Understanding Interactions Among Genetic Algorithm Parameters

Kalyanmoy Deb and Samir Agrawal
Kanpur Genetic Algorithms Laboratory (KanGAL)
Indian Institute of Technology, Kanpur
Kanpur, PIN 208 016, India
E-mail: {deb,samira}@iitk.ac.in

Abstract

Genetic algorithms (GAs) are multi-dimensional and stochastic search methods, involving complex interactions among their parameters. For last two decades, researchers have been trying to understand the mechanics of GA parameter interactions by using various techniques—careful 'functional' decomposition of parameter interactions, empirical studies, and Markov chain analysis. Although the complexities in these interactions are getting clearer with such analyses, it still remains an open question in the mind of a new-comer to the field or to a GA-practitioner as to what values of GA parameters (such as population size, choice of GA operators, operator probabilities, and others) to use in an arbitrary problem. In this paper, we investigate the performance of simple tripartite GAs on a number of simple to complex test problems from a practical standpoint. Since in a real-world situation, the overall time to run a GA is more or less dominated by the time consumed by objective function evaluations, we compare different GAs for a fixed number of function evaluations. Based on probability calculations and simulation results, it is observed that for solving simple problems (unimodal or small modality problems) the mutation operator plays an important role, although GAs with the crossover operator alone can also solve these problems. However, the two operators (when applied alone) have two different working zones for the population size. For complex problems involving massive multi-modality and misleadingness (deception), the crossover operator is the key search operator. Based on these studies, it is recommended that when in doubt, the use of the crossover operator with an adequate population size is a reliable approach.

Keywords: Population sizing, multi-modal functions, deceptive functions, crossover-based GAs, mutation-based GAs.

1 Introduction

There exists a plethora of studies investigating the interactions among different genetic algorithm (GA) parameters for successful application of GAs. This is rightly so, because GA parameters (such as population size, choice of GA operators, operator probabilities, representation of decision variables, and others) interact in a complex way. More importantly, their interactions are largely dependent on the function being optimized (Hart and Belew, 1991). Since these interactions are complex and a complete analysis of all their interactions is difficult to achieve, researchers have used different analysis tools. Some studies carefully isolated interactions between two important parameters and understand their pair-wise effect on GA's performance. These isolated studies are worthwhile and have provided useful guidelines for choosing GA parameters, such as population size (Goldberg, Deb, and Clark, 1992; Harik et al., 1997) and control maps for operator probabilities (Goldberg, Deb, and Thierens, 1992; Thierens and Goldberg, 1993). In order to observe the interactions of various GA parameters, empirical studies have also been used (De Jong, 1975; Eshelman and Schaffer, 1993; Schaffer et al., 1989; Wu, Lindsay, and Riolo, 1997). To study the dynamics of these interactions, more sophisticated stochastic models using Markov chains have also been developed and analyzed (Chakraborty, Deb, and Chakraborty, 1996; Nix and Vose, 1992; Suzuki, 1993; Vose, 1992).

Based on these studies, the following salient observations can be made:

1. An optimal mutation probability is dependent on the representation being used (Tate and Smith, 1993). Similar arguments are also made for the crossover operator, where it is highlighted that an optimal operator is largely dependent on the underlying coding used to represent decision variables (Battle and Vose, 1990; Radcliffe, 1991; Kargupta, Deb, and Goldberg, 1992).

2. The effect of crossover and mutation can be interchanged by using a suitable coding transformation (Culberson, 1994). However, the study does not mention anything about the cost (in terms of function evaluations) needed to find a suitable coding transformation. Although the argument is right in its own sake, it does not help much in terms of deciding which operator to give importance to, from a practical standpoint.

3. Crossover is useful in problems where building block exchange is necessary (Eshelman and Schaffer, 1993; Goldberg, Deb, and Clark, 1992, Spears, 1993). The use of mutation may destroy already-found good information and therefore may not be suitable in such problems. With this in mind, it is then suggested that GAs may work well with a large crossover probability and with a small mutation probability (Goldberg, 1989; Schaffer et al., 1989)

In this paper, we investigate the effect of three GA parameters (population size, crossover probability, and mutation probability) on GA's performance from a practical standpoint. In real-world search and optimization problems, the most time-consuming task is the evaluation of function values. From a practical standpoint, we view the use of a GA to solve a search and optimization problem as follows. Given a problem and a time frame to obtain a solution,

what GA parameter settings must be used to get a good solution (hopefully the true optimum solution or a near true-optimal solution). Since the overall time to run a GA is more or less proportional to the number of function evaluations used, we set the number of function evaluations fixed for all competing GAs. When such a bound on function evaluations is desired for comparing different GAs, most earlier GA parameter studies are not applicable, because in those cases GAs were run till a solution close to the true optimal solution is found or till all population members converge to a small neighborhood or a fixed predefined number of generations (irrespective of population size) is elapsed. We feel that the study of comparing performance of different GAs for a fixed number of function evaluations is practical and useful from GA's applicability in real-world scenarios.

Since a GA's performance (and its parameter setting) depends on the function being solved, we consider five different test functions involving two unimodal functions, one four-peaked function, one massively multi-modal function, and one deceptive function. The performance of different GAs obtained from simulation results are explained using probability calculations. Various levels of mathematical rigor—sometimes borrowing results from published literature and sometimes resorting to Markov chain analysis—have been adopted. The analysis supports the simulation results obtained and shows several insights into the complex working mechanism of a simple genetic algorithm.

2 GAs in Practice

One common problem faced by the GA practitioners is the fixation of GA parameters. Because of the lack of sound theoretical studies specifying 'control maps' for a successful GA run, researchers in the field still resort to parametric studies to determine suitable GA parameters for the problem at hand. In most studies, a fixed representation scheme and a fixed set of operators are chosen and a parametric study is performed with three parameters—population size (N), crossover probability (p_c), and mutation probability (p_m). It is important to note that the GA's performance is largely affected by the representation scheme used to code the decision variables. Due to the lack of knowledge of a good representation scheme in an arbitrary problem, a great deal of effort has been spent to use a flexible representation scheme so that GAs can evolve an efficient representation on the fly (Goldberg, Korb, and Deb, 1989; Kargupta, 1996; Harik, 1997). In this study, we do not address this so-called *linkage* problem and always use a *tight* representation scheme in solving all test problems.

When such parametric studies are to be made, it is important that different GAs must be allocated the same number of total points to search from. Thus, if a total of S number of function evaluations are allocated, a GA with a population size of N must be run for a maximum of $T = S/N$ number of generations, because in each generation N functions are evaluated[1]. Here, we also do not take any special care to not count an individual appearing more than once either in one single population or temporally in subsequent generations.

The minimum number of function evaluations S that must be assigned for a successful application of GAs depends on the function being solved. It is a intuitive that if the

[1]It is noteworthy that, if properly implemented, a GA with a crossover probability of p_c and zero mutation, changes only $p_c N$ strings in each generation, thereby increasing the total number of generations to $S/(p_c N)$. Similarly, for a GA with zero crossover probability and a mutation probability of $p_m < 1/\ell$ per locus, the expected number of generations would be $S/(p_m \ell N)$. However, we ignore such special implementations in this study.

function is *difficult* for GAs to solve, GAs must be allowed more number of points to search from. Although there exists no clear study specifying what would cause GA-difficulty, the following few factors have been suggested elsewhere (Goldberg, 1993; Horn, Goldberg, and Deb, 1994):

1. Multi-modality
2. Deception
3. Isolation
4. Collateral noise

Multi-modality causes difficulty to any search and optimization method, because of the presence of a number of false attractors. For some algorithms (such as gradient-descent methods), only a few modalities may cause enough difficulty. For some algorithms, the difficulty arises only when the number of modalities are huge (we refer to such problems as massively multi-modal problems in this study). Deception causes difficulty to GAs because in these functions lower-order schema information is misleading, thereby causing a GA to get attracted to sub-optimal solutions. Isolation (like the needle-in-the-haystack problem) causes difficulty for any search and optimization algorithm, because in these problems, no information is usually available for the search to proceed in any direction. The collateral noise in functions *hides* the presence of a good sub-solution in a solution, thereby causing a search and optimization algorithm using smaller sample sizes to not detect and emphasize the correct building blocks needed to solve the problem to global optimality (Rudnick and Goldberg, 1991). It is clear that some of the above difficulties are related to each other and more than one of them may be present in an arbitrary problem. In the test functions chosen in this study, we explicitly introduce first two of the above difficulties in two functions.

2.1 Test Functions

2.1.1 Unimodal functions

We choose two unimodal functions, each having only one optimum solution. The first function is unimodal in the Hamming space and the second function is unimodal in the decoded parameter space.

The one-max function (f_1) is used as the unimodal function in the Hamming space. We choose a two-variable unimodal function (Himmelblau's function), often used as a test function in the optimization literature (Deb, 1995; Reklaitis, Ravindran, and Ragsdell, 1983).

$$f_2(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2. \quad (1)$$

The search space is considered in the range $0 \leq x_1, x_2 \leq 6$, in which the above function has a single minimum point at $(3, 2)$ with a function value equal to zero. This function is a easy for most search algorithms, even gradient descent methods do very well on this function (Deb, 1995).

2.1.2 Four-peaked function

This function is the same as the previous one, but the ranges for x_1 and x_2 are extended to $[-6, 6]$. The function has a total of four minima, one in each quadrant. All minima have

function values equal to zero. In order to make one of them the global minimum, we add a term to the above function, which causes the point $(3, 2)$ to become the global minimum with a function value equal to zero:

$$f_3(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 + 0.1(x_1 - 3)^2(x_2 - 2)^2. \quad (2)$$

This function causes difficulty for many classical search methods, including gradient descent methods, where the performance largely depends on the chosen initial solution (Deb, 1995).

2.1.3 Massively multi-modal function

We choose a 10-variable Rastrigin's function, which is also studied extensively in the GA literature:

$$f_4(x_1, \ldots, x_{10}) = 200 + \sum_{i=1}^{10} x_i^2 - 10\cos(2\pi x_i). \quad (3)$$

Each variable x_i lies in the range $[-6, 6]$. The function has a global minimum at $x_i = 0$ with a function value equal to zero. This function has a minimum at every point where the cosine term is one. Thus, there are a total of 13^{10} or $1.38(10^{11})$ minima, of which 2^{10} minima are close to the global minimum point. This function tests an algorithm's performance in handling massive multi-modality, one of the difficulties that a search algorithm often encounters.

2.1.4 Deceptive function

Ten 4-bit trap functions have been used to construct a 40-bit problem, as follows:

$$f_5 = \sum_{i=1}^{10} g(u_i), \quad (4)$$

where the function $g(\cdot)$ is a function of *unitation*[2] (u), shown below:

u	0	1	2	3	4
$g(u)$	3	2	1	0	4

This function has a total of 2^{10} local maxima, of which one solution (the string with all `1`s) is the global maximum, which has a function value equal to 10×4 or 40. This function is difficult to solve because low-order building blocks corresponding to the deceptive attractor (string of all `0`s) is better than that of the global attractor.

2.2 GA Operators and Performance Measure

All of the above test problems are attempted to solve using binary-coded GAs. In all simulations presented here, we use a binary tournament selection without replacement, where two individuals are picked from a shuffled population and the better is copied in the mating pool (Goldberg and Deb, 1990). We use a standard single-point crossover operator with a probability p_c. We use a mutation clock operator (Goldberg, 1989). With this operator, once a mutation is performed, the next bit to be mutated is decided based on an

[2] Unitation is a measure of number of 1s in the 4-bit string.

exponential distribution having a mean p_m. This implementation drastically reduces the random number generations by a factor equal to $1/p_m$, on an expectation. Since we have chosen a small mutation probability, this leverage in time complexity is significant in our studies.

In all the above test functions, the global optimal solution is known a priori. Thus, the success of a GA simulation can be measured whether a solution within ϵ-neighborhood[3] of the global optimal solution is found with a pre-specified number of function evaluations S. It is important to mention that the outcome of a GA simulation depends on the choice of S. If a small number of function evaluations (small S) is assigned, no GAs may have enough function evaluations to work from and all GAs will perform poorly. If a large number of function evaluations (large S, comparable to the search space) is assigned, all GAs will perform equally well. To really compare different GAs, a reasonable amount of S must be chosen. For each problem, we either choose S based on theoretical limits from the literature or choose S such that GAs have a reasonable chance of finding a solution within ϵ-neighborhood of the optimal solution. This is done by choosing an appropriate ϵ value.

In order to reduce the bias of the initial population, we simulate GAs from 50 different initial populations[4] and observe the number of occasions (M) the GA has found the true optimal solution with a maximum of S function evaluations. The performance measure ζ is then defined as the proportion of successful GA simulations, or $\zeta = M/50$. A GA run with a population size of N is terminated if any one of the following criteria is satisfied:

1. A solution within ϵ-neighborhood of the global optimal solution is found, or
2. A maximum of $T = S/N$ number of generations have been elapsed.

Although a maximum of S function evaluations are allowed in any run, some GAs may require fewer function evaluations (F) to solve the problem. In order to investigate the utilization of allowed function evaluations, we have defined an *Unuse Factor* (U) as follows:

$$U = 1 - \frac{F}{S}. \tag{5}$$

When a GA does not find the true optimal solution with S function evaluations, F is set to S and U becomes equal to zero. Thus, a plot of the Unuse factor versus generation will justify the choice of S in any problem.

3 Unimodal Functions

3.1 Onemax Function

We take a 32-bit Onemax problem with $S = 500$. Mühlenbein (1992) showed that with a greedy selection operator and a bit-wise mutation operator, $O(\ell \log \ell)$ function evaluations are required to solve an ℓ-bit Onemax problem. As per estimates given in Goldberg, Deb,

[3] If for a solution x, each variable x_i is within ϵ_i from the global optimal solution, the solution x is defined to be in the ϵ-neighborhood of the global optimal solution.

[4] Along with the other performance measure (the Unuse factor) which shows whether a GA run is successful or not, we felt 50 runs are adequate in each case.

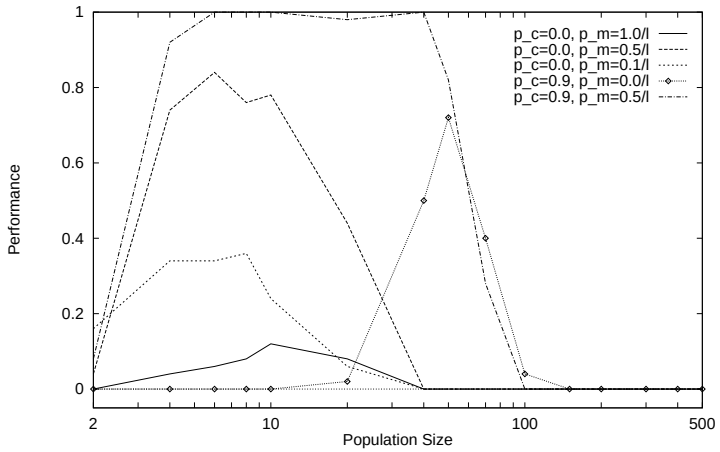

Figure 1: Performance of GAs for different GA parameter settings for the Onemax problem.

and Clark (1992), this problem requires $O(\ell^{1.7})$ function evaluations (for $\ell = 32$, the recommended S is about 1,000 till all population members convergence to an identical solution) with crossover and tournament selection alone. In this study, we have chosen $\epsilon = 0$. Our choice of $S = 500$ is well within these estimates.

Figure 1 shows the performance of GAs with different GA parameter settings. Let us first investigate the performance of GAs with selection and mutation operators alone. GAs with three different p_m values show that $p_m = 0.5/\ell$ works better than other two p_m values. But it is clear from all three performance characteristics that the performance of GAs is poor for very low and very large population sizes. Before we discuss why GAs behave this way for extreme population sizes, let us observe how different GAs have utilized the allocated function evaluations.

Figure 2 shows the mean Unuse Factor in 50 runs for different GA simulations. It is clear that for successful GAs, this factor is large, thereby meaning that smaller function evaluations are required to solve the problem. Whereas for unsuccessful GAs, all allowed function evaluations (S) are spent.

3.1.1 Large populations

When the population is large, the diversity in the initial random population is large and the best solution in the population is expected to be close (in both Hamming and decoded parameter space) to the optimal solution. Since the Onemax function is defined in the Hamming space, we consider the Hamming distance of a string from the optimal string as a measure of distance. Therefore, the Hamming distance of the best string in a population gives a rough estimate of the minimum number of generations needed to reach to the optimum string, when a mutation probability per locus of less than $1/\ell$ is used. Then, if the number of allowed generations S/N is less than the smallest Hamming distance in the initial population, GAs will obviously not work. It then becomes important to find how the expected Hamming distance (say K) of the best string in a population varies with the population size.

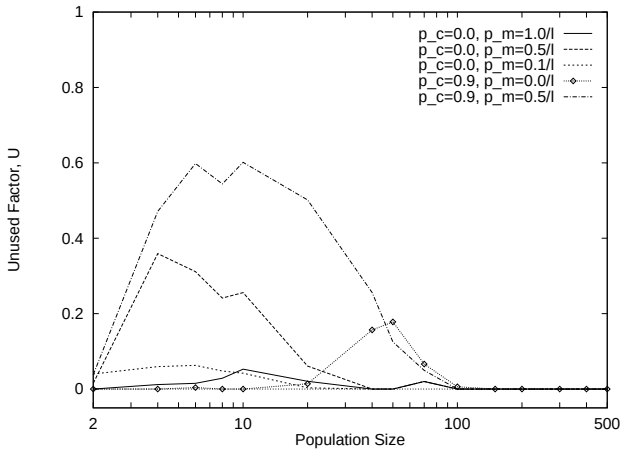

Figure 2: Mean Unuse factor U versus population size.

Let us denote k be the Hamming distance of the best string from the optimal string in an initial random population.

The probability of choosing a string of Hamming distance k from the optimal string is

$$p = \frac{\binom{\ell}{k}}{2^\ell}. \tag{6}$$

The probability of choosing a string which is at a Hamming distance greater than k is

$$q = \frac{\sum_{j=k+1}^{\ell} \binom{\ell}{j}}{2^\ell}. \tag{7}$$

Then, the probability $p(k,i)$ of having i strings at a Hamming distance k and rest $(N-i)$ strings at Hamming distances larger than k is as follows:

$$p(k,i) = \binom{N}{i} p^i q^{N-i}. \tag{8}$$

Summing all these probabilities for $i = 1, 2, \ldots, N$, we have the overall probability of having the best strings at a Hamming distance k from the optimal string in a population of size N:

$$\begin{aligned} P(k) &= \sum_{i=1}^{N} p(k,i), \\ &= (p+q)^N - q^N. \end{aligned} \tag{9}$$

The expected value K of the Hamming distance of the best strings in a population is then calculated as follows:

$$K = \sum_{k=0}^{\ell} k P(k). \tag{10}$$

This value is difficult to calculate exactly. Thus, we calculate this numerically for $\ell = 10$, 20, and 32. Figure 3 shows K for different population sizes. For $N = 1$, this expected

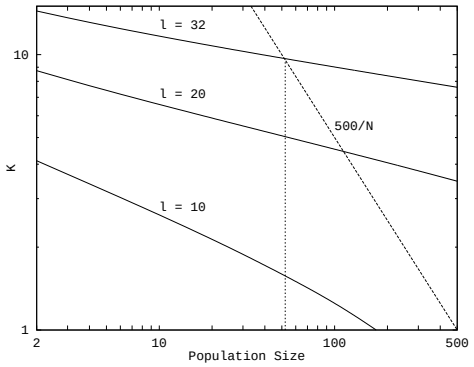

Figure 3: Expected value of the shortest Hamming distance from the optimal string in a random population of different sizes.

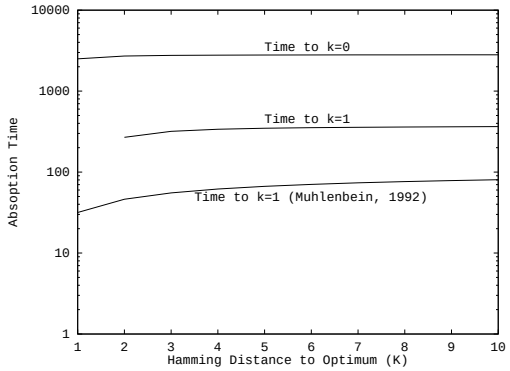

Figure 4: Mean absorption time estimates for different Hamming distances for $\ell = 10$.

shortest Hamming distance is exactly $\ell/2$ and as the population size increases this value reduces almost as $O(N^{-a})$, where a depends on the string length. For string lengths of 10, 20 and 32, the values of a are 0.399, 0.157, and 0.125, respectively. In contrast, the number of generations allowed in a GA run varies as $O(N^{-1})$, which reduces at much higher rate than the above estimate. Thus, for large population sizes, adequate generations are not allowed and GA's performance drops.

Figure 3 also shows the variation of the allowed number of generations $T = 500/N$. For $\ell = 32$ plot, it predicts that for population sizes more than about $N = 50$, mutation-based GAs should not perform well. Figure 1 verifies this fact by showing that all mutation-based GAs did not find the true optimum in any run when the population size is more than 50.

We now investigate why GAs did not perform well with very small populations.

3.1.2 Small populations

It has been discussed elsewhere (Mühlenbein, 1992) that for Onemax problems, the most difficult task is to move from strings having Hamming distances one to the optimal string. This is because the transition probability of this movement for a single string is very small. However, the cumulative probability of this movement for a number of strings could be large. Thus, it is intuitive that there exists a minimum threshold population size, smaller than which GAs will have difficulty reaching the optimum string. In the following, we first calculate the transition probability of moving to the optimal string from any string and then estimate the expected transition time based on Markov chain analysis. Thereafter, we show how this transition probability reduces with population size, giving rise to the concept of a minimum population size below which GAs are not expected to work.

The probability of transition of a string having a Hamming distance of k from the optimal string to a string of Hamming distance k' from the optimal string can be written as follows:

$$p_{k,k'} = \sum_{a=\max(0,k'-k)}^{\min(k',\ell-k)} \binom{k}{k'-a} \binom{\ell-k}{a} p_m^{k-k'+2a} (1-p_m)^{\ell-k+k'-2a}. \qquad (11)$$

All possibilities of different mutations are considered so that a bits are mutated in the $(\ell-k)$ genes which are the same as that in the optimal string and $(k-k'+a)$ bits are mutated in the k genes which are dissimilar with the optimal string. The above equation is valid for all k and k' varying from 0 to ℓ. Forming the complete transition matrix using equation 11, we estimate the mean absorption time (mean time required to move from Hamming distance of k to 0) by using standard procedures. Figure 4 shows this mean time for different k values.

It is important to note that these time estimates are for the mutation operator only. Thus, they correspond to the genetic drift. Although these values are large, the important point is that the mean transfer time required to move from strings of Hamming distance more than one to zero is marginally larger than that for the strings of Hamming distance one. This is also evident from the 'Time to $k = 1$' line shown in the figure, which shows the mean transition times to move from $k > 1$ strings to $k = 1$ strings under mutation alone[5]. The figure also shows the transition time for returning to the optimal string using Mühlenbein's (1992) (1+1, m, hc)-algorithm[6], which uses a greedy selection scheme and thus will give smaller estimates than for GAs with a tournament selection and bit-wise mutation. Nevertheless, both our drift estimates and Mühlenbein's estimates show that the transition from strings of Hamming distance of one to the optimum is the most difficult task. For example, consider Figure 4 and two different scenarios—the transition time for strings of Hamming distance $K = 2$ from the optimal string (a) to reach the optimal string and (b) to reach a string having Hamming distance $K = 1$ from the optimal string. In the first case, the mean time is about 2,712 generations, whereas in the second case it is only 269 generations, thereby taking most of the generations in trying to reach the optimal string from a string which is just one bit different from the optimal string.

Since the most difficult transition is from $k = 1$ to $k = 0$, we calculate the probability of this transition (any one bit is mutated and rest $(\ell - 1)$ bits are unchanged) in the following (which also follows from equation 11 with $k = 1$ and $k' = 0$):

$$p(k = 1, k' = 0) = p_m(1 - p_m)^{\ell-1}. \tag{12}$$

With a population size of N, the probability of having this transition in at least one string is as follows (with the assumption that all strings are at a Hamming distance one from the optimal string):

$$P = 1 - (1 - p(k = 1, k' = 0))^N. \tag{13}$$

It is interesting to note that this probability P is small for small N and large for large N. Thus, with a small population size, the probability of moving towards the optima in one generation is small. This means that under mutation operator, a small population has a small chance of taking a step in the right direction. When a wrong step is taken, it requires a transition time to come back to the optima, thereby causing GAs to spend a considerable number of function evaluations in visiting other states before returning to the optimum string. We verify this argument by finding the number of generations and function evaluations needed to reach the optimal string from a population initialized with strings at a

[5]This plot is generated by redoing the Markov chain analysis assuming $k = 1$ as the absorbing state.

[6]This algorithm works with one solution. If a random mutation with probability p_m is successful, the mutated string is accepted as a new parent; otherwise the original string is retained. This algorithm is supposed to work well on the Onemax problem, but may not be efficient for complex problems, as the study (Mühlenbein, 1992) showed for the (k, ℓ) deceptive problems.

Hamming distance one from the optimal string, for a 32-bit Onemax problem. We run GAs from 1,000 different initial populations of strings of Hamming distance one from the optimal string and continue each run till one copy of the optimal string is found. Figure 5 shows that GAs require a large number of generations to return to the optimum string for smaller population sizes. Although with a small population size the number of function evaluations per generation is less, Figure 6 shows that there exists a minimum population size at which GAs require the smallest total number of function evaluations to find the optimum string. With a fixed number of allowed function evaluations, GAs thus perform poorly for very small population sizes. It is worthwhile to mention here that the above argument can also

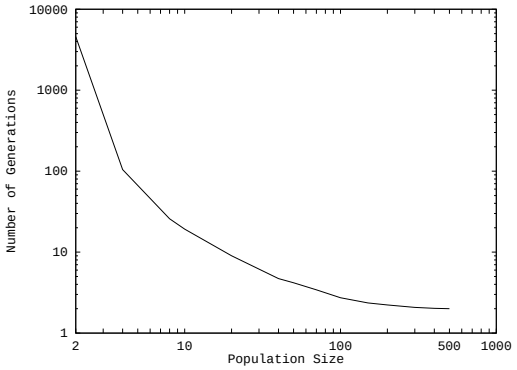

Figure 5: Number of generations needed to find the optimal string for the onemax problem.

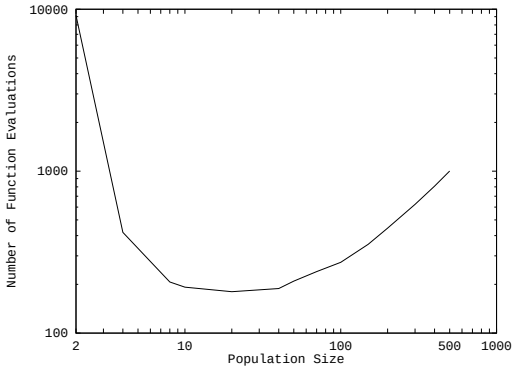

Figure 6: Number of function evaluations needed to find the optimal string for the Onemax problem.

be used to explain why micro-GA applications (Krishnakumar, 1989) do not perform well with very small populations (such as 2 or 3), whereas they perform well on simpler problems with population sizes 5 or more.

The above two subsections suggest that GAs with mutation and selection operators and with a small or a large population size will perform poorly. We also observe this behavior of GAs in the simulation runs for the Onemax problem.

3.1.3 Crossover alone

GA's performance with selection and crossover (no mutation) operators starts to improve only after a critical population size. This is because GAs with crossover as the only search operator require an adequate population size to allow correct schema processing to take place. Population sizes based on correct schema processing have been derived elsewhere (Goldberg, Deb, and Clark, 1992, Harik, et al., 1996) and experimental verification for the population sizing was also shown. Taking the cue from the earlier work, we explain why the performance starts to improve when the population is increased from a critical population size.

Considering schema competition among two competing schemata, Goldberg, Deb, and Clark (1993) showed that a minimum population size (N_s) is necessary to trigger correct building block processing:

$$N_s = 2c\kappa\frac{\sigma_M^2}{d^2}, \tag{14}$$

where c is the tail of the Gaussian distribution relating to the permissible error rate α, κ is the number of competing schemata, and σ_M^2/d^2 is the inverse of the signal-to-noise in the underlying problem. The factor c varies with α as $\alpha = \exp(-c/2)/\sqrt{2\pi c}$. That study also showed that if the population is sized for a error rate α, GAs performance ζ is related to $\zeta = 1 - \alpha$. It can then be argued that if a population size N smaller than N_s is used, such that $N = \xi N_s$, then the effective factor is $c' = \xi c$. With this c', the performance measure is as follows:

$$\zeta = C\left(1 - \frac{\exp(-c\xi/2)}{\sqrt{2\pi c\xi}}\right), \tag{15}$$

where C is a normalization constant to adjust performance measure ζ at N_s. This equation suggests that as the population size increases, the performance measure asymptotically approaches one. Using equation 14, the required population size is $31c$ for the Onemax problem. With 90% confidence level, $c = 1.67$ and the required population size is about 52. This population estimate gives an adequate population size needed for GAs to make correct decisions in the first generation. Although we did not allow our GAs to run till the complete population converges, Figure 1 shows that GAs with about $N = 50$ work the best.

3.2 Himmelblau's Function

We use 12 bits to code each variable. Thus, a complete string is 24 bits long. The total number of search points is 2^{24} or about 16.8 million. For a successful GA run, we choose $\epsilon_1 = \epsilon_2 = 0.01$. This requires a random search method to compute $\frac{(6-0)}{0.02} \times \frac{(6-0)}{0.02}$ or 90,000 function evaluations to find a solution within ϵ-neighborhood of the minimum solution. We allow only 3.33% of this amount (or $S = 0.033 \times 90,000 = 3,000$) to a GA to find a solution in the ϵ-neighborhood (this is only 0.018% of the entire search space).

Figure 7 shows that GAs with mutation as the only search operator can also find a near-optimal solution in most simulations. However, the performance degrades when a smaller

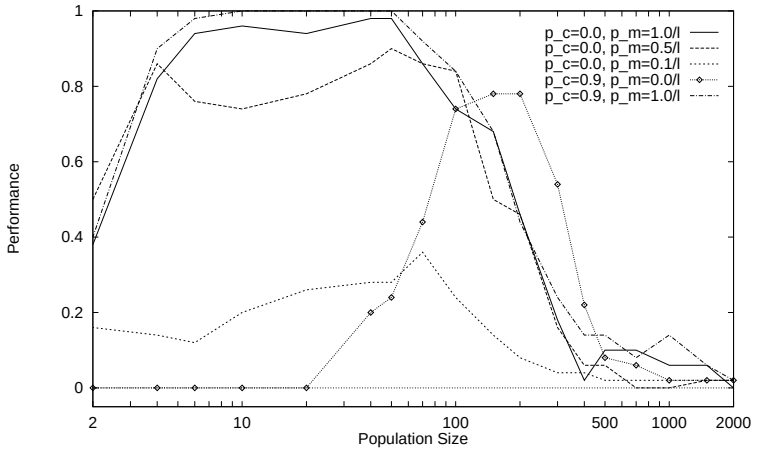

Figure 7: Performance measure versus population size for Himmelblau's unimodal function.

mutation probability is used. Like the Onemax problem, GAs perform poorly for very small and large population sizes. GAs with only crossover operator as the search operator (with

$p_c = 0.9$) start to perform with larger population sizes, where adequate schema processing is possible in favor of the optimum solution. GAs with all three operators perform better than other GAs.

The Unuse Factor U for this function is plotted in Figure 8. Once again, the figure shows that a lesser number of function evaluations are needed in GAs which worked successfully in most of the runs. Both Figures 7 and 8 suggest that GAs with all three operators not only perform the best but also perform with highest efficiency in terms of utilizing the allocated number of function evaluations.

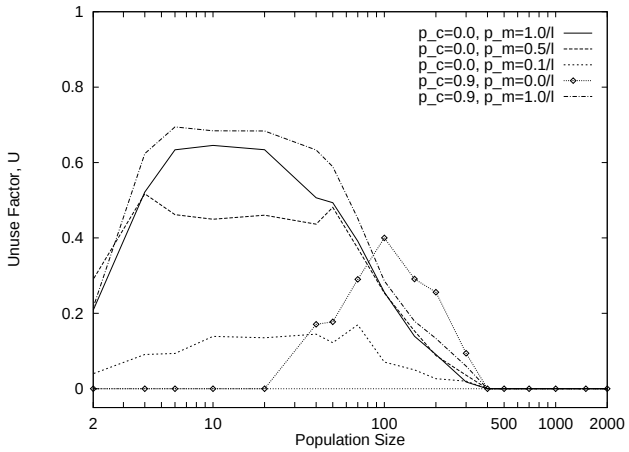

Figure 8: Unuse factor U for Himmelblau's unimodal function.

4 Four-peaked Function

We use 13 bits to code each variable, thereby making the total string length equal to 26. For termination criterion, we choose $\epsilon_1 = \epsilon_2 = 0.01$. We also set the maximum number of function evaluations equal to 2.5% of what a random search method would take, or $S = 9,000$ (this is only 0.013% of the entire search space).

Figure 9 shows the performance of various GAs with different population sizes. We observe that with selection and mutation operators alone, GA's performance is similar to that in the unimodal function. For very small and very large population sizes, GAs again perform poorly.

With selection and crossover operators alone (no mutation operator), the performance is also similar to that in the unimodal function. GAs begin to perform well only after an adequate population size is used. Thereafter, the performance improves steadily. However, when population size is large the performance degrades due to small number of generations allowed.

To investigate what population size is adequate from a schema processing point of view, we use equation 14 for the following two important order-one schema partitions:

1. $x_2 \leq 0$ versus $x_2 > 0$, and

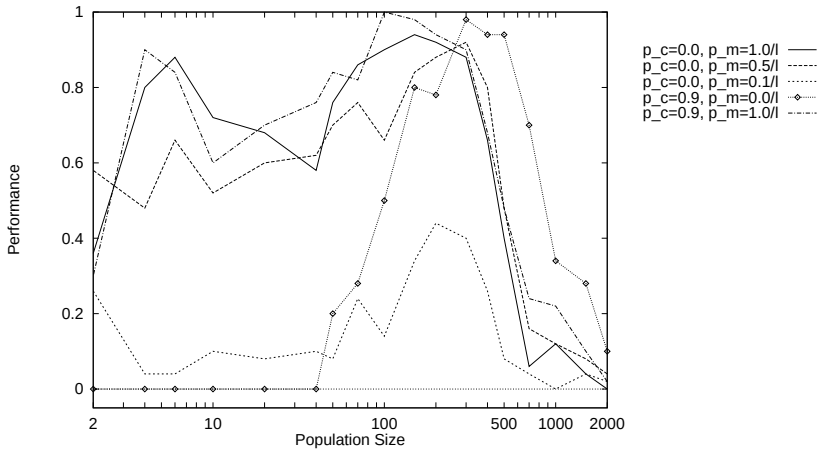

Figure 9: Performance measure of GAs for the four-peaked function.

2. $x_1 \leq 0$ versus $x_1 > 0$.

In terms of the string-coding, the first competition is between the following two schemata:

$$\begin{aligned} (-6 \leq x_1 \leq 6, -6 \leq x_2 \leq 0) &\equiv (**********, \texttt{0}**********) \\ (-6 \leq x_1 \leq 6, 0 < x_2 \leq 6) &\equiv (**********, \texttt{1}**********) \end{aligned}$$

We calculate the mean and variance of the function values by sampling 100,000 points each in the above two regions: $\bar{f}_0 = 332.6$, $\sigma_0^2 = 76851.4$, $\bar{f}_1 = 294.8$, $\sigma_1^2 = 98879.3$. Using the population sizing for $k = 1$, we have the following:

$$\begin{aligned} N_s &= 2c\frac{\sigma_0^2 + \sigma_1^2}{(\bar{f}_0 - \bar{f}_1)^2}. \\ &= 246c. \end{aligned} \tag{16}$$

With 90% confidence level, the required population size is 410.

Similarly, when the second schema competition is considered, the required population sizing is 260,550. This suggests that with a population size of 260,550 or more, GAs can make correct decisions in both schema partitions and GAs may not have any problem proceeding towards the right optimum. However, if a population of size less than 410 is used, GAs make mistakes in both decisions and may not be able to converge to the true optimum. Figure 9 shows that GAs with crossover operator alone needed a population size of about 300 to 500 to work most of the time. For population sizes greater than 500, GA's performance drops due to the lack of adequate number of generations.

When GAs are used with both crossover and mutation operators as the search operators, the performance improves for a wider range of population sizes. Figure 10 shows the variation of the Unuse Factor U with the population size. Once again, GAs require only about 40-60% of the allocated number of function evaluations wherever they work successfully.

In all these simulation plots where GAs are applied on simpler problems, an interesting feature of mutation-based GAs is observed. There seems to be two distinct ranges of population sizes (with a dip in performance measure in intermediate population sizes), where

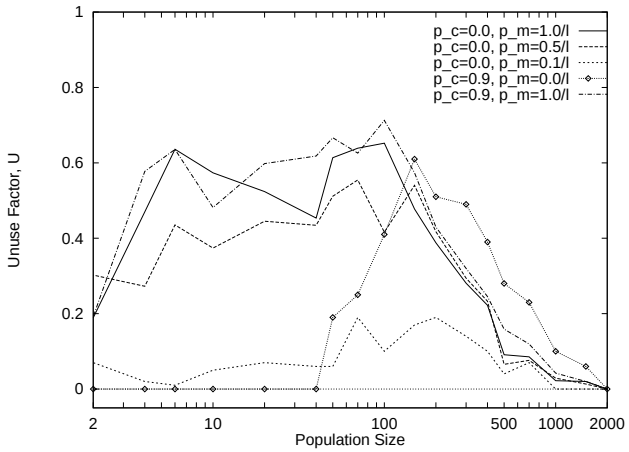

Figure 10: The Unuse factor U for the four-peaked function.

these GAs work the best. Similar performance trends are also observed by other researchers (Oates and Corne, 1998). Although the exact reason for peak performance at two distinct ranges of population sizes of mutation-based GAs is not studied here, it is conjectured that peak performance with smaller population size is purely due to the hill-climbing nature of mutation-based GAs and the peak performance at comparatively larger population size is due to the population effect having a size adequate to allow the needed diversity with the chosen mutation rate.

The similarity in the performances of GAs on all the above three functions (Onemax, Himmelblau's unimodal, and the four-peaked function) predicts the performance of GAs in a generic unimodal or in a simple function:

1. GAs with selection and mutation operators may perform well if an appropriate mutation probability and correct population size are used.

2. GAs with selection and crossover operators perform well at larger population size than those needed for GAs with selection and mutation.

3. GA's applicability increases to a wider range of population sizes with all three operators.

4. In general, mutation-based GAs work better than crossover-based GAs in these problems. The presence of the crossover operator enhances the performance of mutation-based GAs.

We now study the performance of GAs in more complex problems involving massive multimodality and deception.

5 Massively Multi-Modal Function

In the 10-variable Rastrigin's function, we use 13 bits to code each variable, thereby making the complete string equal to 130 bits. For our study, we use $\epsilon_i = 0.1$, so that no local

optimum lies within $\pm\epsilon_i$ from the global optimal solution. In the solution range of $[-6, 6]$, on average, a random search method will require $[(6-(-6))/0.2]^{10}$ or about $6(10^{17})$ function evaluations to find the global optimum. In the simulations here, we only allow a maximum of $S = 45,000$ function evaluations.

Figure 11 shows the performance measure with various GA parameter combinations. We

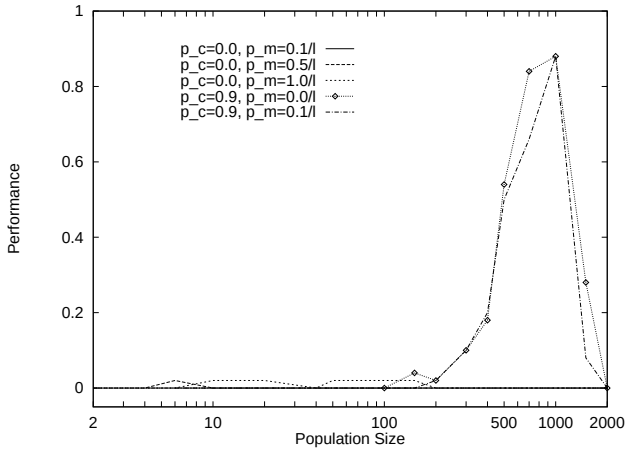

Figure 11: Performance measure of GAs for Rastrigin's massively multi-modal function.

observe that GAs with mutation alone perform miserably, whereas GAs with crossover operator alone finds the global optimal solution in almost 90% of runs. When the mutation operator is added (even with a small probability), GA's performance degrades. Since this problem has many local optima (in fact, there are a total 13^{10} or about $1.38(10^{11})$ local optimal solutions, of which only one is the global optimum), mutation in most cases destroys an already-found good solution. On the other hand, the crossover operator can combine good partial solutions together to form the optimal solution. It is interesting to note that a random initial population of size $(6-(-6))/0.2$ or 60 would have one expected copy of the optimal solution (with ϵ from the true optimum) for each subfunction. Thus, a population size of 60 or more will have one or more copies of the correct sub-solution somewhere in the population, on average. The task of the selection operator is to detect these good sub-solutions in the midst of noise from other subfunctions. This requires a population size which can again be computed using equation 14. Assuming that at a later stage[7] most of the population converges to the best two peaks, we find that the squared noise-to-signal ratio is about 52.0. Noting that there are 9 other subfunctions where the noise can come from, the population sizing becomes about $2c2(52 \times 9)$ or $1,872c$. With 90% confidence limit this sizing amounts to about 3,070. Since we allow only 15 generations with this population size and since the above population sizing is a conservative estimate (as also mentioned in the original study), a good performance is found around a population size of 1,000. Figure 12 shows the Unuse Factor U for different GA runs.

Thus, it is clear that a massively multi-modal function such as this one cannot be solved

[7]This assumption is not particularly bad, because the Rastrigin's function has an overall parabolic structure with a minimum at $x_i = 0$ and the toughest competition happens between the best two minima where the function value difference between the minima is only 1.0.

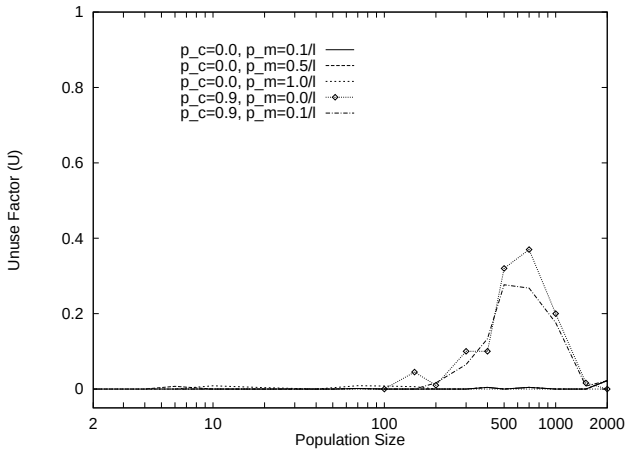

Figure 12: The Unuse factor versus population size for the massively multi-modal function.

using a mutation-based GA. In fact, a bit-wise mutation operator is found to be detrimental in our study. To solve such problems reliably, an adequate population size and a schema-preserving crossover operator are necessary. Although Mühlenbein and Schlierkamp-Voosen (1993) have shown that a breeder GA (BGA) can solve this function in $49n \log n$ (where n is the number of variables) function evaluations, BGA is a real-coded implementation and uses a line search which cannot be implemented for binary strings. Moreover, their crossover operator explicitly exploits the linear separability of subfunctions. Thus, such an approach may not perform well on a more generic function or to a rotated version of the above problem, as discussed in the following paragraph.

When identical experiments are conducted on a rotated Rastrigin's function (the variable vector is mapped into another variable vector by using a random orthonormal matrix before calculating the objective function value), a similar scenario emerged (Figure 13). This

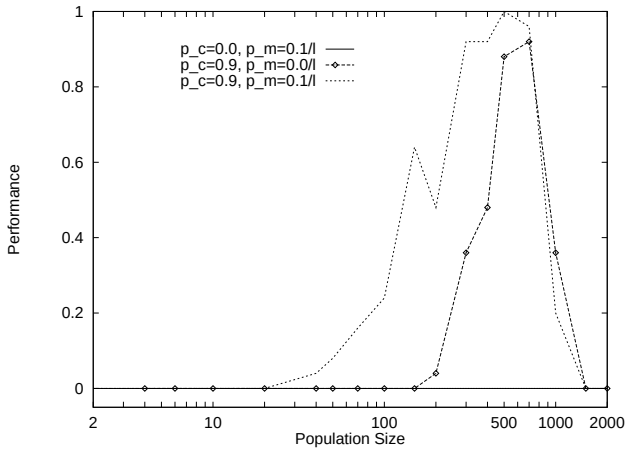

Figure 13: Performance measure of GAs for rotated Rastrigin's massively multi-modal function.

problem is more difficult to solve compared to the above function, simply because all variables are now related to each other. Each variable cannot be adjusted one at a time to reach to the optimum, instead all 10 variables must be adjusted simultaneously. Thus, in order to get reasonable performance measures, we have used $S = 90,000$ and $\epsilon_i = 0.1$ in these experiments. The performance of GAs with crossover operator alone is very similar to that in Figure 11. The performance of GAs with mutation operator alone (with mutation probabilities per locus equal to $0.1/\ell$, $0.5/\ell$ and $1/\ell$) cannot find a solution near the global optimum in any run, whereas the performance of GAs with both crossover and mutation operators is somewhat better than that in Figure 11.

6 Deceptive Function

In this function[8], we use $S = 15,000$ and desire GAs to find the global optimal solution. Figure 14 shows the performance of GAs with different GA parameter values. It is observed

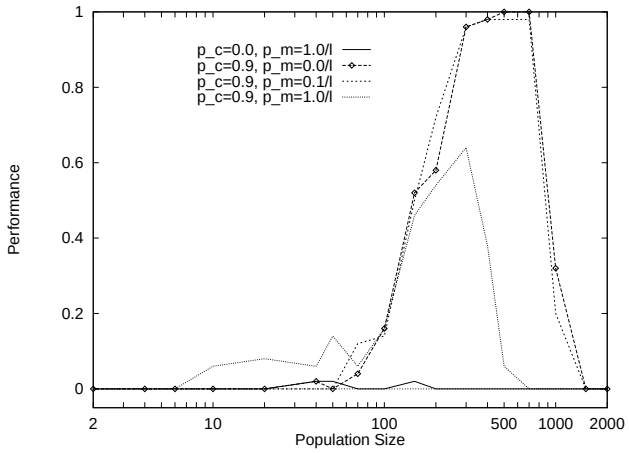

Figure 14: Performance of GAs with different GA parameters. A maximum of $S = 15,000$ function evaluations are used in each case.

that mutation-based GAs cannot find the optimal string with different mutation probabilities ranging from $p_m = 0.1/\ell$ to $1/\ell$. However, crossover-based GAs can find the correct solution in 100% of the runs for a wide range of population sizes. The population sizing estimate using equation 14 is $N = 350c$. With 90% confidence level, this size is about 584. Figure 14 shows that the GA's performance is best around this population size.

When crossover is aided with the mutation operator, the performance either does not improve or improves marginally. However, if a large mutation probability ($p_m = 1/\ell$) is used, the GA's performance degrades drastically. This is because a large mutation destroys the already-found good building blocks in a population. The fall of the GA's performance with

[8] Mühlenbein (1992) showed that order-k delineable GA-deceptive functions (of length ℓ) can be solved using his (1+1, m, hc)-algorithm in $O(\ell^k \log \ell)$ function evaluations. Using the exact form of the equation, we observe that we require more than $2(10^6)$ function evaluations to solve the above problem. We have used a tight coding and have only used a small fraction ($S = 15,000$ which is less than 1%) of this requirement.

a large population is due to smaller number of allowed generations. An Unuse Factor U plot similar to those for the previous functions is observed.

These results again show that if the function is difficult to solve, GAs require a suitable population size to solve the problem. If the population size is adequate, the crossover is a much more reliable approach than a mutation-based approach. Since $S = 15,000$ is very small compared to Mühlenbein's (1992) estimate, we do not observe any significant success of GAs with a mutation operator alone. However, Mühlenbein's estimate is valid for any kind of linkage used in coding the subfunctions, whereas the population sizing estimate used here and described in Goldberg, Deb, and Clark (1992) is valid for tight coding of subfunctions only.

7 Conclusions

In this paper, we have studied the effect of various parameters on the performance of GAs from the point of view of fixed computational cost. In order to investigate the performance of GAs on different fitness landscapes, we have chosen two unimodal functions (one in Hamming space and one in decoded parameter space), one four-peaked yet simple function, a massively multi-modal function, and a deceptive function.

In the following, we make our conclusions based on this study:

- Simulation results show that for unimodal and simple functions, mutation-based approaches have performed better than the crossover-based approaches. With a fixed number of function evaluations, a mutation-based GA performs best with a moderate population size. Too small or too large population sizes are detrimental. With a small population size, the required number of generations is too large to solve the problem with comparable number of function evaluations needed for moderate population size. This also explains why micro-GAs work nicely on simpler problems with a population size of 5 or more, but do not work as well with smaller population sizes. Whereas for a GA with a large population size, the number of allowed generations is not enough to find the optimum.
- GAs with both crossover and mutation operators have performed better than only crossover or mutation-based GAs in simpler problems, like Onemax and Himmelblau's functions.
- An important aspect observed from this study is the following. For simpler problems, although both mutation- and crossover-based approaches can find an optimal or a near-optimal solution, the working range for population size in each of these approaches is quite different. Mutation-based approaches require smaller population sizes compared to that in crossover-based approaches. However, the Unuse Factor graphs have shown that both these approaches required almost the same number of function evaluations.
- When GAs are applied to more complex problems involving massive multi-modality and misleadingness (or deception), a completely different scenario emerged. Mutation-based GAs have failed miserably to solve these problems, whereas crossover-based GAs are still able to solve these problems.
- It is also observed that in order to achieve good performance with either approach, a correct population size was needed. For crossover-based GAs, we have been able

to explain (and match with the simulation results) the required population sizes from a population sizing estimate developed elsewhere (Goldberg, Deb, and Clark, 1992). However, for mutation-based approaches, a theoretical population sizing does not exist. Our analysis supports the existence of such an optimal population sizing for mutation-based approaches. In either case, the correct population size must depend on the function being used.

- Based on these limited simulation results with simple tripartite GAs (binary tournament selection, single-point crossover, and bit-wise mutation operators), it can be concluded that when in doubt about the complexity of the problem at hand, one seems to be better off using a crossover operator with an adequate population size. Crossover-based GAs are more reliable for finding a near-optimal solution than mutation-based GAs in an arbitrary problem.

Since the use of correct population size is a crucial factor for successful GA applications, we strongly feel that more efforts need to be spent in finding correct population sizing estimates. We already have an estimate for subfunction-decomposable problems. What we need is a good yet ready-to-use population sizing estimate for generic problems.

It would be interesting to study the effect of other GA operators such as elitism or two-point crossover operator on the performance of GAs from a point of view of fixed computational cost.

References

Battle, D. L. and Vose, M. D. (1990). Isomorphisms of genetic algorithms. *Foundations of Genetic Algorithms.* 242–251.

Chakraborty, U., Deb, K., and Chakraborty, M. (1996). Analysis of selection algorithms: A Markov chain approach. *Evolutionary Computation, 4* (2). 132–167.

Culberson, J. C. (1994). Mutation-crossover isomorphisms and the construction of discriminating functions. *Evolutionary Computation, 2* (3). 279–311.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International 36* (10), 5140B (1975).

Deb, K. (1995). *Optimization for engineering design: Algorithms and examples.* New Delhi: Prentice-Hall.

Eshelman, L. J. and Schaffer, J. D. (1993). Crossover's niche. *Proceedings of the Fifth International Conference on Genetic Algorithms.* (pp. 9–14).

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* New York: Addison-Wesley.

Goldberg, D. E. (1993). Making genetic algorithms fly: A lesson from the Wright brothers. *Advanced Technology for Developers, 2.* 1–8.

Goldberg, D. E., Deb, K., and Clark. J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems, 6.* 333–362.

Goldberg, D. E., Deb, K., and Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. *Parallel Problem Solving in Nature, 2.* 37–46.

Goldberg, D. E., Deb, K., and Thierens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of SICE, 32*(1), 10–16. *Proceedings of the Fourth International Conference on Genetic Algorithms.* (pp. 190–195).

Goldberg, D. E., Korb, B., and Deb. K. (1989). Messy genetic algorithms: Motivation, analysis, and first results, *Complex Systems, 3.* 493-530.

Harik, G. R. (1997). Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms. (Doctoral Dissertation). Urbana: Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Harik, G., Cantu-Paz, E., Goldberg, D. E., and Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation.* (pp. 7–12).

Hart, W. E. and Belew, R. K. (1991). Optimizing an arbitrary function is hard for the genetic algorithm. *Proceedings of the Fourth International Conference on Genetic Algorithms.* (pp. 190–195).

Horn, J., Goldberg, D. E., and Deb, K. (1994). Long path problems. *Parallel Problem Solving from Nature, III.* (pp. 149–158).

Kargupta, H. (1996). The gene expression messy genetic algorithm. *Proceedings of IEEE Conference on Evolutionary Computation.* (pp. 814–819).

Kargupta, H., Deb, K., and Goldberg, D. E. (1992). Ordering genetic algorithms and deception. *Parallel Problem Solving from Nature, 2.* 47–56.

Krishnakumar, K. (1989). Microgenetic algorithms for stationary and nonstationary function optimization. *SPIE Proceedings on Intelligent Control and Adaptive Systems, 1196*, 289–296.

Mühlenbein, H. (1992). How genetic algorithms really work I: Mutation and hillclimbing. In R. Männer and B. Manderick (Eds.) *Foundations of Genetic Algorithms II.* Amsterdam: North-Holland. (pp. 15–25).

Mühlenbein, H., Schlierkamp-Voosen, D. (1993). Predictive models for breeder genetic algorithm: Continuous parameter optimization. *Journal of Evolutionary Computation, 1* (1). 25–49.

Nix, A. E., and Vose, M. D. (1992). Modelling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence, 5.* 79–88.

Oates, M. J. and Corne, D. (1998). Investigating evolutionary approaches to adaptive database management against various quality of service metrics. *Parallel Problem Solving from Nature, V*, 775–784.

Radcliffe, N. J. (1991). Formal analysis and random respectful recombination. *Proceedings of the Fourth International Conference on Genetic Algorithms.* (pp. 222–229).

Reklaitis, G. V., Ravindran, A. and Ragsdell, K. M. (1983). *Engineering optimization methods and applications.* New York: Wiley.

Rudnick, M. and Goldberg, D. E. (1991). Signal, noise, and genetic algorithms. (IlliGAL Report No. 91005). Urbana, IL: University of Illinois at Urbana-Champaign.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proceedings of the Third International Conference on Genetic Algorithms.* (pp. 51–60).

Suzuki, J. (1993). A Markov chain analysis on a genetic algorithm. *Proceedings of the Fifth International Conference on Genetic Algorithms.* (pp. 146–153).

Spears, W. (1993). Crossover or Mutation? *Foundation of Genetic Algorithms, 2.* 221–237.

Tate, D. M. and Smith, E. A. (1993). Expected allele coverage and the role of mutation in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms.* (pp. 31–37)

Thierens, D, and Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms.* (pp. 38–45).

Vose, M. D. (1992). Modeling simple genetic algorithms. *Foundations of genetic algorithms, 2.* 63–74.

Wu. A., Lindsay, R. K., and Riolo, R. L. (1997). Empirical observation on the roles of crossover and mutation. *Proceedings of the Seventh International Conference on Genetic Algorithms.* (pp. 362–369).