A Multi-operator Self-tuning Genetic Algorithms

Masaki Imagawa, Chin-chih Hsu, Shin-ichi Yamada, Hideji Fujikawa
Master Doctor, Professor, Professor
Department of Electrical and Electronic Engineering
Musashi Institute of Technology
Tel : +81-3-5707-7115
Fax : +81-3-5707-2147
E-mail : imap@ac.cs.musashi-tech.ac.jp

ABSTRACT: A traditional simple genetic algorithm (SGA) simulates evolution as found in natural biological creatures, thus they let mutation happen after crossover with a very small probability. However, some researchers found that a naive evolution (just selection and mutation) performs a hill-climb-like search and can be powerful without crossover. We proposed a Multi-operator Self-tuning GA (MSGA) with an effective of fuzzy reasoning, where the mutation loop is simplified with given each operator a constant population. After running SGA, MGA and MSGA ten times, we got the best solutions. Where we found the MSGA got best solution in our test cases.

KEYWORD: Simple Genetic Algorithm (SGA), Multi-operator Self-tuning GA (MSGA), Fuzzy Reasoning

1. Introduction

A traditional simple genetic algorithm (SGA) simulates evolution as found in natural biological creatures, thus it lets mutation happen after crossover with a very small probability (typically 0.001). This forces most of the evolution to be promoted by crossover. However, some researchers have found that a naive evolution (just selection and mutation) performs a hill-climb-like search and can be powerful without crossover [1-6]. We thus suggest that all genetic operators have their advantages during different stages of search and that the search processes should be adjusted to take advantage of this. For example, crossover may be ineffective in the earlier stages because few good genes are present in the parents.

While observing the searching processes of SGA, we find these phenomena:

(1) Crossover happens right after the initial generation
Crossover is designed for generating offspring that will inherit some good genes from their parents. However, do the parents have any good genes in the initial generation ?

(2) Mutation happens right after crossover with a very small probability
The series processes of a SGA is constructed by simulating natural evolution but it minimizes the effectiveness of mutation. On the other hand, if a bad offspring is generated by crossover then the mutation becomes a dummy operation.

(3) What does a fitness value mean ?
In general, a fitness index shows how good the complete set of genes that make up a chromosome is. Combining good parts of chromosomes by crossover in order to get better offspring may not be useful for all applications, especially when the individual genes have interrelationships (*e.g.* fuzzy rule set). On the contrary, a string with a low fitness value seems to be a bad combination of its contents but it may be a good starting point to climb to an optimum solution.

(4) Cooperation and competition
The reproduction process generates diversity in the gene pool by selecting parents from the mating pool. The fact that the candidates in the mating pool are generated from all the operators means they cooperate to contribute their efforts toward evolution. On the other hand, the selection process selects only the fittest into the mating pool this makes a competition. Therefore, the offspring of an operator that can not generate fit individuals will not survive, this operator then becomes a dummy one.

Crossover is an important operator that allows good genes to be faithfully translated to their offspring. Mutation performs a hill-climb-like search and is a main source of raw material for evolutionary change [7]. Since each type of mutation has its own special approach algorithm, a GA with one mutation does one kind of approach and easily falls into local optimum. If a GA is designed with different types of mutations, it can do the search through different approach routes.

In this paper, mutations are considered the main evolutionary tools while crossover is considered a shifting tool used when individuals fall into local optimums by mutation. We thus propose a Multi-operator Self-tuning GA (MSGA). It is designed with multiple operators -- single-point crossover (*CR*), single-point mutation (*MU*), single-

point copy (*CO*) and single-point exchange (*EX*). The *MU*, *CO*, and *EX* are all considered mutation operators because they select one parent and generate one offspring for reproduction. Therefore, we have two groups of operators, one is a simple crossover only and the other is a group of mutation operators. These two groups will do the search job repeatedly until it reaches a predetermined stop condition.

There are three operators in the mutation group of MSGA, and these operators perform a hill-climb-like search. Since each mutation operator has its own approach path to a solution, and these operators are supposed not to be effective in all stages of searching, we apply fuzzy reasoning to tune their population sizes to give the more effective operators more opportunity to search.

A detailed description of the MSGA and its algorithm will be given in Section 2. The assumptions and test of applying MSGA to optimize Fuzzy control rules is show in Section 3, and some conclusions are described in Section 4.

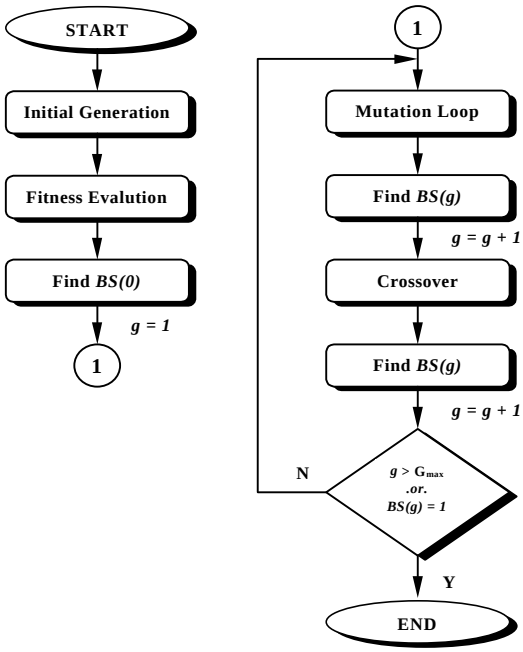

Fig. 1 A flow-chart of MSGA

2. Description of MSGA

2.1 MSGA with fuzzy reasoning

The flow chart of MSGA is shown in Fig. 1. A detailed flow chart of the mutation loop is shown in Fig. 2 and the crossover loop is shown in Fig. 3. The mutation and crossover loops are executed alternatively. A variety of operators are consisted of in mutation loop such as *MU*, *CO*, and *EX* for optimization of fuzzy rules. The mutation group has a set of hill-climb (or local searching) operators which guide individuals in climbing up to local optimum based on the individuals that are generated by crossover. These operators are modified with the eugenic concept under the assumption that the individuals with higher fitness values have a higher probability of breeding better offspring.

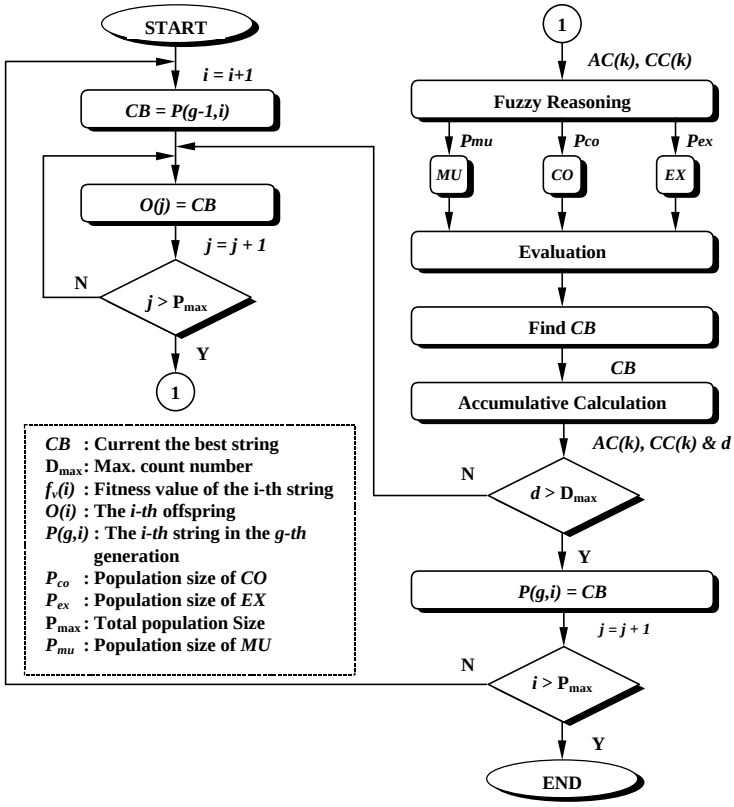

Fig. 2 A flow-chart of the mutation loop

In MSGA, the operator *CR* selects the prior best string, *BS(g-1)*, and one other string in the prior generation, *P(g-1,i)*, as parents and then generates an offspring. The *MU* is designed to select the best prior string as a parent and generate an offspring by randomly changing a randomly selected gene. The *CO* is designed to select the best prior string as a parent and generate an offspring by copying a randomly selected gene to another randomly selected gene. It is designed under the principle that a fit value for one gene may be suitable for another gene. The *EX* is designed to generate an offspring by exchanging two randomly selected genes and it is designed under the principle that the generated contents of genes may be good but the location should be adjusted. The mutation loop is run first because crossover needs good chromosomes to breed offspring.

The best string is preserved in each generation to make sure that the current best genes will not be lost when the operators fail to search out a better new string. This process does not take much time in fitness evaluation and can keep the best chromosome alive from generation to generation until another better one is bred. The "accumulative calculation" is added to calculate the contribution of each operator (see Section 2.2). The fuzzy reasoning is designed to determine population size of each mutation operator sensing the outputs of accumulative calculation. This means that all of the operators have a chance of generating new offspring but the number of generating will be adjusted depending on their search capabilities. How the fuzzy reasoning senses the contributions of each operator and then decides their population sizes will be described in Section 2.3. The terminate condition is set to be met with when the best fitness value is equal to 1 or the generation size, g, reaches a predetermined maximum size (G_{max})

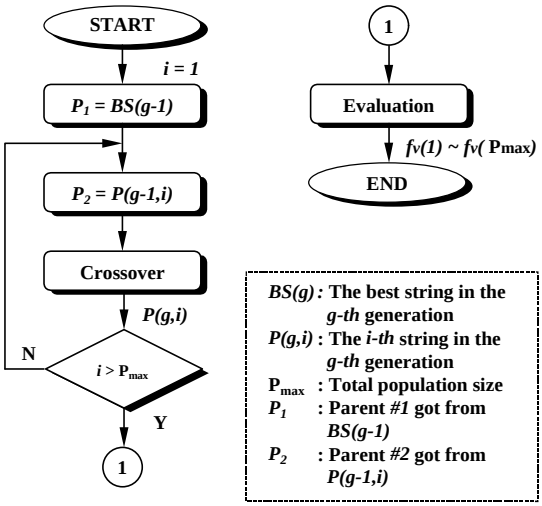

Fig. 3 A flow-chart of crossover

In order to evaluate the effect of fuzzy reasoning in MSGA, a multiple operator GA (MGA) is proposed with a constant population size for each operator in the mutation loop (Fig. 4), the fuzzy reasoning and accumulative calculation thus can be ignored. A comparison of SGA, MGA, and MSGA for the optimization of fuzzy controllers will be described in Section 4.

2.2 Accumulative calculation

As the operators are not always effective during searching, we add a process called "*accumulative calculation*" in MSGA to observe the searching ability of each operator in the mutation group. The contribution calculation is designed to calculate how many fitness values were obtained from the initial mutation cycle up to the current generation for each operator (defined as accumulative contribution, *AC*) and how many fitness values each operator contributed within the current mutation cycle of the current generation (defined as current contribution, *CC*). The procedures of accumulative calculation are described as follows :

(1) To find the best individual and its fitness value in each generation's population.

(2) To check if a successful search has happened.
A successful search is defined as when an operator has produced an individual with a higher fitness value than the best one prior the current mutation cycle.

(3) If successful, then calculate *AC* (*k*) & *CC* (*k*) and reset *d* to 0 else *d* increments by 1; where *k* is an identification number for each operator, *i.e. k* equals 1 for *MU*, 2 for *CO*, and 3 for *EX*.

(4) Condition checks
There are two condition checks after accumulative calculation, the first one checks if current generation's hill-climbing search has reached to a local optimum ($d > D_{max}$) and the second one checks if the hill-climbing mutation has been executed for all of the population.

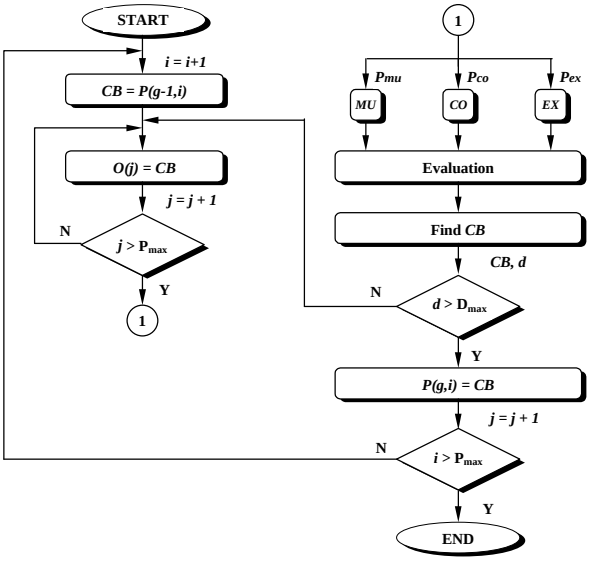

Fig. 4 A flow-chart of mutation loop for MGA

2.3 Fuzzy reasoning

The fuzzy reasoning observes $AC(k)$ & $CC(k)$, then adjusts $P_S(k)$ by "*If - Then*" rules which are represented as :

$$\begin{aligned} &\text{Rule } (i \times j) : \\ &\text{If } AC \text{ is } A_i \text{ and } CC \text{ is } B_j \text{ Then } RP \text{ is } C_{i,j} \\ &\qquad for \quad i = 1 \ldots 3,\ j = 1 \ldots 3 \end{aligned} \tag{1}$$

where AC and CC are linguistic variables and their universes of discourse U are all in [0, AC_{max}], and [0, CC_{max}]. A_i and B_j are three fuzzy subsets (ZE, S, and L) with triangular membership functions (as shown in Fig. 5). RP is a rough estimate population size of each operator, it will be adjusted to keep the total population size equal in each search loop. $C_{i,j}$ is a single tone set which quantizes RP (as shown in Fig. 6), and the fuzzy rules are listed in Table 1. For the inputs $AC(k)$and $CC(k)$, the activation of rule $i \times j$ is determined by :

$$w_k(i,j) = \min \left(m_{A_i}(AC(k)),\ m_{B_j}(CC(k)) \right) \tag{2}$$

Therefore $RP(k)$ is calculated by using a weighted average algorithm as :

$$RP(k) = \frac{\sum_{i=1}^{3}\sum_{j=1}^{3} w_k(i,j) \times C_{i,j}}{\sum_{i=1}^{3}\sum_{j=1}^{3} w_k(i,j)} \tag{3}$$

thus $P_S(k)$ of *k-th* operator is adjusted as :

$$P_s(k) = P_0 + \frac{RP(k)}{\sum_{k=1}^{3} RP(k)} \times P_t \tag{4}$$

or when all $RP(k)$ equal to 0, then $P_S(k)$ is set to be :

$$P_s(k) = P_0 + \frac{P_t}{3} \tag{5}$$

Where P_0 is the minimum allowed population size for each operator and P_t is the distributed shared population size. Therefore the total population size in each mutation cycle will be $P_t+3\times P_0$.

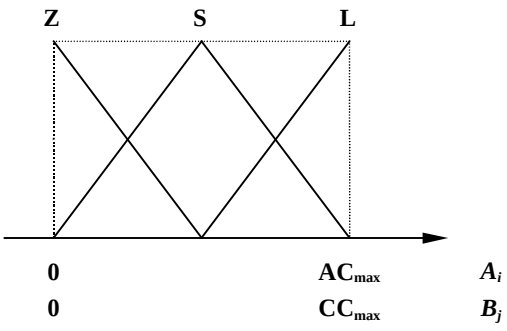

Fig. 5 Membership functions of antecedents

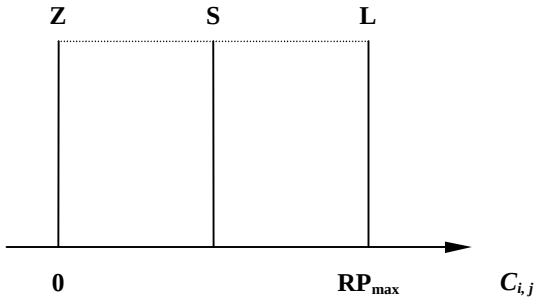

Fig. 6 Membership functions of consequent

Table 1 Rule set of fuzzy reasoning for MSGA

		AC		
		Z	S	L
CC	Z	Z	Z	S
	S	Z	S	L
	L	S	L	L

3. Simulation results of MSGA

At first, we apply MSGA to optimize MRFACS with FIS [8]. Fig. 7 are examples of the average fitness of the mutation loop and crossover loop. It shows that in MSGA, the populations climb up during mutation loop then crossover falls down to other valley and the mutation loop climbs up again. At first, the mutation loop makes a long climb thus takes time. The more closer to the convergent point the less run in mutation loop, thus less time expand in iteration. Moreover, the average fitness values gotten by crossover are lower than those from the prior mutation operations. This means that crossover not only help breed better offspring but also recombine the organisms of individuals (this can be considered as making a shift to new starting points for the mutation loop). For most cases, with crossover the better the parents the better the offspring, thus the searching converges to an optimum solution.

4 Comparisons

In order to show the effectiveness of the fuzzy reasoning in MSGA, the mutation loop is simplified by giving each operator a constant population as shown in Eq. 5 (named MGA). After running SGA, MGA and MSGA ten times, we got the best solutions as listed in Table 2. Where we found that the MSGA got the best solution for our test plant.

Fig. 8 shows the distributions of the best fitness values for each GA, where the radius of SGA is relative lager then MGA and MSGA, this means that the solutions derived by SGA rely on the random number seed more than in MGA or MSGA. This also means that to apply SGA to elicit solution, you need many tries. On the other hand, the smallest radius in MSGA means that MSGA helps you get a solution closer to the global solution and it also leads you to a better solution within a small number of tries.

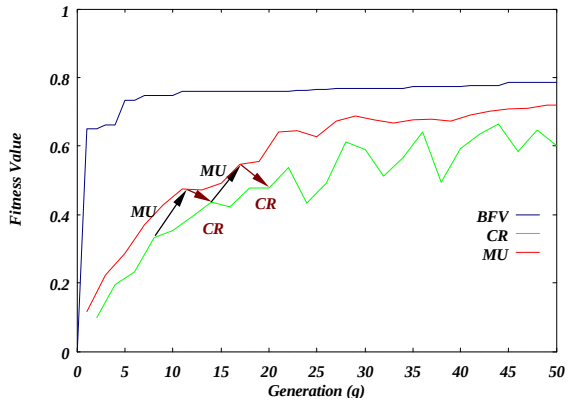

Fig. 7 Average fitness value of CR & MU loops in MSGA

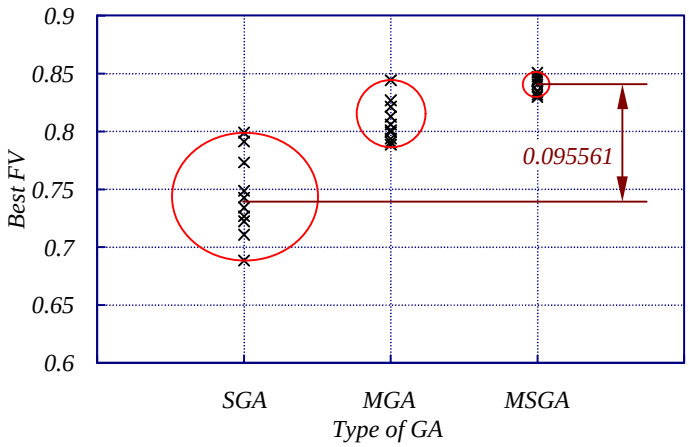

Fig. 8 A comparison of the distribution of the best fitness values between GAs

Table 2 Search results of GAs for TYPE 3 :FIS

SEED	SGA	MGA	MSGA
3773	0.710720	0.801130	0.834682
1379	0.726467	0.788397	0.847213
2515	0.742718	0.790692	0.834085
8328	0.773061	0.826695	0.840635
8111	0.748501	0.793662	0.845276
7275	0.73413	0.805907	0.831355
6318	0.798807	0.799320	0.829795
375	0.791269	0.821045	0.834799
7672	0.722214	0.812673	0.850678
9812	0.688240	0.844070	0.843217
Ave.	0.743613	0.808359	0.839174
Max.	0.798807	0.84407	0.850678
Min.	0.688240	0.788397	0.829795
Rad.	0.055284	0.027837	0.010442

5. Reference

[1] L. Davis, “Genetic Algorithms and Simulated Annealing”, Pitman, 1987

[2] David Beasley, David R. Bull, Ralph R. Martin, “An Overview of Genetic Algorithms : Part 1, Fundamentals”, from ftp : ralph. cm. cf. ac. uk : /pub / Gas / ga_overview.ps, University Computing, Vol. 15 , No. 2, pp. 58 - 69, 1993.

[3] David Beasley, David R. Bull, Ralph R. Martin, “An Overview of Genetic Algorithms : Part 2, Research Topics”, from ftp : ralph. cm. cf. ac. uk : /pub /Gas / ga_overview2.ps, University Computing, Vol. 15, No. 4 pp. 170 - 181, 1993.

[4] J. David Schaffer, Larry J. Eshelman, “On Crossover as an evolutionarily visable strategy”, In R. K. Belew and L. B. Booker, editor, Proc. the fourth ICGA, pp. 61 - 68 Morgan Kaufmann ,1991.

[5] Larry J. Eshelman, “Bit-climbers and Navie Evolution”, GA-Digest, Vol. 5, No. 39, 1991.

[6] William M. Spears, “Crossover or Mutation ?” in L. Darrell Whitley, Ed., Foundations of Genetic Algorithms, 2, Morgan Kaufmann pp. 221 - 237, 1993 .

[7] D. L. Hartl , “A primer of population genetics”, Sinauer Associates Inc. pp. 137, 1988.

[8] C.C. Hsu, S. Yamada, H. Fujikawa, K. Shida, “A Multi-Operator Self-Tuning Genetic Algorithm for Fuzzy Control Rule Optimization”, IECON'96 (Taiwan), pp. 842 - 847