# A Framework for Studying the Effects of Dynamic Crossover, Mutation, and Population Sizing in Genetic Algorithms

Michael A. LEE
University of California
Computer Science Division
Soda Hall
Berkeley, CA 94720-1776 USA
leem@cs.berkeley.edu

Hideyuki TAKAGI*
Kyushu Institute of Design
Dept. of Acoustic Design
4-9-1 Shiobaru, Minami-ku
Fukuoka, Fukuoka 815 JAPAN
takagi@kyushu-id.ac.jp

*Abstract - We introduce a framework for controlling genetic algorithms and use it to study the effect of dynamically modulating genetic algorithm control parameters on search behavior. Our framework includes techniques that can automatically design control strategies for genetic algorithms according to a given search performance metric. Many of the automatically designed strategies exhibit an exponentially decreasing mutation rate behavior. We present experimental results indicating that exponentially decreasing the mutation rate over time contributes more towards an increase on online and offline search performance than populations size or crossover rate modulation.*

## 1 Introduction

The relationship between a genetic algorithm's control parameters and its behavior is complex and has been the topic of much research. For example, the behavior of a genetic algorithm can range from that of random search to hill climbing depending on the genetic algorithm parameters settings. Designing a genetic algorithm that meets resource constraints of a given application may require a substantial amount of genetic algorithm control parameters tuning. In addition, because the genetic algorithm search is a dynamic process, maintaining a given search behavior may require dynamically modulating the control parameters. In this study, we present a framework for controlling genetic algorithm parameters and techniques to automatically design high performance strategies.

The remainder of the paper is divided into 7 major sections. The next section discusses related work and how our work builds on the results presented by others. Section 3 describes our framework for representing search control strategies: the Dynamic Parametric Genetic Algorithm (DPGA) -- a genetic algorithm controlled by a fuzzy knowledge-base. Section 4 presents our technique for automatically designing and tuning genetic algorithm control strategies. Sections 5 and 6 present experiments and results aimed at studying the interaction between mutation rates, crossover rates, and popula-

* Portions of this research were carried out while the author was a Visiting Industrial Fellow at UC Berkeley on leave from Central Research Laboratories, Matsushita Electric Industrial Co., Ltd.

tion sizing. In the final section, we explain how to extend our technique to other learning methods. We also outline areas that warrant further investigation.

## 2 Background and Related Work

Genetic algorithms were introduced and developed by John Holland and his colleagues as a system aimed at explaining and modelling adaptation of natural systems [11]. A basic genetic algorithm involves selection, mixing, and mutation components. Like biological systems, selection is driven by an organism's ability to survive in its environment. If the organism lives long enough to reproduce with other organisms, it will pass its genetic information onto its offspring. Although mature organisms are capable of reproducing, the most fit organisms usually have more opportunities to produce offspring. Mixing in genetic algorithms is usually implemented by crossover operators that combine genetic information from two or more parents. Mutation is a mechanism for reintroducing information that may not be contained in population. The population serves as a organism with distributed knowledge: knowledge is distributed throughout the genes of the entire population.

Recently, genetic algorithms have been removed from their original context and studied in a function optimization context. Much effort has gone into refining the original simple genetic algorithm to speed convergence and improve robustness. Two general approaches toward improving genetic algorithm performance have been 1) development of adaptive mechanisms within the genetic algorithm and 2) optimization of static parameters such as mutation rate or population size. Among the ideas that have been introduced are fitness scaling, generation gap, and rank selection.

Fitness scaling was introduced to adapt the selective pressure according to the worst member of a population. The fitness used for selection is computed relative to the fitness measures the rest of the population. As the entire population moves towards higher fitness values, the pressure to select the better solutions is preserved -- avoiding the evolution of a population of mediocre individuals. Generation gap is another mechanism to prevent premature convergence by allowing structures from the current generation to automatically advance to the next generation.

Baker investigated a selection scheme based solely on rank [3]. Members are sorted according to their fitness values and then a member's rank is used to compute the expected number of offspring it will contribute to the next generation. In this work, Baker compared his selection scheme to others. The results of this work did not show that this method was better than the others; however it did show immunity to the selection problems previously mentioned. For a more complete comparison of selection methods see [7].

Fogarty has studied the effects of a adaptive mutation operators [5]. In this work, several mutation operators were compared; constant mutation rate, mutation rate varied over generation, mutation rate varied over integer representation, and mutation rate varied over generation and integer representation. The results of his experiments

showed that dynamically changing the mutation rate always showed an improvement over the constant case.

Goldberg et al. investigated the effect of population size on GA behavior [8]. A population sizing equation was derived that considers variance of building-block fitness, noise of the genetic operators, and noise in the objective function. The authors determined that with small populations, the GA performance is determined by chance, by mutation, or by another mechanism that serially injects diversity. Among several extensions to this work that were outlined, an online population sizing technique was proposed. The online sizing of the population could be based on information about the problem size, population variance, minimum signal, and order of deception.

A number of researchers have proposed encoding meta-level knowledge directly into the genetic code. The meta-level parameters are subjected to genetic operations as are the application parameters. Bagley proposed including crossover and mutation probabilities in the genetic code [2]. Schaffer and Morishima investigate an adaptive crossover scheme in which crossover points are contained in an extended bit representation in [18]. An additional binary code, equal in length to the original genetic code, is appended to the original code. This extension contains marks or punctuations where crossover points are valid.

Perhaps the first to study genetic algorithms and compare them with conventional gradient techniques is De Jong [4]. In this study, De Jong proposed a set of five functions to represent a wide cross section of function families [4,6]. He designed two measures to quantify a search technique's performance: *online* performance to measure ongoing performance and *offline* performance to measure convergence. Online performance is the running average of all evaluations performed up to a given time:

$$x_{online}(s, e, T) = \frac{1}{T}\sum_{t=1}^{T} f_e(t) \tag{1}$$

where $s$ is the search strategy, $e$ is the environment, $f_e(t)$ is the objective function value at time $t$,. This measure may be appropriate in situations where the cost of evaluating a structure, or population member, is related in a monotonically increasing way to its fitness value (i.e., evaluating a poor solution is more expensive than evaluating a good one). Offline performance is the running average of the best performance value up to a given time:

$$x_{offline}(s, e, T) = \frac{1}{T}\sum_{t=1}^{T} f_e^*(t) \tag{2}$$

where $f_e^*(t)$ is the best function value obtained up to time $t$ and $T$ is the current number of evaluations. This measure can be used when there is no additional cost for evaluating poor structures.

De Jong then explored empirically the relationship between genetic algorithm performances, with respect to these measures, and genetic algorithm parameter settings. From these experiments, the following parameter settings for genetic algorithms were given as settings that give good online and offline performance on the test suite and were subsequently used as common settings: population size=50~100, crossover rate=0.6, mutation rate=0.001.

Grefenstette extended this work by using a meta-level genetic algorithm to find parameter settings that yield robust and high performance genetic algorithms [9]. The task of the meta-level genetic algorithm was to determine population size, crossover rate, mutation rate, window size, generation gap, and selection strategy settings that maximized either the online or offline performance over a range of applications. To measure the quality of a genetic algorithm, Grefenstette formulated a fitness function based on De Jong's online and offline performance measures. This fitness function normalized the genetic algorithm's performance measure on a given application environment with respect to the same performance measure obtained by random search on the same application environment:

$$fitness(s) = \sum_{i=1}^{N} \frac{x_{perf}(s, f_i, T)}{x_{perf}(rand, f_i, T)} \tag{3}$$

where $x_{perf}$ represents the search performance measure, $s$ represents a particular search strategy search strategy, $f_i$ represents the application environments, and $T$ represents the number of evaluations a search was allowed to execute. *rand* represents a random search strategy. In Grefenstette's experiments, the five De Jong functions were used as the application environments.

His results gave parameter settings for best online performance which resulted to be population size=20~30, crossover rate=0.75~0.95, mutation rate=0.0005~0.01 for good online performance and population size=80, crossover rate=0.45, mutation rate=0.01 for good offline performance. In this work, Grefenstette also optimized parameter settings for window size, generation gap, and selection strategy and were given as (window size=7, generation gap=1.0, elite selection) for online and (window size=1, generation gap=0.9, pure selection) for offline respectively. (We will refer to these genetic algorithms as optimized static genetic algorithms.) These are static settings; the settings are entered at the beginning of the run and then are left unchanged throughout the run.

In our work, we extend Grefenstette's research by investigating methods that develop strategies for dynamically changing the genetic algorithm control parameter settings. In the next section, we describe the Dynamic Parametric Genetic Algorithm, our framework for representing genetic algorithm control strategies.

## 3 The Dynamic Parametric Genetic Algorithm

In this section, we present the Dynamic Parametric Genetic Algorithm (DPGA): a framework for representing genetic algorithm control strategies. The DPGA represents a class of genetic algorithms that uses a fuzzy knowledge-base to control its parameters dynamically (see Figure 1). In this figure, a fuzzy inference engine controls the

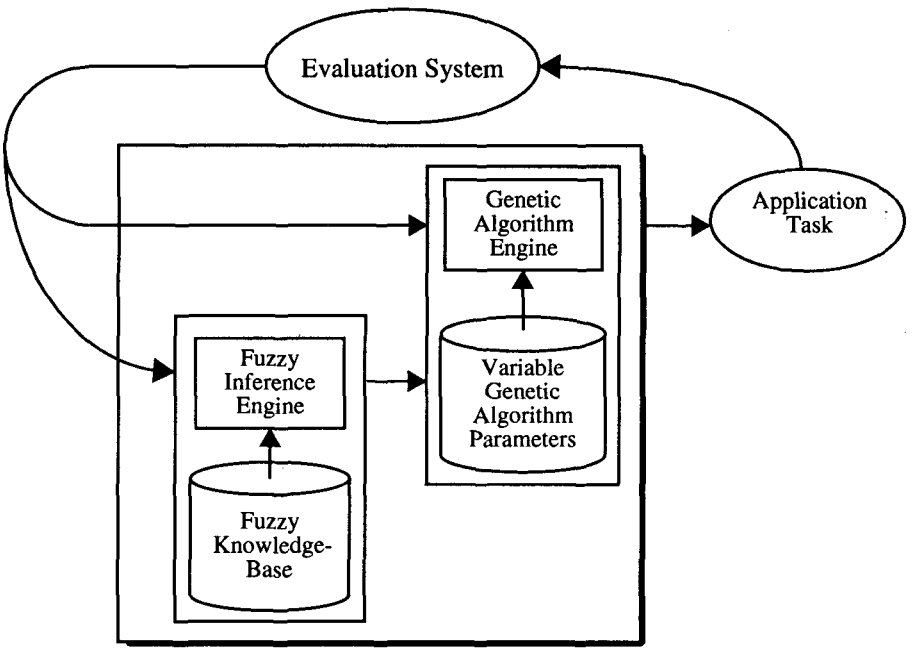


**Fig. 1.** The Dynamic Parametric GA framework: a genetic algorithm with dynamic parameters controlled by a fuzzy knowledge-based system. The fuzzy knowledge based system monitors performance measures from the evaluation system to control genetic algorithm parameters such as population size, mutation rate, or crossover rate.

parameters of a genetic algorithm, such as population sizing and mutation rate. The inputs to the fuzzy system are measures of the genetic algorithm search performance, which we assume can be derived from the quality of the solutions the genetic algorithm generates. The evaluation system is responsible for assigning fitness scores to solutions generated by the genetic algorithm.

Our decision to represent strategies using a fuzzy knowledge-base is predicated on three main points; fuzzy systems:

1. are rich enough to represent a broad range of strategies,
2. facilitates the capture and representation of expert knowledge through the use of linguistic rules,
3. can be automatically tuned and design using any number of numerous methods found in the literature.

Using the expressive richness of fuzzy systems gives us the ability to represent existing control strategies within a single framework. Because of this generality, complex combinations of heuristic strategies can be implemented and compared. With automatic tuning and design techniques available for fuzzy systems, expert knowledge contained in the system can be both refined and augmented. For example, dynamic heuristic strategies of the following form may be represented within our framework:

- **if** generations is low **then** mutation rate is high
- **if** generations is medium **then** mutation rate is medium
- **if** generations is high **then** mutation rate is low
- **if** diversity is low **then** increase mutation
- **if** population average is not increasing **then** increase population size.

Other possibilities could be to control selection of operators and their parameters or even parameterized representations. The point is that the framework we propose is able to self-adapting to achieve a desired search behavior.

## 4 Automatic Acquisition of High Performance Control Strategies

In this section, we present an instance of the class of DPGA and demonstrate how to automatically acquire high performance genetic algorithm control strategies. The example DPGA has dynamic population sizing, crossover rate, and mutation rate. The fuzzy knowledge-based system used to control the DPGA outputs control actions in the form of percent change from current value: change in population size, crossover rate change, and mutation rate change. The ranges of the outputs are set such that the control parameters could not decrease by more than half their current value and could not increase be more than twice their current value. Population size, mutation rate, and crossover rate are bounded by the following ranges respectively: [2,160], [0.0001,1.0], and [0.0001,1.0]. Because the outputs represent changes in values as opposed to absolute values, the initial values of these parameters must be specified. The inputs to fuzzy knowledge-base are population diversity measures in the form of best to average ratio, average to worst ratio, and change in best solutions since the last generation (see Figure 2):

$$x_0 = \frac{\text{average fitness}}{\text{best fitness}} \tag{4}$$

$$x_1 = \frac{\text{worst fitness}}{\text{average fitness}} \tag{5}$$

$$x_2 = \Delta\text{best fitness since last control action} \tag{6}$$

Our technique for automatically acquiring control strategies is based on a fuzzy system design technique proposed by Lee and Takagi [15,16]. This technique uses genetic

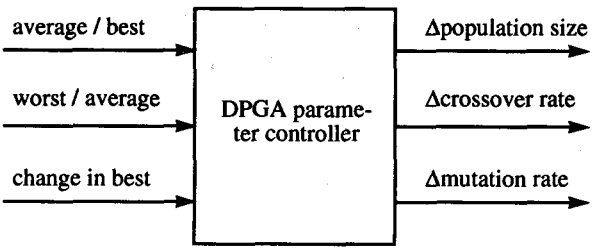


**Fig. 2.** Input and outputs to fuzzy knowledge-based system used in our Dynamic Parametric GA (DPGA) experiments. Inputs measure characteristics of the performance of the genetic algorithm on the environment. Outputs control the population size, crossover rate, and mutation rate.

algorithms to design membership functions, consequent parameters and number of rules of a fuzzy systems. To use this technique, we must design both a genetic representation of the DPGA and a fitness measure. These two points will be covered in the next two subsections.

### 4.1 DPGA System Parameterization and Genetic Coding

Our example DPGA requires two types of specifications: initial condition values for the three control parameters and a fuzzy system. Coding the initial value parameters is straight forward. For the fuzzy knowledge-base each of the three input and three output intervals in our example system can be covered a maximum of three fuzzy sets. The sets are constrained to overlap in such a way that exactly two of the membership functions have non-zero values and three sum of their memberships is 1. The maximum number of rules is determined by the number of output variables multiplied by the maximum number of input set combinations (assuming exactly one set per input dimension is present in the antecedent part of each rule). Each rule has exactly one consequent; one output set can be associated with a given input set combination in the antecedent.

The genetic coding of our DPGA follows naturally from the parameterization. We use a binary coding that includes alleles for initial conditions and alleles for the fuzzy knowledge-based system. The fuzzy knowledge-based system has alleles that determine membership function shape and rule structure [17]. The rule structure is designed such that an allele determines whether which output set, if any, is used as the consequent of a particular inputs set combination.

The fuzzy system uses triangular membership functions, the *min* intersection operator and correlation-product inference procedure. Defuzzification of the outputs is performed using the fuzzy centroid method [13].

### 4.2 Genetic Algorithm Search Performance Measures

The control strategies learned in our experiments were optimized according to perfor-

mance measures and a five function test suite proposed by De Jong [4]: *online* performance to measure ongoing performance and *offline* performance to measure convergence. Our technique follows that of Grefenstette's where performance of a DPGA is measured as a normalized sum of performance on the De Jong functions relative to random search (See Section 2).

# 5 Results

We design separate Dynamic Parametric GAs for optimizing online and offline performance measures. In this section, we will look at the dynamic behavior of the DPGAs and compare the results with a simple static genetic algorithm proposed by De Jong (SSGA)[4], the optimized static online and offline genetic algorithms proposed by Grefenstette (OSGA)[9], and random search (see Table 1 for GA parameter settings).

**Table 1:** Genetic algorithm search parameter settings for simple static GA (SSGA), optimized static GA for online performance (OSGA online), and optimized static GA for offline performance (OSGA offline).

| parameter | SSGA | OSGA online | OSGA offline |
|---|---|---|---|
| population size | 50 | 30 | 80 |
| crossover rate | 0.6 | 0.95 | 0.45 |
| mutation rate | 0.001 | 0.01 | 0.01 |
| generation gap | 1.0 | 1.0 | 0.9 |
| window size | 7 | 1 | 1 |
| selection strategy | Elite | Elite | Pure |

The genetic algorithm used to design Dynamic Parametric GAs itself had fixed parameters of population size=10, crossover rate=0.8, mutation rate=0.0333. It used an elitist selection strategy and window sizes and generation gaps were fixed at 1 and 1.0 respectively. This genetic algorithm was allowed to evaluate 1000 Dynamic Parametric GAs.

We would like to caution the reader that the specific instances of DPGAs presented in the following section represent only one instance of the class of DPGAs. The DPGAs presented exhibit high performance, however it is very likely that other fuzzy rule sets may exhibit equal or better performance. The point of this exposition is to demonstrate the technique for automatically obtaining genetic algorithm control strategies.

## 5.1 Dynamic Parametric GA for Online Performance

The initial population of fuzzy systems and initial conditions used for determining the good online performance was seeded with an individual identical with the best online performing genetic algorithm determined by Grefenstette. In this individual, the population size was set to 30, and the crossover mutation rates were set to 0.95 and 0.01 respectively. For each fuzzy system produced for the Dynamic Parametric GA, the generation gap, window size, and selection strategy were fixed at 1.0, 1, and Elitist.

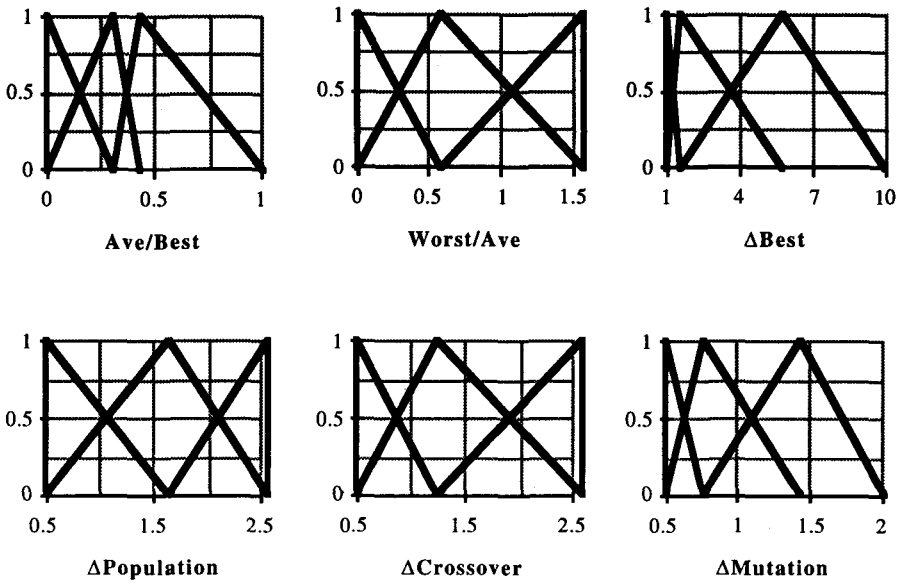


**Fig. 3.** Obtained membership functions for the optimized online fuzzy rule-base. **Ave/Best** represents the ratio between the average fitness and the fitness of the best individual's fitness. **Worst/Ave** represents the ratio between the worst individual's fitness and the average fitness. **ΔBest** represents the amount of change in the best fitness since the last control action. The control actions are **ΔPopulation**, **ΔCrossover**, and **ΔMutation** which control the change in population size, crossover rate, and mutation rate respectively. Each fuzzy set within an input variable is identified from left to right as Small, Medium, or Big. Each fuzzy set within an output variable is identified from left to right as Negative, None, or Positive. See Tables 2, 3, and 4 for rules.

After evaluating 665 fuzzy systems, the meta-level genetic algorithm produced a fuzzy system with the following initial conditions:

| | |
|---|---|
| Initial Population Size: | 10 |
| Initial Crossover Rate: | 0.942647 |
| Initial Mutation Rate: | 0.009903 |

The membership function definitions and rules are given in Figure 3 and Tables 2, 3, and 4. The first three columns represent the antecedent part and the fourth column is the consequent. For example, the first entry in the table should be read: **IF** Ave/Best is Small **AND** Worst/Ave is Small **AND** ΔBest is Small **THEN** change in population size is None.

According to the aggregate online fitness measure defined by (3), this Dynamic Para-

**Table 2:** Fuzzy rules for controlling population size in online Dynamic Parametric GA (see Figure 3 for membership function definition).

| Ave/Best | Worst/Ave | ΔBest | ΔPopulation |
|---|---|---|---|
| Small | Small | Small | None |
| Medium | Small | Small | Negative |
| Small | Medium | Small | Positive |
| Small | Big | Small | Positive |
| Medium | Big | Small | None |
| Small | Small | Medium | Positive |
| Medium | Small | Medium | Negative |
| Big | Small | Medium | Positive |
| Small | Medium | Medium | Positive |
| Medium | Medium | Medium | Negative |
| Big | Medium | Medium | Negative |
| Small | Big | Medium | Negative |
| Medium | Big | Medium | Positive |
| Small | Small | Big | Positive |
| Small | Medium | Big | None |
| Medium | Small | Big | None |
| Big | Small | Big | Positive |
| Medium | Medium | Big | None |
| Big | Medium | Big | Positive |
| Small | Big | Big | None |
| Medium | Big | Big | Positive |
| Big | Big | Big | Positive |

**Table 3:** Fuzzy rules for controlling crossover rate in online Dynamic Parametric GA (see Figure 3 for membership function definition).

| Ave/Best | Worst/Ave | ΔBest | ΔCrossover |
|---|---|---|---|
| Small | Small | Small | Positive |
| Medium | Small | Small | Positive |
| Big | Small | Small | Positive |
| Small | Medium | Small | Negative |
| Medium | Medium | Small | None |
| Medium | Big | Small | Negative |
| Small | Small | Medium | None |
| Small | Medium | Medium | Negative |
| Medium | Medium | Medium | Negative |
| Big | Medium | Medium | Negative |
| Small | Big | Medium | None |
| Medium | Big | Medium | None |
| Big | Big | Medium | None |
| Small | Small | Big | Positive |
| Medium | Small | Big | Negative |

**Table 3:** Fuzzy rules for controlling crossover rate in online Dynamic Parametric GA (see Figure 3 for membership function definition).

| Ave/Best | Worst/Ave | ΔBest | ΔCrossover |
|---|---|---|---|
| Medium | Big | Big | Positive |
| Big | Big | Big | Positive |

**Table 4:** Fuzzy rules for controlling mutation rate in online Dynamic Parametric GA (see Figure 3 for membership function definition).

| Ave/Best | Worst/Ave | ΔBest | ΔMutation |
|---|---|---|---|
| Small | Small | Small | Negative |
| Big | Small | Small | None |
| Medium | Medium | Small | Negative |
| Small | Big | Small | Negative |
| Medium | Big | Small | Negative |
| Big | Big | Small | None |
| Small | Small | Medium | Positive |
| Medium | Small | Medium | None |
| Big | Small | Medium | Positive |
| Medium | Medium | Medium | Negative |
| Big | Medium | Medium | Negative |
| Small | Big | Medium | Positive |
| Medium | Big | Medium | None |
| Big | Big | Medium | None |
| Small | Medium | Big | Negative |
| Small | Small | Big | None |
| Medium | Small | Big | Positive |
| Medium | Medium | Big | None |
| Medium | Big | Big | Negative |
| Big | Big | Big | None |

metric GA exhibited a 380% increase in performance over the optimized static genetic algorithm (calculated by dividing the Dynamic Parametric GA performance by the optimized static genetic algorithm performance) (see Table 5). Data for each of the

**Table 5:** Online performance measures for optimized static GA (OSGA) and Dynamic Parametric GA (DPGA).

| Function | OSGA | DPGA |
|---|---|---|
| 1 | 1069.525635 | 7555.996094 |
| 2 | 1479.180542 | 7952.011230 |
| 3 | 888.739319 | 1463.375610 |
| 4 | 579.341919 | 565.375854 |
| 5 | 833.273376 | 905.430664 |

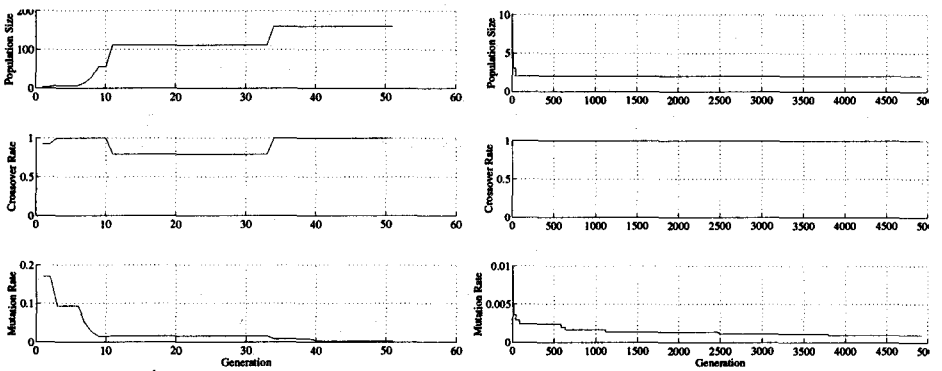


**Fig. 4.** Online performance of the Dynamic Parametric GA (DPGA) optimized for online performance on De Jong Function 3. Also shown are the online performance measures of the optimized static GA (OSGA online), simple static GA (SSGA), and random search (see Table 1 for GA search parameters). The plots on the right show the dynamic control of the population size, crossover rate, and mutation rate for a typical DPGA run on De Jong Function 3.

entries represent values averaged over ten independent runs starting from different initial conditions. The initial population of fuzzy systems and initial conditions used for determining the good online performance was seeded with an individual with static settings as prescribed by 'OSGA online' given in Table 1 (the system had no active rules and initial conditions set as prescribed by 'OSGA online'). For each fuzzy system produced for the Dynamic Parametric GA, the generation gap, window size, and selection strategy were fixed at 1.0, 1, and Elitist. After evaluating 665 fuzzy systems, the meta-level genetic algorithm produced a fuzzy system with 59 rules and the following initial conditions[14]:

| | |
|---|---|
| Initial Population Size: | 10 |
| Initial Crossover Rate: | 0.942647 |
| Initial Mutation Rate: | 0.009903 |

The left plot of Figure 4 shows the online performance vs. evaluations for the DPGA, OSGA online, SSGA, and random search for De Jong Function 3. The data in the figure is averaged from running each GA with ten different initial conditions.

The plots on the right show the dynamic control of the population sizing, crossover rate, and the mutation rate for a typical run on De Jong Function 3. Both the population size and mutation rate decrease toward the minimum value while the crossover rate remains high. The strategy that this particular DPGA has chosen is a conservative approach. Because the elite selection strategy is enabled and the population size goes to two, the search becomes a greedy hill climber. A good solution is not abandoned until a better one is found. In addition, the low mutation rate keeps the exploration relatively local.

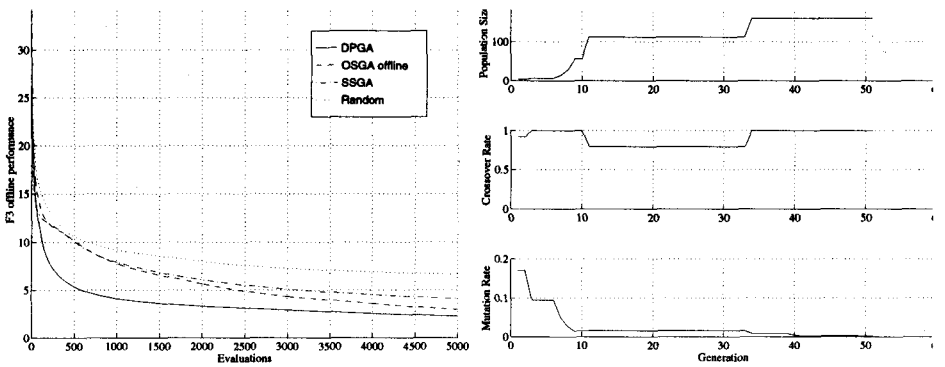


**Fig. 5.** Offline performance of the Dynamic Parametric GA (DPGA) optimized for offline performance on De Jong Function 3. Also shown are the offline performance measures of the optimized static GA (OSGA offline), simple static GA (SSGA), and random search (see Table 1 for GA search parameters). The plots on the right show the dynamic control of the population size, crossover rate, and mutation rate for a typical DPGA run on De Jong Function 3.

### 5.2 Dynamic Parametric GA for Offline Performance

As with the online search, the initial population of fuzzy systems and initial conditions used for determining the good offline performance was seeded with an individual identical with static settings, i.e. no rules, as prescribed by 'OSGA offline' given in Table 1. For each fuzzy system produced for the Dynamic Parametric GA, the generation gap, window size, and selection strategy were fixed at 0.9, 1, and Pure. After evaluating 373 fuzzy systems, the meta-level genetic algorithm produced a fuzzy system with 68 rules and the following initial conditions[14]:

| | |
|---|---|
| Initial Population Size: | 4 |
| Initial Crossover Rate: | 0.922059 |
| Initial Mutation Rate: | 0.170671 |

Figure 5 shows the offline performance vs. evaluations for the DPGA, OSGA offline, SSGA, and random search for De Jong Function 3 (as with the online data, this data is averaged over ten runs). The plots on the right show the dynamic control of the population sizing, crossover rate, and the mutation rate. As in the online control strategy, the mutation rate decrease toward the minimum value while the crossover rate remains high. However, the population size increases toward the maximum value. As the number of evaluations increases, random search becomes more difficult to out-perform. Although we expected the mutation rate to increase over time (a move toward random search behavior) we found that the control strategy relied more on the crossover operator than the mutation as it continued its search.

# 6 The Interaction Effects of Dynamic Crossover, Mutation, and Population Sizing

In this section, we present experimental results on different combinations of dynamic control parameters on offline performance. We compare eight combinations of results as shown in Table 6.

**Table 6:** Experimental combinations of dynamic parameters.

| | experiment identifier | | | | | | |
|---|---|---|---|---|---|---|---|
| control parameters | sss | dss | sds | ssd | dds | sdd | ddd |
| population | static | dynamic | static | static | dynamic | static | dynamic |
| crossover | static | static | dynamic | static | dynamic | dynamic | dynamic |
| mutation | static | static | static | dynamic | static | dynamic | dynamic |

The experiment identifier reflects which control parameters were allowed to change and which fixed for a particular algorithm. These experimental results were generated by disabling the appropriate dynamic control of the DPGA. Figure 6 compares the offline performance for Function 3 using each of the combinations averaged over 20 different runs. The observed performance behaviors separated into three categories: not so good, good, and best (figures (a), (b), and (c) in Figure 6 respectively). The figure shows that in the case of adding only dynamic population sizing, only a marginal improvement is made. Also, in general, modulating the crossover rate offers moderate improvements. In addition, it is usually the case that adding dynamic mutation always results in a performance gain.

# 7 Conclusions and Extensions

We have presented a framework for controlling genetic algorithm and demonstrated how to apply it to study the interdependencies of dynamic crossover, mutation, and population sizing. Our framework includes techniques to automatically design genetic algorithm control strategies according to a given search performance metric. Using these techniques, we have designed adaptive genetic algorithms, referred to as Dynamic Parametric Genetic Algorithms, with control strategies that show improved performance over simple static and optimized static GAs. Further comparisons with other GA methods are warranted.

Results from experiments aimed at isolating the effects of dynamic population sizing, mutation rate, and crossover rate shows that the dynamic mutation contributes most to high online and offline performance.

We would like to emphasize that the experimental results we report have been obtained for a specific instance of the class of Dynamic Parametric GAs; our technique can be applied to systems with other inputs and outputs. Research on rule compaction and determining relevant input variables should be explored. Also, more analysis needs to be performed on the resulting systems. In addition, obtaining search behaviors other than ones directed at high offline and online measures warrant investigation.

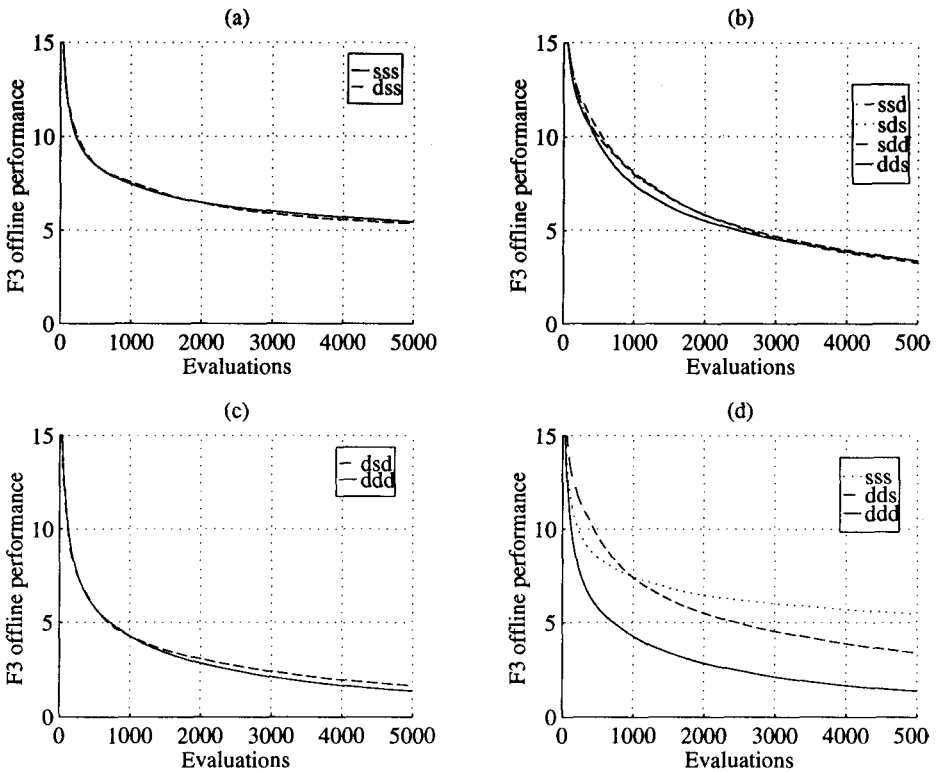


**Fig. 6.** Comparison plots of DPGAs with different dynamic control parameters. The performance characteristics fall into three categories. Each category is shown in plots (a), (b), and(c). Plot (d) groups selected GAs from each of the categories.

The framework presented in this paper can be extended and generalized to realize self-adapting learning algorithms based on other learning algorithms such as simulated annealing or gradient techniques. The underlying benefit that this technique offers is a method to tune and adapt a particular learning algorithm to the problem at hand. The initial cost of using our method to design an adaptive learning algorithm can be high, however, the procedure need not be performed frequently. The goal of our technique is to design a learning algorithm that can be repeatedly applied to a particular class of problems, such as fuzzy system design using a specific fuzzy system representation.

It is known that the efficiency and effectiveness of a particular learning algorithm hinge on:1) the representations they operate on, 2) the settings of the learning algorithm parameters, and 3) the error functions used to drive the learning. Although all instances of the same algorithm are conceptually the same, in practice, systematic differences are always present. Our method provides a technique that can minimize the effects of systematic deficiencies to obtain higher performance learning algorithms.

## 8 Acknowledgments

This research is supported in part by NASA Grant NCC-2-275, EPRI Agreement RP8010-34, and BISC program. The authors would also like to thank Professor David Wessel and the Center for New Music and Audio Technologies at UC Berkeley and Silicon Graphics Inc. for use of computing resources.

## 9 References and Related Publications

[1] Back, T., "Self-adaptation in genetic algorithms," *Proceedings of the First European Conference on Artificial Life*, Paris, France, 1991, pp.263-271.

[2] Bagley, J. D., *The behavior of adaptive systems which employ genetic and correlation algorithms*, Ph.D. Thesis, University of Michigan, 1967.

[3] Baker, J. E., "Adaptive selection method for genetic algorithms," *Proceedings of an International Conference on Genetic Algorithms* (ICGA'85), 1985, pp.101-111.

[4] De Jong, K. A., *An analysis of the behavior of a class of genetic adaptive systems*, Ph.D. Thesis, University of Michigan, 1975.

[5] Fogarty, T. C., "Varying the probability of mutation in the genetic algorithm," *Proceedings of the Third International Conference on Genetic Algorithms* (ICGA'89), 1989, pp.104-109.

[6] Goldberg, D. E., *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Reading, MA, 1989.

[7] Goldberg, D. E. and Deb, K., "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms* (FOGA'90), ed. Rawlins,G., Morgan Kaufmann, San Mateo, CA, 1991, pp.69-93.

[8] Goldberg, D. E., Deb, K. and Clark, J. H., "Genetic algorithms, noise, and the sizing of populations," *Complex Systems*, Vol.6, No.4, 1992, pp. 333-362.

[9] Grefenstette, J. J., "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol.16, No.1, 1986, pp. 122-128.

[10] Hesser, J. and Manner, R., "Towards an optimal mutation probability for genetic algorithms," *Parallel Problem Solving from Nature. 1st Workshop, PPSN 1 Proceedings*, Dortmund, West Germany, 1990, pp.23-32.

[11] Holland, J. H., *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA, 1975.

[12] Karr, C., Freeman, L. and Meredith, D., "Improved Fuzzy Process Control of Spacecraft Autonomous Rendezvous Using a Genetic Algorithm," *SPIE Intelligent Control and Adaptive systems*, 1989.

[13] Kosko, B., *Neural Networks and Fuzzy Systems*, Addison-Wesley, Englewood Cliffs, NJ, 1992.

[14] Lee, M.A., *Automatic Design and Adaptation of Fuzzy Systems and Genetic Algorithms using Soft Computing Techniques*, Ph.D. Thesis, University of California, Davis, 1994.

[15] Lee, M. A. and Takagi, H., "Embedding A Priori Knowledge into an Integrated Fuzzy System Design Method based on Genetic Algorithms," *Proc. of the IFSA Congress* (IFSA'93), Seoul, Korea, 1993, pp.1293-1296.

[16] Lee, M. A. and Takagi, H., "Integrating design stages of fuzzy systems using genetic algorithms," *Proc. IEEE Int. Conf. on Fuzzy Systems* (FUZZ-IEEE '93), San Francisco, CA, 1993, pp.612-617.

[17] Lee, M. A. and Takagi, H., "Dynamic Control of Genetic Algorithms using Fuzzy Logic Techniques," *Proceedings of the Fifth International Conference on Genetic Algorithms* (ICGA'93), Morgan-Kaufmann, San Mateo, CA, 1993, pp.76-83.

[18] Schaffer, J. D. and Morishima, A., "An adaptive crossover distribution mechanism for genetic algorithms," *Proceedings of the Second International Conference on Genetic Algorithms* (ICGA'87), MIT, Cambridge, MA, 1987, pp.36-40.