

Variations on a Simple Genetic Algorithm

*Scott H. Douglas
Rochester Institute of Technology, Rochester, NY*

Abstract

Variations in genetic algorithm (GA) parameters are explored to minimize the time needed to find a solution. The GA implementation attempts to maximize the number of zeros in a bit string. It has pluggable modules for doing different crossover methods and fitness functions. The basic GA code stays the same throughout the entire system. Variable parameters include mutation rate, crossover rate, crossover operator, copy rate, number of generations, population size, and chromosome length.

1. Program Design

Generic GA classes were created for Chromosomes and Populations. An interface was created for defining your own crossover functions and fitness functions. To use the system to solve another problem, you would simply provide a crossover function and fitness function. For fitness functions, large numbers correspond to better fitnesses. You pass these parameters into a Population instance, which creates the initial population. At this point, you can query statistics about the population. To advance to the next generation, you call `nextGeneration()`, which uses the functions you passed in from your main program. An overview of the system can be found in Figure 1.

1.1. Next Generation Algorithm

The `nextGeneration()` function first copies a certain percentage of individuals to the next generation either randomly or by their fitness values.

For the remaining individuals, it randomly selects two parents from the population. Parents with higher fitnesses are more likely to be picked than parents with lower fitnesses. Crossover is performed on a certain number of parents in the population based on the crossover rate. If the two parents don't undergo crossover, then they are copied to the next generation without modification. This process is repeated until the next generation is the same size as the current generation. Afterwards, mutation is employed based on the mutation rate parameter.

1.2. Good Genetic Algorithm Parameters

In this paper, the value of a set of parameters is judged by how well it minimizes the number of fitness evaluations required. The weakest link in most genetic algorithms is the fitness function, which usually requires the most amount of time to run. To find a solution quickly, the number of fitness function evaluations should be minimized.

2. Testing

The components of the system were tested individually. When combined, the algorithm converged on solutions. The output of each program was analyzed to make sure the algorithm was finding solutions in the number of fitness evaluations it reported.

¹Send correspondence to shd0326@cs.rit.edu.

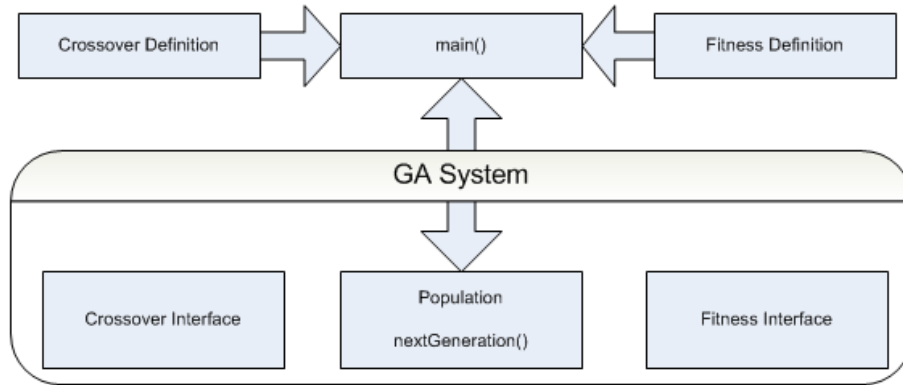


Figure 1: An overview of the genetic algorithm system implemented

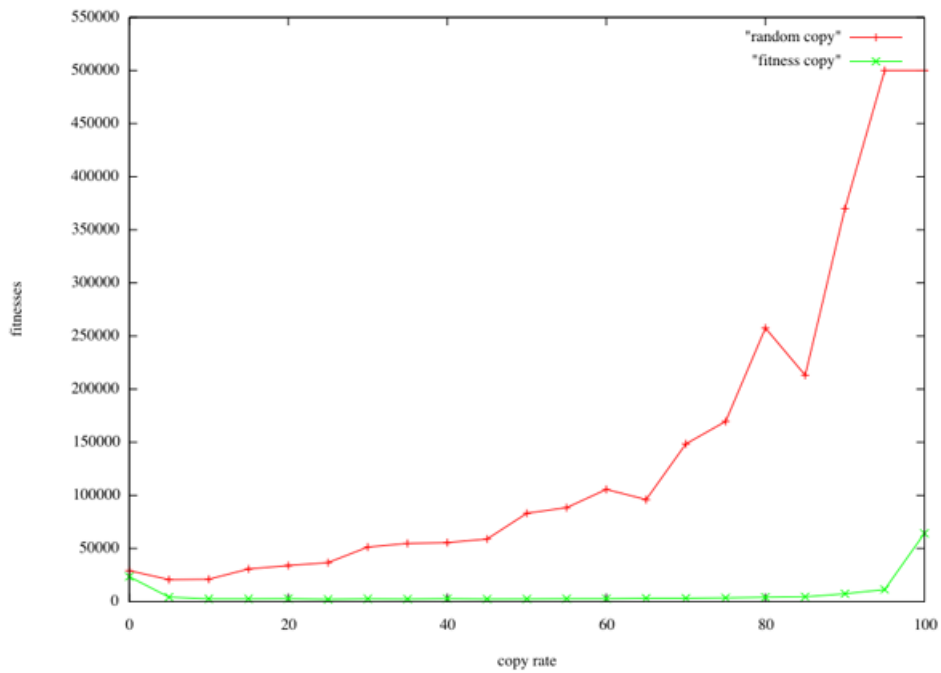


Figure 2: Fitness evaluations required to solve the problem while varying copying parameters. For random copying, a percent of random chromosomes are copied. For fitness copying, a percent of the chromosomes with the highest fitness are copied.

3. Results

3.1. Copy Rate Variation

For this series of tests we looked at the effect of copying a certain percent of the individuals from one generation to the next generation. The first test copied random individuals to the next generation. The next test copied top performers to the next generation. The results are shown in Figure 2.

Randomly selecting a percent of individuals to go to the next population is akin in natural selection to a certain number of individuals surviving by luck. For small random copying rates (5-10%), an improvement was shown in the number of fitness evaluations required to solve the problem. After this, the number of fitness evaluations required increased exponentially until it can no longer solve the problem with copying rates over 95%. This is due to a lack of change between generations because too many chromosomes are copied exactly without modification.

Copying individuals based on fitness lowered the number of fitness evaluations required to solve the problem for copy rate values between 5% and 85%. After 85%, the number of fitness evaluations required began to increase, although still less than when not copying any individuals. The only exception here is when copying all the individuals between populations.

In two scenarios, both copying processes should be equivalent. When copying no individuals to the next generation or when copying all individuals to the next generation, the copying processes have no effect. For these two scenarios, both copying processes should produce similar results. Our results supported this for copying no individuals to the next generation. When copying all individuals, the results diverged. The random copying algorithm could not find solutions when copying everyone between populations. The fitness copying algorithm found a significant amount of solutions. This discrepancy could be because of randomness. With either algorithm, the ini-

tial population will be the final population. The only way to find a solution in this case is to have the solution be randomly generated in the initial population. This aspect of the results should be looked into more.

3.2. Crossover Point Variation

Figure 3 shows the results of varying the number of crossover points in crossover from zero to the length of the chromosome, 30. The graph is roughly symmetrical around length / 2 crossover points. When no crossover occurs and a lot of crossover occurs, the algorithm could not solve the problem in a reasonable amount of time. The sweetspot for the chromosome of length 30 is 10 crossover points.

The results make sense. If you don't crossover enough, there isn't enough sharing of genes. If you crossover too much, good sections of individuals get split up a lot.

3.3. Crossover Rate Variation

Figure 4 shows the results for varying crossover rates of one point crossover. The more crossover you do, the better the algorithm performs. The sweetspot of the crossover rate was 90% crossover. This allowed some individuals with high fitnesses to be copied directly to the next population, which concurs with the results in Figure 2 that copying some individuals to the next population based on fitness decreases the number of fitness evaluations required to solve the problem.

3.4. Mutation Rate Variation

Figure 5 shows the results of varying mutation rates from 0% to 10%. Mutation rates above 10% failed to solve the problem in a reasonable amount of time. No mutation results in a spike of fitness evaluations required because not enough variation is introduced when forming future generations. The sweetspot of this mutation rate was 1% mutation. Any more mutation increased the

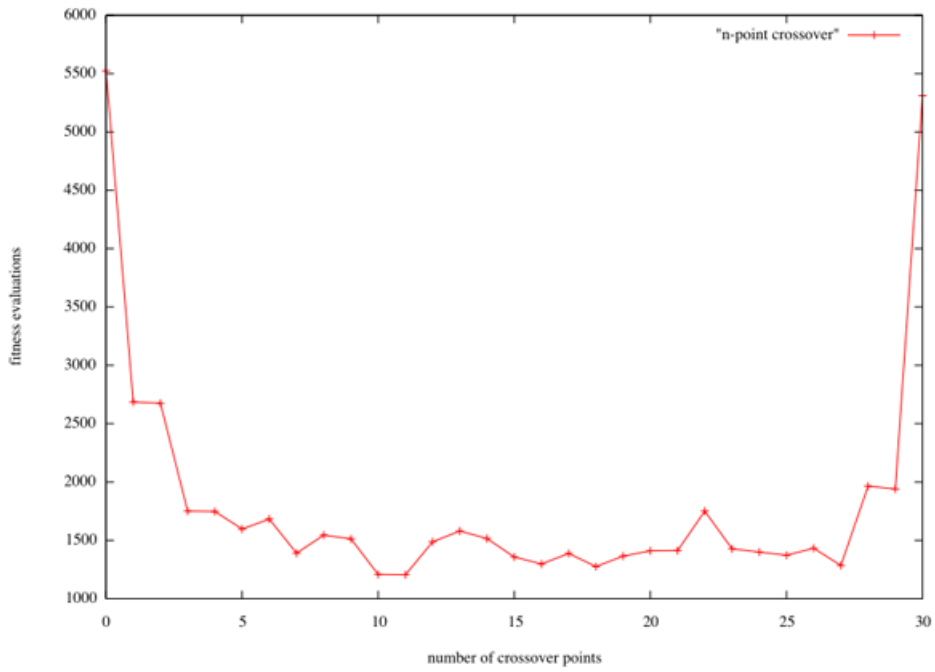


Figure 3: Fitness evaluations required to solve the problem while varying the number of crossover points out of a chromosome with a length of 30.

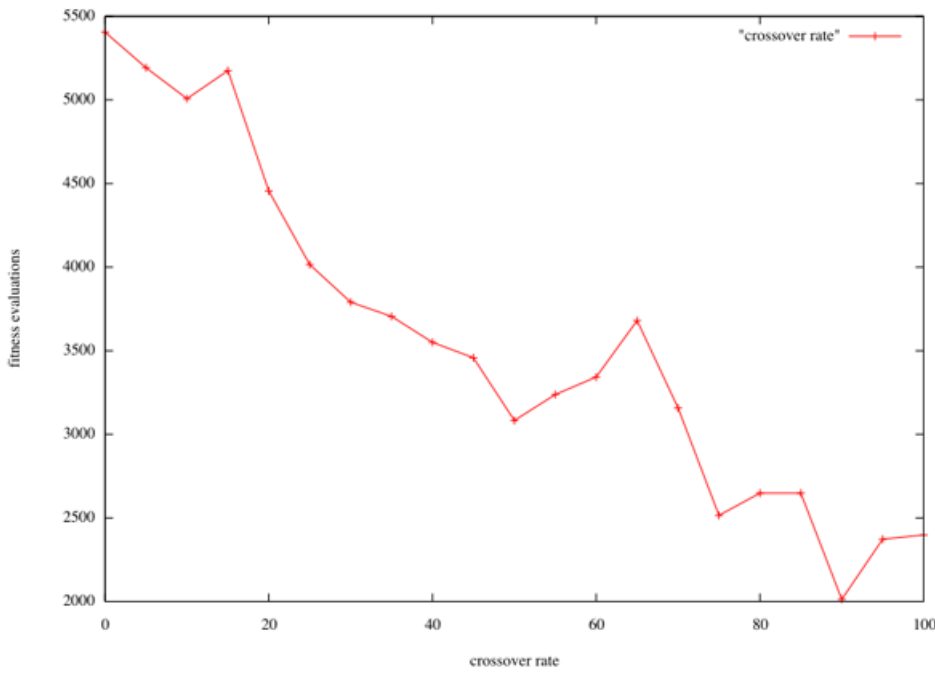


Figure 4: Fitness evaluations required to solve the problem while varying the crossover rate.

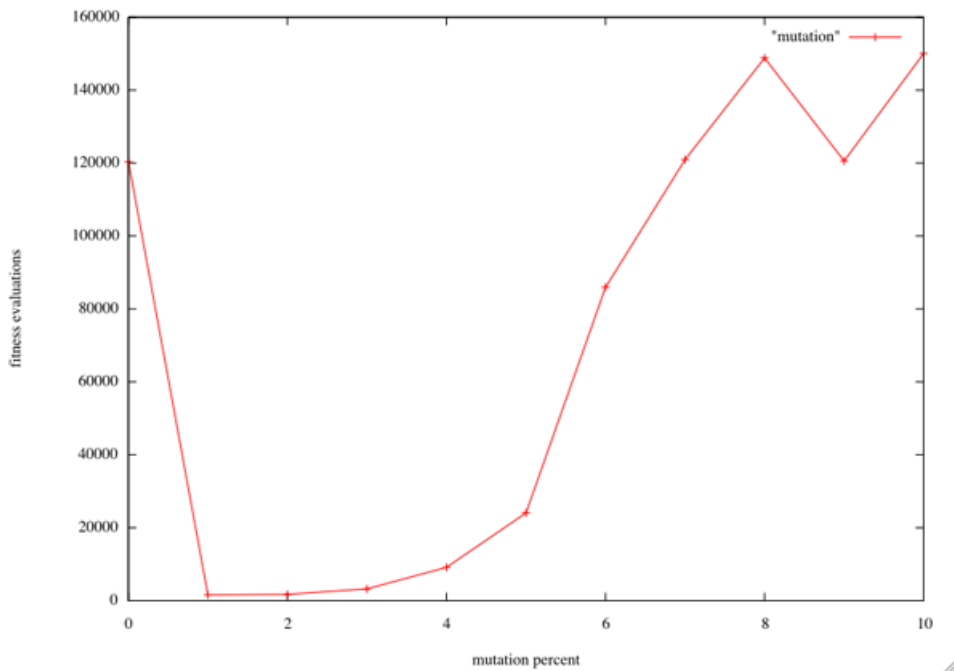


Figure 5: Fitness evaluations required to solve the problem while varying the mutation rate. One through ten percent mutation rates are shown. Everything above ten percent did not finish in a reasonable amount of time.

number of fitness evaluations required. This makes sense because a lot of mutation would break up good genes and prevent them from being passed on.

4. Conclusion

In this paper, we have discussed how varying parameters to a genetic algorithm can affect the number of fitness evaluations needed to find a solution. This has direct impact on the amount of time required to compute the solution.

Copy rate variation showed that the number of fitness evaluations required is reduced if good individuals get to the next generation unchanged. Keeping good chromosomes around to crossover later in a different way may result in better crossovers in the next round that wouldn't happen in this round due to population size constraints. This is similar to the results shown from crossover rate variation because if crossover is only performed n percent of the time, $100 - n$ percent are copied to the new population unchanged.

Crossover point variation showed that multiple crossover points can produce better results by increasing randomness. Experiments should be performed with different length chromosomes to see if the length / 3 metric is a good metric for determining the number of crossover points. The need for randomness was also displayed in the variation of mutation rate. Some variation is successful at reducing the number of fitness evaluations required, but too much variation is counter-productive. As soon as a solution is being zeroed in on, it will be mutated to something else.