Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms

M. Srinivas, and L. M. Patnaik, *Fellow, IEEE*

***Abstract*— In this paper we describe an efficient approach for multimodal function optimization using Genetic Algorithms (GAs). We recommend the use of adaptive probabilities of crossover and mutation to realize the twin goals of maintaining diversity in the population and sustaining the convergence capacity of the GA. In the Adaptive Genetic Algorithm (AGA), the probabilities of crossover and mutation, p_c and p_m, are varied depending on the fitness values of the solutions. High-fitness solutions are 'protected', while solutions with subaverage fitnesses are totally disrupted. By using adaptively varying p_c and p_m, we also provide a solution to the problem of deciding the optimal values of p_c and p_m, i.e., p_c and p_m need not be specified at all. The AGA is compared with previous approaches for adapting operator probabilities in genetic algorithms. The sShema theorem is derived for the AGA, and the working of the AGA is analyzed.**

We compare the performance of the AGA with that of the Standard GA (SGA) in optimizing several nontrivial multimodal functions with varying degrees of complexity. For most functions, the AGA converges to the global optimum in far fewer generations than the SGA, and it gets stuck at a local optimum fewer times. Our experiments demonstrate that the relative performance of the AGA as compared to that of the SGA improves as the epistacity and the multimodal nature of the objective function increase. We believe that the AGA is the first step in realizing a class of self organizing GAs capable of adapting themselves in locating the global optimum in a multimodal landscape.

I. Introduction

GENETIC Algorithms [2], [7], [10], [17] are robust search and optimization techniques which are finding application in a number of practical problems. The robustness of Genetic Algorithms (hereafter referred to as GAs) is due to their capacity to locate the global optimum in a multimodal landscape. A plethora of such multimodal functions exist in engineering problems (optimization of neural network structure and learning neural network weights, solving optimal control problems, designing structures, and solving flow problems) are a few examples. It is for the above reason that considerable attention has been paid to the design of GAs for optimizing multimodal functions.

GAs employ a random, yet directed, search for locating the globally optimal solution. They are superior to 'gradient descent' techniques as the search is not biased towards the locally optimal solution. On the other hand, they differ from random sampling algorithms due to their ability to direct the search towards relatively 'prospective' regions in the search space.

Typically a GA is characterized by the following components:

- a genetic representation (or an encoding) for the feasible solutions to the optimization problem
- a population of encoded solutions
- a fitness function that evaluates the optimality of each solution
- genetic operators that generate a new population from the existing population
- control parameters.

The GA may be viewed as an evolutionary process wherein a population of solutions evolves over a sequence of generations. During each generation, the fitness of each solution is evaluated, and solutions are *selected* for reproduction based on their fitness. *Selection* embodies the principle of 'Survival of the fittest.' 'Good' solutions are *selected* for reproduction while 'bad' solutions are eliminated. The 'goodness' of a solution is determined from its fitness value. The selected solutions then undergo recombination under the action of the *crossover* and *mutation* operators. It has to be noted that the genetic representation may differ considerably from the natural form of the parameters of the solutions. Fixed-length and binary encoded strings for representing solutions have dominated GA research since they provide the maximum number of schemata and as they are amenable to simple implementation.

The power of GAs arises from crossover. Crossover causes a structured, yet randomized exchange of genetic material between solutions, with the possibility that 'good' solutions can generate 'better' ones. The following sentences from [10, pp. 13] aptly summarize the working of GAs:
"..., *the population contains not just a sample of n ideas, rather it contains a multitude of notions and rankings of those notions for task performance. Genetic Algorithms ruthlessly exploit this wealth of information by* 1) *reproducing high quality notions according to their performance and* 2) *crossing these notions with many other high-performance notions from other strings*."

Crossover occurs only with some probability p_c (the crossover probability or crossover rate). When the solutions are not subjected to crossover, they remain unmodified. Notable crossover techniques include the single-point, the two-point, and the uniform types [23].

Manuscript received August 4, 1991: revised August 28, 1992, February 25, 1993, and June 11, 1993. Recommended by Associate Editor Bezdek.

M Srinivas is with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India

L. M. Patnaik is with the Microprocessor Applications Laboratory, Indian Institute of Science, Bangalore 560 012, India.

IEEE Log Number 9400454.

```
Simple Genetic Algorithm ()
{
initialize population;
evaluate population ;
while convergence not achieved
        {
        scale population fitnesses ;
        select solutions for next population ;
        perform crossover and mutation ;
        evaluate population ;
        }
}
```

Fig. 1. Basic structure of a GA.

Mutation involves the modification of the value of each 'gene' of a solution with some probability p_m (the mutation probability). The role of mutation in GAs has been that of restoring lost or unexplored genetic material into the population to prevent the premature convergence of the GA to suboptimal solutions.

Apart from selection, crossover, and mutation, various other auxiliary operations are common in GAs. Of these, *scaling* mechanisms [16] are widely used. *Scaling* involves a readjustment of fitness values of solutions to sustain a steady selective pressure in the population and to prevent the premature convergence of the population to suboptimal solutions.

The basic structure of a GA is illustrated in Fig. 1.

In this paper we describe an efficient technique for multimodal function optimization using GAs. We recommend the use of adaptive probabilities of crossover and mutation to realize the twin goals of maintaining diversity in the population and sustaining the convergence capacity of the GA. With the approach of adaptive probabilities of crossover and mutation, we also provide a solution to the problem of choosing the optimal values of the probabilities of crossover and mutation (hereafter referred to as p_c and p_m respectively) for the GA. The choice of p_c and p_m is known to critically affect the behavior and performance of the GA, and a number of guidelines exist in the literature for choosing p_m and p_c [6], [8], [10], [16], [22]. These generalized guidelines are inadequate as the choice of the optimal p_c and p_m becomes specific to the problem under consideration. Grefenstette [16] has formulated the problem of selecting p_c and p_m as an optimization problem in itself, and has recommended the use of a second-level GA to determine the parameters of the GA. The disadvantage of Grefenstette's method is that it could prove to be computationally expensive. In our approach, p_c and p_m are determined adaptively by the GA itself, and the user is relieved of the burden of specifying the values of p_c and p_m.

The paper is organized as follows. In Section II we discuss the problems of multimodal function optimization, and the various techniques proposed in the literature to overcome the problems. Section III describes our approach of using adaptively varying probabilities of crossover and mutation for multimodal function optimization. In Section IV we compare the AGA with previous techniques at adapting operator probabilities in GAs. In Section V we derive the Schema theorem for GA and analyze the variation of schema fitnesses. In Section VI, we present experimental results to compare the performance of the GAs with and without adaptive probabilities of crossover and mutation. The conclusions and directions for future work are presented in Section VII.

II. Genetic Algorithms and Multimodal Function Optimization

In optimizing unimodal functions, it is important that the GA should be able to converge to the optimum in as few generations as possible. For multimodal functions, there is a need to be able to locate the region in which the global optimum exists, and then to converge to the optimum. GAs possess hill-climbing properties essential for multimodal function optimization, but they too are vulnerable to getting stuck at a local optimum (notably when the populations are small). In this section, we discuss the role of the parameters p_c and p_m (probabilities of crossover and mutation) in controlling the behavior of the GA. We also discuss the techniques proposed in the literature for enhancing the performance of GAs for optimizing multimodal functions.

The significance of p_c and p_m in controlling GA performance has long been acknowledged in GA research [7], [10]. Several studies, both empirical [16], [22] and theoretical [20] have been devoted to identify optimal parameter settings for GAs. The crossover probability p_c controls the rate at which solutions are subjected to crossover. The higher the value of p_c, the quicker are the new solutions introduced into the population. As p_c increases, however, solutions can be disrupted faster than selection can exploit them. Typical values of p_c are in the range 0.5–1.0. Mutation is only a secondary operator to restore genetic material. Nevertheless the choice of p_m is critical to GA performance and has been emphasized in DeJong's inceptional work [6]. Large values of p_m transform the GA into a purely random search algorithm, while some mutation is required to prevent the premature convergence of the GA to suboptimal solutions. Typically p_m is chosen in the range 0.005–0.05.

Efforts to improve the performance of the GA in optimizing multimodal functions date back to DeJong's work [6]. DeJong introduced the ideas of 'overlapping populations' and 'crowding' in his work. In the case of 'overlapping populations', newly generated offspring replace similar solutions of the population, primarily to sustain the diversity of solutions in the population and to prevent premature convergence. The technique however introduces a parameter CF (the crowding factor), which has to be tuned to ensure optimal performance of the GA. The concept of 'crowding' led to the ideas of 'niche' and 'speciation' in GAs. Goldberg's 'sharing function' has been employed in the context of multimodal function optimization; [15] describes a method of encouraging 'niche' formation and 'speciation' in GAs. More recently, Goldberg has proposed a Boltzmann tournament selection scheme [11] for forming and sizing stable sub-populations. This technique is based on ideas from simulated annealing and promises convergence to the global optimum.

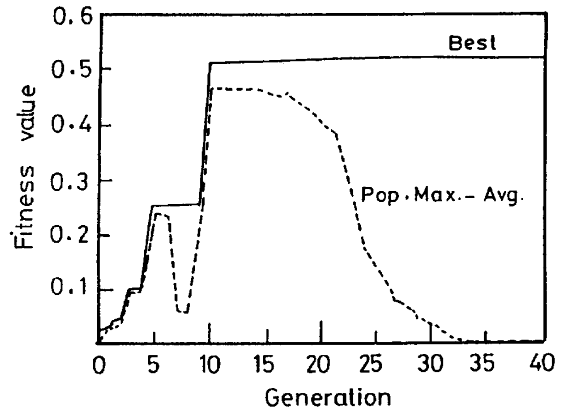


Fig. 2. Variation of $f_{\max} - \overline{f}$ and f_{best} (best fitness).

In all the techniques described above, no emphasis is placed on the choice of p_m and p_c. The choice of p_c and p_m is still left to the user to be determined statically prior to the execution of the GA. The idea of adaptive operators to improve GA performance has been employed earlier [1] [3] [9] [24]. Our approach to multimodal function optimization also uses adaptive probabilities of crossover and mutation, but in a manner different from these previous approaches. We devote Section IV to discuss the above approaches, and compare them with the AGA. In the next section, we discuss the motivation for having adaptive probabilities of crossover and mutation, and describe the methods adopted to realize them.

III. Adaptive Probabilities of Crossover and Mutation

A. Motivations

It is essential to have two characteristics in GAs for optimizing multimodal functions. The first characteristic is the capacity to converge to an optimum (local or global) after locating the region containing the optimum. The second characteristic is the capacity to explore new regions of the solution-space in search of the global optimum. The balance between these characteristics of the GA is dictated by the values of p_m and p_c, and the type of crossover employed [23]. Increasing values of p_m and p_c promote exploration at the expense of exploitation. Moderately large values of p_c (0.5–1.0) and small values of p_m (0.001–0.05) are commonly employed in GA practice. In our approach, we aim at achieving this trade-off between exploration and exploitation in a different manner, by varying p_c and p_m adaptively in response to the fitness values of the solutions; p_c and p_m are increased when the population tends to get stuck at a local optimum and are decreased when the population is scattered in the solution space.

B. Design of Adaptive p_c and p_m

To vary p_c and p_m adaptively, for preventing premature convergence of the GA to a local optimum, it is essential to be able to identify whether the GA is converging to an optimum. One possible way of detecting convergence is to observe the average fitness value $\overline{f}$ of the population in relation to the maximum fitness value $f_{\max}$ of the population. $f_{\max} - \overline{f}$ is likely to be less for a population that has converged to an optimum solution than that for a population scattered in the solution space. We have observed the above property in all our experiments with GAs, and Fig. 2 illustrates the property for a typical case. In Fig. 2, we notice that $f_{\max} - \overline{f}$ decreases when the GA converges to a local optimum with a fitness value of 0.5 (The globally optimal solution has a fitness value of 1.0). We use the difference in the average and maximum fitness values, $f_{\max} - \overline{f}$, as a yardstick for detecting the convergence of the GA. The values of p_c and p_m are varied depending on the value of $f_{\max} - \overline{f}$. Since p_c and p_m have to be increased when the GA converges to a local optimum, i.e., when $f_{\max} - \overline{f}$ decreases, p_c and p_m will have to be varied inversely with $f_{\max} - \overline{f}$. The expressions that we have chosen for p_c and p_m are of the form

$$p_c = k_1/(f_{\max} - \overline{f})$$

and

$$p_m = k_2/(f_{\max} - \overline{f}).$$

It has to be observed in the above expressions that p_c and p_m do not depend on the fitness value of any particular solution, and have the same values for all the solutions of the population. Consequently, solutions with high fitness values as well as solutions with low fitness values are subjected to the same levels of mutation and crossover. When a population converges to a globally optimal solution (or even a locally optimal solution), p_c and p_m increase and may cause the disruption of the near-optimal solutions. The population may never converge to the global optimum. Though we may prevent the GA from getting stuck at a local optimum, the performance of the GA (in terms of the generations required for convergence) will certainly deteriorate.

To overcome the above-stated problem, we need to preserve 'good' solutions of the population. This can be achieved by having lower values of p_c and p_m for high fitness solutions and higher values of p_c and p_m for low fitness solutions. While the high fitness solutions aid in the convergence of the GA, the low fitness solutions prevent the GA from getting stuck at a local optimum. The value of p_m should depend not only on $f_{\max} - \overline{f}$, but also on the fitness value f of the solution. Similarly, p_c should depend on the fitness values of both the parent solutions. The closer f is to $f_{\max}$, the smaller p_m should be, i.e., p_m should vary directly as $f_{\max} - f$. Similarly, p_c should vary directly as $f_{\max} - f'$, where f' is the larger of the fitness values of the solutions to be crossed. The expressions for p_c and p_m now take the forms

$$p_c = k_1(f_{\max} - f')/(f_{\max} - \overline{f}), \quad k_1 \le 1.0 \tag{1}$$

and

$$p_m = k_2(f_{\max} - f)/(f_{\max} - \overline{f}), \quad k_2 \le 1.0. \tag{2}$$

(k_1 and k_2 have to be less than 1.0 to constrain p_c and p_m to the range 0.0–1.0).

Note that p_c and p_m are zero for the solution with the maximum fitness. Also $p_c = k_1$ for a solution with $f' = \overline{f}$, and $p_m = k_2$ for a solution with $f = \overline{f}$. For solutions with subaverage fitness values i.e., $f < \overline{f}, p_c$ and p_m might assume values larger than 1.0 . To prevent the overshooting of p_c and p_m beyond 1.0, we also have the following constraints,

$$p_c = k_3, \quad f' \leq \overline{f} \tag{3}$$

and

$$p_m = k_4, \quad f \leq \overline{f} \tag{4}$$

where $k_3, k_4 \leq 1.0$.

C. Practical Considerations and Choice of Values for k_1, k_2, k_3 and k_4

In the previous section, we saw that for a solution with the maximum fitness value, p_c and p_m are both zero. The best solution in a population is transferred undisrupted into the next generation. Together with the selection mechanism, this may lead to an exponential growth of the solution in the population and may cause premature convergence. To overcome the above stated problem, we introduce a default mutation rate (of 0.005) for every solution in the AGA.

We now discuss the choice of values for k_1, k_2, k_3, and k_4. For convenience, the expressions for p_c and p_m are given as

$$p_c = k_1(f_{\max} - f')/(f_{\max} - \overline{f}), \quad f' \geq \overline{f}, \tag{5}$$

$$p_c = k_3, \quad f' < \overline{f} \tag{6}$$

and

$$p_m = k_2(f_{\max} - f)/(f_{\max} - \overline{f}), \quad f \geq \overline{f}, \tag{7}$$

$$p_m = k_4, \quad f < \overline{f} \tag{8}$$

where $k_1, k_2, k_3, k_4 \leq 1.0$.

It has been well established in GA literature [6] [10] that moderately large values of p_c $(0.5 < p_c < 1.0)$, and small values of p_m $(0.001 < p_m < 0.05)$ are essential for the successful working of GAs. The moderately large values of p_c promote the extensive recombination of schemata, while small values of p_m are necessary to prevent the disruption of the solutions. These guidelines, however, are useful and relevant when the values of p_c and p_m do not vary.

One of the goals of our approach is to prevent the GA from getting stuck at a local optimum. To achieve this goal, we employ solutions with subaverage fitnesses to search the search space for the region containing the global optimum. Such solutions need to be completely disrupted, and for this purpose we use a value of 0.5 for k_4. Since solutions with a fitness value of $\overline{f}$ should also be disrupted completely, we assign a value of 0.5 to k_2 as well.

Based on similar reasoning, we assign k_1 and k_3 a value of 1.0. This ensures that all solutions with a fitness value less than or equal to $\overline{f}$ compulsarily undergo crossover. The probability of crossover decreases as the fitness value (maximum of the fitness values of the parent solutions) tends to $f_{\max}$ and is 0.0 for solutions with a fitness value equal to $f_{\max}$.

In the next section, we compare the AGA with previous approaches for employing adaptive operators in GAs.

IV. Comparison of AGA With Other Adaptive Strategies

The idea of adapting crossover and mutation operators to improve the performance of GAs has been employed earlier [1], [3], [9], [24]. This section reviews these techniques and compares them with our approach.

Schaffer *et al.* [1] discuss a crossover mechanism wherein the distribution of crossover points is adapted based on the performance of the generated offspring. The distribution information is encoded into each string using additional bits. Selection and recombination of the distribution bits occurs in the normal fashion along with the other bits of the solutions.

Davis [3], [4] discusses an effective method of adapting operator probabilities based on the performance of the operators. The adaptation mechanism provides for the alteration of operator probabilities in proportion to the fitnesses of strings created by the operators. Simply stated, operators which create and cause the generation of better strings are alloted higher probabilities. The technique has been developed in the context of a steady-state GA (see [24]), and experimental evidence has demonstrated considerable promise.

Fogarty [9] has studied the effects of varying the mutation rate over generations and integer encodings. Specifically, a mutation rate that decreases exponentially with generations has demonstrated superior performance for a single application.

In an approach employing a form of adaptive mutation, Whitley *et al.* [24] have reported significant performance improvements. The probability of mutation is a dynamically varying parameter determined from the Hamming distance between the parent solutions. The diversity in the population is sustained by subjecting similar solutions to increased levels of mutation.

The adaptation policy in AGA is different from all the approaches described above; [1] is not related to adapting mutation and crossover rates. AGA is different from [3] and [9] as, in the AGA, p_c and p_m are determined for each individual as a function of its fitness. In [9], p_m is varied in a predetermined fashion. In [3] too, the operator probabilities are invariant with the individual fitnesses of solutions, although they are modified periodically based on the average performance of the operators (determined indirectly from the fitnesses of solutions).

The AGA bears closer resemblance to Whitley's adaptive mutation approach [24]. In both cases, the mutation rate is determined specifically for each solution. Both techniques are also derived from the idea of sustaining the diversity in the population without affecting the convergence properties. In Whitley's approach, however, the adaptive mutation technique has been employed in the context of a steady state GA, while we are concerned with generational replacement, in the AGA. Since the steady state GA employs a form of populationary elitism, there is no need to 'protect' the best solutions from the high levels of disruption. In the AGA, the best solutions are explicitly protected from disruption. The criterion for adaptation is also different in both cases: in [24] p_m is varied based on the Hamming distance between solutions, while in our approach p_m and p_c are adapted based on fitness values.

The experimental results in [24] and our own experiments (Section V) demonstrate the efficacy of this line of approach.

V. The Schema Theorem and the 'AGA'

The Schema theorem [7], [10], [17], has been the predominant method for analyzing GAs. Schemata are building blocks that form the solutions, and the Schema theorem predicts the growth of high fitness building blocks at the expense of low fitness ones. The Schema theorem also models the detrimental effects of crossover and mutation on the propagation of schemata from generation to generation. In this section, we derive the Schema theorem for the GA with adaptive p_c and p_m. The notation that we have used in the derivation is as follows. [1] We now derive the expression to predict $N_h(t+1)$

h	: a schema
f_i	: the fitness value of an instance (solution) of schema
$\overline{f}$	: the average fitness value of the population
$\overline{f_h}$	: the average fitness value of schema h
$f_{\max}$	: the maximum fitness value of the population
$\overline{f_h^2}$	: the average of the square of fitness values (second moment of fitness values) for the schema h
$n_i^h(t+1)$	: the expected number of offspring created in generation $t+1$ due to a solution i of schema h (and of the generation t)
$N_h(t)$	: the number of solutions of generation t which are instances of the schema h
$l(h)$	: the *defining length* of the schema h
L	: the length of the solution, i.e., the number of binary bits in the encoded solution.

from $N_h(t)$. The selection criterion that we have used for the GA is that of proportional selection. We first consider the effect of crossover and then generalize the results for mutation.

The expected number of offspring generated by a solution i of the schema h is given by

$$n_i^h(t+1) \geq \frac{f_i}{\overline{f}}\left(1 - \frac{l(h)}{L-1}p_c\right). \quad (9)$$

The expression that we have used for p_c is given by

$$p_c = k_1\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}, \quad f_i \geq \overline{f},$$
$$p_c = k_3, \quad f_i < \overline{f}$$

where $k_1, k_3 \leq 1.0$ and $k_1 \leq k_3$.

After substituting for p_c in (9), we get

$$n_i^h(t+1) \geq \frac{f_i}{\overline{f}}\left(1 - \frac{l(h)}{L-1}k_1\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right), \quad f_i \geq \overline{f} \quad (10)$$

and,

$$n_i^h(t+1) \geq \frac{f_i}{\overline{f}}\left(1 - \frac{l(h)}{L-1}k_3\right), \quad f_i < \overline{f}. \quad (11)$$

[1] In our research, we have used a binary alphabet for encoding the solutions.

To transform the two inequalities of (10) and (11) into one inequality, we recall the assumption made in the previous section that $k_1 = k_3$.

Now, we get a single inequality without any constraints on f_i,

$$n_i^h(t+1) \geq \frac{f_i}{\overline{f}}\left(1 - \frac{l(h)}{L-1}k_1\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right). \quad (12)$$

To get an estimate for $N_h(t+1)$, we consider the summation of n_i^h over all the solutions, i, that are instances of the schema h, i.e.,

$$N_h(t+1) = \sum_{i=1}^{N_h(t)} n_i^h(t+1).$$

Equivalently, from (12), we get,

$$N_h(t+1) \geq \sum_{i=1}^{N_h(t)} \frac{f_i}{\overline{f}}\left(1 - \frac{l(h)}{L-1}k_1\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right). \quad (13)$$

Since, $\left(\sum_{i=1}^{N_h(t)} f_i\right) = \left(N_h(t) \times \overline{f_h}\right)$, and $\left(\sum_{i=1}^{N_h(t)} {f_i}^2\right) = \left(N_h(t) \times \overline{f_h^2}\right)$, (13) gets modified to

$$N_h(t+1) \geq N_h(t)\frac{\overline{f_h}}{\overline{f}}\left(1 - k_1\frac{l(h)}{L-1}\frac{f_{\max}}{(f_{\max} - \overline{f})}\right) + N_h(t)\left(\left(k_1\frac{l(h)}{L-1}\right)\frac{\overline{f_h^2}}{\overline{f} \times (f_{\max} - \overline{f})}\right) \quad (14)$$

After rearranging the terms, (15) can be rewritten as

$$N_h(t+1) \geq N_h(t)\frac{\overline{f_h}}{\overline{f}} - N_h(t)k_1\frac{l(h)}{L-1}\frac{1}{\overline{f} \times (f_{\max} - \overline{f})} \times \left(f_{\max}\overline{f_h} - \overline{f_h^2}\right). \quad (15)$$

(15) represents the schema theorem when adaptive crossover is used in the GA. We now consider some special cases of (15) based on the value of $\overline{f_h}$.

A. Effect of Mutation

When we include the disruptive effects of mutation, (12) may be generalized to the form

$$n_i^h(t+1) \geq \frac{f_i}{\overline{f}}\left(1 - \frac{l(h)}{L-1}k_1\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right) \times \left(1 - k_2\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right)^n. \quad (16)$$

where $\left(1 - k_2\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right)^n$ gives the probability that the solution i survives disruption due to mutation. For $k_2 \ll 1$, the right side of (16) may be approximated to $\frac{f_i}{\overline{f}}\left(1 - \frac{l(h)}{L-1}k_1\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})} - nk_2\frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right)$. It follows that, when $k_2 \ll 1$, the Schema theorem for the AGA may be generalized to

$$N_h(t+1) \geq N_h(t)\frac{\overline{f_h}}{\overline{f}} - N_h(t)K\frac{1}{\overline{f} \times (f_{\max} - \overline{f})} \times \left(f_{\max}\overline{f_h} - \overline{f_h^2}\right) \quad (17)$$

where $K = k_1\frac{l(h)}{L-1} + nk_2$.

When k_2 is comparable with 1.0, a complete Taylor series expansion of the right side of (16) needs to considered. In what follows, we derive the generalized schema theorem for the AGA for all values of k_1 and k_2. The following analysis however generates a slightly inferior lower bound for $N_h(t+1)$ than that obtained in (16) and (15).

We first introduce the notion of the nth fitness moment of a schema h in generation t, defined as

$$M_h^n(t) = \frac{1}{N_h(t)} \sum_{i \in h} (f_i)^n. \tag{18}$$

It may be noted that $M_h^1(t)$ is the average fitness of h and that $M_h^2(t) - (M_h^1(t))^2$ gives the variance of fitness of h.

We also state the following two lemmas without outlining the proofs.

Lemma 1: $M_h^k(t) \geq M_h^l(t) M_h^m(t)$ where $k = l + m$.

Lemma 2: $M_h^p(t) M_h^q(t) \geq M_h^r(t) M_h^s(t)$ where $p + q = r + s$ and $|p - q| > |r - s|$.

Let us recall (16). We may express it in a general form

$$n_i^h(t+1) \geq \sum_{k=1}^{n+2} \alpha_k (f_i)^k. \tag{19}$$

To obtain $N_h(t+1)$, consider the summation of $n_i^h(t+1)$ over all solutions i that are members of h,

$$N_h(t+1) \geq \sum_{i \in h} \sum_{k=1}^{n+2} \alpha_k (f_i)^k. \tag{20}$$

After interchanging the order of summations, (20) may be expressed in terms of the fitness moments of h as

$$N_h(t+1) \geq N_h(t) \sum_{k=1}^{n+2} \alpha_k M_h^k(t). \tag{21}$$

From Lemma 1 it follows that $M_h^k(t) \geq (M_h^1(t))^k$. Consequently (21) may be further reduced to the form

$$N_h(t+1) \geq N_h(t) \sum_{k=1}^{n+2} \alpha_k (M_h^1(t))^k \tag{22}$$

From the structural similarity of (22) and (19), and after considering (16), we gather that (22) may be expressed as

$$N_h(t+1) \geq N_h(t) \frac{\overline{f_h}}{\overline{f}} \left(1 - \frac{l(h)}{L-1} k_1 \frac{(f_{\max} - \overline{f_h})}{(f_{\max} - \overline{f})}\right) \times \left(1 - k_2 \frac{(f_{\max} - \overline{f_h})}{(f_{\max} - \overline{f})}\right)^n. \tag{23}$$

(15) represents the Generalized Schema theorem for the AGA.

A comparison of (15) and (23) ($k_2 = 0$) demonstrates that the former provides a tighter bound than the latter. This is because we have employed the inequality $M_h^k(t) \geq (M_h^1(t))^k$ to obtain a closed form expression for the Generalized Schema theorem.

For $\overline{f_h} = f_{\max}$, we get

$$N_h(t+1) \geq N_h(t) \frac{\overline{f_h}}{\overline{f}}, \tag{24}$$

and for $\overline{f_h} = \overline{f}$, it follows that

$$N_h(t+1) \geq N_h(t) \frac{\overline{f_h}}{\overline{f}} \left(1 - \frac{l(h)}{L-1} k_1\right)(1 - k_2)^n. \tag{25}$$

(24) and (25) are instances of the generalized schema theorem for schemata with fitnesses $f_{\max}$ and $\overline{f}$, and elucidate the adaptation policy.

B. Variation of Schema Fitness

The Schema theorem provides a bound for the growth rate of a schema, but it does not provide any insight into the effect of the genetic operators on the growth rates of instances of a schema. Specifically it would be interesting to characterize the variation of fitness of a schema from one generation to another, which is caused by the different growth rates of the instances of the schema. In this section we compare the variation of the average fitness of a schema under the action of the AGA and the SGA and demonstrate that the AGA tends to induce higher schema fitnesses than the SGA.

In evaluating the expected average fitness of schema h in generation $t + 1$, from the fitnesses of solutions in generation t, we need to focus attention on two components: instances of the schema in generation t which are expected to remain undisrupted, and solutions that are generated due to recombination.

By definition, the average fitness of schema h in generation $t + 1$ may be expressed as

$$M_h^1(t+1)_{\text{AGA}} = \frac{\sum_{i \in h} n'^h_i(t+1) f_i + \sum_{j \in h} n_j f_j}{\sum_{i \in h} n'^h_i(t+1) + \sum_{j \in h} n_j} \tag{26}$$

where $n'^h_i(t+1) = \frac{f_i}{\overline{f}}\left(1 - \frac{l(h)}{L-1} k_1 \frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right)\left(1 - k_2 \frac{(f_{\max} - f_i)}{(f_{\max} - \overline{f})}\right)^n$ and n_j gives the number of instances of j that are generated due to recombination.

Equation (26) may be simplified to the form

$$M_h^1(t+1)_{\text{AGA}} = \frac{N_h(t) \sum_{k=1}^{n+2} \alpha_k M_h^{k+1}(t) + \sum_{j \in h} n_j f_j}{N_h(t) \sum_{k=1}^{n+2} \alpha_k M_h^k(t) + \sum_{j \in h} n_j}. \tag{27}$$

The notation in the above equation is the same as used in the previous section. The corresponding expression for the average fitness in the case of the SGA is given by

$$M_h^1(t+1)_{\text{SGA}} = \frac{N_h(t) \beta M_h^2(t) + \sum_{j \in h} n_j f_j}{N_h(t) \beta M_h^1(t) + \sum_{j \in h} n_j} \tag{28}$$

where $\beta = (1 - p_c)(1 - p_m)^n$.

For purposes of comparison, let us assume that $\sum_{j \in h} n_j f_j$ as well as $\sum_{j \in h} n_j$ are identical in both cases (AGA and SGA). This assumption becomes necessary to be able to focus attention on the disruptive effects of the operators. The complexity of GA dynamics makes it practically impossible to exactly model the effects of recombination in an elegant fashion.

From Lemma 2, it follows that, for each k in (27), $\frac{M_h^{k+1}(t)}{M_h^k(t)} \geq \frac{M_h^2(t)}{M_h^1(t)}$. Consequently, it may easily be shown from (27) and (28) that

$$M_h^1(t+1)_{\mathrm{AGA}} \geq M_h^1(t+1)_{\mathrm{SGA}}. \quad (29)$$

In a similar fashion, we can extend the result to moments of higher orders

$$M_h^i(t+1)_{\mathrm{AGA}} \geq M_h^i(t+1)_{\mathrm{SGA}} \quad (30)$$

(29) and (30) may be explained intuitively from the adaptive mechanism. Since the probability of disruption of a solution with a high fitness value is smaller than that for a solution with a lower fitness value, the expected fitness of a randomly selected solution after disruption is higher than in the case of SGA wherein all solutions are disrupted with the same probability. Thus we observe that the AGA promotes schemata with high fitness values, and also causes the fitnesses of schemata to increase rapidly. Both the effects are the outcomes of the adaptation policy.

VI. Experiments and Results

In this section, we discuss the experiments that we have conducted to compare the performance of the AGA and SGA. For this purpose we have employed several multimodal test problems with varying complexities. The rest of this section is devoted to a discussion of the performance criteria for the GAs, the functions that are to be optimized, experiments, and the comparative results.

A. Performance Measures

As a measure of performance, we consider the average number of generations that the GA requires to generate a solution with a certain high fitness value (called the *threshold*). The average number of generations is obtained by performing the experiment repeatedly (in our case, 30 times) with different and randomly chosen initial populations.

Since the goal of our approach is to prevent the convergence of the GA to a local optimum, we also evaluate the performance of the GA in terms of the number of runs for which the GA gets stuck at a local optimum. When the GA fails to reach the global optimum after a sufficiently large number of generations, we conclude that it has gotten stuck at a local optimum.

B. Functions for Optimization

The choice of suitable functions to verify the performance of GAs is not an easy task. The nature of the optimization function varies a lot from application to application, in terms of the number of local optima, the rate of variation of the objective function, etc. In this research, we have used several multimodal functions with varying complexities. They are the following:

DeJong's f5: This is a spiky function (also known as Shekel's foxholes) with 25 sharp spikes of varying heights. The function has two variables and the solution is encoded using 34 bits. The task of the GA is to locate the highest peak. The expression for $f5$ is as

$$f5 = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}.$$

f6: This is a rapidly varying multimodal function of two variables, and is symmetric about the origin with the height of the barrier between adjacent minima increasing as the global optimum is approached. The variables are encoded using 22 bits each, and assume values in the range $(-100.0, 100.0)$. f6 has been employed earlier [22] for comparative studies, where it is referred to as the 'Sine envelope sine wave function.' The expression for $f6$ is

$$f6 = 0.5 + \frac{\sin^2\sqrt{{x_1}^2 + {x_2}^2} - 0.5}{[1.0 + 0.001({x_1}^2 + {x_2}^2)]^2}$$

f7: [2] This function is also similar to $f6$, but has the barrier height between adjacent minima approaching zero as the global optimum is approached

$$f7 = ({x_1}^2 + {x_2}^2)^{0.25}\left[\sin^2(50({x_1}^2 + {x_2}^2)^{0.1}) + 1.0\right].$$

Order-3 Deceptive: GA-deceptive functions are being used extensively to evaluate the performance of GAs. The order-3 deceptive function depends on three binary bits as shown in Table I. Optimizing the three-bit deceptive function is a trivial exercise. The actual function to be optimized by the GAs is the sum of five such independent functions. The solution string is obtained by concatenating five of the three-bit codes. We used only five subfunctions in our function, mainly to enable the GAs to converge to the optimum with small populations (population size = 100). For a further description of deceptive functions the reader is referred to [12].

TSP: The traveling salesman problem (TSP) involves finding the shortest Hamiltonian cycle in a complete graph of n nodes. The Euclidean distance between any two nodes is computed from their coordinates. An instance of the TSP is specified by n, the number of cities, and the coordinates of the n cities (nodes). In our implementations, we have employed the order crossover operator [21], [10], and a mutation operator that swaps the positions of two randomly chosen cities. p_c and p_m determine the probability with which the operators

TABLE I
A Three-Bit Deceptive Function

Binary Code	Function Value
000	28
001	26
010	22
011	0
100	14
101	0
110	0
111	30

[2] The decoded value of each variable of the functions $f5, f6$, and $f7$ has been shifted by 10% to the left, and wrapped around the upper limit in case the value is less than the lower limit. This has been done to shift the optimal solutions away from Hamming cliffs (see [5]).

are employed. We have chosen the 30-city and 105-city problems (see [24] for coordinates of cities) for comparing the performance of the SGA and the AGA.

Neural Networks: The underlying optimization problem in feedforward neural networks is that of identifying a set of interconnection weights, such that a mean square error defined between a set of output patterns and training patterns is minimized. Each neuron i may be associated with

- An output value O_i,
- A set of k input values $I_{ji} \quad 1 \le j \le k$,
- A threshold value T_i,
- A set of interconnection weights $w_{ji} \quad 1 \le j \le k$
- An *activation value* $A_i = \sum_{j=1}^{k} w_{ji} I_{ji} - T_i$,

The output value of each neuron is typically a nonlinear function of the activation value. In a feed-forward network, the neurons are organized into layers (input, output and hidden), with the inputs of each neuron connected to the outputs of the neurons of the previous layer. The input patterns are applied to the input layer, and the training pattern is compared with the outputs of neurons in the output layer. The mean square error for a given set of weights is evaluated as

$$\text{MSE} = \frac{1}{pN_o} \sum_{i=1}^{p} \sum_{j=1}^{N_o} (O_{ij} - O'_{ij})^2 \qquad (31)$$

where

MSE:	the mean square error
p :	number of input patterns
N_o :	number of output neurons
O_{ij} :	output value of the jth neuron for the ith input pattern
O'_{ij} :	training value of jth neuron for ther ith input pattern.

In our implementation, the output function f is sigmoidal: $f_i = (1 + \exp^{-10A_i})^{-1}$. We also use binary inputs and train the network to generate binary outputs. Further, $T_{ij} = 0.1$ for a binary 0 and $T_{ij} = 0.9$ for a binary 1. w_{ij} and T_j assume values in the range -1.0 to $+1.0$. Each weight is encoded using 8 bits, and the string is formed by concatenating the binary codes for all the weights and threshold values.

We consider three mapping problems,

- *XOR*: 2 inputs, 1 output, 5 neurons, 9 weights, 4 input patterns, the output value is the Exclusive OR of the input bits.
- *4-bit parity*: 4 inputs, 1 output, 9 neurons, 25 weights, 16 input patterns, the output value is 1 if there are an odd number of 1s among the inputs.
- *Decoder Encoder*: 10 inputs, 10 outputs, 25 neurons, 115 weights, 10 input patterns (each having all 0s and a 1 at one of the ten inputs), output pattern is the same as the input pattern.

Test Generation Problem: The primary task of test generation for digital logic circuits is to generate input vectors of logical 0's and 1's that can check for possible faults in the circuit by producing observable faulty response at the primary outputs of the circuit. The problem of generating a test for a given fault has been proved to be NP-complete [18]. In generating tests, it is desirable to detect close to 100% of all the possible faults in the circuit. Test generation as a candidate optimization problem for GAs may be characterized as follows:

- Faults are modelled as being stuck-at-0 or stuck-at-1.
- A test for a fault should (i) generate a logic value at the fault site that is different from the stuck-at value of the fault, (ii) should be able to propogate the fault effect to one of the primary outputs.
- Fault simulation approach to test generation: Input vectors are generated randomly, and then through logic simulation, the faults that the vector detects are identified as being detected.
- Random test generation may be improved by using a search based on a cost associated with each input vector.
- Distance Cost function: $C_v = \sum_{i \in F} L_m - L_{vi}$ where
  C_v: cost associated with a vector v,
  F: set of undetected faults
  L_m: maximum number of gate levels in the circuit
  L_{vi}: level to which the fault effect of i has been propogated by vector v.

The cost C_v is minimum (locally) when a given input vector is a test for a certain fault. It should be noted that the cost function changes as faults are detected and removed from the list of undetected faults. The task for the GA is to minimize the cost C_v. Test circuits for experiments have been chosen from the ISCAS-85 benchmarks [19].

C. Experimental Results

Except for the TSP's, in all our experiments, we have used a population size of 100 for the GAs. 'Scaling' of fitness values, and the Stochastic remainder technique (see [10]) for 'selection' have been used in the GAs. All parameters have been encoded using a fixed point encoding scheme.

For the SGA, we have used values of $p_c = 0.65$ and $p_m = 0.008$.

For the AGA, p_c and p_m are determined according to expressions (5), (6), (7), and (8) given in Section III-C.

The experimental results are presented in Tables II–IV. Table II gives the average number of generations required by each GA for attaining a solution with a fitness value equal to the threshold value 'thresh.'[3] Also tabulated is the number

TABLE II
COMPARISON OF PERFORMANCE OF AGA AND SGA

Function	Str. Len.	Gens.		Stuck		Thresh.	Max. Gens.
		SGA	AGA	SGA	AGA		
XOR	72	61.2	36.73	10	0	0.999	100
4-bit parity	200	399.33	93.43	18	0	0.999	500
Dec. Enc.	920	456.43	71.70	26	0	0.99	500
f5	34	64.06	36.63	7	0	1.00	100
f6	44	173.9	106.56	23	6	0.999	200
f7	44	419.90	220.61	21	5	0.995	500
Order-3 Dec.	15	70.32	105.33	8	9	1.00	200

[3] We are not measuring population convergence based on the mean convergence of each bit since the AGA never converges in the above sense due to the high disruption rates of low fitness solutions.

TABLE III
PERFORMANCE OF AGA AND SGA FOR TSP

Cities	Avg. Tour Length		Optimum Tour Located		Max. Gens.	Pop. Size
	SGA	AGA	SGA	AGA		
30 (424.0)	442.1	430.2	0	7	100	1000
105 (14383)	16344.3	14801.4	0	4	500	2000

TABLE IV
PERFORMANCE OF AGA AND SGA FOR THE TEST GENERATION PROBLEM

Circuit	SGA (Gens.)	AGA (Gens.)	Fault Coverage	Str. Len.
c432	102.10	10.73	99.23%	36
c499	10.91	10.50	98.94%	41
c880	155.23	37.33	100.00%	60
c1355	35.26	31.70	99.49%	41
c1908	122.13	57.93	99.52%	33
c3540	155.43	73.66	96.00%	50
c5315	53.33	21.56	98.89%	178

of instances (out of 30 trials) for which the GAs have gotten stuck at a local optimum. The maximum number of generations that the GAs were executed for, and the string length are also indicated for each of the problems.

The AGA outperforms the SGA for all the problems except the Order-3 Deceptive. For the three neural network problems, AGA has located the optimal solution in every trial, while the performance of the SGA has been poor.

For the TSP's we have used populations of 1000 and 2000 for the 30-city and 105-city problems respectively. The number of function evaluations have been 100,000 and 1,000,000 respectively. Results have been obtained for 10 different trials. For both the problems, the SGA was not able to locate the optimal tour even on one occasion, while AGAs performance has been significantly better, both in terms of the average tour length and the number of instances when the optimal tour was located.

Table IV compares the performance of the AGA and the SGA for the Test Generation problem. The numeral in the circuit name indicates the gate count of the circuit. Once again, the superior performance of AGA is clear. For c432, the SGA requires almost 10 times the number of generations that the AGA needs to locate all detectable faults. Only for c499, the SGA has come close to performing as well as the AGA. The results are averages over 30 different trials for each circuit. It may be noted that the complexity of test generation is not directly dependent on the circuit size, but is controlled by several other factors such as the fanins and fanouts of gates, the number of levels in the circuit, etc.

D. When Does The AGA Perform Well?

The optimization problems considered above span a range of complexities, string lengths, and problem domains. In general, the performance of the AGA has been significantly superior to that of the SGA, while in specific instances such as Order-3 Deceptive problem and for c499 in the Test generation problem, the SGA has performed as well as the AGA or better. The experimental results also point out that the relative performance of the AGA as compared to that of the SGA varies considerably from problem to problem. All the problems that we have considered have some epistaticity present, however the extent of epistaticity varies considerably. For instance, in the neural network problems, the high epistaticity is brought about by the fitness being a complex nonlinear function of the weights, while in the order-3 Deceptive problem, the epistaticity is relatively lower with the fitness contribution due to a bit being affected only by two other bits.

TABLE V
EFFECT OF "MULTIMODALITY" ON THE PERFORMANCE OF THE AGA AND SGA

k	Gens.		Stuck	
	SGA	AGA	SGA	AGA
1	24.10	27.73	0	0
3	75.56	70.91	4	0
5	78.96	71.93	12	3
7	82.76	72.70	13	4

A different aspect of multimodal function optimization is the sensitivity of the optimization technique to the 'multimodality' of the problem, i.e., how the performance varies as the number of local optima in the search space vary. It may be observed from Table II that the relative performance of the AGA with respect to that of the SGA is better for $f6$ and $f7$ than for $f5$. Although the evidence is not conclusive, it appears that the AGA performs relatively better than the SGA when the number of local optima in the search space is large.

To better understand the circumstances under which AGA performs better than the SGA, we have conducted two sets of experiments where we have methodically varied the epistacity in the problem and the number of local optima in the search space. For purposes of convenience, we have chosen the following objective function for these experiments

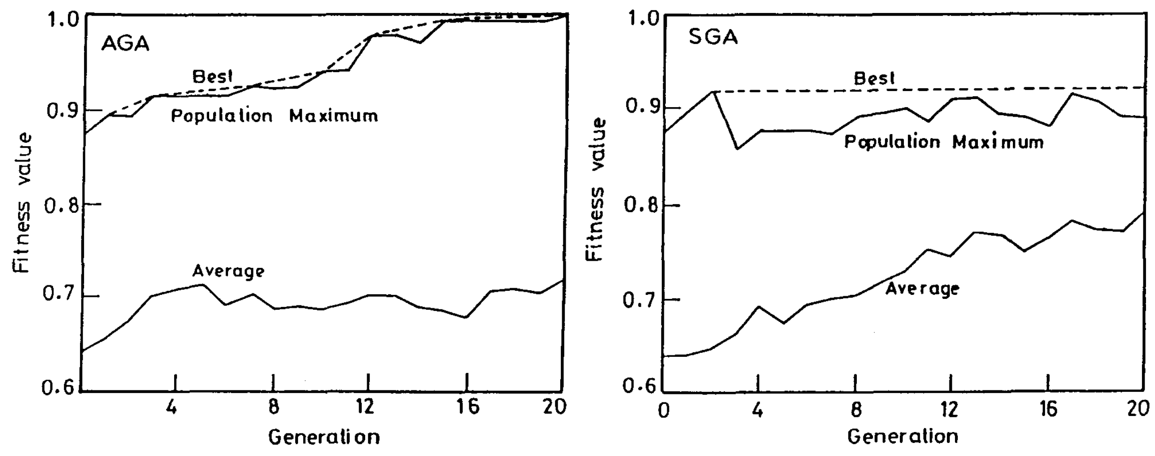
$$f8 = \sum_{i=1}^{P} \left| \frac{\sin(\pi k x_i)}{(\pi k x_i)} \right| \qquad -0.5 \leq x_i \leq 0.5$$

where P gives the number of variables.

The function has one global optimum and P^k local optima for odd values of k.

To characterize the effect of varying the number of local optima in the search space, we have varied k for a fixed value of $P = 5$. Each variable x_i is encoded using 10 bits. The experimental results are presented in Table V. Table V confirms our earlier observation that, with increasing number of local optima the performance of AGA improves steadily over that of the SGA. For $k = 1$, the function is unimodal, and the SGA outperforms the AGA.

Next we consider the effects of varying the epistaticity of the function. We consider strings of length 40, $k = 5$, and we vary the number of parameters P. Correspondingly the number of bits required for encoding each variable also changes. The epistacity increases as P decreases, since the fitness due to a single variable x_i depends on the interactions of $40/P$ bits. From Table VI, it is clear that the relative performance of the



(a) (b)

Fig. 3. Comparison of the Best, Average, and Population-Maximum fitness values for the AGA and SGA.

TABLE VI
EFFECT OF EPISTACITY ON THE PERFORMANCE OF THE AGA AND SGA

Code Len.	Variables	Gens.		Stuck	
		SGA	AGA	SGA	AGA
20	2	92.50	86.13	11	7
10	4	71.73	64.36	5	4
8	5	59.33	52.40	3	3
5	8	35.10	48.86	1	2
4	10	27.73	42.66	0	0

TABLE VII
EFFECT OF $k1$ ON AGA PERFORMANCE: AVERAGE NUMBER OF GENERATIONS FOR CONVERGENCE AND NUMBER OF INSTANCES WHEN AGA GETS STUCK AT A LOCAL OPTIMUM

Function	$k1$=0.2	0.4	0.6	0.8	1.0
f5	47.76 (6)	40.20 (3)	33.46 (3)	41.33 (1)	36.63 (0)
XOR	45.70 (2)	37.36 (0)	42.30 (0)	38.56 (0)	36.73 (0)
Order-3 Dec.	143.13 (11)	125.66 (9)	109.7 (10)	121.66 (13)	105.33 (9)

AGA with respect to the SGA deteriorates as the epistaticity decreases.

E. Sensitivity of AGA to k_1 and k_2

We have already pointed out in Section II that p_c and p_m critically control the performance of the GA. One of the goals of having adaptive mutation and crossover is to ease the user's burden of specifying p_m and p_c. However, our method has introduced new parameters k_1 and k_2 for controlling the adaptive nature of p_m and p_c. To evaluate the effect of k_1 and k_2 on the performance of the AGA, we have monitored the performance of the AGA for varying values of k_1 (0.2–1.0) and k_2 (0.1–0.5). The experimental results are presented in Tables VII and VIII.

On analyzing the results presented in Table VIII for different values of k_2, we notice no dramatic difference in the

TABLE VIII
EFFECT OF $k2$ ON AGA PERFORMANCE: AVERAGE NUMBER OF GENERATIONS FOR CONVERGENCE AND NUMBER OF INSTANCES WHEN AGA GETS STUCK AT A LOCAL OPTIMUM

Function	$k2$=0.1	0.2	0.3	0.4	0.5
f5	60.20 (2)	48.26 (1)	65.33 (1)	44.33 (2)	36.63 (0)
XOR	52.26 (0)	51.66 (0)	49.73 (0)	44.60 (0)	36.73 (0)
Order-3 Dec.	122.30 (15)	117.43 (12)	98.66 (8)	96.53 (6)	105.33 (9)

performance of the AGA in terms of the average number of generations required for convergence. The fact that the performance of the GA hardly varies with the value of k_2 shows that the AGA is not sensitive to the external parameter k_2—one of the goals of our research. The AGA gets stuck at local optima fewer times for higher values of k_2 than for lower values of k_2. The results justify our choice of $k_2 = 0.5$ for the AGA.

Table VII demonstrates the steady improvement in performance of AGA as k_1 is increased. This may be expected since a large value of k_1 maximizes the recombination of schemata, while the best schemata are yet retained due to the adaptation policy.

To illustrate how the AGA works in a fashion different from that of the SGA, we plot in Fig. 3, the variation of the average, best, and population-maximum fitness values for the SGA and the AGA. The XOR function is being optimized, and the population size is 100. On comparing the two plots, we observe that the average fitness of the population increases gradually for the AGA (approximately 0.025 per generation) while it increases rapidly for the SGA (0.075 per generation). A careful observation of Fig. 3 reveals that, in the first 5 generations, the average fitness for the AGA increases rapidly (0.12 per generation), remains rather flat until about the 15th generation, and once again increases quickly (0.1 per generation). The relatively flat zone (generations 5 to 15) occurs when the AGA has not yet located the global optimum, and has only located a locally optimal solution with a fitness of 0.92. Another feature that stands out is the fluctuation of the population-maximum fitness for the SGA, indicating that the SGA loses the best solutions often. For the AGA, since the best solution in each population is being propagated to the subsequent generation with minimal disruption, the population-maximum fitness is increasing most of the time. The lower average fitness value of the AGA indicates that the population has remained scattered in the search space and has not gotten stuck at any local optimum. We have not only achieved a better convergence rate to the global optimum, but have also prevented the AGA from getting stuck at the local optimum (with a fitness value of around 0.92) that the SGA has succumbed to.

VII. Conclusion

Recent research on GAs has witnessed the emergence of new trends that break the traditional mold of 'neat' GAs that are characterized by static crossover and mutation rates, fixed length encodings of solutions, and populations of fixed size. Goldberg has introduced the notions of variable-length solutions for GAs in [13] and [14], and has shown that the 'Messy GAs' perform very well. Davis [4], [3] has recommended the technique of adapting operator probabilities dynamically based on the relative performance of the operators.

In this paper, we adopt a 'messy' approach to determine p_c and p_m, the probabilities of crossover and mutation. The approach is different from the previous techniques for adapting operator probabilities as p_c and p_m are not predefined, they are determined adaptively for each solution of the population. The values of p_c and p_m range from 0.0 to 1.0 and 0.0 to 0.5 respectively. It might appear that the low values of p_c and the high values of p_m might either lead to premature convergence of the GA or transform the GA into a random search. However, it is the manner in which p_c and p_m are adapted to the fitness values of the solutions, that not only improves the convergence rate of the GA, but also prevents the GA from getting stuck at a local optimum. In the adaptive GA, low values of p_c and p_m are assigned to high fitness solutions, while low fitness solutions have very high values of p_c and p_m. The best solution of every population is 'protected', i.e., it is not subjected to crossover, and receives only a minimal amount of mutation. On the other hand, all solutions with a fitness value less than the average fitness value of the population have $p_m = 0.5$. This means that all subaverage solutions are completely disrupted and totally new solutions are created. The GA can, thus, rarely get stuck at a local optimum.

We have conducted extensive experiments on a wide range of problems including TSP's, neural network weight-optimization problems, and generation of test vectors for VLSI circuits. In most cases, the AGA has outperformed the SGA significantly. Specifically, we have observed that, for problems that are highly epistatic and multimodal, the AGA performs very well.

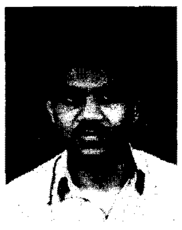
In this work, we have chosen one particular way of adapting p_c and p_m based on the various fitnesses of the population. The results are encouraging, and future work should be directed at developing other such adaptive models for the probabilities of crossover and mutation. A similar dynamic model for varying the population size in relation to the fitnesses of the population is certainly worth investigating. We hope that this paper, along with Goldberg's 'Messy Genetic Algorithms', lays the foundations for a new class of adaptive, self organizing GAs.

Acknowledgment

The authors thank all the three reviewers for their critical evaluation of the paper, which has considerably helped in improving the overall quality of the work presented here.

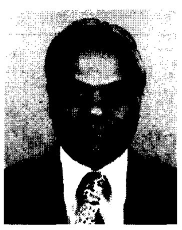
References

[1] J. D. Schaffer and A. Morishma, "An adaptive crossover mechanism for genetic algorithms," in *Proc. Second Int. Conf. Genetic Algorithms,* 1987, pp. 36–40.
[2] L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing,* London: Pitman, 1987.
[3] L. Davis, "Adapting operator probabilities in genetic algorithms," *Proc. Third Int. Genetic Algorithms,* 1989, pp. 61–69.
[4] ——— (Ed), "Handbook of Genetic Algorithms," Van Nostrand Reinhold, 1991.
[5] ———, "Bit climbing, representational bias, and test suite design' in *Proc. Fourth Int. Conf. Genetic Algorithms,* pp. 18–23, 1991.
[6] K. A. DeJong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan (1975).
[7] K. A. DeJong, "Genetic algorithms: A 10 year perspective" in *Proceedings of an International Conference of Genetic Algorithms and their Applications,* (J Greffenstette, editor), Pittsburgh, July 24–26, 1985, pp. 169–177.
[8] K. A. DeJong, "Adaptive system design: a genetic approach," *IEEE Trans Syst, Man, and Cybernitics,* vol. 10, No. 9, pp. 566–574, Sep. 1980.
[9] T. C. Fogarty, "Varying the probability of mutation in genetic algorithms," *Proc. Third Int. Con. Genetic Algorithms,* 1989, pp. 104–109.
[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison Wesley. 1989.
[11] D. E. Goldberg, "A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing," *Complex Syst.,* vol. 4, pp. 445–460, 1990.
[12] D. E. Goldberg, "Genetic algorithms and Walsh functions: Part II, deception and its analysis," *Complex Syst.,*vvol. 3, pp. 153–171, 1989.
[13] D. E. Goldberg et. al., "Messy genetic algorithms: motivation, analysis and first results," *Complex Syst.,* vol. 3, pp. 493–530, 1989.
[14] D. E. Goldberg et. al., "Messy genetic algorithms revisited: Studies in mixed size and scale," *Complex Syst.,* vol. 4, pp. 415–444, 1990.
[15] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," *Proc. Second Int Conf. Genetic Algorithms,* 1987, pp. 41–49.
[16] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans Syst. Man, and Cybernietics,* vol. SMC-16, No. 1, pp. 122–128, Jan./Feb. 1986.
[17] J. H. Holland, *Adapation in Natural and Artificial Systems,* "Ann Arbor: Univ. Michigan Press, 1975.
[18] O. H. Ibarra and S. K. Sahni, "Polynomially complete fault detection problems," *IEEE Trans. Computers,* vol. C-24, pp. 242–247, Mar. 1975.
[19] "Special session: Recent algorithms for gate-level atpg with fault simulation and their performance assessment," *Proc.1985 IEEE Int. Symp. Circuits and Syst. (ISCAS),* June 1985, pp. 663–698.
[20] Jurgen Hesser and Reinhard Manner, "Towards an optimal mutation probability for genetic algorithms," *Proceedings of the First Workshop,* PPSN-I, pp. 23–32, 1990.
[21] I. M. Oliver, D. J. Smith and J. R. C. Holland, "A study of permutation crossover operators on the travelling salesman problem," *Proc. Second Int. Con. Genetic Algorithms,* 1987, pp. 224–230.
[22] J. D. Schaffer et. al., "A study of control parameters affecting online performance of genetic algorithms for function optimization" *Proc. Third Int. Conf. Genetic Algorithms,* 1989, pp. 51–60.
[23] W. M. Spears and K. A. DeJong, "An analysis of multipoint crossover," in the *Proc. 1990 Workshop of the Foundations of Genetic Algorithms,* 1991, pp. 301–315.
[24] D. Whitley and D. Starkweather, "Genitor-II: A distributed genetic algorithm," *J.. Expt. Theor. Artif. Int.,* vol. 2 , pp. 189–214, 1990.
[25] D. Whitley et. al., "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing,* vol. 14, pp. 347-361, 1990.



M. Srinivas (M'92) graduated from the Indian Institute of Technology, Madras, India, in 1989. From July, 1989–July, 1990, he was employed by the Centre for Development of Advanced Computing, Bangalore. He is currently a Ph.D. student at the Department of Computer Science and Automation, Indian Institute of Science, Bangalore.

His research interests include the theory and design of genetic algorithms, neural networks, stochastic optimization, and optimization in VLSI CAD algorithms.



L. M. Patnaik (S'75–M'76–SM'86–F'92) obtained the Ph.D. in real-time systems in 1978 and the D.Sc. in computer science and architectures in 1989, both from the Indian Institute of Science, Banglore, India.

He is currently Professor with the Electrical Sciences Division at Indian Institute of Science. He also directs a research group in the Microprocessor Applications Laboratory at the Institute.

Dr. Patnaik is a Distinguished Lecturer of the IEEE, Region 10. He was awarded the Dr. Vikram Sarabhai Research Award in 1989 and the Dr. Ramlal Wadhwa Award in 1992 by the Institution of Electronics and Telecommunications Engineers. He is a Fellow of the Indian National Sciences Academy, the Indian Academy of Sciences, The National Academy of Sciences, the Indian National Academy of Engineering, the Institution of Electronics and Telecommunications Engineers, and the Institution of Engineers. He is a Fellow of the Computer Society of India, a life member of the VLSI Society of India and the Instrument Society of India, and a Founding Member of the executive committee of the Association for the Advancement of Fault-Tolerant and Autonomous Systems. His name Appears in *Who's Who in the World* (eighth edition), *Reference Asia, Asia's Who's Who of Men and Women of Achievement,* and *Distinguished Computer Professionals of India.* He was the Program Chair (1990, 1991) and General Chair (1992) for the IEEE-sponsored International Conference on VLSI Design and a Member of the program committee for the Sixth International Parallel Processing Symposium (1992) and the Twenty-Second Annual Symposium on Fault-Tolerant Computing (1992). In 1993, he was the Program Chair (Asia/Australia) for the IEEE Symposium on Parallel and Distributed Processing, a Member of the Asian subcommittee for ACM Multimedia '93 Symposium, and a member of the executive committee and Coordinator, Asia and Australia, for the IEEE Technical Committee on Parallel Processing. He has been Chairman of the IEEE Computer Society chapter of the Bangalore section, for the past two years. He is the Chairman of the Indian Transputer User Group. He is a member of the editorial boards of the following publications:*International Journal of High Speed Computing, Journal of Computer-Aided Design, The Computer Journal, Parallel Algorithms and Applications, and VLSI Design: An International Journal of Custom Chip Design, Simulation and Testing.* He is Editor of *Computer Science and Informatics.*