International Journal of Fuzzy Logic and Intelligent Systems, vol. 5, no. 4, Dec 2005 pp. 310-315

Fast Optimization by Queen-bee Evolution and Derivative Evaluation in Genetic Algorithms

Sung Hoon Jung*

*** Department of Information and Communication Engineering, Hansung Univ.**

Abstract

This paper proposes a fast optimization method by combining queen-bee evolution and derivative evaluation in genetic algorithms. These two operations make it possible for genetic algorithms to focus on highly fitted individuals and rapidly evolved individuals, respectively. Even though the two operations can also increase the probability that genetic algorithms fall into premature convergence phenomenon, that can be controlled by strong mutation rates. That is, the two operations and the strong mutation strengthen exploitation and exploration of the genetic algorithms, respectively. As a result, the genetic algorithm employing queen-bee evolution and derivative evaluation finds optimum solutions more quickly than those employing one of them. This was proved by experiments with one pattern matching problem and two function optimization problems.

Key words : Genetic Algorithms, Queen-bee Evolution, Derivative Evaluation, Premature convergence, Optimization

1. Introduction

Genetic Algorithms (GAs) have been applied to many scientific and engineering problems including even those that conventional methods could not solve [1-9]. Their performances mainly depend on encoding schemes, recombination and evaluation operations, parameters of operations, and the algorithms themselves [1,3,6,7,10]. In order to improve their performances; researchers have focused on two main streams, adapting operator probabilities of GAs [9,11-17] and modifying crossover and mutation operators [13,18-20].

We have introduced a new direction for improving the performances of GAs in terms of evolution [20] and evaluation [21]. The queen-bee evolution [20] makes GAs fast evolve to the fittest individual called queen-bee and also fast search new areas by strong mutation. The derivative evaluation [21] allows GAs to focus on the rapidly evolved individuals. The genetic algorithm employing queen-bee evolution (termed QGA) [20] was maximally about 1,000 times faster than the simple genetic algorithms (termed SGA) [1]. The derivative evaluation [21] also increases the performances of GAs.

In this paper, we fuse both queen-bee evolution and derivative evaluation into genetic algorithm for fast optimization (termed NGA). The queen-bee evolution allows the NGA to fast approach to the optimal solutions and also enables the NGA to control the premature convergence probability by strong mutation rates. Moreover, the derivative evaluation helps the NGA to generate good offsprings by focusing the rapidly evolved individuals like gradient based search algorithms. Finally, synergy effects of the two operations make it possible for GAs to find the global optimum quickly.

The NGA is tested on one combinational problem and two function optimization problems used in [20,22] with various parameter set. It was shown from experiments that the NGA was superior to SGA and was better than that of employing only queen-bee evolution. This result indicates that the queen-bee evolution and derivative evaluation cooperate with each other for upgrading the optimization capability of GAs.

This paper is organized as follows. Section 2 reviews the queen-bee evolution and derivative evaluation methods. Section 3 describes proposed fusion algorithms of queen-bee evolution and derivative evaluation. In section 4, experimental parameters and results are discussed. This paper concludes in section 5.

2. Queen-bee evolution and derivative evaluation

In queen-bee evolution, all individuals selected as parents for generating next generation are crossbred with the queen-bee, the fittest individual in a generation [20]. This increases the exploitation of genetic algorithms and also increases the possibility of premature convergence [5,9]. In order to prevent the premature convergence, exploration must also be strengthened [20]. The degree of exploration is controlled by the ξ rate between the number of normally mutated individuals and the number of strongly mutated individuals. The normally mutated individuals are mutated with the normal mutation probability p_m and the strongly mutated individuals are with the strong mutation probability $p_m'(\gg p_m)$, respectively. if $\xi=1$, for example, then all individuals are mutated with normal mutation probability. $\xi=1$ means lowest exploration and $\xi=0$ means highest

Manuscript received Nov. 7, 2005; revised Dec. 5, 2005.
This research was financially supported by Hansung University in the year of 2005.

exploration. The ξ value must be carefully selected according to the problems. More detailed description about queen-bee evolution is available in [20].

Derivative evaluation [21] is devised for enhancing the optimization speed of genetic algorithms. In derivative evaluation, i th individual is evaluated as:

$$f_i(t) = \begin{cases} f_i^D(t) & \text{if } f_i^D(t) > 0 \\ 0 & otherwise, \end{cases} \quad (1)$$

where $f_i(t)$ is new fitness of ith individual and $f_i^{D(t)}$ is derivative fitness of ith individual. The derivative fitness is defined as:

$$f_i^D(t) = f_i^o(t) - f_i^p(t) = f_i^o(t) - \frac{f_j^o(t-1) + f_k^o(t-1)}{2},$$

where $f_i^o(t)$ is original fitness evaluated by a fitness function and $f_i^p(t)$ is the parent fitness, which is calculated by average value of the original fitness of j and k parents. If an individual is not evolved than their parents, its new fitness becomes zero as shown in equation (1). This property makes genetic algorithms focus on the evolved individuals and helps genetic algorithms approach to optimum solutions fast. It, however, also increases the possibility that genetic algorithms falls into premature convergence. But, under premature convergence, this property helps genetic algorithms get out of the local optimum by making the genetic algorithms focus on newly evolved areas not the local optimum area. When all individual have zero fitness (this occurs when genetic algorithms fall into premature convergence), genetic algorithms randomly select parents because roulette wheel selection is not possible in this condition and random selection more or less helps genetic algorithms get out of the premature convergence. Derivative evaluation provides a gradient based search effect to genetic algorithms and this effect makes individuals be evolved more quickly to optimum solutions.

3. Proposed genetic algorithm

Queen-bee evolution and derivative evaluation are fused into a genetic algorithm as shown in Algorithm 1.

Algorithm 1 Proposed genetic algorithm

```
// t : time //
// n : population size //
// P : populations //
// ξ : normal mutation rate ( 1 − ξ: strong mutation rate) //
// p_m : normal mutation probability //
// p_m' : strong mutation probability //
// I_q : a queen-bee individual //
// I_m : selected individuals //
1 t ← 0
2 initialize P(t)
3 evaluate P(t) (♣)
4     calculate original and parents fitness
5     calculate new fitness
6 while (not termination-condition)
7 do
8     t ← t + 1
9     select P(t) from P(t−1) (♠)
10        P(t) = {(I_q(t−1), I_m(t−1))}
11    recombine P(t)
12        do crossover
13        do mutation (♠)
14            for i = 1 to n
15                if i≤(ξ×n)
16                    do mutation with p_m
17                else
18                    do mutation with p_m'
19                end if
20            end for
21    evaluate P(t) (♣)
22        calculate original and parents fitness
23        calculate new fitness
24 end
```

In Algorithm 1, ♣ lines indicate derivative evaluation and ♠ lines imply queen-bee evolution. All the others except for these two modified operations are the same as those of SGA. Initial individuals generated by a random function consist of a population pool. They are evaluated by derivative evaluation (show ♣ lines) and assigned with new fitness. In selection (show lines 9 and 10), the new fitness of individuals is used for selecting parents ($I_m(t-1)$ in Algorithm 1) for generating next generation. However, the queen-bee $I_q(t-1)$ is selected by original fitness of individuals. Therefore, recombination is done between the fittest individual $I_q(t-1)$ and the rapidly evolved individual $I_m(t-1)$. This makes it possible for the genetic algorithm to fast evolve the individuals. However, this does not always show a positive effect. In many cases, this can also make genetic algorithms fall into premature convergence. In order to decrease the probability of premature convergence, strong mutation with large mutation rate p_m' is applied to the individual with the ξ rate (show the 14 to 20 lines in Algorithm 1). Like the original derivative evaluation, if new fitness of all individuals is zero, then the $I_m(t-1)$ is randomly selected. Even though the $I_m(t-1)$ is randomly selected, it does not make offsprings degrade, because most individuals in this condition are sufficiently evolved. Reversely, this can help the genetic algorithm not to fall into local optimum.

4. Experimental Results and Discussion

The proposed method was tested on one combinational problem and two function optimization problems used in [20,22]. These three functions are given as follows.

$$f_1 = \sum_{j=1}^{h} m_j \begin{cases} m_j = 1 & \text{if } T_j = I_j \\ m_j = 0 & \text{if } T_j \neq I_j \end{cases}$$

$$f_2 = 100(x_1^2 - x_2)^2 + (1 - x_1)^2, \quad where -2.048 \le x_i \le 2.048$$

$$f_3 = 0.5 - \frac{\sin(\sqrt{x_1^2 + x_2^2})\sin(\sqrt{x_1^2 + x_2^2}) - 0.5}{(1.0 + 0.001(x_1^2 + x_2^2))(1.0 + 0.001(x_1^2 + x_2^2))}, \quad where -10 \le x_i \le -10 \qquad (2)$$

In equation 2, f_1 is a bit pattern matching problem between the target pattern T and an individual's pattern I; and f_2 and f_3 are DeJong function 2 and Mexican hat function, respectively. In f_1, each individual is evaluated by the number of matching bits between the individual I and target T. Therefore, optimum fitness is same as the number of bits h. Figure 1 shows the input-output relations of three functions.

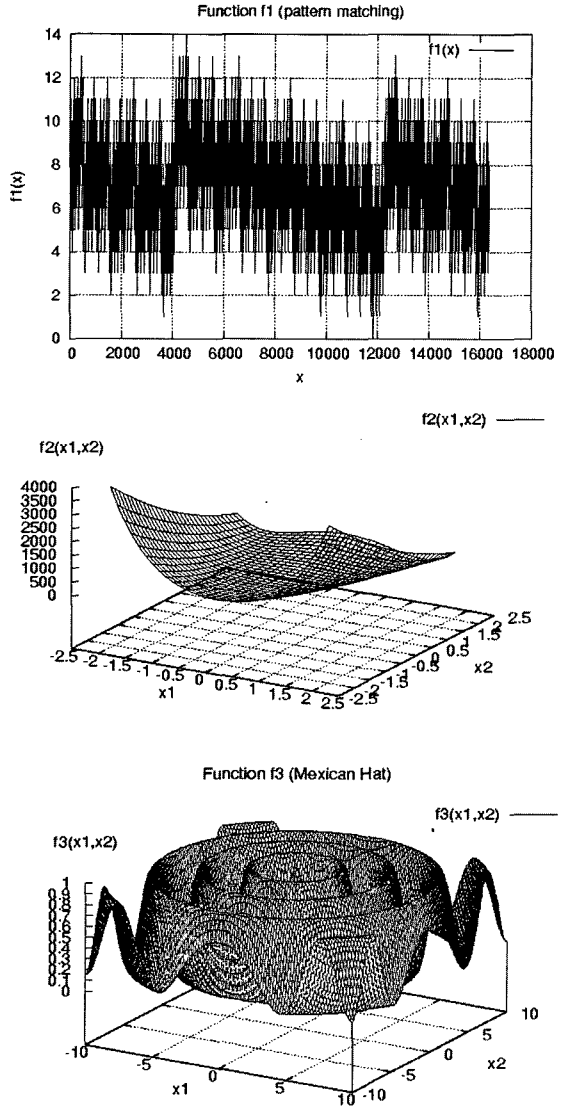

Figure 1 : Experimental Functions

We experiment with these three optimization functions using SGA, QGA, and NGA on the parameter set as shown in Table 1.

Table 1: Parameters for experiments

Parameters	values
crossover probability (p_c)	0.6
Normal mutation probability (p_m)	0.05
population size	10
Individual length	18, 24 bits
Normal mutation rate (ξ)	0.4, 0.6, 0.8, 1.0
Strong mutation probability (p_m')	0.6, 0.8, 1.0

The performances of NGA have been measured and compared to those of SGA and QGA. Except the algorithms, all other parameters---initial individuals, the probabilities of crossover and mutation, the population size, and the length of bit strings---are same in all experiments. Table 2 and 3 show experimental results in case that the individual length is 18 bits and 24 bits, respectively. Since GAs generally show somewhat different results according to the initial individuals, we measured experimental results with average values of 10 runs using different random number seeds. When GAs find the optimal solution, the number of generations at that generation is recorded and the results of 10 runs are averaged. This averaged value is an experimental result for one experiment. In table 2 and 3, the best results of QGA and NGA are marked by asterisk. In relatively simple problem f_1 in that the number of local optima are small and the local optima are not distributed to wide ranges, strong exploitation by queen-bee evolution and weak exploration by $\xi = 1$ show the best result at QGA as shown in the table 2 and 3. Considerably complex functions such as f_2 and f_3 functions, however, need strong mutation for preventing from premature convergence. Since the mutation rate ξ and strong mutation probability p_m' affects to the performances of GAs in most cases, these values must be carefully selected for best performances. We will study the selection method of ξ and p_m' for enhancing performances of GAs as a further work. The last four rows of each table showed the performances of SGA and the performance comparison between QGA and NGA. In all experiments, the performances of QGA and NGA are superior to those of SGA. Most performances of NGA are better than those of QGA except for some cases. The performance indexes are the minimum generation ratio (min. / min.), the average generation ratio (avg. / avg.) and the average generation ratio except for the result on $\xi = 1$ (avg. † / avg. †). In most optimization problems except for some simple problems, the parameter setting of $\xi = 1$ is not recommended because it can make GAs fall into premature convergence. Thus, the (avg. † / avg. †) measure is better than the (avg. / avg.) measure in most optimization problems. In our experiment, the function f_1 is relatively simple

Table 2: Experimental results (18 bits)

ξ	p_m	f_1		f_2		f_3	
		QGA	NGA	QGA	NGA	QGA	NGA
0.4	0.6	846.6	140.1	3008.0	528.1	27656.2	15246.0
	0.8	772.4	162.2	1197.9	523.6	11661.4	12042.4
	1.0	145.2	63.0	105.1	(*) 65.5	9143.6	8694.7
0.6	0.6	119.6	31.5	247.9	119.9	14792.7	5896.7
	0.8	117.8	28.5	213.1	150.5	5161.3	6314.1
	1.0	66.0	33.5	108.2	68.7	(*) 2738.5	4258.0
0.8	0.6	27.8	(*) 15.8	319.3	517.7	7382.7	2279.7
	0.8	42.8	22.0	82.2	338.1	4749.8	3273.5
	1.0	35.6	23.2	(*) 74.3	75.4	3266.3	(*) 1625.4
1.0		(*) 19.1	17.3	107361.8	1423324.2	7281.4	19784.8
SGA		1148.3		3365.1		17394	
min. / min.		19.1 / 15.8 = 1.20		74.3 / 65.5 = 1.13		2738.5 / 1625.4 = 1.68	
avg. / avg.		219.29 / 53.71 = 4.08		11271.78 / 142571.17 = 0.07		9383.39 / 7941.53 = 1.18	
avg. † / avg. †		241.53 / 57.75 = 4.18		595.11 / 265.27 = 2.24		9616.94 / 6625.61 = 1.45	

Table 3: Experimental results (24 bits)

ξ	p_m	f_1		f_2		f_3	
		QGA	NGA	QGA	NGA	QGA	NGA
0.4	0.6	29611.7	696.4	173448.1	6836.4	449218.0	280680.9
	0.8	19683.3	738.7	77715.8	4896.5	612616.6	113630.2
	1.0	1709.6	148.8	1632.5	434.7	4573.0	5001.5
0.6	0.6	1126.6	105.6	3111.1	438.6	305081.6	51827.4
	0.8	1280.6	89.6	2563.8	349.7	138789.2	35466.1
	1.0	338.7	64.2	488.9	182.3	(*) 1691.9	4060.2
0.8	0.6	167.4	31.9	949.1	2186.1	27889.8	6950.7
	0.8	78.0	44.9	579.5	240.5	54183.6	3089.0
	1.0	91.0	34.9	(*) 255.6	(*) 96.1	2357.2	(*) 1808.4
1.0		(*) 41.7	(*) 24.7	189801.5	4226414.4	20795.2	18510.7
SGA		64201.3		59078.5		776085.3	
min. / min.		41.7 / 24.7 = 1.7		255.6 / 96.1 = 2.7		1691.9 / 1808.4 = 0.9	
avg. / avg.		5412.8 / 197.9 = 27.3		45054.5 / 424207.5 = 0.1		161719.6 / 52102.5 = 3.1	
avg. † / avg. †		6009.6 / 217.2 = 27.6		28971.6 / 1740.1 = 16.6		177377.8 / 55834.9 = 3.1	

problem, then the performances of QGA and NGA on the $\xi=1$ shows considerably good performances. In f_2 and f_3 problems, however, QGA and NGA on the $\xi=1$ take many generations to reach global optimum. In the experimental results with 18 bits as shown in the table 2, the NGA outperforms the SGA and shows better performances than the QGA in the (min. / min.) and (avg. † / avg. †) measures. This trend does not largely change at experimental results with 24 bits. s shown in the table 3, NGA is better than QGA in the viewpoints of two measures for f_1. In f_2, however, NGA is worse than QGA in the viewpoints of average generation ratio. This is because the NGA shows the worst performance at $\xi=1$. As we already mentioned before, we do not recommend $\xi=1$ in complex functions. In the measure (avg. † / avg. †), the NGA shows better results than QGA for f_2. Unlike the function f_1 and f_2, the NGA on f_3 shows poor performance than QGA in the viewpoint of minimum generation. However, NGA shows somewhat better result than QGA in the average generation point of view. We think that the effect of derivative evaluation is small in very complex functions such as f_3. Since QGA is considerably optimized method in comparison to the SGA as shown in [20], it is very difficult to find a new method to enhance the performance of QGA. However, the derivative evaluation can be a good cooperation method for upgrading the QGA.

5. Conclusion

In this paper, we proposed a fusion method of queen-bee evolution and derivative evaluation for enhancing the optimization speed of genetic algorithms. Their fusion makes GAs fast approach to the global optimum with less probability of premature convergence. This resulted in upgrading the performances of GAs. We tested our proposed method with one combinational problem and two function optimization problems. Experimental results showed that the proposed genetic algorithm found the optimal solution faster than the simple genetic algorithm and the queen-bee genetic algorithm in most cases. This indicates that the fusion of queen-bee evolution and derivative evaluation can be a good method for enhancing the optimization performance of genetic algorithms. For more trustworthy application of our method, however, the selecting of proper mutation rate ξ and strong mutation probability $p_m^{'}$ according to the properties of optimization problems must be researched as a further work.

References

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

[2] 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.

[3] M. Srinivas and L. M. Patnaik, ``Genetic Algorithms: A Survey,'' *IEEE Computer Magazine*, pp. 17--26, June 1994.

[4] J. L. R. Filho and P. C. Treleaven, ``Genetic-Algorithm Programming Environments,'' *IEEE Computer Magazine*, pp. 28--43, June 1994.

[5] D. Beasley, D. R. Bull, and R. R. Martin, ``An Overview of Genetic Algorithms: Part 1, Fundamentals,'' Technical Report.

[6] D. B. Fogel, ``An Introduction to Simulated Evolutionary Optimization,'' *IEEE Trans. on Neural Networks*, vol. 5, pp. 3--14, Jan. 1994.

[7] 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.

[8] 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.

[9] C. Xudong, Q. Jingen, N. Guangzheng, Y. Shiyou, and Z. Mingliu, ``An Improved Genetic Algorithm for Global Optimization of Electromagnetic Problems,'' *IEEE Trans. on Magnetics*, vol. 37, pp. 3579--3583, Sept. 2001.

[10] J. A. Vasconcelos, J. A. Ramirez, R. H. C. Takahashi, and R. R. Saldanha, ``Improvements in Genetic Algorithms,'' *IEEE Trans. on Magnetics*, vol. 37, pp. 3414--3417, Sept. 2001.

[11] L. Davis, ``Adapting Operator Probabilities in Genetic Algorithms,'' in *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, pp. 61--69, 1989.

[12] 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.

[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] 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.

[15] A. Tuson and P. Ross, ``Adapting Operator Settings In Genetic Algorithms,'' *Evolutionary Computation*, vol. 6, no. 2, pp. 161--184, 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, ``Self-tuning of Operator Probabilities in Genetic Algorithms,'' in *Journal of Electronics Engineers of Korea*, vol. 37, pp. 29--44, Sep. 2000.

[18] 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.

[19] 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.

[20] S. H. Jung, ``Queen-bee evolution for genetic algorithms,'' *Electronics Letters*, vol. 39, pp. 575--576, Mar. 2003.

[21] S. H. Jung, ``Derivative Evaluation and Conditional Random Selection for Accelerating Genetic Algorithms,'' *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 5, pp. 21--28, Mar. 2005.

[22] K. DeJong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

Sung Hoon Jung

Sung Hoon Jung received his B.S.E.E. degree from Hanyang University, Korea, in 1988 and M.S. and Ph.D. degrees from KAIST, in 1991 and 1995, respectively. He joined the Department of Information and Communication Engineering at the Hansung University in 1996, where he is an associate professor. His research interests are in the field of intelligent systems, and in particular of application of neural networks, fuzzy logic, evolutionary computation algorithms (genetic

algorithms, evolutionary programming and evolution strategy) to intelligent systems such as intelligent control systems, intelligent characters of computer games, and so on. He is a member of the Korea Fuzzy Logic and Intelligent Systems Society (KFIS) and Institute of Electronics Engineers of Korea (IEEK).

Phone : 02-760-4344
Fax : 02-760-4435
E-mail : shjung@hansung.ac.kr