# Optimum Population Size and Mutation Rate for a Simple Real Genetic Algorithm that Optimizes Array Factors

Randy L. Haupt
Utah State University
Electrical and Computer Engineering
4120 Old Main Hill
Logan, UT 84322-4120
haupt@ieee.org
435-797-2841

Sue Ellen Haupt
Utah State University
Mechanical and Aerospace Engineering
4130 Old Main Hill
Logan, UT 84322-4130
Suehaupt@helios.ece.usu.edu
435-797-2952

**abstract** *The population size and mutation rate of a genetic algorithm have great influence upon the speed of convergence. Most genetic algorithm enthusiasts use a large population size and low mutation rate due to the recommendations of several early studies. These studies were somewhat limited. This paper presents results that show a small population size and high mutation rate are actually better for many problems.*

## I. Parameter Selection for a Simple Genetic Algorithm

Applications of a genetic algorithm (GA) to the optimization of electromagnetics problems started in the early 1990s [1], [2] and have exploded since then. The optimization of array patterns using a GA is particularly attractive for the synthesis of patterns that have desirable characteristics. Most of the work has followed traditional GA philosophy when choosing the population size and mutation rate of the GA: a relatively large population and a low mutation rate is used. The choice of population size and mutation rate can vary the run time of the GA by several orders of magnitude.

The first intensive study of GA parameters was done by De Jong [3] and is nicely summarized in Goldberg [4]. De Jong looked at both on-line and off-line performance of the GAs. On-line performance is an average of all costs up to the present generation. Off-line performance is the best cost found up to the present generation. He tested five algorithms of varying levels of complexity on five different cost functions while varying mutation rate, population size, crossover rate, and the generation. De Jong found that a small population size improved initial performance while large population size improved long-term performance. A higher mutation rate was good for off-line performance while low mutation rate was best for on-line performance. The highest mutation rate used was 0.1.

Grefenstette [5] used a meta GA to optimize the on-line and off-line performance of GAs based on varying six algorithm parameters: population size, crossover rate, mutation rate, scaling window, and whether or not elitism was used. A cost function evaluation for the meta GA consisted of a GA running until 5000 cost function evaluations were performed on one of the De Jong test functions and normalizing the result relative to that of a random search algorithm. Each GA in the population evaluated each of the De Jong test functions. The second step in this experiment took the 20 best GAs found by the meta GA and let them tackle each of the five test functions for five independent runs. The best GA for on-line performance had a population size of 30 and mutation rate of 0.01. The best off-line GA had a population size of 80 and mutation rate of 0.01. He concluded that good results could be obtained with a wide selection of GA parameters.

Schaffer, et. al. reported results on optimum parameter settings for a binary GA using a Gray code [6]. This approach added five more cost functions to the De Jong test function suite. They had discrete sets of parameter values (population size=10, 20, 30, 50, 100, and 200; mutation rate = 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, and 0.10; crossover rate = 0.05 to 0.95 in increments of 0.10; and 1 or 2 crossover points) that had a total of 8400 possible combinations. Each of the 8400 combinations was run with each of the test functions. They averaged the results over 10 independent runs. The GA terminated after 10,000 function evaluations. The best on-line performance resulted for the following parameter settings: population size =20 to 30 (relatively small), crossover rate = 0.75 to 0.95, mutation rate = 0.005 to 0.01 (the highest rates tested), and two point crossover.

Thomas Back [7, 8, 9] has done more recent analyses of mutation rate. He showed that for the simple counting problem, the optimal mutation rate is $1/\ell$ where $\ell$ is the length of the chromosome [7]. He later showed that an even quicker convergence can be obtained by beginning with even larger mutation rates (on the order of ½) and letting it gradually adapt to the $1/\ell$ value [8]. In later work [9], he compared this evolutionary GA approach with evolutionary strategies and showed that this adaptation is similar to the self-adaptation of parameters that characterizes evolutionary strategies approaches.

Gao [10] computed a theoretical upper bound on convergence rates in terms of population size, encoding length, and mutation probability in the context of Markhov Chain models for a canonical GA. His resulting theorem showed that the larger the probability of mutation and the smaller the population, the faster the GA should converge. However, he discounted these results as not viable for long-term performance.

Most of these previous studies were done with binary GAs. More engineers are discovering the benefits of using real parameter GAs, namely that a continuous spectrum of parameters can be represented. Our previous work with real GAs [11] devised a simple check to determine the best population size. The GA optimized several functions, and the results were averaged over 100 independent runs. The population size times the number of iterations (i.e., the total number of chromosomes evaluated) was kept constant. The "goodness" of the algorithm was judged by the minimum cost found. For both binary and continuous parameter GAs, a small population size allowed to evolve for many generations produced the best results. Similar sensitivity analyses with a wider range of mutation rates suggested that mutation rates in the range of 0.05 to 0.35 found the lowest minima.

A quick search of web sites on GAs also show conflicting evidence for the best parameters to use. Some sites [12, 13] suggest that GA performance may be improved for smaller population sizes and higher mutation rates. In addition, enough of our colleagues and students have found similar results for their GA problems that we decided further study is necessary.

These previous studies have shown that parameter settings are sensitive to the cost functions, options in the GA, bounds on the parameters, and performance indicators, which must be carefully considered. In addition, the optimum parameters seem to depend on whether the GA is just beginning its descent or whether it has advanced into the fine-tuning of the solution stage. Consequently, different studies result in different conclusions about the optimum parameter values depending on the problem and the parameters explored. Davis recognized this issue [14] and outlined a method of adapting the parameter settings during a run of a GA [15]. He does this by including operator performance in the cost. Operator performance is the cost reduction caused by the operator divided by the number of children created by the operator. Yet most GA practitioners still stick to large population sizes and very low mutation rates.

This paper extends the work in [11] from the optimization of contrived mathematical functions to the optimization of array factors. The goal is to help users of GAs select appropriate population sizes and mutation rates in order for their GAs to find the best answer as quickly as possible. Thus, emphasis is placed on off-line performance since we only care about the closeness of the final answer to the actual answer and not all the extraneous solutions included in the averaging of the on-line indicator. This paper reports the results of experiments to determine the optimum population size and mutation rate for a simple real GA on the types of problems that might be typical in electrical engineering. Since we want to minimize the run time of the GA, the criteria for judging the "goodness" of the results is the number of calls to the objective function required for solution. This is the metric that determines computer wall clock time to complete the solution. In addition, we choose to count function calls to the cost function as the criteria for how well the GA is performing. This choice is more in keeping with the usual engineering requirement to minimize run time. The parameters that produce the minimum number of function calls to produce an acceptable solution are deemed the "best." A solution is "acceptable" when a predetermined value close to the minimum is found. This definition is consistent with finding the deepest well, then diving to the bottom with a fast local optimizer. Determining the optimum population size and mutation rate must take into account the random components of the GA. Therefore, we average over a large number of runs of our GA before choosing the best parameters.

The GA used here is termed a real GA because the variables to be optimized are continuous and are not converted to binary values. Figure 1 shows a flow chart of a simple real GA. In each block of the flow chart, choices must be made on how to perform the GA operations in that block. The GA in this paper uses a roulette wheel proportional weighting selection and the single point crossover using the method described in [11]. Elitism is used. These are common choices used in practice and are constants for this particular study.

The results of this investigation show that, for the problems solved here, small population size and relatively large mutation rate are far superior to the large population sizes and low mutation rates that are used by most of the papers presented in the electromagnetics community and by the GA community at large. Such results suggest that future research consider carefully what parameters are appropriate to the particular problem.

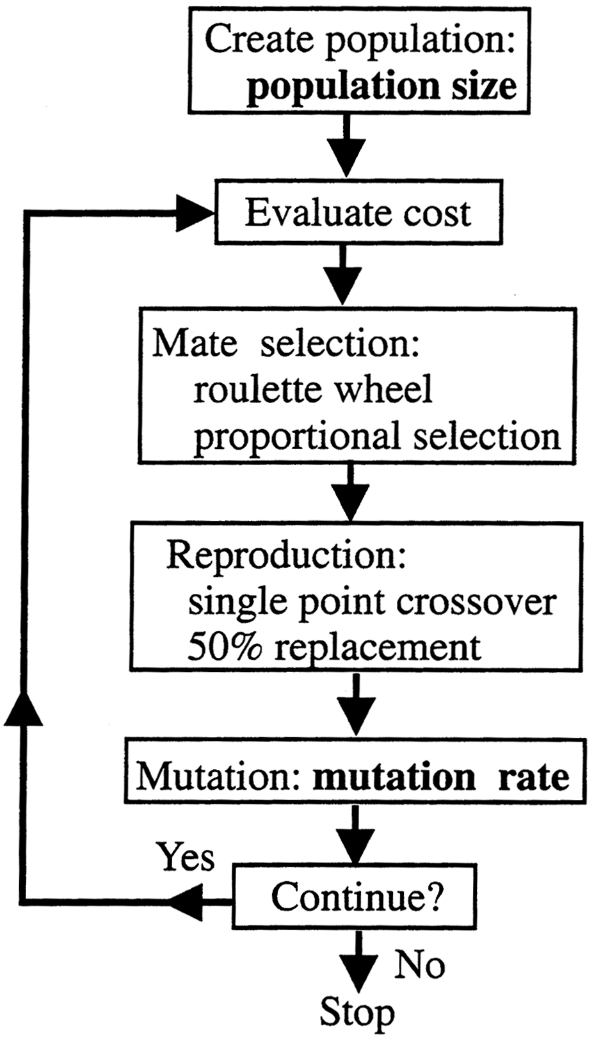

**Figure 1. Flow chart of the real GA. Finding the optimum mutation rate and population size would cause the GA to find an acceptable solution faster.**

## II. A Simple Undulating Function

The first example is a highly undulating function with many local minima. This function is

$$f(x,y) = x\sin(x) + 1.1\sin(y) \text{ for } 0 \le x, y \le 10 \qquad (1)$$

Figure 2 shows a graph of this function. The global minimum over the specified range is -18.5547 at (x,y) = (9.0390,8.6682).

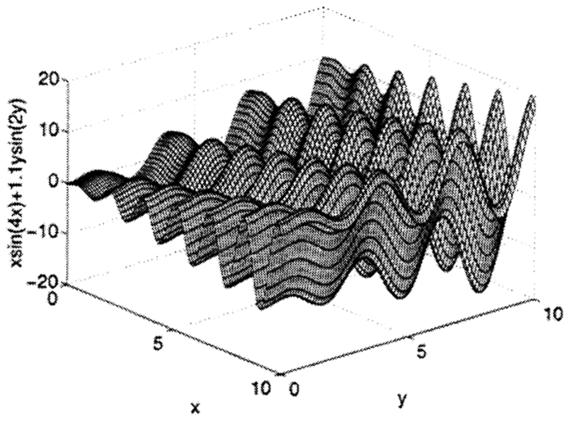

**Figure 2. Plot of the first function minimized by the GA.**

Doing single runs of a GA for different sets of population sizes and mutation rates doesn't yield sufficient information due to the statistical nature of the GA. To dampen the effects of the random processes, results are averaged over many runs for each set of parameters. Thus, the GA is run for one set of parameters until the solution is found. After performing T independent runs, the results for the T trials are averaged.

We posed the problem to minimize the function with the fewest number of function evaluations. Many engineering and scientific applications require the evaluation of very complex fitness functions. These function evaluations drive the time needed for the GA to converge. Therefore, our criteria for how "good" a GA run performs are a count of the number of calls to the cost function. A function evaluation is necessary for each new offspring (mutated or not) plus each mutated member of the old population. If a new offspring is selected to be mutated 3 times, then only one function evaluation is done. Otherwise, a high mutation rate would force a large number of unnecessary function evaluations.

One problem with a GA is determining when the "correct" answer is found. We addressed this issue in two ways for the function in (1). The first method used –18.5 as stopping criteria. Figure 3 shows the number of function calls vs. the number of GA runs averaged for a stopping point of –18.5. Oscillations occur until the GA is averaged about 150 times. For these criteria, we would not consider the average to be stable until about 150 runs have been averaged.

The second method of defining the "correct" solution was less rigorous but probably more realistic. The second lowest minimum for is -16.9847 and occurs at (x,y) = (7.4697,8.6681). Thus, if we obtain a value less than this local minimum, we are assured that we have found the valley of the global minimum. From there, we could use the solution as a first guess for a local optimizer that would quickly converge on the actual minimum point. Since this two-step process is often applied in practice, we stop the function when the cost is less than –17. Figure 4 shows the number of function calls vs. the number of GA runs averaged for a stopping point of –17. These results indicate that averaging as few as 40 or 50 runs would give a reasonably consistent average. Note that using –17 as the stopping point resulted in about ¼ of the runs needed for averaging than using –18.5 as the stopping point.

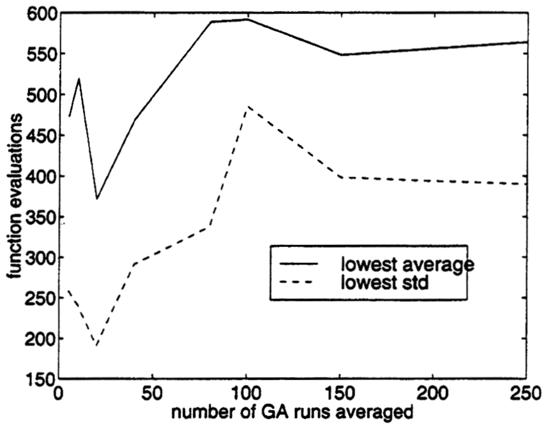

**Figure 3. These plots show both the average and the standard deviation of the number of function evaluations when the GA was stopped for a fitness that was less than –18.5.**

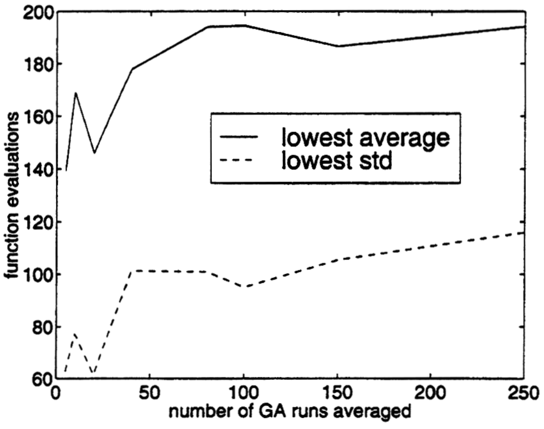

**Figure 4. These plots show both the average and the standard deviation of the number of function evaluations when the GA was stopped for a fitness that was less than –17.**

Now that we have determined the number of runs needed to average the GA to find the optimum parameter set, the GA with stopping criteria of –17 is averaged over 40 runs with mutation rates and population sizes of:

| | |
|---|---|
| mutation rate: | .01 to .49 in increments of .02 |
| population size: | 4, 12, 20, 28, 36, 44, 52, 60 |

We analyze the number of cost function evaluations required to converge for three different cost functions. Figure 5 shows the number of function calls required to find a point lower than –17. Very low mutation rates result in a huge number of function calls. Small population sizes seem to generally require fewer function calls than larger ones. The results indicate that a GA with a small population size (<16) and a mutation rate between .15 and .5 works best.

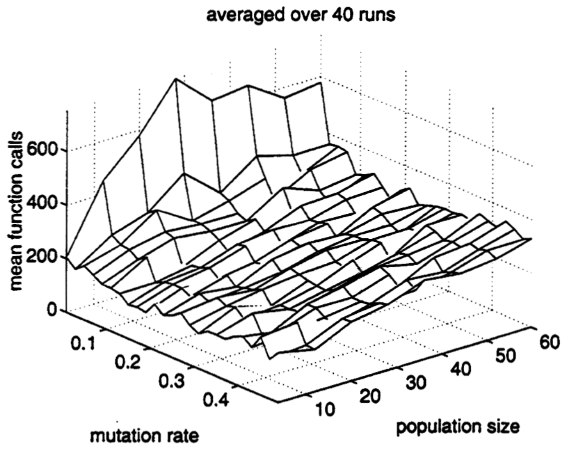

**Figure 5. The mean number of function calls are plotted vs. the mutation rate and population size when the GA is averaged over 40 runs.**

## III. Optimizing Side Lobe Tapers – Example 1

It is well known that a low sidelobe taper can be analytically found using a variety of methods including the Dolph-Chebychev and Binomial distributions. The point here is to just use a test case for the GA where we know the best solution – a binomial array. In fact, local optimizers provide excellent solutions for this problem as well. The authors are not advocating that an antenna designer should use a GA to find an amplitude taper for an array. There are many other much better techniques.

### *Problem Formulation*

The goal of the optimization is to find the weighting for a linear array that produces the minimum maximum sidelobe level. The objective function is given by

$$f = \max \textit{ sidelobe of} \left\{ \sum_{n=1}^{N} a_n e^{jp_n} e^{j(n-.5)kdu} \right\} \quad (2)$$

where
N = number of array elements
$a$ = vector of amplitude weights
$p$ = vector of phase weights
k = 2π/wavelength
d = element spacing
u = angle variable

In the cases presented, only $a$ or $p$ are optimized but not both in the same GA run. Thus, the number of parameters to be optimized is the length of the $a$ or $p$ vector. The array factor is calculated from broadside to endfire, and a search is performed to find all the peaks. The highest peak (outside the main beam) is returned as the cost of the function call. Most of the electromagnetics community use elitism and off-line performance for the various applications reported. These assumptions are used but not tested here.

The GA was run 500 times to find the minimum maximum sidelobe level for a 29 element array. Figure 6 shows three independent plots of the average number of function calls to reduce the sidelobe level below –25 dB vs. the number of GA runs included in the average. The lines become very close when the number of runs exceeds 250. That's a lot of averaging. Figure 7 shows the previous plot enlarged in the region of 1 to 25 averages. This region clearly shows that averaging the runs is critical to making valid interpretations of the data. When averaging is used, the number of function calls varies within a range of 500 for the three trials. At ten runs in the average, the number of function calls varies by 90 and at 20 the variation is down to 76. Averaging more than 100 runs adds a high level of confidence in any conclusions made concerning the optimum population size and mutation rate.

## *Results*

The GA is first used to find the optimum amplitude taper for an 18 element uniformly spaced array (d = 0.5 wavelengths). The taper is symmetric about the center of the array and the two center elements have an amplitude of one. Whenever the minimum maximum sidelobe level falls below 25 dB below the peak of the main beam or the number of function calls exceeds 50,000, the algorithm stops. The GA was run 20 independent times and the results were averaged for the following population sizes and mutation rates:

Population size = 4, 8, 12, …, 64

Mutation rate=.01, .02, .03,…, .4

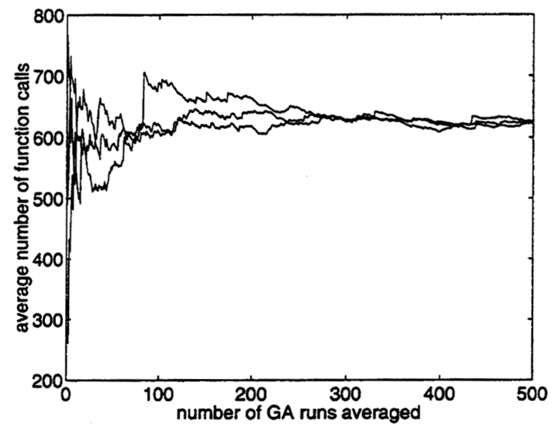

**Figure 6. Plot of the average number of function calls used by a GA to find the minimum maximum sidelobe amplitude taper of an 18 element linear array. The GA was run for up to 500 averages on three independent occasions.**

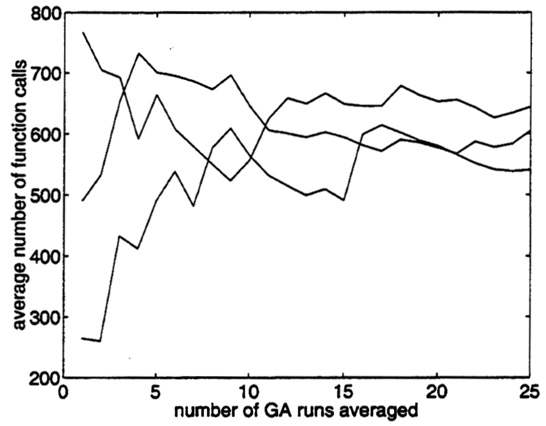

**Figure 7. This plot magnifies the left region of the graph in Figure 6.**

Figure 8 displays a plot of the average number of function calls vs. population size and mutation rate when the results were averaged over 20 independent runs. This graph is very low when the mutation rate is less than 20%, except for a subregion where the population size and mutation rate are small. Figure 9 shows another result where 20 independent runs were averaged and the population size varied from 4 to 128 and the mutation rate was between 1 and 19%. This plot shows the minimum number of function calls gradually increases as population size increases. GAs take a long time to converge when the population size is small and the mutation rate is small because population diversity comes at a slower rate.

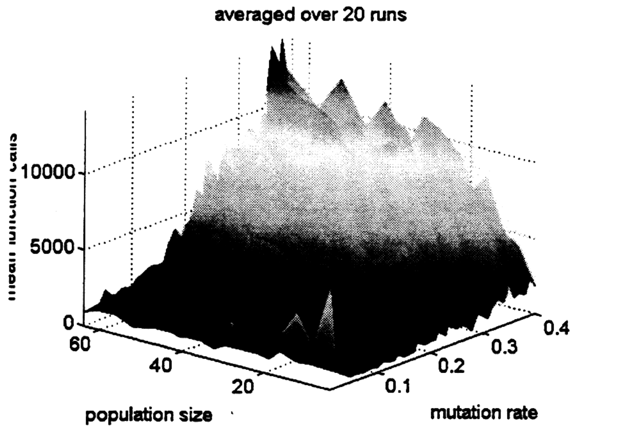

**Figure 8. The GA performed best (used the lowest number of function calls) when the population size was small and the mutation rate around 10%.**

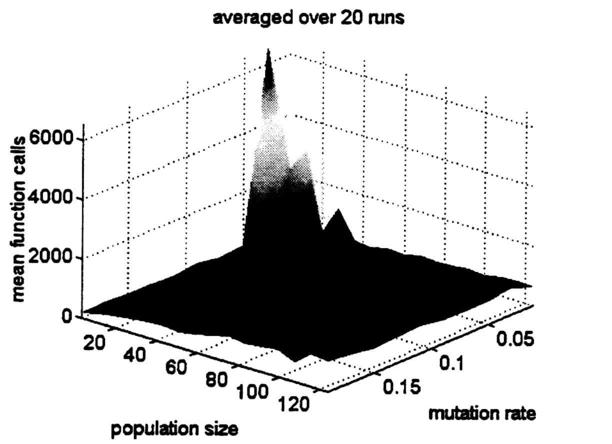

**Figure 9. The lower mutation was run again and the range of population sizes was increased.**

A strong region of performance in Figure 8 occurs between a population size of 4 to 16 and a mutation rate of 0.1 to 0.2. Figure 10 shows this region when the GA is averaged over 50 runs. The plot shows a population size of 8 or less and a mutation rate of 13% or less produce excellent results. Still afraid of abandoning conventional wisdom, the region between a population size of 4 and 128 and a mutation rate of 0.0 to 0.05 is examined, averaging the GA over 50 runs. Results, shown in Figure 11, are best for the smallest population sizes and mutation rate of 5%. Again, the region of low population size and low mutation rate yields slow convergence. Avoiding that range, it's quite apparent that the average number of function calls increases as the population size increases. Mutation rate doesn't seem to play much of a factor above a population size of 30. The next best mean number of calls was for a population size of 8 and mutation rate of 15% then mutation rate of 20%. These results are consistent with those in Figure 8. The poor performance of the large population sizes and a population size of 4 with mutation rate of 20% was predicted in Figure 10.

In order to become more confident with the results presented in the previous figures, the GA was averaged over 500 runs for several different mutation rates and population sizes as shown in Table 1. Results (in number of function calls) from running a GA 200 times to find the optimum amplitude taper for an 18 element array that minimizes the maximum sidelobe level. A single GA run stopped when the sidelobe level went below –25dB or the number of function calls exceeded 50,000. The minimum and maximum number of function calls over the 200 runs as well as the mean, and standard deviation of the number of function calls are shown here. A population size of 4 with a mutation rate of 15% produced the best average results.

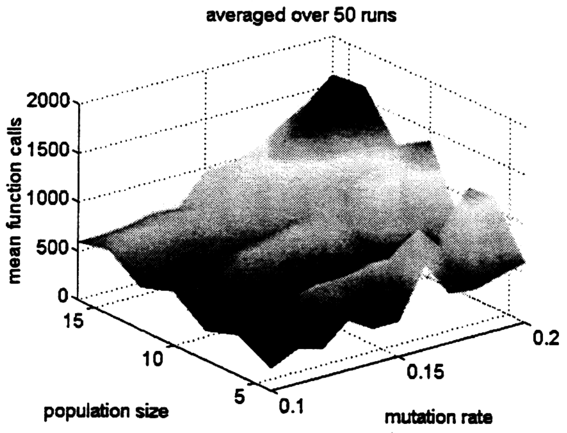

**Figure 10. This graph shows that a small population size and mutation rate of 0.1 causes a GA to find an answer in the fewest number of function evaluations.**

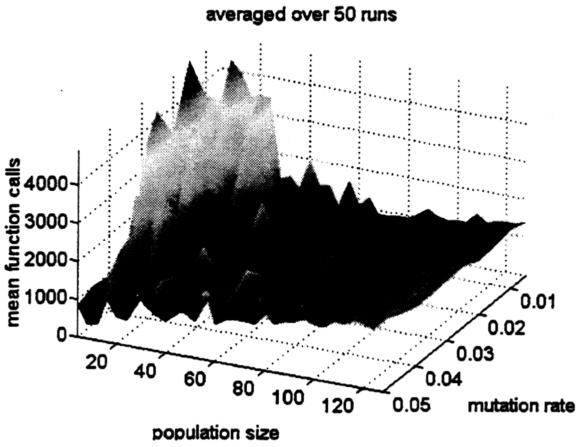

**Figure 11. Small population sizes and low mutation rates cause the GA to perform poorly. Note that, aside from very small population sizes, the mean number of function calls increases with population size independent of mutation rate.**

**Table 1. Results (in number of function calls) from running a GA 200 times to find the optimum amplitude taper for an 18 element array that minimizes the maximum sidelobe level. A single GA run stopped when the sidelobe level went below –25dB or the number of function calls exceeded 50,000. The minimum and maximum number of function calls over the 200 runs as well as the mean, and standard deviation of the number of function calls for the 200 runs are shown here.**

| Run | Mutation rate | Population size | minimum | maximum | mean | standard deviation |
|---|---|---|---|---|---|---|
| 1 | 0.15 | 4 | 26 | 3114 | 398 | 455 |
| 2 | 0.20 | 4 | 110 | 50002 | 7479 | 12798 |
| 3 | 0.15 | 8 | 60 | 2457 | 461 | 332 |
| 4 | 0.20 | 8 | 49 | 2624 | 654 | 466 |
| 5 | 0.01 | 64 | 300 | 50031 | 1158 | 3498 |
| 6 | 0.02 | 64 | 277 | 11818 | 1028 | 911 |
| 7 | 0.01 | 128 | 393 | 2535 | 1410 | 365 |
| 8 | 0.02 | 128 | 1215 | 50071 | 10208 | 16077 |

## IV. Optimizing Side Lobe Tapers – Example 2

The next example finds a low sidelobe taper for a linear array. Table 2 shows the results (in number of function calls) from running a GA 100 times to find the optimum phase taper that minimizes the maximum sidelobe level of a 40 element array. A single GA run stopped when the sidelobe level went below –14dB or the number of function calls exceeded 50,000. The minimum and maximum number of function calls over the 100 runs as well as the mean, and standard deviation of the number of function calls for the 100 runs are shown here. Once again the number of function calls is smallest for the smaller population sizes coupled with relatively large mutation rates.

It should be noted that even for the best parameters used in these tables, not all runs converged as evidenced by the maximum entries greater than 50,000. This fact has two implications. The first is that the mean number of function calls in the table would actually be higher if a limit were not in place. The second implication is that one should always be prepared to do multiple runs when using a GA since convergence is not assured.

## V. Optimizing Side Lobe Tapers – Example 3

In this example, a GA is run for 100,000 function evaluations in order to find the optimum amplitude taper for a 20 element array that minimizes the maximum sidelobe level. Table 3 shows the results in dB. The minimum and maximum result as well as the mean and standard deviation of the best sidelobe level for the 100 runs are shown here. The population size of 4 and 8 with 15% mutation rate outperformed the GA's with population sizes of 64 and 128 with a mutation rate of 2%.

**Table 2. Results (in number of function calls) from running a GA 100 times to find the optimum phase taper that minimizes the maximum sidelobe level of a 40 element array. A single GA run stopped when the sidelobe level went below –14dB or the number of function calls exceeded 50,000. The minimum and maximum number of function calls over the 100 runs as well as the mean, and standard deviation of the number of function calls for the 100 runs are shown here.**

| Run | Mutation rate | Population size | minimum | maximum | mean | standard deviation |
|---|---|---|---|---|---|---|
| 1 | 0.15 | 4 | 134 | 50002 | 2973 | 5856 |
| 2 | 0.20 | 4 | 163 | 50000 | 5232 | 9744 |
| 3 | 0.15 | 8 | 168 | 8223 | 1827 | 1510 |
| 4 | 0.20 | 8 | 124 | 21307 | 3220 | 3604 |
| 5 | 0.01 | 64 | 614 | 50024 | 7914 | 15040 |
| 6 | 0.02 | 64 | 546 | 50036 | 6624 | 13130 |
| 7 | 0.01 | 128 | 955 | 50043 | 4791 | 9708 |
| 8 | 0.02 | 128 | 933 | 50033 | 3942 | 7636 |

**Table 3. Results (in dB) from running a GA for 100,000 function evaluations in order to find the optimum amplitude taper for a 20 element array that minimizes the maximum sidelobe level. The minimum and maximum result as well as the mean, and standard deviation of the best sidelobe level for the 100 runs are shown here.**

| Run | Mutation rate | Population size | minimum | maximum | mean | standard deviation |
|---|---|---|---|---|---|---|
| 1 | 0.15 | 4 | -57.5 | -28.4 | -36.1 | 4.6 |
| 3 | 0.15 | 8 | -46.0 | -29.5 | -36.5 | 3.3 |
| 6 | 0.02 | 64 | -42.5 | -27.1 | -32.5 | 3.3 |
| 8 | 0.02 | 128 | -41.2 | -28.0 | -32.5 | 2.5 |

## VI. Conclusions

The results of the numerical experiments presented in this paper suggest that the best mutation rate for GAs used on these problems lies between 5 and 20% while the population size should be less than 16. These results disagree with some of the previous studies cited and common usage. The primary reasons for these results are that off-line performance was used and that a broader range of population size and mutation rate was included. In addition, the criteria judged here is the number of function evaluations, which is a good indicator of the amount of computer time required to solve the problem.

A way to interpret these results is in the context of analyzing the trade-offs between exploration versus exploitation. Traditionally, large populations have been used to thoroughly explore complicated cost surfaces. Crossover is then the operator of choice to exploit promising regions of phase space by combining information from promising solutions. The role of mutation is somewhat nebulous. As stated by Back [8], mutation is typically considered as a secondary operator of little importance. Like us, he found that larger values than typically used are best for the early stages of a GA run. In one sense, greater exploration is achieved if the mutation rate is great enough to take the gene into a different region of solution space. Yet a mutation in the less critical genes may result in further exploiting the current region. Perhaps the larger mutation rates combined with the lower population sizes act to cover both properties without the large number of function evaluations required for large population sizes. Iterative approaches where mutation rate varies over the course of a run such as done by Back [8, 9] and Davis [15] are likely optimal, but require a more complex approach and algorithm. Note that when real parameters, small population sizes, large mutation rates, and an adaptive mutation rate are used, the algorithm begins to lurk more in the realms of what has been traditionally referred to as evolutionary strategies. We feel that names are a mute point and choose to do what we find works best for a problem. In particular, we prefer the engineering approach of switching to a different optimization algorithm once the global well is found, since at that point the more traditional optimization algorithms become more efficient.

When the population sizes are as small as found here, tournament selection offers no advantage to roulette wheel selection, so an evaluation of the trade-off between these selection operators was not done. Selecting a small population size takes a very small amount of computer time. When doing the calculations for Table 3, the GA runs with large population size took at least 10% longer to run than the GAs with small population sizes for a fixed number of function calls. This difference can be attributed to the weighting and ranking in the selection operator.

These results are not totally alone. They are confirmed by our own prior results in [11] as well as those of Back [7, 8, 9] and predicted by the theory of Gao [10]. Also De Jong [3] found that a small population size and high mutation rate worked best during the initial generations and off-line performance. This is consistent with the results here since the algorithm is stopped when a prescribed minimum in the valley of the true minimum is found. If the GA were then used to pass results to a local optimizer, the GA need only work on the problem a short time.

Although these conclusions strictly apply to only the problems presented, in practice we have found many other problems where similar principles applied. No attempt has been made to thoroughly investigate all possible combinations of parameters. We chose to concentrate on population size and mutation rate after our own experience with optimizing GA performance. We make no claims that this is a definitive analysis: our purpose is merely to suggest that future GA practitioners consider a wider range of possible combinations of population size and mutation rate.

## Bibliography

[1] E. Michielssen, et. al., "Design of lightweight, broad-band microwave absorbers using genetic algorithms," IEEE MTT Transactions, Vol. 41, No. 6/7, Jun/Jul 93, pp. 1024-1031.

[2] R.L. Haupt, "Thinned arrays using genetic algorithms," IEEE AP-S Transactions, Vol. 42, No. 7, Jul 94, pp. 993-999.

[3] K. A. De Jong, "Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. Dissertation, The University of Michigan, Ann Arbor, MI, 1975.

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.

[5] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," IEEE Trans. on Systems, Man, and Cybernetics 16, January/February 1986, p. 128.

[6] J. D. Schaffer, et. al., "A study of control parameters affecting online performance of genetic algorithms for function optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, ed. J. D. Schaffer, Los Altos, CA: Morgan Kaufmann, June 4-7, 1989, pp. 51-60.

[7] T. Back, "Optimal mutation rates in genetic search," in *Proceedings of the 5th International Conference on Genetic Algorithms*, ed. S. Forrest, Morgan Kaufmann, 1993, pp. 2-9.

[8] T. Back and M. Schutz, "Intelligent mutation rate control in canonical genetic algorithms," in *Foundations of Intelligent Systems 9th International Symposium*, ed. Z.W. Ras and M. Michalewicz, Springer Verlag, Berlin, 1996, pp. 158-167.

[9] T. Back, "Evolution strategies: An alternative evolutionary algorithm," in *Artificial Evolution*, ed. J.M. Alliot, et al., Springer Verlag, Berlin, 1996, pp. 3-20.

[10] Y. Gao, "An upper bound on the convergence rates of canonical genetic algorithms," *Complexity International*, Vol. 5, 1998.

[11] R.L. Haupt and S.E. Haupt, *Practical Genetic Algorithms*, New York: John Wiley & Sons, 1998.

[12] http://www.revolver.demon.co.uk/ga/

[13]http://www.irg.lbl.gov/COMPS/numerical/NUMgas.html

[14] L. Davis, "Adapting operator probabilities in genetic algorithms," *in Proceedings of the Third International Conference on Genetic Algorithms*, ed. J. D. Schaffer, Los Altos, CA: Morgan Kaufmann, 1989, pp. 61-67.

[15] L. Davis, ``Parameterizing a genetic algorithm," in *Handbook of Genetic Algorithms*, ed. L. Davis, New York: Van Nostrand Reinhold, 1991.