International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:8, 2009

Selective Mutation for Genetic Algorithms

Sung Hoon Jung

Abstract—In this paper, we propose a selective mutation method for improving the performances of genetic algorithms. In selective mutation, individuals are first ranked and then additionally mutated one bit in a part of their strings which is selected corresponding to their ranks. This selective mutation helps genetic algorithms to fast approach the global optimum and to quickly escape local optima. This results in increasing the performances of genetic algorithms. We measured the effects of selective mutation with four function optimization problems. It was found from extensive experiments that the selective mutation can significantly enhance the performances of genetic algorithms.

Keywords—Genetic algorithm, selective mutation, function optimization

I. Introduction

Genetic algorithms (GAs), robust and systematic optimization paradigms, have been successfully applied to many scientific and engineering problems [1–11]. Their performances, however, have been considerably limited by some problems, typically premature convergence problems [5, 12]. Most individuals in a prematurely converged situation are located at some local optimum areas and they can't get out of the local optimum areas because the exploration power of mutation is low. If we increase the exploration power by setting the mutation probability to high, then the speed of convergence to global optimum areas also becomes slow. Therefore, we need a smart method to cope with this problem.

Some methods to solve this problem have been introduced to date [13–15]. In this paper, we propose another method to alleviate this premature convergence phenomenon by additionally selective mutation of individuals according to their ranks. We assume that if an individual has low rank, then it is far from the global optimum. Thus, we additionally mutate the most significant part of individual's string. Whereas if an individual has high rank, then we regard it to be near the global optimum and mutate the least significant part of the individual's string. Since this is not always true, the additional, selective mutation of individuals does not always help the genetic algorithms. The probability that the selective mutation helps the genetic algorithms, however, is larger than the probability that it prevents genetic algorithms from approaching the global optimum. As a result, this selective mutation allows genetic algorithms to increase their performances.

In order to measure the effects of selective mutation, we experimented our genetic algorithm with selective mutation using four function optimization problems that have been typically used to date. From the results, we can find that the selective mutation can considerably contribute the search capability of genetic algorithms. This selective mutation can also be simply incorporated into the various types of genetic algorithm without any large modification.

Sung Hoon Jung is with the Department of Information and Communication Engineering, Hansung University, South Korea, e-mail: shjung@hansung.ac.kr

This paper is organized as follows. Section II describes proposed selective mutation for genetic algorithms. The experiments with four function optimization problems and their results are given in section III. This paper concludes in section IV

II. Selective Mutation for Genetic Algorithms

Most population-based, reproductive, optimization algorithms such as genetic algorithms, ant colony optimization, and particle swarm optimization had a critical problem called *premature convergence problem* [12, 13, 15]. This problem occurs when highly fit parents in a population pool breed many similar offsprings in the early evolution time. If the highly fit individuals are local optima areas, then newly generated offsprings from the parents are also near the local optima areas. The crossover, one of the operations of genetic algorithms, can not generate quite different offsprings from their parents because it uses only acquired information. Thus, we regard the crossover as an exploitation operation. The mutation, an exploration operation, can search new areas in contrast to the crossover, but it can also not change many bits in the individuals because the mutation rate is too low. If we set the mutation rate to high value, then genetic algorithms approach the global optimum very slowly. As a result, it is very difficult for genetic algorithms to escape this premature convergence problem. This considerably makes the performances of genetic algorithms degrade.

In this paper, we introduce selective mutation for solving this problem. In most function optimization problems, their input variables are encoded into the binary strings of individuals. Since the binary strings represent binary numbers for each variable, the higher the bit position of string is, the larger the bit weight has. From this, we think that it is helpful to mutate some part of strings of individuals according to their fitness. That is, if an individual has low fitness, then we mutate the most significant part in order to largely change because we regard the individual to be far from the global optimum. Otherwise, we mutate the least significant part in order to do fine tuning because we think that the individual has high probability to be near global optimum. This is why we call our method *selective mutation*. Algorithm 1 shows our genetic algorithm with selective mutation.

Algorithm 1 Genetic Algorithm with Selective Mutation
// t : time //
// n : population size //
// s : string length //
// N : the number of grades //
// $P^r_{1,\cdots,N}$: the part of rank //
// $P^s_{1,\cdots,N}$: the part of string //
// r_i : the rank of ith individual //

```
    // P : populations //
1   t ← 0
2   initialize P(t)
3   evaluate P(t)
4   while (not termination-condition)
5   do
6       t ← t + 1
7       select P(t) from P(t − 1)
8       recombine P(t)
9           do crossover
10          do normal mutation
11      evaluate P(t)
12      do selective mutation (▲)
13          sort P(t) with fitness and rank
14          divide the strings into N parts, P^s
15          divide the rank into N parts, P^r
16          for i = 1 to n
17              if r_i ∈ P^r_j
18                  do mutation a randomly selected bit in the part,
                    P^s_{N−j+1}
19              end if
20          end for
21  end
```

As shown in the Algorithm 1, genetic algorithms are composed of four key processing, initialize $P(t)$, evaluate $P(t)$, select $P(t)$, and recombine $P(t)$. We add the 'do selective mutation' to the original genetic algorithm (see ▲). Selective mutation is composed of four processes: 1) sort population $P(t)$ with fitness and rank individuals according to the fitness, 2) divide the string of individuals into N parts corresponding to the bit positions, 3) divide the rank into N parts, and 4) mutate a randomly selected bit in the string part, P^s_{N-j+1}, if the rank of each individual is in the rank part P^r_j.

For example, assume that the population size n is 20, the string length s is 16, and the number of grade N is 2. Then, the strings are divided to two parts $P^s_{1,2}$ and the rank is divided to two parts $P^r_{1,2}$. If the rank of an individual is 3, then it belongs to rank part P^r_1, finally one bit randomly selected in the string part P^s_2 is mutated. Since the string part P^s_2 consists of least significant eight bits, the mutation of one bit in the part changes a little distance from its original position in the phenotype variable. In other case, if an individual belongs to P^r_2, then one bit randomly selected in the string part P^s_1 is mutated. This implies that bad individuals are changed very much because the string part P^s_1 is composed of most significant eight bits. This is additional effect because this selective mutation is additionally performed after the normal mutation. This selective mutation can make genetic algorithms fast approach to the global optimum and quickly get out of premature convergence. As a result, it will increase the performances of genetic algorithms.

III. Experimental Results

In order to measure of performances, our selective mutation was tested on typical four function optimization problems that have been largely used at previous papers [12, 15, 16]. The four functions are given in Equation 1.

$$\begin{aligned} f_1 &= 3000 - 3(x^2 + y^2) \\ f_2 &= 0.5 - \frac{sin(\sqrt{x^2+y^2}) sin(\sqrt{x^2+y^2}) - 0.5}{(1.0 + 0.001(x^2+y^2))(1.0 + 0.001(x^2+y^2))} \\ f_3 &= (x^2 + y^2)^{0.25} sin(50(x^2 + y^2)^{0.1} + 1)^2 \\ f_4 &= 100(x^2 - y)^2 + (1 - x)^2 \end{aligned} \quad (1)$$

Functions $f_1 \sim f_4$ are a simple function, a Mexican hat function, a Shafer function 2, and DeJong function 2, respectively. Figure 1 shows the input-output relations of four functions. Function f_1 is relative simple in that it has only one global optimum at the $(0, 0)$ point. Unlike the function f_1, function f_2, Mexican hat function, has many local optima around the global optimum at the $(0, 0)$ point. The local optima act as an obstacle for genetic algorithms to approach the global optimum. On the other hands, function f_3 has multiple optima at four peaks near $((-10, -10)$, $(-10, 10)$, $(10, -10)$, $(10, 10))$ and its value is about 14.3. DeJong function 2 (f_4) has only one global optimum at $(-2.048, -2.048)$ point. Although it is viewed as simple, but it is not easy because there are a lot of local optimum in the direction of y axis near the global optimum and opposite side.

We experimented our method using the typical parameters as shown in Table I. Since four functions have x and y axes,

TABLE I
Parameters for experiments

Parameters	Values
Selection method	roulette wheel selection
Crossover probability (p_c)	0.6
Mutation probability (p_m)	0.05
Population size	10,100
Individual length	16, 22 bits
Number of grades	2
Number of runs	100

TABLE II
Experimental results

Ind. len.	16 bits			
pop. size = 10	ORG		SM	
function	avg.	dev.	avg.	dev.
f_1	2073.95	2373.47	808.02	896.85
f_2	1931.08	2132.20	1423.86	1453.60
f_3	262.14	236.97	142.74	139.41
f_4	1706.78	2406.69	623.34	850.50
Ind. len.	22 bits			
pop. size = 10	ORG		SM	
function	avg.	dev.	avg.	dev.
f_1	96835.90	113165.78	50375.16	60475.82
f_2	83134.31	88359.47	65109.63	68910.11
f_3	4564.23	4472.40	2548.27	2736.64
f_4	10318.83	9848.34	5136.48	5568.53
Ind. len.	16 bits			
pop. size = 100	ORG		SM	
function	avg.	dev.	avg.	dev.
f_1	173.51	168.17	57.74	50.37
f_2	200.49	197.36	131.68	142.39
f_3	20.30	19.93	15.73	13.76
f_4	8418.19	16506.74	341.04	756.38
Ind. len.	22 bits			
pop. size = 100	ORG		SM	
function	avg.	dev.	avg.	dev.
f_1	9277.01	8785.41	3581.25	2990.29
f_2	9070.89	11252.53	6508.52	5401.40
f_3	232.09	199.69	177.04	199.83
f_4	62486.01	133464.30	6869.73	13486.20

16 bits of individual length mean that each axis has 8 bits resolutions. The number of grades is set to 2, therefore, if the rank of an individual is higher than the half of individuals, a bit in the half of least significant strings is randomly selected and mutated and vice versa. In order to show the statistical results,

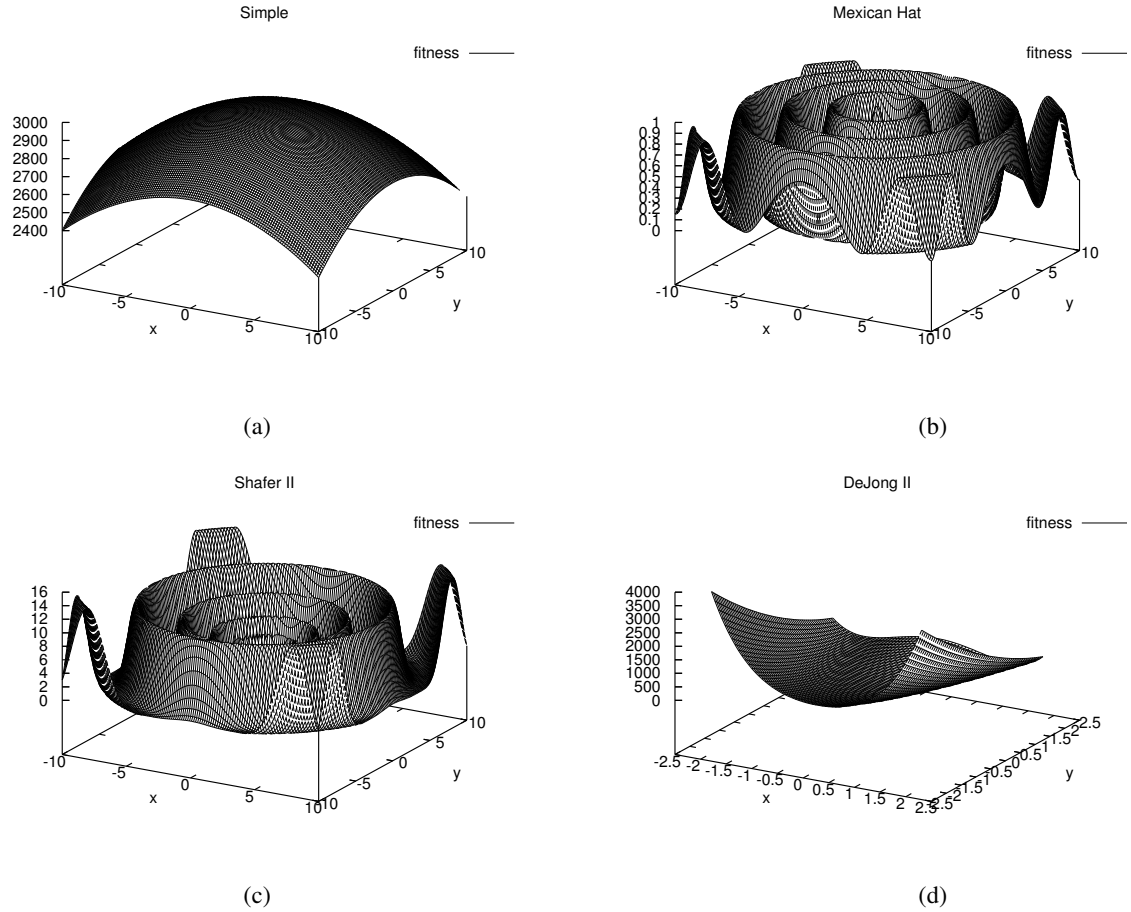

Fig. 1. Experimental functions (a) f_1 (b) f_2 (c) f_3 (d) f_4)

we executed the 100 runs for each experiment because the performance of genetic algorithms depends on the initial seed number of random functions. The total results are summarized at Table II. We recorded the generation number when at least an individual in the pool found the global optimum. In Table II, ORG and SM mean the methods of using only normal mutation and using normal and additional selective mutation, respectively; and the avg. and dev. are the average and standard deviation of 100 runs, respectively. As shown in the table, our selective mutation increases the performances of genetic algorithms for all functions, especially for the function of f_4. The ratio of enhanced performances is very similar to both cases of 16 bits and 22 bits. For examples, the function f_1 finished about twice faster at both cases.

In the case that population size is 100, genetic algorithms find the global optimum more faster than the case that population size is 10 except for the case of function f_4. It is very natural because if large individuals concurrently find the global optimum, then the probability to approach global optimum will increase. We wonder why genetic algorithms slowly find the global optimum even if the number of individual increases in the case of function f_4. Our observation is that since the local optimum area near $(2.048, -2.048)$ is quite broad in the case of function f_4, genetic algorithms easily fall in the area and can not easily get out of the area. This results in making the genetic algorithms stay long at the local optimum area. In the case of large individuals, some dominant individuals at the local optimum area are dominantly selected as parents and dominantly generate offsprings near them. If genetic algorithms fall in the local optimum area, it is hard to get out of the area. The more the number of individuals, the better the genetic algorithms are hard to get out of the area. This is because in the case of small number of individuals, some individuals that are quite changed from their parents can be located near global optimum area and can be dominant for getting out of the local optimum area. On the other hand, even if some individuals are quite changed to the global optimum area, it is hard for the individuals to dominate in the case of the large number of individuals. Since the probability that the roulette wheel selection selects the quite changed individuals located at the global optimum area becomes low when the number of individuals are large. As opposite case, if some individuals located in global optimum area are dominant, then the genetic algorithms can find the global optimum fast. Thus, the standard deviation of the results in the case of function f_4

is considerably large.

As shown in the Table II, this problem is very diminished in our method. In 16 bits, the result of ORG in the 100 individuals is worse about five times than that in the 10 individuals, but the result of SM is better about two times. In 22 bits, the ORG shows bad results of six times in 100 individuals, but the SM shows really similar results to those of the 10 individuals. From this we can conclude that our method helps the genetic algorithms get out of local optimum and results in increasing the performances.

IV. Conclusion

In this paper, we introduced a new additional mutation termed selective mutation for enhancing the performances of genetic algorithms. The selective mutation additionally changes a bit in the specific part of strings of individuals according to their ranks. If the ranks of individuals are high, then a bit in the least significant part is changed, and vice versa. This helps genetic algorithms fast approach to global optimum and escape local optima in the case that individuals fall in premature convergence. This two effects make genetic algorithms find global optimum more quickly and result in increasing the performances of genetic algorithms. Our selective mutation will be easily incorporated into the other types of genetic algorithms in order to increase their performances.

References

[1] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison–Wesley, 1989.
[2] M. Srinivas and L. M. Patnaik, "Genetic Algorithms: A Survey," *IEEE Computer Magazine*, pp. 17–26, June 1994.
[3] J. L. R. Filho and P. C. Treleaven, "Genetic-Algorithm Programming Environments," *IEEE Computer Magazine*, pp. 28–43, June 1994.
[4] D. Beasley, D. R. Bull, and R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals," Technical Report obtained from http://home.ifi.uio.no/∼jimtoer/GA_Overview1.pdf.
[5] D. B. Fogel, "An Introduction to Simulated Evolutionary Optimization," *IEEE Transactions on Neural Networks*, vol. 5, pp. 3–14, Jan. 1994.
[6] H. Szczerbicka 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.
[7] 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.
[8] C. Xudong, Q. Jingen, N. Guangzheng, Y. Shiyou, and Z. Mingliu, "An Improved Genetic Algorithm for Global Optimization of Electromagnetic Problems," *IEEE Transactions on Magnetics*, vol. 37, pp. 3579–3583, Sept. 2001.
[9] J. A. Vasconcelos, J. A. Ramirez, R. H. C. Takahashi, and R. R. Saldanha, "Improvements in Genetic Algorithms," *IEEE Transactions on Magnetics*, vol. 37, pp. 3414–3417, Sept. 2001.
[10] E. Alba and B. Dorronsoro, "The exploration/exploitation tradeoff in dynamic cellular genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 9, pp. 126–142, Apr. 2005.
[11] V. K. Koumousis and C. Katsaras, "A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance," *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 19–28, Feb. 2006.
[12] J. Andre, P. Siarry, and T. Dognon, "An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization," *Advances in engineering software*, vol. 32, no. 1, pp. 49–60, 2001.
[13] J. E. Smith and T. C. Fogarty, "Operator and parameter adaptation in genetic algorithms," *Soft computing : a fusion of foundations, methodologies and applications*, vol. 92, no. 2, pp. 81–87, 1997.
[14] 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.
[15] S. H. Jung, "Queen-bee evolution for genetic algorithms," *Electronics Letters*, vol. 39, pp. 575–576, Mar. 2003.
[16] K. DeJong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

Sung Hoon Jung He is a professor in the Department of Information and Communication Engineering, Hansung University. He received his B.S degree from Hanyang University, Korea, in 1988 and M.S. and Ph.D. degrees from KAIST, in 1991 and 1995, respectively. His research interests are in the field of intelligent systems and he recently starts working on systems biology. Dr. Jung is a member of KIIS and IEEK.