A Modified Genetic Algorithm for Neurocontrollers

Il-Kwon Jeong, Changkyu Choi, Jin-Ho Shin and Ju-Jang Lee
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusong-dong Yusong-gu Taejon 305-701 Korea
Fax: 82-42-869-3410 E-mail: jjlee@ee.kaist.ac.kr

ABSTRACT

Genetic algorithms are getting more popular nowadays because of their simplicity and robustness. Genetic algorithms are global search techniques for optimizations and many other problems. A feed-forward neural network that is widely used in control applications usually learns by back propagation algorithm(BP). However, when there exist certain constraints, BP cannot be applied. We apply a genetic algorithm to such a case. To improve hill-climbing capability and speed up the convergence, we propose a modified genetic algorithm(MGA). The validity and efficiency of the proposed algorithm, MGA are shown by various simulation examples of system identification and nonlinear system control such as cart-pole systems and robot manipulators.

1. Introduction

Control of a certain system can be represented as finding an appropriate input for the desired output response. For that purpose, we generally need a model describing the characteristics of the system, but it is difficult to get such a model when the system is unknown or time-varying. Adaptive control and variable structure control have been studied to overcome the uncertainties of the model. However, it is hard to apply them to an unknown complicated system, since both of them have some constraints.

Genetic algorithms(GA's) are used in various control system problems nowadays[2][7][12]. A genetic algorithm is a search method based on the natural selection and genetics while neural networks and fuzzy theory originate from human information processing and inference procedures[4][9]. The current main search methods assume the smooth search space and the existence of its derivative, and most of them are using the gradient following technique.

GA is different from conventional optimization methods in several ways. GA is a parallel and global search that searches multiple points so it is more likely to get the global solution. It makes no assumption on the search space so, it is simple and can be applied to various problems. In control area, it has been used in identification, adaptation and neural network controller. However, GA is inherently slow and not good at fine tuning of the solutions.

On the other hand, neural networks have advantages of learning and various input-output mapping capability. There are many paradigms for neural networks, and feed forward networks are frequently used for complex nonlinear systems modeling and control[1][3]. Back propagation(BP) learning algorithm for feed forward networks has the problems of local minima and parameter sensitivity [6]. GA can be a suitable learning algorithm for neural networks, since it does not have those problems. But, as mentioned earlier on, GA is slow and poor at fine tuning. Therefore, GA-BP, a hybrid GA merged with BP was proposed[10], but, was not desirable due to the constraint imposed by BP.

Simulated annealing(SA) is another important algorithm which is powerful in optimization and high order problems, but it is very slow[8]. An adaptive GA merged with SA(ASAGA) was proposed to improve SA[5]. This algorithm was designed to preserve the merits of both SA and GA, but has more computational burden than ordinary simple GA.

In this paper, we propose a modified genetic algorithm(MGA) which is designed to improve the speed of convergence to the solution and the hill-climbing capability without complicating the algorithm. The paper is organized as follows. In section 2, general feature of a simple genetic algorithm is briefly described. In section 3, a modified genetic algorithm(MGA) is presented. In section 4, various applications of MGA to control problems and simulation results are included.

2. A Simple Genetic Algorithm

GA is a search method based on the natural selection and genetics. GA is computationally simple yet powerful and it is not limited by assumptions about the search space. The most important goal of optimization should be improvement. Although GA cannot guarantee that the solution will converge to the optimum, it tries to find the optimum, that is, it works for the improvement. GA's are different from normal search procedures in four ways[4].

1. GA's work with a coding of the parameter set, not the parameters themselves.

2. GA's search from a population of points, not a single point.

3. GA's use objective function information, not derivatives or other auxiliary knowledge.

4. GA's use probabilistic transition rules, not deterministic rules.

A simple GA is really easy to use, yet powerful. GA searches for the solution by transforming the individuals in a population by genetic operators and determining the population for the next generation. Usually, each individual is encoded using binary number. In a simple GA, following three basic genetic operators are used.

Reproduction : Reproduction probability is proportional to the fitness value(objective function value) of a string(individual).

Crossover : Crossover needs mating of two individuals. The information of two randomly selected individuals is partly interchanged according to the crossover site. Crossover is applied to take valuable information from both parents, and it is applied with the crossover probability.

Mutation : This operator insures against a bit loss and can be a source of new bits. Since mutation is a random walk through the string space, it must be used sparingly.

There are three differences of GA from random search. First, the existence of the direction of search due to the selection probability. Second, the fact that the better strings make more offsprings and finally, being likely to be improved in average fitness after generations.

3. A Modified Genetic Algorithm

GA is a useful search algorithm because of its simplicity and robustness. However, it has three major limitations. First, the performance is degraded as the problem size grows. Secondly, premature convergence occurs when GA cannot find the optimal solution due to loss of some important character(genes) in strings. The reason is that GA heavily depends on crossover and the mutation probability is generally too small to move the search to other space. Lastly, GA lacks hill-climbing capability. The reason is also that the mutation probability is much smaller than the crossover probability. There has been much work to prevent premature convergence for small populations: using the rank of the fitness values so that the selectivity is not proportional to the fitness value, scaling the fitness value according to the gene loss, changing the genetic operator, constraining mating (incest prevention), lowering the fitness values of the similar strings, adjusting mutation probability or inserting new genes, using parallel GA when the population is large, merging GA with another method, etc.,

To prevent premature convergence and to improve hill-climbing capability, we suggest a modified algorithm. It consists of fitness modification and the modified mutation probability.

Fitness value for a certain string is determined by the following rule.

$$fitness' = \begin{cases} k \times fitness_{avg}, \text{ if } fitness \geq k \times fitness_{avg} \\ fitness, \text{ other case} \end{cases} \quad (1)$$

where *fitness* is the original fitness value and *fitness'* is the modified value. $fitness_{avg}$ is the average of *fitness* values and k is a positive constant greater than 1. It can be thought that this simple rule represents just a fitness lowering procedure. But, this rule is based on another philosophy. Sudden emergence of a super string, by which we mean a string that has large fitness, may cause the premature convergence. We need to make the search rather slower to scan wider range of the search space without causing the premature convergence. (1) provides such scheme. It makes the algorithm consider the strings that have fitness values above certain threshold as the same ones, so the diversity of the population is increased. As the population evolves, the average fitness becomes larger according to the schema theorem[4]. This makes the algorithm search toward the optimum gradually.

The modified mutation probability, p_m is given as

$$p_m(i+1) = \begin{cases} p_{m0}, \text{ if the fittest is the same for } N_{reset} \text{ generation} \\ p_m(i), \text{ if } p_m(i) \times k_1 \leq p_{m\_low} \\ p_m(i) \times k_1, \text{ other case} \end{cases} \quad (2)$$

where i is the generation number. p_{m0} , p_{m_low} and k_1 is a positive constant less than 1. N_{reset} is a constant.

Thus, the adaptive mutation probability can be enlarged whenever needed. The enlarged mutation probability increases the diversity of the population so it prevents the population from premature convergence. This acts as a source for lost genes and makes the algorithm adaptive. When the mutation probability is equal to its lowest value p_{m_low} , it operates as a normal mutation operator. We apply the elitist strategy to maintain the solution of good quality under the condition of randomized search. Though several new parameters were introduced into MGA, determining them is not difficult. Based on our experience, following ranges are recommended: $p_{m0} \in [0.5,1]$, $p_{m\_low} \in [0,0.1]$, $k \in [1,10]$, $k_1 \in [0.8,1]$, $N_{reset} \in [1,100]$.

In this paper, we apply MGA to various control problems. As mentioned earlier, GA is suitable for a learning method of neural networks when conventional method cannot work. To apply MGA to neural networks, we should be able to code the information of a neural network. Since we only deal with the optimization of weights here, all the information of a neural network can be represented as an array of weight values. Because binary coding needs too many bits for a string, though we use it in the section 4.1 for a system identification problem, neural network representation in this paper uses the string that consists of real values with constant bound. Mutation is defined as adding a small random number to a real number in a string.

4. Applications to Control Problems

4.1 System parameter identification

The problem considered here is the same as those in [5][7]. It is presented for comparison purpose. The object system is a discrete time system:

$$A(q^{-1})y(t) = B(q^{-1})u(t-d) \qquad (3)$$

The objective is identifying $A(q^{-1})$, $B(q^{-1})$ and delay d using the given input $u(t)$ and the output $y(t)$. We define the error sequence as

$$\eta(t) = y(t) - \hat{y}(t) \qquad (4)$$

with

$$\hat{A}(q^{-1})\hat{y}(t) = \hat{B}(q^{-1})u(t-\hat{d}) \qquad (5)$$

The fitness function to be maximized is

$$F(t) = 1/\sum_{i=0}^{w}(\eta(t-i))^2 \qquad (6)$$

where w represents window size.

The system polynomials, poles and zeros in the reparameterized plane are the following:

$$\begin{aligned} A(q^{-1}) &= 1.0 - 1.5q^{-1} + 0.7q^{-2} \\ B(q^{-1}) &= b_0(1.0 + 0.5q^{-1} + 0.0q^{-2}) \end{aligned} \qquad (7)$$

$$[p_1, p_2] = [0.75,\ -0.37] \quad [z_1, z_2] = [-0.25,\ 0.25] \quad (8)$$

where b_0 is 1 and the delay d was set to 1.

We apply a simple GA, ASAGA(adaptive simulated annealing genetic algorithm)[5] and MGA to identify p_1, p_2, z_1, z_2, b_0 and d. b_0 is assumed to be in [0, 2] and the poles and zeros in [-1, 1]. In this problem, we use binary encoding. 7 bits were used for each parameter except for d (2 bits), so the resolution is slightly smaller than 0.02. A string consists of 37 bits. We use p_c = 0.8, p_{m0} = 1.0, p_{m_low} = 0.01, k = 2.5, k_1 = 0.9, population size = 100 and window size w = 30. Input for the sample data is

$$u(t) = \sin(t) - \sin(t/2.5) + random(-1 \sim 1) \qquad (9)$$

Fig. 1 shows the identification result of the poles with simple GA and Fig. 2 with ASAGA. The true value of p_2 is -0.371. However, the limitation on the resolution due to coding makes p_2 equal to -0.375. Fig. 3 shows the result with MGA. It shows the better hill-climbing and optimum finding capability than simple GA.

Though ASAGA shows the best performance among the three methods, MGA shows good performance using much smaller computation compared to ASAGA[5].

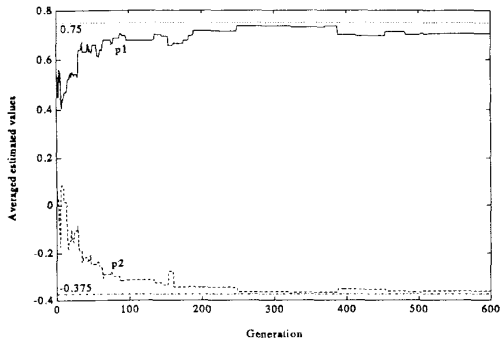

Fig. 1: Identification of the poles using simple GA

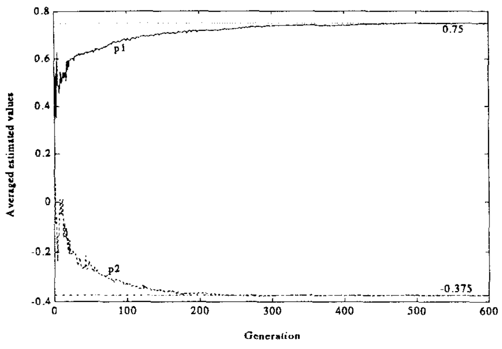

Fig. 2: Identification of the poles using ASAGA

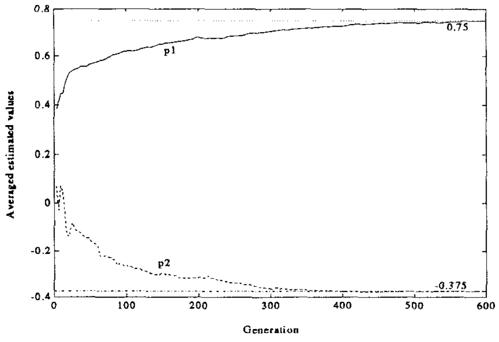

Fig. 3: Identification of the poles using MGA

4.2 Cart-pole problems

Pole balancing problem is difficult since the system is nonlinear. The problem becomes harder when we do not have any a priori information about the cart-pole system. Two cases of this kind of problems are considered in this section. The first is the balancing of a pole on a cart(single pole problem) and the second is the balancing of two poles on a cart(double pole problem). The double pole problem is more difficult than the single pole problem. Fig. 4 and Fig. 5 show the single pole and the double pole systems respectively. The simulation procedure is similar to that of [11] except that we use MGA instead of the evolutionary programming method.

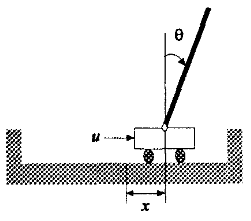

Fig. 4: Single pole system

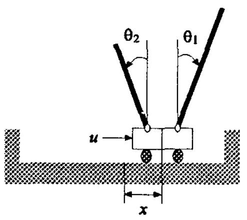

Fig. 5: Double pole system

The equations of motions for the single pole problem can be found in many papers. Those are

$$\ddot{\theta}=\frac{(M+m)g\sin\theta-\cos\theta[u+ml\dot{\theta}^2\sin\theta]}{(4/3)(M+m)l-ml(\cos\theta)^2} \quad (10)$$

$$\ddot{x}=\frac{u+ml[\dot{\theta}^2\sin\theta-\ddot{\theta}\cos\theta]}{M+m} \quad (11)$$

where M is the mass of the cart and m is the mass of the pole. l is the half of the pole length and u is the control force.

The equations of motions for the double pole system can be derived from dynamic equilibrium equations. It is well known for the double pole system that the poles must be of different length in order to be balanced. The equations are given as

$$\ddot{x}=\frac{u+\frac{3}{4}[m_1\sin\theta_1(l_1\dot{\theta}_1^2-g\cos\theta_1)-m_2\sin\theta_2(l_2\dot{\theta}_2^2-g\cos\theta_2)]}{M+m_1(1-\frac{3}{4}\cos^2\theta_1)+m_2(1-\frac{3}{4}\cos^2\theta_2)} \quad (12)$$

$$\dot{\theta}_1=[\tfrac{3}{4}(g\sin\theta_1-\ddot{x}\cos\theta_1)]/l_1 \quad (13)$$

$$\dot{\theta}_2=[\tfrac{3}{4}(g\sin\theta_2+\ddot{x}\cos\theta_2)]/l_2 \quad (14)$$

where the following notation was used.

l_1	Half length of long pole
l_2	Half length of short pole
m_1	Mass of long pole
m_2	Mass of short pole
x	Cart position
$\dot{x}$	Cart velocity
θ_1	Angle of long pole
θ_2	Angle of short pole
$\dot{\theta}_1$	Angular velocity of long pole
$\dot{\theta}_2$	Angular velocity of short pole

The control objective is to balance the system such that the poles do not fall beyond a predefined vertical angle(15°) and cart remains within the bounds of the horizontal track(± 2.4 meters from the center) using the neural network output as the control force to the cart. Where state variables are used as the neural network input. The equations of motions for both the single pole and the double pole problems are simulated using a Euler integration method with a step size of 0.01s. A population of neural networks is randomly initialized with each set of weights and biases. The simulation parameters are M = 1.0 *kg*, m = m_1 = 0.1 *kg*, m_2 = 0.01 *kg*, l = l_1 = 0.5 *m* and l_2 = 0.05 *m*.

Feed forward neural networks with four and six nodes in the input layer(corresponding to the states), ten nodes in the hidden layer and a single node in the output layer is used. The bias nodes in the input and hidden layers are set to 1.0. The activation function of the nodes is given by

$$f(x)=-1+2/(1+e^{-x}) \quad (15)$$

The output of the neural network was scaled so that it can continuously vary between -10*N* and +10*N*.

The coding for the weights and biases of the neural network is a vector of real numbers between -10.000 and 10.000. Because the system is assumed to be unknown a priori, a conventional method like BP can not be applied. A population of 100 networks was used. MGA parameters are as following: p_c = 0.8, p_{m0} = 0.5, p_{m_low} = 0.03, N_{reset} = 5, k = 2.5 and k_1 = 0.9.

The fitness measure for a network is the simulated time until failure occurs. No other information was used. MGA was able to discover a good controller that was successful for more than 100,000 time steps in about 200 and about 650 generations for the single-pole and double-pole respectively.

Fig. 6 shows the control result of the single pole system using the neurocontroller found by MGA. Initial condition for θ is 0.1 *rad* and all other states are zeros. Fig. 7 shows the result of the double pole system. θ_1 is initially 2.5° and all other states are zeros.

4.3 Robot manipulator control

The nonlinear system considered here is a one-link and a two-link robot manipulator. Fig. 8 shows the two-link manipulator system. A neural network is used as a nonlinear PD-type controller for the robot system. The control objective here is making the manipulator follow the desired joint trajectory.

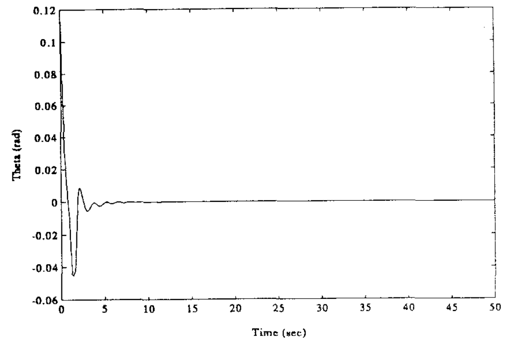

Fig. 6: Results of the single pole system

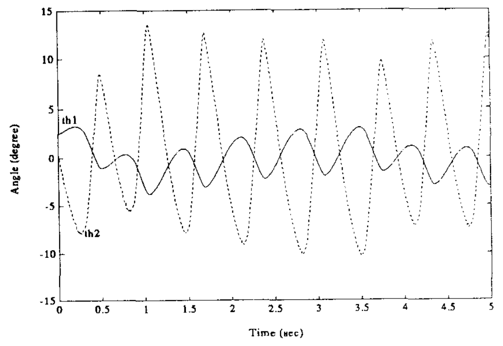

Fig. 7: Results of the double pole system

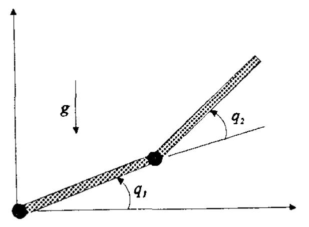

Fig. 8: Two-link manipulator

The equation of motion for the one-link manipulator is given by

$$\tau = \frac{ml^2}{3}\ddot{q} + \frac{mgl}{2}\cos q \qquad (16)$$

where m is the mass of the link, l is the length of the link, τ is the torque, g is the gravitational acceleration and q is the joint angle. The simulation parameters are $m = 10\ kg,\ l = 1.0\ m,\ g = 9.8\ m/\sec^2$.

The equations of motions for the two-link manipulator is given by

$$\begin{aligned}\tau_1 =& [m_1 l_{g1}^2 + I_1 + m_2(l_1^2 + l_{g2}^2 + 2l_1 l_{g2} C_2 + I_2]\ddot{q}_1 + [m_2(l_{g2}^2 + l_1 l_{g2} C_2) + I_2]\ddot{q}_2 \\ &- m_2 l_1 l_{g2} S_2(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2) + m_1 g l_{g1} C_1 + m_2 g(l_1 C_1 + l_{g2} C_{12})\end{aligned} \qquad (17)$$

$$\tau_2 = [m_2(l_{g2}^2 + l_1 l_{g2} C_2) + I_2]\ddot{q}_1 + (m_2 l_{g2}^2 + I_2)\ddot{q}_2 + m_2 l_1 l_{g2} S_2 \dot{q}_1^2 + m_2 g l_{g2} C_{12} \qquad (18)$$

where the following notations were used.

q_i	Joint angle of joint i
m_i	Mass of link i
I_i	Moment of inertia of link i
l_i	Length of link i
$l_{gi}(= l_i / 2)$	Distance between joint i and the center of mass of link i

$$S_i = \sin q_i\ ,\ C_i = \cos q_i\ ,\ C_{ij} = \cos(q_i + q_j)$$

The simulation parameters are m_1 = 10 kg, m_2 = 5 kg, $l_1 = l_2$ = 1.0 m, I_1 = 10/12 kgm^2, I_2 = 5/12 kgm^2, g = 9.8 m/sec^2.

The desired joint trajectory is chosen to satisfy zero initial and final velocity. It is given as

$$\begin{aligned}q_d(t) = q_{d1}(t) &= 6t^5 - 15t^4 + 10t^3 - \pi/2,\ t \in [0,1] \\ q_{d2}(t) &= 6t^5 - 15t^4 + 10t^3, \qquad t \in [0,1]\end{aligned} \qquad (19)$$

Neural network controller receives the error and the derivative of error between the desired angle and the actual angle, and the output of the neural network is used as the torque applied to the joint. It consists of input, hidden and output layers, and the hidden layer consists of four hidden neurons. The only information that MGA uses are the error and the derivative of error. MGA trains the neural network controller by maximizing the following fitness function.

$$fitness = 1 / \int_{t=0}^{1} (\sum_i error_i^2 + \dot{error}_i^2)dt \qquad (20)$$

MGA parameters are the same as the previous section. All initial errors are set to 0. Fig. 9 shows the result for the one-link manipulator. Fig. 10 and Fig. 11 show the results for the two-link manipulator. As shown in the results, MGA found a good neural network controller for these highly nonlinear robot manipulators.

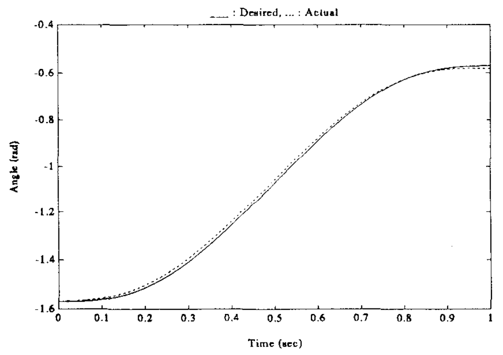

Fig. 9: Joint angle (1 link robot)

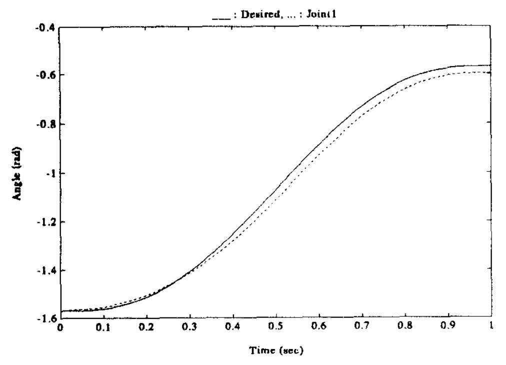

Fig. 10: 1st joint angle (2 link robot)

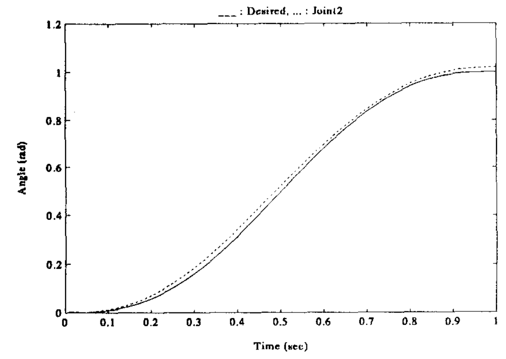

Fig. 11: 2nd joint angle (2 link robot)

5. Conclusion

We have proposed a modified genetic algorithm (MGA) which was designed to prevent the premature convergence and to speed up the convergence to the solution. The results of the various examples studied here indicate that MGA can be used as a good trainer for neural networks controlling nonlinear unstable plants using small information and no a priori knowledge. MGA has adaptive characteristics and improved hill-climbing capability compared to the simple GA. The control problems considered in this paper is hard to be solved by BP because desired input/output pairs are unknown for unknown systems. Further work includes simultaneous search for the weights and the architecture of a neurocontroller for those nonlinear systems.

References

[1] A. G. Barto, R. S. Sutton and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 834-846, Sep., 1983.

[2] C. L. Karr and E. J. Gentry, "Fuzzy control of pH using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 1, no. 1, pp. 46-53, Feb., 1993.

[3] C. W. Anderson, "Learning to control an inverted pendulum using neural networks," *IEEE Control Syst. Mag.*, pp. 31-37, Apr., 1989.

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[5] I. K. Jeong and J. J. Lee, "Genetic algorithms and neural networks for identification and control," *Proceedings of the First Asian Control Conference*, pp. 697-700, 1994.

[6] J. Hertz, A. Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley, 1991.

[7] K. Kristinsson and G. A. Dumont, "System identification and control using genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 5, pp. 1033-1046, Sep., 1992.

[8] L. Davis, *Genetic Algorithms and Simulated Annealing*. Reading, MA: Pitman Publishing, 1987

[9] L. Davis, *Handbook of Genetic Algorithms*. Reading, MA: Van Nostrand Reinhold, 1991.

[10] M. McInerney and A. P. Dhawan, "Use of genetic algorithms with backpropagation in training of feed-forward neural nwtworks," *IEEE International Conference on Neural Networks*, pp. 203-208, 1993.

[11] N. Saravanan and D. B. Fogel, "Evolving neurocontrollers using evolutionary programming," *IEEE International Conference on Evolutionary Computing*, pp. 759-763, 1994.

[12] Y. Ichikawa and T. Sawa, "Neural network application for direct feedback controllers," *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 224-231, Mar., 1992.