Elijah Tai

Catherine Chamberlain

SLS 12 Understanding Darwinism

November 12, 2018

This is just the write up. The rest of the project (videos, etc.) is at: **https://drive.google.com/file/d/1gbUXxP4yakntKdkmBOg9J7AYC3ogCNF8/view?usp=sharing**

**Abstract**:

From a biological stand point, natural selection is the primary phenomenon that leads to adaptation of species. However, from a purely quantitative stand point, natural selection is the process of optimizing a cost function in which survival and reproduction is the end goal. Such a quantitative definition of evolution leads to the topic of evolutionary programming, which attempts to minimize a cost function. In their paper, Huang Han et al defines evolutionary programming (EP)s as, “a minimization problem, denoted by the 2-tuple , is to find an *n*-dimensional vector, such that ,” and they suggest that determining the correct mutation function is key to solving the problem efficiently.[[1]](#footnote-1)

In this project, I hope to simulate evolution and the general principles of evolutionary programming and demonstrate how natural selection can lead to intelligent behavior. To do so, I will define an organism as having two physical traits: an amount of food, and a location . Each organism also will have three decision matrices: *movLoc*, a matrix that takes into account the relative location of the organism and its nearest neighbor, *movFood*, a matrix that takes into account the food values of the organism and its nearest neighbor, and, *movCombined*, which weights the interaction between the two previous matrices. Furthermore, organisms will have some basic functions: *move(self)*, eat(Organism o), and reproduce(self). (Note that reproduction is asexual and does not require a partner organism in this simulation). When an organism disappears (it runs out of *food*, or gets eaten), a random organism will reproduce with a probability proportional to the sizes of the surviving organism. Therefore, organisms who survive with the most amount of food for the longest amount of time will generate more offspring. During the process of the simulation, I will quantify intelligent behavior as the proportion of correct movements to incorrect movements (the instances a larger organism chases a smaller one, or a smaller organism flees a larger one, divided by the instances of the opposite occurring). Intelligence score of generation g =. Thus, if natural selection occurs in the simulation, we should see this score rise.

**Code and Annotations**:

Organism Constructor

def \_\_init\_\_(self, food, x, y, movLoc, movFood, movCombined):

# Physical Attributes

self.x = x

self.y = y

self.food = food

# Genetic Attributes

self.movLoc = np.array(movLoc).reshape(2,1)

self.movFood = np.array(movFood).reshape(1,2)

self.movCombined = np.array(movCombined).reshape(1,2)

An organism has the following attributes: an amount of food, an x position and a y position. The organism can control its movement via three matrices: moveLoc, a matrix that handles the location of the nearest organism, movFood a matrix that handles the relative amounts of food of the neighbor and the organism itself, and movCombined, which weights the importance of the two matrices. The assumption is that this data will be enough to create intelligent movement.

Organism Actions

def live(self):

self.detect()

self.move()

if self.food < 20:

population.remove(self)

elif self.food > FOOD\_MAXIUM:

self.food = FOOD\_MAXIUM

self.eat()

self.render()

Then the organism can live. First, it detects the nearest neighbor. Then, it moves based on its matrices and the information detected. If it has less food then 20, then the organism starves and gets removed from the population. On the other end of the spectrum, there is a cap of the maximum amount of food of an organism which that organism cannot exceed. Then, based in its location, the organism will try to eat its neighbor. Finally, the location of the organism is graphed on a graphical user interface (using the python library Pygame) in order to visualize the movement of the organism.

Detection

def detect(self):

# Sets info of first organism in the list

nearestOrganism= [LARGENUMBER,LARGENUMBER,LARGENUMBER,LARGENUMBER]

for i in range (len(population)):

if population[i] != self:

distX = self.x - population[i].x

distY = self.y - population[i].y

# Update if it is the least displacement

if (distX\*\*2 + distY\*\*2 < nearestOrganism[0]\*\*2 + nearestOrganism[1]\*\*2):

nearestOrganism[0] = distX

nearestOrganism[1] = distY

nearestOrganism[2] = population[i].food

nearestOrganism[3] = i

self.distNX = nearestOrganism[0]

self.distNY = nearestOrganism[1]

self.Nfood = nearestOrganism[2]

self.Nindex = nearestOrganism[3]

Here, the program runs through each organism and finds the one with the least displacement from the current organism. Then, the program stores the information of the neighbor’s x position, y position, food, as well as that neighbor’s index within the population so that it can be easily identified later. This then gives the organism information as to how to move.

Movement

def move(self):

global CORRECT\_MOVEMENT

global INCORRECT\_MOVEMENT

The program references a global variable that keeps track of correct and incorrect movements. This does not influence the actual movement of the organism, but will be used to see if the organisms are indeed getting more intelligent and adapting over time. Below, the program multiplies the movement matrices, or the “genetic material,” to determine a movement result.

tempD = np.array([self.distNX/WIDTH, self.distNY/HEIGHT]).reshape(1,2)

tempF = np.array([self.food/(self.food+self.Nfood), self.Nfood/(self.food+self.Nfood)]).reshape(2,1)

movResult = np.matmul(self.movCombined, np.matmul(np.matmul(tempF, self.movFood), np.matmul(self.movLoc, tempD)))[0]

Since movement is a metabolic cost, the program takes this into account. The organism must consumer food proportional to the distance it wishes to travel.

movMag = abs(movResult[0]) + abs(movResult[1])

self.food -= MOVEMENTCOST \* movMag

Then, the program determines whether the organism chose a “good” or “correct” direction. This is defined as a large organism chasing a small organism, or a small organism fleeing a large organism.

if (MOVEMENTSPEED\*movMag\*movResult[0] > 0):

if (self.distNX > 0 and self.Nfood < self.food):

CORRECT\_MOVEMENT += 1

else:

INCORRECT\_MOVEMENT += 1

if (MOVEMENTSPEED\*movMag\*movResult[0] < 0):

if (self.distNX < 0 and self.Nfood < self.food):

CORRECT\_MOVEMENT += 1

else:

INCORRECT\_MOVEMENT += 1

if (MOVEMENTSPEED\*movMag\*movResult[1] > 0):

if (self.distNX > 0 and self.Nfood < self.food):

CORRECT\_MOVEMENT += 1

else:

INCORRECT\_MOVEMENT += 1

if (MOVEMENTSPEED\*movMag\*movResult[1] < 0):

if (self.distNX < 0 and self.Nfood < self.food):

CORRECT\_MOVEMENT += 1

else:

INCORRECT\_MOVEMENT += 1

Now, the organism can update its position accordingly.

self.x += MOVEMENTSPEED \* movMag \* movResult[0]

self.y += MOVEMENTSPEED \* movMag \* movResult[1]

Regardless of whether or not the organism moves, it still has a rest cost merely by existing and being alive.

self.food -= EXISTENCECOST

Eating

In order for the organism to eat, it must be within range, which is defined by the position and the amount of food (which correlates to size) of the organism. When eating, the probability of successful predation varies by the square of the difference in size. This way, the larger organism with more food will likely eat the smaller organism with less food, but there is a small probability that the opposite will happen. If an organism gets eaten, then the predator will gain the prey’s food amount, and the prey will be removed from the population.

if (self.distNX\*\*2+self.distNY\*\*2 < ((math.log10(self.food) + math.log10(self.Nfood))\*5)\*\*2):

# Successful predation (just a proprotion based on food levels) Literally eat or be eaten

if random.uniform(0,1) < self.food\*\*2 / (self.food\*\*2 + self.Nfood\*\*2):

self.food += self.Nfood

del population[self.Nindex]

else:

population[self.Nindex].food += self.food

population.remove(self)

del self

Reproduction

An organism will reproduce at the cost of half of its total food amount. First, it sets its child’s movement matrices, (or genetic material) to the same as itself. Then, it “mutates” one random value by a small amount. This allows the species to change slowly over time. This reproductive process is asexual, so the organism does not require a mate to reproduce. This is because the species is by construction cannibalistic because it eats its own species. Therefore, coding a decision tree of when to eat a neighbor and when to mate with a neighbor is difficult.

def reproduce(self):

self.food = self.food/2

childMovLoc = [self.movLoc[0],self.movLoc[1]]

childMovFood = [self.movFood[0][0], self.movFood[0][1]]

childMovCombined =[self.movCombined[0][0],self.movCombined[0][1]]

# Mutations

mutation = np.random.uniform(0, 2)

if mutation == 0:

childMovLoc[np.random.uniform(0, 1)] += random.uniform(0, MUTATIONMAXIMUM)

if mutation == 1:

childMovFood[np.random.uniform(0,1)] += random.uniform(0, MUTATIONMAXIMUM)

if mutation == 2:

childMovCombined[np.random.uniform(0, 1)] += random.uniform(0, MUTATIONMAXIMUM)

# Create the child and add it to the population based on the mutated genetic material.

population.append(Organism(np.random.uniform(500,1500), np.random.uniform(0,WIDTH),np.random.uniform(0,HEIGHT), childMovLoc, childMovFood, childMovCombined))

Starting the Program

Now that the program has fully defined an organism, it can attempt to simulate how that organism would behave and interact with other organisms. To begin with, it creates a number of organisms (I set this value POPULATION\_SIZE to 30, though any number can work) with random values and locations.

for c in range (0,POPULATION\_SIZE):

population.append(makeOrganism(

np.random.uniform(1000,1500),

np.random.uniform(0,WIDTH),

np.random.uniform(0,HEIGHT),

[np.random.uniform(-1,1),np.random.uniform(-1,1)],

[np.random.uniform(-1,1),np.random.uniform(-1,1)],

[np.random.uniform(-1,1),np.random.uniform(-1,1)]))

Then, it lets each organism live. This constitutes detecting, moving, and eating.

for org in population:

org.live()

Now, every 500 frames, the program gives us the average food of all the organisms, and the “intelligence score” of the generation.

displayCount += 1

if (displayCount >= displayRatioEvery):

displayCount = 0

totalFood = 0

for org in range(len(population)):

totalFood += population[org].food

print ("AVERAGE FOOD: "+str(totalFood/len(population)))

# Print the intelligence score of the generation.

print("EXPECTED MOTION: "+str(CORRECT\_MOVEMENT/(INCORRECT\_MOVEMENT+CORRECT\_MOVEMENT)))

# Reset the values

CORRECT\_MOVEMENT = 0

INCORRECT\_MOVEMENT = 0

And finally, when an organism dies (whether by starvation or predation), one surviving organism gets to reproduce, leaving the total population size constant. The probability that an organism gets to reproduce varies with the square of the amount of food of that organism. This is in accordance with the idea of “survival of the fittest,” where fitness is coded by the amount of food an organism was about to eat.

while (len(population) < POPULATION\_SIZE):

# Calculate total food.

totalFood = 0

for org in range(len(population)):

totalFood += population[org].food\*\*2

# Reproduction is random.

reproductionChance = round(np.random.uniform(0,totalFood),3)

# Determine which organism reproduces.

counter = 0

for org in range(len(population)):

counter += population[org].food\*\*2

if (counter >= reproductionChance):

population[org].reproduce()

break

And finally, when an organism dies (whether by starvation or predation), one surviving organism gets to reproduce, leaving the total population size constant. The probability that an organism gets to reproduce varies with the square of the amount of food of that organism. This is in accordance with the idea of “survival of the fittest,” where fitness is coded by the amount of food an organism was about to eat.

**Results:**

Interestingly, there are two common outcomes, each of which occurs about half the time. The first is that the organisms will try to eat each other all the time. See the attached video for the simulation example and the data collected. For this scenario:

Meanwhile, for the second scenario, the organisms seem to constantly flee each other. The organisms evolve to do this about half the time. See the attached video to see the simulation and the data gathered.

One possible explanation for these two differing types of results could be that the organisms are finding different local minima to the same problem of survival. While in the first scenario, the organisms seem to decide to do the incorrect move more and more often, the trend is clear with an r2 of 0.94. Therefore, the organisms are clearly evolving, though not how I expected. In the second scenario, the evolution seems to be logarithmic—the organisms obtain intelligent behavior very quickly early on, but this rate of learning decreases over time. In both scenarios, the organisms gain more food and become more successful, as shown in the graphs above. If this is the measure of overall fitness of the species in general, then the organisms are becoming more and more fit.

**Connections to Class:**

The model was built using a lot of the concepts introduced during class. The basic concepts come from Darwin: there are limited resources (food), so only the fittest can survive for the longest period of time. He writes:

Owing to [the] struggle for life, any variation, however slight and from whatever cause proceeding, if it be in any degree profitable to an individual, in its infinitely complex relations to other organic beings and to external nature, will tend to the preservation of that individual, and will generally be inherited by its offspring.[[2]](#footnote-2)

In this way, my program seems to model natural selection and intraspecific competition; organisms in the model mutate, the fittest tend to survive, and the fittest generally have more offspring which will inherit positive adaptations. Daniel Dennett writes, “evolution will occur whenever and wherever three conditions are met: replication, variation (mutation), and differential fitness (competition).”[[3]](#footnote-3) In this way, the simulation fulfills the basic criteria for evolution to occur.

While the program takes into account the general principle of natural selections, there are many complexities that are not fully considered. For example, to take into account the metabolic cost of moving, I subtracted a constant from the amount of food an organism had stored (proportional to the distance moved). However, in real life this is not exactly the case. For example, a study on rhesus monkey found that the metabolic cost of movement is not a constant, and changes with caloric restriction over time.[[4]](#footnote-4) Similarly, a group of biologists found that reducing the center of mass of humans lowers the metabolic cost of human walking,[[5]](#footnote-5) meaning that this metabolic cost should also be subject to natural selection and alteration, while in the simulation it was constant for all organisms. This is one of many instances of over-simplification, leading to unrealistic results.

The results of the simulation are interesting because when repeatedly running the simulation, two distinct and opposite scenarios repeatedly arise. This reminds me of the paper by John Maynard Smith which uses a word game to introduce the topic of intermediates and amino-acid alterations. Continuing off Smith’s analogy, perhaps what is happening in the simulation is that there are two possible chains that occur. For example, WORD -> WORE -> GORE -> GONE -> GENE could be a sequence of positive alternations that lead scenario 1.[[6]](#footnote-6) But, WORD -> WORK -> PORK -> PORE could also be a sequence of positive alternations, but it would lead to a different scenario. Another explanation is that the initialization of organisms is random, so the organisms do not all start with the same genetic material each time one runs the simulation. This is analogous to starting at slightly different words, which would likely lead to different ending words. Therefore, it is not surprising that the simulation, when run repeatedly, may perform slightly differently each time.

Overall, the simulation is an interesting way to visualize some of the concepts learned in class, tying in together natural selection, intraspecific competition, metabolic costs, and other ideas that are central to Darwinian evolution.

.

1. Han Huang et al., “Evolutionary Programming with a Simulated-Conformist Mutation Strategy,” *Soft Computing* 22, no. 2 (January 1, 2018): 659–76, https://doi.org/10.1007/s00500-016-2365-x. [↑](#footnote-ref-1)
2. Darwin and Charles, *Origin of Species* (Start Publishing Llc, 2012), 26. [↑](#footnote-ref-2)
3. Mark D. Pagel, *Encyclopedia of Evolution* (Oxford ; New York: Oxford University Press, 2002). [↑](#footnote-ref-3)
4. Yosuke Yamada et al., “Long-Term Calorie Restriction Decreases Metabolic Cost of Movement and Prevents Decrease of Physical Activity during Aging in Rhesus Monkeys,” *Experimental Gerontology* 48, no. 11 (2013): 1226–35, https://doi.org/10.1016/j.exger.2013.08.002. [↑](#footnote-ref-4)
5. Justus Ortega and Claire Farley, “Minimizing Center of Mass Vertical Movement Increases Metabolic Cost in Walking,” *Journal of Applied Physiology* 99, no. 6 (2005): 2099–2107, https://doi.org/10.1152/japplphysiol.00103.2005. [↑](#footnote-ref-5)
6. John Maynard Smith, “Natural Selection and the Concept of a Protein Space,” *Nature* 225, no. 5232 (1970): 563–64, https://doi.org/10.1038/225563a0. [↑](#footnote-ref-6)