Pergamon

Engng Applic. Artif. Intell. Vol. 9, No. 5, pp. 523–532, 1996

PII: S0952-1976(96)00049-8
0952-1976/96 $15.00+0.00

Contributed Paper

Adaptive Simulated Annealing Genetic Algorithm for System Identification

IL-KWON JEONG
Korea Advanced Institute of Science and Technology, Korea

JU-JANG LEE
Korea Advanced Institute of Science and Technology, Korea

(Received August 1994; in revised form March 1995; received for publication June 1996)

Genetic algorithms and simulated annealing are leading methods of search and optimization. This paper proposes an efficient hybrid genetic algorithm named ASAGA (Adaptive Simulated Annealing Genetic Algorithm). Genetic algorithms are global search techniques for optimization. However, they are poor at hill-climbing. Simulated annealing has the ability of probabilistic hill-climbing. Therefore, the two techniques are combined here to produce an adaptive algorithm that has the merits of both genetic algorithms and simulated annealing, by introducing a mutation operator like simulated annealing and an adaptive cooling schedule. The validity and the efficiency of the proposed algorithm are shown by an example involving system identification.

Keywords: Genetic algorithm, simulated annealing, system identification.

1. INTRODUCTION

Genetic algorithms (GAs) are search methods based on natural selection and genetics, while neural networks and fuzzy theory originate from human information processing and inference procedures.[1,2] GAs are currently being used in various problems, including control problems. In the control area, the GA has been used in identification, adaptation and neural-network controllers.[3–5]

Calculus-based search methods usually assume a smooth search space, and most of them use the gradient-following technique. A GA is different from conventional optimization methods in several ways. The GA is a parallel and global search technique that searches multiple points, so it is more likely to obtain a global solution. It makes no assumption about the search space, so it is simple and can be applied to various problems. However, GAs are inherently slow, and are not good at fine-tuning solutions.

Correspondence should be sent to: Il-Kwon Jeong,Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, 373-1 Kusong-dong, Yusong-gu, Taejon 305-701, Korea. E-mail: jik@odyssey.kaist.ac.kr.

Simulated annealing (SA) is another important algorithm which is powerful in optimization and high-order problems, but is very slow.[6] SA uses random processes to help guide the form of its search for minimal energy states. A GA merged with SA has been proposed to improve SA.[7] This algorithm uses simulated-annealing crossover (SAR) and simulated-annealing mutation (SAM) operators instead of standard ones. It has the merit that populations can be kept small. However, it has three drawbacks. SAR is far from the standard crossover which is a main component of a GA. One would have to redesign a cooling schedule for the algorithm at each time of execution. Because the algorithm has a strong resemblance to SA, it has the same drawback of slowness as SA.

In the analysis and design of control systems, it is necessary to have a mathematical model of the given plant. Such a mathematical model, here called simply a "model", must describe the system dynamics as completely as possible. There are two approaches to obtaining a mathematical model of a plant. One approach is to obtain a model based on physical laws; the other is to use experimentation. In general, it may not be possible to obtain an accurate model by applying physical laws

only. Some of the plant parameters must be determined by experiment. Constructing mathematical models and estimating optimal parameter values by experimental means is called "system identification".

This paper proposes an adaptive simulated annealing genetic algorithm (ASAGA), which is designed to improve the speed of convergence to the solution, to be implemented easily, and to be robust to the search space. The paper is organized as follows. In Section 2, genetic algorithms and simulated annealing are described. In Section 3, the proposed algorithm, named ASAGA, is presented. In Section 4, an application of ASAGA to a problem of system identification is included, with simulation results, to show the performance of the proposed algorithm.

2. GENETIC ALGORITHMS AND SIMULATED ANNEALING

2.1. Genetic algorithms

A GA is a search method based on natural selection and genetics. The central theme of the research on GAs has been the robustness, and the balance between the efficiency and the efficacy necessary for survival in many different environments. GAs are computationally simple, yet powerful, and are not limited by assumptions about the search space.

Current search methods can be classified into three groups: calculus-based, enumerative and random method. All of these methods lack robustness.[1] On the other hand, SA searches for the minimal energy state and uses random processes. The important thing to recognize is that a randomized search does not necessarily imply a directionless search. If more-human-like optimization tools are needed, the most important goal of optimization should be improvement. Although a GA cannot guarantee that the solution will converge to the optimum, it tries to find the optimum, that is, it works towards an improvement. GAs are different 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.

The GA may be thought of 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 on the basis of their fitness. The probability of survival of a solution is proportional to its fitness value. This process is based on the principal of "survival of the fittest". The reproduced solutions then undergo recombination, which consists of crossover and mutation. A 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 are simple to implement.[1,2]

A simple GA is really easy to use, yet powerful. It uses three basic genetic operators: reproduction, crossover and mutation. Reproduction is a process in which individual strings (solutions) are copied according to their fitness values (objective function values). Crossover requires a mating of two randomly selected strings. The information on the strings is partly interchanged according to a randomly chosen crossover site. Crossover is applied to take valuable information from the parents, and it is applied with a certain probability. Mutation is the occasional random alteration of the value of a string position. Mutation insures against 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 between GA and random searches. First, there exists a direction of the search, due to the selection probability. Second, the better strings generate more offspring. Finally, it is likely to be improved in average fitness after some generations.

2.2. Simulated annealing

Simulated annealing (SA) is a stochastic computational technique derived from statistical mechanics for finding near-global minimum-cost solutions to large optimization problems.[6] One should understand the basics of statistical mechanics to appreciate the relationship between techniques in statistical physics and large optimization problems. Statistical mechanics is the study of the behavior of very large systems of interacting components, such as atoms in a fluid, in thermal equilibrium at a finite temperature.

Suppose that a configuration of a system is identified with the set of spatial positions of the components. If a system is in thermal equilibrium at temperature T, then the probability $\pi_T(s)$ that the system is in a given configuration s depends on the energy $E(s)$, and follows the Boltzmann distribution

$$\pi_T(s) = \frac{\exp\left[\frac{-E(s)}{kT}\right]}{\sum_{w \in S} \exp\left[\frac{-E(w)}{kT}\right]} \tag{1}$$

where k is Boltzmann's constant and S is the set of all possible configurations.

For the simulation of the behavior of a system in thermal equilibrium at temperature T, suppose that at time i the system is in configuration q. A candidate r for the configuration at time $i + 1$ is generated randomly and

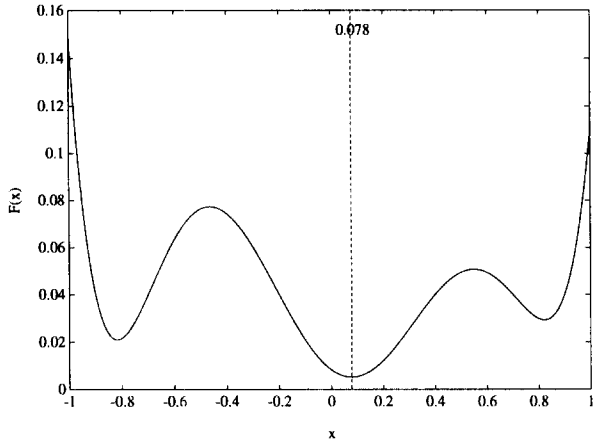

Fig. 1. The objective function $F(x)$.

accepted according to

$$p = \frac{\pi_T(r)}{\pi_T(q)} = \exp\left[\frac{-(E(r) - E(q))}{kT}\right]. \quad (2)$$

If p is greater than 1, that is, the energy of r is strictly less than the energy of q, then configuration r is accepted as the new configuration for time $i + 1$. If p is less than or equal to 1, then configuration r is accepted as the new configuration, with the probability p. Thus, configurations of higher-energy states may be attained. It is proved that as time i goes to infinity, the probability that the system is in a given configuration s equals $\pi_T(s)$, and the distribution of configurations converges to the Boltzmann distribution. Thus, at low temperatures the low-energy states predominate because of the nature of the Boltzmann distribution.

It is not sufficient to simply lower the temperature to achieve low-energy configurations. One must use an annealing process, where the temperature of the system is elevated, and then gradually lowered, spending enough time at each temperature to reach thermal equilibrium. The following preparatory steps are needed to apply the simulation of annealing to optimization problems.

1. One has to identify the analogues of the physics concepts in the optimization problem itself. Note that the energy function becomes the objective function, and the configurations of particles becomes the configurations of parameter values. Finding a low-energy configuration becomes finding a near-optimal solution, and temperature becomes the control parameter for the simulation.
2. One has to select a cooling schedule (annealing schedule), consisting of a set of decreasing temperatures, together with the amount of time to spend at each temperature.
3. One should have a way of generating and selecting new solutions.

The cooling process is inherently slow. Researchers have determined a cooling schedule that is sufficient for convergence. Specifically, for a given sequence of temperatures $\{T_i\}$ that satisfies equation (3) for a large constant c, the probability that the system is in configuration s as $i \to \infty$ is equal to $\pi_0(s)$, which means that the system is in the minimum-energy configuration with probability 1. In practice, however, it is often unnecessary to adhere to this conservative schedule in order to achieve acceptable results

$$T_i \to 0 \text{ as } i \to \infty \text{ and } T_i \geq \frac{c}{\log i}. \quad (3)$$

2.3. Comparison of a GA with SA using an example

Consider the following example.

Example: Find the minimum of the objective function $F(x)$, when $-1 < x < 1$

$$F(x) = (x + 0.9)(x + 0.7)(x + 0.2)(x - 0.4) \times (x - 0.7)(x - 0.9) + 0.04. \quad (4)$$

It is known by calculation that the true solution is about 0.078. Figure 1 shows the objective function. $F(x)$ has two local minima as well as the global minimum in the specified region of x.

For the simulation of GA and SA, the following settings were used.

- *GA*: x is coded as an 8-bit binary number and scaled to produce a value between -1 and 1, and the simple GA is used. The population consists of eight strings, crossover probability p_c is 0.8 and mutation probability p_m is 0.01.
- *SA*: equation (2) is used as the probability of acceptance of a new solution. A cooling schedule is

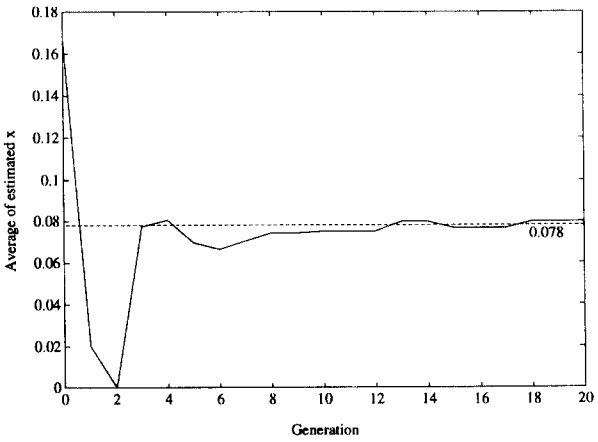

Fig. 2. GA, the average result of ten simulations of estimating x minimizing $F(x)$.

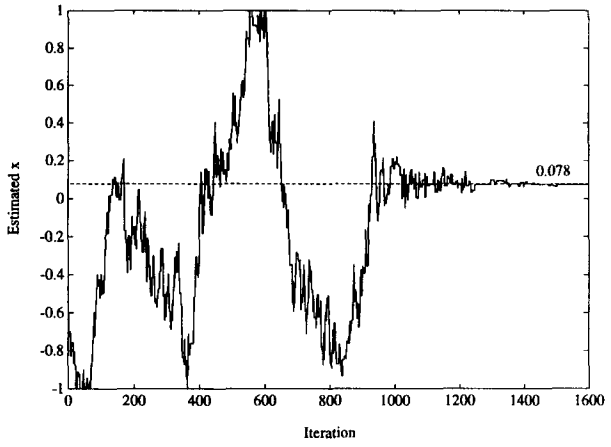

Fig. 3. SA, estimated value of x minimizing $F(x)$.

selected to satisfy equation (3). In the simulation, equations (5) and (6) are used as the probability of acceptance and the cooling schedule respectively

$$p(i) = \exp[-\Delta E / T(i)] \tag{5}$$

$$T(i) = \frac{T_0}{\log i} \tag{6}$$

where $i = 1, 2, 3, \ldots$ and ΔE is the change in the objective function value. $p(i)$ is considered to be 1 when i is equal to 1.

Figure 2 shows the average of the estimated solutions from ten independent executions of the GA, and Fig. 3 shows the estimated solution using SA. Figure 2 indicates that the GA finds the solution quickly, but it does not always determine the true solution. On the other hand, one can see the drifting of the solution and slow convergence to the global minimum in Fig. 3. Because the example here is rather simple, the GA could get a near-minimum solution in the initial population, which consists of randomly chosen strings.

3. ADAPTIVE SIMULATED ANNEALING GENETIC ALGORITHM

3.1. Limitations of the GA

The GA is a very useful algorithm because of its versatility. However, it has three major limitations. First, the performance is degraded as the problem size grows. As in the case of the simple example in Section 2.3, the GA outperforms SA. But, as the problem size grows, the GA requires a larger population to obtain a satisfactory solution. This situation occurs, for example, when the GA is used for the optimization of the weights or the structure of a large neural network. Secondly, premature convergence occurs when the GA cannot find the optimal solution due to loss of some important characters (genes) in the strings. The reason is that the GA depends heavily on crossover, and the mutation probability is generally too small to move the search into another space. To overcome this problem, a large population can be used so that it includes many characters. However, this is not desirable because the computational burden is increased and the speed of convergence is slow. There has been much work done 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 similar strings, adjusting the mutation probability or inserting new genes, using a parallel GA when the population is large, merging the GA with another method, etc. Another limitation of the GA is that it lacks a hill-climbing capability. The reason is also that the probability of mutation is much smaller than the probability of crossover.

3.2. Previous work on hybrid techniques using SA

SA can be applied to a high-order problem, since it has a stochastic hill-climbing capability and the solution state cannot stay at a fixed point for a long time. Therefore, there has been much work on hybridizing the GA with SA for performance improvement. Sirag and Weisser[8] proposed the thermodynamic genetic operator which uses $\exp(-\theta_m/T)$ as the mutation probability. There is a method in which the evolution of the operator probabilities is based on an assignment of credit from previous generations. Certain work uses SA-flavored Gaussian mutations, $\Delta s = N(0, \sigma)$. SAGA consists of a GA stage and an SA stage.[9]

Recently, Adler[7] proposed a new hybridization of GA with SA. In that algorithm, the hybridization of GA with SA was designed so that each algorithm maintains its own identity. The algorithm uses simulated-annealing crossover (SAR) and simulated-annealing mutation (SAM). It has some drawbacks. First, SAR is far from the standard crossover which is a main component of a GA. Second, one has to redesign a cooling schedule for the algorithm at each time of execution. Finally, because it is focused on SA rather than GA, it is like a parallel SA, and slow due to the cooling schedule.

The next section proposes a new hybrid GA which uses a new mutation operator like simulated annealing and an adaptive cooling schedule. Thus, repeated cooling schedules become possible.

3.3. Proposed hybrid genetic algorithm

A new hybridization of a GA with SA is proposed: the adaptive simulated annealing genetic algorithm (ASAGA). In merging a GA with SA, it is difficult to maintain each algorithm's own identity perfectly. When Adler[7] tried to solve this problem, the resulting algorithm converged slowly. Therefore, a simple and effective

Table 1. Proposed hybrid genetic algorithm

ASAGA (Adaptive Simulated Annealing Genetic Algorithm)	
Step 1:	$i = 1$, Initialization of population, Determine T_0.
Step 2:	If the generation number is enough or stopping criterion is satisfied then go to step 11 or else go to step 3.
Step 3:	Calculation of fitness values of each string.
Step 4:	Probabilistic determination of the number of offspring proportional to the fitness value.
Step 5:	Reproduction and formation of the new generation.
Step 6:	Random mating. If the string is the fittest then make no change to it or else apply crossover operator with the crossover probability p_c.
Step 7:	Generate a random string, and if a better solution is acquired then accept the string and go to step 9.
Step 8:	Accept the string generated in step 7 with the probability of $i^{(1/f-1/f')/T_0}$ where f' is the new fitness and f is the old fitness values.
Step 9:	If the fittest is the same for N_{reset} generations then $i = 1$ and reset T_0, else $i = i + 1$.
Step 10:	Go to step 2.
Step 11:	The fittest is the solution. Algorithm ends.

hybrid algorithm is proposed here to improve the performance of the GA. To preserve the merits of the GA no change is made to the crossover operator. The proposed algorithm, named ASAGA, includes the merits of SA by changing the mutation operator. This is basically different from only changing the operator probability. The new mutation operates like simulated annealing as follows. It generates a random string, which it accepts if the string is better than the original string. Otherwise, the string is accepted according to the probability given in equation (5). For the cooling schedule which is related to the speed of ASAGA, equation (6) is used again. SA searches for the minimal energy state while the GA searches for the string with the maximum fitness value. Therefore, equation (5) can be rewritten as

$$p(i) = \exp\left[-\left(\frac{1}{f'} - \frac{1}{f}\right)/T(i)\right] \tag{7}$$

where $i = 1, 2, 3, \ldots$, f is the fitness value of the original string and f' is the fitness value of the randomly generated string. i is increased by 1 at each generation. Again, $p(i)$ is considered to be 1 when i is equal to 1. From equations (6) and (7)

$$p(i) = \exp\left[-\frac{\left(\frac{1}{f'} - \frac{1}{f}\right) \cdot \log i}{T_0}\right] = i^{\left(\frac{1}{f} - \frac{1}{f'}\right)/T_0}. \tag{8}$$

Determining T_0 is generally difficult because it depends on the problem being solved. In a simulation of ASAGA, T_0 is defined as the reciprocal of the fitness value of the worst string in the population corresponding to $i = 1$.

To insure both high search speed and good solution quality, an adaptive rule is suggested

$$\textbf{Adaptive rule: } \text{reset } i \text{ to 1 when } Flag_{\text{reset}} = True \tag{9}$$

$$Flag_{\text{reset}} = \begin{cases} True, \text{ if the fittest solution is} \\ \quad \text{the same for } N_{\text{reset}} \text{ generations} \\ False, \text{ otherwise} \end{cases} \tag{10}$$

where N_{reset} is a positive integer constant. $Flag_{\text{reset}}$ is true when the algorithm is stuck in a certain region of the search space, that is, the algorithm may get stuck at a local minimum, and this means that certain characters for the solution are required. If the cooling schedule is started again by equation (9), then the probability of accepting inferior solutions again becomes large. Thus, the algorithm may have a chance to escape the local minimum and search for the global minimum. Therefore, equations (9) and (10) reduce the probability of failing to find the global solution. The elitist strategy, that is, always transferring the best solution to the next generation without any alterations, is applied in order to maintain a good quality solution under the condition of randomized search due to the repeated cooling schedules.

The proposed algorithm is shown in Table 1. The modified mutation operator offers the GA a hill-climbing capability. From the viewpoint of GA, the modified mutation operator functions in the same way as the standard one did. From the viewpoint of SA, it can be thought that SA deals with multiple points which are presented through the selection process of the GA, and this is desirable for SA which has to search the neighborhood of current solutions. Strings with high fitness make more offspring which have to be applied to SA, so SA operates near the good solution. This is also good for the reproduction and crossover of the GA. The proposed algorithm behaves like SA when the population size is small, and like a GA when the population size is large.

4. SYSTEM IDENTIFICATION

Identification means the determination of the model of a dynamic system from input/output measurements. The knowledge of the model is necessary for the design and implementation of a high-performance control system. System identification is an experimental approach for determining the dynamic model of a system. It includes four steps:[10]

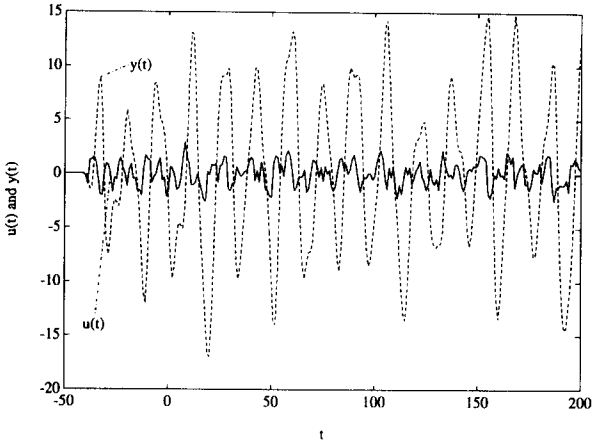

Fig. 4. Input and output for the sample.

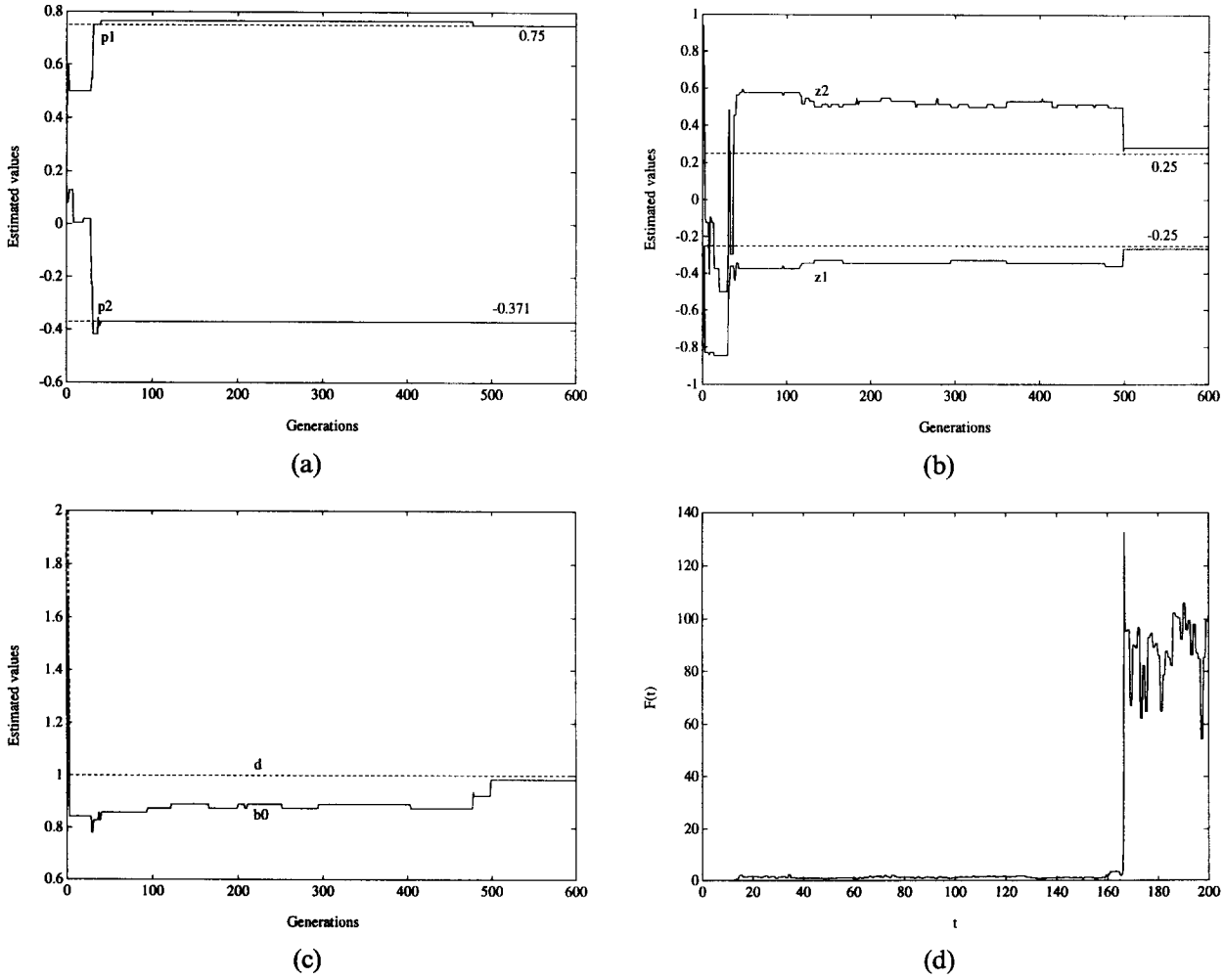

Fig. 5. Simple GA, the best estimation result among 20 simulations. (a) p_1, p_2; (b) z_1, z_2; (c) b_0, d; (d) $F(t)$.

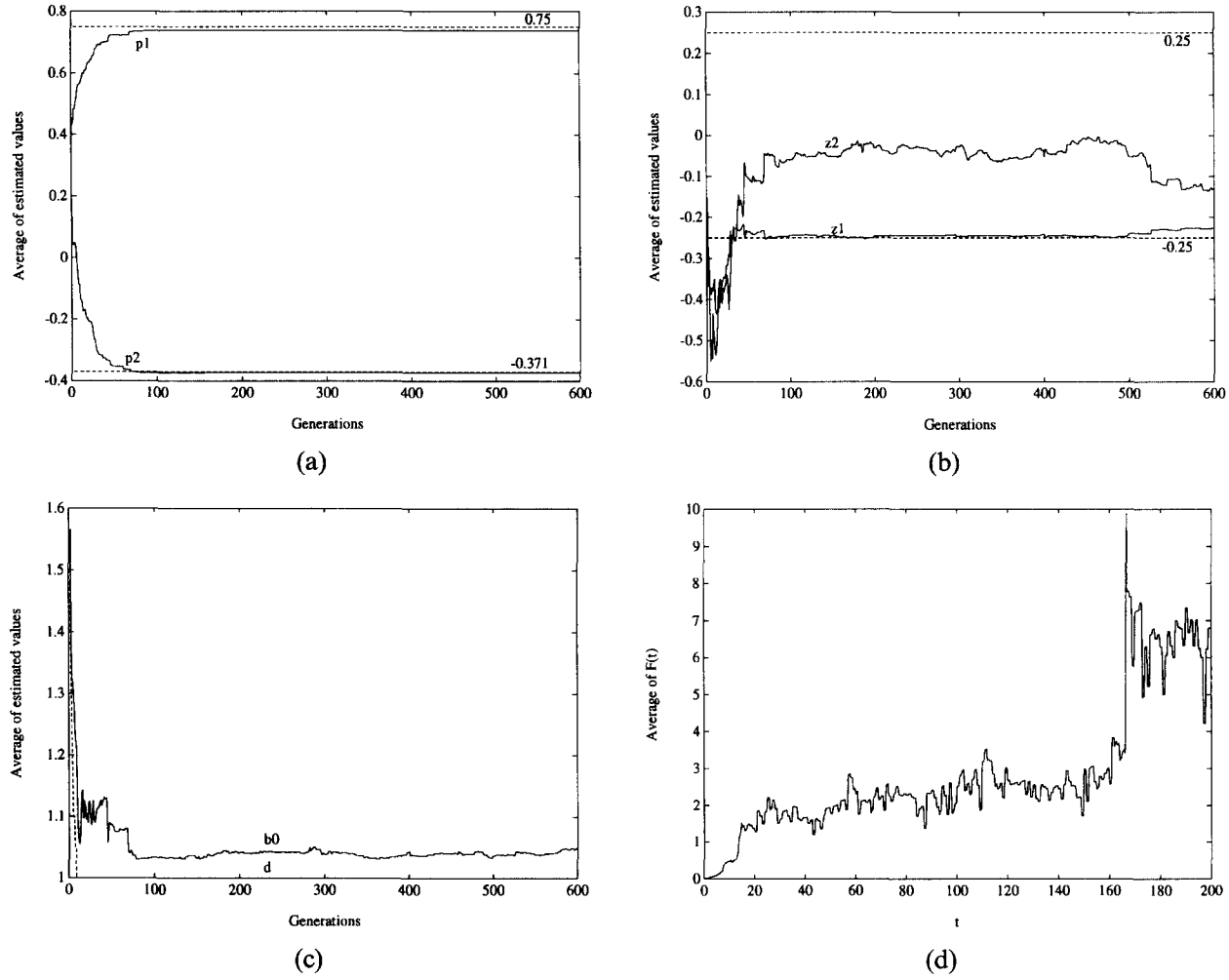

Fig. 6. Simple GA, the average estimation result of 20 simulations. (a) p_1, p_2; (b) z_1, z_2; (c) b_0, d; (d) $F(t)$.

1. Input/output data acquisition under an experimentation protocol.
2. Choice of the model structure and complexity.
3. Estimation of the model parameters.
4. Validation of the identified model.

ASAGA is applied here to the problem of estimating model parameters to show its performance. Consider a system described by an ARMAX model:

$$A(q^{-1})y(t) = B(q^{-1})u(t-d) + C(q^{-1})e(t) \tag{11}$$

where q^{-1} is the backward shift operator, i.e. $u(t-1) = q^{-1}u(t)$, while $y(t)$, $u(t)$ and $e(t)$ are output, input and noise respectively. The objective is to identify the system polynomials, $A(q^{-1})$ and $B(q^{-1})$, and the delay d using the given input $u(t)$ and the output $y(t)$. The error sequence is defined as

$$\eta(t) = y(t) - \hat{y}(t) \tag{12}$$

with

$$\hat{A}(q^{-1})\hat{y}(t) = \hat{B}(q^{-1})u(t-\hat{d}) \tag{13}$$

where the estimated output, $\hat{y}(t)$, is the output of a system driven by the actual input $u(t)$, and $\hat{A}(q^{-1})$ and $\hat{B}(q^{-1})$ are the estimates of $A(q^{-1})$ and $B(q^{-1})$ respectively.

The fitness function to be maximized is chosen as

$$F(t) = \frac{1}{\sum_{i=0}^{w}[\eta(t-i)]^2} \tag{14}$$

where w is the window size.

The simulated system is the same as that in Ref. 3 for the purpose of comparison, and $C(q^{-1})$ is not used. The system polynomials, poles and zeros are as follows:

$$\begin{aligned} A(q^{-1}) &= 1.0 - 1.5q^{-1} + 0.7q^{-2} \\ B(q^{-1}) &= b_0(1.0 + 0.5q^{-1} + 0.0q^{-2}) \end{aligned} \tag{15}$$

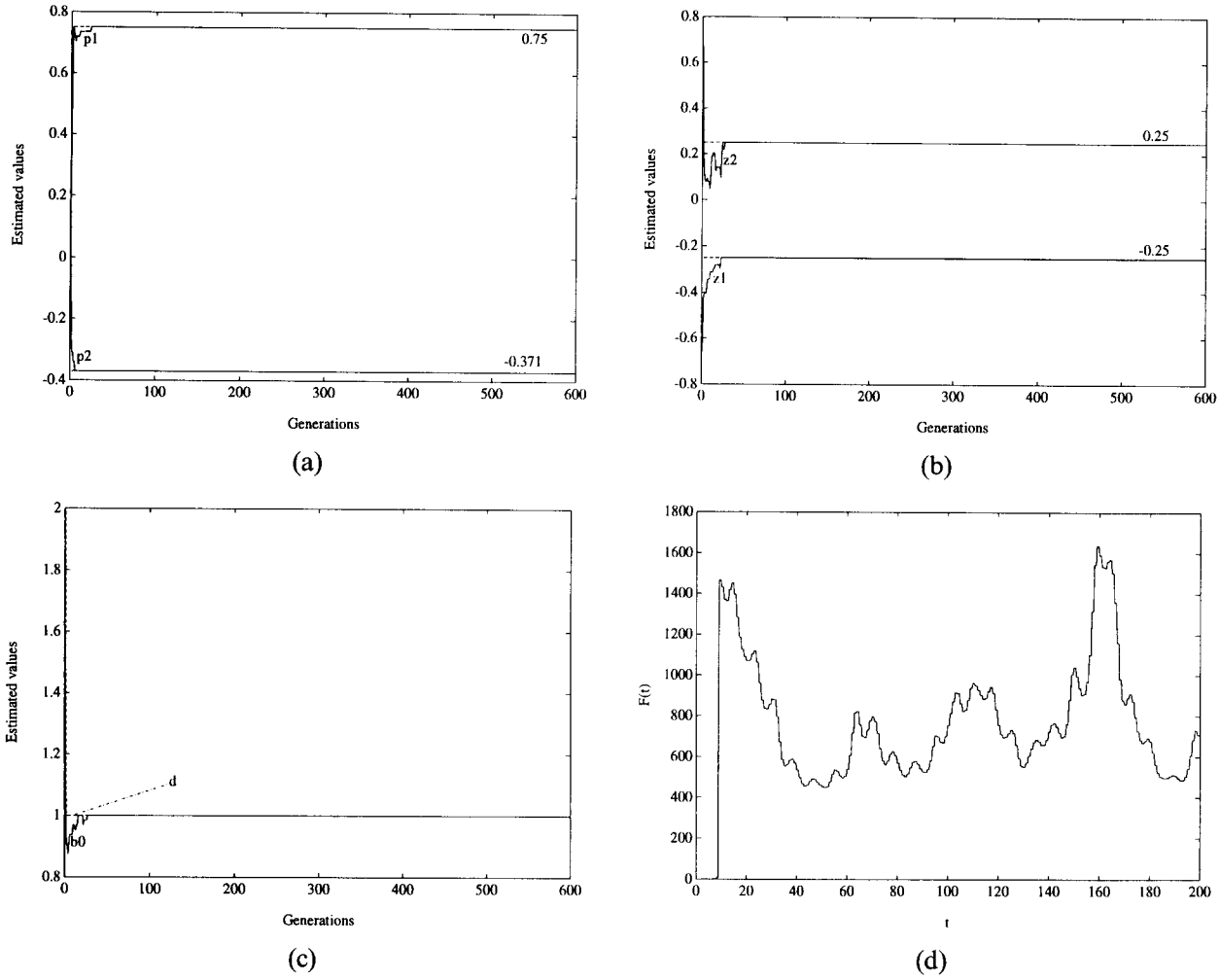

Fig. 7. ASAGA, the best estimation result among 20 simulations. (a) p_1, p_2; (b) z_1, z_2; (c) b_0, d; (d) $F(t)$.

$$\begin{aligned} &\text{poles: } (0.7500 \pm j0.3708) \\ &\text{zeros: } (-0.2500 \pm 0.2500) \end{aligned} \tag{16}$$

where the gain, b_0 is 1. The delay, d, is 1.

To apply the GA, a reparameterization is introduced. In the reparameterized plane, $[a, b]$ means a pair of real roots, $(a \pm b)$, when b is positive, and it means a pair of complex roots, $(a \pm jb)$, when b is negative or zero. Thus, any complex conjugate poles can be represented as two real numbers by using reparameterization. In the reparameterized plane, poles and zeros can be rewritten as

$$\begin{aligned} &\text{poles: } [0.7500, -0.3708] = [p_1, p_2] \\ &\text{zeros: } [-0.2500, +0.2500] = [z_1, z_2]. \end{aligned} \tag{17}$$

A simple GA and ASAGA are applied to identify p_1, p_2, z_1, z_2, b_0 and d. In the simulation, seven bits were used for each parameter except for d (two bits). Thus, a string is 37 bits long. In the simple GA, $p_c = 0.8$, $p_m = 0.1$, population size = 100 and window size = 30. Values for p_c and p_m were determined by trial and error in order to show the best performance. b_0 is assumed to be between 0 and 2, and p_1, p_2, z_1 and z_2 are assumed to be in the $[-1, 1]$ range, so the resolution is slightly finer than 0.02. Although the true value of p_2 is -0.3708, the limitation on the resolution due to coding makes the best estimated p_2 equal to $-0.375(= -1 + 80/2^7)$. So, p_2 is assumed to be in the $[-0.996, 1.004]$ range, which makes the best estimated value of p_2 equal to -0.371. In ASAGA, N_{reset} in equation (10) is 3, and the other conditions are the same as the simple GA.

The test input for the sample data is chosen as

$$u(t) = \sin(t) - \sin(t/2.5) + \text{random}(-1, 1) \tag{18}$$

where random $(-1, 1)$ is a random number between -1 and 1.

The input and output of the system are shown in Fig. 4. One simulation was done using 200 samples with three generations per sample, that is, 600 generations for each run. 20 simulations were done for each algorithm.

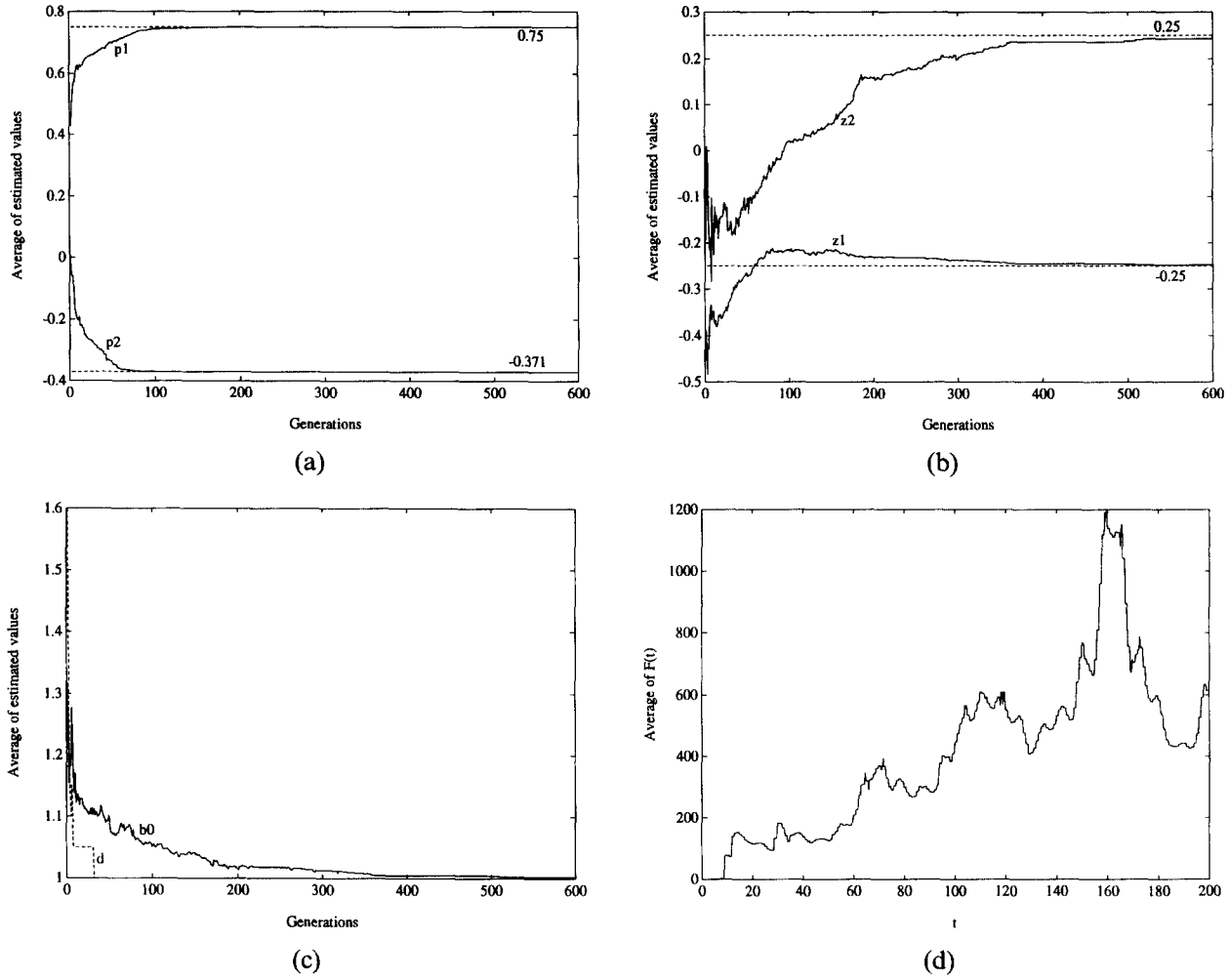

Fig. 8. ASAGA, the average estimation result of 20 simulations. (a) p_1, p_2; (b) z_1, z_2; (c) b_0, d; (d) $F(t)$.

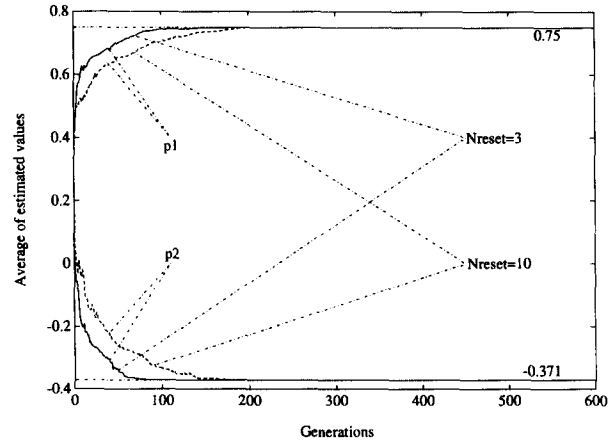

Fig. 9. ASAGA, p_1, p_2: the average estimation result of 20 simulations using $N_{\text{reset}} = 3, 10$.

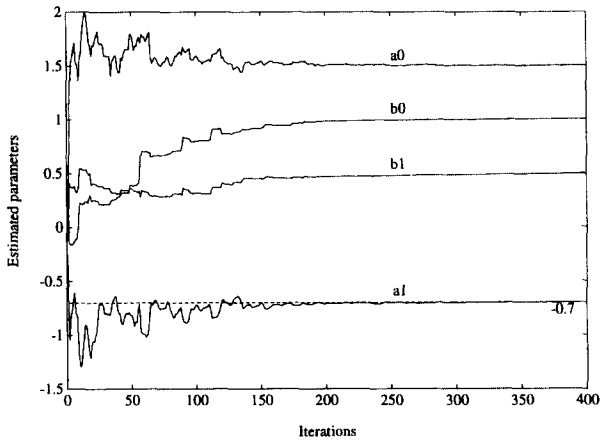

Fig. 10. Estimated parameters using the gradient algorithm.

Figure 5 shows the best result among 20 simulation results using the simple GA. It shows six estimated parameters and $F(t)$ for each generation. Figure 6 shows the averaged parameters and $F(t)$ over 20 simulations. The results obtained using ASAGA are shown in Figs 7 and 8, which correspond to Figs 5 and 6 respectively.

Comparing Figs 5 and 7, it can be seen that ASAGA has a good hill-climbing capability. Note that ASAGA has found the exact parameters. Figures 6 and 8 show that ASAGA consistently outperforms the simple GA. To see the effect of the adaptive cooling schedule the problem was solved 20 times by ASAGA using $N_{\text{reset}} = 10$. Figure 9 shows the averaged identification results for p_1 and p_2 using 3 and 10 for N_{reset}. It shows faster convergence when N_{reset} is 3, which indicates that repeated cooling schedule contributes to the fast convergence, and thus small values of N_{reset} are good. However, too-small values of N_{reset} may reduce the SA feature of ASAGA.

For a comparison with a recursive method, a gradient algorithm is introduced. The algorithm uses the following series-parallel identification model:

$$\hat{y}(t+1) = \sum_{i=0}^{n-1} \hat{a}_i(t)y(t-i) + \sum_{j=0}^{m-1} \hat{b}_j(t)u(t-j) \qquad (19)$$

and the adaptation law:

$$\hat{a}_i(t+1) = \hat{a}_i(t) - \eta \cdot \frac{e(t+1)y(t-i)}{1 + \eta(\sum_{i=0}^{n-1} y^2(t-i) + \sum_{j=0}^{m-1} u^2(t-j))}$$

$$\hat{b}_j(t+1) = \hat{b}_j(t) - \eta \cdot \frac{e(t+1)u(t-j)}{1 + \eta \cdot (\sum_{i=0}^{n-1} y^2(t-i) + \sum_{j=0}^{m-1} u^2(t-j))} \qquad (20)$$

where η is a positive adaptation gain. Equation (15) can be represented as

$$y(t+1) = 1.5y(t) - 0.7y(t-1) + u(t) + 0.5u(t-1). \qquad (21)$$

True values of a_0, a_1, b_0 and b_1 in equation (19) are 1.5, −0.7, 1.0 and 0.5 respectively. Figure 10 shows the recursive identification result using equations (19) and (20) with $\eta = 2$ and the same input as earlier. The recursive method did not do as well as ASAGA. The recursive method requires more sample data, and it is very difficult to determine the optimal adaptation gain. Note that the recursive identification method and ASAGA have solved basically different problems. The recursive identification method uses the identification model, while ASAGA does not know any system information, and the recursive identification can adapt only one step per one sample.

5. CONCLUSION

An efficient hybrid genetic algorithm named ASAGA (Adaptive Simulated Annealing Genetic Algorithm) has been proposed. ASAGA was designed to preserve the merits of both SA and GA, to improve GA while not greatly altering the identities of each algorithm, and to speed up the convergence. The differences from previous work on GA with SA are the use of the standard crossover operator and the introduction of the adaptive cooling schedule. An example of discrete-time system identification shows that the proposed algorithm, ASAGA, is effective and superior to the simple GAs, and even better than the gradient algorithm. It has also been shown that the adaptive cooling schedule contributes to the hill-climbing and the fast convergence. Identification using ASAGA has been shown to be basically different from recursive identification methods.

More theoretical analysis of ASAGA, and applications to more versatile control system structures, should be done in order to complete the exploration of the ideas that have been presented here.

REFERENCES

1. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA (1989).
2. Davis, L., *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York (1991).
3. Kristinsson, K. and Dumont, G. A., System identification and control using genetic algorithms. *IEEE Trans. Syst. Man Cybern.* **22**(5), 1033–1046 (1992).
4. Karr, C. L. and Gentry, E. J., Fuzzy control of pH using genetic algorithms. *IEEE Trans. Fuzzy Syst.* **1**(1), 46–53 (1993).
5. Ichikawa, Y. and Sawa, T., Neural network application for direct feedback controllers. *IEEE Trans. Neural Networks* **3**(2), 224–231 (1992).
6. Davis, L., *Genetic Algorithms and Simulated Annealing*. Pitman Publishing, London (1987).
7. Adler, D., Genetic algorithms and simulated annealing: a marriage proposal. *IEEE Int. Conf. on Neural Networks*, pp. 1104–1109 (1993).
8. Sirag, D. and Weisser, P., Toward a unified thermodynamic genetic operator. *Proc. Second Int. Conf. on Genetic Algorithms*, pp. 116–122 (1987).
9. Brown, D., Huntley, C. and Spillane, A., A parallel genetic heuristics for the quadratic assignment problem. *Proc. Third Int. Conf. on Genetic Algorithms*, pp. 406–415 (1989).
10. Landau, I. D., *System Identification and Control Design*. Prentice Hall, New Jersey (1990).