A Comparison of Two Computer Simulated Populations of Animals Generated by Genetic Algorithms

Nick Bruhnev
4/11/2016

Table of Contents:

All the code for this project can be viewed at
github.com/nickpeterb/GAproject.git

Purpose

The purpose of this project is to determine which of two simulated populations are larger after 5,000 generations: a population of mere foragers, and a population of predators. While the animals generated in this simulation are entirely hypothetical, the algorithm could very easily be modified to reflect the attributes, mating habits, and mutation of very real animals. This could allow for more accurate prediction of the population of future generations, especially for animals that may be endangered or on the brink of extinction.

Hypothesis

If simulation #1 is created so that the animals need must forage for food, and simulation #2 is created that animals must hunt for their food, then after 5,000 generations overall population of animals in simulation #1 will be higher because predators must deal with various forms of resistance by prey, whereas the foragers need only find their food, and be big enough to eat it.

Review of Literature

No one can deny the ubiquity of computers in this day and age, and yet they have never strayed from their original purpose: problem solving. The first computers, were, of course, giant calculators, and for all the amazing things they can do now, in reality all that is going on underneath are very fast calculations. So it would come as no surprise that programmers, the people who turn computers from hunks of metal to the most mind blowing creations of humankind, have come up with more ingenious and efficient ways to solve problems. Specifically, solving problems by mimicking parts of our real world. This paper will explore one type of such program, genetic algorithms. Genetic algorithms have existed for some time now, and have made themselves very useful. In one notable example, a genetic algorithm was used

by NASA to determine the the best shape for the antenna of the ST5 spacecraft. The algorithm was able to create an antenna with a much higher radiation pattern than any human could have made. In this experiment, one genetic algorithm will contain a population of herbivores that will forage for their food, and another algorithm with a population of carnivores that will hunt for their food. The populations of each animal will vary over time depending on the different algorithms, making the two algorithms the independent variables and the populations the dependent variables. All of this will be able to run on any computer, and will be accurate enough to draw conclusions from. Those attributes are what allows genetic algorithms to teach us a lot about our real world, as will be done in this paper. ("Evolved Antenna," 2015)

Genetic algorithms (GAs) are really a method of solving any problem that requires the optimization of some process. This gives them the potential to be useful in many fields. They can be used to optimize a route from point to point to use the shortest path, or the most efficient moves for a robot, or, in this case, the best way to find food and the best way to avoid predators. They solve these optimization problems by simulating evolution, in other words, optimization by natural selection. GAs are a type of artificial intelligence, alongside machine learning and other such programs that imitate physical phenomena. They belong in the type of artificial intelligence dubbed evolutionary algorithms, that deal specifically with using natural selection. What differentiates genetic algorithms form other forms of evolutionary algorithms, such as genetic programming, is how the data is expressed and stored in the algorithm. In GAs the data that the algorithm is evolving, the “animals”, are simply sets of numbers or letters that indicate different traits in that animal, whereas in genetic programming is much more complex where each animal is its own separate artificial intelligence using a method that simulates a network of neurons in a brain. However such a level of complexity is not always needed.

Ultimately the goal of a GA is to use numbers and math to simulate evolution unto a set of data. (Dolan, 2009) ("Genetic Algorithms in Plain English," n.d.) (Holland, 2012) (Hosch, n.d.)

That data set, in a genetic algorithm, is called the chromosome. Much like a real chromosome, chromosomes in GAs store information about the animal. In humans, our chromosome is encoded as DNA, in a double helix. While humans have many sets of chromosomes, a chromosome in a GA contains all of the genetic material of the animal. A chromosome in a GA is really the animal itself. These chromosomes are most often stored as long sets of numbers, often stored as one long number. Each number is called an allele, because, just like real alleles, they are what make each chromosome different from one another. Chromosomes are just a string of alleles, represented as numbers. Unlike in humans, alleles in GAs also often determine the behavior of the individual. They may also store information about the animal's surroundings, which could be related to the information that is gathered by our senses. Exactly what is stored by the chromosome and separate alleles depends on the specific genetic algorithm being written, what exactly it is optimizing for, and how the rest of the program is structured. The chromosome is central to the genetic algorithm, because without it the rest would fall apart. The very first step executed by any genetic algorithm is creating a new, completely random population of organisms. So it does this by creating many completely random long numbers and having those be the chromosomes of the initial population. To achieve its goal of properly simulating real world natural selection, GAs need to first simulate the organisms themselves. ("Crossover (genetic Algorithm)," 2015) (Eck, 2011)

Now that the all the "animals" have been created, it is time to start the process of natural selection and evolution. Natural selection is a simple process: the organisms that have not evolved to adapt to their environment naturally die off, whereas the ones that have adapted

survive and then reproduce. Knowing that, the first thing a genetic algorithm needs to do, right after generating the initial random population, is evaluate them, so the subpar ones can later be killed and the rest allowed to reproduced. Here is what is called the fitness function. It takes in a chromosome and gives it a score based on a series of criteria. In nature, these criteria are based on the particular animal's environment. However, in a simulated environment like a genetic algorithm the criteria that the organisms must adhere to to survive are completely at the discretion of the programmer. This is why GAs can been applied to many different kinds of optimization problems, because the alleles of the chromosomes can be made to represent anything, and the fitness function can be made to evaluate them on any number of rules. The fitness number of a chromosome is what allows genetic algorithms to solve optimization problems. The program is always optimizing for either the highest fitness possible or the lowest fitness possible (less can be better in some cases) . Once the most optimum (highest or lowest) fitness has been reached, the problem is said to have been solved. ("Genetic Algorithms in Plain English," n.d.) ("Genetic Algorithm Hello World," n.d.)

Now that all the chromosomes have been evaluated, the ones that are not fit enough need to be removed. Genetic algorithms do this in the next step, referred to as selection. The selection function takes in all the scores of the chromosomes and uses math to select which animals will be allowed to live on. There are multiple types of selection, which use different maths to select a slightly different portion of the population. All of them have the end result of selecting the chromosomes with the best fitness scores, but they are not identical and each have pros and cons and are to be used in different circumstances. The most simple of selection methods would be to just pick the top 50% of chromosomes, but this is never used because it is not what really happens in nature (it is not always the top 50% that survive in the real world). One of the most popular types of selection is Roulette Wheel Selection because it does a good

job of selecting members of the population in a manner proportional to the overall fitness of the population. It does this by assigning a probability of selection to each member, in a way so that the larger the fitness of the member the larger the probability of selection. The larger the fitness (or smaller if it is that kind of fitness function) the larger the chance it has to survive. (Dolan, 2009)

Like all organisms in the real world, these survivors have one ultimate purpose: to reproduce. This allows them to pass on their superior genes to the next generation. In genetic algorithms, reproduction is called crossover, because it works in a slightly different manner. Chromosomes are randomly paired together to be parents, and then produce usually two children that are a combination of the alleles from both parents. The most common types of crossover are Cut and Splice crossover and Two Point crossover. Cut and Splice crossover randomly chooses a point for each parent in each of their string of numbers that is their chromosomes. Then Child 1 receives the left half of Parent 1's chromosome (left of the random point), and the right half of Parent 2's chromosome. Those are combined to make Child 1's chromosome. The opposite occurs for Child 2, where it receives Parent 1's right half of the chromosome and Parent 2's left half. Since the point at which the chromosomes are split is random (not always an exact half), this results in the chromosomes of the children being different lengths than that of the parents and each other. Two Point crossover works quite differently, where two points are chosen in each of the parent's chromosome, except the points are exactly the same in both parents. This divides each parent's chromosome into 3 unequal parts. Child 1 receives Parent 1's first third of their chromosome, Parent 2's middle third, and Parent 1's last third. Just like in Cut and Splice, Child 2 is the negative of Child 1, where Child 2 receives Parent 2's first third, Parent 1's middle third, and Parent 2's last third.

Using the crossover method an entirely new population of animals is created, and then entire original population is killed off, even the ones that were selected (they have reproduced, so they are no longer needed). Finally, the last step of a genetic algorithm executes: evolution. In GA terms it is referred to as mutation, and, just like all the other components to a genetic algorithm, it mimics real world evolution very well. There is always a percent likelihood that mutation will occur. The unofficial standard is 0.01%, but it really depends on how the other parts of the GA are structured. It is also not uncommon for it to be 1%, although any higher is generally not ideal because real world mutation is caused by mistakes when genes are being copied from a parent to a child, a slow process that is not very likely to happen. In 0.01% of the time when any given chromosome does mutate, a random allele is chosen, and then changed to a different completly random number. This simulates the mistakes that would have been made during real life crossover (reproduction). ("Crossover (genetic Algorithm)," 2015) (Holland, 2012) ("Genetic Algorithms in Plain English," n.d.)

The new population has been created, maybe slightly mutated, and now it is time to repeat the entire process. The new population is put through the fitness function, then the selection function, the survivors of selection then reproduce using the crossover function, creating an even newer population, which also has a chance of being mutated. Each new population may or may not have a higher average fitness than the last, but if the algorithm is made well then there should be overall growth. Through repeating this process the genetic algorithm eventually reaches the optimum fitness, which is ultimate solution to an optimization problem.

Similar algorithms have been created in the past, like the one made by David Eck in 2011. It contains animals called eaters that move around depending on the alleles in their chromosome and interact with their environment, eating food when they come across it. As time

goes on they get better at finding food. This program is very similar to one of the algorithms that will be created in this experiment, and will be a useful guide while constructing it. While there similar GAs out there, never have they been built in such a way that the population is what is ultimately what is being compared and tested. The results of this research would tell us more about how these specific changes in the ecosystem of an animal can affect its numbers. (Hosch, n.d.)

Materials

Hardware:

- Windows Laptop with 8GB RAM and Intel Core i5

Software:

- Chrome Browser
- Code editor (**Atom** was used)
- Genetic.js (An extension to javascript that allows for easier genetic programming)

Procedure

For this project, create two separate programs, GAone and GAtwo. GAone contains the foragers, and GAtwo the hunters. Both use the Genetic.js library created by Sean of the subprotocol.com website ("Genetic.js," n.d.). Genetic.js works using a series of functions that are all used in the evolution process. The purpose and details of the main functions are described in the Review of Literature.

Function	GAone	GAtwo
seed() Creates the initial population	Returns an object with three properties: **speed**, **mouthSize**, and **fitness**. **speed** is set to a random number between 1 - 50, **mouthSize** to a number between 1-10, and **fitness** is set to 0.	Returns an object with 5 properties: **hunterSpeed** : 1 - 50 **hunterStrength**: 1 - 10 **hunterFitness** : 0 **preySpeed** : 1 - 50 **preyArmour**: 1 - 10

fitness(individual) Calculate the fitness of an animal	Returns the individual's fitness, that in this case is stored in the chromosome of the individual. **//return individual.fitness**	Returns the individual's fitness, that in this case is stored in the chromosome of the individual. **//return individual.hunterFitness**
mutate(individual) Mutates an animal	There is a 50% chance either the **speed** or the **mouthSize** will be mutated, then there is a 50% chance that whichever property was chosen will increase or decrease by a random amount.	There is a 50% chance either the **hunterSpeed** or the **hunterStrength** will be mutated, then there is a 50% chance that whichever property was chosen will increase or decrease by a random amount. There is a 50% chance either the **preySpeed** or the **preyArmour** will be mutated, then there is a 50% chance that whichever property was chosen will increase or decrease by a random amount.
crossover(mother, father) Creates two children from two parents	**Child1** has the speed of its **mother** and the **mouthSize** of its **father**. **Child2** has the speed of its **father** and the **mouthSize** of its **mother**. The fitness of both children are set to 0.	**Child1** has the **hunterSpeed** and **preySpeed** of its **mother**, and the **hunterStrength** and **preyArmour** of its **father**. **Child2** has the **hunterSpeed** and **preySpeed** of its **father**, and the **hunterStrength** and **preyArmour** of its **mother**. Both **hunterFitness** are set to 0.
optimize(fitness, fitness) Determines the *better* fitness score	Set to the pre made **Genetic.Optimize.Maximize**, which returns the larger fitness score	Set to the pre made **Genetic.Optimize.Maximize**, which returns the larger fitness score
generation(pop, gen, stats) Invoked at every generation	Loop through population and set every **fitness** to 0 to make sure every member of the next generation will have a fitness of 0.	Loop through population and set every **hunterFitness** to 0 to make sure every member of the next generation will have a fitness of 0.
notification(pop, gen, stats, isFinished) Used for displaying the results of your simulation	Add the generation number and the population number to a two column table	Add the generation number and the population number to a two column table

Download the Genetic.js library twice. Rename them geneticOne.js and geneticTwo.js.

In geneticOne.js

Find the part the code with the comment **//seed the population**. Delete the code inside the loop and replace it with following code. There is a (**animal.speed / 55**)% chance the animal will find food. If it does, then there is a (**animal.mouth / 11**)% chance the animal will have a mouth big enough to eat it. If both turn out, then increase the **animal.fitness** by 1, and add the encoded animal to the population list.

Above the line of code **this.entities = newPop**, add the following code. Create a loop identical the the one you just wrote, however remove the line of code that adds the animal to the list. Add else statement on both probabilities, so if the animal is too slow or its mouth too small, it is removed from the list (killed by starvation).

In geneticTwo.js

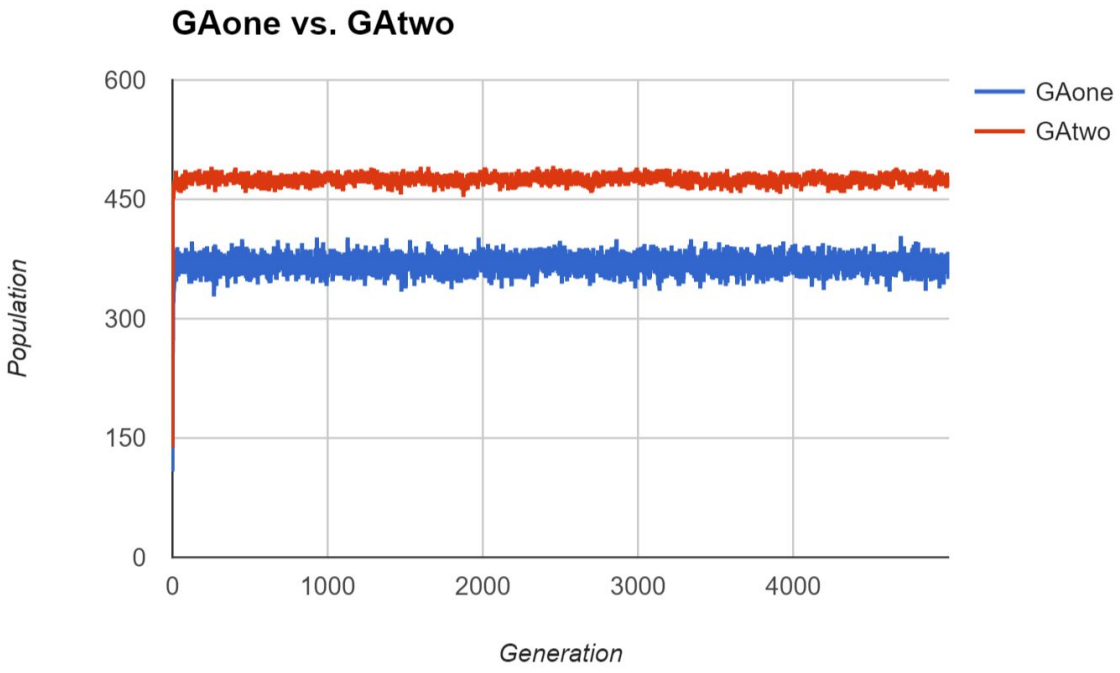
Find the part the code that **//seed the population**. Delete the code inside the loop and replace it with following code. If the **hunterSpeed** is greater than or equal to **preySpeed**, there is a 98% chance the hunter will catch its prey. If it does, and the **hunterStrength** is greater than or equal to the **preyArmour** (in other words strong enough to pierce the prey's armour) then increase the **hunterFitness** by 1 and add the encoded animal to the population list.

Above the line of code **this.entities = newPop**, add the following code. Create a loop identical the the one you just wrote, however remove the line of code that adds the animal to the list. Add

else statement on both probabilities, so if the hunter cannot catch its prey or cannot pierce its armour, it is removed from the list (killed by starvation).

Both GAone and GAtwo were configured in the config object to have 5000 iterations (generations), a size of 500 (initial population size), a crossover of 0.9, and a mutation of 0.3.

Results



GAtwo (the predators) had a consistently higher population than GAone (the foragers). While both populations started out very low in their first 30 or so generations, they eventually leveled out. After the first 30 generations, GAone had a max of 404 and a min of 328, while GAtwo had a max of 492 and a min of 453. GAone overall had a considerably larger range of fluctuation than GAtwo.

Conclusion

There may have been some experimental error in that the exact numbers would be different every time the program is run, and this is one one running. However the overall trends are unlikely to be very different. The hypothesis was incorrect. The population of animals that were carnivores forced to hunt for their food was higher than the number of animals that were herbivores that need only find food and be able to eat it. This does not reflect the behavior of real world animals in any way. However, it very easily could have. Scientists could study an animal for a year, be it a herbivore or carnivore, and roughly figure out how to algorithmically express that animals fitness, mutation rate, crossover rate, optimization method, and selection method. These estimates could then be put in this exact program or very similar to this one, and it would output reasonably accurate predictions of that animal's population size for the next 50 or even 100 years. So while the hypothesis was wrong, the purpose of this project was very much achieved.

Bibliography

Crossover (genetic algorithm). (2015, October 18). Retrieved October 29, 2015, from https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

Dolan, K. (2009). What is Genetic Programming? Retrieved from http://geneticprogramming.us/What_is_Genetic_Programming.html

Eck, D. J. (2011, August 22). Genetic Algorithms Demo Docs. Retrieved October 26, 2015, from http://math.hws.edu/eck/jsdemo/ga-info.html

Genetic Algorithm Hello World. (n.d.). Retrieved from http://subprotocol.com/system/genetic-hello-world.html

Genetic Algorithms. (2015, September 17). Retrieved October 29, 2015, from https://en.wikipedia.org/wiki/Genetic_algorithm

Genetic Algorithms in Plain English. (n.d.). Retrieved from http://www.ai-junkie.com/ga/intro/gat1.html

Holland, J. H. (2012). Genetic algorithms. Retrieved 2015, from http://www.scholarpedia.org/article/Genetic_algorithms

Hosch, W. L. (n.d.). Genetic algorithm. In *Encyclopedia Britannica*. Retrieved October 29, 2015, from http://school.eb.com.ezproxy.niles-hs.k12.il.us:2048/levels/high

Evolved antenna. (2015, July 3). Retrieved November 23, 2015, from https://en.wikipedia.org/wiki/Evolved_antenna

Genetic.js. (n.d.). Retrieved from http://subprotocol.com/system/genetic-js.html