

# Crossover in Function Optimization

*Scott H. Douglas<sup>1</sup>*  
*Rochester Institute of Technology, Rochester, NY*

## Abstract

Function optimization is the process of finding the global maximum or minimum over a certain range of a function. One of the major algorithms for function optimization is hill climbing, but this method often gets hung up on local maximums or minimums. Genetic algorithms (GAs) don't get caught on local maximums or minimums because of their stochastic nature and use of mutation. This paper uses a GA to perform function optimization, and then analyzes how different forms of crossover affect the number of fitness evaluations required to converge on a solution for a simple ten variable function.

## 1. Introduction

Genetic Algorithms (GAs) are effective search techniques that overcome some of the downfalls of other search techniques. GAs are a good solution to function optimization problems because they don't get hung up on local minimums or maximums. A GAs performance is at least as good as random search.

The convergence of a GA is often slower than traditional numeric techniques that have a good initial guess. The GA will spend time checking chromosomes that are no where near the solution. Optimization of the GA process would prove useful.

Crossover is one major method that GAs use to converge on a solution. Crossover is the process of exchanging bits between parent chromosome bit strings. There are many types of crossover available, but each one has different results. In recent experiments, we have noticed that uniform crossover generally provides a solution with the fewest amount of required fitness evaluations. Since fitness evaluations are costly, they are a good metric of how a GA is performing. By running a function optimization with different crossover operators, we hope to find results to show the uniform operator works best.

## 2. Software Design

### 2.1. Functional Description

The GA system must evolve a bit string that represents the variables in an equation for the global minimum or maxi-

imum of another variable in the equation.

### 2.2. System Overview

The system implemented has minor changes from the system implemented in the paper "Variations on a Simple Genetic Algorithm" by Scott Douglas. Generic GA classes were created for Chromosomes and Populations. An interface was created for defining custom crossover and fitness functions. To use the system to solve a problem, simply provide a crossover function and a fitness function that measures the fitness of the bit string according to the problem to solve. Pass these parameters into a Population instance, which randomly generates the initial population. At this point, statistics about the population can be queried. To advance to the next generation, call `nextGeneration()`. This function copies a certain percentage of individuals to the next generation based on fitness. For the remaining individuals, tournaments of two individuals are held to find two parents. Crossover is performed based on the crossover rate. This is done until the new generation is full. Mutation is employed based on the mutation rate parameter. The next generation process is repeated until the solution is found or a maximum number of generations has been exceeded.

### 2.3. Initial Parameters

- Crossover Rate: 0.7
- Crossover Type: 1 Pt.
- Mutation Rate: 0.001
- Max. Generations: 1,000
- Population Size: 50
- Chromosome Length - Proportional to Parameters of GA Fitness
- Copy Rate: 0.1

### 2.4. Encoding

Each parameter of the fitness function is encoded as a float between 0.0 and 1.0. To do this, each parameter of the fitness has thirty two bits assigned to it. These bits are shifted

<sup>1</sup>Send correspondence to [shd0326@cs.rit.edu](mailto:shd0326@cs.rit.edu).

into an unsigned int to form an unsigned int between zero and the maximum unsigned int. At this point, that number is divided by the maximum unsigned int to give a float between 0.0 and 1.0.

This method provides a robust way to search through a specific range of values. Since a bit string can be interpreted by the fitness function any way, a float ranging from 0.0 to 1.0 can be used to come up with any range of numbers with the most precision possible with thirty two bits of data. For example, to get a number between 0 and 40, take the float and multiply it by 40. To get a number between 30 and 70, multiply the float by 40 and then add 30.

### 3. Testing

In order to ensure the system is working properly before beginning experimentation, testing was performed on a one-variable function, two-variable function, and ten-variable function with known maximums and minimums. The initial parameters were used. The results of the tests show that the system implemented converges to a correct solution for functions with different numbers of variables to optimize. The ten-variable function is then used for experimentation since we know the system can converge on a solution to it.

#### 3.1. One-Variable Optimization

The function used for one variable optimization was:

$$x^4 - 12x^3 + 15x^2 + 56x - 60 \quad (1)$$

over a range from  $[-512, 512]$ . See Figure 1 for a graph of the function over this range. See Figure 2 for a graph of the function at the global minimum, which was derived using a TI-83 to be  $(7.8102038, -703.7288)$ . The GA determined the minimum after an average of 19,944 fitness evaluations.

#### 3.2. Two-Variable Optimization

The Bohachevsky F1 function is defined as

$$x^2 + 2y^2 - 0.3\cos(3\pi x) - 0.4\cos(4\pi y) + 0.7 \quad (2)$$

. When bounded from  $[-50, 50]$  for  $x$  and  $y$ , the function has more than 50 maximas and more than 50 minimas. See Figure 3 for a plot of this function and Figure 4 for a graph of the function's contour. The optimum minimum is located at  $(0, 0)$ . The radius of the optimum minimum is 0.2. The algorithm found the optimum minimum within the radius after an average of 45,439 fitness evaluations.

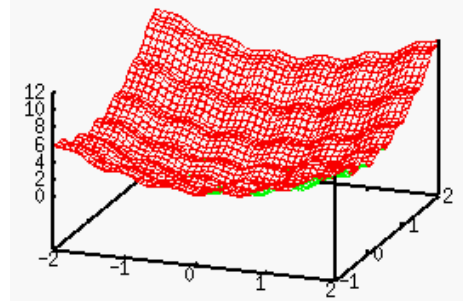


Figure 3: The Bohachevsky F1 courtesy of <http://www.daimi.au.dk/ursem/EAdocs/EA/testproblems/package-summary.html>.

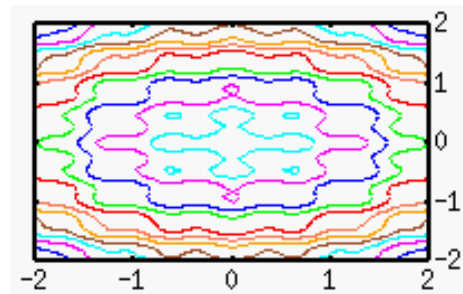


Figure 4: The Bohachevsky F1 contour courtesy of <http://www.daimi.au.dk/ursem/EAdocs/EA/testproblems/package-summary.html>.

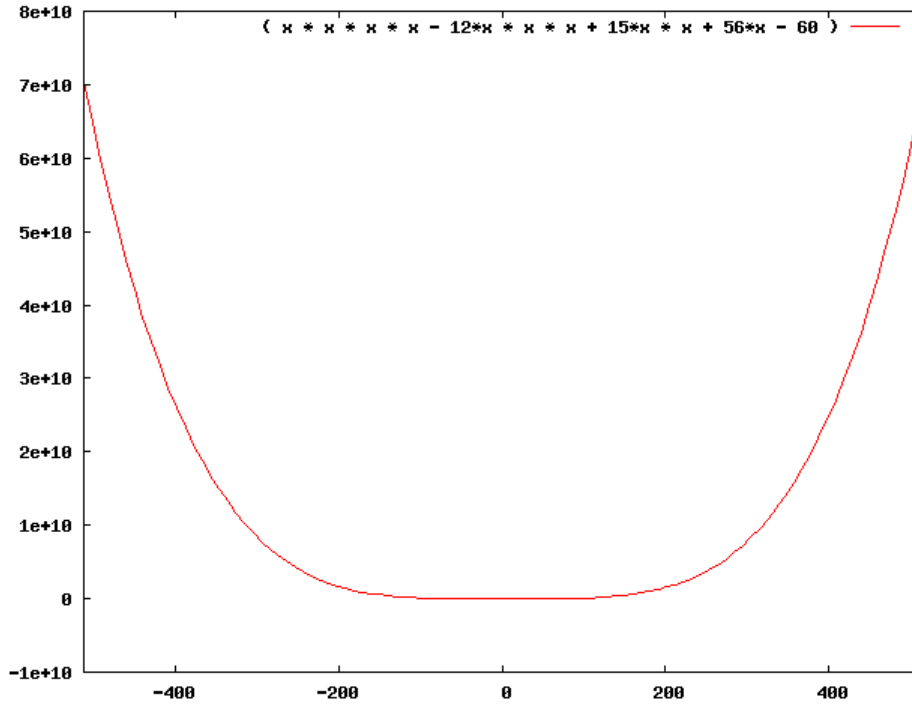


Figure 1: The function  $x^4 - 12x^3 + 15x^2 + 56x - 60$  graphed over the search space from -512 to 512.

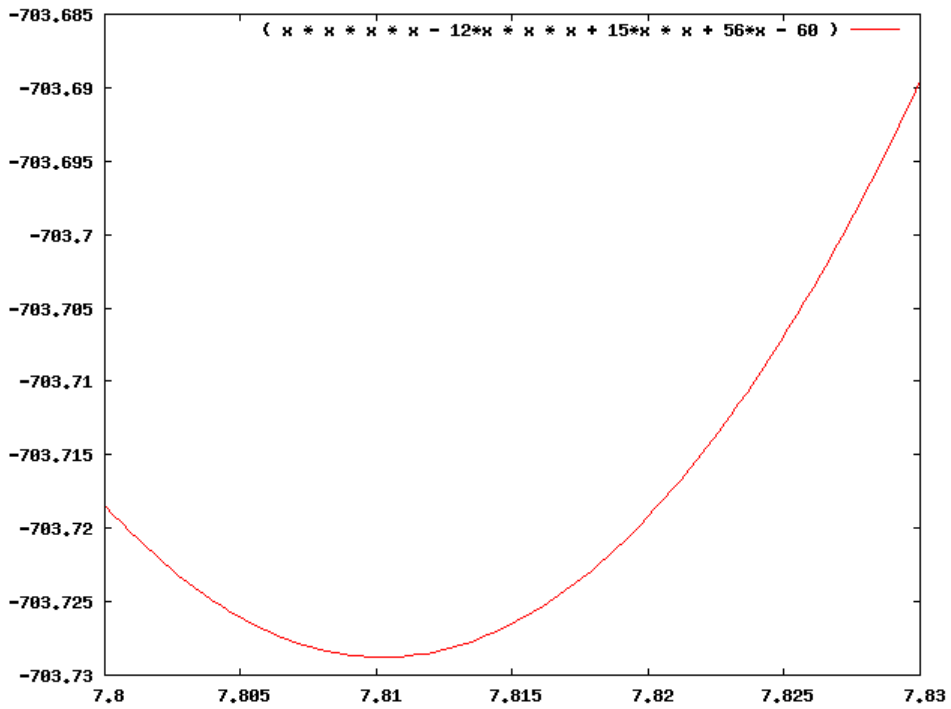


Figure 2: The function  $x^4 - 12x^3 + 15x^2 + 56x - 60$  graphed from 7.8 to 7.83 to clearly show its minimum value.

### 3.3. Ten-Variable Optimization

The ten variable optimization function is simple. The algorithm tried to maximize

$$\frac{(x1 \times x2 \times x3 \times x4 \times x5)}{(x6 \times x7 \times x8 \times x9 \times x10)} \quad (3)$$

where all variables are bounded between 1.0 and 10.0. To maximize the function, the numerator variables should be as large as possible and the denominator variables should be as small as possible. The algorithm was able to maximize the function after an average of 42,844 fitness evaluations.

## 4. Experimentation

Using the ten variable optimization, the type of crossover was varied to determine how it affected the number of fitness evaluations used to find the solution. The types of crossover studied are included below:

- One Point - A point is randomly chosen, and the bits to the right of that point are exchanged between parent chromosomes.
- Two Points, Second Left - A point is randomly chosen. A second point is randomly chosen to the left of the first point. If the first point is the first bit, the second point becomes the second bit. The bits in between the two points are exchanged between the parent chromosomes.
- Two Points, that Could Be Equal - Two points are randomly chosen. If both points are equal, one point crossover occurs. If the points are unique, then the bits between the points are exchanged between the parent chromosomes.
- Two Different Points - Two points are randomly chosen. The points are not allowed to be the same. If the same point is picked, it is discarded, and another point is picked. The bits in between the two points are exchanged between the parent chromosomes.
- Uniform - At each bit location, there is a 50% chance that the bits at that location will be exchanged between the two parents.

## 5. Results

All two point and one point crossover methods yielded almost equal results. The only significant deviation was uniform crossover, which performed much better than the other crossover operators. Five trials were used for each crossover operator. The average number of fitness evaluations required to find the solution is included below:

- One Point - 41,430
- Two Different Points - 42,578
- Two Points, Second Left - 41,568
- Two Points, that Could Be Equal - 42,307
- Uniform - 36,135

## 6. Conclusion

As expected, uniform crossover yielded the best results for this function optimization. This is probably due to the bit string representation. Each variable to be optimized is a string of thirty two bits. One point crossover, two point crossover where the crossover points could be equal, and two point crossover where the crossover points must be different each affect half of the variables on average. Two point crossover where the second crossover point must be left of the first crossover point should affect fewer variables. Uniform crossover usually affects all variables due to its operation on individual bits with a frequency of 50%. These variations of all variables seem to help keep the function from catching local maximums and local minimums in functions with multiple variables.

The five variables in the numerator were encoded next to each other, and the five variables in the denominator were encoded together. It would be interesting to see if alternating the encoding had any effect on the number of fitness evaluations required.

Future work could include testing the crossover operators on different types of functions. These functions could include multi-modal functions, single modal functions, and functions with small or large average derivatives. The hard part will be coming up with a clear way of separating functions into categories.