A Generalized Self Organizing Evolutionary Algorithm and Its Efficient Application to Control Problems

일반화된 자기 형성 진화 알고리즘의 개발과 제어 문제에 대한 효율적 응용에 대한 연구

Il-Kwon Jeong and Ju-Jang Lee
(정 일 권, 이 주 장)

요 약: 널리 쓰이는 진화 알고리즘은 크게 두 가지가 있다. 유전 알고리즘과 진화 기법이 그것이다. 이들 알고리즘은 실행 전에 사용자가 정해주어야 하는 변수들을 가지고 있다. 본 논문에서는 이 두 알고리즘을 일반화시키고 집단의 크기, 교차변이 연산자 적용 확률 그리고 돌연변이 연산자 적용 확률과 같은 변수들을 알고리즘이 수행되는 동안 스스로 정하는 일반화된 자기 형성 진화 알고리즘을 제안한다. 제안된 알고리즘의 타당성과 효용성은 시스템 동정화와 다개체 시스템 제어의 두 가지 복잡한 제어 문제에 대한 적용을 통해 보여진다.

Keywords: evolutionary algorithm, self organizing

I. Introduction

There are two widely used evolutionary algorithms. One is the genetic algorithm and the other is the evolutionary strategy. Genetic algorithms (GAs) are robust search and optimization techniques which are based on the natural selection and genetics. GAs are applied to a number of practical problems nowadays. GAs are different from conventional optimization methods in several ways. The 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, and neural network controller [2][3]. 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 values. 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 reproduced solutions then undergo recombination which consists of crossover and mutation. 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].

The evolutionary strategy (ES) is another simulated evolution technique [5][6]. Both in the GA and the ES, a population of individual is arbitrarily initialized and evolves towards the better regions of the search space by means of a stochastic process of selection, mutation, and recombination if appropriate. These methods differ in the specific representation, mutation operations, and selection procedures. While the GA emphasizes chromosomal operators based on the observed genetic mechanisms (e.g., crossover and bit mutation), the ES emphasizes the adaptation and diversity of behavior from parent to offspring over successive generations.

In this paper, we propose a generalized self organizing evolutionary algorithm (GSOEA) for multimodal function optimization which is designed to merge the ES into the GA and to set its control parameters such as population size, crossover probability, and mutation probability adaptively during the execution of it. The choice of the crossover probability, p_c and the mutation probability, p_m is known to critically affect the behavior and performance of an evolutionary algorithm. 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 the 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 an algorithm. In our algorithm, p_c, p_m, and the size of the population are determined by the GSOEA itself to realize the twin goals of maintaining diversity in the population and sustaining the convergence capacity of the algorithm.

There have been similar works to improve an evolutionary algorithm. Jeong and Lee incorporated the simulated annealing technique into a GA [2]. They used a fitness modification technique and an adaptive mutation probability in [3]. A local improvement operator was introduced in [10]. The crossover and the mutation probabilities vary according to the population maximum fitness and the population mean fitness in [7]. But, their adaptive rules are not general in the sense that the rules only apply to the solutions above average. Bryant used the relative credit of each genetic operator over some generations [8]. However, his algorithm needs more memory and longer computation time compared to above-mentioned algorithms. Smith and Fogarty investigated the use of genetically

접수일자 : 1997. 1. 7., 수정완료 : 1997. 5. 14.
정일권, 이주장 : 한국과학기술원 전기및전자공학과
※ This paper was supported (in part) by NON DIRECTED RESEARCH FUND, Korea Research Foundation

encoded mutation rates within a steady state GA [9]. Again their algorithm needs much memory, and it cannot be applied to a generational GA. This paper is organized as follows. In section 2, a generalized self organizing evolutionary algorithm (GSOEA) is proposed. In section 3, we present control applications of GSOEA. The conclusions are presented in the last section.

Ⅱ. Generalized self organizing evolutionary algorithm

1. Generalization of evolutionary algorithms

The genetic algorithm and the evolutionary strategy are very similar to each other and each has its own merits or demerits. So it is natural to combine them to produce a generalized evolutionary algorithm which preserves the merits of the both algorithms. A generalized evolutionary algorithm (GEA) is based on a very simple concept and is simply constructed as follows. The GEA uses reproduction, selection, crossover. It can use a binary string or a real valued vector for a solution, and it can use a gaussian mutation for a solution of real valued vector. When a GEA uses a binary encoded string, the GEA becomes an ordinary GA. When a GEA uses a real valued vector for a solution, a gaussian mutation, and no crossover operator, the GEA becomes an ES. Thus, a GA or an ES can be viewed as a special case of the GEA. The word '*generalized*' was used in that sense.

2. Generalized self organizing evolutionary algorithm

A generalized self organizing evolutionary algorithm (GSOEA) consists of GEA described in the previous subsection and an adaptive mechanism determining its control parameters automatically.

In optimizing unimodal functions, it is important that an evolutionary algorithm (EA) should be able to converge to the optimum in as few generations as possible. For multimodal functions, there is a need to be able to locate the region in which the global optimum exists, and then converge to the optimum. An usual EA possesses poor hill-climbing capability, and it is vulnerable to getting stuck at a local optimum, especially when the populations are small. The significance of p_c and p_m in controlling EA performance has long been acknowledged in EA 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 EA into a purely random search, while some mutation is required to prevent the premature convergence of the EA to suboptimal solutions. The population size also affects the EA performance. Premature convergence may occur when the size of a population is small, while a population of large size makes the algorithm slow. Usually, the choice of p_c, p_m, and the population size is left to the user to be determined statically prior to the execution of the EA. Typical values of p_c and p_m are in the range [0.5, 1.0] and [0.001, 0.1], respectively.

In order to overcome the above-stated problem of difficulty in choosing the EA parameters, we suggest the following expressions which are the main components of the GSOEA, and they determine p_c and p_m adaptively in response to the fitness values of the solutions.

$$p_c = k_1(f_{\max} - f')/(f_{\max} - f_{\min}) + k_2 \qquad (1)$$
$$k_1 + k_2 \le 1$$
$$p_m = k_3(f_{\max} - f)/(f_{\max} - f_{\min}) + k_4 \qquad (2)$$
$$k_3 + k_4 \le 1$$

where $f_{\max}$ is the maximum fitness value and $f_{\min}$ is the minimum fitness value of a population. f' is the larger of the fitness values of the solutions to be crossed, and f is the fitness of the solution to be mutated. k_1 k_2, and k_4 are positive constants. 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. $N_{pop} \leftarrow N_{pop} - n_2$, $k_3 \leftarrow c \times k_3$, go to step 2.

where n_1, n_2, and n_{reset} are positive constants of integers, and $0 < c < 1$. As we can see from (1), p_c is a linear function of f', and it varies from $k_1 + k_2$ to k_2 as f' changes from $f_{\min}$ to $f_{\max}$. Similarly, p_m varies from $k_3 + k_4$ to k_4 as f changes from $f_{\min}$ to $f_{\max}$. Thus, the higher (lower) the fitness of a solution, the lower (higher) the probability of crossover or mutation of the solution. The idea behind the equations is that the near-optimal (high fitness) solutions should be prevented from the disruption by crossover or mutation, and they can aid in the convergence of the GSOEA. Inferior (low fitness) solutions have higher values of p_c and p_m, which promote the GSOEA to explore the search space. Therefore, we are able to preserve 'good' solutions of the population while the low fitness solutions prevent the GSOEA 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 near zero from its initial value and the mutation operator becomes to behave like a normal one with the probability of k_4. It has been empirically proved that the adaptive mutation probability accelerates the convergence and improves the hill-climbing property of an EA [3]. When the fittest (the best solution) is the same for n_{reset} generations, that is, the algorithm is getting stuck at a local optimum, p_m is enlarged to its initial value to possibly direct the search to the global optimum, and the population size N_{pop} is increased by n_1 to search wider region of the search space. Otherwise, N_{pop} is decreased by n_2 at every generations to speed up the algorithm.

There remains the problem of the choice of values for k_1, k_2, initial k_3, k_4, n_1, n_2. $k_1 + k_2$ ($k_3 + k_4$) and k_2 (k_4) are the upper and the lower limit of p_c (p_m), respectively. Considering the typical ranges for p_c and p_m, we assign k_2

and k_4 a value of 0.5 and 0.01, respectively, and use a value of 0.5 for k_1 and initial k_3. n_1 and n_2 are determined by trial and error. Other parameters are adopted from [3].

Ⅲ. Control applications

1. System identification

The problem considered here is the same as those in [1][2]. 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 identifying the system polynomials, $A(q^{-1})$, $B(q^{-1})$, 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 the window size. The system polynomials, poles and zeros in the re-parameterized 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 the GSOEA to identify p_1, p_2, z_1, z_2, b_0 and d. The gain b_0 is assumed to be in the range [0, 2], and the poles and zeros in [-1, 1].

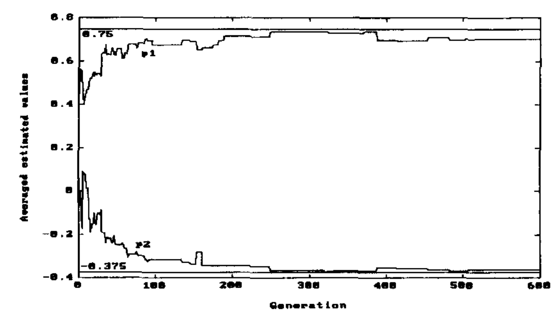

Fig. 1. Identification of the poles using a simple GA.

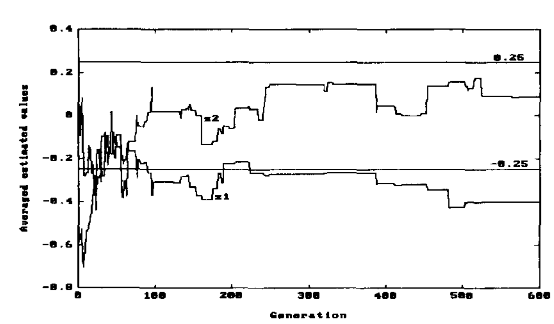

Fig. 2. Identification of the zeros using a simple GA.

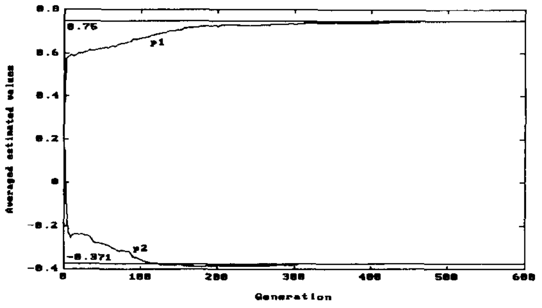

Fig. 3. Identification of the poles using the GSOEA.

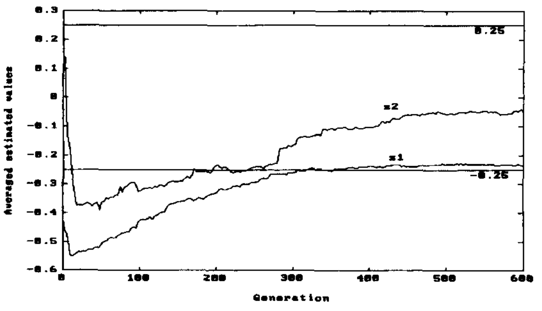

Fig. 4. Identification of the zeros using the GSOEA.

We use a binary encoding for the simple GA. 7 bits are assigned to each parameter except d (2 bits), so the resolution is slightly smaller than 0.02. A string consists of 37 (= 7 × 5 + 2) bits. A real valued vector with 6 elements which correspond to the parameters of the poles, the zeros, the gain, and the delay is used for a string of the GSOEA since the current problem to be solved involves real valued parameters. We use p_c=0.8, p_m=0.01, N_{pop}=100, and w=30 for the simple GA. k_1= k_2=initial k_3=0.5, k_4=0.01, initial N_{pop} =50, n_{reset}=5, n_1=6, n_2=2, c=0.9, and w=30 are used for the GSOEA. Input for the sample data is given as

$$u(t) = \sin(t) - \sin(t/2.5) + random(-1\sim1) \tag{10}$$

where $random(-1 \sim 1)$ is a uniform random number in the range (-1, 1). One simulation is done using 200 samples with 3 generations per one sample, that is, 600 generations. Both the algorithms are simulated 10 times. Figure 1 ~ 2 show the average of the identification results of the poles and zeros with the simple GA. Figure 3 ~ 4 show the average of the results with the GSOEA. The GSOEA shows the better hill-climbing and optimum finding capability than the simple GA. The average of the average population sizes of 10 simulations is 43.4, which is much smaller than that of the simple GA though the performance of the GSOEA is much better.

2. Multi-agent system control

2.1 Problem description

Our objective here is to investigate how relatively simple agents can adaptively learn to solve a complex problem. Each agent should learn simple behaviors which are collectively sufficient to solve the problem. Agents have to decompose the problem effectively but this decomposition should be an emergent property of adaptive learning and not pre-programmed. It is an important motivation of this application that a problem should be solved with the minimal possible direction from the programmer or the trainer. We apply the simple GA and the GSOEA to finding a rule table for agents.

The experimental problem in this section is a soccer game in a simplified soccer model environment which will be explained later. The players (agents controlled by rule tables) have to learn to play the game. Two players with the same agent-type make up a team in our simulation. It is assumed that the behavior of the players of one agent-type is pre-defined, because it is difficult to design an appropriate fitness function capable of evaluating the behavior of two teams simultaneously. The task of a team is to score a goal in a limited period while at the same time not to lose control of the ball to the opposing team. Success of a team depends on players learning to co-operate in order to score a goal.

Each player behaves independently of the other, and only knows about the existence of other players and the ball in the view window which will be described in the next section. So there is no direct communication between players, and their knowledge of the aims of other players is also indirect. Hence each player interacts with a highly dynamic environment. Learning (modifying the rule table) takes place through feedback gained from actions in the environment.

In our simulation, the simplified soccer model consists of 5×6 cells. The goal posts are located at the center of the top and the bottom of the ground, and each player and the ball are located as shown in Fig. 5 at the beginning of a game. There are two agent-types, A and B. The behavior of the agent-type B is predefined and that of the agent-type A is to be learned. We have simulated the following two situations.

Case 1 : Agent B_1 and B_2 are fixed during the game, that is, they just act like obstacles.

Case 2 : The ball is initially at the upper right corner of the ground. B_1 in front of the goal post is fixed. It may be thought as a goalkeeper. B_2 moves to capture the ball, and dribbles to the opponent's goal post when it captures the ball. So, it interferes in other players' actions.

2.2 Simplified soccer model

A simplified soccer (SS) model is similar to the effector automata (EA) model[11]. Both in the EA and the SS models, a cellular space is defined where individual processing units (automata or agents), operating in parallel, receive input from their local neighborhood, and produce an output using a pre-defined rule. Each cell is a location in space, and agents (automata) are entities that can occupy the cells. In the SS model, the output is an action to effect, such as moving to a neighboring cell. Agents with the same agent-type use the identical rules.

Figure 5 shows a playground of the SS model with 4 agents, A_1, A_2, B_1, and B_2, and a ball. It is assumed that each agent can move, or detect other objects (agents, goal posts, and a ball) in one of the following directions: top, left, right, and bottom. Figure 6 shows a view window of an agent. The orientation of a view window is fixed against the playground during a game. The center of the view window corresponds to the center of the agent. The agent can detect an agent of the same agent-type in the region I, II, and III, and can detect an agent of different agent-type in the region I. Goal posts in the region I and III can be detected. The ball is always detected; the position of the ball is either center, upper, lower, left, or right, according to the sector it belongs to.

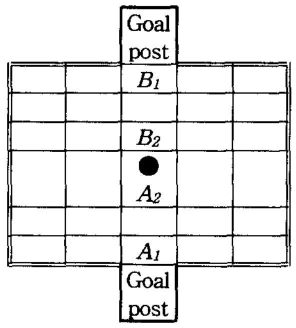

Fig. 5. A playground of the SS model.

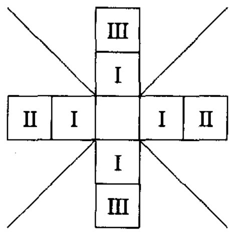

Fig. 6. A view window of an agent. ● represents the ball.

The behavior of each agent is governed by a rule table. An entry of a rule table is a condition-action rule of the form:

$$C(\cdots)L(\cdots)R(\cdots)T(\cdots)B(\cdots) \rightarrow action \qquad (11)$$

where CLRTB stands for center, left, right, top, and bottom.

Table 1. Actions used for the SS model.

action	description
MOVE [DIR]	move one cell in the specified direction.
DRIBBLE [DIR]	move one cell with the ball in the specified direction only when currently possessing the ball.
KICK [DIR]	kick the ball by two cells in the specified direction only when currently possessing the ball

The actions possible for the SS model are listed in Table 1. There are three actions; MOVE, DRIBBLE, and KICK. We need a simulator for the SS model. The simulator simulates the movements of agents and the ball according to the rule table for a given time steps, and scores a game. A game ends when a team scores a goal or the given time steps is over. If there are n agents and the ball in a cell, then one of the agents obtains the ball with the probability $1/n$.

2.3 Methodology

Some aspects to be considered to use the simple GA or the GSOEA are as follows:

• Chromosome representation and population size : A rule table of condition-action rules is indexed implicitly by the neighborhood pattern in the view window. In the view window, there are $11={}_4C_0+{}_4C_1+{}_4C_2$, $5={}_4C_0+{}_4C_1$, 2, and 2 possible configurations of agents of agent-type B, an agent A, the goal post of B, and the goal post of A, respectively. There are 5 (4 sectors and the center) configurations of the ball. The action part of a rule requires three bits; two bits for direction and one bit for an action type; MOVE or KICK. When an agent has the ball, MOVE is automatically recognized as DRIBBLE. Thus, a rule table encoded in a binary string requires $3300=11\times5\times2\times2\times5\times3$ bits. Both the simple GA and the GSOEA use a binary string for a solution.

• Fitness : The fitness function is defined as

$$f=\sum_{i=1}^{i=n_{\text{iter}}}F(i) \tag{12}$$

$$F(i)=F_0+\sum_{t=0}^{t_{final}}(f_{goal-A}(t)+f_{possess-A}(t)+f_{goal-B}(t)+f_{possess-B}(t)) \tag{13}$$

where i represents the ith simulated game, n_{iter} is a given iteration number of simulations, and t_{final} represents the time when the game ends. A game ends at t=30 when no goal occurs. In our simulation, n_{iter} is 5 and a value of 400 is assigned to F_0. Other functions in (13) are defined as in the following.

$$f_{goal-A}(t)=\begin{cases}1000, & \text{If the team } A \text{ scores a goal at time } t\\ 0, & \text{Otherwise}\end{cases} \tag{14}$$

$$f_{goal-B}(t)=\begin{cases}-90, & \text{If the team } B \text{ scores a goal at time } t\\ 0, & \text{Otherwise}\end{cases} \tag{15}$$

$$f_{possess-A}(t)=\begin{cases}10, & \text{If a player in the the team } A \text{ possesses the ball at time } t\\ 0, & \text{Otherwise}\end{cases} \tag{16}$$

$$f_{possess-B}(t)=\begin{cases}-10, & \text{If a player in the the team } B \text{ possesses the ball at time } t\\ 0, & \text{Otherwise}\end{cases} \tag{17}$$

• Control Parameters : We use the same parameters that were used in the previous application.

2.4 Simulation results and discussion

Figure 7 and Fig. 8 show the average results of 10 independent simulations for the case 1 and the case 2 using a simple GA. Figure 9 and Fig. 10 show the average results for the case 1 and the case 2 using the GSOEA, respectively. Each graph illustrates the average maximum fitness values of the team A versus generation numbers. Though we have used the elitist strategy the maximum fitness value is fluctuating. This is due to our probabilistic ball possessing policy. So, in absolute terms the fitness values indicate little of importance. The trend is far more revealing. In the case of the present experimental task the increase over generations indicates that the team is learning more and more appropriate behavior. The players have displayed co-operative behavior such as passing (using KICK action) the ball to their team-mate in order to make it easier to score a goal.

In the case 1, the simple GA and the GSOEA found a solution (rule-table capable of making the players of the team A score a goal) in a few generations. It shows that the result using the GSOEA is better and more oscillatory than that using the simple GA, which means that the GSOEA has more statistic hill-climbing capability. Furthermore, the GSOEA shows steady results throughout the generations.

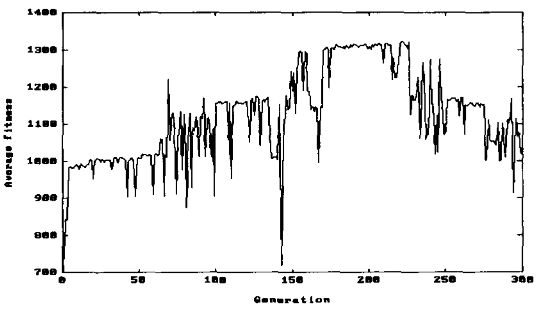

Fig. 7. The result of the case 1 using a simple GA.

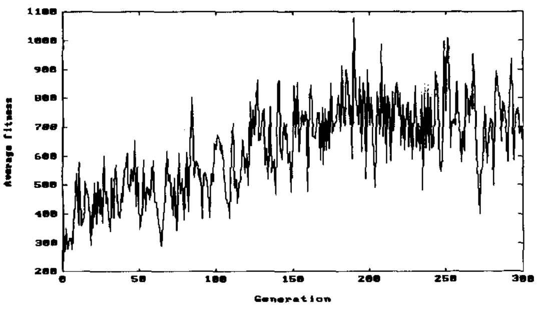

Fig. 8. The result of the case 2 using a simple GA.

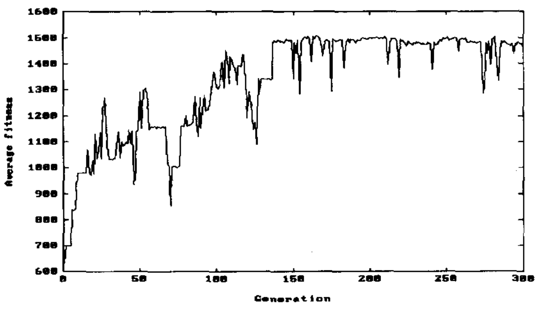

Fig. 9. The result of the case 1 using the GSOEA.

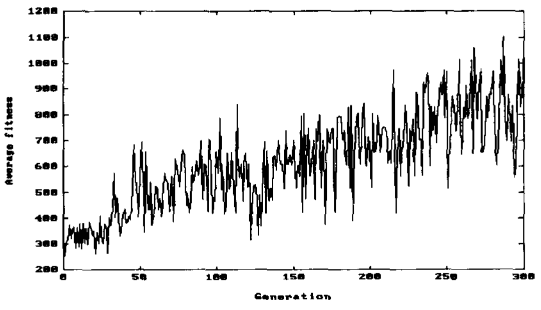

Fig. 10. The result of the case 2 using the GSOEA.

The GSOEA found a solution after about 220 generations in the case 2 which is a more difficult situation than the case 1. The fitness value is more oscillatory than that of the case 1. The players moved to the ball first and scored a goal using dribbles and passes. Usually, scoring a goal occurs after 20 time steps in the case 2, while it occurs after about 10 time steps in the case 1. Again the GSOEA shows steady increasing fitness over generations.

Ⅳ. Conclusions

A generalized self organizing evolutionary algorithm (GSOEA) was designed to generalize evolutionary algorithms, to prevent the premature convergence, and to sustain the convergence capacity of the GA. The GSOEA determines p_c, p_m, and N_{pop} automatically using its own rules, so we do not have to determine the values for the parameters prior to the execution of the algorithm. The GSOEA was applied to two problems; system identification and control of multi-agents playing a simplified soccer game. Simulation results indicate that the GSOEA has adaptive characteristics and improved hill-climbing capability compared to the simple GA. Using the adaptive population size, the execution time of the algorithm was significantly lowered. The theoretical analysis of the GSOEA remains for further study.

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] I. K. Jeong and J. J. Lee, "Adaptive simulated annealing genetic algorithm for control applications," *Int. J. Sys. Sci.*, vol. 27, no. 2, pp. 241-253, 1996.

[3] I. K. Jeong and J. J. Lee, "A modified genetic algorithm for neurocontrollers," *IEEE International Conference on Evolutionary Computation*, pp. 306-311, 1995.

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

[5] T. Back and H. P. Schwefel, "Evolutionary computation: an overview," *IEEE International Conference on Evolutionary Computation*, pp. 20-29, 1996.

[6] Z. Michalewicz, "Evolutionary computation: practical issues," *IEEE International Conference on Evolutionary Computation*, pp. 30-39, 1996.

[7] 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.

[8] B. A. Julstrom, "What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm," *Proceedings of the Sixth Int. Conf. on Genetic Algorithms*, pp. 81-87, 1995.

[9] J. Smith and T. C. Fogarty, "Self adaptation of mutation rates in a steady state genetic algorithm," *IEEE International Conference on Evolutionary Computation*, pp. 318-323, 1996.

[10] 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.

[11] D. Lohn and J. A. Reggia, "Discovery of self-replicating structures using a genetic algorithm," *IEEE International Conference on Evolutionary Computation*, pp. 678-683, 1995.

정 일 권

1992년 한국과학기술원 전기및전자공학과 졸업. 동대학원 석사(1994). 1994년 ~ 현재 한국과학기술원 전기및전자공학과 박사과정 재학중. 관심 분야는 진화연산, 신경회로망, 퍼지제어, 인공지능, 로보틱스.

이 주 장

제어·자동화·시스템공학 논문지 제 1권 제 2호 참조.