Genetic Search: Analysis Using Fitness Moments

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

Abstract—Genetic Algorithms are efficient and robust search methods that are being employed in a plethora of applications with extremely large search spaces. The directed search mechanism employed in Genetic Algorithms performs a simultaneous and balanced, exploration of new regions in the search space and exploitation of already discovered regions.

This paper introduces the notion of fitness moments for analyzing the working of Genetic Algorithms (GAs). We show that the fitness moments in any generation may be predicted from those of the initial population. Since a knowledge of the fitness moments allows us to estimate the fitness distribution of strings, this approach provides for a method of characterizing the dynamics of GAs, in particular the average fitness and fitness variance of the population in any generation may be predicted.

We introduce the technique of fitness-based disruption of solutions for improving the performance of GAs. Using fitness moments, we demonstrate the advantages of using fitness-based disruption. We also present experimental results comparing the performance of a standard GA and GAs (CDGA and AGA) that incorporate the principle of fitness-based disruption. The experimental evidence clearly demonstrates the power of fitness based disruption.

Index Terms—Search methods, genetic algorithms, fitness moments, adaptive variation of parameters, controlled disruption.

1 INTRODUCTION

Genetic Algorithms are probabilistic search methods that employ a search technique based on ideas from natural genetics and evolutionary principles. They were conceived by Holland [15] in 1975, and since then, they have emerged as general purpose, robust search techniques (see [10], [14], [15], [24]).

Genetic Algorithms 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 Genetic Algorithm (hereafter referred to as 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 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 the population of feasible solutions to the optimization problem evolves over a sequence of generations. During each generation, the fitness of each solution is evaluated, and solutions are selected for reproduction based on the fitness values. 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 reproduction 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 for simple implementation. Goldberg [12] provides some theoretical perspectives for the choice of low cardinality alphabets for encoding the solutions.

Crossover causes the exchange of ‘genes’ between two randomly selected ‘parents’ to form new ‘offspring.’ The crossover occurs only with some probability (the crossover rate), and when the solutions are not subjected to crossover, they remain unmodified. The intuition behind crossover points to the possibility of structured exchange of ‘genes’ between ‘good’ solutions to form ‘better’ offspring. The notable crossover techniques include the single-point, the two-point, and the uniform types [29].

Mutation involves the modification of the values of each ‘gene’ of a solution with some probability (the mutation rate). Though the traditional role of mutation in GAs has been that of restoring lost or unexplored genetic material into the population, recent reports [19] have shown that mutation can play an important role in the working of GAs.

Apart from selection, crossover, and mutation, various other auxiliary operations are common in GAs. Of these, scaling mechanisms are widely used. Scaling involves a re-
adjustment 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 regions in the search space.

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

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. Structure of a simple GA.

1.1 Theoretical Foundations

The theoretical foundations of GAs are set around the ‘Schema theorem’ [15], and the ‘implicit parallelism’ result [15], [10]. The ‘Schema theorem’ (also referred to as the ‘Fundamental theorem of GAs’) states that schemata with average fitnesses for short defining lengths are sampled by the GA at exponentially increasing rates. The following is a formal expression of the Schema theorem:

\[ N(h, t + 1) \geq N(h, t) \frac{f(h, t)}{f(t)} \left[ 1 - p_c \frac{\delta(h)}{l - 1} - p_m \phi(h) \right] \]  

(1)

where

\[ f(h, t) \] average fitness value of the representatives of the schema \( h \) in generation \( t \)
\[ f(t) \] average fitness value of the population in generation \( t \)
\[ p_c \] crossover probability
\[ p_m \] mutation probability
\[ \delta(h) \] defining length of the schema
\[ \phi(h) \] order of the schema
\[ N(h, t) \] expected number of representatives of a schema \( h \) in generation \( t \)

The working of Genetic Algorithms involves the processing of a large number of schemata (much larger than the number of strings in the population). This capacity of the GA to simultaneously process a large number of schemata is referred to as implicit parallelism. It has been shown that approximately \( N \) schemata are processed by the GA implicitly while it processes only \( N \) strings [10], [15]. The success of GAs in exploring complex fitness landscapes has often been attributed to the implicit parallelism in the processing of schemata.

In [4] and [20], modifications of the schema theorem have been proposed: In [4], an attempt has been made at transforming the inequality of (1) into an equality relation by considering the schema gains due to crossover and mutation. In [20], the Schema theorem is refined by the addition of a new term to model the effect of crossover between strings of the same schema.

Previous work relating to the convergence properties and dynamics of GAs includes [11], [18], [27], [28], [30], [31], [11], [30], and [18] describe Markov chain analyses of Genetic algorithms. [11] describes a Markov chain analysis of a single-locus binary allele finite population GA. An analysis of genetic drift has been performed for the above case. In [30], the simulated annealing convergence theory based on a Markov chain model has been extrapolated to analyze the convergence properties and stationary distributions of GAs. [18] also deals with the steady state distributions of GAs based on the state transition matrices for selection and recombination introduced in [31].

In [31], the phenomenon of ‘Punctuated Equilibria’ in GAs has been analyzed by a stability analysis of the selection and recombination operators in terms of the eigenvalues of the matrices representing the operators.

In [27], the notion of Binomially Distributed Populations has been introduced to exactly characterize the dynamics of GAs. A efficient GA simulator (GASIM) based on the above populationary model has been developed and validated for functions of unitation. It has been demonstrated that the model can be extended to cover any general objective function [28].

1.2 Aim of this Paper

The central theme of this paper is the concept of fitness moments of schemata to analyze and predict the behaviour of GAs based on the distribution of fitness values in the initial population. We introduce the notion of fitness moments and derive simple expressions that relate the fitness moments for any generation to those of the initial population. By evaluating the fitness moments of the initial population, we may predict the fitness moments in any generation. Particularly, the average fitness and the standard deviation of fitness values in any generation may be evaluated. From the theory developed in the paper, we predict that the performance of GAs in converging to the optimal solution in the search space may be improved by controlling the disruption rate of solutions on the basis of their fitness values. Simple implementation of GAs embodying the above principle demonstrate our claim.

1.3 Motivation

Following is the motivation for developing the ‘theory of fitness moments’ for analyzing the behaviour of GAs.

The Schema theorem may be represented in a simpler form by considering operator-adjusted fitness values (see [13]):

\[ N(h, t + 1) \geq N(h, t) \frac{f'(h, t)}{f(t)} \]  

(2)

where \( f'(h, t) \) is the operator-adjusted fitness value of the schema \( h \), i.e.,

\[ f'(h, t) = \left[ 1 - p_c \frac{\delta(h)}{l - 1} - p_m \phi(h) \right] f(h, t) \].

---

1 A schema is a similarity template describing a subset of strings (a string is the encoded representation of a solution) with similarities at certain positions.
2 The physical distance between the outermost fixed positions of a schema.
3 The number of fixed positions in the schema.
The above equation may be applied to the generations \( t \) and \( t - 1 \) to relate the expected number of representatives of \( h \):

\[
N(h, t) \geq \frac{N(h, t-1) f'(h, t-1)}{f(t-1)} \tag{3}
\]

Combining (2) and (3) we may relate \( N(h, t+1) \) and \( N(h, t-1) \) as follows:

\[
N(h, t+1) \geq N(h, t-1) \frac{f'(h, t) f'(h, t-1)}{f(t) f(t-1)} \tag{4}
\]

Similarly we may express \( N(h, t+1) \) in terms of \( N(h, t-T) \) as,

\[
N(h, t+1) \geq N(h, t-T) \prod_{i=t-T}^{t} \frac{f'(h, i)}{f(i)} \tag{5}
\]

Particularly, we may obtain \( N(h, t+1) \) in terms of \( N(h, 1) \), the number of instances of \( h \) in the initial population:

\[
N(h, t+1) \geq N(h, 1) \prod_{i=1}^{t} \frac{f'(h, i)}{f(i)} \tag{6}
\]

Equation (6) lends a suggestion that we may evaluate \( N(h, t+1) \) from \( N(h, 1) \), which in turn can be evaluated, or estimated from the distribution of instances of \( h \) in the search space. However, we also need to express \( f'(h, i) \) and \( f(i) \) in terms of some mathematical characteristics of the initial population.

With the above stated goal in mind, this paper introduces the notion of fitness moments (moments of fitness values) for analyzing the behavior of Genetic Algorithms.

### 1.4 Organization of the Paper

The paper is organized as follows. Section 2.1 introduces the notions of fitness moments of schemata. In Section 2.2, we characterize the effects of selection by relating the fitness moments in different generations. In Section 2.3, the corresponding relation between the fitness moments is derived after incorporating the effects of crossover and mutation. In Section 3, we show that the performance of the standard GA can be improved by controlling the extent of disruption of solutions on the basis of their fitness values. In Section 4, we demonstrate the potential of the ‘fitness based disruption’ technique by means of two implementations of GAs that incorporate this principle: the Controlled Disruption Genetic Algorithm (CDGA), and the Adaptive Genetic Algorithm (AGA). A brief discussion of other similar approaches is presented, and experimental results are reported to illustrate the performance improvement. The conclusions are presented in Section 5.

### 2 Fitness Moments of Schemata

The following notation will be followed in the paper:

- \( \eta_k(t) \): expected number of copies of the solution \( k \) in generation \( t \)
- \( f_k \): fitness value of the solution \( k \)
- \( f(h, t) \): average fitness of the instances of schema \( h \) in generation \( t \)
- \( M(h, t) \): \( i \)th fitness moment of schema \( h \) in generation \( t \)
- \( R(h) \): number of possible strings of the schema \( h \)
- \( \mathcal{N}(h, t) \): number of instances of \( h \) in generation \( t \)
- \( \bar{f}(t) \): average fitness of the population in generation \( t \)
- \( M(t) \): \( i \)th fitness moment of the population in generation \( t \)
- \( \mathcal{N} \): population size
- \( R \): number of possible strings

#### 2.1 Schemata and Fitness Moments

Before introducing the idea of fitness moments of schemata, we briefly discuss the notion of schemata. A schema (also referred to as a building block or similarity template) represents the subset of all possible strings that have similarities at certain positions (fixed positions). The standard representation for a schema includes an additional default symbol (apart from the symbols of the alphabet) such as a ‘+’ at all loci that are not the fixed positions of the schema. The order of a schema, \( o(h) \), is the number of fixed positions in the schema. The defining length of a schema, \( l(h) \), is the distance between the outermost fixed positions of the schema.

A simple example illustrates these concepts. For strings of length 6 defined on a binary alphabet, \( 0^{a}0^{b}0^{c}0^{d}0^{e}0^{f} \) is a schema that represents the set of solutions \( 000010, 000100, 001010, 010010, 010001, 011010, 011011 \). The defining length of \( 0^{a}0^{b}0^{c}0^{d}0^{e}0^{f} \) is 4 and its order is 3.

The average fitness of a schema in generation \( t \) is the average fitness of instances (strings) of the schema. We now extend the definition of average fitness of a schema to the higher order moments of fitness of a schema.

We define \( M(h, t) \), the \( i \)th fitness moment of schema \( h \) in generation \( t \), as

\[
M_i(h, t) = \frac{1}{\mathcal{N}(h, t)} \sum_{k=1}^{\mathcal{R}(h)} \eta_k(t+1) f_k^i \tag{7}
\]

where \( \mathcal{N}(h, t) = \sum_{k=1}^{\mathcal{R}(h)} \eta_k(t) \). \( M(h, t) \) is the average fitness of schema \( h \), and the variance of fitness of \( h \) may be expressed as\( M(h, t) - (M(h, t))^2 \).

#### 2.2 Proportional selection

In this section, we derive the expressions that relate the fitness moments of \( h \) in different generations, while considering only the action of proportional selection. In the next section, these results are extended to include the disruptive effects of crossover and mutation.

Consider the \( i \)th fitness moment of schema \( h \) in generation \( t+1 \):

\[
M_i(h, t+1) = \frac{1}{\mathcal{N}(h, t+1)} \sum_{k=1, k \neq h}^{\mathcal{R}(h)} \eta_k(t+1) f_k^i \tag{8}
\]

Recalling the expression for proportional selection, i.e.,

\[
\eta_k(t+1) = \eta_k(t) \frac{f_k}{\bar{f}(t)} \tag{9}
\]

For a binary alphabet \( \mathcal{R}(h) = 2^{l-o(h)} \), where \( o(h) \) is the number of fixed positions in the string.
and substituting for \( n_i(t + 1) \) in (8), we obtain
\[
M'(h, t + 1) = \frac{1}{N(h, t + 1)} \sum_{k=1,\text{c.h.}}^{R(h)} n_i(t) \frac{(f_i(t))^{i+1}}{f_i(t)}.
\] (10)

\( N(h, t + 1) \) may be expressed in terms of \( N(h, t) \) as
\[
N(h, t + 1) = N(h, t) \frac{M'(h, t)}{f_i(t)}.
\] (11)

Note that
\[
\frac{1}{N(h, t)} \sum_{k=1,\text{c.h.}}^{R(h)} n_i(t) (f_i(t))^{i+1} = M^{i+1}(h, t).
\] (12)

From (10), (11), and (12), we get
\[
M'(h, t + 1) = \frac{M^{i+1}(h, t)}{M'(h, t)}.
\] (13)

The above equation states that the \( i \)-th fitness moment of \( h \) in generation \( t + 1 \) is equal to the ratio of the \( (i+1) \)-th fitness moment of \( h \) in generation \( t \) to the average fitness of \( h \) in generation \( t \).

Using (13), we may relate the fitness moments in consecutive generations of the GA in the following fashion:

\[
M^{i+1}(h, t) = \frac{M^{i+2}(h, t - 1)}{M'(h, t - 1)}
\]

\[
M^{i+2}(h, t - 1) = \frac{M^{i+3}(h, t - 2)}{M'(h, t - 2)}
\]

\[
M^{i+3}(h, t - 2) = \frac{M^{i+4}(h, t - 3)}{M'(h, t - 3)}
\]

\[\vdots\]

\[
M^{i+k}(h, t - k + 1) = \frac{M^{i+k+1}(h, t - k)}{M'(h, t - k)}
\]

\[\vdots\]

\[
M^{i+k-1}(h, 2) = \frac{M^{i+k}(h, 1)}{M'(h, 1)}.
\]

From the above set of equations, it may be shown easily that
\[
M^{i+1}(h, t) = \frac{M^{i+k+1}(h, t - k)}{\prod_{j=t-k}^{t-1} M'(h, j)}.
\] (14)

For \( i = 0 \) (14), we get the average fitness of \( h \) in generation \( t \) to be
\[
M'(h, t) = \frac{M^{i+k}(h, t - k)}{\prod_{j=t-k}^{t-1} M'(h, j)}.
\] (15)

Equivalently, we may reformulate (15) as,
\[
\prod_{j=t-k}^{t} M'(h, j) = M^{i+k}(h, t - k).
\] (16)

The above equation is interesting as it shows that the \((k + 1)\)-th fitness moment of \( h \) in generation \( t - k \) is equal to the product of the \( k + 1 \) average fitnesses of \( h \) in generations \( t - k \) to \( t \).

We now use (16) as a basis to show that the average fitness of \( h \) in generation \( t \) can be calculated from the fitness moments of the first generation. On substituting \( k = t - 1 \) in (16), we get
\[
\prod_{j=1}^{t} M'(h, j) = M'(h, 1).
\] (17)

Replacing the variable \( t \) by \( t - 1 \) in the above equation, we also get
\[
\prod_{j=1}^{t-1} M'(h, j) = M'(h, 1).
\] (18)

From (17) and (18), it follows that
\[
M'(h, 1) = \frac{M^{i+1}(h, 1)}{M^{i+1}(h, 1)}.
\] (19)

A rearrangement of terms yields the following form of (19):
\[
M'(h, t) = \frac{M'(h, 1)}{M^{-1}(h, 1)}.
\] (20)

Equation (20) signifies that the average fitness of \( h \) in generation \( t \) is equal to the ratio of the \((i+1)\)-th fitness moment of \( h \) in the first generation to the \((t-1)\)-th fitness moment of \( h \) in the first generation.

In a similar fashion, it can be shown that the \( i \)-th fitness moment of \( h \) in generation \( t \) is equal to the average fitness of \( h \) in generation \( t \) to the \((i+1)\)-th fitness moment of \( h \) in the first generation.

Following are the salient steps in deriving the above result. By substituting \( t - k = 1 \) and replacing \( t + 1 \) by \( t \) in (14) we get the following equation:
\[
M'(h, t) = \frac{M^{i+k+1}(h, t - k)}{\prod_{j=t-k}^{t-1} M'(h, j)}.
\] (21)

However, we know from (18) that,
\[
\prod_{j=t-k}^{t-1} M'(h, j) = M^{-1}(h, 1).
\] (22)

Consequently we may rewrite (21) as
\[
M'(h, t) = \frac{M^{i+k+1}(h, t - k)}{M^{-1}(h, 1)}.
\] (23)

2.2.1 Fitness Moments of the Population

Since the order-0 schema (i.e., the schema with no fixed positions) represents all possible strings over the alphabet under consideration, the fitness moments of the population are identical to the fitness moments of the order-0 schema. Thus, the following relations are true for fitness proportional selection:
\[
M'(h, t) = \frac{M'(h, 1)}{M^{-1}(h, 1)}.
\] (24)
\[ M'(t) = \frac{M^{i+1}(t)}{M^{i-1}(t)} \]  

In the next section, the above described analysis of fitness moments is extended to include the effects of crossover and mutation. In the rest of the paper we focus our attention on the fitness moments of the population only.

2.3 Effects of Crossover and Mutation

The effects of recombination and disruption due to crossover and mutation have not been considered in the preceding section; (24) and (25) are representative of selection alone. In this section, we extend the analysis of fitness moments to incorporate the effects of crossover and mutation. We model the disruptive and generative effects of the crossover and mutation operators.

We recall (1) which represents the Schema theorem:

\[ N(h, t+1) \geq N(h, t) \frac{f(h, t)}{f(t)} \left[ 1 - p_c \frac{\delta(h)}{1 - \delta(h)} - p_m \phi(h) \right]. \]  

Calling \( 1 - p_c \frac{\delta(h)}{1 - \delta(h)} - p_m \phi(h) \) the disruption constant \( \mathcal{K}_h \), we may rewrite (26) as,

\[ N(h, t+1) \geq N(h, t) \mathcal{K}_h \frac{f(h, t)}{f(t)}. \]  

The basic form of the above expression is similar to that of (11) except for the disruption factor \( \mathcal{K}_h \) and the inequality relationship. In a fashion similar to that described in Section 2.2, the \( i \)th fitness moment of generation \( t+1 \) may be expressed in terms of the \((i+1)th\) fitness moment of generation \( t \) as,

\[ M'(h, t+1) \geq \mathcal{K}_h \frac{M^{i+1}(h, t)}{M^{i}(h, t)}. \]  

The counterpart of (16) takes the form,

\[ \prod_{j=t-k}^{t} M'(h, j) \geq \mathcal{K}_h \mathcal{K}^{t-k} M^{1+k}(h, t-k). \]  

The equation relating the \((i)th\) fitness moment of the first generation to the average fitnesses of the subsequent generations now becomes the following inequality:

\[ \prod_{j=1}^{t} M'(h, j) \geq \mathcal{K}^{i-1} M'(h, 1). \]  

The above statement states that the product of the average fitnesses of \( h \) in the first \( t \) generations is greater than the \( i \)th fitness moment of \( h \) in the first generation times \( \mathcal{K}^{i-1} \).

For the order-0 schema (29) may be specifically expressed as

\[ \prod_{j=t-k}^{t} M'(j) \geq \mathcal{K}^{t-k} M^{1+k}(t-k). \]

where \( \mathcal{K} = 1 - p_c - p_m \).

In the previous section, we have arrived at the average fitness of \( h \) by a simple algebraic manipulation of (17). The same approach may not be used here for obtaining the average fitness of the population.

2.3.1 A Simple Model For the Fitness Moments of The Generated Solutions

We overcome the above stated problem by extending (1) to include a ‘generative’ term that represents the expected number of strings of a schema generated as a consequence of crossover and mutation. The expected percentage of strings that are disrupted, and consequently the percentage of strings that are ‘regenerated’ during each generation are given by \( \mathcal{K} \). First we assume that the \( i \)th fitness moment of the generated solutions is a generation-independent constant \( \alpha \) (for example \( \alpha \) may correspond to the average fitness of all instances of the 0-order schema). Hence, the expected contribution of the generated solutions to the average fitness of the population is given by \( (1 - \mathcal{K}) \alpha^i \).

Consequently the \( i \)th fitness moment in generation \( t+1 \), and the \((i+1)th\) fitness moment in generation \( t \) may be related as:

\[ M'(t+1) = \mathcal{K} \frac{M^{i+1}(t)}{M^{i}(t)} + (1 - \mathcal{K})\alpha^i \]  

Using the above equation as a basis, it may be shown that the average fitness of the population in any generation \( t \) is related to the fitness moments of the population in the first generation as

\[ m^i(t) = \frac{\mathcal{K}^{i-1} M^i(1) + (1 - \mathcal{K})\alpha \sum_{j=1}^{i-1} M^j(1)\alpha^{j-1} \mathcal{K}^{j-1}}{\mathcal{K}^{i-2} M^{i-1}(1) + (1 - \mathcal{K})\alpha \sum_{j=1}^{i-2} M^j(1)\alpha^{j-2} \mathcal{K}^{j-1}} \]  

We consider two cases of the above equation for different values of \( \alpha \).

For \( \alpha = 0 \), the above equation reduces to the following form:

\[ M'(t) = \mathcal{K} \frac{M'(1)}{M'(1)} \]  

The above case corresponds to the hypothetical situation wherein all the generated solutions have zero fitness, with \( M'(t) \) depending only on \( M'(1) \) and \( \mathcal{K}' \) only.

For \( \alpha = 1 \) we get

\[ m^i(t) = \frac{\mathcal{K}^{i-1} M^i(1) + (1 - \mathcal{K})\sum_{j=1}^{i-1} M^j(1)\mathcal{K}^{j-1}}{\mathcal{K}^{i-2} M^{i-1}(1) + (1 - \mathcal{K})\sum_{j=1}^{i-2} M^j(1)\mathcal{K}^{j-1}} \]  

In the situation depicted by the above equation, all solutions generated due to crossover and mutation have a fitness of unity.

Though the above two cases represent two hypothetical situations, they are useful since we conjecture that the fitness moments of the population in a practical situation will be bounded by the corresponding fitness moments for these extreme cases, as the average fitness of the generated solutions is bounded by the values 0 and 1.
2.3.2 A More Realistic Model

We now consider a more realistic model for the fitness moments of the generated solutions, that is depicted by the following equation:

$$M'(t+1) = \mathcal{K} \frac{M'^{t+1}(t)}{M'(t)} + (1 - \mathcal{K}) M'(1)$$  \hspace{1cm} (36)

In the above equation, the fitness moments for the generated solutions are assumed to be identical to the fitness moments of the initial population, wherein the solutions are uniformly distributed in the search space.

A very cumbersome expression results when (36) is generalized to relate the fitness moments of any generation to the fitness moments of the initial population. We avoid going into the details of the derivation. Alternately, we calculate the average fitness in any generation by recursively applying (36) to the fitness moments of the initial population. The next section presents a comparison of the average fitness predicted from (36), and the observed average fitness for a simple GA. The comparison is performed over a wide range of crossover and mutation probabilities, and two objective functions.

2.3.3 Experimental Evidence

Our experiments are based on a simple GA with the following details: two point crossover, and fitness proportional selection.

The fitness moments of the initial population, and the average fitness of the population in the thirtieth generation have been recorded for a range of crossover and mutation probabilities. Using (36) recursively on the fitness moments of the initial population, we have also calculated the average fitness in the thirtieth generation. We consider two objective functions for a proof of principle demonstration; one unimodal and one multimodal. The unimodal function is the shifted form of DeJong's s1 [7]. The multimodal function has five variables and 243 local optima. The value of each variable is shifted by 10% to the left, and wrapped around the upper limit of the permissible range in case the value is less than the lower limit. This is done to shift the optimal solutions away from Hamming cliffs (see [5]). The expressions for the objective functions are as follows:

$$S_f1(x_i) = \sum_{i=1}^{3} x_i \text{, } -5.12 \leq x_i < 5.12$$

$$S_f6(x_i) = \sum_{i=1}^{5} \sin(5\pi/2x_i) \text{, } -1.0 \leq x_i < 1.0$$

The average fitness has been recorded after 30 generations for

1) different values of $p_c$ with $p_m$ being set to zero,
2) different values of $p_m$ with $p_c$ being set to zero.

Tables 1 and 2 illustrate the predicted and observed average fitnesses for Sf1 and Sf6.

The following may be inferred from Tables 1 and 2: For zero crossover and mutation probabilities, the predicted value of the average fitness is in close agreement with the observed average fitness. For increasing values of $p_m$, the predicted average fitness decreases at a faster rate than that for the observed average fitness. Similarly, with increasing values of $p_c$, the predicted values of the average fitness decrease rapidly, while the observed values remain approximately constant.

<table>
<thead>
<tr>
<th>TABLE 1</th>
<th>PREDICTED AND OBSERVED AVERAGE FITNESS AFTER 30 GENERATIONS FOR Sf1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_c$</td>
<td>$p_m$</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.001</td>
</tr>
<tr>
<td>0.0</td>
<td>0.005</td>
</tr>
<tr>
<td>0.0</td>
<td>0.01</td>
</tr>
<tr>
<td>0.0</td>
<td>0.05</td>
</tr>
<tr>
<td>0.0</td>
<td>0.1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.5</td>
<td>0.005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 2</th>
<th>PREDICTED AND OBSERVED AVERAGE FITNESS AFTER 30 GENERATIONS FOR Sf6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_c$</td>
<td>$p_m$</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.001</td>
</tr>
<tr>
<td>0.0</td>
<td>0.005</td>
</tr>
<tr>
<td>0.0</td>
<td>0.01</td>
</tr>
<tr>
<td>0.0</td>
<td>0.05</td>
</tr>
<tr>
<td>0.0</td>
<td>0.1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.5</td>
<td>0.005</td>
</tr>
</tbody>
</table>

2.3.4 Modified Expression for the Disruption Factor

The cause for the deviation of the observed average fitness from the predicted average fitness is the nonuniform distribution of the generated solutions in the search space. While every solution that undergoes recombination or mutation is disrupted, the change in its fitness value will depend on the nature of crossover and mutation. For example, an alteration of the less significant bits of each parameter-code due to mutation will cause a relatively small change in the value of the variable, and consequently a small change in the fitness of the solution. Only mutation occurring on the most significant bits of the binary codes will perturb the solutions sufficiently to warrant the assumption that new solutions are uniformly distributed in the search space.

Though solutions may be disrupted due to crossover, the resultant change in the fitness moments of the population may be neglected. Consider two situations:

1) The population has converged to a small region in the search space, and the strings are similar at a large number of loci.
2) The population is scattered in the search space.

In the first case, crossover occurs between 'similar' solutions and produces 'offspring' that are also similar to
the ‘parents.’ The ‘offspring’ will then have fitnesses that are very close to the fitnesses of the ‘parents.’ The change in the value of the fitness moments will not be significant. In the second case, the ‘offspring’ also will be scattered in the search space. Again the resultant change in the fitness moments of the population is expected to be small. The effect of varying $p_n$ will not be as pronounced as that of varying $p_m$. This relative insensitivity of the performance of GAs to the crossover probability (as compared to the mutation probability or population size) has also been observed in [19].

### Table 3

**Predicted and Observed Average Fitness after 30 Generations for S11: Modified Expression**

<table>
<thead>
<tr>
<th>$p_m$</th>
<th>Pred. Av.</th>
<th>Obs. Av.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.952</td>
<td>0.944</td>
</tr>
<tr>
<td>0.001</td>
<td>0.951</td>
<td>0.957</td>
</tr>
<tr>
<td>0.005</td>
<td>0.947</td>
<td>0.954</td>
</tr>
<tr>
<td>0.01</td>
<td>0.943</td>
<td>0.957</td>
</tr>
<tr>
<td>0.05</td>
<td>0.900</td>
<td>0.919</td>
</tr>
<tr>
<td>0.1</td>
<td>0.863</td>
<td>0.871</td>
</tr>
<tr>
<td>0.5</td>
<td>0.672</td>
<td>0.888</td>
</tr>
</tbody>
</table>

### Table 4

**Predicted and Observed Average Fitness After 30 Generations for S16: Modified Expression**

<table>
<thead>
<tr>
<th>$p_m$</th>
<th>Pred. Av.</th>
<th>Obs. Av.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.948</td>
<td>0.947</td>
</tr>
<tr>
<td>0.001</td>
<td>0.947</td>
<td>0.944</td>
</tr>
<tr>
<td>0.005</td>
<td>0.942</td>
<td>0.940</td>
</tr>
<tr>
<td>0.01</td>
<td>0.938</td>
<td>0.944</td>
</tr>
<tr>
<td>0.05</td>
<td>0.900</td>
<td>0.900</td>
</tr>
<tr>
<td>0.1</td>
<td>0.852</td>
<td>0.849</td>
</tr>
<tr>
<td>0.5</td>
<td>0.512</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Consequently we have considered a modified expression for the disruption constant $K$ that is independent of $p_m$:

$$K = (1.0 - p_m)^2$$

(37)

where $n$ is a problem specific parameter that may be intuitively associated to the number of bits in a string that significantly alter the fitness of the solution. Tables 3 and 4 give the average fitness of the population that have been predicted using the modified expression for the disruption constant.

A very good fit between the observed and predicted average fitness has been obtained for $n = 6$ (S11) and $n = 10$ (S16). The above values of $n$ correspond to two bits per variable i.e., mutation of the first two significant bits in each parameter code cause a significant change in the fitnesses.

Tables 5, 6, and 7 present the predicted and observed average fitness after 30 generations for three other objective functions:

- $Sf3 = \sum_{i=1}^{5} x_i^2$  
  $-5.12 < x_i \leq 5.12$

- $Sf7 = 0.5 + \frac{\sin^2(\sqrt{x_i^2 + x_i^2} - 0.5)}{1.00 + 0.001(x_i^2 + x_i^2)}$  
  $-100.0 < x_i \leq 100.0$

- $Sf9 = nA + \sum_{i=1}^{5} x_i^2 - \cos(2\pi x_i)$  
  $-5.12 < x_i \leq 5.12$

where $n = 20$, $A = 10$, $\omega = 2\pi$

### Table 5

**Predicted and Observed Average Fitness after 30 Generations for S13**

<table>
<thead>
<tr>
<th>$p_m$</th>
<th>Pred. Av.</th>
<th>Obs. Av.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>0.737</td>
<td>0.738</td>
</tr>
<tr>
<td>0.005</td>
<td>0.719</td>
<td>0.736</td>
</tr>
<tr>
<td>0.01</td>
<td>0.701</td>
<td>0.710</td>
</tr>
<tr>
<td>0.05</td>
<td>0.589</td>
<td>0.610</td>
</tr>
<tr>
<td>0.1</td>
<td>0.543</td>
<td>0.547</td>
</tr>
<tr>
<td>0.5</td>
<td>0.614</td>
<td>0.494</td>
</tr>
</tbody>
</table>

### Table 6

**Predicted and Observed Average Fitness after 30 Generations for S17**

<table>
<thead>
<tr>
<th>$p_m$</th>
<th>Pred. Av.</th>
<th>Obs. Av.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>0.885</td>
<td>0.900</td>
</tr>
<tr>
<td>0.005</td>
<td>0.862</td>
<td>0.852</td>
</tr>
<tr>
<td>0.01</td>
<td>0.825</td>
<td>0.822</td>
</tr>
<tr>
<td>0.05</td>
<td>0.755</td>
<td>0.758</td>
</tr>
<tr>
<td>0.1</td>
<td>0.740</td>
<td>0.754</td>
</tr>
<tr>
<td>0.5</td>
<td>0.746</td>
<td>0.748</td>
</tr>
</tbody>
</table>

### Table 7

**Predicted and Observed Average Fitness after 30 Generations for S19**

<table>
<thead>
<tr>
<th>$p_m$</th>
<th>Pred. Av.</th>
<th>Obs. Av.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>0.720</td>
<td>0.717</td>
</tr>
<tr>
<td>0.005</td>
<td>0.691</td>
<td>0.693</td>
</tr>
<tr>
<td>0.01</td>
<td>0.662</td>
<td>0.650</td>
</tr>
<tr>
<td>0.05</td>
<td>0.614</td>
<td>0.613</td>
</tr>
</tbody>
</table>

For S13 and S17, (with five and two variables, respectively), the optimal values for $n$ have been observed to be 10 and four, again corresponding to two bits per variable. For Rastrigin’s function—S19, a highly oscillatory function with 20 variables, the best fit for $n$ is 20, signifying one bit per variable. In each case the difference between the observed and predicted average fitnesses has been consistently less than 5%. This highlights the robustness of the analysis technique, albeit the need to choose the value for $n$.

### Table 8

**Effect of Number of Variables on $n$**

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f1$</th>
<th>$f2$</th>
<th>$f3$</th>
<th>$f5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>10</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 8 shows how $n$ varies with the number of variables $v$, for $f1$, $f2$, $f3$, and $f5$. It is clear that the optimal value of $n$ increases with the number of variables of the objective function, and corresponds to approximately 1 bit per variable for large $v$.

Table 9 illustrates the dependence of $n$ on the multimodality of the function. The objective function is similar to S16:

$$Sf6(x_i) = \sum_{i=1}^{5} \frac{\sin(k \omega x_i)}{k \omega x_i}$$

$-1.0 < x_i < 1.0$.
$n$ has been tabulated for different values of $k$. As the number of local optima increases, the value of $n$ tends towards 5, i.e., $n$ corresponds to 1 bit per variable.

### TABLE 9

<table>
<thead>
<tr>
<th>$k$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

On the basis of the empirical evidence, we suggest the following heuristic for $n$: $n = \beta v$ where $v$ is the number of variables, and $1.0 \leq \beta \leq 2.0$, with $\beta = 2.0$ for unimodal functions, and lower values of $\beta$ as the objective function becomes increasingly multimodal.

To summarize, in this section, we have focused on a novel analysis technique for GAs that may be employed to predict the average fitness of the population in any generation from the fitness moments of the initial population. Experimental evidence points to the success of this approach.

In the next section, we show that a controlled disruption of solutions on the basis of their fitness results in an improved performance of the GA. We demonstrate our claim with an implementation of a GA that incorporates the above principle.

### 3 FITNESS-BASED DISRUPTION OF SOLUTIONS

Most of GA implementations incorporate static crossover and mutation probabilities that remain unchanged during an entire run of the GA. This results in an equal disruption probability for all solutions, irrespective of their fitnesses. Relatively higher values of the crossover and mutation probabilities improve the exploratory power of the GA at the cost of losing potentially ‘good’ solutions. On the other hand, the GA tends to exploit ‘good’ solutions at lower values of the crossover and mutation probabilities, but may converge to suboptimal solutions. A tradeoff between exploitation and exploration is achieved by the GA, and this tradeoff depends critically on the mutation and crossover probabilities.

In [23] a new technique is discussed, wherein the mutation and crossover probabilities have been made dynamic and dependent on the fitness of each solution, to simultaneously improve the exploration and exploitation capacities of the GA. Theoretical and experimental evidence has been provided to illustrate the superior performance of the new approach.

In this section, using the notion of fitness moments, we show that a fitness dependent disruption factor, as employed in [23], enhances the performance of the GA. To illustrate the idea, we consider the case wherein the disruption factor for a solution $i$ is given by,

$$ \mathcal{K} (f_i) = \mathcal{K}_0 f_i $$  \hspace{1cm} (38)

In conjunction with the Schema theorem (2) for the order-0 schema, and using (38), we get the following relation for the $i$th fitness moment in the generation $t+1$:

$$ M'(t+1) \geq \mathcal{K}_0 \frac{M'^{i+2}(t)}{M'(t)} $$  \hspace{1cm} (39)

The counterpart of (40) for the standard GA is given by:

$$ M'(t+1) \geq \mathcal{K} \frac{M'^{i+2}(t)}{M'(t)} $$  \hspace{1cm} (40)

We compare the right sides of (39) and (40). To make an unbiased comparison, the total number of expected disruptions should be equal in both the cases. For the case of fitness dependent disruption, the expected number of disruptions (per solution) is given by $\mathcal{K}_0 M'(t)$. The expected number of disruptions for the standard GA is given by $\mathcal{K} M'(t)$. Where $\mathcal{K}$ is the disruption factor itself. Equating the two factors, we get $\mathcal{K} = \mathcal{K}_0 M'(t)$.

After simplification, the comparison reduces to $M'^{i+2}(t)$ and $\mathcal{K} M'(t)$ and $\mathcal{K} M'(t)$. It can be easily shown that the fitness moments satisfy the following property

$$ M'^{i+2}(t) \geq \mathcal{K} M'(t) M'(t) $$  \hspace{1cm} (41)

for any $p, q$, and $t$. Appendix A gives a simple proof for the above relation.

Consequently we get

$$ M'^{i+2}(t) \geq \mathcal{K} M'(t) M'(t) $$  \hspace{1cm} (42)

The above equation indicates that, in any generation, a higher average fitness can be expected for a GA that incorporates fitness-based disruption than for a GA with a constant disruption factor. A better online performance is expected since the average fitness is expected to be higher in each generation. Also, a better offline performance may be expected since the ‘good’ solutions have a lower probability of being disrupted than the ‘bad’ solutions. We conjecture that a GA that uses a form of fitness-based disruption that ‘protects’ solutions with higher fitnesses and vigorously recombinates solutions with lower fitnesses should demonstrate a superior performance.

In the next section we demonstrate our claim by giving a comparison of performances between a standard GA and two GAs that incorporate fitness-based disruption: Controlled Disruption Genetic Algorithm (CDGA) and the Adaptive Genetic Algorithm (AGA).

### 4 FITNESS BASED DISRUPTION FOR IMPROVING GENETIC SEARCH

This section describes two implementations of GAs that incorporate fitness-based disruption of solutions. In the first case, the position of the crossover point and the positions of the bits that may be mutated are determined on the basis of the fitness of the solution to achieve a control over the disruption probability of the solution. In the second case, the crossover and mutation probabilities are determined for each solution based on its fitness. In both cases, our idea is to ‘protect’ the best solutions of the population to exploit the information they contain in locating better solutions, while we disrupt solutions with lower fitnesses to acceler-
ate search of new regions of the search space, thus simultaneously improving the exploitative and exploratory capacities of the GA.

4.1 Previous Work

The GA-variants that we propose in this section are different from the standard GA since they employ dynamic strategies to control the disruption of solutions. In the literature several other dynamic strategies have been proposed for enhancing GA performance. We briefly review some of these techniques.

ARGOT [22], Delta Coding [33], and DPE [21] strategies lay emphasis on controlling the accuracy of representation of parameters and the range of the parameters of the objective function, to improve the convergence of the GA and to obtain more precise solutions without added computational effort. ARGOT uses varying parameter resolutions and dynamic parameter boundaries to improve the optimization by the GA. Delta Coding represents solutions as 'delta' values around partial solutions and limits the search to the 'delta hypercubes.' The 'delta hypercube' may expand or contract in response to the performance of the GA. DPE is also similar to ARGOT as it uses a dynamic parameter encoding strategy. The accuracy of the parameter codes is increased by discarding the most significant bits and by introducing new bits at the least significant positions.

The idea of adapting crossover and mutation operators to improve the performance of GAs has been employed earlier [1], [6], [9], [33].

Schaefer and Morishima [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 [6] 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 allotted higher probabilities. 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 and Starkweather [33], 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.

CDGA is similar to ARGOT, Delta Coding, and DPE as it distinguishes between bits of one parameter-codes from those of another. However, CDGA differs from ARGOT, Delta Coding, and DPE, as the emphasis is on controlling the extent of disruption of the solutions at any level of encoding precision, and not on dynamically changing the encoding precision. The principle of fitness-based disruption may be used in conjunction with ARGOT, Delta Coding, or DPE, and not as an alternative.

The adaptation policy in AGA is different from all the approaches in [1], [6], [9], and [33]. [1] is not related to adapting mutation and crossover rates. AGA is different from [6] 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 [6] 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 [33]. 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.

4.2 Controlled Disruption Genetic Algorithm (CDGA)

Fixed point binary codes have been common in GA research and practice for representing continuous parameters. The values that each parameter can assume are mapped linearly to the \( 2^b \) (\( b \) is the number of bits in the binary code) integer values of the binary codes. The concatenation of the binary codes for each parameter yields the 'genetic' representation of the solution.

In the standard GA, each bit of the string is flipped with a probability \( p_m \) (the mutation rate). Though the mutation rate for each bit is the same, the change in the value of the parameter induced by flipping a bit depends on the position of the bit in the binary code of the parameter. The mutation of the left most bit (most significant bit) in the binary code causes a change in value corresponding to half the total range of the parameter, while the right most bit of the parameter induces a change of \( \frac{1}{2^b} \) of the total range of the parameter. In both cases the solution is said to be 'disrupted,' although the extent of disruption varies considerably. The extent of disruption of the solution may be characterized by the Euclidean distance between the disrupted and disrupted solutions in the search space. We may control the Euclidean distance between the disrupted solution and the undisrupted solution by controlling either the value of \( p_m \) or by restricting mutation to specific bits in the parameter codes.

Crossover also invariably leads to the disruption of the solutions that are crossed. The traditional single-point crossover involves an exchange of the bits of two solutions beyond a randomly chosen point. In general, the crossover point may fall within the binary code of a parameter (alternately it may fall between the codes of two parameters). Consequently, the corresponding parameter code gets disrupted when the solution is crossed, and its value gets modified. As in the case of mutation, the extent of disruption of the parameter depends on whether the crossover point lies closer to the left boundary of the parameter code or the right boundary of the parameter code. A control over the position of the crossover point within the parameter code facilitates a control over the disruption of the parameter.
Function

\[ SF_1 = \sum_{i=1}^{n} z_i^2 \]
\[ SF_2 = 100(e[i] - x_i)^2 + (1 - x_i)^2 \]
\[ SF_3 = \sum_{i=1}^{n} (x_i) \]
\[ SF_4 = \sum_{i=1}^{n} x_i + N(0, 1) \]
\[ SF_5 = 0.003 + \sum_{j=1}^{n} \frac{j}{n+1} \sum_{i=1}^{n} (x_i - x_j)^2 \]
\[ SF_6 = \sum_{i=1}^{n} \frac{e[i]}{e[i]^2 + (1 + e[i] + x_i)^2} \]
\[ SF_7 = 0.5 \times \left[ \frac{1}{e[i] + x_i} \right] \]
\[ SF_8 = (x_i^2 + x_j^2)^{0.50} \left[ \sin^2(50(x_i^2 + x_j^2)^{0.50}) + 1.0 \right] \]

Range of parameters

-5.12 ≤ x_i ≤ 5.12
-5.12 ≤ x_i ≤ 5.12
-2.048 ≤ x_i ≤ 2.048
-1.28 ≤ x_i ≤ 1.28
-65.536 ≤ x_i ≤ 65.536
-1.0 ≤ x_i ≤ 1.0
-100.0 ≤ x_i ≤ 100.0
-100.0 ≤ x_i ≤ 100.0

Number of bits

30
50
24
240
34
50
44
44

Fig. 2. Testbed of functions.

We implement fitness-based disruption in CDGA by:

- explicitly identifying the bits in each parameter code that may be mutated
- by controlling the position of the crossover point within the parameter code

We define a left mutation limit, at the position

\( l_m, 0 < l_m < b \), such that only the bits to the right of the left mutation limit are subjected to mutation with a probability \( p_m \).

Also we define a left crossover limit, at the position

\( l_c, 0 < l_c < b \), the crossover point is restricted to lie to the right of the left crossover limit of a randomly chosen parameter of the solution.

\( l_m \) is determined from the fitness value of the solution as follows:

\[ l_m = o + (b - o) \times \left( \frac{1.0 - f_i - \bar{f}}{f_{\text{max}} - \bar{f}} \right) \text{ if } f_i \leq \bar{f} \]
\[ l_m = 0 \text{ if } f_i > \bar{f} \]

where

\( f_i \) is the fitness of the solution \( i \),
\( f_{\text{max}} \) is the maximum fitness in the population,
\( \bar{f} \) is the average fitness of the population,
\( o \) is an offset.

\( l_c \) is determined from the fitness values of the two solutions to be crossed as follows:

\[ l_c = o + (b - o) \times \left( \frac{1.0 - f' - \bar{f}}{f_{\text{max}} - \bar{f}} \right) \text{ if } f_i \leq \bar{f} \]

where

\( f' \) is the larger of the fitnesses of the two solutions that are crossed.

\( o \) defines the position of the right most point in the binary code that the left crossover limit or left mutation limit may attain. The bits to the right of the oth bit are mutated irrespective of the fitness of the solution. This sustains the local-search capacity of the GA.

For any solution with a fitness less than the average fitness of the population, mutation and crossover occur as in the standard Genetic algorithm. Only solutions with a fitness greater than the average fitness of the population are subjected to controlled disruption. The solution with the maximum fitness in the population may have only the bits to the right of the oth bit mutated.

The next section discusses the experiments, and the performance of CDGA.

4.2.2 Experiments

We have compared the performance of CDGA with that of a standard GA. In both the GAs, we have used similar techniques (i.e., scaling of fitness values, single point crossover) and the same control parameters (population size, \( p_m, p_c \)) to make an unbiased comparison.

We have used a test suite of eight functions for our experiments (see Fig. 2). SF1 - SF8 are the functions in DeJong's test suite. SF6 is a smoothly varying multimodal function over five variables. SF7 and SF8 are highly oscillatory functions that have been used in previous studies [19]. For each function the variables have been shifted to the left by 10% to prevent the optimum from occurring at a Hamming cliff [5].

The control parameters that have been used are those suggested in [19]: \( p_m = 0.005, p_c = 0.65 \); Population size = 30. It has to be noted that our goal is not to maximize the performance of the GA. We only aim at demonstrating the advantage of using the fitness-based disruption technique. As a measure of performance, we have used the number of generations required to find a solution with a specified high fitness, called 'threshold' (e.g., 0.9999).

<table>
<thead>
<tr>
<th>Function</th>
<th>CDGA</th>
<th>SGA</th>
<th>Threshold</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF1</td>
<td>30.5</td>
<td>56.1</td>
<td>0.9999</td>
<td>4</td>
</tr>
<tr>
<td>SF2</td>
<td>61.1</td>
<td>70.5</td>
<td>0.99999</td>
<td>2</td>
</tr>
<tr>
<td>SF3</td>
<td>15.9</td>
<td>20.9</td>
<td>1.00</td>
<td>4</td>
</tr>
<tr>
<td>SF4</td>
<td>112.8</td>
<td>150.6</td>
<td>0.05</td>
<td>4</td>
</tr>
<tr>
<td>SF5</td>
<td>48.2 (3)</td>
<td>40.7(2)</td>
<td>1.00</td>
<td>7</td>
</tr>
<tr>
<td>SF6</td>
<td>43.2</td>
<td>61.40</td>
<td>0.995</td>
<td>5</td>
</tr>
<tr>
<td>SF7</td>
<td>61.7</td>
<td>75.2(2)</td>
<td>0.995</td>
<td>11</td>
</tr>
<tr>
<td>SF8</td>
<td>152.7(1)</td>
<td>195.4(3)</td>
<td>0.995</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 10 gives the experimental results. The number of generations (averaged over 30 runs) required by the CDGA and the SGA for finding a solution with a fitness greater than the threshold are tabulated. The numbers in parentheses indicate the number of runs for which the CDGA or the SGA converged prematurely to suboptimal solutions. The 'threshold' to be achieved, and the offset, \( o \), are indicated for each function. The optimal value of \( o \) was obtained for each function from preliminary experiments.

For all the unimodal functions (SF1 - SF4), the CDGA shows a significantly better performance than the SGA. The CDGA also performs better than the SGA on all the multimodal functions except SF5.
TABLE 11
COMPARISON OF PERFORMANCE OF AGA AND SGA FOR THE NEURAL NETWORK PROBLEMS

<table>
<thead>
<tr>
<th>Func</th>
<th>SGA (Gens.)</th>
<th>AGA (Gens.)</th>
<th>SGA (Stuck)</th>
<th>AGA (Thresh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>72</td>
<td>61.20</td>
<td>36.73</td>
<td>10</td>
</tr>
<tr>
<td>4-bit parity</td>
<td>200</td>
<td>399.33</td>
<td>93.43</td>
<td>18</td>
</tr>
<tr>
<td>Dec. Enc.</td>
<td>920</td>
<td>456.43</td>
<td>71.70</td>
<td>26</td>
</tr>
</tbody>
</table>

The results of Table 10 indicate that the technique of fitness-based disruption improves the exploitation by the GA as indicated by the performance of CDGA on f51 - f54. However, this enhanced performance has been achieved without losing the exploratory power, as indicated by the performance of the CDGA on f56 - f58.

4.3 Adaptive Genetic Algorithm

In CDGA, we have aimed at controlling the disruption of the binary encoded continuous parameters, induced by the crossover and mutation operators. This technique has limited application since all search spaces are not continuous, and as the encodings for continuous variables need not always be on a binary alphabet. A more general way of controlling the disruption of solutions may be obtained by appropriately varying the crossover and mutation probabilities of solutions. This technique proves to be very effective in improving the performance of GAs on a wide variety of problem domains. A detailed discussion of the AGA, its analysis, and implementation may be found in [23]. Here, we just wish to point out the key features of AGA, and its performance on two very complex problems for which we have obtained very impressive results.

As in CDGA, our goal is to ‘protect’ the best solutions, while we employ the solutions with subaverage fitnesses for exploring new regions in the search space. Basically, the crossover rate \( p_c \) and the mutation rate \( p_m \) are determined explicitly for each solution in the population depending on the fitnesses of the solutions. Higher the fitness of a solution, lower are the values of \( p_c \) and \( p_m \) and vice versa. The variation of \( p_c \) and \( p_m \) as a function of the fitness is given by the following equations:

\[
p_c = k_1 \frac{f_{\text{max}} - f'}{f_{\text{max}} - \bar{f}}, \quad f' \geq \bar{f}, \quad (43)
\]

\[
p_c = k_3, \quad f' < \bar{f} \quad (44)
\]

and

\[
p_m = k_2 \frac{f_{\text{max}} - f}{f_{\text{max}} - \bar{f}}, \quad f \geq \bar{f}, \quad (45)
\]

\[
p_m = k_4, \quad f < \bar{f}, \quad (46)
\]

where \( f', f_{\text{max}} \), and \( \bar{f} \) are fitnesses as defined earlier for CDGA. The constants \( k_1, k_2, k_3, k_4 \) all lie in the range 0–1. Extensive experimentation and analysis have led us to choose the following values for these constants: \( k_1 = k_2 = 1.0, \) and \( k_3 = k_4 = 0.5 \). Further, to avoid premature convergence of the population to suboptimal solutions, we have a default mutation rate of 0.005 for all the solutions.

We demonstrate the relative performance improvement of AGA over SGA on three instances of a Neural Network weight optimization problem—XOR, Parity, and Decoder-Encoder—and on seven instances of a Test Generation problem for VLSI circuits. A detailed study of these GA applications may be found in [25] and [26]. Appendix B gives a cursory description of these problems.

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 (43), (44), (45), and (46) given in Section 3.3.

TABLE 12
PERFORMANCE OF AGA AND SGA FOR THE TEST GENERATION PROBLEM

<table>
<thead>
<tr>
<th>Circuit</th>
<th>SGA (Gens.)</th>
<th>AGA (Gens.)</th>
<th>Fault Coverage</th>
<th>Stuck (Len.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>102.10</td>
<td>10.73</td>
<td>99.23 %</td>
<td>36</td>
</tr>
<tr>
<td>c499</td>
<td>10.91</td>
<td>10.50</td>
<td>98.94 %</td>
<td>41</td>
</tr>
<tr>
<td>c850</td>
<td>155.23</td>
<td>37.33</td>
<td>100.00%</td>
<td>60</td>
</tr>
<tr>
<td>c1355</td>
<td>35.26</td>
<td>31.70</td>
<td>99.49%</td>
<td>41</td>
</tr>
<tr>
<td>c1908</td>
<td>122.13</td>
<td>57.93</td>
<td>99.52 %</td>
<td>33</td>
</tr>
<tr>
<td>c3540</td>
<td>155.43</td>
<td>73.66</td>
<td>96.00%</td>
<td>50</td>
</tr>
<tr>
<td>c5315</td>
<td>53.33</td>
<td>21.56</td>
<td>98.89%</td>
<td>178</td>
</tr>
</tbody>
</table>

The experimental results are presented in Tables 11 and 12. Table 11 gives the average number of generations required by each GA for attaining a solution with a fitness value equal to the threshold value ‘Threshold’. Also tabulated is the number 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.

For the three neural network problems, the AGA has located the optimal solution in every trial, while the performance of the SGA has been poor. Table 12 compares the performance of the AGA and the SGA for the Test Generation problem. The numerical in the circuit name indicates the gate count of the circuit. Once again, the superior performance of AGA is clear. For c432, the AGA requires almost ten 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 parasitics and fanouts of gates, the number of levels in the circuit, etc.
5 Conclusions

Genetic search methods are emerging as general purpose and robust search techniques in a wide spectrum of application domains. Their distinguishing feature is their capacity to locate the global optimum in a multimodal search landscape. This paper has introduced the idea of fitness moments for analyzing the functioning of Genetic Algorithms. We have shown that the fitness moments in any generation \( t \) may be evaluated from the fitness moments of the initial population. In particular, the average fitness of any generation may be calculated. The results developed in the paper may be used to predict the number of generations required by the GA to achieve a given average fitness of the population, and consequently a prescribed degree of convergence.

We have derived exact relations between the fitness moments of different generations for the case wherein proportional selection alone characterizes the GA. When considering crossover and mutation in addition to selection, we have approximated the fitness moments of the generated solutions by the fitness moments of the initial population. The predicted results are in close agreement with the experimental observations when a modified expression is used for the disruption factor.

We have introduced the concept of fitness-based disruption to simultaneously improve the exploitative and exploratory power of the GA. Using fitness moments, we provide a simple analysis for the enhanced performance of a GA that incorporates fitness-based disruption. Experimental results from two implementations of GAs incorporating fitness-based disruption—Controlled Disruption Genetic Algorithm (CDGA) and the AGA (Adaptive Genetic Algorithm)—are encouraging.

One of the directions for future work is the development of a preprocessing tool which estimates the fitness moments in any specific generation from the fitness moments of the initial population. Before using the GA for locating the optimal solution, the preprocessing tool may be used to choose the initial population that provides the best values of fitness moments. The preprocessing tool may also be used to arrive at a stopping criterion for the GA. A comparison of the predicted fitness moments for successive generations can be performed, and the GA may be stopped at a point when further improvement in their values becomes insignificant. The preprocessing tool also has an application in the choice of the control parameters for the GA. The performance of the GA for a given parameter set may be gauged from the predicted fitness moments.

CDGA and AGA are improvements on the standard GA. However, it appears that fitness-based disruption can be used in conjunction with other efficient GA implementations such as the CHC GA [8] and GENITOR [33] to improve their performance.

Appendix A

Lemma. \( M^{eq} \geq M^e M^d \)

Proof. Consider

\[
M^{eq} - M^{d} = \frac{1}{N} \sum_{i=1}^{R} n_i (f_i)^{eq} - \left[ \frac{1}{N} \sum_{i=1}^{R} n_i (f_i)^e \right] \left[ \frac{1}{N} \sum_{i=1}^{R} n_i (f_i)^d \right]
\]

The right side of (47) may be rewritten as:

\[
\frac{1}{2N^2} \left( 2N \sum_{i=1}^{R} n_i (f_i)^{eq} - 2 \sum_{i=1}^{R} n_i (f_i)^e \sum_{i=1}^{R} n_i (f_i)^d \right)
\]

Expressing \( N \) as \( \sum_{j=1}^{R} n_j \), and changing the summation indices, (48) may be split into four terms as

\[
\frac{1}{2N^2} \sum_{i=1}^{R} \sum_{j=1}^{R} n_i n_j (f_i)^{eq} - \sum_{i=1}^{R} n_i \sum_{i=1}^{R} n_i (f_i)^{eq} - \sum_{i=1}^{R} \sum_{j=1}^{R} n_i n_j (f_i)^e (f_j)^d
\]

Equivalently (49) may be expressed as

\[
\frac{1}{2N^2} \left( \sum_{i=1}^{R} \sum_{j=1}^{R} n_i n_j (f_i)^e (f_j)^d - \sum_{i=1}^{R} \sum_{j=1}^{R} n_i n_j (f_i)^e (f_j)^e \right)
\]

The term \((f_i)^e (f_j)^d - (f_i)^e (f_j)^e\) is greater than or equal to zero for any \( x \) and \( y \). Consequently \( M^{eq} \geq M^e M^d \).

Appendix B

Neural Network Weight-Optimization Problem: 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 \( l_{ji} \), \( 1 \leq j \leq k \)
- A threshold value \( T_i \)
- A set of interconnection weights \( w_{ji} \), \( 1 \leq j \leq k \)
- An activation value \( A_i = \sum_{j=1}^{k} w_{ji} + T_i \)

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,
\[ \text{MSE} = \frac{1}{pN_o} \sum_{j=1}^{p} \sum_{i=1}^{N_o} \left( O_j - O_j' \right)^2 \]  

(51)

where

- \( \text{MSE} \): the mean square error
- \( p \): number of input patterns
- \( N_o \): number of output neurons
- \( O_j \): output value of the \( j \)th neuron for the \( i \)th input pattern
- \( O_j' \): training value of \( j \)th neuron for the \( i \)th input pattern

In our implementation, the output function \( f \) is sigmoidal: \( f(t) = (1 + \exp(-10t))^{-1} \). 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 binary 1. \( w_{ij} \) and \( T_{ij} \) 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:** two inputs, one output, five neurons, nine weights, four input patterns, the output value is the Exclusive OR of the input bits.
- **4-bit parity:** four inputs, one output, nine neurons, 25 weights, 16 input patterns, the output value is 1 if there are an odd number of 1's among the inputs.
- **Decoder Encoder:** 10 inputs, 10 outputs, 25 neurons, 115 weights, 10 input patterns (each having all 0's 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 0s and 1s 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 [16]. 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 modeled as being stuck-at-0 or stuck-at-1.
- A test for a fault should
  1) generate a logic value at the fault site that is different from the stuck-at-value of the fault,
  2) should be able to propagate 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: \( C_v = \sum_{i \in F} L_{vi} - L_{si} \)
  where
  - \( C_v \): cost associated with a vector \( v \),
  - \( F \): set of undetected faults
  - \( L_{vi} \): maximum number of gate levels in the circuit
  - \( L_{si} \): level to which the fault effect of \( i \) has been propagated 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 [17].

**References**


M. Srinivas graduated from the Indian Institute of Technology, Madras, India, in 1989. From July 1989 to July 1990, he was employed with the Center for Development of Advanced Computing, Bangalore. From August 1990 to July 1993, he was a PhD student at the Department of Computer Science and Automation, Indian Institute of Science, Bangalore. After submitting his PhD thesis, he was employed by Motorola India Electronics Limited. His research interests include theory and design of genetic algorithms, neural networks, stochastic optimization, and optimization in VLSI CAD Algorithms.

L.M. Patnaik obtained his PhD in 1978 in the area of real-time systems and his DSc in 1989 in the areas of computer systems and architectures, both from the Indian Institute of Science, Bangalore, India. Currently, he is a professor with the Electrical Sciences Division at the Indian Institute of Science. He directs his research group in the Microprocessor Applications Laboratory at the institute. During the past two decades of his service at the institute, his teaching, research, and development interests have been in the areas of parallel and distributed computing, computer architecture, CAD of VLSI systems, computer graphics, theoretical computer science, real-time systems, neural computing, and genetic algorithms. In these areas, he has published more than 110 papers in refereed international journals and more than 120 papers in refereed international conference proceedings on the theoretical, software, and hardware aspects of the above areas of computer science and engineering. He is a co-editor of a book on VLSI system design published by Tata McGraw-Hill and the author of a book on functional programming to be published by Springer-Verlag. He has supervised several research theses in the above areas.

In the areas of parallel and distributed computing, and neural computing, he has been a principal investigator for a large number of government-sponsored research projects and a consultant to a few industries. He is a distinguished lecturer of the IEEE Region 10. In recognition of his contributions in the areas of electronics, informatics, telecommunications, and automation, he was presented the Dr. Vikram Sarabhai Research Award in 1989. He was presented the Dr. Ramaiah Wadhwa Award for 1992 by the Institution of Electronics and Telecommunication Engineers for his contributions during the past 10 years in the field of electronics and telecommunication.

Dr. Patnaik is a fellow of the IEE, Indian National Science Academy, Indian Academy of Sciences, National Academy of Sciences, and Indian National Academy of Engineering. He is a senior member of the Computer Society of India; a life member of the VLSI Society of India and Instrument Society of India; a founding member of the executive committee of the Association for the Advancement of Fault-Tolerant and Autonomous Systems; and a fellow of the Institution of Electronics and Telecommunications Engineers and the Institution of Engineers. His name appears in the Eighth Edition of Who's Who in the World, published by Marquis Publications; Reference Asia, Asia's Who's Who of Men and Women of Achievement; and the Directory of Distinguished Computer Professionals of India.

He has been the program chair (1990 and 1991) and general chair (1992) of the IEEE-sponsored International Conference on VLSI Design; member, program committee for the Sixth International Parallel Processing Symposium (1992) and the 22nd Annual International Symposium on Fault Tolerant Computing (1992), both sponsored by the IEEE; program chair (Asia/Australia), IEEE Symposium on Parallel and Distributed Processing, 1993; member, Asian Subcommittee, ACM Multimedia '93 Symposium; and member, Executive Committee and Coordinator, Asia and Pacific Rim 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 and is chairman of the Indian Transputer User Group (ITUG). He is a member of the editorial boards of the International Journal of High Speed Computing, Journal of Computer-Aided Design, VLSI Design, An International Journal of Custom Chip Design, Simulation, and Testing, The Computer Journal, and Parallel Algorithms and Applications; and editor of the Journal of Computer Science and Informatics.