An Analysis of Genetic Algorithms for String Matching

Connor Clayton, Jesse Corson, and Spencer Tubbs - Group 11

Abstract - This paper represents an exploration on how we can use genetic algorithms to solve problems. This will include an analysis on the algorithm used, as well as an exploration on how varying inputs have drastic effects on resolution speed.

1. Introduction

Genetic algorithms are currently being used in a wide array of fields. Shipping companies have used genetic algorithms to optimize routes by solving the Traveling Salesman Problem [1]. Robotics companies have used genetic algorithms to learn proper behaviors and motions [2]. Genetic algorithms are also used for a wide range of genetic and molecular biological problems. [3] These are just a few examples of this highly useful technique.

You may ask, "what is a genetic algorithm?". By the end of this document we're hoping to have given you a better understanding of what a genetic algorithm does. PHD student Vijini Mallawaarachchi sums up what a genetic algorithm is rather simply:

> "A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation." [4]

What is considered as "fit" is specific to the problem you are trying to solve. In our problem today we are trying to find a string that matches a target string *exactly,* character-for-character. A given string is more "fit" than another string based on how many characters are in the correct position when compared to our target string.

2. Algorithm

Our algorithm works as follows:

a. Establishes an initial population of 16 individuals. Each individual is an 8 character long string composed of random integers.
b. Calculate each individual's fitness level. Fitness for our algorithm gives points in a linear fashion for each character in the correct position. Our target string we searched for: “12345678”.
c. Create a new population by selecting pairs of individuals to “reproduce”. Individuals will be more likely to be selected based on their fitness values.
d. Reproduce - members of the new population are created by performing a cross between parents. A random mutation can also occur here.
e. After we have a new population, we check to see if any members of the new population match our target string. If not, we repeat the process beginning at step b.

3. Experiments

We ran a total of 5 experiments each testing a different variable. However, we used the same default values for each experiment. Our default values for each experiment are as follows:

a. <u>Individual length:</u> 8 characters
b. <u>Target String:</u> “12345678”
c. <u>Population Size:</u> 16 individuals
d. <u>Mutation Chance:</u> 20%
e. <u>Potential Values per Character:</u> 1-8
f. <u>Perfect Match:</u> Yes

Each experiment will detail which variable was under test and the results found from altering that variable.

We performed 5 separate experiments. Each experiment was exposed to 5 different values for our dependant variable, with each value being run 5 separate times for a total of 25 computations per experiment.

Experiment 1

Variable under test: Population Size

We decided to see what effect population size would have on finding our target string.

The number of generations it took to find our target string for each of our population sizes was as follows:

Population Size	Avg # of Generations
4	176021
8	3124
16	110
64	24
1024	7

See chart A on page 7

It was most surprising to see how limiting a small population size can be. The difference between a population of 4 and a population of 8 is drastic. We also noticed that there is a point to where increasing your population size has diminishing returns. The efficiency increase from a population size of 16 to 64 was about 86 generations. The increase from a population size of 64 to 1024 was only an efficiency increase of 17 generations.

Increasing your population size simply means more opportunities to find an individual you are looking for, and the results show that effect.

Experiment 2

Variable under test: Target String Length

The second variable we decided to test was the target string length that we are searching for. This is another factor that we were expecting to see some drastic variation in resolution times. The word lengths and results are shown below:

Word Length	Avg # of Generations
2	6.8
4	32.8
6	96
8	278
16	322588

See chart B on page 7

We found that word length doesn’t seem to have a huge factor in generation times with smaller numbers. We really don’t see the effect of word length until we tested it on a length of 16. This can be easily explained by exponential growth. Of all the variables tested across all experiments, slightly changing the length of our target string seemed to have the most drastic effect on resolution times.

Experiment 3

Variable under test: Mutation Probability

Our third experiment we decided we would test how mutation probability would affect our results. Mutation occurs during the reproduction step of our algorithm, as a crossover is being made between two individuals of a population. Here are the results:

Mutation Probability	**Avg # of Generations**
10%	192.6
20%	211.8
50%	178.4
80%	516.8
100%	1185.2

See chart C on page 8

We found that this experiment produced the most surprising results out of all the experiments performed. This experiment was also the only experiment that produced a graph with a local min/max value. All other experiments produced results that were either always increasing or always decreasing. For us, mutation is mostly beneficial somewhere around 50% of the time.

Individuals of the population are dependant on mutations occuring to get the values that they need. If a certain character is rare in a population, mutation provides a way to increase the occurrence of that character. Because of this, if mutation rates are too low we don’t gain the benefit of those characters being introduced into our population and more reproductions must occur to achieve our results.

Experiment 4

Variable under test: Stopping Condition

The fourth variable we put under test was the overall stopping condition. Another way to think of this is the fitness benchmark we are trying to achieve. By default we were only accepting strings that were flawless as our final product. We wanted to test what would happen if we allowed a final string to contain a certain number of errors. The results of experiment 4 are shown:

Stopping Condition (# of wrong characters allowed)	Avg # of Generations
6	1
4	6
2	44
1	96
0	135

See chart D on page 8

As you decrease your requirement for a perfect string your solution times also decrease. This experiment produced similar results as Experiment 2 which tested total string length. This experiment is something we feel like you would see in nature. Rarely in nature would your fitness requirement be perfect. This is another reason why we decided to test these different fitness levels.

Experiment 5

Variable under test: Number of values per letter.

The last variable we tested were the number of values possible per character. Here were the results:

Values per Letter	Avg # of Generations
2	8.2
5	112.8
8	183.2
9	223
10	206.6

See chart E on page 9

We found this to be an applicable study because we see these vary amounts in nature. When dealing with DNA we have 4 potential values. Alpha characters we have 26. In our case we were using digits so we picked a range of values just using digits.

The time difference between number of values per letter didn't change as drastically as we were expecting. This is also another case of exponential growth. As you approach more values per letter eventually our generation amount would begin to increase drastically as well.

Conclusion

For most experiments we ran an increase in complexity meant a decrease in probability. Whenever probability of a string being found decreased the number of generations it took to find a solution increased. This pattern was true for all experiments except for Experiment 3.

Experiment 3 functioned by altering the random probability of a mutation occurring. This experiment produced the most unpredictable and surprising results out of all of our variables tested. With the values and amount of tests we ran it was hard to determine what an optimal value would be for mutation rates. Future experiments with more values tested would be able to determine the optimal value for mutation rates according to our simulation.

We can see the power of mutation by looking at the influenza virus. One reason why you have to consistently get new vaccines year after year is due to the viruses incredible ability to adapt quickly over time. It does this through mutations.

Dr. Karron with the Center for Immunization Research stated: “It’s not that they know which changes will outwit the human immune system, but they’re replicating very rapidly.” [5] Rapid replication means you achieve more generations over a small period of time. More generations means a higher potential for change. Couple these rapid generations with a high mutation rate and you get a lot of change, which is exactly what a virus is looking for.

In the end, finding a solution always came down to churning through generation after generation and letting the genetic algorithm do what it was built to do. What works in nature seems to work in string-matching as well.

Chart A

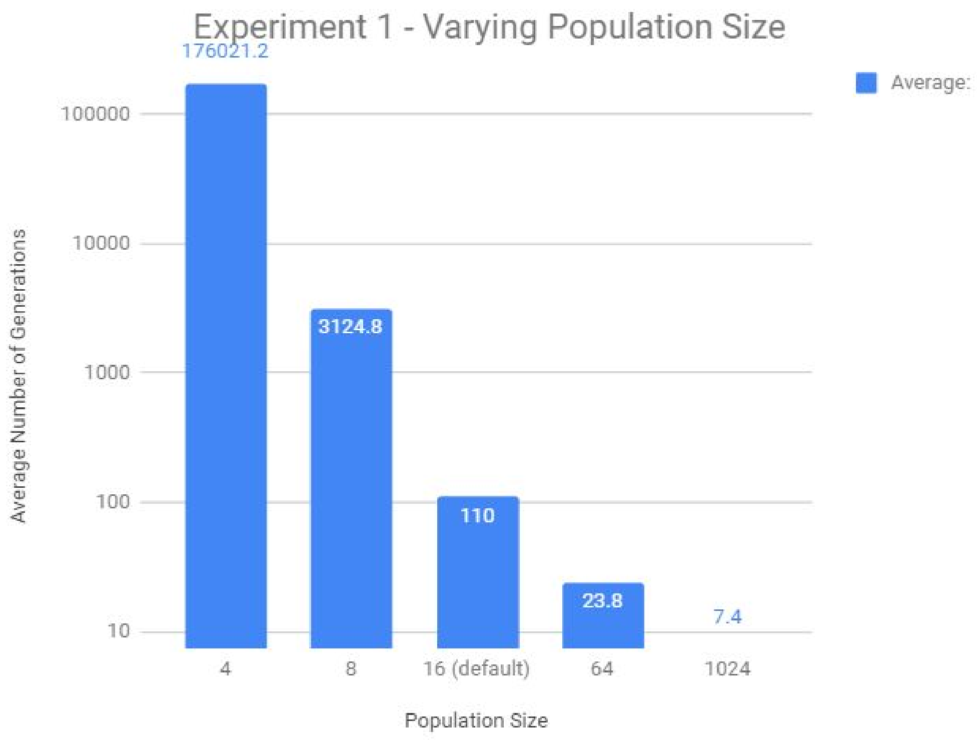


Chart B

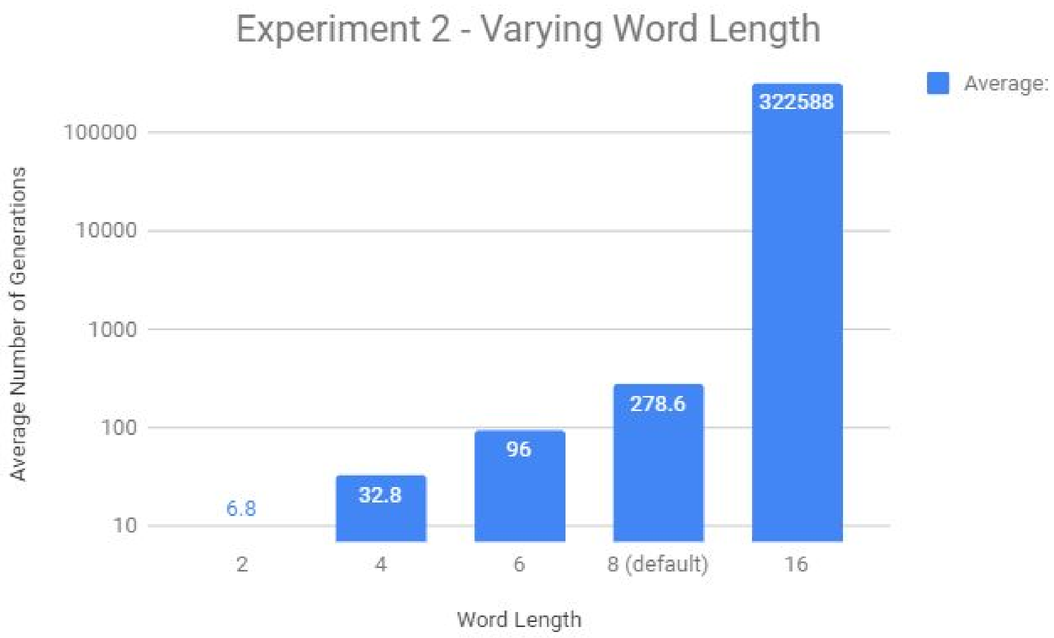


Chart C

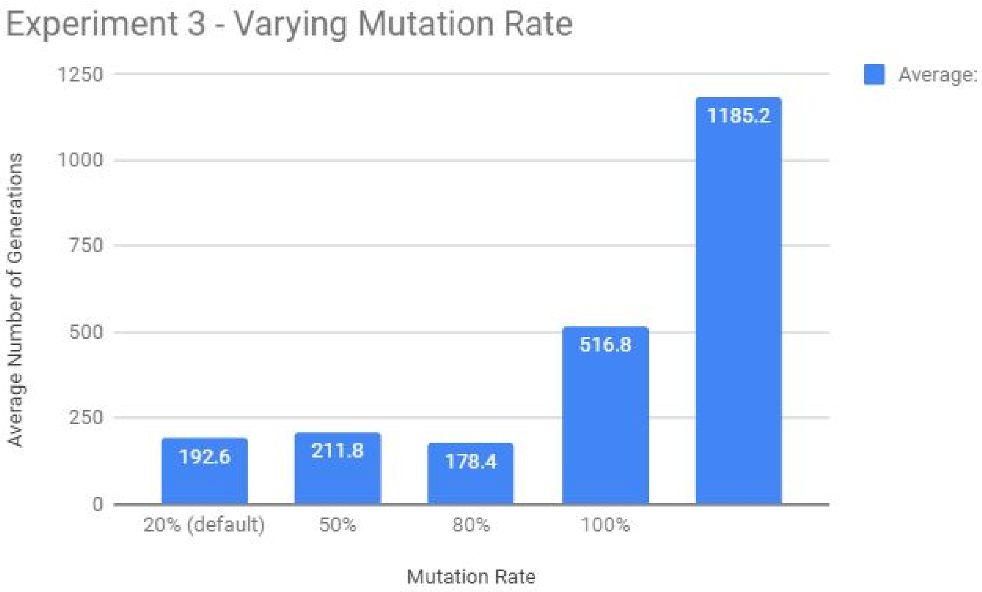


Chart D

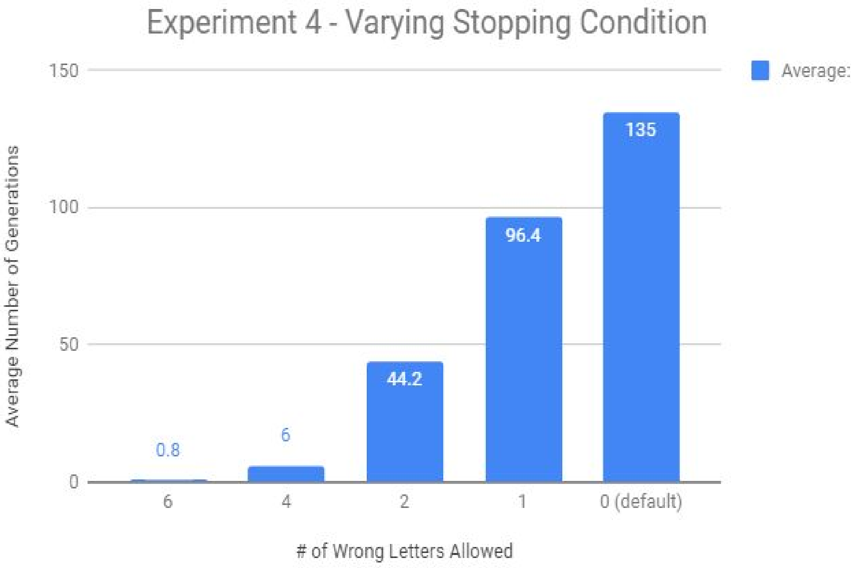
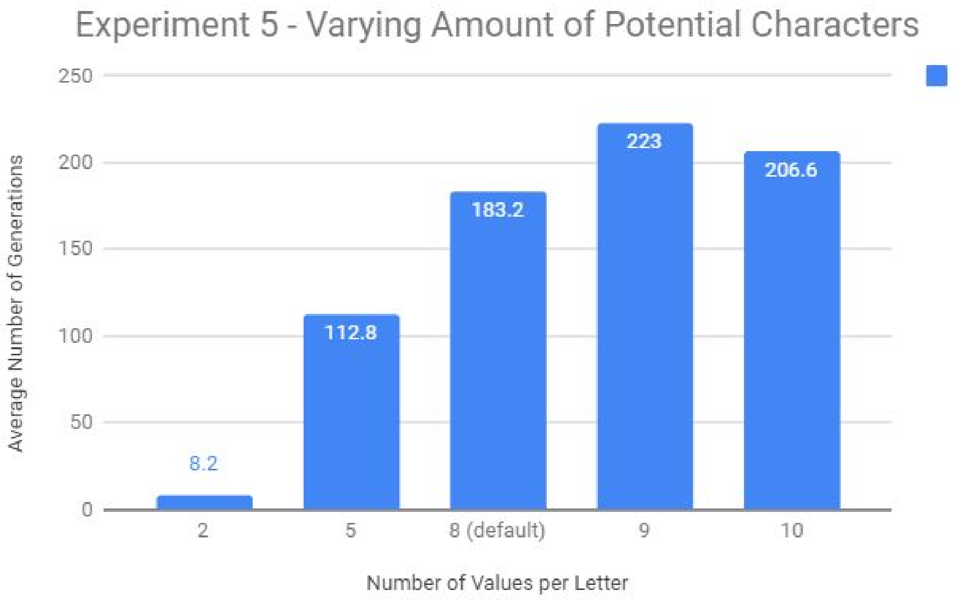


Chart E


Experiment 5 - Varying Amount of Potential Characters
250
200
150
100
50
0
Average Number of Generations
8.2
112.8
183.2
223
206.6
2
5
8 (default)
9
10
Number of Values per Letter

Works Cited

[1] Grefenstette, John, et al. “Genetic Algorithms for the Traveling Salesman Problem.” *Indiana University*, Vanderbilt University, www.cs.indiana.edu/~vgucht/Genetic_Algorithms_for_the_Travelling_Salesman Problem.pdf.

[2] Walker, Matthew and Christopher H. Messom. “A Comparison of Genetic Programming and Genetic Algorithms for Auto-tuning Mobile Robot Motion Control.” *DELTA* (2002).

[3] Levin, Michael. “Application of Genetic Algorithms to Molecular Biology ...” *Tufts University*, Harvard Medical School, ase.tufts.edu/biology/labs/levin/publications/documents/1995ga_dna.pdf.

[4] Mallawaarachchi, Vinini. “Introduction to Genetic Algorithms - Including Example Code.” *Towards Data Science*, Towards Data Science, 8 July 2017, towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.

[5] Haelle, Tara. “Why You Need the Flu Shot Every Year.” *The New York Times*, The New York Times, 12 Dec. 2017, www.nytimes.com/2017/12/12/smarter-living/why-you-need-the-flu-shot-every-year.html.