# A Sequential Niche Technique for Multimodal Function Optimization

David Beasley*
Department of Computing Mathematics,
University of Wales, College of Cardiff, Cardiff, CF2 4YN, UK

David R. Bull†
Department of Electrical and Electronic Engineering,
University of Bristol, Bristol, BS8 1TR, UK

Ralph R. Martin‡
Department of Computing Mathematics,
University of Wales, College of Cardiff, Cardiff, CF2 4YN, UK


Technical Report No. 93001, February 1993§


**Abstract**

A technique is described which allows unimodal function optimization methods to be extended to efficiently locate *all* optima of multimodal problems. We describe an algorithm based on a traditional genetic algorithm (GA). This involves iterating the GA, but uses knowledge gained during one iteration to avoid re-searching, on subsequent iterations, regions of problem space where solutions have already been found. This is achieved by applying a fitness derating function to the raw fitness function, so that fitness values are depressed in the regions of the problem space where solutions have already been found. Consequently, the likelihood of discovering a *new* solution on each iteration is dramatically increased. The technique may be used with various styles of GA, or with other optimization methods, such as simulated annealing. The effectiveness of the algorithm is demonstrated on a number of multimodal test functions. The technique is at least as fast as fitness sharing methods. It provides a speedup of between 1 and $10p$ on a problem with $p$ optima, depending on the value of $p$ and the convergence time complexity.

Keywords: genetic algorithm, fitness sharing, fitness derating function, multimodal optimization, simulated annealing.

## 1 Introduction

A great deal of research has gone into finding efficient methods for locating the single, global optimum of a function. (This may be a maximum or a minimum; for simplicity, and without loss of generality, we shall assume that maximization is the aim.) In various tasks, however, a function may have several maxima which we wish to locate.

For example, in the field of mechanical design, we may have a fitness function which rates highly those designs which would perform well. There may be several quite different designs which would perform equally well, but some of these designs may be better than others in terms of ease of manufacture, ease of maintenance, reliability, etc. Usually it is difficult to find a satisfactory way to combine all these factors into a *single* fitness score, so we must resort to *multi-objective* optimization, in which the fitness function returns several scores, each relating to one of the attributes to be optimized. This adds greatly to the complexity of the search process. An alternative scheme is to use only the most important criteria in constructing the fitness function, and have

§Later published with the same title by MIT Press in *Evolutionary Computation* **1**(2), (1993), pp101–125
*email: David.Beasley@cm.cf.ac.uk
†email: Dave.Bull@bristol.ac.uk
‡email: Ralph.Martin@cm.cf.ac.uk

the GA generate several alternative solutions. The human designer can then use his judgement to select the best design from those generated.

Another example is the location of resonance points in a mechanical or electrical system (Deb & Goldberg, 1990). If the fitness function gives the resonant amplitude of the system under particular conditions, we may be interested in locating all resonant frequencies with amplitudes above a particular threshold, not simply the frequency of greatest resonance. Frequencies of large resonance need to be identified because the designer generally wishes to minimize or maximize all such resonances, depending on the application.

In the genetic algorithm (GA) field, only a modest amount of research has been done on the topic of locating all maxima of a multimodal function. This paper presents a new iterative technique based on solving a multimodal problem one peak at a time. This technique may also be of use where only a single, global maximum is required, but the presence of sub-optimal peaks makes the global optimum difficult to find. The technique is demonstrated in the context of a traditional GA. However, it is essentially an add-on technique, and can be used to extend any other optimization method based purely on a fitness function, such as simulated annealing.

We assume that we wish to locate all the maxima of a multimodal function whose height exceeds a certain *threshold*. Such maxima we refer to as the *maxima of interest*. As in earlier work on fitness sharing by Goldberg & Richardson (1987), Deb (1989), and Deb and Goldberg (1989), we assume (a) that we know, or can estimate, how many maxima of interest the function has, and (b) that these maxima are spread approximately evenly throughout the search space. Nevertheless, ideas for overcoming these limitations are discussed in Section 5.3. We do not assume that all maxima are of the same height.

Below we review existing techniques for finding multiple solutions in function optimization.

## 1.1 Iteration

Iteration is the simplest technique which can be used with any optimization method to produce multiple solutions. If the optimization method incorporates stochastic behaviour, then there is a likelihood that successive runs will come up with different solutions. But this is rather an unintelligent approach. However many times the algorithm is iterated, there is no guarantee that it will locate all maxima of interest.

Applying iteration to any single-solution technique, the very best we can hope for is that if there are $p$ maxima, we will have to iterate $p$ times. Using blind iteration, to compensate for the duplicate solutions which are likely to arise, we can expect to have to increase the number of iterations by a factor, which we shall call the *redundancy factor*, $R$. If all maxima have an equal likelihood of being found, it can be shown that $R$ has a value given by:

$$R = \sum_{m=1}^{p} \frac{1}{m} \tag{1}$$

This can be approximated (Jolley, 1961) for $p > 3$, by:

$$R \approx \gamma + \log p \tag{2}$$

where $\gamma \approx 0.577$ (Euler's constant).

This value is relatively small—even for $p = 1000$, $R$ is only 7.5. However, if all maxima are *not* equally likely to be found, the value of $R$ will be much higher.

## 1.2 Parallel sub-populations

Another technique which can yield multiple solutions is a GA where the population is divided into multiple sub-populations which evolve in parallel. If there is no communication between sub-populations, this is directly equivalent to iterating a single smaller population many times. Similarly, there is no guarantee that different sub-populations will converge on different maxima. Most implementations utilize some degree of communication between the sub-populations, to allow good genes to spread. However, the effect of this is to reduce the diversity of solutions, and the whole population will eventually converge on a single solution (Grosso, 1985). Local mating schemes suffer similar problems (Davidor, 1991).

## 1.3 Fitness sharing

Goldberg and Richardson (1987) describe the idea of fitness sharing in a GA as a way of promoting stable sub-populations, or *species*.[1]

[1] Earlier work aimed at promoting speciation exists. However, in most cases speciation was employed as a method for maintaining diversity, rather than finding multiple solutions.

The idea of sharing comes from an analogy with nature. In natural ecosystems, there are many different ways in which animals may survive (grazing, hunting, on the ground, in trees, etc.) and different species evolve to fill each role. Each role is referred to as an ecological *niche*. For each niche the physical resources are finite, and must be shared among the population of that niche. This provides a disincentive for all creatures to occupy a single niche. Instead, creatures evolve to form sub-populations in different niches. The size of each sub-population will reflect the availability of resources in the niche.

The analogy in function optimization is that the location of each maximum represents a niche, and by suitably sharing the fitness associated with each niche, we can encourage the formation of stable sub-populations at each maximum. Sharing is carried out on the basis that the fitness "payout" within each niche is finite, and must be shared between all individuals in that niche. Consequently, the fitness awarded to an individual will be inversely proportional to the number of other individuals in the same niche. The total payout for a niche is set equal to the height of the maximum, so larger maxima can support proportionally larger sub-populations.

However, a fundamental problem is: *where are the niches?* How do we decide if two individuals are in the same niche, and should therefore have their fitness values shared? To do this accurately, we need to know *where* each niche is, and how big it is. Obviously we cannot know where the niches are in advance, otherwise we would not have a problem to solve! Goldberg and Richardson (1987) approach this by assuming that if two individuals are close together, within a distance known as the *niche radius*, then their fitness values must be shared. Closeness, in this instance, is measured by the distance in a single-dimension decoded parameter space. (A method for choosing the niche radius is presented in (Deb & Goldberg, 1989), and this is discussed later.)

Although their technique is successful, its disadvantages have been pointed out by Smith, Forrest and Perelson (1992). To compute accurately the fitness of an individual involves calculating its distance from every other member of the population. The total time complexity will depend on the time taken for the basic GA, plus the additional time taken to perform the fitness sharing calculations. This *additional* complexity is $O(N^2)$ (where $N$ is the population size). This is a disadvantage if $N$ is large, and $N$ *must* be large if we hope to find many optima. Goldberg (1989b) recommends that for a multimodal GA expected to find $p$ different solutions, we need a population $p$ times larger than is necessary for a unimodal GA. This is because a certain number of individuals are necessary to accurately locate and explore each maximum. If a GA is to support stable sub-populations at many maxima, the total population size must be proportional to the number of such maxima.

The above argument assumes that the population is spread equally among all maxima of interest. However, this may not be the case for two reasons. Firstly, under the standard fitness sharing scheme, high fitness peaks will contain proportionally more individuals than low peaks. Secondly, there may be peaks in the fitness function which we are *not* interested in, but which capture a proportion of the population. This was a significant problem for Goldberg, Deb and Horn (1992). This uneven distribution requires that $N$ is increased, to ensure that even the smallest peak of interest contains a sufficiently large sub-population. The factor by which $N$ must be increased we call the *peak ratio*, $\phi$. If $\rho$ is the proportion of peaks which are of interest, and $f_{min}$ is the fitness of the smallest peak of interest, then $\phi$ is determined as:

$$\phi = \frac{\text{average fitness of all peaks}}{\rho f_{min}} \tag{3}$$

So the population size required is $N = np\phi$, where $n$ is the population size needed to find one solution.[2]

Oei, Goldberg and Chang (1991) suggest several improvements to the original fitness sharing algorithm. They say that the time complexity of the fitness sharing calculations may be reduced from $O(N^2)$ to $O(Np)$ if we *sample* the population, instead of computing the distance to every other member. They also describe how, in conjunction with tournament selection, a *niche-size threshold* parameter, may be used to limit the maximum number of individuals in each niche. This should prevent highly-fit maxima from gaining significantly more individuals than less-fit maxima. So the effective value of $\phi$ will be between 1 (if the uninteresting peaks have low fitness relative to $f_{min}$) and $1/\rho$ (if the uninteresting peaks have fitnesses close to $f_{min}$). Assuming that these suggestions are viable, we can compute the time complexity as follows:–

The time taken for the basic GA to find a single peak is proportional to the number of function evaluations performed before convergence; this is approximately $(\alpha N g_c)$, where $\alpha$ is the time for one function evaluation, and $g_c$ is the number of generations to convergence. The additional time for the fitness sharing is approximately $(\beta A N p g_c)$, where $\beta$ is the time to compute the distance from one individual to another, and $A$ is the number of individuals we must sample in each niche. Using $A = 5$ (Oei *et al*, 1991), and setting $N = n\phi p$, the time

[2] Methods of choosing suitable values for $n$ are dealt with by Goldberg (1989b). Broadly, the optimum value for $n$ depends on chromosome length.

complexity, $\mathcal{C}_{share}$, is

$$\mathcal{C}_{share} = O(n\phi p g_c(\alpha + 5\beta p)) \tag{4}$$

In Section 2.5 we compare this value with one we derive for our sequential niche algorithm.

### 1.4 Other methods

Glover (1989) describes a method known as *tabu search*, which may be used in the field of combinatorial optimization. The method employs steepest-ascent hillclimbing on a single trial solution, and keeps a record of recently-made moves. These are used to prevent the trial solution re-visiting previously explored areas. This allows progress to continue even when a local optimum has been reached. Glover reports success for a number of combinatorial optimization problems, but it is not clear whether the same technique would work well on function optimization. The sequential niche technique is conceptually similar, but instead of creating a prohibited (tabu) *path* to prevent backtracking, prohibited *regions* are created instead, centered around previously located local and global optima.

Smith *et al* (1992) describe a technique which can promote niche formation in a classifier system GA. They demonstrate that it has a similar effect to fitness sharing, but avoids the limitations that the number of peaks must be known, that the peaks must be evenly spread, and that there is an $O(N^2)$ time complexity component. Unfortunately, their technique is only applicable to classifier systems, and not to multimodal function optimization.

### 1.5 Summary

We have outlined previous work in this area, the most important being the idea of fitness sharing. In Section 2 we will describe some of the ideas which led to the development of the new technique, and we give the details of the algorithm and how it is implemented. Sections 3 and 4 describe the test functions we tried, and the results obtained. Finally, Section 5 discusses the results, and suggests avenues for future work.

## 2 The sequential niche technique

### 2.1 Basic Principles

Ackley (1987) compared a number of function optimization techniques, including GAs and hillclimbers. When an optimization technique located a sub-optimum peak, he iterated the technique until the (known) global maximum had been found. He concluded that to solve difficult problems, (with local maxima), we must search until we locate one of the maxima, and then, "with lessons learned, try again." But blind iteration does *not* learn lessons. When we iterate a conventional GA, everything learned in the previous run is forgotten. The sequential niche technique described here is based on the idea of carrying over knowledge learned in one run to each subsequent run.

Once one maximum has been found, there is no need to search for it again. So our approach is to eliminate that peak from the fitness function, using a method akin to sharing. As mentioned above, sharing is complicated to perform if the locations of the niches are unknown. But after one run of the GA, the location of one of the niches *is* known. On the second run of the GA, we assume that this niche is conceptually already filled, and few further rewards are available to any individuals which might stray into the area. Instead, individuals are forced to converge on an unoccupied niche, which is in turn also conceptually assumed filled in the third run. This process can continue until we decide (using some criterion, such as the number of maxima we expect in the function) that all maxima have been located.

This algorithm is similar to sharing in its approach. The sharing algorithm essentially works by dynamically modifying the fitness landscape according to the location, in each generation, of individuals in the population. This is done in such a way as to encourage the dispersion of individuals throughout the landscape, which, hopefully, leads to better exploration of the search space, and the identification of all maxima. Conversely, our sequential niche algorithm works by modifying the fitness landscape according to the location of *solutions* found in previous iterations (unlike the sharing algorithm, the fitness landscape remains static during each run). This is done in such a way as to discourage individuals from re-exploring areas where solutions have been found, thereby encouraging them to locate a maximum elsewhere.

In the sharing algorithm, once the population begins to converge on multiple peaks, cross breeding between different peaks is likely to be unproductive. Deb (1989) showed that a mating restriction scheme was able to improve the sharing algorithm for this reason. If mating between individuals on different peaks is to be prohibited, performance would be improved by instead running a number of separate, smaller population GAs, one exploring each peak. It might be possible to achieve this with a parallel GA, where the sub-populations on each processor evolve separately *after* an initial period in which they evolve together, using some algorithm which ensures that no two sub-populations are converging on the same peak. However, we do not adopt this approach because it appears unnecessarily complicated.

## 2.2 The basic algorithm

Any problem to be solved by a GA needs a fitness function. For the sequential niche algorithm we also need a *distance metric*. This is simply a function which, given two chromosomes, returns a value related to how "close" they are to each other. The most trivial measure of distance is the Hamming distance, although as Deb and Goldberg (1989) showed, sharing works better if we use a distance measure based on the decoded parameter space. One reason for this is that—for binary-coded chromosomes at least—large differences between chromosomes may correspond to small differences in parameter space (at a Hamming cliff, for example). By using the parameter space to determine the distance between two chromosomes, we avoid any topological distortion introduced by the coding scheme.

Typically, a chromosome (the genotype) represents a number of parameters (the phenotype), and they are mapped to a fitness value. If there are $k$ parameters, then each individual can be thought of as occupying a point in $k$-dimensional space. The distance between two individuals can be taken as the Euclidian distance between them. (As suggested by Deb (1989), if the parameter ranges in different dimensions are widely different, it may be desirable to normalize each parameter to cover the range 0 to 1 before the Euclidian distance is determined.) There are other ways to define a distance metric, but we shall use the Euclidian distance. Our algorithm does not rely on any particular distance metric.

Having defined a fitness function and distance metric, the algorithm works as follows (terms in italics are explained below):

1. Initialize: equate the *modified fitness function* with the raw fitness function.
2. Run the GA (or other search technique), using the modified fitness function, keeping a record of the best individual found in the run.
3. Update the modified fitness function to give a depression in the region near the best individual, producing a new modified fitness function.
4. If the raw fitness of the best individual exceeds the *solution threshold*, display this as a *solution*.
5. If not all solutions have been found, return to step (2).

A single application of this overall algorithm we refer to as a *sequence*, since it consists of a sequence of several GA runs. Knowledge of niche locations (maxima) is propagated to subsequent runs in the same sequence.

A **solution** is a set of $k$ parameters which define the position of a maximum of interest.

The **solution threshold** represents the lower fitness limit for maxima of interest. (It is assumed that there are a known number, $p$, of "interesting" maxima, all with fitnesses greater than the solution threshold.) Its value is set individually for each problem. If no information is available about the likely fitness values of the maxima of interest, the solution threshold may be set to zero. In this case, the algorithm is terminated after the first $p$ peaks have been found.

The **modified fitness function**, $M(x)$, for an individual, $x$, is computed from the raw fitness function, $F(x)$, multiplied by a number of *single-peak derating functions*. Initially we set $M_0(x) \equiv F(x)$. At the end of each run, the best individual, $s_n$, found in that run is used to determine a single-peak derating function, $G(x, s_n)$. The modified fitness function is then updated according to:

$$M_{n+1}(x) \equiv M_n(x) * G(x, s_n) \tag{5}$$

## 2.3 Derating functions

Various forms for the single-peak derating function, $G$, are possible; two were tested: power law, Eqn. (6), and exponential, Eqn. (7). Here, $r$ is the niche radius (see below), $d_{xs}$ is the distance between $x$ and $s$, as determined by the distance metric. In Eqn. (6), $\alpha$ is the power factor, which determines how concave ($\alpha > 1$) or convex ($\alpha < 1$) the derating curve is, with $\alpha = 1$ giving a linear function. When $d_{xs} = 0$, $G_p(x, s) = 0$. In Eqn. (7), $m$ is the derating minimum value, which defines $G$ when $d_{xs} = 0$ (such that $G(x, s) = m$). $m$ must be greater than 0, since $\log(0) = -\infty$. $m$ also determines the concavity of the derating curve, with smaller values of $m$ producing more concavity. For both forms, when $d_{xs} \geq r$, we set $G(x, s) = 1$.

$$G_p(x,s) = \begin{cases} (d_{xs}/r)^{\alpha} & \text{if } d_{xs} < r \\ 1 & \text{otherwise} \end{cases} \tag{6}$$

$$G_e(x,s) = \begin{cases} \exp(\log m * (r - d_{xs})/r) & \text{if } d_{xs} < r \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

Several power law and exponential derating curves are shown in Figures 1 and 2 for $r = 1$.

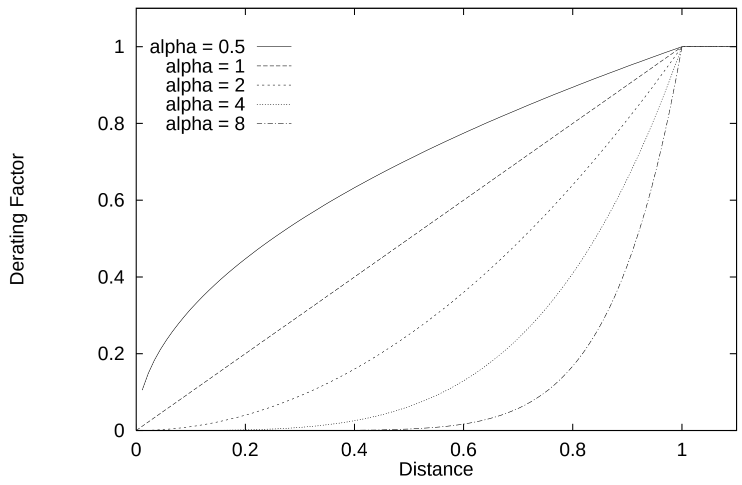



Figure 1: Power law Fitness Derating Curves

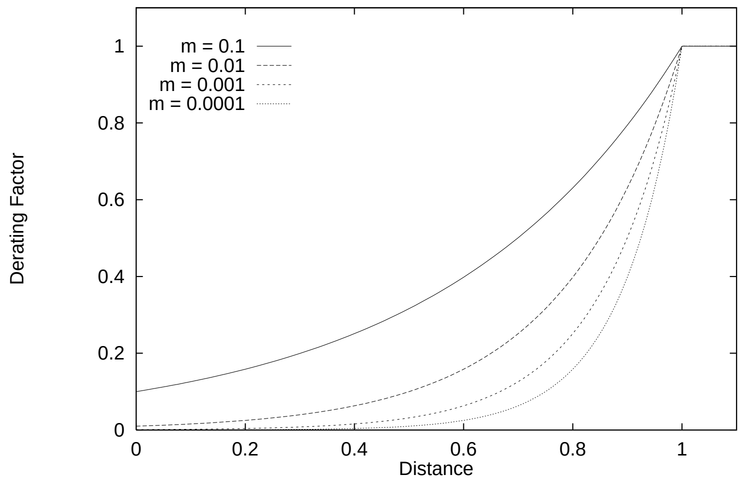



Figure 2: Exponential Fitness Derating Curves

The sharing function described by Goldberg and Richardson (1987), Deb (1989) and Deb and Goldberg (1989) performs the task of reducing the fitness of an individual depending on its distance from each other individual in the population. In a similar way, our fitness derating function performs the task of reducing the fitness of an individual depending on its distance from each best individual found in previous runs.

This is illustrated in Figure 3, which shows the raw function, $F(x) = \sin^2(2\pi x)$, and the modified function after it has been multiplied by the single-peak derating function centered on $x = 0.25$, using power-law derating with $r = 0.25$ and $\alpha = 2$.

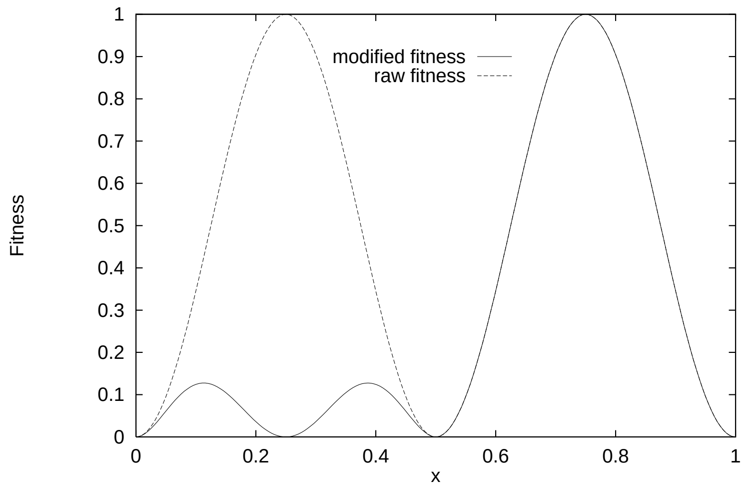



Figure 3: Suppression of one peak by a derating function

Note that evidence of the suppressed peak still remains, in the form of two small peaks. The size of these introduced peaks can be made arbitrarily small by using larger values of $\alpha$. However, there are dangers in using too large a value, and these are discussed in Section 5.1.2.

To determine the value of niche radius, $r$, we use the same method as Deb (1989), which is as follows. If we imagine that each of the $p$ maxima in the function are surrounded by a hypersphere of radius $r$, and that these hyperspheres do not overlap, and that they completely fill the $k$-dimensional problem space, then if the parameter range for each dimension is normalized to be 0 to 1, $r$ is given by:

$$r = \frac{\sqrt{k}}{2 * \sqrt[k]{p}} \tag{8}$$

Deb's technique is simple, but requires that we know (or estimate) the number of maxima in the function, and assumes that all maxima are equally spread throughout the search space. Clearly, there will be functions where these restrictions are not satisfied. A thorough investigation of the issue of niche radius is a topic for further research, but some ideas are discussed in Section 5.3. For our present experimental investigations, we adopt the simple approach to niche radius given above.

## 2.4 Termination conditions

Deciding when to halt a GA run and start the next iteration is not a trivial task. Towards the end of a run of any traditional GA, the search efficiency falls off since the population becomes more uniform, causing exploration to rely increasingly on mutation (Goldberg, 1989b). For our algorithm, we need to decide at what point in the run we are unlikely to improve on the best individual found so far, and then terminate the run. The technique we adopt is to record the population average fitness over a *halting window* of $h$ generations, and terminate the run if the fitness of any generation is not greater than that of the population $h$ generations earlier. Values of $h$ between 5 and 20 gave good results with the test functions we tried. More complex functions with longer convergence times might need larger values. A run is also terminated if an individual is found with a known maximum fitness target, or a maximum number of generations has been reached.

## 2.5 Time Complexity

To determine the time complexity, we assume that the performance of a GA trying to locate any one of several maxima in a multimodal function is approximately equal to that of a GA trying to locate the global maximum in a similar, but unimodal function. To test this assumption, we ran 250 trials of a conventional GA on each of three 30-bit binary-coded functions (population size = 20, halting window = 20). The first function had 100 equally spaced, equal height maxima. The second function was an exponential "spike," with a single maximum. The third was a summation of the first two, giving a single global maximum, and 99 local maxima.

We compared the number of function evaluations required to locate any one of the maxima of the first function with the number required to locate the global maximum of the second and third functions. The GA ran equally quickly on both the unimodal functions. But on the function with 100 equal maxima, the GA was approximately twice as fast. Not only that, but the accuracy of solutions was improved by a factor of 100.

So, if a traditional GA needs a population of $n$ to locate the maximum of a unimodal function, and converges in $g_c$ generations, it would appear that the addition of more maxima tends to *reduce* the convergence time, and *increase* the accuracy of solutions found. Therefore, it would seem conservative to assume that the sequential niche algorithm requires a population of equal size, and converges in an equal number of generations, when locating any one of the maxima of a multimodal function.

Our technique involves multiple runs of a GA, and the initial run will have a time complexity identical to that of the unmodified, underlying GA. But as peaks are located, successive runs in the algorithm will get slower. This is because the complexity of computing the fitness derating function increases linearly with the number of peaks found. Using the same notation as in Eqn. (4), the first run will have a complexity $O(\alpha n g_c)$. On the second run, in each generation the algorithm will need to compute the distance of each individual from the peak found in the first run. This will require additional time, proportional to the population size, $n$. In the third run, there will be two previous peaks to compute the distance from, so the additional time will be proportional to $(2n)$. If the fitness of any uninteresting peaks is small compared with that of the smallest interesting peak, (i.e. $\phi = 1$), a total of approximately $p$ runs will be needed. The total additional time will therefore be proportional to $(0+n+2n+3n+\cdots(p-2)n+(p-1)n)$. This is $((p-2)n*(p-1)/2)$, or approximately $(p^2*n/2)$. The total time for the underlying GA to perform these $p$ runs is proportional to $(npg_c)$. Denoting the time constants as $\alpha$ and $\beta$, as in Section 1.3, the overall time complexity, $\mathcal{C}_{seq}$, can be written as:

$$\mathcal{C}_{seq} = O(npg_c(\alpha + 0.5\beta p)) \tag{9}$$

For comparison, the complexity of the sharing algorithm, from Eqn. (4), is

$$\mathcal{C}_{share} = O(n\phi p g_c(\alpha + 5\beta p))$$

So, for $\phi = 1$, the two complexities are of a similar form; sharing requires one run with a population of size $np$, while the sequential technique requires $p$ runs with a population size of $n$. (If the uninteresting peaks have a fitness close to that of the lowest interesting peak, both methods will be slower; sharing will require an increased population size, and the sequential technique will require additional runs.) In this sense, the two methods differ only in that the sequential niche technique requires ten times fewer distance calculations. However, if $g_c$ varies with population size, the time complexities will be quite different.

Putting $g_c = O(f(N))$, where $N$ is the population size, the speedup, $\mathcal{S}$, achieved using the sequential technique is

$$\mathcal{S} = \frac{f(np)(\alpha/\beta + 5p)}{f(n)(\alpha/\beta + 0.5p)} \tag{10}$$

The number of generations to convergence, $g_c$, is highly dependent on both the particular problem, and the particular GA used, but approximate estimates have been made. Goldberg (1989b) gives the number of generations for an optimum individual to take over the population as $O(\log N)$. In a study of a unimodal, discrete function, Mühlenbein and Schlierkamp-Voosen (1993) found that $g_c$ was independent of $N$ when no mutation was used, for values of $N$ above a certain threshold. However, for a continuous, multimodal function with mutation they found $g_c$ to be $O(\log N)$. Another estimate is that $g_c$ is $O(N)$ for certain problems (H. Mühlenbein and J.J. Grefenstette, *personal communication*, September 30th 1992). Nevertheless, these estimates of $g_c$ as $O(1)$, $O(\log N)$ or $O(N)$ all agree that $g_c$ will not decrease with $N$, and on this basis, the sequential niche technique will always be faster than sharing.

The speedup factor will also depend on the number of peaks, $p$, relative to the ratio of times for fitness evaluation and distance calculation, $\alpha/\beta$. In most real problems of interest, we would expect a function evaluation to take longer than a distance calculation (Goldberg & Deb, 1991), and typically $\alpha/\beta \gg 1$. Therefore for small $p$, ($p < n$ and $p \ll \alpha/\beta$), the speedup factors for $g_c = O(1)$, $O(\log N)$ and $O(N)$ are:

$$\mathcal{S}_1 \approx 1 \tag{11}$$

$$1 < \mathcal{S}_{\log N} < 2 \tag{12}$$

$$\mathcal{S}_N \approx p \tag{13}$$

For large $p$, ($p \gg \alpha/\beta$ and $p \gg n$), the speedup factors are:

$$\mathcal{S}_1 \approx 10 \tag{14}$$

$$\mathcal{S}_{\log N} \approx 10(1 + \log p / \log n) \quad (15)$$
$$\mathcal{S}_N \approx 10p \quad (16)$$

This suggests that the sequential niche technique will always be better than traditional fitness sharing, and will be especially useful in problems where $p$ is large.

# 3 Test Functions

Our algorithm was tested on a number of functions used by previous researchers. The functions described in Section 3.1 are multimodal, but only the global maximum is of interest. In the results (Section 4), we show that unlike a traditional GA, the sequential niche technique locates the global maximum easily. More important are those functions described in Section 3.2. These are multimodal functions in which *all* maxima are of interest. Our results show that, like the fitness sharing approach, the sequential niche technique can efficiently locate all maxima.

For the tests, the GA used at the heart of this algorithm was a traditional one, based on Goldberg's SGA (Goldberg, 1989a). It used generational replacement, stochastic remainder selection without replacement, linear fitness scaling (with a factor of 2), and single-point crossover.

## 3.1 Trap functions

Trap functions are deceptive functions in which the fitness is a function of the count, $c$, of the number of ones in the binary chromosome (sometimes referred to as the *unitation* (Goldberg, 1992)).

### 3.1.1 Two Peak Trap

In his comparison of various optimization techniques, Ackley (1987) found that a two peak trap function was very hard to optimize. The fitness function is defined by Eqn. (17), and is shown in Figure 4.

$$F(c) = \begin{cases} 160/15 * (15 - c) & \text{if } c < 15 \\ 200/5 * (c - 15) & \text{otherwise} \end{cases} \quad (17)$$

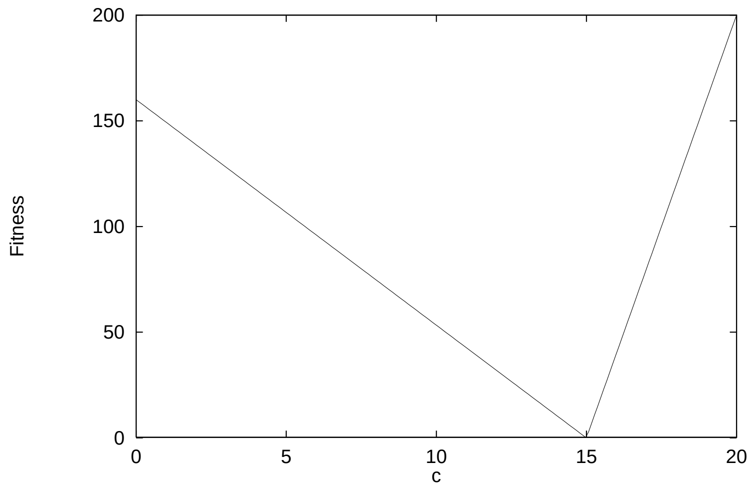



Figure 4: Two Peak Trap

The global maximum is at $c = 20$, and has a fitness of 200, but there is a false maximum at $c = 0$, with a fitness of 160. Values of $c$ between 1 and 15 lead towards the false peak.

The traditional problem here is not so much to find *both* peaks, but simply to avoid getting stuck on the false peak, and to find the real one. Difficulties arise for a traditional GA because the problem is deceptive (Deb & Goldberg, 1991). The distance metric used is the difference in the unitation.

### 3.1.2 Fully deceptive two peak trap

Deb and Goldberg (1991) point out that Ackley's two peak trap function is not *fully* deceptive. They provide a formula for determining full deception, and, with other parameters the same, full deception is achieved if the height of the local maximum is greater than 186. We used a height of 199.9 to test their suggestion that the problem would be harder to solve if it were fully deceptive.

### 3.1.3 Central Two Peak Trap

Ackley (1987) also mentions a variation of the two peak trap function which he considers to be even harder to solve. Instead of a single local maximum, this function has $\binom{20}{10} = 184756$ local maxima in *binary* space. In *unitation* space, however, it still has only one local maximum. This is located at $c = 10$. The fitness function is defined by Eqn. (18), and is shown in Figure 5.

$$F(c) = \begin{cases} 160/10 * c & \text{if } c < 10 \\ 200/5 * (20 - c) & \text{if } c > 15 \\ 160/5 * (15 - c) & \text{otherwise} \end{cases} \tag{18}$$

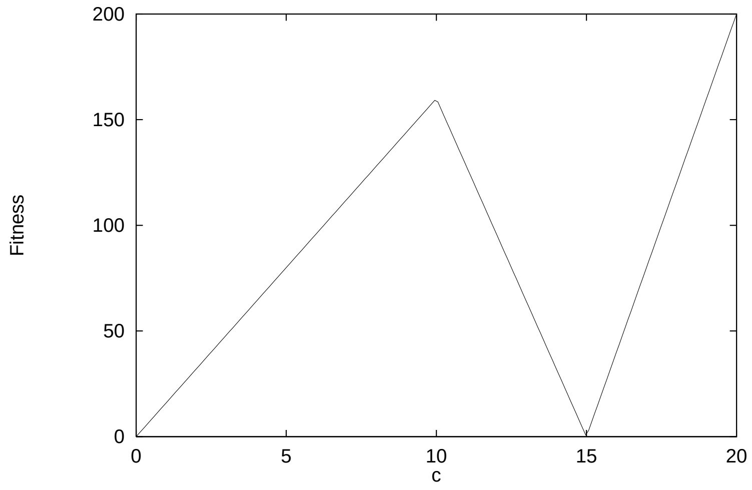



Figure 5: Central Two Peak Trap

## 3.2 Binary-coded Functions

Binary-coded functions are those in which the chromosome represents one or more binary-coded parameters, and the fitness is a function of these parameters. Deb (1989) describes several functions he used to test his sharing GA. These are shown in Figures 6 to 10. (Three of these functions are also described in (Deb & Goldberg, 1989).) For functions F1 to F4, a 30-bit binary-coded chromosome was used to represent $x$ in the range 0 to 1. We use the difference in $x$ value as the distance metric.

### 3.2.1 F1 - Equal maxima

The first of Deb's functions has 5 equally spaced maxima of equal height:

$$F1(x) = \sin^6(5\pi x) \tag{19}$$

The maxima are at $x$ values of 0.1, 0.3, 0.5, 0.7 and 0.9.

### 3.2.2 F2 - Decreasing maxima

The second function is like F1, but the maxima decrease in height exponentially:

$$F2(x) = \exp\left(-2\log(2) * \left(\frac{x-0.1}{0.8}\right)^2\right) * \sin^6(5\pi x) \tag{20}$$

The maxima are at the same $x$ values as for F1, but the peak heights vary from 1.0 to 0.25.

#### 3.2.3 F3 - Uneven maxima

The third function is like F1, but the maxima are unevenly spaced:

$$F3(x) = \sin^6(5\pi(x^{3/4} - 0.05)) \tag{21}$$

The maxima are at $x$ values of approximately 0.08, 0.246, 0.45, 0.681 and 0.934, all with a height of 1.0.

#### 3.2.4 F4 - Uneven decreasing maxima

The fourth function is like F3, but the maxima decrease in height exponentially.

$$F4(x) = \exp\left(-2\log(2) * \left(\frac{x-0.08}{0.854}\right)^2\right) * \sin^6(5\pi(x^{3/4} - 0.05)) \tag{22}$$

The maxima are at the same $x$ values as F3, with the same heights as F2.

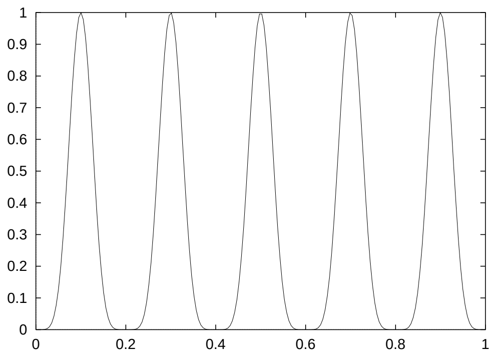



Figure 6: F1 Equal Maxima

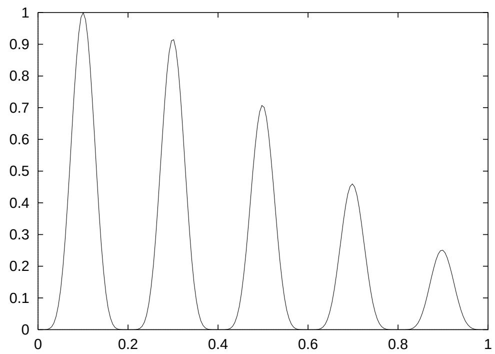



Figure 7: F2 Decreasing Maxima

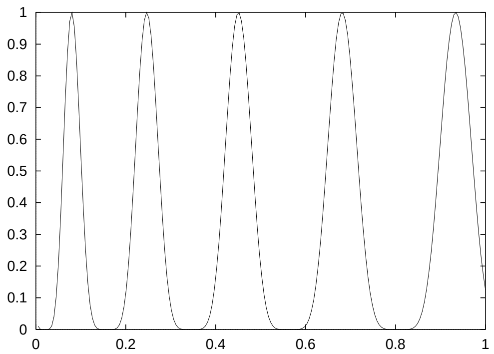



Figure 8: F3 Uneven Maxima

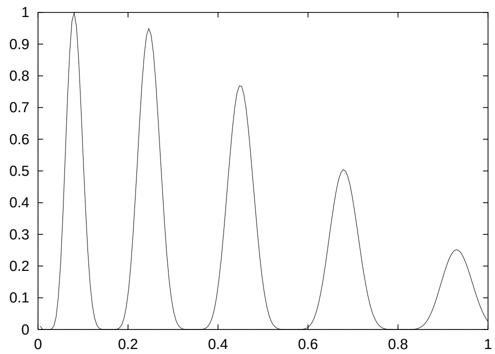



Figure 9: F4 Uneven Decreasing Maxima

#### 3.2.5 F5 - Himmelblau's function

This is a function of two variables, $x, y$, (which Deb modified to be a maximization problem) given by:

$$F5(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2 \tag{23}$$

This has four maxima of equal height (200) at approximately (3.58, −1.86), (3.0, 2.0), (−2.815, 3.125) and (−3.78, −3.28). A 30-bit chromosome is used to represent two 15-bit binary-coded parameters, in the range $-6 \leq x, y \leq +6$. The distance metric is the Euclidian distance in $x$-$y$ space.

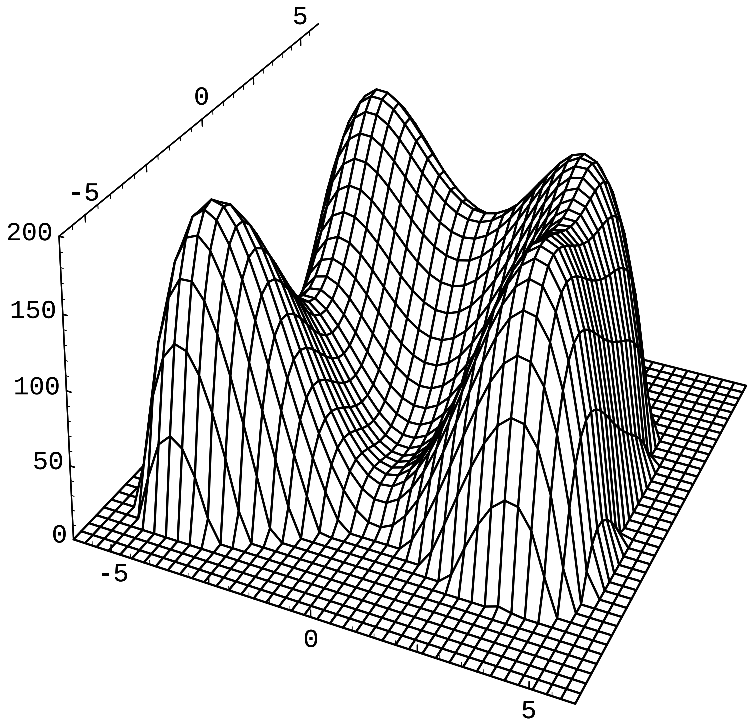



Figure 10: F5 Modified Himmelblau's Function

# 4 Results

## 4.1 Performance Measures

Objectively measuring the performance of a GA is not a trivial matter, and surprisingly, little formal progress has been reported in this area. Factors to be taken into account are: *speed* – how quickly (either in cpu time, elapsed time, or number of function evaluations) the algorithm completes; *accuracy* – how close the solution(s) found are to the optimum solutions; *success rate* – what proportion of runs converge to an optimal rather than sub-optimal solution; and *consistency* – how much variation in the previous three criteria can be expected from one run to the next.

Depending on how we adjust the parameters of a GA, we can usually trade off speed for accuracy, or speed for success rate. But it is impossible to compute a single 'figure of merit' for an algorithm, taking all factors into account, because the relative importance of these criteria will vary with application. Instead, we shall use a measure which combines *speed* and *success rate* as our primary means of comparing algorithms, and also quote figures for *accuracy* and *consistency*. Each of these terms is defined below.

We assume that the goal of running the GA is to find a *complete* solution set, i.e. a set of all the maxima of interest in a function. For any instance of use of any problem solving method, there is always a possibility that a complete solution set will not be found. To collect performance statistics in this case, the overall algorithm is simply (blindly) iterated as many times as necessary to collect a complete solution set, taking the solutions found in all iterations together. This assumes, of course, that a complete solution set can be recognized once it is found.

**Success rate** is the proportion of sequences which come up with the desired *solution set*. For low success rates, we can expect to have to iterate the algorithm several times before a complete solution set has been collected.

**Evaluations expected** is a measure of how many function evaluations, on average, we can expect to have to perform to find the complete solution set. The number of evaluations is counted over all runs in a sequence, and if a particular sequence fails to find the solution set, we iterate the algorithm, and *continue counting* the number of evaluations performed. We also quote the 95% confidence margin on this value. (This figure includes no allowance for the number of fitness derating calculations performed, so care must be taken if using it to compare the speed of different algorithms.)

**Consistency** of the algorithm is expressed in terms of the sample standard deviation in the evaluations expected measure.

**Accuracy** is expressed in terms of the *root-mean-square* (RMS) error in the solutions found. Values are computed by finding the distance between each solution produced by the algorithm and the nearest exact solution. This distance is then squared, and the RMS value is the square-root of the mean of these values. This gives an indication of the likely error in a particular solution, in the same units as the original parameter space. There are many ways in which this this accuracy figure might be "normalized," to allow comparison between different problems. All have some drawbacks, so we simply quote the basic RMS figure. Accuracy figures are not given if all solutions were found exactly.

An **average runs** figure is also given. This indicates how many runs of the GA were needed, on average, to find all solutions. Like the evaluations expected figure, the number of runs is aggregated if several sequences are necessary to find the solution set. Ideally, using the sequential niche algorithm, the number of runs should equal the number of solutions to be found.

## 4.2 Trap Functions

These functions are interesting because they are highly deceptive, and simple iteration of a traditional GA is a totally ineffective strategy for locating the global optimum. Yet, as we show below, the iteration of a GA with the addition of a fitness derating function gives good results.

### 4.2.1 Two Peak Trap

The results from Ackley's (1987) tests on the 20-bit trap problem using his technique known as *iterated genetic search* are shown in Table 1, along with a summary of our own results on the same task using an iterated traditional GA (for comparison), and the sequential niche GA (results are given more fully later on).

| **Algorithm** | **Success rate** | **Evaluations expected** |
|---|---|---|
| Iterated Genetic Search (Ackley) | 0.00% | >1000000 |
| Iterated Traditional GA | 0.01% | 2300000 |
| Sequential Niche GA | 77.60% | 4900 |

Table 1: Comparison of results of different techniques on the trap function

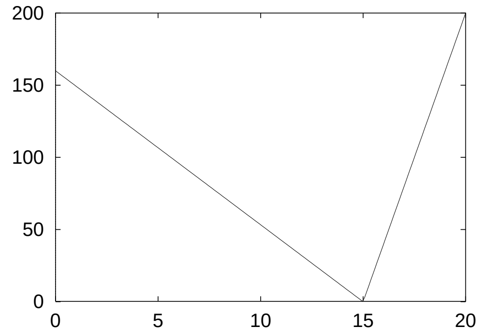
Ackley's **Genetic Search** algorithm had a population of 50, $P_{cross}$ (crossover probability) of 1, and $P_{mutation}$ (mutation probability) varied between 0.25 and 0.0175. He performed up to fifty trials, but abandoned further trials as soon as one failed to find a solution within his limit of 1 million evaluations. To see just how good (or bad) a GA was at this problem, we tried a **traditional GA** with more conventional parameters (population = 50, $P_{cross} = 0.8$, $P_{mutation} = 0.01$) over a much larger number of trials. Out of 92000 runs, only 12 found the optimum solution. Usually these appeared in the first generation, which implies that a random search would have done equally well. With the same basic parameters as the traditional GA, the **sequential niche GA** found the optimum solution in 194 out of 250 trials, in an average of 4900 evaluations.[3]

The basic parameters used with the sequential niche GA were: population = 50, $P_{cross} = 0.8$, $P_{mutation} = 0.01$, halting window = 5, power law derating, and $\alpha = 2$, unless otherwise stated. (These parameters were chosen after a small amount of tuning had been done – see below.) Niche radius, determined from Eqn. (8), was 5.

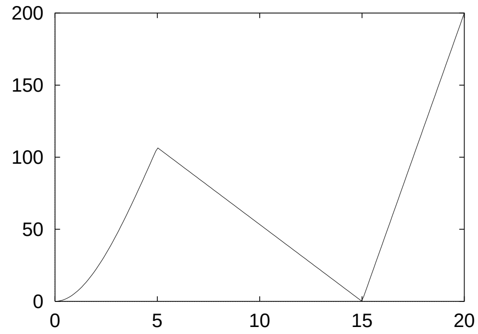
A typical sequence has four runs which find successive peaks at $x$ values of 0 (11), 5 (1), 10 (1) and 20 (16). (The figures in brackets are the typical generation numbers in which the solutions were found.) The "peaks" at 5 and 10 are artifacts of the algorithm, corresponding to peaks in the *modified* fitness function, but not peaks

[3] Ackley's own method, known as SIGH, solved the trap problem in only 780 evaluations. However, this surprisingly good result was due to the fact that SIGH includes a heuristic which makes it especially good at this particular problem.

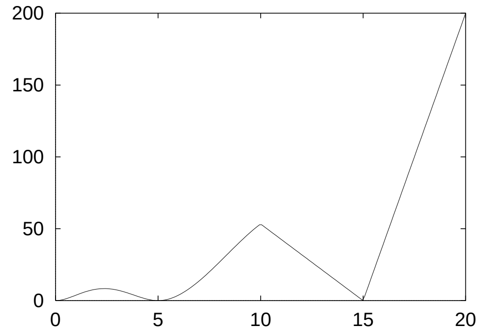
in the unmodified function, as illustrated in Figure 11. (Techniques for rejecting these introduced peaks are discussed in Section 5.)



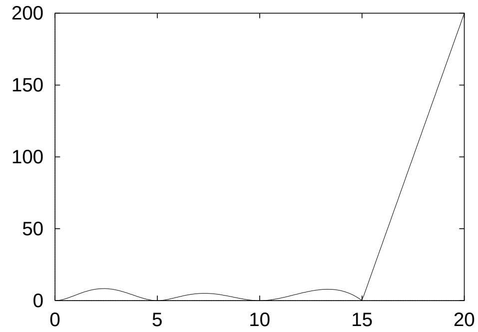

(a) First run




(b) Second run




(c) Third run




(d) Fourth run

Figure 11: Modified fitness functions on successive runs on the two peak trap function

The fourth run in a sequence generally locates the maximum at 20 after about 16 generations. However, in some sequences, the fourth run is stopped too early. This may happen if a very good, but sub-optimum individual, (with a value of 18, perhaps), appears by chance in an early generation. This boosts the average fitness of an early generation, which in turn causes the run to be terminated early, before it has converged on the global maximum. When the algorithm achieves a "near-miss," the modified fitness becomes much reduced at the global maximum, making it difficult for subsequent runs to find it.

Extensive tests were carried out with the sequential niche algorithm to evaluate the two types of derating functions. Two hundred and fifty separate sequences were carried out in each case. Each sequence had an upper limit of 6 runs in which to find the global maximum. Table 2 shows the performance measures (as described above) in each case. Most figures are rounded to two significant digits, to avoid giving the appearance of false accuracy.

These results clearly show that good performance can be obtained over a range of different derating functions, so long as the derating curve is sufficiently concave. Among the power functions with $\alpha = 2$, 4 and 8, and exponential functions with m = 0.01, 0.001 and 0.0001, there is statistically little to choose between them in terms of the expected number of evaluations to a solution.

For all further tests, (except where otherwise stated), we used power function derating, with $\alpha = 2$.

Further tests were carried out to determine the effect of changing other parameters.

**Halting window.** We tried increasing the halting window from 5 to 10. This helps to avoid the early termination and "near-miss" problem described above. It therefore increases the success rate (to 88%), and reduces the average number of runs per sequence required (to 4.8). However, it extends the length of all runs, so the overall effect on the number of evaluations expected is negligible ($5200 \pm 340$). Further increases in halting window would be expected to degrade the performance, since success rate and runs per sequence approach asymptotic limits, while the length of runs may increase without bounds.

| Derating function | Success rate | Ave. runs | Evals expected | Conf. margin | Std dev |
|---|---|---|---|---|---|
| power, $\alpha = 0.5$ | 15% | 38.6 | 29000 | $\pm$6000 | 19000 |
| power, $\alpha = 1$ | 44% | 13.1 | 11000 | $\pm$1400 | 7300 |
| power, $\alpha = 2$ | 78% | 5.7 | 4900 | $\pm$400 | 2900 |
| power, $\alpha = 4$ | 79% | 5.6 | 4700 | $\pm$400 | 2900 |
| power, $\alpha = 8$ | 73% | 6.3 | 5200 | $\pm$420 | 2900 |
| exp, m = 0.1 | 38% | 14.4 | 14000 | $\pm$1900 | 9500 |
| exp, m = 0.01 | 79% | 5.5 | 4900 | $\pm$380 | 2700 |
| exp, m = 0.001 | 72% | 6.4 | 5600 | $\pm$500 | 3400 |
| exp, m = 0.0001 | 74% | 6.1 | 5400 | $\pm$530 | 3700 |

Table 2: Results of the trap function

**Maximum runs per sequence.** In the original experiments this value was set to 6. We also tried values of 4, 12 and 24, with halting window = 5 (see Table 3). Once again, increasing the max runs/sequence has positive and negative effects. If there is a "near-miss," it may take many, many runs to locate the global maximum—so allowing a large upper limit on the runs per sequence can tend to increase the expected evaluations. On the other hand, a "not so near miss" may be able to locate the global maximum given a few extra runs—which can be faster overall than giving up and starting a new sequence from scratch. These two effects seem largely to cancel out.

| Max runs/ sequence | Success rate | Ave. runs | Evals expected | Conf. margin | Std dev |
|---|---|---|---|---|---|
| 4 | 80% | 5.0 | 4500 | $\pm$260 | 1900 |
| 6 | 78% | 5.7 | 4900 | $\pm$400 | 2900 |
| 12 | 88% | 6.3 | 5100 | $\pm$400 | 3000 |
| 24 | 90% | 7.1 | 6400 | $\pm$1100 | 9100 |

Table 3: Varying the max runs/sequence on the trap function

#### 4.2.2 Fully deceptive two peak trap

With the height of the local maximum increased to 99.95% of that of the global maximum, after 250 trials we found no statistically significant difference in the results. ($4700 \pm 350$ evaluations, 95% confidence; std dev = 2500). This hardly seems surprising. The first three runs in a sequence find it no harder (perhaps slightly easier) to locate the local maximum at 0, and the two introduced maxima at 5 and 10. These maxima are adequately suppressed by the derating function, making the job of locating the maximum at 20 no more difficult than before.

#### 4.2.3 Central Two Peak Trap

Again, using the sequential niche algorithm, this was no more difficult than the original trap problem; in fact it turned out to be easier. With similar conditions to the trap function, this was solved in an average of 3000 evaluations ($\pm$230, 95% confidence; std dev = 1700), needing only 4.1 runs per sequence. Ackley's own technique, SIGH, (Ackley, 1985) which performed well on the two peak trap problem, exceeded the 1 million evaluations limit on this problem.

### 4.3 Binary-coded Functions

For the tests on Deb's functions F1 to F5, we took the population size used by Deb, 100, and scaled it down according to the number of maxima in the function. That is, we used a population of 20 for F1 to F4, and 26 for F5 (our GA supported only even-sized populations). This compensates for the fact that our algorithm does not have to maintain a population at *each* maximum. Other parameters applied to all algorithms were $P_{cross} = 0.9$, $P_{mutation} = 0.01$, halting window = 20. The niche radius (determined using Eqn. (8)) was 0.1 for F1-F4, and 4.24 for F5. (These are the same parameters as used by Deb, except that he used $P_{mutation} = 0$.) The results over 250 sequences (again, rounded to two figures) are shown in Table 4.

| Function | Success rate | Ave. runs | Evals. expected | Conf. margin | Std dev | RMS error |
|---|---|---|---|---|---|---|
| F1 Equal maxima | 99% | 5.1 | 1900 | ±74 | 600 | 0.0043 |
| F2 Decreasing maxima | 90% | 5.6 | 3300 | ±140 | 1100 | 0.0075 |
| F3 Uneven maxima | 100% | 5.2 | 1900 | ±79 | 630 | 0.0039 |
| F4 Uneven decreasing maxima | 99% | 5.1 | 3000 | ±66 | 530 | 0.0041 |
| F5 Himmelblau's function | 76% | 6.1 | 5500 | ±530 | 3700 | 0.20 |

Table 4: Results on functions F1 to F5

These results demonstrate that, for the 1-dimensional problems F1-F4, virtually every sequence finds all five maxima, and we can expect all five solutions to be found in little over five runs. The 2-dimensional problem, F5, achieves a high success rate, with three-quarters of sequences finding all four maxima. The lower success rate pushes up the average number of runs required. The RMS error value for F5 appears high in comparison with F1 to F4, but this is because the parameter space is larger (it covers a range from $-6$ to $+6$ in two dimensions, rather than just 0 to 1 in one dimension). All RMS error values are less than 10% of the niche radius. This shows that the multiple maxima of a function can be located in little over $p$ runs, with reasonable accuracy.

In addition, for F2, tests were carried out with various values of $\alpha$ (see Table 5). As with the trap function, we found that $\alpha = 0.5$ or 1 gave poor results, while $\alpha = 2$, 4 or 8 gave good results.

| $\alpha$ | Success rate | Ave. runs | Evals. expected | Conf. margin | Std dev | RMS error |
|---|---|---|---|---|---|---|
| 0.5 | 2% | 306 | 190000 | ±120000 | 120000 | 0.0090 |
| 1 | 75% | 6.6 | 4000 | ±290 | 2000 | 0.0074 |
| 2 | 90% | 5.6 | 3300 | ±140 | 1100 | 0.0075 |
| 4 | 96% | 5.2 | 3100 | ±89 | 700 | 0.0070 |
| 8 | 100% | 5.1 | 3000 | ±68 | 550 | 0.0065 |

Table 5: Results of varying $\alpha$ on function F2

# 5 Discussion

The results clearly show that the sequential niche algorithm can give good results. In the case of the trap functions, generally considered to be almost insoluble by a traditional GA, our algorithm located all the maxima in parameter space (i.e. all the maxima in unitation space). In doing so, we also located the global maximum without difficulty. In the case of the binary-coded functions, the sequential niche algorithm, like the sharing algorithm used by Deb (1989), found all maxima in a reasonable time.

Unfortunately, these results cannot be compared directly with Deb's. The aim of Deb's thesis was to show that a sharing function can evolve and maintain stable sub-populations at all maxima of a variety of multimodal functions. He did not attempt to optimize his algorithm with regard to speed of convergence or accuracy of results. Our sequential niche algorithm does not attempt to simultaneously maintain subpopulations at each maximum. Instead it directly addresses the more utilitarian task of locating each maximum. Our results show that using the sequential niche technique, the $p$ peaks of a function such as F1-F5 can be located in little more than $p$ runs of a traditional GA. The ultimate speed and accuracy of these solutions depend on the qualities of the particular underlying GA implementation. Like Deb, we make little attempt to optimize these performance measures. Instead our results demonstrate that the sequential niche technique requires fewer runs than blind iteration, and our theoretical analysis has shown that the time complexity is an improvement over the sharing algorithm.

Fundamentally, both the sequential niche and the fitness sharing methods rely on a certain *regularity* in the search space. They assume that regions close to maxima of interest do not contain *other* maxima of interest. However, the validity of this assumption depends not only (as would be expected) on the coding scheme used, but also on how we define *closeness*. As pointed out in Section 2.2, our algorithm does not itself depend on any particular distance metric, but the *success* of the algorithm most certainly *will* be affected by whether a suitable distance metric is used or not.

This applies equally to the fitness sharing algorithm. Goldberg *et al* (1992) had difficulty solving a trap problem composed of five sub-functions. They used the Hamming distance as the distance metric. If they had instead used Euclidian distance in a 5-dimensional unitation space, their task would have been simple (i.e. phenotypic sharing rather than genotypic sharing).

## 5.1 Problems with niche radius

If an inappropriate niche radius is used, problems will arise. For simplicity, we made the same assumptions as Deb (1989) that maxima are evenly distributed throughout the search space. This leads to Eqn. (8), which gives a single value for the niche radius which is used for all niches in the problem. In general, however, maxima will *not* be evenly distributed, so we will underestimate the size of some niches, while overestimating the size of others. The problems which this leads to also occur with the fitness sharing algorithm. If too small a niche radius is used with the fitness sharing algorithm, Smith *et al* (1992, p12) pointed out that individuals *near* a peak may gain a foothold if their fitness is not (significantly) shared with individuals *at* the peak. Alternatively, if too large a niche radius is used, solutions may be missed entirely (Smith *et al*, 1992; Goldberg *et al*, 1992). Similar problems occur with the sequential niche technique, and it is easy to see why when the modified fitness function is considered.

### 5.1.1 Small niche radius

As was illustrated in Figure 3, modifying the fitness function introduces new peaks. This effect will be exacerbated if the niche radius is too small, or an insufficiently concave derating function is used (e.g. $\alpha < 2$). The result is that extra runs may be required, to suppress the introduced peaks, as happened while solving the two peak trap problem. A similar problem is illustrated in Figure 12. The raw fitness function has the equation $f(x) = \sin^2(2\pi x^2)$, giving two peaks of equal height, but unequal width. Eqn. (8) gives the simplistic niche radius as 0.25. When the derating function, with $\alpha = 2$, is applied to the wider peak, two new peaks are introduced, near $x = 0.3$ and $x = 0.6$. (In problems with higher dimensionality, many more than two peaks may be introduced.) The introduced peaks will generally be at a distance of slightly less than the niche radius from the solution found. No matter what type of derating function is used (power-law, exponential, etc.), peaks will always be introduced in this way.

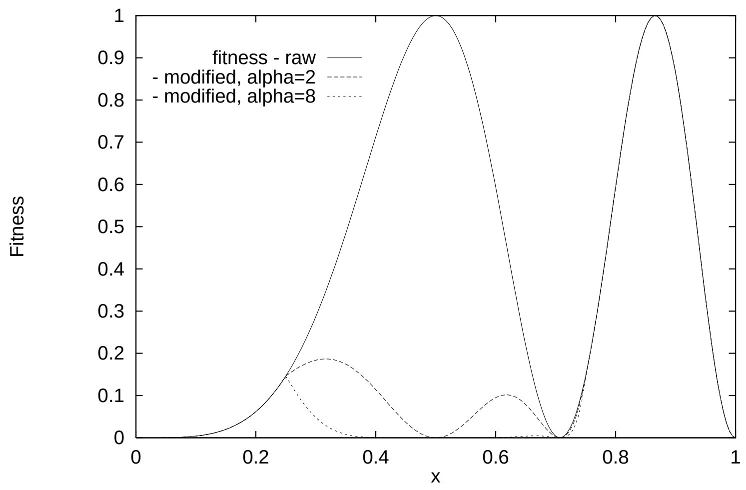



Figure 12: Peaks introduced by the use of too small a niche radius

Figure 12 also shows that using a larger value for $\alpha$ can reduce the size of an introduced peak. But it is also clear that however large a value we use for $\alpha$, we cannot eliminate introduced peaks entirely.

### 5.1.2 Large niche radius

Over estimation of the niche radius brings problems too. Figure 13 shows the same function as in Figure 12, but with the niche radius doubled. Despite using a larger niche radius, with $\alpha = 2$ the introduced peaks are still in evidence. Worse than this, the adjacent peak has been affected. Its height has been significantly reduced, and its position shifted slightly. The height reduction could lead to a peak not being recognized as a solution, if the solution threshold has been set too high.

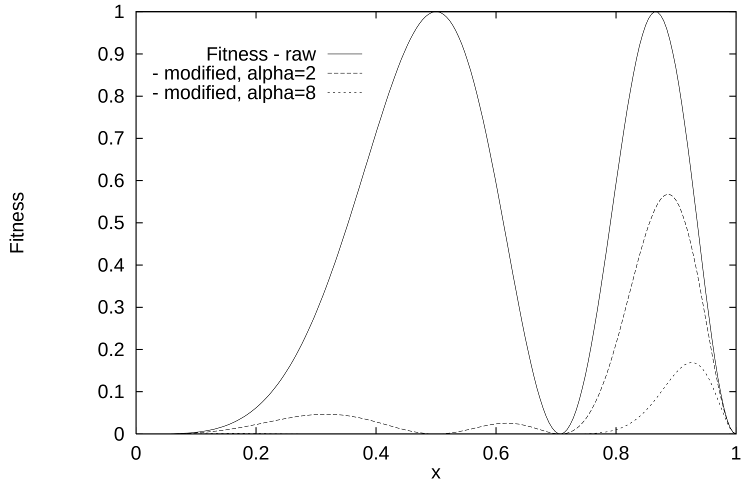



Figure 13: Effects of using too large a niche radius

With the higher value of $\alpha$, the effect on the adjacent peak is even more drastic. This demonstrates why we cannot use too high a value of $\alpha$—we risk losing maxima of interest, or getting a seriously incorrect solution value, if our niche radius has been over-estimated.

### 5.1.3 Solutions

Fortunately, reasonable solutions to both of these problems can be devised. Introduced peaks will often be much smaller than the true peaks, and their effects will not be noticed. If they are of moderate size, they may need additional runs in the sequence to suppress them. In extreme cases, an introduced peak might be large enough to be mistaken for a real peak. Such rogue "solutions" should not be difficult to detect. Firstly, they are located at a distance from a true peak which is always equal to, or slightly less than the niche radius. Secondly, performing a local search technique, such as hill climbing, on the unmodified fitness function, would cause a rogue solution to converge on a previously found real solution.

The use of a local search technique would also overcome the inaccuracy introduced by the shift in position of an adjacent peak when too large a niche radius is used.

The only remedy for peaks lost through the use of too large a niche radius is to use a smaller niche radius. In fact, the best approach to all these problems is to try and get the niche radius correct. This is discussed further in Section 5.3.

Similar problems arise with the fitness sharing algorithm if the niche radius is too large or small, but solutions might not be quite so simple. For example, if individuals are trapped in peaks introduced by too small a niche radius, a larger population would be required, to fill the additional "niches." This would not be easy to either detect or implement. With the sequential niche algorithm, however, these additional "niches" get filled quite automatically by additional runs.

## 5.2 Other Problems

Several other problems can be identified in the current implementation, mostly related to the type of problems which afflict GAs in general. These give rise to three symptoms:

- *Inaccuracy.* Solutions are not entirely accurate
- *Incompleteness.* Sequences sometimes fail to find all solutions
- *Extra runs.* More runs are often required than the number of solutions

The causes of and solutions to these symptoms are considered below.

### 5.2.1 Inaccuracy

This is a standard problem with all traditional GAs (DeJong, 1985). One solution is to use a hybrid method in which a GA finds approximate solutions, then to use a local search technique, such as hill climbing, (using the raw fitness function), to give a more accurate result (Goldberg, 1989a; Davis, 1991).

This problem also arises, and can be solved in the same way, if too large a niche radius is used, or the derating function is too concave, as was discussed in Section 5.1.2.

### 5.2.2 Incompleteness

This is mainly a side effect of locating solutions very inaccurately. The difficulties of deciding exactly when to halt a particular run (see Section 2.4) mean that occasionally a run is halted before it has properly converged. This generally leads to the determined location of the maximum being inaccurate. Often the peak will not be counted as a solution, since the fitness at this point is below the solution threshold. But this inaccurate solution is probably within the niche radius, so the effect of the fitness derating function causes the fitness at the true peak to be reduced. This makes the true peak less likely to be found in subsequent runs, so the sequence may reach its upper "maximum runs" limit without ever locating the true peak.

There are two solutions to this problem. Firstly, we can try to avoid halting runs before they have converged. But the more conservative we are in judging convergence, the more we will increase the average number of generations needed per run. The overall effect may be to increase the average number of evaluations required. Sometimes it is better to have an algorithm which fails perhaps 25% of the time, but is fast, than to have one which always succeeds, but is much slower. The second solution is the same as the solution to the problem of inaccuracy—use a local search technique. This approach solves both problems at once, without necessarily adding greatly to the number of evaluations required.

### 5.2.3 Extra runs

There are three causes of this problem. Firstly, it can be caused by convergence on unwanted local maxima in the raw function. Runs do not always locate a maximum with a fitness above the solution threshold—so the peak found does not count as a solution. Convergence on local maxima, again, is a standard problem with all GAs, and obviously we would like to discover a GA which does not have this fault. Since such a GA is unlikely to be invented, it is worthwhile to make use of the knowledge we gain from locating a local maximum. Our sequential niche algorithm does just this, and ensures that subsequent runs in the same sequence do not converge again on local maxima already found. We thereby turn a deficiency in the GA to our advantage. This may not be the *most* efficient way to find the all the maxima of interest, but at least we can be sure that successive runs in a sequence have a decreasing likelihood of getting stuck on uninteresting maxima of the raw function.

The second cause is that a run may converge on a local maximum in the *modified* fitness function, as discussed in Section 5.1.1.

The third cause is incompleteness, requiring iteration of the algorithm. This effect would be eliminated if the problem of incompleteness was solved.

## 5.3 Future Work on Niche Radius

In general, it is impossible to choose a single value of niche radius which will be correct for all maxima of interest, since niches (i.e. maxima) in a problem will *not*, in general, be all the same size. To properly address the question of niche radius, we need to allow the niche radius to be sized *individually* for each maximum.

For Deb's (1989) sharing algorithm, assumptions about niche size, shape and distribution have to be made in advance. He assumes that each niche is of the same size and shape (hyperspherical), and that niches are evenly distributed throughout the search space. Clearly, these are reasonable assumptions if we know nothing about the topography of the search space.

An advantage of the sequential niche technique, however, is that information about the topography of the fitness function is steadily built up from run to run. We may begin with certain assumptions, but we do not have to stick to them all the way through.

Rather than assuming a fixed size and shape for all niches, a better algorithm would allow individual parameters to be stored for each niche. Rather than assuming each niche to be hyperspherical, a hyperellipsoidal shape would be more general—in effect giving a different "radius" in each dimension. As each peak is discovered, its size, in each dimension, could be estimated, and recorded along with the peak. This would make the algorithm faster and more accurate. Methods for estimating niche parameters are a subject of further research by the authors.

# 6 Conclusions

We have described a new technique for locating all maxima of a multimodal function. It involves multiple runs of a GA, each finding one maximum. After each maximum has been found, the fitness function is modified to prevent subsequent runs re-searching nearby regions of the problem space.

An advantage of the sequential niche algorithm is that it represents a simple extension to existing unimodal optimization techniques. It is an "add-on" technique which can be used with any kind of GA, or other techniques such as simulated annealing.

Compared with simple (blind) iteration, the sequential niche technique is able to avoid re-discovering solutions which have already been found. This makes the sequential niche technique faster by a factor of at least $\log p$, where $p$ is the number of peaks in the function.

Fitness sharing, a more sophisticated technique introduced by Goldberg and Richardson (1987), and investigated further by Deb (1989), suffers from a number of problems. If the function has many peaks, a large population is required for the GA. In addition, the simplistic concept of a single-valued niche radius restricts the application of the sharing algorithm to problems where the maxima are approximately equally spread throughout the search space.

The sequential niche algorithm we have presented performs well on the functions which Deb (1989) used to test his sharing algorithm. Although it suffers from similar problems to the sharing algorithm, smaller population sizes can be used. This reduces the time complexity—by a factor of approximately $10(1 + \log p/\log n)$ when $p$ is large, assuming that the number of generations to convergence is $O(\log n)$ (where $n$ is the population size, and $p$ is the number of peaks in the fitness function).

We have explored the issue of derating functions, and our findings seem to be relevant to fitness sharing functions also. The exact shape of the derating curve is not important, the only essential factor being that it should be reasonably concave; for example, a power-law function of order 2 or 4. Concavity is required in order to prevent any introduced peaks from being too large. The niche radius used is important for a similar reason. The problems which arise from an incorrect choice of niche radius have also been reported with the fitness sharing algorithm. However, their origin is easier to visualize by considering the modified fitness function. The use of different niche radii for each niche would bring improvements.

The sequential niche technique may prove to be a useful method when applied to practical problem solving which requires the location of multiple optima of multimodal functions.

# 7 References


Ackley, D. H. (1985). A connectionist algorithm for genetic search. In *Proceedings of the First International Conference on Genetic Algorithms* (pp.121–135). Lawrence Erlbaum Associates.

Ackley, D. H. (1987). An empirical study of bit vector function optimization. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing* (Ch. 13). Pitman.

Davidor, Y. (1991). A naturally occurring niche and species phenomenon: the model and first results. In *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp.257–263). Morgan Kaufmann.

Davis, L. (1991). *Handbook of genetic algorithms.* Van Nostrand Reinhold.

Deb, K. (1989). *Genetic algorithms in multimodal function optimization.* Masters thesis, TCGA Report No. 89002. The University of Alabama, Department of Engineering Mechanics.

Deb, K., & Goldberg, D.E. (1989). An investigation of niche and species formation in genetic function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms* (pp.42–50) Morgan Kaufmann.

Deb, K. & Goldberg, D.E. (1990). Natural Frequency calculation using Genetic Algorithms. *Developments in Theoretical and Applied Mechanics*, **XV**, pp.94–101.

Deb, K., & Goldberg, D.E. (1991). *Analyzing deception in trap functions.* IlliGal Report No. 91009. University of Illinois at Urbana-Champaign, Department of General Engineering.

DeJong, K. (1985). Genetic Algorithms: A 10 year perspective. In *Proceedings of the First International Conference on Genetic Algorithms*, (pp.169–177). Lawrence Erlbaum Associates.

Glover, F. (1989). Tabu Search. *ORSA Journal on Computing*, **1**(3), pp.190–206.

Goldberg, D.E. (1989a). *Genetic Algorithms in search, optimization and machine learning.* Addison Wesley.

Goldberg, D.E. (1989b). Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms* (pp.70–79). Morgan Kaufmann.

Goldberg, D.E. (1992). Construction of high-order deceptive functions using low-order Walsh coefficients. *Annals Math. A.I.*, **5**, pp.35–48

Goldberg, D.E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In G.J.E. Rawlins (Ed.), *Foundations of Genetic Algorithms*, (pp.69–93). Morgan Kaufmann.

Goldberg, D.E., Deb, K., & Horn, J. (1992). Massive Multimodality, Deception, and Genetic Algorithms. In R. Männer and B. Manderick (Eds.), *Parallel Problem Solving from Nature, 2* (pp.37–46). North-Holland.

Goldberg, D.E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms* (pp.41–49) Lawrence Erlbaum Associates.

Grosso, P.B. (1985). *Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model.* PhD thesis. The University of Michigan, Ann Arbor.

Jolley, L.B.W. (1961). *Summation of series.* Dover.

Mühlenbein, H. & Schlierkamp-Voosen, D. (1993). Predictive Models for the Breeder Genetic Algorithm. I. Continuous Parameter Optimization. *Evolutionary Computation*, **1**

Oei, C.K., Goldberg, D.E., & Chang, S. (1991). *Tournament selection, niching and the preservation of diversity.* IlliGal Report No. 91001. University of Illinois at Urbana-Champaign, Department of General Engineering.

Smith, R.E., Forrest, S., & Perelson, A.S. (1992). *Searching for diverse, cooperative populations with genetic algorithms.* TCGA Report No. 92002. The University of Alabama, Department of Engineering Mechanics.