Fuzzy Rule Optimization Using Genetic Algorithms with Adaptive Probability

적응 확률을 갖는 유전자 알고리즘을 사용한 퍼지규칙의 최적화

Sung Hoon Jung*
정 성 훈*

ABSTRACT

Fuzzy rules in fuzzy logic control play a major role in deciding the control dynamics of a fuzzy logic controller. Thus, control performance is mainly determined by the quality of fuzzy rules. This paper introduces an optimization method for fuzzy rules using GAs with adaptive probabilies of crossover and mutation. Also we design two fitness measures to satisfy control objectives by partitioning the response of a plant into two parts.

An initial population is generated by an automatic fuzzy rule generation method instead of random selection for fast approaching to the final solution. We employed a nonlinear plant to simulate our method. It is shown through simulation that our method is reasonable and can be useful for optimizing fuzzy rules.

요 약

퍼지제어에서 퍼지규칙은 퍼지제어기의 제어결정을 내리는데 중요한 역할을 한다. 그래서, 제어성능은 주로 퍼지규칙의 질에 의해서 결정된다. 본 논문에서 우리는 교차와 돌연변이의 확률이 적응적으로 변화되는 유전자 알고리즘을 사용하여 퍼지규칙을 최적화 하는 방법을 기술한다.

또한 본 논문에서 우리는 플랜트의 응답을 두개의 부분으로 나누어 제어 목적을 만족하게 하는 적합도 측정 방식을 제안한다. 좀더 빠른 해답을 얻기 위해 우리는 초기의 퍼지규칙으로 무작위적인 규칙을 사용하지 않고 자동으로 퍼지규칙을 생성하는 방법을 사용하여 초기 퍼지규칙으로 사용했다. 이렇게 얻어진 퍼지규칙이 좋은 것인지를 보여주기 위해 비선형 플랜트를 이용하여 시뮬레이션 해보았다. 시뮬레이션 결과 우리의 방법이 합리적이고 유용한 것임이 밝혀졌다.

I. Introduction

Fuzzy logic control (FLC) methods have been widely employed to control highly nonlinear plants from the late 1980s [1, 2, 3]. In fuzzy logic control (FLC), fuzzy rules play a major role for deciding the control dynamics of a fuzzy logic controller [1]. Thus, control performance of FLC is mainly determined by

*School of Information and Computer Engineering, Hansung Univ.
한성대학교 정보전산학부

the quality of fuzzy rules [4, 5].

Fuzzy rules have been mainly developed by human experts so far because membership functions in fuzzy logic represents linguistic terms. Recently, self organizing fuzzy control methodologies that fuse the fuzzy logic control and the neural network theory have been introduced [4, 5, 6, 7]. Such methodologies, however, do not guarantee good control performance although they provide self organizing methods for fuzzy logic controllers.

To increase the control performance, this paper proposes an optimization method for fuzzy rules using genetic algorithms (GAs). Initial population in which a chromosome represents a fuzzy rule set can be randomly selected. In this paper, however, we used an automatic fuzzy rule generation method [8, 9] for fast approaching to the final solution. In our method, the probabilities of crossover and mutation are adaptively changed to focus on the final solution. Fitness of each chromosome (i. e., each fuzzy rule set) is directly calculated from simulation of fuzzy logic control with a controlled plant.

To weight the importance of the responses of the plant, the fitness calculation is classified into *transient response fitness* and *steady-state response fitness*. A controller designer can select an appropriate weight value according to the control requirements.

We simulated our method with a highly nonlinear plant. It is shown through simulation that the control performance of a fuzzy logic controller with optimized fuzzy rules is considerably better than that of a fuzzy logic controller with initial fuzzy rules even when the control performance with initial fuzzy rules are very poor.

This paper is organized as follows.

Section 2 briefly describes an automatic fuzzy rule generation method and simple genetic algorithms. The fuzzy rule optimization method is provided in section 3. Section 4 shows the simulation results. Conclusion is given in section 5.

Ⅱ. Brief Review: Genetic Algorithms

Fuzzy rules to be optimized can be obtained by random initialization. In this case, however, the population is converged slowly because the fitness of genes have low values. It has also been known that initial population affects the quality of the final solution. Thus, an automatic fuzzy rule generation algorithm introduced in [8, 9] is employed to provide a GA with an initial population. More information about AFRG can be found in [8, 9]. GAs are adaptive methods for solving search and optimization problems [10, 11, 12, 13, 14]. They are based on the genetic processes of biological organisms.

Natural populations evolve through natural selection and survival of the fittest over many generations. GAs simulate those processes in natural populations which are essential to evolution. It has been shown that these GAs can well be applied to real world problems if they have been suitably encoded. The combination of good characteristics from different ancestors can sometimes produce a superfit offspring, whose fitness is greater than that of either parent. In this way, species evolve to become more and more well suited to their environment.

The power of GAs comes from the fact that the technique is robust, and that it can deal successfully with a wide range of problem areas, including those which are difficult for other methods to solve. GAs are not guaranteed to find the global optimum solutions to a problem, but they are generally good at finding *acceptably good* solutions to problems *acceptably quickly*.

Algorithm 1 shows the structure of a simple genetic algorithm.

Algorithm 1. Simple Genetic Algorithm

// t: time //

// P: populations //

1. $t \leftarrow 0$

2. initialize $P(t)$

3. evaluate $P(t)$
4. *while* (*not* termination-condition)
5. *do*
6. $t \leftarrow t+1$
7. select $P(t)$ from $P(t-1)$
8. recombine $P(t)$
9. evaluate $P(t)$
10. **end**

Before a GA can be applied to a problem, the problem must be coded. Then, a specific number of initial populations are generated according to the coding method. These coded genes are evaluated to measure the fitness of the solutions. To make the next generation, selection and recombination are necessary. This procedure does not stop before a satisfiable solution is obtained. More detailed descriptions about GAs are available in [10. 11].

Ⅲ. Fuzzy Rule Optimization

Even if the AFRG algorithm generates relatively good fuzzy rules, it is not guaranteed that generated rules have good enough quality to be used in real applications without any modification. Also even if the quality of the generated fuzzy rules are good their optimization is necessary if they can be more optimized. Combining these two methodologies enables a fuzzy logic controller to be equipped with the ability of self fuzzy rule generation without any intervention of human experts. We use GAs to optimize fuzzy rules. Original GAs have two problems.

First, after a population converges to the globally optimal solution, constant probabilities of crossover and mutation cause the disruption of the near-optimal solutions. Second, search space of a population remains large even if it approaches the globally optimal solution. These make GAs inefficient.

In this paper, we employed modified GAs to solve these two problems. Before a GA can be run, a suitable *coding* (or *representation*) for the problem must be devised. We also require a *fitness function*, which assigns a figure of merit to each coded solution. During the run, parents must be selected for reproduction and recombined for generating of offspring. It is assumed that the shapes of membership functions are predetermined.

Under this assumption, we coded a fuzzy rule set using the order of rule table. That is, output linguistic terms are ordered to convert a bit string chromosome according to a predefined order. Each linguistic term is represented by a bit string as shown in Table I.

Table I. Coding of Fuzzy Linguistic Values

Values	NB	NM	NS	ZO	PS	PM	PB	NULL
Coding	000	001	010	011	100	101	110	111

An unnecessary rule set is deleted through iteration by employing a *don't care* conditioned fuzzy label, NULL in the rule set. That is, if all of the fuzzy labels of output membership functions are composed of nulls then this rule set is deleted in the next iteration step.

Figure 1 shows a chromosome of a fuzzy rule set.

Fig. 1 A chromosome of a fuzzy rule set

The coded bit strings are used as chromosomes in GA and the optimal bit string corresponding to optimal rule set is searched through the fitness value. A fitness value of each chromosome decided by a fitness function is very important in GAs because this value represents the quality of solution of a chromosome. The fitness function returns a single numberical *fitness* or *figure of merit*, which is supposed to be proportional to the *utility* or *ability* of the individual which that chromosome represents.

In our problem, the fitness value indicates how

much current fuzzy rules are adequate to control a plant. During the search process using our GA, a selected string is decoded into a rule set and this rule set is applied to the plant. The result of fuzzy logic control using the decoded rule set is employed as a fitness measure. That is, the inverse value of the total sum square error (TSSE) derived from the results of the control is taken as a fitness value, i. e., $f_i = \frac{1}{tsse_i}$.

We call this measure *simulation-based fitness measure*.

The highly fit individuals in a population have high opportunities to *reproduce*. As a result, new individuals, the offspring, are produced. Offspring shares some features taken from each parent. The least fit individuals in the population have low probability of being selected for reproduction; and they finally die out. A whole new population of possible solutions is thus produced by selecting the best individuals from the current generation and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by the good individuals of the previous generation.

In this way, over many generations, good characteristics are spread throughout the population, being mixed and exchanged with other good characteristics as they go. By favoring the mating of the more fit individuals, the most promising areas of the search space are explored. Having selected two parents, their chromosomes are recombined, typically using the mechanisms of *crossover* and *mutation*.

Crossover takes two individuals and cuts their chromosome strings at some randomly chosen position to produce two head segments and two tail segments. The tail segments are then swapped over to produce two new full length chromosomes. Figure 2 shows two points of crossover.

The two offsprings each inherit some genes from each parent. Crossover is not usually applied to all pairs of individuals selected for mating. A random choice is made, whether the likelihood of crossover being applied is typically between 0.6 and 1.0. If crossover is not applied, offspring are produced simply by duplicating the parents. This gives each individual a chance of passing on its genes without the disruption of crossover. Mutation is applied to each child individually after crossover. It randomly alters each chromosome with a small probability typically 0.001. Figure 3 shows the six points mutation.

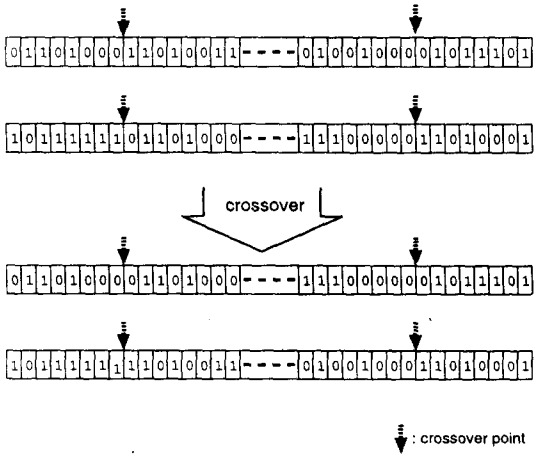


Fig. 2 Crossover of two chromosomes

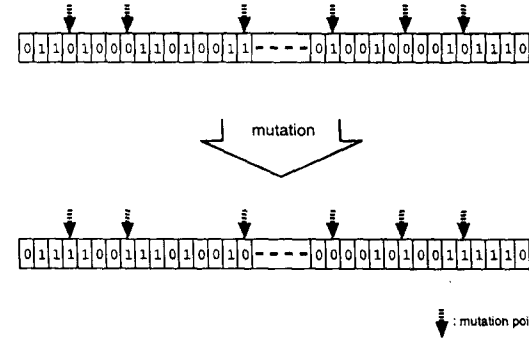


Fig. 3 Mutation of two chromosomes

This reproduction procedure is repeated until the given performance objective is achieved. In our method, the probabilities of crossover and mutation are large initially; for example, p_c and p_m are 1.0 and

0.05, respectively. The values of probabilities are dynamically varied in proportion to the ratio between the initial maximum value fitness and the current maximum fitness value.

Definition 1: Adaptive p_c and p_m

Let the initial maximum fitness value and the current maximum fitness value be $f^i_{\max}$ and $f^c_{\max}$, respectively, and the initial probabilities of crossover and mutation be p^i_c and p^i_m, respectively, then the adaptive probabilities of crossover and mutation are given as:

$$p_c = p^i_c \times \frac{f^i_{\max}}{f^c_{\max}}$$

$$p_m = p^i_m \times \frac{f^i_{\max}}{f^c_{\max}}$$

This adaptive probability strategy provides a GA with the focusing mechanism on the final solution. With this strategy, the disruption in the near-optimal solutions is considerably reduced. This strategy does not make the GA fall into a local minimum because it broadly searches the whole solution space initially with high probabilities of crossover and mutation.

In our method, the fitness of each chromosome (i. e., each fuzzy rule set) is directly calculated from simulation of fuzzy logic control with a controlled plant. That is, the inverse of the total sum square error (TSSE) between set points and plant's responses is used as a measure of fitness of a chromosome.

To weigh the importance of the responses of the plant, the fitness calculation is classified into *transient response fitness* and *steady-state response fitness*.

$$f_i = \frac{1}{tsse_i} = \frac{1}{\sum_{k=1}^{M} \frac{e_k \times e_k}{W}^{\frac{1}{2}}} \quad : \textit{transient response fitness}$$

$$f_i = \frac{1}{tsse_i} = \frac{1}{\sum_{k=1}^{M} e_k \times e_k^{\frac{1}{2}}} \quad : \textit{steady state response fitness}$$

where i is the $i'th$ chromosome, k is the number of control steps, and M is the maximum control step. In the equations, e_k is the error between the reference value and a current value of the plant output when the simulation step is k and W is a constant weight value in real domain.

A controller designer can select an appropriate weight value according to the control requirements. For example, if a control designer wants to obtain fuzzy rules that show good control performance in the transient response, then he sets the W to be greater than one.

Otherwise, he sets W to be less than one. In the calculation of fitness, set points that are applied to the controlled plant are randomly selected from -1.0 to 1.0 with uniform distribution.

An initial population is generated by an automatic fuzzy rule generation method instead of random selection for fast approaching to the final solution. Algorithm 2 shows the steps of the simulation-based fitness measure.

Algorithm 2. Fitness measure

{*evaluating population*}

1. decode each chromosome into a fuzzy rule set
2. simulate fuzzy logic control with the fuzzy rule set
3. obtain the total sum square error value($tsse_i$)
4. set the inverse value of $tsse_i$ to the fitness value of the chromosome (i. e., $f_i = \frac{1}{tsse_i}$)

With the fitness measure, our fuzzy rule optimization algorithm operates as shown in Algorithm 3.

Algorithm 3. Optimized Fuzzy Rule Generation

// N: the size of population //

// M: the number of not evolved iteration //

1. **for** i = 0 to $N-1$ {*initialize population*}
2. generate an initial fuzzy rule set by AFRG
3. encode the fuzzy rule set into a chromosome
4. **end for**

5. fitness measure of all chromosomes
6. **while** *NOT* termination condition **do**
7. **if** adaptive probability mode **then**
8. $p_c = p_c^i \times \frac{f^i_{\max}}{f^c_{\max}}$
9. $p_m = p_m^i \times \frac{f^i_{\max}}{f^c_{\max}}$
10. **end if**
11. crossover operation
12. mutation operation
13. fitness measure of all chromosomes
14. find the maximum fitness value
15. **if** *NOT* evolved during M **then**
16. terminate
17. **end if**
18. **end while**

Figure 4 shows the overall simulation setup for optimization of fuzzy rules.

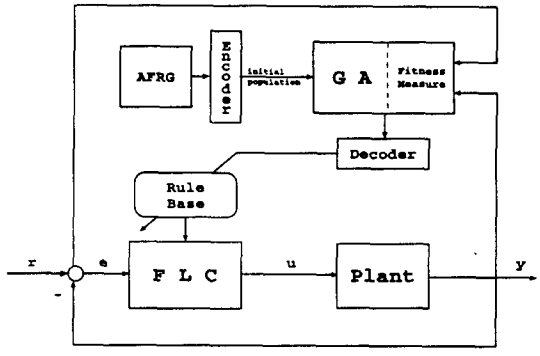


Fig. 4 Overall simulation setup (AFRG : Automatic Fuzzy Rule Generation)

Initial population is generated by AFRG and encoder. With the initial population, the GA works with a simulated fuzzy logic controller for fitness measure.

Ⅳ. Simulation Results

A very nonlinear plant is applied to our control system to show the performance of our method. The nonlinear plant is given by a difference equation form as follows.

$$plant : y(k+1) = \frac{y(k)}{1+y^2(k)}\, u(k) + u(k)$$

The number of the control length for one time series and the number of set points for gathering statistics are set to 100, respectively. Mamdani's inference method and LGM (Level Grading Method) defuzzification method [15] are used to control the plant. Asymmetric several linguistic terms, *NB, NS, NZ, ZO, PZ, PS, PB*, are employed as shown in Figure 5.

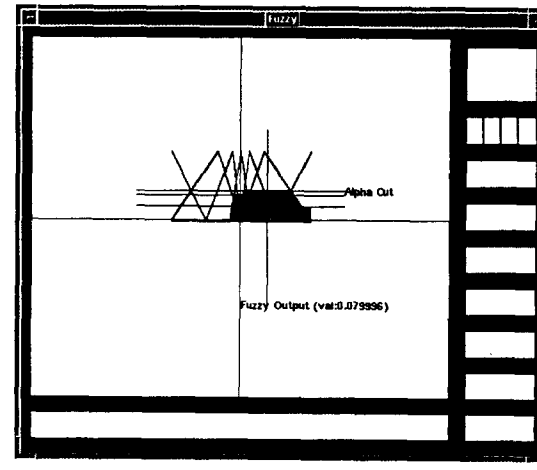


Fig. 5 Membership functions

First, we shows a result that is generated from a GA with constant probabilities of crossover and mutation. Figure 6 shows the result. As you can see, the result is considerably poor. Especially, the steady state error is very large.

Figure 7 shows two simulation results generated from a GA with adaptive probabilities of crossover

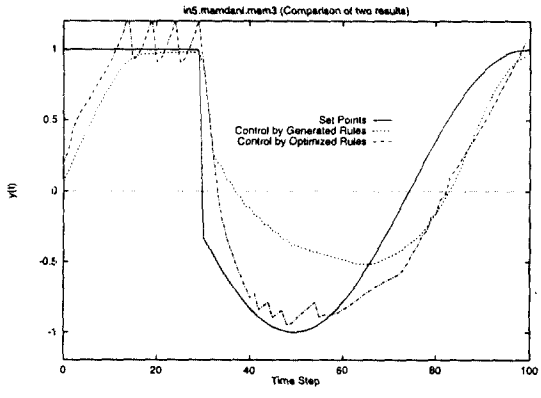
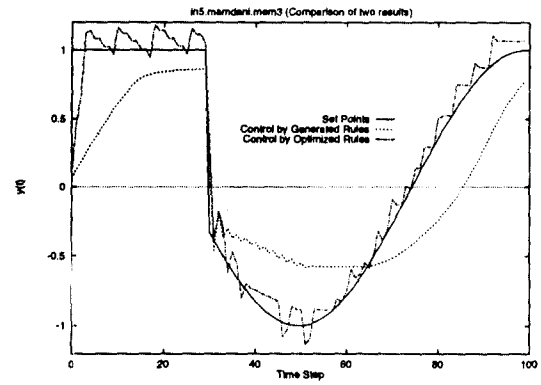


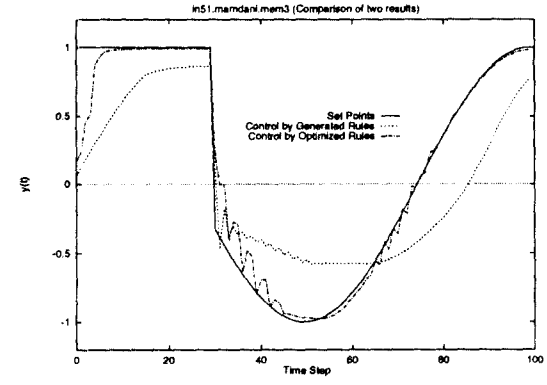
Fig. 6 Result of constant probability

and mutation. In the simulations, the GA is finished only when the fitness is not optimized further. The average generations for the GA to finish are about from 40 to 60.

In the result figures, normal fitness is not to take the weight value, W, and weighted fitness is to take the weight value within transient response in calculation fitness. Figure 8 shows another result using weighted fitness measure. In this case, although the performance under the generated rules is very poor, the optimized rules show a relatively good performance. This indicates that the optimization technology with genetic algorithm is considerably useful.



(a)



(b)

Fig. 7 (a) normal fitness (b) weighted fitness

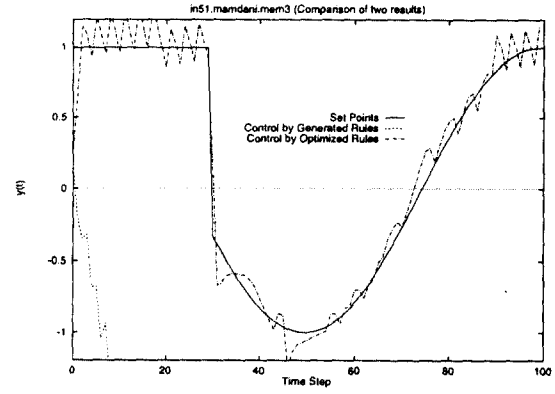


Fig. 8 Weighted fitness

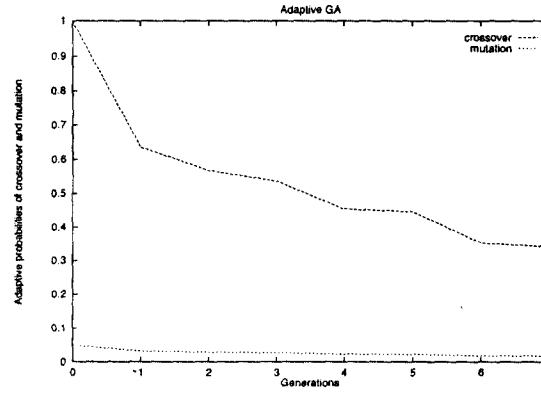


Fig. 9 Adaptive probabilities of crossover and mutation

Figure 9 shows the variation of probabilities in adaptive GA.

As shown in the result figures, initial fuzzy rules are optimized by GAs to some degree. Adaptive GA shows a generally better performance, but not always because GA operators are not guaranteed that offsprings are always better than parents. Similarly, the weighted fitness measure shows better results in the steady state, but not always. The adaptive GA does not perturbate near the final solution because the probabilities of crossover and mutation are low. This adaptive methodology prevents a GA from diverging near the final solution. In the case that the final solution falls into a local optima, however, this property also may provide a disadvantage. Though this can give the disadvantages, it will not often occur because our algorithm searches broadly initially.

V. Conclusions

This paper introduced an optimization methodology using a GA with adaptive probabilities of crossover and mutation. To provide a GA with a good initial population, the AFRG method is employed. We describe two methods for fitness measure. Simulation results show this optimization method for fuzzy rules is useful in the practical automatic fuzzy rule generation.

References

1. C. C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller-Part I/II," *IEEE Trans. on Systems, Man and Cybernetics,* vol. 20, pp. 404-435, MARCH/APRIL 1990.
2. H. Hellendoorn and C. Thomas, "Defuzzification in Fuzzy Controllers," *Journal of Intelligent and Fuzzy Systems,* vol. 1, pp. 109-123, 1993.
3. W. Pedrycz, *Fuzzy Control and Fuzzy Systems.* Research Studies Press, 1989.
4. J.-S. R. Jang, "Self-Learning Fuzzy Controllers Based on Temporal Back Propagation," *IEEE Trans. on Neural Networks,* vol. 3, pp. 714-723, Sept. 1992.
5. H. R. Berenji and P. Khedkar, "Learning and Tuning Fuzzy Logic Controllers Through Reinforcements," *IEEE Trans. on Neural Networks,* vol. 3, pp. 724-740, Sept. 1992.
6. C.-T. Lin and C. G. Lee, "Neural-Network-Based Fuzzy Logic Control and Decision System," *IEEE Trans. on Computers,* vol. 40, pp. 1320-1336, Dec. 1991.
7. C.-C. Lee, "Intelligent Control Based on Fuzzy Logic and Neural Net Theory," *Proceedings of the International Conference on Fuzzy Logic,* pp. 759-764, July 1990.
8. S. H. Jung, "Automatic Fuzzy Rule Generation by Simulating Human Knowledge Gathering Process," *Journal of the Korea Fuzzy Logic and Intelligent Systems Society,* vol. 5, pp. 12-17, Dec. 1995.
9. S. H. Jung, T. G. Kim, and K. H. Park, "Automatic Fuzzy Rule Generation by Simulating Human Control Strategies," *Proceedings of the Korea Fuzzy Math and Systems Society,* vol. 4, pp. 72-78, Nov. 1994.
10. M. Srinivas and L. M. Patnaik, "Genetic Algorithms: A Survey," *IEEE Computer Magazine,* pp. 17-26, June 1994.
11. J. L. R. Filho and P. C. Treleaven, "Genetic-Algorithm Programming Environments," *IEEE Computer Magazine,* pp. 28-43, June 1994.
12. G. A. Vignaux and Z. Michalewicz, "A Genetic Algorithm for the Linear Transportation Problem," *IEEE Trans. on Systems, Man and Cybernetics,* vol. 21, pp. 445-452, MARCH/APRIL 1992.
13. 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.
14. M. Srinivas and L. M. Patnaik, "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms," *IEEE Trans. on Systems, Man and Cybernetics,* vol. 24, pp. 656-667, Apr. 1994.

15. S. H. Jung, K. H. Cho, T. G. Kim, and K. H. Park, "Defuzzification Method for Multishaped Output Fuzzy Sets," *Electronics Letters*, vol. 30, pp. 740-742, Apr. 1994.

정 성 훈(Sung Hoon Jung) 정회원

1988년 : 한양대학교 전자공학과 졸업
1991년 : 한국과학기술원 전기 및 전자공학과 졸업 (공학석사)
1995년 : 한국과학기술원 전기 및 전자공학과 졸업 (공학박사)
1996년~현재 : 한성대학교 정보전산학부 전임강사
※주관심분야 : 지능 제어, 뉴로 퍼지, 유전자 알고리즘, 모델링/시뮬레이션