Terrain-Based Genetic Algorithm (TBGA): Modeling Parameter Space as Terrain

V. Scott Gordon
Computer Science Dept.
Sonoma State University

Rebecca Pirie
Alcatel, USA

Adam Wachter
Alcatel, USA

Scottie Sharp
Eclipsys Corp

Abstract

The Terrain-Based Genetic Algorithm (TBGA) is a self-tuning version of the traditional Cellular Genetic Algorithm (CGA). In a TBGA, various combinations of parameter values appear in different physical locations of the population, forming a sort of terrain in which individual solutions evolve. We compare the performance of the TBGA against that of the CGA on a known suite of problems. Our results indicate that the TBGA performs better than the CGA on the test suite, with less parameter tuning, when the CGA is set to parameter values thought in prior studies to be good. While we had hoped that good solutions would cluster around the best parameter settings, this was not observed. However, we were able to use the TBGA to automatically determine better parameter settings for the CGA. The resulting CGA produced even better results than were achieved by the TBGA which found those parameter settings.

1 INTRODUCTION

Genetic algorithms are search algorithms based loosely on the principles of natural evolution, particularly genetic evolution. They have been useful mainly for optimization problems, such as finding the shortest path through a set of cities. By applying simplified notions of selection, crossover, mutation, and survival of the fittest to an artificial population of candidate solutions, a genetic algorithm evolves solutions with relatively little tailoring of the solution method to the problem domain. The user selects the population size, mutation rate, number of crossover points, etc. We refer to these values as GA *parameters.*

When a genetic algorithm is used for optimization, the user attempts to find a set of parameters whereby the best solution found by the algorithm is very good or at least acceptable. Selecting good parameter values (e.g., mutation rate, etc.) can be complicated, and is affected not only by the nature of the algorithm, but also the nature of the optimization task (Back, 1993; Davidor, 1992; Goldberg, 1989; Grefenstette, 1986; Schaffer, 1989).

Various researchers have attempted to design algorithms, which are self-tuning; that is, that adjust parameter values dynamically and therefore require less hand tuning. Fogarty (1989), Schraudolph and Belew (1990), Tsutsui and Fujimoto (1993), Smith (1993), and others have proposed methods for varying parameter values such as mutation rates, crossover mechanisms, etc. over time or in response to time-variant conditions (e.g., fitness improvement over time).

Our approach is simpler. Rather than dynamically changing parameters during a run, we instead spread a range of parameter values evenly along the axes of the population, so that each location in the population space has a different combination of parameters. Strings residing in different physical locations in the population structure are subjected to different parameter settings (although the parameter values in neighboring cells are roughly similar). The variances in parameter values over the space form a sort of terrain, and therefore we dub such an algorithm a *terrain-based* genetic algorithm (TBGA).

2 TBGA

One could envision a variety of ways to implement a TBGA. One simple and natural population structure on which to implement a TBGA is the grid structure often referred to as a cellular genetic algorithm. This is the approach we chose.

Cellular genetic algorithms (CGA), sometimes called massively-parallel GA's, assign one individual per processor and limit mating to within demes (neighborhoods). It is not necessary to use a massively parallel computer to implement a CGA, as it can be (and usually is) simulated on a single processor with a 2-dimensional matrix. There are many types of CGA's, depending on the selection and replacement mechanisms employed, and the size/structure of the demes. By 1989, numerous researchers had developed massively parallel genetic algorithms independently. Goldberg called it the pollination model in his 1989 textbook (Goldberg, 1989). Manderick and Spiessens called it the FG (fine-grained)

model (Manderick, 1989). Muhlenbein and Gorges-Schleuter had by this time dubbed it ASPARAGOS (Gorges-Schleuter, 1989; Muhlenbein, 1991). Whitley's cellular automata model of these algorithms (Whitley, 1993) led us to agree that the term CGA is the most descriptive, and is the one we use. In an earlier study, Gordon et.al. examined a variety of CGA's that were not terrain-based (Gordon, Bohm, and Whitley, 1994). Baluja examined a three-dimensional CGA (Baluja, 1993) which might be useful for implementing future TBGAs.

Our TBGA is based on a common CGA described in a previous study (Gordon, Mathias, and Whitley, 1994) called a *fixed-topology Deme-4* CGA. Each individual is processed at every generation, and an individual's mate is selected from the best of the four strings located above, below, left, and right (see Figure 1). Crossover is always performed, yielding two offspring, and mutation is applied to each offspring. If either resulting offspring has a better or equal fitness than the original node, then that node is replaced by the most fit offspring. Edge elements wrap around, forming a torus.

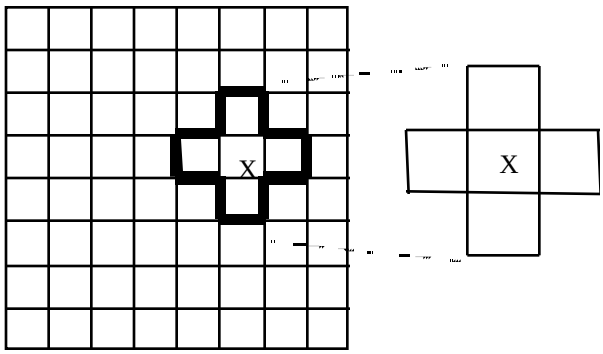

Figure 1: The Deme for Node X in a Deme-4 CGA

Since the CGA described has two dimensions and therefore two axes, we chose two parameters as terrain variables: *mutation rate* and *number of crossover points.* Spreading parameter values across the grid is a simple concept. One could simply select appropriate minimum and maximum values for the parameter in question, and use a linear (or some non-linear) distribution to determine the actual parameter value to use at a given cell.

Consider the 8x8 CGA in Figure 2. In this example, mutation rates are spread linearly along the X-axis. The maximum mutation rate is 35%; the minimum rate is 0%. Mutation rates for each cell is shown along the X-axis at the bottom of the grid. Note that every cell in a given column has the same mutation rate.

The number of crossover points parameter is also shown in Figure 2, spread along the Y-axis. The minimum number of points is 1, the maximum is 8. Values for each cell are shown along the y-axis at the left of the grid. Note that every cell in a given row has the same value.

Even though several cells share the same mutation rate (for example), every cell has a different combination of parameters. Thus, in a TBGA with parameters shown as in Figure 2, cells in the far upper left have low mutation rates and few crossover points; cells at the far lower left have low mutation rates and many crossover points, etc. In theory, a TBGA could utilize a wide variety of parameter combinations, using this topological approach. Extending the method to three parameters could be done, for example, with a 3-dimensional CGA.

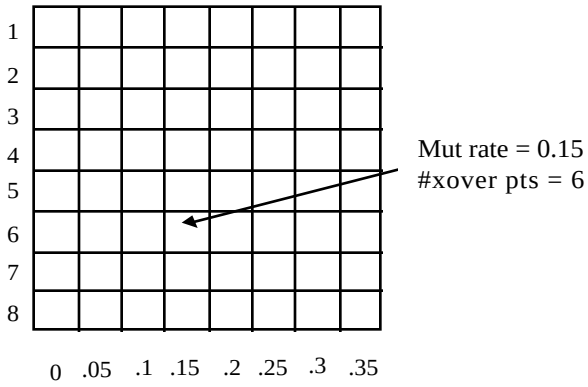

Figure 2: *Mutation rate* Spread Along the X-axis of a CGA, *Number of Crossover pts* Spread Along the Y-axis.

For some parameters, a linear distribution scheme such as shown in Figure 2 may not be ideal. For example, it would not be a good approach for number of crossover points. It would be more appropriate to model a variety of functionally different values, such as 1-point, 2-point, uniform, and other commonly used values. A linear distribution would lead to a disproportionate number of values which behave essentially like uniform crossover.

In the TBGA implemented, we used a 20x20 grid of cells, with the following ranges and non-linear distributions of parameters:

- *Number of Crossover Points:* These are distributed linearly from 0 to 8 over the first 15 rows, and then distributed linearly from 8 to strlen/2 over the last 5 rows, where strlen is the string length utilized by the encoding. The intent of this distribution is to have most of the population employ a variety of commonly-used values (0-8 points), while setting aside a region where uniform crossover is applied.
- *Mutation Rate:* Values are distributed linearly from 0 to .75/strlen over the first 15 columns, and then distributed linearly from .75/strlen to 3/strlen over the last 5 columns (strlen defined above). This distribution enabled the majority of the population to employ a variety of typical mutation rates, while setting aside a region where rather high mutation rates occur.

Finally, we rearranged the distribution of parameters by a process which we called *sifting*. Consider again the distribution of crossover values shown in Figure 2. Since CGAs are toroidal (the edges wrap around), each of the cells along the bottom have neighbors in the top row. This means that there is a region where neighboring cells have maximally different parameter values. We did not wish to explore the effects of radically divergent parameter values within a deme, instead preferring to concentrate on systems where neighboring cells always have similar parametric effects.

To achieve a uniformly smooth distribution of parameters, while still retaining the toroidal CGA topology, we placed the maximum parameter value in the center, then placed each decreasing value on alternating sides. After this sifting process, the parameter values shown in Figure 2, for example, would be rearranged as shown in Figure 3.

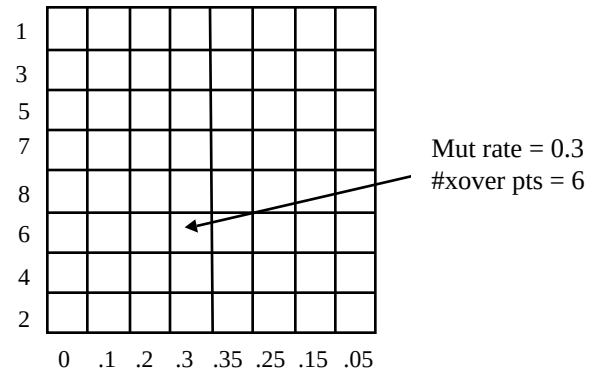

Figure 3: *Mutation Rate* (along the X-axis*)* and *Number of Crossover Points* (along the Y-axis), after Sifting

3 TEST SUITE

Our test suite consists of three numeric problems, one deceptive problem, and a knapsack problem. The numeric problems, all minimization functions, are hereafter referred to as the **F2**, **F4**, and **Rastrigin** functions. F2 and F4 are from DeJong's suite and are well documented, F2 being a somewhat difficult multimodal function and F4 involving a large solution space plus Gaussian noise (DeJong, 1975). The Rastrigin function was described in (Muhlenbein, 1991), and used frequently as a GA test problem because of its large search space and many local minima. The function has 20 parameters x_i in [-5.12,5.12] where:

$$f(x_i \mid i = 1,n) = 200 + \sum_{i=1}^{20} x_i^2 - 10\cos(2\pi x_i)$$

The ugly 4-bit deceptive problem (hereafter referred to as D4) is a 40-bit artificially-constructed problem introduced by Whitley (1991) in which ten fully-deceptive 4-bit subproblems are interleaved. In general, the three bits of each subproblem X appear in positions X, 10+X, 20+X, and 30+X. Whitley's problem is based on a similar 3-bit problem introduced by Goldberg, Korb, and Deb (1989). Gordon and Whitley discussed the implications of using deceptive problems to measure performance (Gordon, Whitley, 1993).

The zero-one knapsack problem is defined as follows. Given n objects with positive weights W_i and positive profits P_i, and a knapsack capacity M, determine a subset of the objects represented by a bit vector X_i such that

$$\sum_{i=1}^{n} X_i W_i \le M \text{ and } \sum_{i=1}^{n} X_i P_i \text{ maximal}$$

A greedy approximation to the global optimum can be found by selecting objects by profit/weight ratio until the knapsack cannot be further filled.

A simple GA encoding for the knapsack problem is to let each bit represent the inclusion or exclusion of one of the n objects from the knapsack. A bitstring of length n can be used to represent candidate solutions. If the objects are sorted by profit/weight ratio, then the greedy approximation appears as a series of 1s followed by a series of 0s. The difficulty with this representation is that it is possible to generate infeasible solutions. Setting too many bits to 1 might overflow the capacity of the knapsack. Gordon and Whitley (1993) considered two methods for handling overflow, and also described two knapsack problems. Here, we use the 20-object problem described by Bohm and Egan (1992), and the *penalty* evaluation method for handling overflow described by Gordon and Whitley.

Gordon, Bohm, and Whitley (1994) have argued that genetic algorithms perform poorly on these knapsack problems (and much larger instances) when compared with depth-first and branch-and-bound search methods. The knapsack experiments presented here are therefore useful only for comparing the various genetic algorithms against each other.

4 NATURE OF STUDY

For each function in the test suite, we compare the performance of the TBGA against that of the standard CGA. While we did not expect the TBGA to outperform the CGA, comparable performance might suggest that the TBGA is adequate and easier to use, because it requires less tuning. Parameter settings in the hand-tuned CGA were those determined by trial-and-error in Gordon and Whitley (1993) to be good. Number of crossover points was set to two, and mutation rate (the likelihood that a given bit is flipped) was 1/(2*strlen), where strlen is the length of the string used for the encoding.

We run both genetic algorithms across the test suite of functions for 30 runs. We plot the average fitness of the best strings found at each generation. These plots are shown in Figures 4-7 (we do not include a plot for F4 because the two algorithms perform identically). Each plate also includes a plot for a "tbga-tuned" CGA, which will be described later in the paper. We also qualitatively assess the speed of finding solutions, especially when both of the algorithms find the global optimum.

Each problem in the test suite is algebraically converted to a minimization function with a minimum of zero. The only exception to this is DeJong's F4 which has a minimum of -6.

5 TEST RESULTS

To our surprise, the self-tuning TBGA performed better than the CGA on four of the five test problems, and was only slightly outperformed by CGA on the knapsack problem (this will be explained later). Table 1 shows the average fitness of the best strings found at the end of the runs for each algorithm on each problem (the better performance of the two is shown in boldface).

Table 1: Average Best Fitnesses by GA's on Test Suite

	D4	F2	F4	Rastrigin	Knapsack
TBGA	**9.67**	**0**	**-3.049**	**1.14**	0
CGA	9.93	.00016	-3.047	2.3	**0**

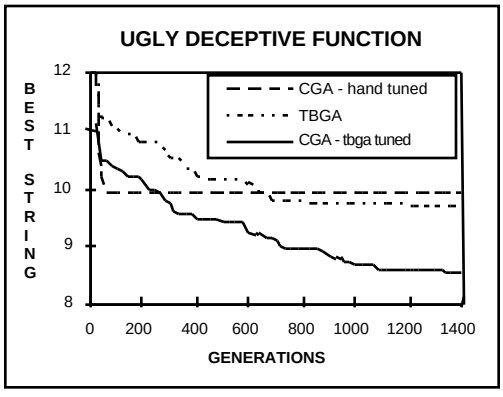

Figure 4: Test Results for Ugly Deceptive Function

In the deceptive function (**D4**), Figure 4 shows clearly that the TBGA starts more slowly than the hand-tuned CGA, but ultimately produces better solutions than the CGA. We continued running the algorithm further (to 2500 generations), but the relative performance indicated in Figure 4 did not change.

In DeJong's **F2** function, Figure 5 shows the TBGA significantly outperform the hand-tuned CGA. All TBGA runs had solved the problem before the 318th generation, whereas some of the CGA runs had not solved the problem even after 500 generations.

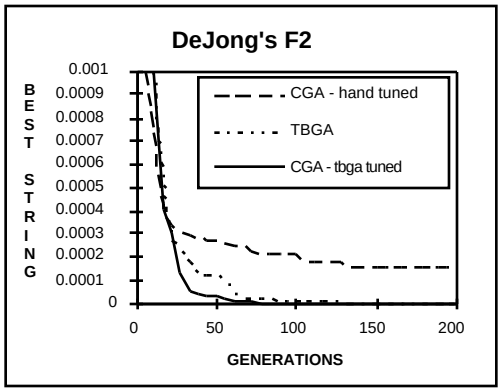

Figure 5: Test Results for DeJong's F2 Function

In DeJong's **F4** function, the performance of the two algorithms is nearly indistinguishable. We ran the algorithms to 1000 generations, after which the TBGA was finding, on average, slightly better but practically identical solutions as the CGA.

The **Rastrigin** function was clearly optimized more effectively by the TBGA than the hand-tuned CGA, as shown in Figure 6. The latter appears to have prematurely converged.

The **Knapsack** function was the only one in which the TBGA was outperformed by the CGA. The performance difference was very slight, as shown in Figure 7. Note that the TBGA was the fastest algorithm at finding very good solutions, although ultimately the hand-tuned CGA found the global optimum slightly sooner (all runs by the 31st versus 34th generations, respectively).

Additionally, we ran some tests with a TBGA that varied only one of the parameters at a time. This was done to ensure that we ruled out independent effects of the application of TBGA method to each parameter. The TBGA performed best when *both* parameters were varied along the axes, and so we do not report those results here.

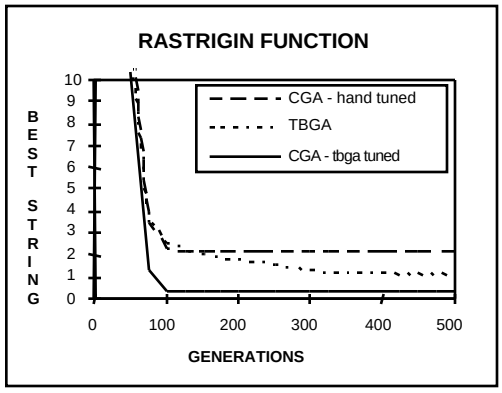

Figure 6: Test Results for the Rastrigin Function

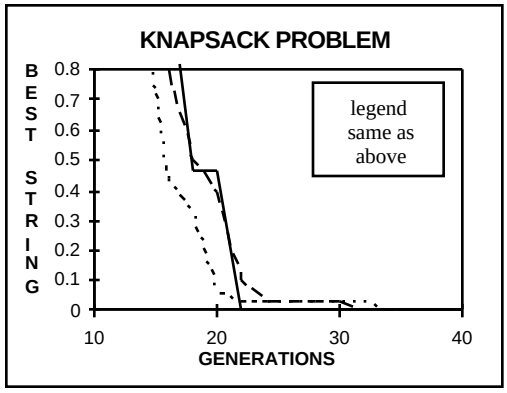

Figure 7: Test Results for the Knapsack Function

6 FINDING GOOD PARAMETER SETTINGS

The experiments described above led us to question whether the parameters chosen for the CGA were in fact the best, or even good. Of course we had no way of answering that question, short of trying all combinations of parameter settings. We theorized that the TBGA might be a useful tool for finding good parameter settings to use in a standard CGA. To do this, we needed to find where (physically) in the population structure the best solutions were evolving. We could then utilize the parameter settings that existed at that location, and run a CGA, fixed at those settings, on the same problem.

We started by having the TBGA report the location of each newly found current best string. Unfortunately, these locations did not always cluster around specific areas, but rather were spread around the grid, over a fairly wide range of parameter settings. So in lieu of a better indicator, we simply tallied, for each location, the number of times a new current best string appeared there. From this we produced a weighted average of the parameter settings at those locations. This did provide a single set of parameter values that we could try in the CGA.

For example, suppose we were using a 3x3 grid, and say that we ran the TBGA for 10 generations, tallying, for each cell, how often a new current best string appeared there. Since each cell also has a unique value for mutation we might tally data similar to that shown in Table 2. The suggested mutation rate would then be the sums of the mutation rates, weighted by the tallies, or:

```
[5*(.01) + 3*(.02) + 2*(0)]/10 = .011
```

After using this technique to find suggested parameter settings for each problem, we then re-ran the experiments above (with the exception of DeJong's F4) using the parameter values suggested by the TBGA. In every case, using the parameters suggested by TBGA greatly improved the performance of the CGA to such an extent that it even outperformed the TBGA. We observed this whether we substituted one or both of the suggested parameter values. Here we report only the results when substituting both suggested parameter values.

Table 2: Example Tally for Finding Mutation Rate

CELL	Mutation Rate	#bests
(1,1)	0	0
(1,2)	0	0
(1,3)	0	2
(2,1)	.01	0
(2,2)	.01	5
(2,3)	.01	0
(3,1)	.02	3
(3,2)	.02	0
(3,3)	.02	0

The particular parameter values suggested by the TBGA in the manner described above are next shown in Table 3.

Table 3: Parameter Values Suggested by TBGA

	D4	F2	Rastrigin	Knapsack
Mutation rate	.0186	.0463	.00328	.0336
#Xover pts	8	6	12	6

Figures 4-7 also include the tests of the CGA tuned with the TBGA-suggested parameter values. The results are striking: in every case, the parameter values suggested by the TBGA produces markedly better results than both TBGA and the original CGA, with the TBGA performing roughly in between.

7 CONCLUSIONS

The Terrain-Based Genetic Algorithm (TBGA) is a self-tuning version of the traditional Cellular Genetic Algorithm (CGA) in which parameter values are spread around the population forming a sort of terrain in which individuals evolve. Preliminary results have shown the TBGA to be useful both as a GA function optimizer, and as a powerful tool for extracting better performance from a CGA. The algorithm is simple to implement and easier to use than a CGA as it requires less parameter tuning.

The performance of the TBGA as a function optimizer, as compared to our CGA, is a pleasant surprise. We conducted our experiments expecting the CGA, with it's parameters tuned during a previous study, to outperform the TBGA. Instead, we watched the TBGA, without any tuning at all, immediately outperform our CGA on the majority of functions in our test suite. This indicates that the TBGA may have good potential as a general-purpose workhorse genetic algorithm.

We also expected the TBGA to solve problems in such a way that the solutions were clustered around particular parameter settings. In fact, we saw the best solutions spread over a fairly wide range of parameter settings. Perhaps the TBGA is utilizing each of the various parameter settings in some collective manner, representing an essentially different algorithm than *any* genetic algorithm with fixed parameter settings. More study is needed to determine whether the TBGA exhibits a discernible pattern of parameter utilization. Further study is also needed to explore how best to apply a TBGA when more than two parameters require tuning.

Despite the lack of obvious clustering of good solutions in a TBGA's population structure, we were still able to utilize the locations of good TBGA solutions as an indicator for good CGA parameter settings. We used a simple method of counting the frequency of best strings for each cell, and were surprised to find that the parameter settings suggested by computing a weighted sum of the corresponding parameter values were superior when used in our CGA on our test suite.

The TBGA is not only a promising method for solving optimization problems, it is also a vehicle for finding very good parameter values to use in existing programs. We do not know whether the TBGA can be further modified in such a way as to outperform *all* CGAs, or whether a CGA, fine-tuned by a TBGA, always produces the best results. Further study needs to be done in this area.

Acknowledgements

This work was supported in part by a California State University RSCAP grant.

The authors also wish to thank the anonymous reviewers for their careful reading and insightful comments.

References

T. Back (1993). Optimal Mutation Rates in Genetic Search. *Proceedings of the 5th International Conference on Genetic Algorithms*, pg 2.

S. Baluja (1993). Structure and Performance of Fine-Grain Parallelism in Genetic Search. *Proceedings of the 5th International Conference on Genetic Algorithms*, pg 155.

A. Bohm and G. Egan (1992). Five Ways to Fill Your Knapsack. *Proceedings of the Second Sisal Workshop*, LLNL CONF-9210270.

Y. Davidor and O. Ben-Kiki (1992). The Interplay Among the Genetic Algorithm Operators: Information Theory Tools used in a Holistic Way. *Parallel Problem Solving from Nature 2*, pg 75.

K. DeJong (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan.

T. C. Fogarty (1989). Varying the Probability of Mutation in the Genetic Algorithm. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pg 104.

D. Goldberg (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley.

D. Goldberg (1989). Sizing Populations for Serial and Parallel Genetic Algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pg 70.

D. Goldberg, B. Korb, and K. Deb (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems 3*, pg 493.

V. Gordon, A. Bohm, and D. Whitley (1994). A Note on the Performance of Genetic Algorithms on Zero-One Knapsack Problems. *ACM Symposium on Applied Computing* (SAC'94), Genetic Algorithms and Combinatorial Optimization Track, pg 194.

V. Gordon, K. Mathias, and D. Whitley (1994). Cellular Genetic Algorithms as Function Optimizers: Locality Effects. *ACM Symposium on Applied Computing* (SAC'94), Genetic Algorithms and Combinatorial Optimization Track, pg 237.

V. Gordon and D. Whitley (1993). Serial and Parallel Genetic Algorithms as Function Optimizers. *Proceedings of the 5th International Conference on Genetic Algorithms*, pg 177.

M. Gorges-Schleuter (1989). ASPARAGOS - An Asynchronous Parallel Genetic Optimization Strategy. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pg 422.

J. Grefenstette (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol SMC-16, No.1, January/February 1986, pg 122.

H. Muhlenbein, M. Schomisch, and J. Born (1991). The Parallel Genetic Algorithm as Function Optimizer. *Proceedings of the 4th International Conference on Genetic Algorithms*, pg 271.

B. Manderick and P. Spiessens (1989). Fine-Grained Parallel Genetic Algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pg 428.

J. D. Schaffer, R. Caruana, L. Eshelman, and R. Das (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pg 51.

N. Schraudolph and R. Belew (1990). *Dynamic Parameter Encoding for Genetic Algorithms*. CSE Technical Report #CS 90-175, University of California, San Diego Computer Science and Engr. Dept.

R. Smith (1993). *Adaptively Resizing Populations: An algorithm and analysis*. (TCGA Report No. 93001) Tuscaloosa: University of Alabama.

S. Tsutsui and Y. Fujimoto (1993). Forking Genetic Algorithm with Blocking and Shrinking Modes (fGA). *Proceedings of the 5th International Conference on Genetic Algorithms*, pg 206.

D. Whitley (1991). Fundamental Principles of Deception in Genetic Search. *Foundations of Genetic Algorithms.* Morgan Kaufmann, pg 221.

D. Whitley (1993). Cellular Genetic Algorithms. *Proceedings of the 5th International Conference on Genetic Algorithms*, pg 658.