'95 KACC (1995. 10. 23 ~ 25)

Application of Self Organizing Genetic Algorithm

Il-Kwon Jeong 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: jik@odyssey.kaist.ac.kr

Abstract In this paper we describe a new method for multimodal function optimization using genetic algorithms(GAs). We propose adaptation rules for GA parameters such as population size, crossover probability and mutation probability. In the self organizing genetic algorithm(SOGA), SOGA parameters change according to the adaptation rules. Thus, we do not have to set the parameters manually. We discuss about SOGA and those of other approaches for adapting operator probabilities in GAs. The validity of the proposed algorithm will be verified in a simulation example of system identification.
Keywords Genetic algorithm, System identification, Self organizing

1. Introduction

Genetic algorithms(GAs) are robust search and optimization techniques which are based on the natural selection and genetics. GAs are applied in a number of practical problems nowadays. GAs are different from conventional optimization methods in several ways. GA is a parallel and global search method which searches multiple points so it is more likely to get the global optimum. It makes no assumption on the search space so, it can be easily applied to various problems. In control area, it has been applied to identification [1], adaptation [2] and neural network controller [3][6][7]. However GAs are inherently slow and not good at fine tuning of the solutions.

The GA may be thought as an evolutionary process where a population of solutions evolves over a sequence of generations. During each generation, the fitness(goodness) of each solution is calculated, and solutions are selected for reproduction based on their fitness. The probability of survival of a solution is proportional to its fitness value. This process is based on the principle of 'Survival of the fittest'. The reproducted solutions then undergo recombination which consists of crossover and mutation. We should note that genetic representation may differ from the real form of the parameters of the solutions. Fixed-length and binary encoded strings have been widely used for representing solutions since they provide the maximum number of schemata and as they are simple to implement [4][5].

In this paper we describe a self organizing genetic algorithm(SOGA) for multimodal function optimization. The choice of the crossover probability, p_c and the mutation probability, p_m is known to critically affect the behavior and performance of the GA. Though a number of generalized guidelines exist in the literature for choosing p_c and p_m, these guidelines are inadequate as the choice of optimal p_c and p_m becomes specific to the problem under consideration. The size of a population is another important parameter that affects the performance of the algorithm. In our algorithm, p_c, p_m and the size of the population are determined adaptively by the GA itself to realize the twin goals of maintaining diversity in the population and sustaining the convergence capacity of the GA.

The paper is organized as follows. In section 2, general feature of a simple genetic algorithm is briefly described. Section 3 describes our algorithm, SOGA. In section 4, we present simulation results to compare the performance of SOGA with that of a simple GA. The conclusions are presented in section 5.

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 diffrent from normal search procedures in four ways.

1. GAs work with a coding of the parameter set, not the parameters themselves.
2. GAs search from a population of points, not a single point.
3. GAs use objective function information, not derivatives or other auxiliary knowledge.
4. GAs use probabilistic transition rules, not deterministic rules.

Following the model of evolution, GA establish a population of individuals, where each individual cor-

responds to a point in the search space. An objective function is applied to each individual to evaluate their fitness. Using genetic operators, a next generation is formed based upon the survival of the fittest. Therefore, the evolution of individuals from generation to generation tends to result in fitter individuals, solutions, in the search space. Empirical studies have shown that genetic algorithms do converge on global optima for many problems including NP-hard ones.

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 informations 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 over generations.

3. Self Organizing Genetic Algorithm

In optimizing unimodal functions, it is important that the GA should be able to converge to the optimum in as few generations as possible. For multimidal functions, there is a need to be able to locate the region in which the global optimum exists, and then converge to the optimum. GAs possess poor hill-climbing capabilities and, they are vulnerable to getting stuck at a local optimum especially when the populations are small. The significance of p_c and p_m in controlling GA performance has long been acknowledged in GA research. The higher the value of p_c, the quicker are the new solutions introduced into the population. As p_c increases, however, solutions can be disrupted faster than selection can exploit them. Large value of p_m transform the GA into a purely random search, while some mutation is required to prevent the premature convergence of the GA to suboptimal solutions. The population size also affects the GA performance. Premature covergence may occur when the population size is small, while a large population size makes the algorithm slow. Usually, the choice of p_c, p_m and population size is left to the user to be determined statically prior to the execution of the GA.

To overcome the above-stated problem of difficulty in choosing the GA parameters, we suggest the following expressions which are main components of SOGA.

$$p_c = k_1(f_{max} - f')/(f_{max} - f_{min}) + k_2 \qquad (1)$$

$$k_1 + k_2 \leq 1$$

$$p_m = k_3(f_{max} - f)/(f_{max} - f_{min}) + k_4 \qquad (2)$$

$$k_3 + k_4 \leq 1$$

where f_{max} is the maximum fitness value and f_{min} is the minimum fitness value. f' is the larger of the fitness values of the solutions to be crossed. k_3 and the population size, N_{pop} are changed adaptively using the following procedure.

1. Initialize k_3.
2. $i \leftarrow i + 1$, generation i.
3. If the fittest is the same for n_{reset} generations, then $N_{pop} \leftarrow N_{pop} + n_1$ and go to step 1.
4. $N_{pop} \leftarrow N_{pop} - n_2$, $k_3 \leftarrow c * k_3$, go to step 2.

where n_1 and n_2 are positive constants of integers, and $c < 1$. As we can see from (1), p_c is a linear function of f' which varies from $k_1 + k_2$ to k_2 as f' changes from f_{min} to f_{max}. p_m varies in a similar fashion. Thus, the higher the fitness of a solution, the lower the probability of crossover or mutation of the solution. Therefore, we are able to preserve 'good' solutions of the population while the low fitness solutions prevent the GA from getting stuck at a local optimum. Note that k_3 is designed to decrease exponentially over generations. After few generations k_3 vanishes to 0 from its initial value and the mutation operator becomes to behave like a normal one. But, when the fittest is the same for n_{reset} generations, that is, the GA is getting stuck at a local optimum, p_m is enlarged to its initial value to move the search to the global optimum and, the population size N_{pop} is increased to search wider region of the search space. When the algorithm is in its normal operation state, N_{pop} is decreased at every generations to speed up the algorithm.

There have been similar works to improve the GA. [6] incorporates simulated annealing technique into the GA. [7] uses a fitness modification technique and an adaptive mutation probability. A local improvement operator is introduced in [8]. [9] proposes adaptive probabilities of crossover and mutation. However, its adaptive rules are not as general as ours and, the adaptation of the population size is not considered.

4. Simulation

The problem considered here is the same as those in [1][6]. It is presented for comparison purpose. The object system is a discrete time system:

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

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

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

with

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

The fitness function to be maximized is

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

where w represents window size. The system polynomials, poles and zeros in the reparameterized plane [1] are the following:

$$A(q^{-1}) = 1.0 - 1.5q^{-1} + 0.7q^{-2} \tag{7}$$

$$B(q^{-1}) = b_0(1.0 + 0.5q^{-1} + 0.0q^{-2}) \tag{8}$$

$$[p_1, p_2] = [0.75, -0.37],\ [z_1, z_2] = [-0.25, 0.25] \tag{9}$$

where b_0 is 1 and the delay d was set to 1. We apply a simple GA and SOGA 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 $[-1, 1]$. 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 used $p_c = 0.8$, $p_m = 0.01$, $N_{pop} = 100$ and $w = 30$ for the simple GA. We used $k_1 = k_2 = 0.5$, initial $k_3 = 0.9$, $k_4 = 0.01$, initial $N_{pop} = 50$, $n_{reset} = 5$, $n_1 = 6$, $n_2 = 2$, $c = 0.9$ and $w = 30$ for SOGA. Input for the sample data is

$$u(t) = \sin(t) - \sin(t/2.5) + random(-1 \sim 1) \tag{10}$$

We show the input and output used for the sample in Fig. 1. One simulation was done using 200 samples with 3 generations per one sample, that is, 600 generations. 10 simulations were done for each algorithm. Fig. 2 $\sim$ 4 show the average of the identification results of the poles with simple GA. The true value of p_2 is -0.371. But, the limitation on the resolution due to binary coding makes p_2 equal to -0.375. Fig. 5 $\sim$ 7 show the average of the results with SOGA. It shows the better hill-climbing and optimum finding capability than simple GA. The average of the population sizes of 10 simulations was found to be 43.4, which is much smaller than that of the simple GA though the performance of SOGA is much better than that of the simple GA.

5. Conclusion

We have proposed a self organizing genetic algorithm(SOGA) which was designed to prevent the premature convergence and to sustain the convergence capacity of the GA. SOGA determines p_c, p_m and N_{pop} automatically using its adaptive rule so, we do not have to determine the values for the parameter prior to the execution of GA. Simulation results indicate that SOGA has adaptive characteristics and improved hill-climbing capability compared to the simple GA. Using adaptive population size, execution time of the algorithm is significantly lowered. Furthur work includes the theoretical analysis of SOGA.

References

[1] 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.

[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] 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.

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

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

[6] 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.

[7] I. K. Jeong and J. J. Lee, "A modified genetic algorithm for neurocontrollers," *IEEE International Conference on Evolutionary Computing*, to appear, 1995.

[8] J. A. Miller, W. D. Potter, R. V. gandham and C. N. Lapena, "An evaluation of local improvement operators for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 5, pp. 1340-1351, Sep., 1993.

[9] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 656-667, April, 1994.

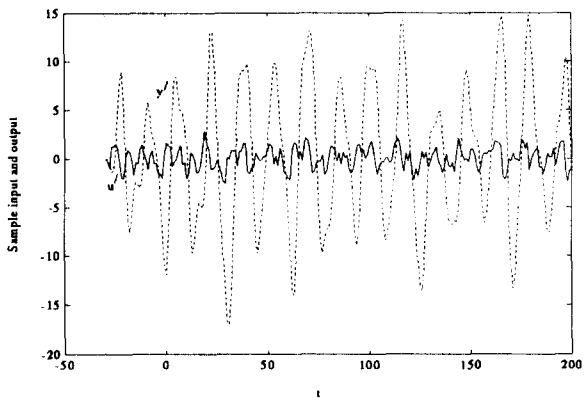

Fig. 1. The sample input and output

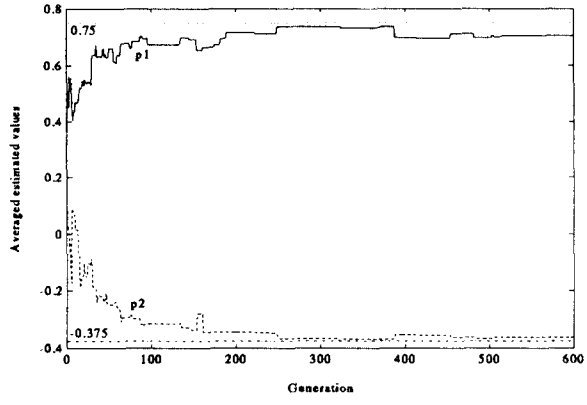

Fig. 2. Identification of the poles using a simple GA

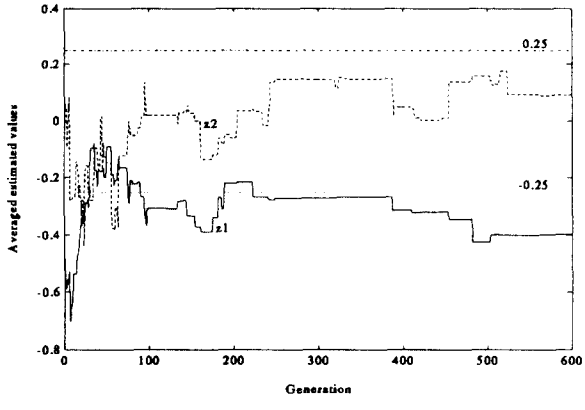

Fig. 3. Identification of the zeros using a simple GA

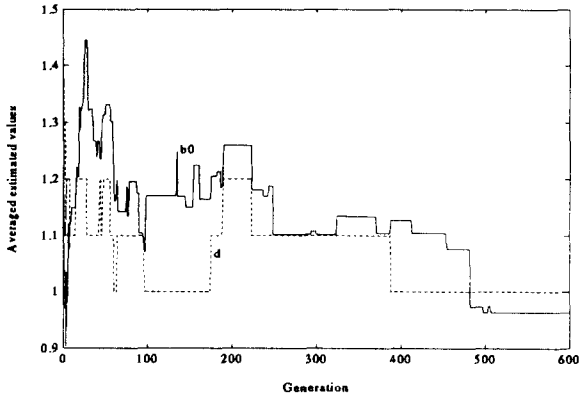

Fig. 4. Identification of b_0 and d using a simple GA

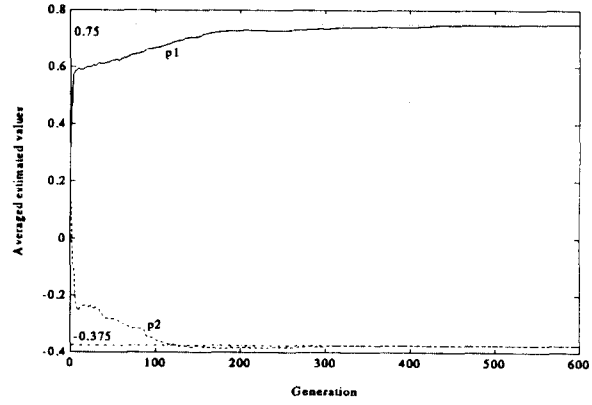

Fig. 5. Identification of the poles using SOGA

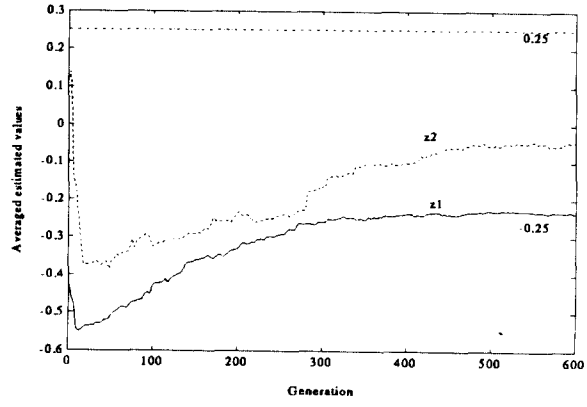

Fig. 6. Identification of the zeros using SOGA

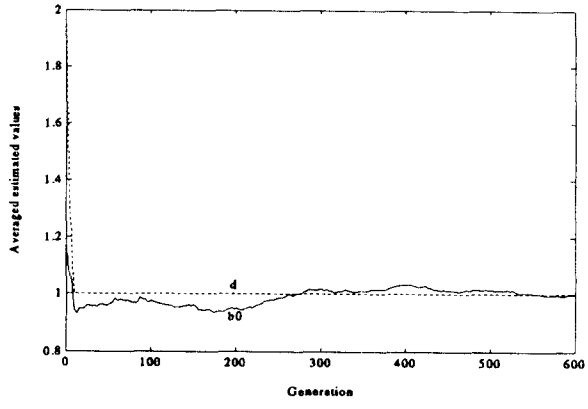

Fig. 7. Identification of b_0 and d using SOGA