Applied Intelligence 7, 257–267 (1997)
© 1997 Kluwer Academic Publishers. Manufactured in The Netherlands.

Using Disruptive Selection to Maintain Diversity in Genetic Algorithms

TING KUO AND SHU-YUEN HWANG
Department of International Trade, Takming Junior College of Commerce, Institute of Computer Science and Information Engineering, National Chiao Tung University

Abstract. Genetic algorithms are a class of adaptive search techniques based on the principles of population genetics. The metaphor underlying genetic algorithms is that of natural evolution. With their great robustness, genetic algorithms have proven to be a promising technique for many optimization, design, control, and machine learning applications. A novel selection method, *disruptive selection*, has been proposed. This method adopts a *nonmonotonic* fitness function that is quite different from conventional *monotonic* fitness functions. Unlike conventional selection methods, this method favors both superior and inferior individuals. Since genetic algorithms allocate exponentially increasing numbers of trials to the observed better parts of the search space, it is difficult to maintain diversity in genetic algorithms. We show that Disruptive Genetic Algorithms (DGAs) effectively alleviate this problem by first demonstrating that DGAs can be used to solve a nonstationary search problem, where the goal is to track time-varying optima. Conventional Genetic Algorithms (CGAs) using proportional selection fare poorly on nonstationary search problems because of their lack of population diversity after convergence. Experimental results show that DGAs immediately track the optimum after the change of environment. We then describe a spike function that causes CGAs to miss the optimum. Experimental results show that DGAs outperform CGAs in resolving a spike function.

Keywords: genetic algorithm, disruptive selection, diversity, nonstationary search problem, spike function

1. Genetic Algorithms (GAs)

The fundamental theory behind genetic algorithms was presented in Holland's pioneering book [29]. GAs are population-based search techniques that maintain populations of potential solutions during searches. A potential solution usually is represented by a string with a fixed bit-length. In order to evaluate each potential solution, GAs need a payoff (or objective) function that assigns a scalar payoff to any particular solution. Once the representation scheme and evaluation function are determined, a GA can start searching. Initially, often at random, GAs create a certain number (called the population size) of strings to form the first generation. Next, the payoff function is used to evaluate each solution in this first generation. Better solutions obtain higher payoffs. Then, on the basis of these evaluations, some genetic operators are employed to generate the next generation. The procedures of evaluation and generation are iteratively performed until the optimal solution(s) is (are) found or the time alloted for computation ends [19, 23, 29].

The three primary genetic operators focused on by most researchers are selection, crossover, and mutation. They are described below.

1. *Selection* (or *Reproduction*): The population of the next generation is first formed by using a probabilistic reproduction process, of which there are two general types: generational reproduction and steady-state reproduction. Generational reproduction replaces the entire population with a new one in each generation. By contrast, steady-state reproduction [47, 52] replaces only a few individuals in each generation. No matter what type of reproduction is used, individuals with higher fitness usually have

a greater chance of contributing offspring. Several methods may be used to determine the fitness of an individual. Proportional [18, 29] and ranking [2] are the main schemes used in GAs. The resulting population, sometimes called the intermediate population, is then processed using crossover and mutation to form the next generation.

2. *Crossover*: A crossover operator manipulates a pair of individuals (called parents) to produce two new individuals (called offspring) by exchanging segments from the parents' coding. By exchanging information between two parents, the crossover operator provides a powerful exploration capability. A commonly used method for crossover is called one-point crossover. Assume that the individuals are represented as binary strings. In one-point crossover, a point, called the crossover point, is chosen at random and the segments to the right of this point are exchanged. For example, let $x1 = 101010$ and $x2 = 010100$, and suppose that the crossover point is between bits 4 and 5 (where the bits are numbered from left to right starting at 1). Then the children are $y1 = 101000$ and $y2 = 010110$. Several other types of crossover operators have been proposed, such as two-point crossover, multi-point crossover [31], uniform crossover [1, 47], shuffle crossover [15], and partially mapped crossover [17].
3. *Mutation*: By modifying one or more of an existing individual's gene values, mutation creates new individuals to increase variety in a population. The mutation operator ensures that the probability of reaching any point in the search space is never zero.

Genetic algorithms have been applied in many diverse areas, such as function optimization [31], the traveling salesman problem [20, 26], scheduling [8, 48], neural network design [28, 40], system identification [34], vision [5], control [33], and machine learning [14, 24, 32]. Goldberg's book [18] provides a detailed review of these applications.

1.1. Why GAs Work?

A good search technique must have two abilities: *exploration* and *exploitation*. Holland [29] indicated that a GA contains these two properties simultaneously. The notation of *schema* must be introduced first in order to understand how genetic algorithms can direct the search towards high-fitness regions of the search space. A schema is a set of individuals in the search space. In most GAs, individuals are represented by fixed-length binary strings that express a schema as a pattern defined over alphabet {0, 1, *}, and describe a set of binary strings in the search space. In the pattern of a schema, 1's and 0's are referred to as *defined bits*; the number of defined bits is called the *order* of that schema. The distance between the leftmost and rightmost defined bit is referred to as the *defining length* of the schema. For example, the order of of **0*11**1 is 4, and the defining length of **0*11**1 is 6. A bit string x is said to be an *instance* of the schema s if x has exactly the same bit values in exactly the same loci as bits defined in s. For example, 00011 and 00110 are both instances of schema 00*1*, but neither 10011 nor 00000 is an instance of schema 00*1*. Schema refers to the conjecture (hypothesis, explanation) as to why its instances are so good (or bad). A schema can be viewed as a defining hyperplane in the search space. Usually, schema and hyperplane are used interchangeably.

For each selection algorithm, it is clear that

$$\mathrm{tsr}(H(t)) = \sum_{x \in H(t)} \frac{\mathrm{tsr}(x, t)}{m(H(t))}, \tag{1}$$

where H is a hyperplane, $H(t)$ is the set of individuals that are in the population $P(t)$ and are instances of hyperplane H, $m(H(t))$ is the number of individuals in the set $H(t)$, and $\mathrm{tsr}(H(t))$ is the growth rate of the set $H(t)$ without the effects of crossover and mutation. Most well-known selection algorithms use proportional selection, which can be described as

$$\mathrm{tsr}(x, t) = \frac{u(x)}{\bar{u}(t)}, \tag{2}$$

where u is the fitness function and $\bar{u}(t)$ is the average fitness of the population $P(t)$. Thus,

$$\mathrm{tsr}(H(t)) = \sum_{x \in H(t)} \frac{u(x)}{\bar{u}(t) m(H(t))} = \frac{\bar{u}(H(t))}{\bar{u}(t)}, \tag{3}$$

where $\bar{u}(H(t))$ is the average fitness of the set $H(t)$.

The *Schema Theorem* [29, 30], a well-known fundamental GA theorem, explains the power of GAs in terms of how schemata are processed. It provides a lower bound on the change in the sampling rate for a single schema from generation t to generation $t + 1$. Since each individual is an instance of 2^L schemata, GAs implicitly process more schemata than explicitly process individuals. This property is known as *implicit parallelism* and is the best explanation of how GAs work. Before stating the Schema Theorem, it is important to note the assumption that is used to simplify the analysis. It is assumed that crossover within the defining length of a schema is always destructive and that gains from crossover are ignored. That is, the following Schema Theorem is a conservative view.

Schema Theorem. In a genetic algorithm using a proportional selection algorithm, a one-point crossover operator, and a mutation operator; for each hyperplane H represented in $P(t)$ the following holds:

$$M(H(t+1)) \geq M(H(t)) \left(\frac{\bar{u}(H(t))}{\bar{u}(t)}\right) \times \left(1 - \frac{p_c d(H)}{L-1}\right)(1-p_m)^{o(H)}. \quad (4)$$

Here,

- $M(H(t+1))$ is the number of individuals expected to be instances of hyperplane H at time $t+1$ under the genetic algorithm, given that $M(H(t))$ is the expected number of individuals at time t;
- p_c is the crossover rate,
- p_m is the mutation rate,
- $d(H)$ is the defining length of hyperplane H,
- $o(H)$ is the order of hyperplane H, and
- L is the length of each string.

The term $\frac{\bar{u}(H(t))}{\bar{u}(t)}$ denotes the ratio of the observed average fitness of hyperplane H to the current population average. This term dominates the rate of change of $M(H(t))$, subject to the destructive terms $(1 - \frac{p_c d(H)}{L-1})(1-p_m)^{o(H)}$. Clearly, $M(H(t+1))$ increases if $\bar{u}(H(t))$ is above the average fitness of the current population (when the destructive terms are small), and vice versa. The destructive terms denote the effect of breakup of instances of hyperplane H caused by crossover and mutation. The term $(1 - \frac{p_c d(H)}{L-1})(M(H(t))$ specifies an upper bound on the *crossover loss*, the loss of instances of H resulting from crosses that fall within the defining length $d(H)$ of H. The term $(1-p_m)^{o(H)}$ gives the proportion of instances of H that escape a mutation at one of the $o(H)$ defined bits of H. In short, the Schema Theorem predicts changes in the expected number of individuals belonging to a hyperplane between two consecutive generations. Clearly, short, low order, above-average schemata receive exponentially increasing numbers of trials in subsequent generations.

2. Disruptive Selection

Clearly, the Schema Theorem is based on the fitness function rather than the objective function. The fitness function determines the productivity of individuals in a population. In general, a fitness function can be described as

$$u(x) = g(f(x)), \quad (5)$$

where f is the objective function and $u(x)$ is a nonnegative number. The function g is often a linear transformation, such as

$$u(x) = af(x) + b, \quad (6)$$

where a is positive when maximizing f and negative when minimizing f, and b is used to ensure a nonnegative fitness. Two definitions concerning the selection strategy are described below [25].

Definition 1. A selection algorithm is *monotonic* if the following condition is satisfied:

$$\mathrm{tsr}(x_i) \leq \mathrm{tsr}(x_j) \leftrightarrow u(x_i) \leq u(x_j). \quad (7)$$

Definition 2. A fitness function is *monotonic* if the following condition is satisfied:

$$u(x_i) \leq u(x_j) \leftrightarrow \alpha f(x_i) \leq \alpha f(x_j). \quad (8)$$

Here (and hereafter) $\alpha = 1$ when maximizing f and $\alpha = -1$ when minimizing f.

In [36], we proposed a *nonmonotonic* fitness function instead of a *monotonic* fitness function. A nonmonotonic fitness function is one for which Eq. (8) is not satisfied for some individuals in a population.

2.1. *Nonmonotonic Fitness Functions*

All conventional GAs use monotonic fitness functions, even through they do not provide good performance for all types of problems. Nonmonotonic fitness functions can extend the class of GAs. As suggested above, a worse solution also contains information that is useful for biasing the search. This idea is based on the following fact. Depending upon the distribution of function values, the fitness function landscape can be more or less mountainous. It may have many high-value peaks beside steep cliffs that fall to deep low-value gullies. On the other hand, the landscape may be a smoothly rolling one, with low hills and gentle valleys. In the former case, a current worse solution, through the mutation operator, may have a greater chance of "evolving" towards a better future solution. In order to exploit such current worse solutions, we define the following new fitness function.

Definition 3. A fitness function is called a *normalized-by-mean* fitness function if the following condition is satisfied:

$$u(x) = |f(x) - \overline{f}(t)|. \tag{9}$$

Here, $\overline{f}(t)$ is the average value of the objective function f of the individuals in the population $P(t)$. Clearly, the normalized-by-mean fitness function is a type of *nonmonotonic* fitness function. We shall refer to a monotonic selection algorithm that uses the *normalized-by-mean* fitness function as *disruptive selection* [35]. Now, we can give a formal definition of directional selection, stabilizing selection, and disruptive selection as follows.

Definition 4. A selection algorithm is *directional* if it satisfies

$$\text{tsr}(x_i) \leq \text{tsr}(x_j) \leftrightarrow \alpha f(x_i) \leq \alpha f(x_j). \tag{10}$$

Definition 5. A selection algorithm is *stabilizing* if it satisfies

$$\begin{aligned}\text{tsr}(x_i) \leq \text{tsr}(x_j) \leftrightarrow |f(x_i) - \bar{f}(t)| \\ \geq |f(x_j) - \bar{f}(t)|.\end{aligned} \tag{11}$$

Definition 6. A selection algorithm is *disruptive* if it satisfies

$$\begin{aligned}\text{tsr}(x_i) \leq \text{tsr}(x_j) \leftrightarrow |f(x_i) - \bar{f}(t)| \\ \leq |f(x_j) - \bar{f}(t)|.\end{aligned} \tag{12}$$

Next, we shall examine schema processing under the effect of disruptive selection.

Since sampling errors are inevitable, conventional GAs do not perform well in domains that have large variances within schemata. It is difficult to explicitly compute the observed variance of a schema that is represented in a population and then use this observed variance to estimate the real variance of that schema. Hence, we will try to use another statistic, a schema's observed deviation from the mean value of a population, to estimate the real variance of the schema. By using this statistic, we can determine the relationship between two schemata.

Definition 7. H_i is more *remarkable* than H_j in $P(t)$ ($H_i \geq_{R,t} H_j$) if

$$\frac{\sum_{x \in H_i(t)} |f(x) - \overline{f}(t)|}{m(H_i(t))} \geq \frac{\sum_{y \in H_j(t)} |f(y) - \overline{f}(t)|}{m(H_j(t))}. \tag{13}$$

That is, on average, H_i has a larger deviation from $\overline{f}(t)$ than H_j has. Since disruptive selection favors extreme (both superior and inferior) individuals, H_i will receive more trials in subsequent generations. We can now characterize the behavior of a class of genetic algorithms as follows.

Theorem 1. *In any GA that uses a* monotonic *selection algorithm and the normalized-by-mean fitness function, for any pair of hyperplanes H_i, H_j in the population $P(t)$,*

$$H_i \geq_{R,t} H_j \rightarrow tsr(H_i(t)) \geq tsr(H_j(t)). \tag{14}$$

Proof: $H_i \geq_{R,t} H_j$ implies

$$\frac{\sum_{x \in H_i(t)} |f(x) - \bar{f}(t)|}{m(H_i(t))} \geq \frac{\sum_{y \in H_j(t)} |f(y) - \bar{f}(t)|}{m(H_j(t))}.$$

By Eq. (5), we can conclude that $u(H_i(t)) \geq u(H_j(t))$. Thus, by Eq. (3), it is clear that $\text{tsr}(H_i(t)) \geq \text{tsr}(H_j(t))$. □

Hence, using disruptive selection, a GA implicitly allocates more trials to schemata that have a large deviation from the mean value of a population. It is important to note that this extension is still consistent with Holland's schema theorem.

3. Convergence vs. Diversity

Premature convergence—loss of population diversity before optimal or at least satisfactory values have been found—has long been recognized as a serious failure mode for genetic algorithms. Genetic diversity helps a population adapt quickly to changes in the environment, and allows GAs to continue searching for productive niches, and avoid becoming trapped at local optima [43]. In genetic algorithms, it is difficult to maintain diversity because the algorithm allocates exponentially increasing numbers of trials to the observed best parts of the search space. In this section we will show that disruptive selection effectively alleviates this problem. We first demonstrate that DGAs can be used to solve a nonstationary search problem, where the goal is to track time-varying optima. CGAs fare poorly on nonstationary search problems because of their lack of population diversity after convergence. We will then describe a spike function that causes CGAs to miss the optimum. Experimental results show that DGAs outperform CGAs in solving the spike function. A brief survey of related works on the problem of premature convergence is first given.

3.1. Related Works

Several mechanisms have been proposed to deal with premature convergence in genetic algorithms. These include restricting the selection procedure (crowding models), restricting the mating procedure (assortative mating, local mating, incest prevention, etc.), explicitly dividing the population into several subpopulations (parallel GAs), and modifying the fitness function (fitness sharing). These mechanisms are outlined as follows.

1. Crowding models [31]: DeJong [31] proposed a crowding scheme in which new individuals are more likely to replace existing individuals that are similar to themselves based on genotypic similarity. Syswerda and Whiteley [47, 51] added a new individual to the population only if it was not identical to any existing individuals.
2. Fitness sharing [12, 13, 21]: Because Goldberg's *sharing functions* reduces the fitness of individuals by an amount proportional to the number of "similar" individuals in the population, it can be viewed as an indirect mating strategy [21].
3. Assortative mating [6, 7]: The population is prevented from becoming too homogeneous by limiting the hybridization effects of crossover. That is, this method restricts crossover, allowing it to occur only between functionally similar individuals.
4. Local mating [10, 11, 39, 45]: The population is arranged geometrically (e.g., into a grid) and crossover occurs only between individuals that are geographically "near" one another.
5. Incest prevention [16]: Eshelman's *incest prevention* mechanism is a more direct means of preventing genetically similar individuals from mating. Individuals are randomly paired for mating, but are only mated if their Hamming distance is above a certain threshold. The threshold is initially set to the expected average Hamming distance of the initial population, and then is allowed to drop as the population converges.
6. Parallel GAs [9, 22, 27, 42, 49, 50, 53]: The population is explicitly divided into several subpopulations. Each subpopulation evolves independently, but with periodicaly migrations of individuals from one subpopulation to another.

Since the genetic material is primarily determined by the selection phase of the genetic algorithm. Evaluation does not alter the individuals and crossover does not alter the alleles. Thus, the selection phase is responsible for the diversity of the population. When conventional selection methods are used, it is hard to maintain diversity in the population because that short, low-order, above-average schemata will receive exponentially increasing numbers of trials. However, if disruptive selection is used, it tends to preserve diversity somewhat longer because that disruptive selection favors both superior and inferior solutions. In the next two sections we will show the effectiveness of disruptive selection on maintaining population diversity.

3.2. Nonstationary Search Problems

Conventional GAs are known to perform poorly on nonstationary search problems, where the goal is to track time-varying optima [41]. This is because of their lack of population diversity after convergence. Smith and Goldberg [44] examined the effects of adding diploid representation and dominance operators in GAs applied to a nonstationary search problem. In this section, we apply DGAs to solve the same problem [37]. Experimental results show that DGAs immediately track the optimum after a change of environment.

Smith and Goldberg [44] successfully adopted a diploid GA instead of a haploid GA, a typical view of GA, to solve a nonstationary search problem. It has been recognized that in natural genetics, *diploidy* and *dominance* increase the survivability of species in time-varying environments [4]. In natural genetics, haploid chromosomes are found primarily in very simple organisms. More complex organisms often have diploid chromosomes, which contain *twice* the information necessary for specifying the organism's structure. Conflicts that occur between the two halves of a diploid chromosome are resolved by a *dominance relationship,* which, in its simplest form, decides on which of the conflicting genes will eventually be expressed in the organism itself.

Typical GAs use *haploid* representations of potential problem solutions. That is, each individual in the population is a bit string that contains sufficient information to specify a solution to the desired problem. By contrast, a diploid GA uses two bit strings, each of which is sufficient to specify a complete solution, to represent an individual. Two strings are decoded, based on a dominance relationship, into a single *expressed* string whose fitness is evaluated. For example, assume 1 is the *dominant allele* and 0 is the *recessive allele*, the following diploid individual,

101101011
001010110

decodes to the following expressed string:

101111111.

Smith and Goldberg showed that those additions greatly increase the efficacy of GAs in time-varying environments. That increased performance is made possible by *abeyant recessive alleles.* They also showed that those recessive alleles increased population diversity without the disruptive effects of high mutation rates. This diversity allowing the GA to renew its search process as the problem varied over time. How do dominance and diploidy improve a GA's performance on nonstationary search problems? They do so because recessive alleles in a diploid GA preserve population diversity after convergence.

Consider a problem in which some individuals are inferior for some period of time. After this period, these individuals become superior. In such cases, it may be desirable to preserve the originally inferior individuals for some period of time since conditions favorable to them may occur.

3.2.1. Example: The Knapsack Problem. We choose as a test case, the knapsack problem used by Smith and Goldberg [44]. Knapsack problems are a class of common but difficult (NP-complete) problems. The knapsack may correspond to a truck, a ship, or a silicon chip. A variety of industrial problems can be reduced to knapsack problems, including cargo-loading, stock-cutting, project-selection, and budget-control. There are many variations of this problem [38]. Given n objects, each with weight w_i and value v_i, and a knapsack that can holds a maximal weight W. The problem is to find out how to pack the knapsack such that the objects in it have the maximal value among all possible ways the knapsack can be packed. Mathematically, the problem is

$$\max \sum_{i=1}^{n} x_i v_i$$

subject to the total weight constraint

$$\sum_{i=1}^{n} x_i w_i \le W,$$

where the x_i's are variables that can be set to either 0 or 1; the maximal weight W, the v_i's, and the w_i's are given problem parameters; and n is the number of objects available. A 17-object, 0–1 knapsack problem and its optimal solutions associated with different total weight constraints are listed in Table 1. The optimal packings associated with these two constraints were determined by standard methods [46]. They are $\sum_{i=1}^{17} x_i v_i = 71$ when $W = 60$ and $\sum_{i=1}^{17} x_i v_i = 87$ when $W = 104$.

We used a binary string of length of 17, where the ith locus represents the ith variable x_i, to code a potential solution x. The total weight constraint was enforced by a penalty function during the GA evaluation phase. The evaluation function is defined as follows.

$$f(x) = \begin{cases} \sum_{i=1}^{n} x_i v_i & \text{if } \sum_{i=1}^{n} x_i w_i \le W \\ \sum_{i=1}^{n} x_i v_i \\ \quad -20(\sum_{i=1}^{n} x_i w_i - W)^2 & \text{if } \sum_{i=1}^{n} x_i w_i > W. \end{cases} \tag{15}$$

In order to simulate a nonstationary search problem, the limited total weight W was switched from 60 to 104.

We set the GA parameters to the parameters Smith and Goldberg used. The population size was 150, the crossover rate was $p_c = 0.75$, and the mutation rate was $p_m = 0.001$. We replicated twenty runs of this problem. Table 2 lists the experimental results of applying GAs to solving a nonstationary 0–1 knapsack problem.

The figures in Table 2 indicate the number of runs that reach the optimum. CGAs converged to the optimum in one of the two switching conditions, but failed to have sufficient population diversity to continue searching when the total weight constraint is changed. DGAs consistently re-discovered the optimum when the limited total weight W is switched from 60 to 104. Note that Smith and Goldberg [44] also showed very good performance for the nonstationary knapsack problem with diploid GA. The optimal is consistently

Table 1. A 17-object, 0–1 knapsack problem with optimal solutions.

Object i	Value v_i	Weight w_i	Opt. $x_i(W = 60)$	Opt. $x_i(W = 104)$
1	2	12	0	0
2	3	5	1	1
3	9	20	0	1
4	2	1	1	1
5	4	5	1	1
6	4	3	1	1
7	2	10	0	0
8	7	6	1	1
9	8	8	1	1
10	10	7	1	1
11	3	4	1	1
12	6	12	1	1
13	5	3	1	1
14	5	3	1	1
15	7	20	0	1
16	8	1	1	1
17	6	2	1	1
Total:	91	122	13	15
			$\sum_{i=1}^{17} x_i v_i = 71$	$\sum_{i=1}^{17} x_i v_i = 87$
			$\sum_{i=1}^{17} x_i w_i = 60$	$\sum_{i=1}^{17} x_i w_i = 100$

Table 2. Number of successful runs out of 20 runs of solving a nonstationary problem.

	$W = 60$	$W = 104$
CGAs	8	1
DGAs	14	19

rediscovered when the weight constraint is switched every 15 generations between 104 and 60. In our study, the weight constraint is switched every 30 generations between 104 and 60.

3.3. Spike Function

To further examine the effectiveness of disruptive selection at maintaining population diversity, we conducted another experiment aimed at solving for a spike function. Spike functions causes CGAs to miss the optimum [3]. This function has a gentle slope over more than 99.2% of the search space. A steep, low-value spike exists in the remaining region.

For example, bisect a 24-bit string into two 12-bit integers a and b. Let $x = a + (b/2048)$, thus x in the range [0, 4097). If x is in [16, 32) then set $f(x)$ equal to $33 - x$ (i.e., in the range (1, 17]) else set $f(x)$ equal to $31 + (x/2048)$ (i.e., in the range [31, 34)). It can be seen that the probability that $f(x)$ falls inside the spike is only 1/256 and the probability that $f(x)$ falls outside the spike is 255/256. The goal is to minimize the function.

We replicated ten runs of this function for each combination of the following parameter settings: $p_c = 0.35, 0.65, 0.95$ and $p_m = 0.01, 0.001$. Here, p_c and p_m represent the crossover rate and the mutation rate, respectively. Each run searched 200 generations with the best solution recorded at each generation. In all cases a population size of 50 was used. Figures 1 and 2 show that DGAs outperformed CGAs.

The performance was measured by averaging the best solutions of ten runs at every 20 generations. Note that the values are divided by a factor of 10. For $p_m = 0.01$, DGAs found the optimal solution 1.00 in all runs. On the other hand, CGAs always found solutions in the range [31, 34). For $p_m = 0.001$, the performance was not as good as that for $p_m = 0.01$ whereas old methods still found solutions in the range [31, 34). It is worth noting that in [3], Baker applied several

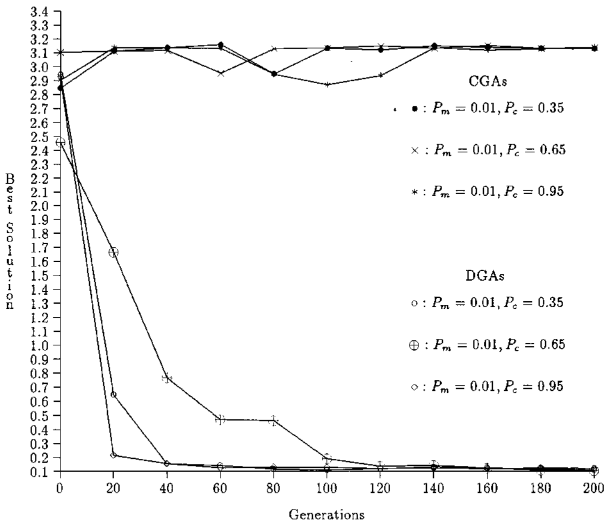

Figure 1. Average of best solutions of ten runs at every 20 generations ($p_m = 0.01$).

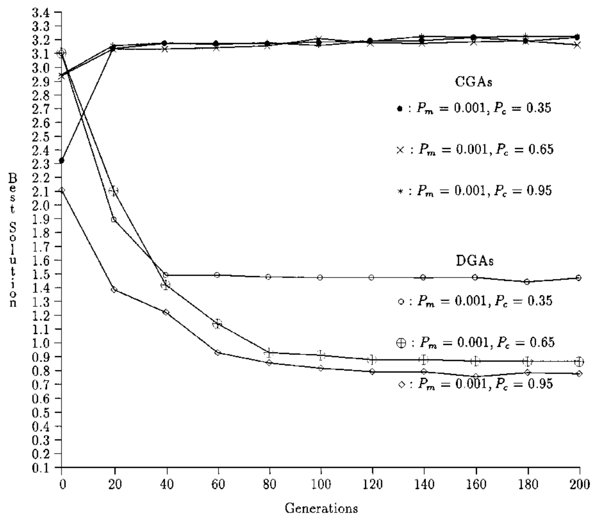

Figure 2. Average of best solutions of ten runs at every 20 generations ($p_m = 0.001$).

non-proportional selection algorithms (e.g., Ranking, and Static Deletion) to this spike function and also achieved the optimum. In [3], the various runtime parameter settings are the same as we used except that they replicated four runs of this function with $p_c = 0.6$ and $p_m = 0.001$.

4. Discussion and Conclusions

Genetic algorithms apply natural selection to mimic evolution. While natural selection may optimize *organisms*, it does not optimize *optimization*. Since all conventional GAs use a monotonic fitness function and apply the "survival-of-the-fittest" principle to reproduce new populations, they can be viewed as processes of evolution based on directional selection. In [36], we proposed a type of disruptive selection that uses a nonmonotonic fitness function. The major difference between disruptive selection and directional selection is that disruptive selection devotes more trials to both better solutions and worse solutions than it does to moderate solutions, whereas directional selection allocates their attention according to the performance of each individual. Since disruptive selection favors both superior and inferior individuals, DGAs will very likely perform well on problems easily solved by CGAs.

The purpose of our approach is to enlarge the domains that GAs work. Disruptive selection does not, of course, outperform conventional selection methods in all kinds of problems. However, disruptive selection is well suited for some kinds of problems and can serve as a supplement to conventional selection methods in solving problems that are hard for conventional GAs to optimize. The experimental results reported in [35] show that GAs using the proposed method easily find the optimum of a function that is non-deceptive but GA-hard. Since sampling errors are inevitable, conventional GAs do not perform well with functions that have large variances within schemata. Using disruptive selection, GAs implicitly allocate more trials to schemata that have a large deviation from the mean value of the current population. This allocation strategy implicitly allocates more trials to schemata that have large variances. Experimental results reported in [35] also show that DGAs find the optima of a deceptive function more quickly and reliably than CGAs do. This could be because the global optima of a deceptive function are surrounded by worst solutions and local optima are surrounded by better solutions. Since disruptive selection also favors inferior individuals, DGAs are immune to traps.

CGAs are known to perform poorly on nonstationary search problems, where the goal is to track time-varying optima. This is because the lack of population diversity after convergence causes CGAs to fare poorly on nonstationary search problems. Smith and Goldberg examined the effects of using dipolid representation and dominance operator to solve a nonstationary search problem. However, their approach needed more bits to represent a solution and needed another decoding mechanism to evaluate a solution. By contrast, our approach can be used to solve a nonstationary search problem by only using a different fitness function. Experimental results show that DGAs immediately track the optimum after the change of environment. Experimental results also show that DGAs outperform CGAs in resolving a spike function that causes CGAs to miss the optimum.

In practice, for solving GA-hard problems, we can implement a parallel GA in which directional, disruptive, and even stabilizing selection can be used in different nodes and migration of good solutions occurs between different nodes periodically. Thus, as a supplement to directional selection, disruptive and stabilizing selection promises to be helpful in solving problems that are hard for conventional GAs to optimize.

References

1. D.H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*, Kluwer Academic Publishers: Boston, MA, 1987.
2. J.E. Baker, “Adaptive selection methods for genetic algorithms,” in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, edited by J.J. Grefenstette, Lawrence Erlbaum Associates: Hillsdale, NJ, July 1985, pp. 101–111.
3. J.E. Baker, “An analysis of the effects of selection in genetic algorithms,” Ph.D. thesis, Vanderbilt University, Nashville, 1989.
4. R.J. Berry, *Genetics*, English University Press: London, 1965.
5. B. Bhanu, S. Lee, and J. Ming, “Self-optimizing image segmentation system using a genetic algorithm,” in *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Applications*, edited by R.K. Belew and L.B. Booker, Morgan Kaufmann: San Mateo, CA, July 1991, pp. 362–369.
6. L.B. Booker, “Intelligent behavior as an adaptation to the task environment,” Ph.D. thesis, Univ. of Michigan, 1982.
7. L.B. Booker, “Triggered rule discovery in classifier systems,” in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 265–274.
8. G.A. Cleveland and S.F. Smith, “Using genetic algorithms to schedule flow shop releases,” in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 160–169.
9. J.P. Cohoon, S.U. Hegde, W.N. Martin, and D.S. Richards, “Punctuated equilibria: A parallel genetic algorithm,” in *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, edited by J.J. Grefenstette, Lawrence Erlbaum Associates: Hillsdale, NJ, July 1987, pp. 148–154.
10. R.J. Collins and D.R. Jefferson, “Selection in massively parallel genetic algorithms,” in *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Applications*, edited by R.K. Belew and L.B. Booker, Morgan Kaufmann: San Mateo, CA, July 1991, pp. 249–256.
11. Y. Davidor, “A naturally occurring niche and species phenomenon: The model and first results,” in *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Applications*, edited by R.K. Belew, Morgan Kaufmann: San Mateo, CA, July 1991, pp. 257–263.
12. K. Deb, “Genetic algorithms in multimodal function optimization,” Technical Report, TCGA Report No. 89002, University of Alabama, 1989.
13. K. Deb and D.E. Goldberg, “An investigation of niche and species formation in genetic function optimization,” in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 42–50.
14. M. Dorigo and U. Schnepf, “Genetic-based machine learning and behavior-based robotics: A new synthesis,” *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-23, no. 1, pp. 141–154, 1993.
15. L.J. Eshelman, R.A. Caruana, and J.D. Schaffer, “Biases in the crossover landscape,” in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 10–19.
16. L.J. Eshelman and J.D. Schaffer, “Preventing premature convergence in genetic algorithms by preventing incest,” in *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Applications*, edited by R.K. Belew and L.B. Booker, Morgan Kaumann: San Mateo, CA, July 1991, pp. 115–122.
17. D.E. Goldberg, “Genetic algorithms and rules learning in dynamic system control,” in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, edited by J.J. Grefenstette, Lawrence Erlbaum Associates: Hillsdale, NJ, July, 1985, pp. 8–15.
18. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley: Reading, MA, 1989.
19. D.E. Goldberg, “Sizing populations for serial and parallel genetic algorithms,” in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J. David Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 70–79.
20. D.E. Goldberg and R. Lingle, Jr., “Alleles, loci, and the traveling salesman problem,” in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, edited by J.J. Grefenstette, Lawrence Erlbaum Associates: Hillsdale, NJ, July 1985, pp. 154–159.
21. D.E. Goldberg and J. Richardson, “Genetic algorithms with sharing for multimodal function optimization,” in *Proceedings of the Second International Conference on Genetic Algorithms*, edited by J.J. Grefenstette, Lawrence Erlbaum Associates: Hillsdale, NJ, July 1987, pp. 41–49.
22. M. Gorges-Schleuter, “Asparagos an asynchronous parallel genetic optimization strategy,” in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 422–427.
23. J.J. Grefenstette, “Optimization of control parameters for genetic algorithms,” *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-16, no. 1, pp. 122–128, 1986.
24. J.J. Grefenstette, “Credit assignment in rule discovery systems based on genetic algorithms,” *Machine Learning*, vol. 3, no. 2/3, pp. 225–245, 1988.
25. J.J. Grefenstette and J.E. Baker, “How genetic algorithms work: A critical look at implicit parallelism,” in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 20–27.

26. J.J. Grefenstette, R. Gopal, B.J. Rosmaita, and D.V. Gucht, "Genetic algorithms for the traveling salesman problem," in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, edited by J.J. Grefenstette, Lawrence Erlbaum Associates: Hillsdale, NJ, July 1985, pp. 160–168.
27. P.B. Grosso, "Computer simulation of genetic adaptation: Parallel subcomponent interaction in a multilocus model," Ph.D. thesis, Univ. of Michigan, 1985.
28. S.A. Harp, T. Samad, and A. Guha, "Towards the genetic synthesis of neural networks," in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 360–369.
29. J.H. Holland, *Adaptation in Natural and Artificial System*, The University of Michigan Press: Ann Arbor, MI, 1975.
30. J.H. Holland, "Searching nonlinear functions for high values," *Applied Mathematics and Computation*, vol. 32, pp. 255–274, 1989.
31. K.A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. thesis, Univ. of Michigan, 1975.
32. K.A. De Jong, "Learning with genetic algorithms: An overview." *Machine Learning*, vol. 3, no. 2/3, pp. 121–138, 1988.
33. C.L. Karr, "Design of an adaptive fuzzy logic controller using a genetic algorithm," *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Applications*, Morgan Kaufmann: San Mateo, CA, July 1991, pp. 450–457.
34. K. Kristinsson and G.A. Dumont, "System identification and control using genetic algorithms," *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-22, no. 5, pp. 1033–1046, 1992.
35. T. Kuo and S.Y. Hwang, "A genetic algorithm with disruptive selection," *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-26, no. 2, pp. 299–307, 1996.
36. T. Kuo and S.Y. Hwang, "A genetic algorithm with disruptive selection," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by S. Forrest, Morgan Kaufmann: San Mateo, CA, July 1993, pp. 65–69.
37. T. Kuo and S.Y. Hwang, "A study on diversity and convergence in distruptive genetic algorithms," in *Proceedings of 1994 International Computer Symposium*, National Chiao Tung University, Hsinchu, Taiwan, Republic of China, Dec. 1994, pp. 145–150.
38. U. Manber, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley: Reading, MA, 1989.
39. H. Mühlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization," in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J. David Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 416–421.
40. G.F. Miller, P.M. Todd, and S.U. Hegde, "Designing neural networks using genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 379–384.
41. E. Pettit and K.M. Swigger, "An analysis of genetic-based pattern tracking and cognitive-based component tracking models of adaptation," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 327–332, 1983.
42. C.B. Petty, M.R. Leuze, and J.J. Grefenstette, "A parallel genetic algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, edited by J.J. Grefenstette, Lawrence Erlbaum Associates: Hillsdale, NJ, July 1987, pp. 155–161.
43. R.E. Smith, S. Forrest, and A.S. Perelson, "Searching for diverse, cooperative population with genetic algorithms," *Evolutionary Computation*, vol. 1, no. 2, pp. 127–149, 1993.
44. R.E. Smith and D.E. Goldberg, "Diploidy and dominance in artificial genetic search," *Complex Systems*, vol. 6, no. 3, pp. 251–285, 1992.
45. P. Spiessens and B. Manderick, "A massively parallel genetic algorithm: Implementation and first results," in *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Applications*, edited by R.K. Belew and L.B. Booker, Morgan Kaufmann: San Mateo, CA, July 1991, pp. 279–286.
46. M.M. Syslo, N. Deo, and J.S. Kowalik, *Discrete Optimization Algorithms with Pascal Programs*, Prentice-Hall: Englewood Cliffs, NJ, 1983.
47. G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 2–9.
48. G. Syswerda and J. Palmucci, "The application of genetic algorithms to resource scheduling," in *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Applications*, edited by R.K. Belew and L.B. Booker, Morgan Kaufmann: San Mateo, CA, July 1991, pp. 502–508.
49. R. Tanese, "Distributed genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 434–440.
50. R. Tanese, "Distributed genetic algorithms for function optimizaiton," Ph.D. thesis, Univ. of Michigan, 1989.
51. D. Whitley, "The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," in *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, edited by J.D. Schaffer, Morgan Kaufmann: San Mateo, CA, June 1989, pp. 116–123.
52. D. Whitley and J. Kauth, "Genitor: A different algorithm," in *Proc. of Rocky Mountain Conference on Artificial Intelligence*, 1988, pp. 118–130.
53. D. Whitley and T. Starkweather, "Genitor II: A distributed genetic algorithm," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 2, no. 3, pp. 189–214, 1990.

Ting Kuo received the B.S. degree in Industrial Engineering from National Tsing Hua University, in 1979. From 1979 to 1981, he served in the Chinese Army as a logistics officer. From 1981 to 1982, he worked at Shih-Lin Dyeing & Weaving Co., Ltd. as an Industrial Engineer. From 1983 to 1991, he worked at the Technical Research Division of the Institute for Information Industry, Taipei, Taiwan. He received the M.S. and Ph.D. degrees in Computer Science and Information Engineering from National Chiao-Tung University in 1990 and 1995, respectively. His current research interests include Genetic Algorithms, Artificial Intelligence, and Scheduling.

Shu-Yuen Hwang received the B.S. and M.S. degrees in electrical engineering from National Taiwan University in 1981 and 1983, respectively, and the M.S. and Ph.D. degrees in computer science from the University of Washington in 1987 and 1989, respectively. During 1989–1995, he was appointed as Associate Professor of Department of Computer Science and Information Engineering, National Chiao-Tung University, and was Director of CSIE during 1993–1995. Since 1995, he has been appointed as Full Professor of CSIE. His research interests include computer vision, artificial intelligence, computer simulation, and mobil computing.