論文2000-37CI-5-4

유전자 알고리즘에서 연산자 확률 자율조정

(Self-tuning of Operator Probabilities in Genetic Algorithms)

鄭 成 薰 *

(Sung Hoon Jung)

요 약

진화연산 분야에서 연산자 확률을 조정하는 것은 주 연구분야 중 하나이다. 그 이유는 적당한 연산자 확률을 설정하는 것이 매우 지루하고 어려울뿐만 아니라 유전자 알고리즘의 성능향상에 매우 중요하기 때문이다. 많은 연구자들이 연산자 확률을 설정하거나 조절하는 여러가지 알고리즘을 소개했다. 그러나, 실험결과는 그리 만족할 만한 것이 아니었다. 더군다나, Tuson은 그의 논문에서 "연산자 조정은 반드시 좋은 것만은 아니다" 라고 주장하였다[1, 2]. 본 논문에서 우리는 유전자 알고리즘에서 연산자 확률을 자율조정하는 새로운 방법을 제안한다. 제안한 알고리즘을 4개의 함수와 한 개의 조합최적화 문제에 적용하여 테스트하고 일정한 유전자 확률을 갖는 단순 유전자 알고리즘과 Srinivas[3]가 제안한 알고리즘과 비교하였다. 실험결과는 본 논문에서 제안한 알고리즘이 다른 방법보다 상당히 우수함을 보였다. 이전의 방법과 비교해 볼 때 제안한 알고리즘은 계산량이 적고 연산자 확률을 진화시키기 위한 새로운 연산없이 상호 진화하며 진화를 위한 새로운 파라메터가 필요없는 등의 3가지 장점을 갖고 있다.

Aabstract

Adaptation of operator probabilities is one of the most important and promising issues in evolutionary computation areas. This is because the setting of appropriate probabilities is not only very tedious and difficult but very important to the performance improvement of genetic algorithms. Many researchers have introduced their algorithms for setting or adapting operator probabilities. Experimental results in most previous works, however, have not been satisfiable. Moreover, Tuson have insisted that "the adaptation is not necessarily a good thing" in his papers[1, 2] . In this paper, we propose a self-tuning scheme for adapting operator probabilities in genetic algorithms. Our scheme was extensively tested on four function optimization problems and one combinational problem; and compared to simple genetic algorithms with constant probabilities and adaptive genetic algorithm proposed by Srinivas *et al*[3]. Experimental results showed that our scheme was superior to the others. Our scheme compared with previous works has three advantages: less computational efforts, co-evolution without additional operations for evolution of probabilities, and no need of additional parameters.

* 正會員, 漢城大學校 情報電算學部

(School of Information and Computer Engineering, Hansung Univ.)

※ 본 연구는 2000년도 한성대학교 교내연구비의 지원을 받아 연구되었음.

接受日字:1999年12月27日, 수정완료일:2000年8月25日

Ⅰ. Introduction

Genetic algorithms(GAs), robust and systematic optimization paradigms, have been successfully applied to many scientific and engineering problems [3~11]. The performances of the GAs are considerably dependent upon the operator probabilities[1,2,12~16]. Finding appropriate operator probabilities is quite hard and time-consuming task because these vary with the problem being considered and the encoding and reproduction methods[1,14]. Many researchers have tried to automate this process with several algorithms[1,2,12~16]. Tuson[1,2] classified adaptation algorithms into two groups: co-evolutionary methods and learning-rule methods. More detailed classification was recently proposed by Hinterding *et al.*[14]. In[14], the authors classified adaptation algorithms into two orthogonal dimensions, *i.e.* adaptation type and adaptation level. Although some previous works have shown little improvement of performance[3,15,16], some works have shown not good results[1,2](more detailed discussions are available in section Ⅱ. Tuson *et al.*.[2] have even addressed that "operator adaptation is not necessarily a good thing" in his final discussions. Moreover, most previous methods have two difficulties: they need empirical setting of additional parameters in spite that such parameters are sensitive to performances; and they need empirical setting of initial probabilities of individuals that affects performances. These make those methods difficult to apply because such parameters must be also optimized.

In this paper, we propose a self-tuning scheme for adapting operator probabilities. The characteristics of our approach is summarized as: randomly assigned initial operator probabilities, co-evolution of operator probabilities with normal evolution of solutions without additional operations and additional parameters for evolution of operator probabilities. In our scheme, only two additional computations — calculation of new fitness and manipulation of operator probabilities in reproduction operation — are necessary. Finally, our scheme is a simple and robust method without much computational efforts and additional parameters.

With four function optimization problems and one combinational problem, we extensively experimented with our scheme, an adaptive genetic algorithm (AGA) proposed by Srinivas *et al.*[3], and simple genetic algorithms(SGA) with constant operator probabilities[4]. Experimental results showed that our scheme was superior to the others and very robust in spite of its simplicity. This paper is organized as follows. In section Ⅱ, previous works related to this topic are introduced briefly. Section Ⅲ describes the operations of proposed self-tuning scheme. The experimental results and discussions are provided in section Ⅳ. This paper concludes in section Ⅴ.

Ⅱ. Previous works

The adaptation of operator probabilities in GAs have been studied by many researchers with several algorithms[1,2,12~16]. The objectives of adaptation of operator probabilities are summarized as: reducing the amount of time that is spent finding suitable operator probabilities, increasing the quality of solutions obtained, and allowing the GA to find a solution of a given quality more quickly. This section described previous works in terms of classification. Tuson[1,2] classified adaptation algorithms into two groups: co-evolutionary methods, that encode operator probabilities into each member of the population; and learning-rule methods, that adapt the probabilities based on measurement of the operator productivities[15]. More detailed classification for adaptation of any parameters not to restrict to operator probabilities in evolutionary computation was recently proposed by Hinterding *et al*[14]. In[14], the authors classified adaptation algorithms into two orthogonal dimensions, *i.e.*, adaptation type and

adaptation level. The adaptation type dimension consists of two main categories: static (off-line adaptation) and dynamic (online adaptation), with the latter one divided further into deterministic, adaptive, and self- adaptive[17]. The adaptation level dimension consists of four categories: environment, population, individual, and component. In environment level adaptation, the environment such as the penalties in the fitness function, weights within the fitness function, and the fitness of an individual is changed. We introduced an environment-level adaptation scheme in the paper[17]. This scheme took not only the original fitness, but also its improved fitness not to fall into a premature convergence phenomenon. Population level adaptation is to apply the changed parameter to all members of the population. On the other hands, the individual level adaption affects only each individual of the population. Component level adaptation adjusts parameters local to some component or gene of an individual in the population.

In this paper, we focus on the adaptation of operator probabilities. Static methods are to give static operator probabilities obtained to the GA. Unlike the static methods, dynamic adapting methods adaptively change the initial probabilities during the GA run. The deterministic methods are to change the probabilities by only deterministic rules without any informations of the GA such as fitness[14]. For example, a deterministic equation[16] for mutation probability can be used as:

$$p_m = 0.1 - 0.09 \times \frac{g}{G}$$

where g is the generation number, where $0 \le g \le G$ and p_m is the mutation probability varied from 0.1 to 0.01. In adaptive methods, the adaptation algorithms use some informations of feedback from the GA such as fitness[14]. Tuson [2] investigated a COBRA(Cost Operator-Based Rate Adaptation) algorithm that adjusted the probabilities based on measurement of operator productivities — the average improvement in fitness from parents to offsprings. The algorithm is as follows. Given k operators, $o_1, \cdots, o_k$, let $b_i(t)$ be the *benefit* (in other words, operator productivities), $c_i(t)$ be the cost, and $p_i(t)$ be the probability of i th operator. Then the COBRA consists of the following steps.

1. The user decides a set of fixed probabilities p_i.
2. After G evaluations (the gap of operator probability readjustments), rank the operators according their values of b_i/c_i, and assign the operators their new probabilities according to their rank (i.e. the highest probability to the operator with the highest value of b_i/c_i).
3. Repeat step 2 every G evaluations.

Srinivas *et al.*[3] have introduced another adaptive method. Since the performance of our method is compared with that of this method in section IV, we will describe their adaptation method in detail in section IV.

In self-adaptive methods, the operator probabilities are directly encoded into each member of the population, allowing them to 'co-evolve' with the solutions.

Ho *et al.*[16] introduced a component-level adaptive algorithm. They used three operator probabilities on three groups of population. These operator probabilities are updated using some rules with a predefined step size S. According to the average fitness of the three groups, the three operator probabilities are readjusted with the step size S. This algorithm can be applied to adapting the mutation probability and crossover probability by running two modules individually. More detailed descriptions are available in[16].

Some previous methods that Tuson[1,2] investigated did not show good results. Tuson[1,2] from his experiments insisted that "operator adaptation is not necessarily a good thing". Moreover, previous methods have some difficulties in application. First, most methods have several additional parameters, which affects the performance. These parameters must be set by empirical tuning or adapted using

another adaptation algorithm. In self-adaptive methods, for example, the operator probabilities are encoded and evolved using a co-evolution crossover and mutation with their probabilities[2]. The encoding and co-evolution of probabilities make GAs complicated and increase the number of additional parameters. As another example, the gap G in the COBRA algorithm[2] and the step size S in[16] must be carefully selected by a user. In[15], the variable k and k^2 (decreasing fractions of the credit for parents and grandparents) and ϕ (reassigned ratio of operator probabilities) are additional parameters. Second, most methods need many additional informations such as the encoded probabilities in self-adaptive methods and the benefit and cost values during G generations in the COBRA algorithm. Third, the initial probabilities in most methods are empirically given by user selection. In[15], the initial probabilities are determined as $R_i=(R_L+R_U)/2$, where R_i is the initial probabilities and R_L and R_U are the lower and upper bounds that must be given by user selection, respectively. The user must decide a set of fixed probabilities in the COBRA algorithm. In[16], the initial probabilities of four operators are set to a constant value by user selection (0.2 in his experiments). Although the initial probabilities are adaptively changed during a GA run, the initial probabilities affects the performances of algorithms considerably[15]. From these perspectives, it is not sure that the previous methods are useful to do adaptation of operator probabilities.

Ⅲ. Self-tuning of Operator Probabilities

This section describes proposed self-tuning scheme of operator probabilities, which can be classified into an individual-level, adaptation method. The major ideas of our self-tuning scheme can be described as follows.

- Initial operator probabilities are set to not a constant value but uniformly distributed random values within predefined minimum and maximum limits. Therefore, no the other setting algorithms or empirical tuning are necessary for initial probabilities. The operator probabilities at each individual are co-evolved through the normal operations of GAs without additional co-evolving operations.
- The selection operation of GAs must select individuals with good probabilities as well as individuals with good fitness. To do this, the evaluation operation must consider not only how much the individuals are fit into the goal, but also how much the probabilities of the individuals are adequate. For the latter case, our evaluation operation uses the information how much the fitness of an offspring is improved from that of parents (we call this fitness improved fitness). This is because individuals with good probabilities will be evolved better than those with bad probabilities. By these selection and evaluation, the GAs with our scheme evolve not only solutions but the probabilities.
- Reproduction operations are the same as the SGA except for manipulating of operator probabilities.

In what follows, we will refer to the GA with the self-tuning method as a self-tuning genetic algorithm (STGA) against the SGA proposed by Goldberg[4]. In the STGA, initial crossover and mutation probabilities of ith individual are randomly set as:

$$p_c^i = rand({}^{\min}p_c, {}^{\max}p_c), i=1,\cdots,N \quad (1)$$

$$p_m^i = rand({}^{\min}p_m, {}^{\max}p_m), i=1,\cdots,N \quad (2)$$

where the N is the number of individuals; $rand(\cdot,\cdot)$ generates uniformly distributed random values within predefined limits, ${}^{\min}p_c$ and ${}^{\max}p_m$ for crossover probabilities, ${}^{\min}p_m$ and ${}^{\max}p_m$ for mutation probabilities; and the p_c^i and p_m^i are the initial crossover and mutation probabilities of i th individual, respectively. In the papers[1~3,8,9,12~16], the authors mentioned that GAs works successfully at

moderately large values of crossover probability p_c and small values of mutation probability p_m. It has been well known that too small values of crossover probability prevented a GA from exploration of new solutions in search space and too large values of crossover probability disrupted the solutions faster than selection can exploit them. It has been also known that too large values of mutation probability made a GA act as a random search algorithm and too small values of mutation probability prevented a GA from getting out of a local optimum. In our experiments, the ${}^{\max}p_c$ and ${}^{\min}p_c$ are set to 0.8 and 0.5 respectively; and the ${}^{\max}p_m$ and ${}^{\min}p_m$ to 0.2 and 0.001 respectively.

For the selection of individuals with good operator probabilities as well as the individuals with good fitness, we introduce new fitness for employing the improved fitness in the evaluation operation. For definition of the new fitness and improved fitness, we first define parents fitness as follows.

Definition 1: Parents fitness

Let two offsprings o_1 and o_2 be generated from the two parents p_1 and p_2, then the parents fitness $f^p_{o_1}$ and $f^p_{o_2}$ of the two offsprings are defined as:

$$f^p_{o_1}=f^p_{o_2}=(f_{p_1}+f_{p_2})/2.0 \quad \text{if do crossover} \qquad (3)$$
$$f^p_{o_1}=f_{p_1}, f^p_{o_2}=f_{p_2} \quad \text{otherwise}$$

where f_{p_1} and f_{p_2} are the original fitness of the two parents, respectively.

With this parents fitness, the improved fitness is defined.

Definition 2: Improved fitness

Let the fitness of ith individual be f_i and its parents fitness be f^p_i, then the improved fitness f^*_i of the individual is given as:

$$f^*_i=f_i-f^p_i \qquad (4)$$

With the improved fitness, the new fitness used in selection operation of our scheme is defined as follows.

Definition 3: New fitness

Let the f_i, f^p_i, and f^*_i be defined the same as definition 2, then the new fitness f^n_i of ith individual is defined as:

$$f^n_i=\begin{cases} f_i+f^*_i & \text{if } f^*_i>0 \\ 1/S & \text{otherwise} \end{cases} \qquad (5)$$

where S is the population size.

From the definition 3, some individuals whose improved fitness is greater than zero will survive in proportion to the sum of its fitness and the improved fitness. On the other hands, the others will be difficult to survive. We used $1/S$ instead of the f_i as new fitness for the case $f^*_i\le 0$ because it was revealed from experiments that using $1/S$ showed better performance than using f_i. Also, using $f_i+f^*_i$ for the case $f^*_i>0$ showed better performance than using only f^*_i. From these results, we can conclude that the improved fitness in our scheme plays a more important role to evolution than the fitness. It was also observed from experiments that our scheme showed better performance as the new fitness for the case $f^*_i\le 0$ as small. However, we used not zero but $1/S$ for preventing the sum of new fitness from being zero when no individuals are improved[1] When no individuals are improved, all individuals in our scheme will have same probability to be selected. These fitness definitions are somewhat similar to our previous paper[17]. In the paper, however, the improved fitness is used for only preventing a GA from falling the premature convergence phenomenon without considering evolution of operator probabilities. As a result, the definitions and algorithms of the paper are considerably different from those of this paper.

Algorithm 1 shows the overall structure of our scheme based on the previous descriptions.

Algorithm 1 Proposed Adaptation Algorithm

// t : time //
// P : populations //
// N : the number of individuals //
// f_i, f_i^p, f_i^*, f_i^n : fitness, parents fitness, improved fitness, and new fitness of i th individual //
// p_1, p_2, c_1, c_2 : two parents and generated two offsprings //
// p_c^i, p_m^i : crossover and mutation probabilities of i th individual //
// $rand(min, max)$: random float number generation within min and max limits //
// ${}^{min}p_c$, ${}^{max}p_c$: lower and upper limits of crossover probability //
// ${}^{min}p_m$, ${}^{max}p_m$: lower and upper limits of mutation probability //

1 $t \leftarrow 0$
2. initialize $P(t)$
[1]If the sum of new fitness becomes to zero, then the roulette wheel selection does not operate properly.
3. **for** $i=1$ to N
4. set initial crossover probability p_c^i to $rand({}^{min}p_c, {}^{max}p_c)$
5. set initial mutation probability p_m^i to $rand({}^{min}p_m, {}^{max}p_m)$
6. **end for**
7. evaluate $P(t)$
8. set f_i using the fitness function
9. set f_i^p to zero
10. If $f_i > f_i^p$ **then**
11. set f_i^n to $f_i + f_i - f_i^p = 2f_i$
12. **else**
13. set f_i^n to $1/S$
14. **end of**
15. **While** (**not** termination-condition)
16. **do**
17. $t \leftarrow t+1$
18. select $P(t)$ from $P(t-1)$
19. recombine $P(t)$
20. **for** $i=1$ to N with step 2
21. set ith and $i+1$th individuals to p_1 and p_2
22. set $p_c = (p_c^{p_1} + p_c^{p_2})/2$
23. decide whether do crossover or not with crossover probability p_c
24. **if** do crossover **then**
25. set $p_c^{c_1} = p_c^{c_2} = p_c$
26. set $p_m^{c_1} = p_m^{c_2} = (p_m^{p_1} + p_m^{p_2})/2$
27. set $f_{c_1}^p = f_{c_2}^p = (f_{p_1} + f_{p_2})/2$
28. **else**
29. set $p_c^{c_1} = p_c^{p_1}$ and $p_c^{c_2} = p_c^{p_2}$
30. set $p_m^{c_1} = p_m^{p_1}$ and $p_m^{c_2} = p_m^{p_2}$
31. set $f_{c_1}^p = f_{p_1}$ and $f_{c_2}^p = f_{p_2}$
32. **end if**
33. do mutation offsprings c_1 and c_2 with $p_m = p_m^{c_1}$ and $p_m = p_m^{c_2}$ respectively
34. **end for**
35. evaluate $P(t)$
36. set f_i using the fitness function
37. If $f_i > f_i^p$ **then**
38. set f_i^n to $f_i + f_i^* = f_i + f_i - f_i^p$
39. **else**
40. set f_i^n to $1/S$
41. **end if**
42. **end**

The initial operator probabilities are randomly assigned by uniformly distributed random number generator within two limits. By doing randomly assignment of initial probabilities, it is not necessary that the selection of constant initial probabilities for crossover and mutation. This is an advantage of our scheme. Initially, the parents fitness of all individuals is set to zero because the individuals have no parents. As a result, the new fitness of all individuals with $f_i > 0$ is set to $2f_i$ and the new fitness of the others with $f_i = 0$ to $1/S$ initially. In recombination operation (in other words, reproduction operation), the crossover probabilities of two parents

selected is averaged for deciding a crossover probability. With this crossover probability, it is decided whether the crossover operation do or not. If do crossover, then the crossover probabilities, mutation probabilities, and parents fitness of offsprings are calculated by averaging those of two parents. Otherwise, such values are set to those of parents. With the decided mutation probability by above processing, two offsprings are mutated. In the evaluation, f_i is obtained by the fitness function and f_i^n is set to $f_i+f_i-f_i^p$ if $f_i>f_i^p$ or to $1/S$ otherwise. Our scheme can be regarded as a combination method of GAs and gradient based search.

The main advantages of our work compared to the previous works are summarized as follows. First, although our scheme co-evolve, it does not need more additional informations such as encoded probabilities and update rules for the probability part that are necessary in previous co-evolutionary algorithms. Second, our scheme does not use any additional operations to evolve the operator probabilities. Third, while most previous methods need additional parameters that must be empirically set by a user in spite that the performances are sensitive to those, our scheme does not need any additional parameters. Fourth, the initial probabilities of individuals in our scheme are set to uniformly distributed random values within predefined limits while in the other schemes a user must select constant probabilities that affects the performances. Finally, it can be viewed that our scheme is a simple and robust method without much computational efforts and additional parameters that a user must set.

Ⅳ. Experimental Results and Discussion

Our scheme was tested on four function optimization problems and one combinational problem. The four functions were chosen to cover the wide types of possible functions as follows.

$$\begin{aligned} &G_1: y=x^2, && \text{where } -6\le x\le 10\\ &G_2: y=x\cdot sgn(x), && \text{where } -4\le x\le 10\\ &G_3: y=10x\cdot sgn(x)(\sin(10\pi x)+1)(\sin(\pi x)+1) && \\ &\qquad \text{,where } -10\le x\le 20 && (6)\\ &G_4: y=10x\cdot(\sin(\pi x)+1)\exp && \\ &\qquad (x\cdot sgn(x)/2), \text{ where } -15\le x\le 10 && \end{aligned}$$

The y values are used as fitness values in experiments.

A combinational problem G_5 was employed in order to measure the performances in combinational optimization. The G_5 problem is a type of pattern matching problems as depicted in Figure 2(b). A bit pattern whose length is the same as the individual length is randomly generated and GA finds the pattern. The bit pattern generated is different for each run and the probability of occurring 1 and 0 is 0.5. The fitness value of ith individual I_i in the population pool is calculated using following fitness function.

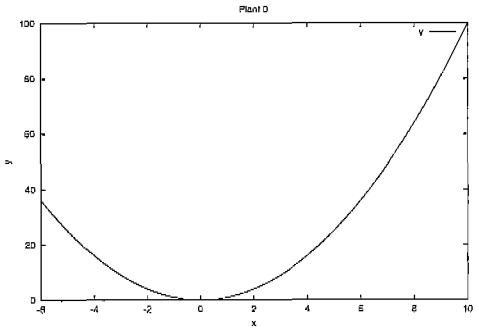
$$f(I_i)=\sum_{j=1}^{h} f^j \begin{cases} f^j=1 & \text{if } T^j=I_i^j \\ f^j=0 & \text{if } T^j\neq I_i^j \end{cases} \qquad (7)$$

where h is the number of bit, f^j is the fitness value of jth bit, T^j is jth bit in the given bit pattern, and I_i^j is the jth bit of ith individual. Therefore, the optimum fitness value is h.

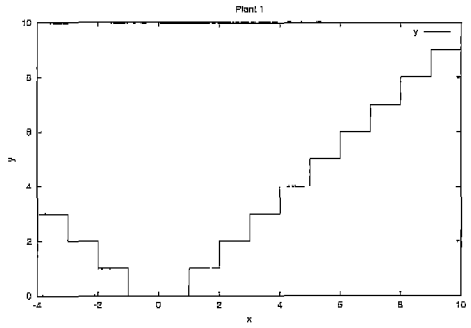
Figure 1 and 2 (a) show the x-y plots of the four functions and the fitness of the pattern matching problem when the given bit pattern is represented by 16 bits and all bits of the pattern are zero, respectively.

Functions G_1 and G_2 are relatively simple unimodal functions. While functions G_3 and G_4 are quite complex multimodal functions with many local maxima. Function G_2 and the pattern matching problem cannot be optimized by means of any conventional gradient techniques since there is no

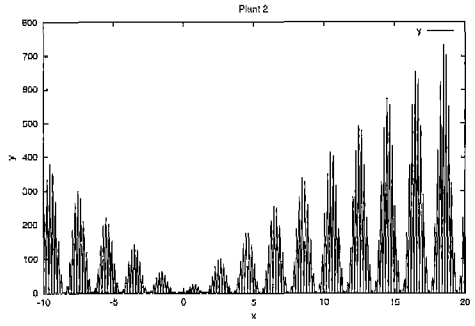
gradient information available.



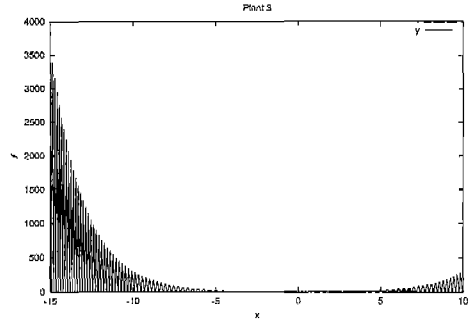
(a)



(b)

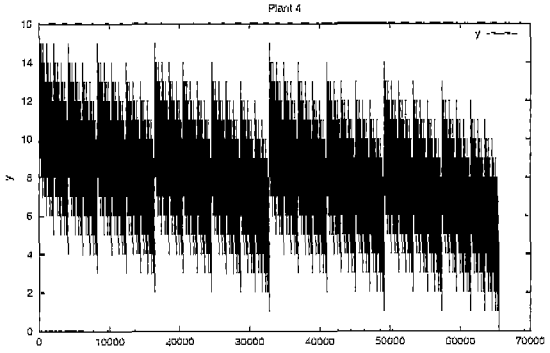


(c)

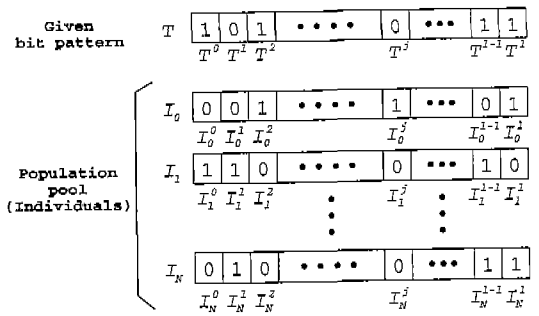


(d)

그림 1. 4개의 테스트 함수
Fig. 1. Four Test Functions.



(a)



(b)

그림 2. 패턴일치 문제
Fig. 2. One Pattern Matching Problem.

The performances of our scheme have been measured and compared to those of SGA with constant values of operator probabilities. For comparison of previous adaptive algorithm, the AGA[3] was also implemented and tested. The main idea of their method is that the operator probabilities must be adjusted by considering the fitness of each individual. In their method, the operator probabilities are adaptively adjusted at individual level by following equations.

$$p_c = \begin{cases} k_1(f_{\max} - f')/(f_{\max} - \bar{f}), & f' \geq \bar{f} \\ k_3, & f' > \bar{f} \end{cases} \quad (8)$$

$$p_c = \begin{cases} k_2(f_{\max} - f')/(f_{\max} - \bar{f}), & f' \geq \bar{f} \\ k_4, & f' < \bar{f} \end{cases} \quad (9)$$

where $f_{\max}$ and $\bar{f}$ are the maximum fitness and averaged fitness at a generation, f is the fitness of an individual, and f' is the large value of fitness values of two parents. First, we consider the

meaning of the equation 8 and 9 in the case of $f \geq \bar{f}$ or $f \geq \bar{f}$. The numerator term in such cases is to give more chances to be evolved to the individuals with low fitness and not to give chances to be destroyed to the individuals with best fitness. That is, the best individuals with $f = f_{max}$ or $f' = f_{max}$ are not destroyed by crossover and mutation since the p_c and p_m become to zero. On the other hands, the more the f' or f decrease, the more the operator probabilities increase. The denominator term is not to get stuck at a local optimum (in other words, not to fall into a premature convergence phenomenon). If a GA gets stuck at a local optimum, the operator probabilities will increase because the f or f' will be close to the f_{max}. This helps a GA escape from the local optimum. If f or f' equals to the $\bar{f}$, then the p_c and p_m become to one. For bounding the maximum values of p_c and p_m, k_1 and k_2 that must be set by a user are employed.

In the case of $f' < \bar{f}$ or $f < \bar{f}$, next, the p_c and p_m are set to the constant values k_3 and k_4, respectively. This is to give same chances to be evolved to the individuals with less fitness than $\bar{f}$. They set k_1 and k_3 to 1.0; and k_2 and k_4 to 0.5 for their experiments. The authors compared their method to the SGA with only one operator probabilities, *i.e.*, $p_c = 0.65$ and $p_m = 0.008$. Although their method have shown little improvement of performance than the SGA with the constant operator probabilities, it is not sure that their method is adequate for adaptation of operator probabilities. This is because the performance of SGA is very dependent on constant operator probabilities as shown in our experiments. It was revealed from our experiments that their method showed poor performance than the SGA with some constant probabilities as shown in Table III, V, and VII. To make matters worse, it was found that their equation 8 and 9 have a fatal problem in the case that the value of f_{max} is equal to $\bar{f}$. Thus, we modified the equation 8 and 9 for our experiments as $p_c = k_3$ and $p_m = k_4$, where $f_{max} = \bar{f}$. In the case of $f_{max} \neq \bar{f}$, the original equations was used. This modification does not greatly affect the performance of the AGA because the possibility this situation occurs is not much.

Except the adaptive scheme, all other parameters — initial individuals, the population size, and the length of bit strings — are same in experiments. Since the performances of GAs are dependent on the initial individuals[1~4], the performance of each experiment is measured by average values of 30 runs. To measure the performances according to change of population size and the length of bit strings, we set the parameters as shown in Table I.

표 I. 테스트 문제 파라메터
Table I. Parameters for Test Problems.

Experiments	population size	The length of bit strings
Exp. #1	50	16
Exp. #2	100	20
Exp. #3	200	24

The performances of SGA with constant operator probabilities are measured under the combinations of four crossover probabilities and four mutation probabilities, *i.e.* sixteen combinations of constant operator probabilities. Table II shows the selected crossover and mutation probabilities.

표 II. 사용한 교배와 돌연변이 확률
Table II. Selected Crossover and Mutation Probabilities.

p_c	0.5	0.6	0.7	0.8
p_m	0.001	0.005	0.01	0.2

The performances in each experiment are measured with three measures. Let the g_r be the first generation number where the GA finds the optimum solution at the r th run, then the performance indexes are defined as follows.

1) I : the averaged value of g_r for 30 runs, *i.e.*, $I=\sum_{r=1}^{30} g^r/30$.
2) σ_I : the standard deviation of g_r for 30 runs, *i.e.*, $\sigma_I=\sqrt{(g_r-I)^2/30}$.
3) D : the number of getting stuck at a local optimum for 30 runs. To measure this index, we set the maximum number of generations to 50,000.

The first measure I addresses the searching or optimizing ability of a GA and the second measure σ_I indicates the robustness against the changing of initial individuals. The possibility that a GA gets stuck at a local optimum is measured by third index. Table Ⅲ shows the experimental results of four function optimization problems and the pattern matching problem in Exp. #1. As already described above, the performances of SGA was measured under sixteen combinations of operator probabilities. For simplicity, only the best result of SGA (denoted as SGA*) was presented in Table Ⅲ. In SGA*, the p_c and p_m represent the operator probabilities that produce the best performance. In STGA and AGA, the values of operator probabilities for one run were measured only at the final generation by averaging the values of operator probabilities of each individual (we call these probabilities final operator probabilities). In Table Ⅲ, the p_c and p_m in STGA and AGA indicate the averaged values of final operator probabilities for 30 run. In order to show the performances of the SGA according to the operator probabilities, we select the two problems G_4 and G_5 as typical problems of function optimization and combinational optimization, respectively. Table Ⅳ shows the performances of the SGA under sixteen operator probabilities.

표 Ⅲ. 실험 #1에서의 실험결과
Table Ⅲ. Experimantal Results in Exp. #1.

Problem	Method	I	σ_I	D	p_c	p_m
G_1	STGA	11.80	2.88	0	0.651	0.096
	AGA	167.00	192.47	0	0.048	0.072
	SGA*	94.03	75.37	0	0.800	0.050
G_2	STGA	347.03	377.62	0	0.643	0.101
	AGA	3714.40	3483.43	0	0.380	0.262
	SGA*	237.97	239.99	0	0.600	0.050
G_3	STGA	43.73	51.23	0	0.655	0.083
	AGA	432.93	330.35	0	0.118	0.100
	SGA*	62.00	56.01	0	0.700	0.100
G_4	STGA	18.27	18.84	0	0.647	0.084
	AGA	175.47	152.61	0	0.098	0.084
	SGA*	34.50	33.84	0	0.800	0.050
G_5	STGA	7.30	1.72	0	0.651	0.093
	AGA	465.37	403.55	0	0.479	0.320
	SGA*	45.27	25.57	0	0.700	0.050

표 Ⅳ. 실험 #1에서 함수 G_4와 G_5의 SGA 실험 결과
Table Ⅳ. Experimantal Results in SGA for G_4 and G_5 in Exp. #1.

Problem	index	$p_m \backslash p_c$	0.5	0.6	0.7	0.8
G_4	I	0.001	3421.5	26209.0	33979.8	36558.10
		0.05	48.90	77.87	60.73	34.50
		0.01	42.97	49.73	52.13	58.23
		0.2	125.40	127.77	114.93	124.47
	σ_I	0.001	21705.6	22739.0	22845.0	21377.2
		0.05	41.08	97.72	65.45	33.84
		0.01	40.87	40.3	38.21	50.00
		0.2	95.55	106.10	138.13	110.32
	D	0.001	19	12	20	21
		0.05	0	0	0	0
		0.01	0	0	0	0
		0.2	0	0	0	0
G_5	I	0.001	59.27	48.13	53.90	45.30
		0.05	65.77	53.90	45.27	45.57
		0.01	128.23	124.13	98.07	122.53
		0.2	430.57	416.20	498.97	532.60
	σ_I	0.001	63.61	61.08	53.96	43.00
		0.05	42.40	31.35	25.57	44.51
		0.01	98.08	105.56	73.22	104.06
		0.2	361.12	331.38	536.87	367.79
	D	0.001	0	0	0	0
		0.05	0	0	0	0
		0.01	0	0	0	0
		0.2	0	0	0	0

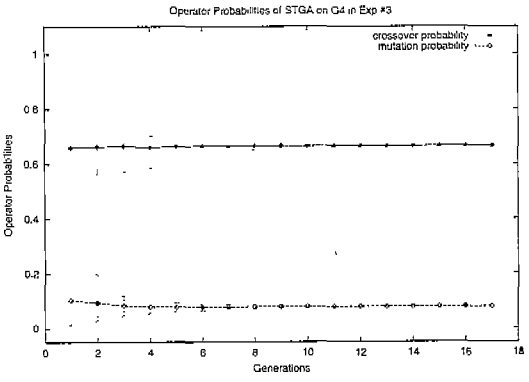
As shown in Table Ⅲ~Ⅶ, the performances of STGA are superior to those of AGA and mostly better than those of the SGA*. In the case of only G_2 in Exp. #1 and Exp. #2, the STGA shows a little worse performance than the SGA* This is caused that the G_2 has no gradient informations available that the STGA uses as described in section Ⅲ. From the experimental results, we observed that the STGA is more effective in case of the problems that have gradient informations. the STGA shows considerably better performance than the others. Especially in the case of G_1 in Exp. #3, the STGA was fast about 110 times than the AGA and about 170 times than the SGA*. As the population size and the length of bit strings increase, the performances of STGA except for the case of G_2 problem are more and more better than the others. This indicates that the STGA is more effective in the cases of more complicate parameters. From the σ_I performance index, we can know that the STGA is more robust against changing the initial individuals than the others in most problems.

표 Ⅴ. 실험 #2에서의 실험결과
Table Ⅴ. Experimantal Results in SGA for G_4 and G_5 in Exp. #1.

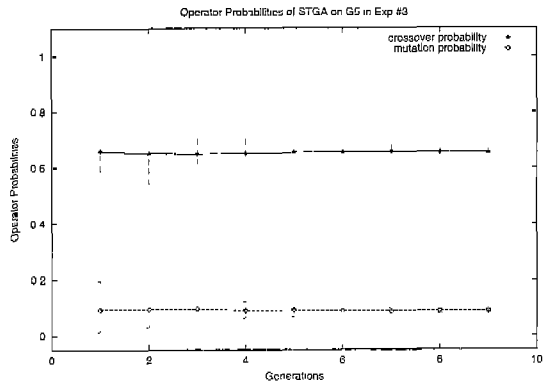
Problem	Method	I	σI	D	pc	pm
G_1	STGA	16.47	5.09	0	0.651	0.092
	AGA	905.30	1038.20	0	0.027	0.050
	SGA*	635.67	524.68	0	0.700	0.050
G_2	STGA	1571.57	1625.82	0	0.643	0.094
	AGA	18066.57	15192.36	0	0.286	0.222
	SGA*	1304.87	1728.57	0	0.700	0.050
G_3	STGA	43.93	21.06	0	0.651	0.083
	AGA	1146.00	1075.54	0	0.097	0.083
	SGA*	127.10	96.97	0	0.700	0.050
G_4	STGA	17.33	10.73	0	0.650	0.087
	AGA	297.47	282.85	0	0.041	0.041
	SGA*	96.87	67.19	0	0.800	0.050
G_5	STGA	8.97	1.54	0	0.648	0.097
	AGA	3232.63	2445.69	0	0.484	0.325
	SGA*	29.40	15.76	0	0.800	0.001

표 Ⅵ. 실험 #2에서 함수 G_4와 G_5의 SGA 실험 결과
Table Ⅵ. Experimantal Results in SGA for G_4 and G_5 in Exp. #2.

Problem	index	$p_m \backslash p_c$	0.5	0.6	0.7	0.8
G_4	I	0.001	29754.23	32455.23	31634.50	27287.17
		0.05	117.50	100.00	112.00	96.87
		0.01	192.13	231.33	120.00	221.73
		0.2	630.07	767.43	757.77	753.37
	σ_I	0.001	22491.21	22624.60	22785.89	23836.18
		0.05	86.91	57.96	74.16	67.19
		0.01	204.30	250.45	119.91	182.38
		0.2	592.39	672.36	724.14	562.05
	D	0.001	15	17	18	15
		0.05	0	0	0	0
		0.01	0	0	0	0
		0.2	0	0	0	0
G_5	I	0.001	40.73	37.90	35.43	29.40
		0.05	156.00	224.63	127.63	152.93
		0.01	853.57	791.17	721.23	821.30
		0.2	3663.37	5014.90	2675.70	2575.93
	σ_I	0.001	25.50	14.21	25.54	15.76
		0.05	123.58	169.83	106.59	106.92
		0.01	911.11	784.38	617.98	742.64
		0.2	3044.79	4803.73	2167.79	3208.93
	D	0.001	0	0	0	0
		0.05	0	0	0	0
		0.01	0	0	0	0
		0.2	0	0	0	0



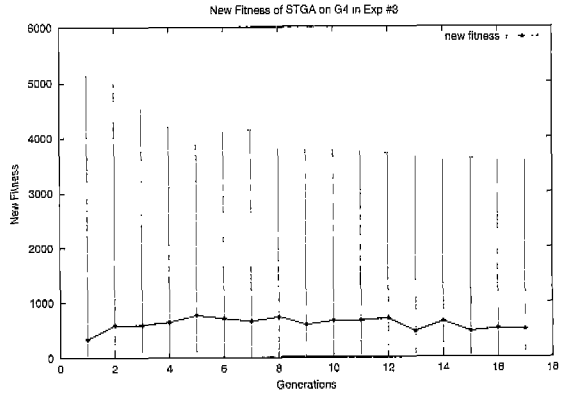
(a)



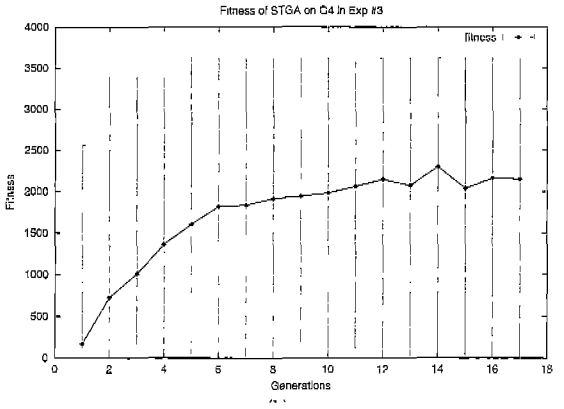
(b)

그림 3. 실험 #3에서 STGA의 연산자 확률 (a) G_4 (b) G_5

Fig. 3. Operator probabilities of STGA in Exp. #3 (a) G_4 (b) G_5.



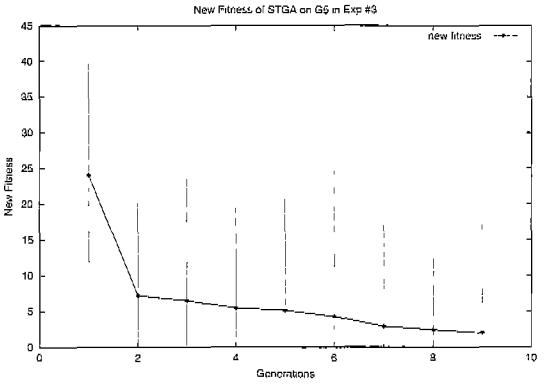
(a)



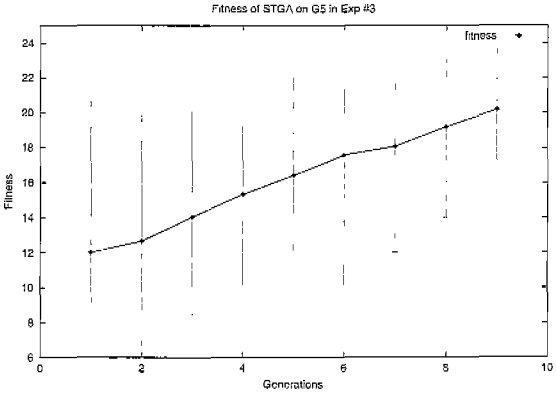
(b)

그림. 4. 실험 #3 G_4에서 STGA의 적합도 및 새 적합도 (a) 새 적합도 (b) 적합도

Fig. 4. Fitness and New Fitness of STGA on G_4 in Exp. #3 (a) New Fitness (b) Fitness.



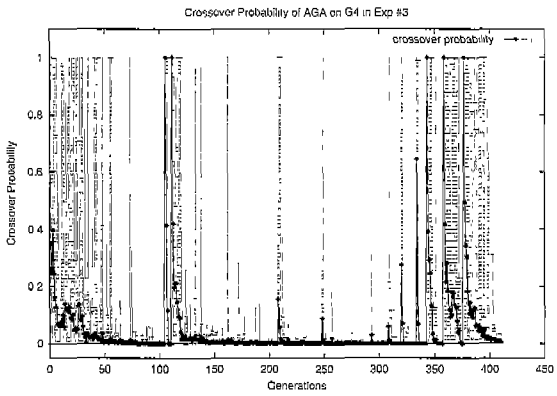
(a)



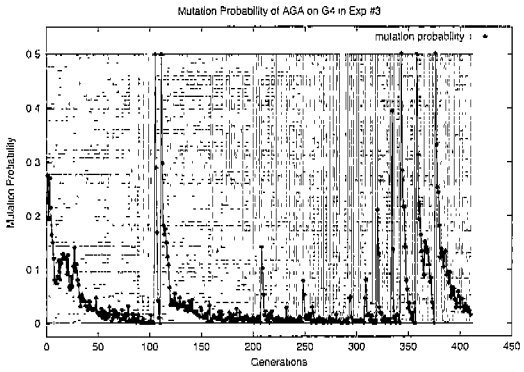
(b)

그림 5. 실험 #3 G_5에서 STGA의 새 적합도 (a) 새적합도 (b) 적합도

Fig. 5. Fitness and New Fitness of STGA on G_5 in Exp. #3 (a) New Fitness (b) Fitness.



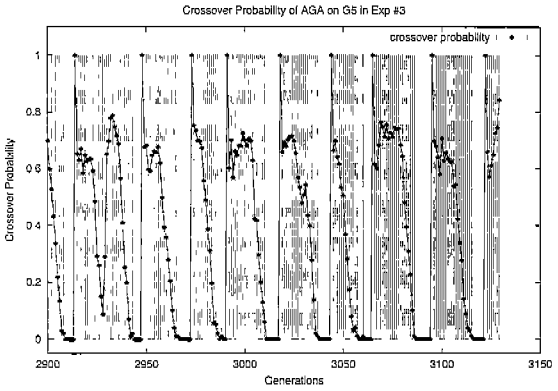
(a)



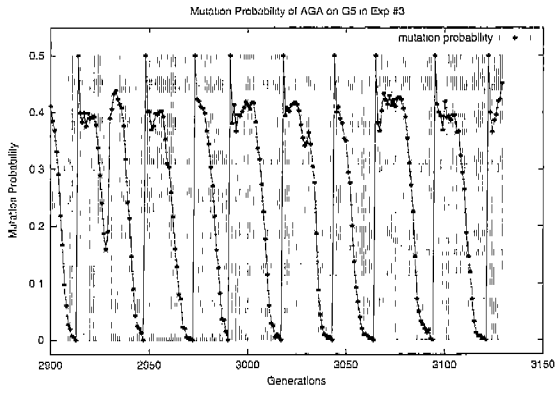
(b)

Fig. 6. 실험 #3 G_4에서 AGA의 연산자 확률 (a) 교배 (b) 돌연변이

Fig. 6. Operator probabilities of AGA on G_4 in Exp. #3 (a) Crossover (b) Mutation.



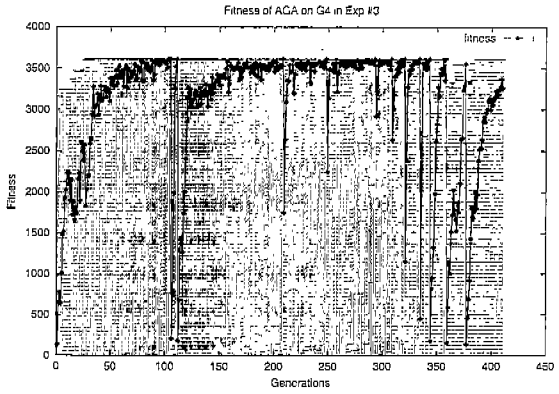
(a)



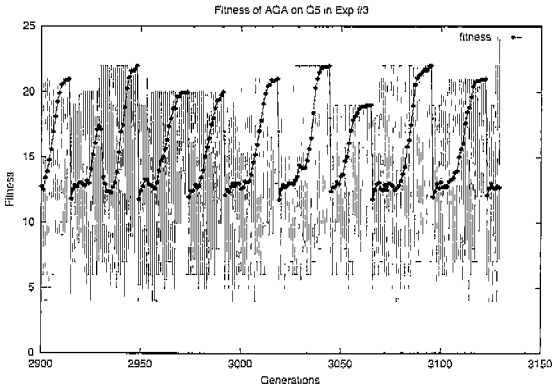
(b)

그림 7. 실험 #3에서 G_5에서 AGA의 연산자 확률 (a) 교배 (b) 돌연변이

Fig. 7. Operator probabilities of AGA on G_5 in Exp. #3 (a) Crossover (b) Mutation.



(a)



(b)

그림 8. 실험 #3에서 AGA의 적합도 (a) G_4 (b) G_5.

Fig. 8. Fitness of AGA in Exp. #3 (a) G_4 (b) G_5.

표 Ⅶ. 실험 #3에서의 실험결과

Table Ⅶ. Experimantal Results in Exp. #3.

Problem	Method	I	σI	D	pc	pm
G_1	STGA	20.27	3.13	0	0.647	0.089
	AGA	2282.47	1853.17	0	0.037	0.042
	SGA*	3511.73	3578.65	0	0.800	0.001
G_2	STGA	10719.50	10878.44	0	0.652	0.098
	AGA	42902.73	14737.31	0	0.162	0.139
	SGA*	12136.00	9977.11	0	0.600	0.050
G_3	STGA	29.33	32.98	0	0.647	0.085
	AGA	3385.60	4007.55	0	0.034	0.040
	SGA*	408.97	382.27	0	0.500	0.050
G_4	STGA	17.73	5.37	0	0.652	0.082
	AGA	926.00	671.41	0	0.016	0.019
	SGA*	301.63	344.84	0	0.600	0.050
G_5	STGA	9.20	1.19	0	0.652	0.096
	AGA	12558.00	11623.40	0	0.493	0.326
	SGA*	27.93	10.39	0	0.700	0.001

표 Ⅷ. 실험 #3에서 함수 G_4와 G_5의 SGA 실험 결과

Table Ⅷ. Experimantal Results in SGA for G_4 and G_5 in Exp. #3.

Problem	index	$p_m \backslash p_c$	0.5	0.6	0.7	0.8
G_4	I	0.001	24789.93	18689.17	27071.17	20844.07
		0.05	354.93	301.63	344.40	331.37
		0.01	979.37	942.30	1126.17	773.63
		0.2	5043.60	6724.87	4560.00	7484.87
	σ_I	0.001	23920.51	22566.07	24515.89	23856.13
		0.05	330.18	344.84	297.13	269.20
		0.01	1080.14	754.67	974.39	679.33
		0.2	7480.32	6338.18	4750.72	7796.72
	D	0.001	14	10	16	12
		0.05	0	0	0	0
		0.01	0	0	0	0
		0.2	0	0	0	0
G_5	1	0.001	34.67	37.90	35.43	29.40
		0.05	677.73	224.63	127.63	152.93
		0.01	6342.30	791.17	721.23	821.30
		0.2	18752.50	5014.90	2675.70	2575.93
	σ_I	0.001	12.74	7.85	10.39	7.51
		0.05	588.34	340.17	848.38	389.93
		0.01	5436.10	3865.95	4370.48	5770.58
		0.2	13523.16	13754.38	17600.19	17169.94
	D	0.001	0	0	0	0
		0.05	0	0	0	0
		0.01	0	0	0	0
		0.2	3	1	4	3

With the problems G_4 and G_5 in Exp. #3, we observed the operator probabilities and fitness for one run. Figure 3 shows the minimum, maximum, and average values of crossover probability and mutation probability of STGA on G_4 and G_5 in Exp. #3. In a generation, the upper bar and lower bar indicate the maximum and minimum values respectively and the mark represents the average value. As shown in Figure 3, the operator probabilities of individuals settle down to constant values in a few generations. During these generations, the STGA select individuals with good operator probabilities based on the values of new fitness. Figure 4 and 5 show the new fitness and fitness of STGA on G_4 and G_5 in Exp. #3, respectively. As shown in the figures, the average value of fitness for the most generations increase as the generations increase. In AGA, however, the operator probabilities are greatly varied from zero to one in case of crossover probability and from zero to 0.5 in case of mutation probability as shown in Figure 6 and 7. Moreover, the average values of fitness are greatly changed. Note that the operator probabilities and the fitness of G_5 (see Figure 7 nd Figure 8 (b) are closely related. In the generation near by 2910, the average values of crossover and mutation probabilities are nearly zero. This indicates that the AGA falls into a local optimum, but all individuals are not same. In this case, it is very difficult that the AGA gets out of the local optimum because most operator probabilities are nearly zero. After a few generations, all individuals become same. In this case, the crossover probabilities and the mutation probabilities of all individuals become to one and to 0.5, respectively. Since this makes all individuals of AGA disrupt, the average value of fitness rapidly decreases. From the viewpoint of average fitness, the average value of fitness when this occur is the nearly same as the first generation (see near by 110 generations in Figure 8 (a)). This represents that the AGA destroys nearly all informations found to escape the local optimum when the AGA falls into a local optimum. From these facts, we can conclude that the AGA is very difficult to escape the local optimum and pays a lot of cost to get out of the local optimum. This makes the performances of AGA poor.

V. Conclusion

This paper proposed a self-tuning scheme for adapting operator probabilities in genetic algorithms. In the proposed scheme, we assigned uniformly distributed random values within two limits to the operator probabilities of individuals and modified the

evaluation operation for evolving the operator probabilities as well as the solutions. By taking not only original fitness but also improved fitness in the evaluation operation, the operator probabilities as well as the solutions were co-evolved. From this, our scheme can be regarded as a combination method of GAs and gradient based search. With four function optimization problems and one combinational problem, we extensively experimented with the proposed scheme, an adaptive genetic algorithm (AGA)[3], and simple genetic algorithms (SGA) with constant operator probabilities. Although the proposed scheme did not have any additional operations and any additional parameters for evolution of operator probabilities, it was observed from extensive experiments that the proposed scheme was superior to the others and was very effective and useful especially in the problems with gradient informations available. The AGA method showed relatively poor performances because the method was very ineffective to get out of the local optimum. As future works, we will experiment with more complicated problems and will apply this scheme to the genetic algorithms with the other reproduction operators and to the other genetic algorithms such as steady state genetic algorithms.

References

[1] A. Tuson, "Adapting Operator Probabilities in Genetic Algorithms," Master thesis, Dept. of Artificial Intelligence, University of Edinburgh, UK, 1995.

[2] A. Tuson and P. Ross, "Adapting Operator Settings In Genetic Algorithms," *Evolutionary Computation*, vol. 6, no. 2, pp. 161~184, 1998.

[3] M. Srinivas and L. M. Patnaik, "Genetic Algorithms: A Survey," *IEE Computer Magazine*, pp. 17~26, June 1994.

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

[5] J. L. R. Filho and P. C. Treleaven, "Genetic Algorithm Programming Environments," *IEEE Computer Magazine*, pp. 28~43. June 1994.

[6] C. L. Karr and E. J. Gentry, "Fuzzy Control of pH Using Genetic Algorithms," *IEEE Trans. on Fuzzy Systems*, vol. 1, pp. 46~53, Jan. 1993.

[7] D. Beasley, D. R. Bull, and R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals," Technical Report, obtain from ftp://ralph.cs.cf.ac.uk/pub/papers/GAs/ga_overview[12].ps.

[8] M. Srinivas and L. M. Patnaik, "Adaptive Probabilities of Grossover and Mutation in Genetic Algorithms," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 24, pp. 656~667, Apr. 1994.

[9] D. B. Fogel, "An Introduction to Simulated Evolutionary Optimization," *IEEE Trans. on Neural Networks*, vol. 5, pp. 3~14, Jan. 1994.

[10] H. Szczerbika and M. Becker, "Genetic Algorithms: A Tool for Modelling, Simulation, and Optimization of Complex Systems," *Cybernetics and Systems: An International Journal*, vol. 29, pp. 639~659, Aug. 1998.

[11] R. Yang and I. Douglas, "Simple Genetic Algorithm with Local Tuning: Efficient Global Optimizing Technique," *Journal of Optimization Theory and Applications*, vol. 98, pp. 449·465, Aug. 1998.

[12] L. Davis, "Adapting Operator Probabilities Iin Genetic Algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, pp. 61~69, 1989.

[13] R. Hinterding, "Gaussian Mutation and Self-adaption in Numeric Genetic Algorithms," in *Proceedings of the 2nd IEEE International Conference on Evolutionary Computation*, pp. 384~389, 1995.

[14] R. Hinterding, Z. Michalewicz, and A. E. Eiben, "Adaptation in Evolutionary

Computation: A Survey," *in Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, pp. 65~69, 1997.

[15] M. C. Sinclair, "Operator-probability Adaptation in a Genetic-algorithm/Heuristic Hybrid for Optical Network Wavelength Allocation," in *proc. IEEE Intl. Conf. on Evolutionary Computation (ICEC'98), Anchorage, Alaska, USA*, pp.840~845, 1998.

[16] C. W. Ho, K. H. Lee, and K. S. Leung, "A Genetic Algorithm Based on Mutation and Crossover with Adaptive Probabilities," in *Proceedings of the 1999 Congress on Evolutionary Computation,* vol. 1, pp.768~775, 1999.

[17] s. H. Jung, "Two Dimensional Evaluation Scheme in Genetic Algorithms," *Journal of Electrical Engineering and Information Science,* vol. 4, pp.561~570, Oct. 1999.

저 자 소 개

鄭 成 勳(正會員)

1966년생. 1988년 2월 한양대학교 전자공학과 학사. 1991년 2월 한국과학기술원 전기 및 전자공학과 공학석사. 1995년 2월 한국과학기술원 전기 및 전자공학과 공학박사. 1995년 3월~1996년 2월 한국과학기술원 전기및전자공학과 위촉연구원. 1996년 3월~1998년 2월 한성대학교 정보전산학부 전임강사. 1998년 3월~현재 한성대학교 정보전산학부 조교수. 주 관심분야는 지능시스템, 퍼지 및 신경망, 진화연산, 지능제어, 패턴인식, 시뮬레이션 등임