**Genetic Algorithms for Strings**

**(String Evolution)**

**Written By:**

Ryan Taylor Dickey

and

Daniel Huan Nguyen

During class, we briefly discussed genetic algorithms which peaked our interest as well as ignited the fuel for this paper. These algorithms are “adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics”[1]. The most interest fact about this topic is that these algorithms give the machine the ability to “learn” how to do a task as opposed to teaching the machine how to do that task. An example of this comes from an article we followed during the process of creating our game. Burak Kanber says, “hand-coding a walking routine will almost certainly fail. Even if you succeed in making a robot walk, the next robot that comes off the line might have a slightly different center of balance, and that algorithm you slaved over no longer works. Instead of enduring the inevitable heartbreak, you might use a GA to ‘teach the robot to learn to walk’ rather than simply ‘teaching the robot to walk’“[2]. Genetic Algorithms are used as an informed search technique in order to obtain the solution to the problem, or in most cases, a “close enough” solution to the problem. In general, genetic algorithms do not know the end goal to the problem at hand. Therefore, they must use a fitness value, or its inverse, a cost value, in order to find the solution. Just as with biology, the algorithm involves alleles connected together to form genes which are then stranded together to make a chromosome. When you create multiple chromosomes, you create the population. The convention follows as the strongest survives and the weakest dies. The strong chromosomes live on to mate with each other in order to produce children and keep their genes alive. There are chances where generations aren’t making progress when only mating, we can prevent this by allowing mutation on chromosomes to kick start a mating process where two new children will be born. Since the mutation is used for diversity in the generations, then we do this less frequently than the mating (crossover) process[4]. We have touched on some main key phrases. Since we are talking about strings for our paper, we will use strings for examples to explain the process of genetic algorithms. After we define the terms, we will discuss the process taken by genetic algorithms to fully understand what is going on to produce an output that is close to the solution, or is the solution, we are looking for.

**Alleles**

In biology, alleles are variations of genes[5]. In computer science, we refer to these as the binary digits (bits) the computer uses to manipulate data (i.e. 0’s and 1’s).

**Genes**

Genes can mutate to take alternate forms[5]. In computer science, we term bits strung together to form relatable data as genes. For instance, when you have the letter ‘a’, in binary it is 01100001.

**Chromosomes**

When you string several genes together you form words. An example would be the word ‘at’ which has a binary flow of 01100001 01110100.

**Population**

The collection of these chromosomes in a space is what we call the population. In order for our chromosomes to evolve, we need to have more than one chromosome in the population.

**Mating**

Mating involves taking two chromosomes which have the best value (fitness or cost depending on which method suits your need). Your algorithm will have a crossover point where it takes two strings and swaps the last part. For example, if you have the words Supetruck and Firerman, and you set the crossover point to be right after the fourth character of each word, after the e, then once you mate the two strings you would produce the children Superman and Firetruck. Doing this concept over generations will eventually produce children with value levels close enough to the desired results.

**Mutating**

When your mating becomes stagnant then you need to give it a push. Changing one gene will produce a desired result for your mating process to thrive again. The programmer will place the method for mutating chromosomes in the code. Each chromosome is given a chance to mutate. Then the algorithm will finish and produces the best answer it could. Mutations keep the population diverse.

**Cost Vs. Fitness**

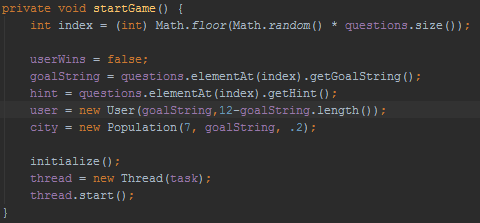
If a genetic algorithm knows the end goal, as we did with our code, then the goal value of whichever method they use would be the “exact match” value. For instance, if you are using the fitness value for your algorithm, then you will know the informed search method found the exact match once its fitness value reaches 100%. Inversely, if you are using the cost value for your algorithm, then you will know the search found the exact match once its cost value reaches 0%. However, as we stated before, the algorithm typically does not know what the end goal is, and therefore, would find a best solution for the problem.

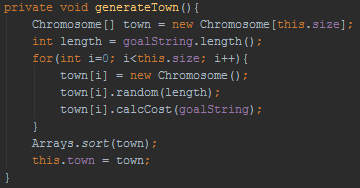
**The Process**

The process of a genetic algorithm is exactly how you would expect it to be. The strong survive and the weakest die. So what steps are there and how do you jump start this process? There is a good scholarpedia article[3] which discusses one common set of steps to take. Though, there can be more than one way to create your genetic algorithm. Let’s discuss the steps from the article. First, you start with creating a “population of N individual strings of”[3] genes. Next, select the two chromosomes with the best fitness/cost. Then you would mate them using the crossover process discussed above with mutations occurring every once in a while. This produces the two children which are considered the “next generation”[3]. After this, you repeat the process “N/2 times to produce a new generation of N individuals”[3].

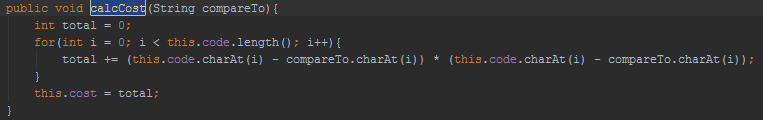
Knowing all this, we were interested in seeing this work in a game. Our idea for implementing genetic algorithms to solve a problem is to build a game which put a user against the computer. The game in mind was inspired by Riddle Me That by SecretBuilders Games. Riddle Me That gives a riddle which the answer is one word between 3-7 characters long. The user sees these characters as blank spaces. Underneath the blank spaces are 12 jumbled up letters. These letters consist of the correct letters for the answer as well as random letters to throw the player off. This version is not against a computer and has no time limit in answering questions. It would be interesting to use genetic algorithms for strings in order to make a person versus computer styled gameplay where the game would have questions pertaining to a topic of the user’s choice (i.e Artificial Intelligence). The start of the game is just a menu screen with a button called Start Game. The user will click the button and the screen will load the game screen. The game screen consists of the user question pane, the computer’s progress pane, the user’s answer field, twelve panes of jumbled letters (some of which are contained in the answer), and a reset button to ease the annoyance of having to reload the game. The question pane houses the question the user has to answer as well as the hint (underlines) to note how big a word is or if it is a two word answer. When the user enters correct letters in corresponding positions, those letters are filled in the hint on the question pane. The computer’s progress pane shows the progress, in percentage, that the computer has done. Recall that we are using the cost value which is the inverse of the fitness value. Therefore, our algorithm is attempting to make its chromosome child a value of zero. However, we use simple mathematics to give us the percentage going from zero to one hundred percent. Once the computer’s progress percentage reaches one hundred percent, this means the cost value of the child chromosome is zero and the solution has been found. The user is informed that they have lost and the computer’s pane shows the correct answer. The answer field is where the user will type their guesses. Once they type their guess, they hit the enter key to compare their guess against the solution. For each character they get correct, the hint will reveal that character. The twelve panes on the bottom contain the letters of the word plus random letters to throw the user off. These panes are not clickable. They are strictly to guide the user with certain letters to use. The reset button brings the user back to the main menu with the start game button.

Once the game has started, the user will have a couple of seconds to read the question before the computer begins solving the problem. It will use genetic algorithms for strings by taking the number of chromosomes we want in the population, the goal string, and the chance of mutation, and calling the function to begin the process of mating and mutating chromosomes for the solution. We created a new thread named task which we call the function to allow the computer to find the answer in parallel with the user guessing. When the user hits the start game button, the program calls the method startGame().  The functionality is as follows:

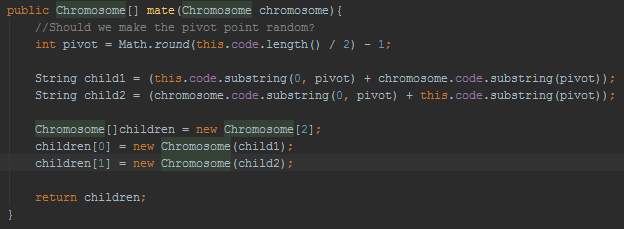


Notice the city variable being assigned as a new Population class with the parameters we specified above. This creates 7 chromosomes with a 20% chance at mutating. The creation of the chromosome is in the Population class. In this code, we created a new Chromosome array of the population size. Then we traversed the array to add a new Chromosome to each cell.  
                      

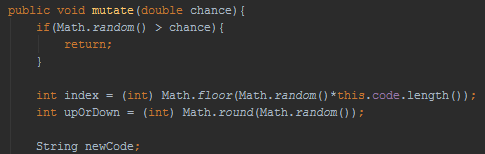
Next, I want to discuss the cost value. This is how the computer knows how close it is to finding the solution. The function takes in the goalString as a parameter, then it sets a new integer variable, named total, to zero. After that, it goes through a for loop from of the childs length, then subtracts the goalString’s ASCII number from the childs ASCII number at i and squares it. These values are then stored into total to be assigned to the chromosomes cost value.



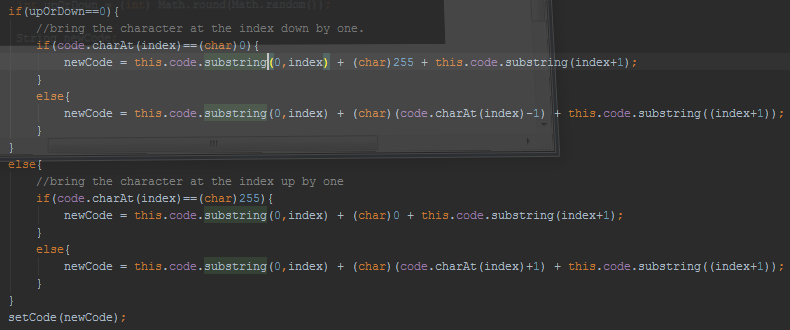
The last two functions I want to show you in this paper are the mating and mutating functions. First, the mate function. This function takes is called using a chromosome and is sent another chromosome as a parameter to mate it with. The pivot point is where we are going to be doing the crossover operation on in the string. We create the two new children based on the pivot point and then assign the two new children to an array named children.



The last thing I wanted to show was the mutation method we wrote to keep diversity in the population. At first, we check if the chromosome is going to mutate. If the chance is greater than the number generated by Math.random() then we will continue with the mutation.



Continuing with the mutation, we assign the index we will be mutating, whether we will be going up or down with the mutation, and create a new string named newCode to store the new string in. Next, we check if the upOrDown variable is 0. If it is, then we will bring the character at the index down by one, otherwise we will go into the else portion of the code to bring the character at the index up by one.



This has been a very brief overview of the code and you are welcome to go through the code to gain an understanding of the behind the scenes of StringEvolution.

In conclusion, we have used bits, bytes, and words as well as related them with alleles, genes, and chromosomes from biology to form a population. We have seen that we can write computer programs to use natural selection to find a solution to a problem at hand by mating and mutating chromosomes to form new generations after generations of children chromosomes with each child, more or less, getting closer to desired fitness/cost value and more importantly the end goal.

**References**

[1]<http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/hmw/article1.html>

[2]<http://burakkanber.com/blog/machine-learning-genetic-algorithms-part-1-javascript/>

[3] <http://www.scholarpedia.org/article/Genetic_algorithms>

[4] [What is the role of mutation and crossover probability in Genetic algorithms](http://www.researchgate.net/post/What_is_the_role_of_mutation_and_crossover_probability_in_Genetic_algorithms)

[5] <http://www.diffen.com/difference/Allele_vs_Gene>