recently proposed various alternative, intelligent PID controllers [2].

As a popular optimization algorithm, the GA had been widely used to turn PID parameters. Soltoggio [16] proposed an improved GA for tuning controllers in classical first and second order plants with actuator nonlinearities. Chen and Wang [3] used the population-based distribution GA to optimize a PID controller, and found that the search capability of the algorithm was improved by competition among distribution populations in order to reduce the search zone. The PID controllers based on GAs have good performance and have been applied in practice. However, the standard GAs have some disadvantages, such as premature convergence and low convergence speed. In this paper, a self-organizing Genetic Algorithm (SOGA) is proposed. This new algorithm improves the efficiency of global search and was found to avoid premature convergence. Finally, the SOGA is employed to optimize PID parameters.

The organization of this paper is as follows. In Section 2, the structure of the GA-PID controller is discussed. In Section 3, the dominant selection and the cyclic mutation operator are proposed. In Section 4, the self-organizing genetic algorithm is described and its performance is analyzed. Results are presented in Section 5 and conclusions in Section 6.
2. Structure of GA-PID controller

2.1. PID control law

A PID controller is a feedback controller that makes a plant less sensitive to changes in the surrounding environment as well as to small changes in the plant itself. The continuous form of a PID controller, with input $e$ and output $u$, is given by

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt}$$

where $u(t)$ is the controlled output, $K_p$ is the proportional gain, $K_i$ and $K_d$ stand for the integral and derivative gains, respectively.

The discrete velocity-type PID control law is described as

$$\Delta u(k) = K_p [e(k) - e(k - 1)] + K_i T_s e(k) + \frac{K_d}{T_s} [e(k) - 2e(k - 1) + e(k - 2)]$$

where $K_p$, $K_i$, and $K_d$ are PID turning parameters, and $T_s$ stands for the time constant.

2.2. Fitness function

GAs search for the optimal solution by maximizing a given fitness function, that is, an evaluation function that provides a measure of the quality of the solution to the problem. In the control process, the objective is to minimize the cost function, defined as the Integral Absolute Error (IAE), which determines the performance of any industrial process. At the same time, the square of the controller output is included to avoid exporting a large control value. Thus, the cost function is written as

$$F(t) = \int_0^\infty (\sigma_1 |e(t)| + \sigma_2 u^2(t) + \sigma_3 e y(t)) dt + \sigma_3 T_r$$

where $e(t)$ and $u(t)$ are used to represent the system error and the control output at time $t$, $T_r$ is the rising time, and $\omega_i$ ($i = 1, 2, 3$) are weight coefficients.

To avoid overshooting, a penalty value is adopted in the cost function. That is, once overshooting occurs, the value of overshooting is added to the cost function. Hence, Eq. (3) is rewritten as

$$F(t) = \int_0^\infty (\sigma_1 |e(t)| + \sigma_2 u^2(t) + \sigma_4 |e y(t)|) dt + \sigma_3 T_r$$

where, $\sigma_4$ is a coefficient and $\sigma_4 > \sigma_i$, $e y(t) = y(t) - y(t - 1)$, and $y(t)$ is the output of the controlled object. The minimization objective function is transformed to be a fitness function:

$$\text{Fitness} = 1/(F(t) + 1)$$

3. Genetic algorithm

A GA is an intelligent optimization technique that relies on the parallelism found in nature, in particular its searching procedures are based on the mechanics of natural selection and genetics. GAs were first conceived in the early 1970s by Holland [7]. GAs are used regularly to solve difficult search, optimization, and machine-learning problems that have previously resisted automated solutions [12]. They can be used to solve difficult problems quickly and reliably. These algorithms are easy to interface with existing simulations and models, and they are easy to hybridize. GAs include three major operators: selection, crossover, and mutation, in addition to four control parameters: population size, selection pressure, crossover and mutation rate. Population-based optimization methods are addressed in [10,13]. This paper is concerned primarily with the selection and mutation operators.

3.1. Selection

Selection is a genetic operator that chooses a chromosome (individual) from the current generation’s population to be included in the population of next generation. The selection operator is the algorithmic embodiment of evolutionary biology. Selection demonstrates the phenomenon of “survival of the fittest” and determines the evolutionary trajectory of the GA. Current selection operators include roulette, tournament, top percent, and best selection. The use of a selection operator can-
not produce a new schema, but can eliminate some less useful ones. This is the main reason that premature convergence occurs [20]. So, improving the selection operator is one of our tasks.

It is natural for the dominant individuals in the population, who possess high fitness, to have more resources and offspring. Low fitness individuals tend to die before they mature enough to pass on their genes. More than a century ago, an Italian economist, Vilfredo Pareto, investigating the shape of the Italian personal income distribution, found that 80% of the property in Italy was owned by 20% of the Italian population. As a result, he proposed the 80/20 rule, also known as the Pareto Principle. It is the famous power law distribution. This principle holds across many fields, such as physics, computer science, biology, demography and sociology [8]. In some of the literature related to GAs, researchers especially stress the dominant individuals, who have better fitness, in order to improve the performance of their genetic algorithm. Examples of algorithms following this logic are the optimum family genetic algorithm [9] and bee evolutionary genetic algorithm [11]. Therefore, a new selection operator, termed the dominant selection operator (DSO), is proposed in this paper. The DSO strengthens the action of dominant individuals and weakens the action of the inferior individuals in the process of evolution. The selection probability of individual $x$ can be defined as

$$P(x, t) = \frac{fu(x)}{u_{\text{max}}(t)}$$

(6)

where $u(x)$ is the fitness of the individual $x$, $u_{\text{max}}(t)$ stands for the fitness of the best individual in the generation $t$, and $\beta$ is a selection pressure turning coefficient.

3.2. Crossover

Crossover is a genetic operator that combines two chromosomes in order to produce a new chromosome possessing some characteristics of each parent. The key idea of crossover is that the child may be better than both of the parents if it takes the best characteristics from each of the parents. The general crossover operator includes one-point, two-point and uniform operation. Crossover occurs during evolution according to a user-definable crossover probability, which is usually modeled as a fixed value. Nevertheless, researchers have presented some online methods for tuning the probability of crossover. San José-Revuelta, for example, used the individuals’ fitness entropy to adjust the crossover probability [15]. This paper does not advance any new concept to the crossover operator.

3.3. Mutation

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new individuals being added to the population. With these new individuals, GAs may be able to arrive at a better solution. Mutation is an important part of the genetic search because it helps to prevent the population from stagnating at any local optimum. Mutation occurs during evolution according to a certain mutation probability. Although the mutation operator is the only mechanism for directly generating new schema, the mutation probability should generally be set to a fairly low value. If it is set too high, the search will turn into a primitive random search. The following are the general types of mutation: flip bit, boundary, non-uniform, uniform, and Gaussian.

Many formulas have been suggested for the calculation of mutation probability [17,20,21]. For example, Fogarty [4] proposed a relationship between the mutation probability and the generation number. The relationship is denoted as:

$$p_m(t) = \frac{1}{240} + \frac{0.11375}{2^t}$$

(7)

where $t$ is the current generation number. Hesser and Männer [6] provided a more general relationship:

$$p_m(t) = \sqrt{\frac{c_1}{c_2}} \exp(-c_3 t / 2)$$

(8)

where $l$ is the length of the chromosome, $n$ is the population size, and $c_i$ ($i = 1, 2, 3$) denote coefficients. The value of the coefficient $c_i$ is estimated based on the particular situation. Bäck and Schütz [1] suggested a more appropriate empirical relationship:

$$p_m(t) = \left(2 + \frac{l - 2}{T - 1} t\right)^{-1}$$

(9)

where $T$ is the max generation number, $t$ is the current generation number. The test results on some difficult combination optimization problems have shown that Eq. (9) had a good performance.

In the practical application of the preceding approaches, premature convergence occurs when the population is small. Fogel [5] proposed an adaptive method for controlling the mutation rate. However, this method requires a calculation of the affinity of the population, which increases the computational load and decreases the search speed.

In order to obtain a simple mutation rate control rule, we focused on biological evolution processes. Paleontologists [14] proved, using fossils, that five biological mass extinction events occurred during biological evolution on earth. Moreover,
these mass extinction events occurred at intervals of approximately 6200–6500 years. During each period, the species to become extinct included the dominant group (e.g. the Jurassic dinosaurs). From ancient species to human beings, evolutionary processes have indicated that a cyclical mutation process might exist; that is, that large-scale mutations might occur at certain times. As an analogy for the biological process, we have designed a cyclical mutation operator (CMO) where mutation probability varies periodically. The mutation rate function is defined as:

\[ p_m(t) = \frac{\alpha[t - (k + 0.5)T_c]^2}{T_c^2} \]  

(10)

where \( t \) is the current generation number, \( T_c \) indicates the mutation period, \( \alpha \) represents a mutation turning coefficient, and \( k \) is the number of the period.

4. Self-organizing genetic algorithm

4.1. Description of the algorithm

By introducing DSO and CMO, the self-organizing genetic algorithm is proposed. The SOGA presented here also implements an elitism strategy, where the elite are formed by selecting some individuals with better objective value from the population. Fig. 1 shows the schematic representation of the SOGA. In practice, the SOGA consists of the following steps.

Step (1): Initialize the parameters of the algorithm: the size of population \( N \), crossover probability \( p_c \) and the size of the dominant population \( N_s \);
Step (2): Generate original population;
Step (3): Evaluate fitness of individuals and reserve \( N_s \) dominant individuals. (This step corresponds to the “Elite Selection” operation in Fig. 1);
Step (3.1): Reserve the best individual in the population;
Step (3.1): Reserve \( N_s - 1 \) individuals who have the least parameter distance with the best individual where the parameter distance between individual \( i \) and the best individual is defined as:

\[ d^i = \sum_{k=0}^{p_n} (p_{ik}^i - p_{ik}^{\text{max}})^2 \]  

(11)

where \( p_n \) is the number of the parameter, \( p_{ik}^i \) indicates the \( k \)th parameter variable of individual \( i \), and \( p_{ik}^{\text{max}} \) indicates the \( k \)th parameter variable of the best individual.

Step (4): Set the iterative generation index \( I_g = 0 \);
Step (5): Judge whether the terminate rule is satisfied or not. If satisfied, then proceed to step (10). Otherwise continue onto step (6);
Step (6): Set the number of the offspring individual \( n_i = 0 \);
Step (7): Genetic operation. Judge whether \( n_i \geq N \). If \( n_i \geq N \), then proceed to step (8), otherwise do the following:
need a higher computational load in order to obtain good global search performance. SOGA demonstrates good global search performance. Nevertheless, the best of these other four approaches only converged in 15% of the time. So, as compared to these standards, and SOGA showed a slightly better convergence than the others. SOGA converged properly in 100% of case in function 4. Nevertheless, the SGA only occasionally found the optimum. In function 3, SGA could not find the optimum, while BSGA runs in function 3 and function 4. In functions 1 and 2, BSGA, DMGA, AGA and SOGA could find the optimum during nearly every run, though the SGA only occasionally found the optimum. In function 3, SGA could not find the optimum, while BSGA and SOGA showed a slightly better convergence than the others. SOGA converged properly in 100% of case in function 4. Nevertheless, the best of these other four approaches only converged in 15% of the time. So, as compared to these standards, SOGA demonstrates good global search performance.

The mean MCT of SOGA nearly equaled those of BSGA and AGA. Hence, compared with other algorithms, SOGA did not need a higher computational load in order to obtain good global search performance.

4.2. Performance analysis of SOGA

In order to analyze the performance of our algorithm, four commonly used test functions were selected. Performance was evaluated along two dimensions: solution quality and efficiency. Solution quality refers to the ability of an algorithm to find optimal solutions. Efficiency measures the speed and the success ratio of an algorithm in obtaining the optimal solution. These experiments were implemented in VC++ on a PC with PIV2.4G CPU and 512M RAM. The test functions are listed as follows.

Function 1:

\[ f_1: \max f(x, y) = 1 + x \sin(4\pi x) - y \sin(4\pi y + \pi) + \frac{\sin\left(6\sqrt{x^2 + y^2}\right)}{6\sqrt{x^2 + y^2 + 10^{-15}}}, \quad x, y \in [-1, 1] \quad (12) \]

Function 2:

\[ f_2: \max f(x, y) = 3(1 - x)^2 \exp[-x^2 - (y + 1)^2] - 10\left(\frac{x}{5} - x^3 - y^5\right) \exp[-x^2 - y^2] - \frac{1}{3} \exp[-(x + 1)^2 - y^2], \quad x, y \in [-3.3] \quad (13) \]

Function 3:

\[ f_3: \max f(x, y) = \sin(5.1\pi x + 0.5)^{30} \exp\left[-\frac{4\log_2(x - 0.0667)^2}{0.64}\right] \sin(5.1\pi y + 0.5)^{30} \exp\left[-\frac{4\log_2(y - 0.0667)^2}{0.64}\right], \quad x, y \in [0, 1] \quad (14) \]

Function 4:

\[ f_4: \max f(x, y) = -[20 + x^2 - 10 \cos(2\pi x) + y^2 - 10 \cos(2\pi y)], \quad x, y \in [-5.0, 5.0] \quad (15) \]

Fig. 2 shows maps of these test functions. Obviously, each test function is a multimodal function with many local optima. Therefore, they are good for testing the ability of a search algorithm to identify the global optimum.

To evaluate the algorithm search capability, the performance of SOGA is compared with a standard GA (SGA), an adaptive GA (AGA) [20], an improved GA as presented in [1] (abbreviated to BSGA), and a GA with an adaptive mutation rate discussed in [18] (abbreviated to DMGA), using the test functions described above. The parameters of each GA are shown in Table 1. The mutation probability column represents the initial value for the mutation probability. Experimental results are given in Table 2, averaged over 1000 runs.

In Table 2, the symbol ‘/’ indicates no statistical data in row 5 and 6, because SGA could not find the optimum for 1000 runs in function 3 and function 4. In functions 1 and 2, BSGA, DMGA, AGA and SOGA could find the optimum during nearly every run, though the SGA only occasionally found the optimum. In function 3, SGA could not find the optimum, while BSGA and SOGA showed a slightly better convergence than the others. SOGA converged properly in 100% of case in function 4. Nevertheless, the best of these other four approaches only converged in 15% of the time. So, as compared to these standards, SOGA demonstrates good global search performance.

The mean MCT of SOGA nearly equaled those of BSGA and AGA. Hence, compared with other algorithms, SOGA did not need a higher computational load in order to obtain good global search performance.
Table 3 shows the statistical results for evolution generation in finding the optimum solution across 1000 runs. It can be seen that the mean evolution generation that finds an optimal solution of SOGA is smaller than that of the others across all the test functions. Results indicated that the SOGA has good convergence speed as compared with other methods. In some cases, the MSE of the SOGA is larger than that of other methods; in function 2, the MSE of SGA is 2.13 and the MSE of SOGA is equal to 2.19; in function 3, the MSE values of DMGA and AGA are lower than that of SOGA. However, the NFO of SOGA is larger than others in those cases. Besides, it can be seen that the mean MSE value of SOGA is low, obviously. In essence, SOGA possessed stable convergence property.

Comparative results indicated that the new GA has good global search ability. During almost every run, the SOGA could find the optimum of the four complex test functions.

The plot in Fig. 3 shows the convergence of our algorithms (SOGA), AGA, BSGA and DMGA for function 4. Note that the self-organizing genetic algorithm (SOGA) converged much faster than the others. Though function 4 is a multimodal function with many peaks of nearly equal height, the SOGA could escape these local optima and converge to the global optimum within 250 generations. In all test cases, SOGA outperformed AGA, and SGA.

Table 1
Parameters of the GAs.

<table>
<thead>
<tr>
<th></th>
<th>Population size</th>
<th>Crossover probability</th>
<th>Mutation probability</th>
<th>Max number of evolution</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGA</td>
<td>50</td>
<td>0.7</td>
<td>0.01</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>BSGA</td>
<td>50</td>
<td>0.7</td>
<td>0.01</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>DMGA</td>
<td>50</td>
<td>0.7</td>
<td>0.5</td>
<td>1000</td>
<td>$\lambda = 1.1, \omega = 1.5$</td>
</tr>
<tr>
<td>AGA</td>
<td>50</td>
<td>0.7</td>
<td>0.01</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>SOGA</td>
<td>50</td>
<td>0.7</td>
<td>1.0</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

Table 2
The experimental results of four test functions. NFO represents the number of finding the optimum; MCT is the mean computational time.

<table>
<thead>
<tr>
<th></th>
<th>NFO</th>
<th>MCT (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGA</td>
<td>BSGA</td>
</tr>
<tr>
<td>$f_1$</td>
<td>53</td>
<td>1000</td>
</tr>
<tr>
<td>$f_2$</td>
<td>38</td>
<td>1000</td>
</tr>
<tr>
<td>$f_3$</td>
<td>0</td>
<td>998</td>
</tr>
<tr>
<td>$f_4$</td>
<td>0</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3
The statistical results of evolution generation. MEG represents the mean evolution generation finding optimum, and MSE is the mean square error of evolution generation.

<table>
<thead>
<tr>
<th></th>
<th>MEG</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SGA</td>
<td>BSGA</td>
</tr>
<tr>
<td>$f_1$</td>
<td>50.21</td>
<td>40.32</td>
</tr>
<tr>
<td>$f_2$</td>
<td>80.23</td>
<td>75.40</td>
</tr>
<tr>
<td>$f_3$</td>
<td>/</td>
<td>120.71</td>
</tr>
<tr>
<td>$f_4$</td>
<td>/</td>
<td>760.41</td>
</tr>
</tbody>
</table>
The plot in Fig. 4 shows the mutation probability evolution of formulas (7), (9) and (10), marked with M1, M2 and CM, respectively. As it shown, the mutation probability values of M1 and M2 become smaller during successive iteration. Hence, once the algorithm reaches a local optimum, it is prone getting trapped in the local optimum, which results in a premature convergence, especially when searching within a small population. However, the value of CM changes cyclically from 0 to 1, which helps to overcome premature convergence. These test results proved that SOGA has good global convergence.

In order to analyze the search path taken by the SOGA, the number of every individual selected as a parent $n_i (i = 1, \ldots, N)$ was recorded. All individuals in a generation were sorted by ranking and were tagged with a ranking number $t_i (i = 1, \ldots, N)$. All individuals in the initial population were marked as a parent. The population size for each generation was 50. Finally, all record data were processed in the form of a power law by performing a nonlinear regression analysis. The regression analysis function is expressed as:

![Fig. 3. Convergence for function $f_4$.](image1)

![Fig. 4. Evolution of mutation probability.](image2)
\[ y = ax^b \]  

After applying the regression analysis, the power law coefficients, \(a\) and \(b\), were determined to be 7249.2 and -2.6293, respectively. The coefficient of determination, which is represented by the symbol \(r^2\), was equal to 0.7969. Thus, taking logarithm of both \(n_i\) and \(t_i\), we found the logarithm distribution as shown in Fig. 5.

Fig. 5 shows that the relationship between each individual ranking and the number of the selection of individuals follows an exponent of the power law distribution. The power law relationship is a feature of self-organizing system [19]. Consequently, the new GA represents the self-organization phenomenon of complex systems under the dominant selection operator during evolution. This self-organization property makes the algorithm more efficient.

In addition, using the cyclic mutation operation, the SOGA had the following desired effects:

1. Over the long-term, the mutation probability tends to remain small. Here, the crossover operator plays a major role and the algorithm could exhaustively search within the solution space.
2. At the end of the evolution period, the mutation probability becomes larger and allows the GA to jump out of the local search.

Hence, the SOGA can perform online adjustments of its mutation probability.

4.3. Effects of the parameters in SOGA

SOGA has three main parameters: the selection pressure turning coefficient \(\beta\) in Eq. (6), the mutation turning coefficient \(\alpha\) and the mutation period \(T_c\) in Eq. (10). In order to analyze the effects of these three parameters, we used the SOGA to search the optimum of 20 test functions with variable parameter values. Moreover, every function was tested 1000 times when the value of this parameter was varied. Then, the mean values of NFO, MEG, MCT and MSE were calculated. In our experiments, the values of \(\beta\) were in the range of \([0.1, 1]\); the values of \(T_c\) were in the range of \([10, 100]\); and \(\alpha\) \(\in [0.4, 4]\). The statistical results are shown in Figs. 6–8. (Note that detailed calculation results and the test functions can be seen by visiting the website: http://www.wcsteam.com.cn/soga_test.htm.)

Fig. 6 shows the statistical calculation results with the variety of \(\beta\). The other parameters \(\alpha\), \(T_c\) the population size \(N\), and the crossover probability \(P_c\) were equal to 4, 50, 50, and 0.7, respectively. As shown in Fig. 6, the value of the NFO becomes smaller with the increase of \(\beta\). Moreover, when \(\beta = 0.2\), the maximum NFO is achieved and equals 995.8; the minimum is 980.65 where \(\beta = 1.0\). In addition, the values of MSE and MEG rise gradually; and the value of MCT declines. In conclusion, the value of \(\beta\) does not affect the search performance of SOGA greatly.

Fig. 7 represents the statistical results for the variety of \(T_c\) with \(\beta = 0.7\), \(\alpha = 4\), and \(N = 50\). We can see from Fig. 7 that the value of \(T_c\) had little effect on the values of MCT and MEG. In general, the error of MSE is in the range of \([-0.1, 0.1]\). The value of \(T_c\) had little influence on the value of NFO when \(T_c\) is larger than 30.

Fig. 8 shows the statistical search results for the variety of the parameter \(\alpha\). In this simulation, \(\beta = 0.7\), \(T_c = 50\) were used. As shown in Fig. 8, the value of NFO and MCT increased at the beginning, and reached a plateau when \(\alpha \geq \alpha_0\) (a threshold). Furthermore, the value of MSE became smaller and was stable when \(\alpha \geq 2.0\). It had no effect on the value of MEG. As a result, the value of \(\alpha\) had little effect on the performance of SOGA, when it changed within \([2.0, 4.0]\).

In summary, altering the values of these three turning parameters in the SOGA had little effect on its performance, especially within certain ranges. Thus the new algorithm was quite robust and had good global search performance.
5. Simulation results

To further test the applicability of the new SOGA algorithm in construction with its dominant selection operator and cyclic mutation operator, we examined the tuning of the optimal parameters for PID controllers for the following plants using both the SOGA method as well as the standard GA method.

**Fig. 6.** Statistical results for the selection pressure turning coefficient $\beta$.

**Fig. 7.** Statistical results for the mutation period $T_c$.

**Fig. 8.** Statistical results for the mutation turning coefficient $\alpha$. 

5. Simulation results

To further test the applicability of the new SOGA algorithm in construction with its dominant selection operator and cyclic mutation operator, we examined the tuning of the optimal parameters for PID controllers for the following plants using both the SOGA method as well as the standard GA method.
Table 4
Parameters of different PID controllers.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>SGA-PID</th>
<th>SOGA-PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_p$</td>
<td>11.702</td>
<td>19.390</td>
</tr>
<tr>
<td>$K_i$</td>
<td>4.162</td>
<td>4.119</td>
</tr>
<tr>
<td>$K_d$</td>
<td>6.499</td>
<td>5.151</td>
</tr>
</tbody>
</table>

Table 5
Parameters of different PID controllers for plant 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>PID</th>
<th>NCD-PID</th>
<th>SOGA-PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_p$</td>
<td>0.63</td>
<td>1.9193</td>
<td>2.98</td>
</tr>
<tr>
<td>$K_i$</td>
<td>0.0504</td>
<td>0.0978</td>
<td>0.096</td>
</tr>
<tr>
<td>$K_d$</td>
<td>1.9688</td>
<td>8.1678</td>
<td>12.7</td>
</tr>
</tbody>
</table>

Fig. 9. Step response curves of plant 1 with different controllers.

Fig. 10. Step response curves of plant 2 with different controllers.
Plant 1: \( G(s) = \frac{1.6}{s^2 + 2.584s + 1.6} \)  \( \quad (17) \)

Plant 2: \( G(s) = \frac{15}{50s^3 + 43s^2 + 3s + 1} \)  \( \quad (18) \)

Plant 2 is a nonlinear system with saturation \([-1, +1]\) and a rate limit \(\pm 0.8\). In these simulations, the following parameters were used for the SOGA: the maximum number of generations, \( N \) was 100; the size of population was 50; the size of the dominant population was 10; \( P_r = 0.6 \). The parameters of the SGA were: \( P_r = 0.6 \) and \( P_m = 0.01 \). The system sampling time was 0.05 second and the output was in the range of \([-10, 10]\). Other relevant system variables were \( K_p \in [0, 20] \), \( K_i \in [0, 20] \), and \( K_d \in [0, 10] \). The weight coefficients in the objective function were \( \omega_1 = 0.999 \), \( \omega_2 = 0.001 \), \( \omega_3 = 2.0 \), and \( \omega_4 = 50 \).

Finally, the optimal parameters of the PID controller for plant 1 are listed in Table 4 using both SGA and SOGA methods. The parameters of PID controller for Plant 2 are listed in Table 5 using the standard PID, the PID based on Nonlinear Control Design (NCD-PID), and the SOGA-PID.

Fig. 9\(^1\) shows the unit step response curves of the plant using both the standard GA method and the SOGA method, respectively. The blue curve represents the step response with the open-loop controller. The green curve is the step response using the SGA method and the red curve is the result of the SOGA method. Compared with the others, it can be observed that the PID controller tuned by SOGA shows the minimum settling time and minimum peak overshoot. Fig. 10 indicates the unit step response curves of Plant 2. Curve 1 is the stop response with the standard PID method. Curve 2 is the step response with NCD-PID and curve 3 is the result with SOGA-PID. Curve 3 has a small overshoot and a short response time.

6. Conclusions

We propose a novel self-organizing genetic algorithm. This SOGA included a dominant selection operator and a cyclical mutation operator. The performance test results showed that the improved GA was able to effectively avoid premature convergence, and also demonstrated good optimization performance. At the same time, the improved GA presents the self-organization phenomenon found in complex systems during evolution. Finally, we have successfully employed the SOGA optimization method for evolving PID controller parameters. These simulation results showed that the SOGA holds great promise for use in tandem with existing controllers to tune these parameters; it was faster and had a better response compared with SGA.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments and suggestions, which significantly improved the paper. This work was supported by the National Natural Science Foundation of China under Grant No. 50505034.

References


\(^1\) For interpretation of color in Fig. 9, the reader is referred to the web version of this article.