# A Brief Introduction to The Topic Of Genetic Algorithms

Avraham Leff

January 7, 2023

## 1 Motivation

See the "general requirements document" for the purpose and scope of *"self-educate"* assignments.

I urge you to begin "self-educating" **now** before I release the actual assignment (in approximately a couple of weeks).

## 2 Introduction

Quoting from the good piece on Wikipedia (or see this rendering):

> In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. Some examples of GA applications include optimizing decision trees for better performance, automatically solve sudoku puzzles, hyperparameter optimization, etc.

From your perspective of CS courses, genetic algorithms are a radical departure from the algorithms in your toolbox. I certainly agree that genetic algorithms are not used for everyday problems. They may be useful when the types of algorithms with which you're familiar don't seem to work (at least in a reasonable amount of time).

A GA is a *search heuristic* inspired by Charles Darwin's theory of *natural evolution*[1]. That theory uses the concept of *natural selection*: the fittest individuals in a population will tend to reproduce (producing the next generation's population) more than less fit individuals. GAs are a "simulation" of natural selection to solve computational problems.

A GA includes a population of *chromosomes*. Chromosomes are composed of *genes* that specify certain traits, which are used to solve some problem. A *fitness function* defines "how well" a chromosome solves a given problem. A GA executes over the course of a number of generations. In each generation, chromosomes that are more fit are more likely to be selected to reproduce. There is also a probability in each generation that two chromosomes will have their genes merged. This is known as *crossover*. In addition, in each generation, genes in a chromosome may *mutate* (i.e., may *change randomly*).

GAs terminate when:

- Either: the fitness function of an individual in the population crosses some specified *threshold* (so we can return that individual),
- Or: the algorithm has executed some specified maximum number of generations. In that case, the GA return the "best" individual (the one with the highest "fitness value")

One reason why GAs aren't used more broadly is that they depend on three (partially or fully) randomly determined operations

- *Selection*,
- *Crossover*,
- *Mutation*.

A GA's behavior can therefore differ greatly from one execution to another, and may simply be unable to find a good solution in a reasonable amount of time. Their role (if any) is for problems where we have no good deterministic algorithms: GAs allow us to quasi-randomly explore the solution space for that problem.

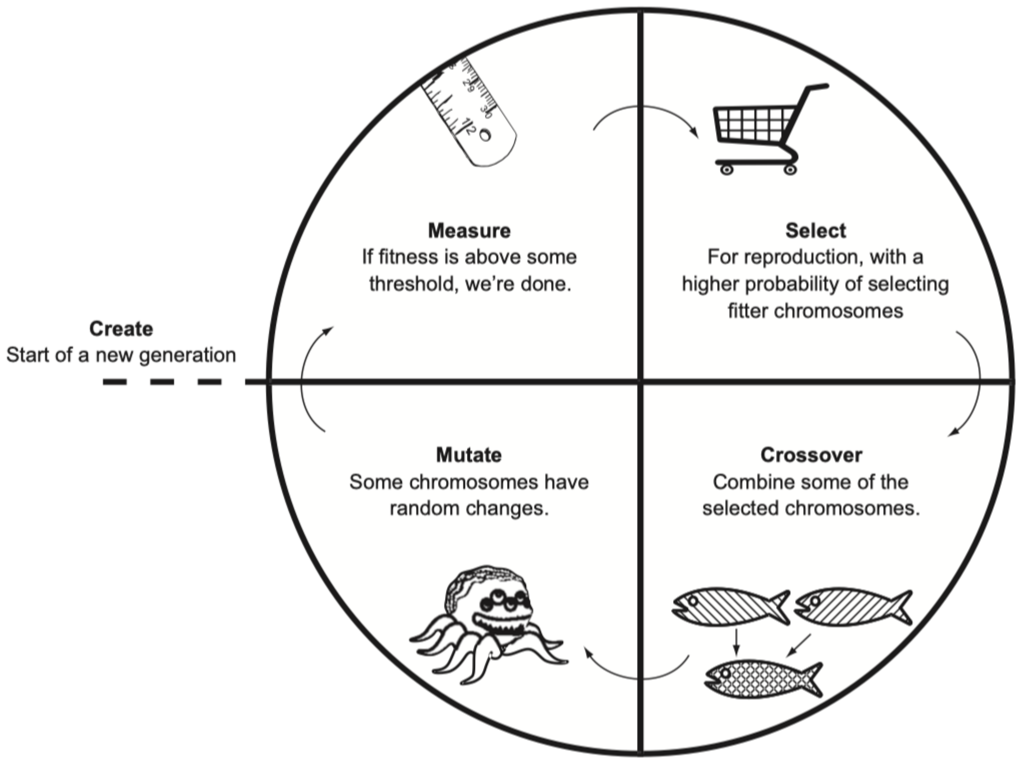
[1]This is not the place to discuss questions such as whether that theory is buttressed by experimental evidence, or not. Here, we're using that theory as an analogy that motivates our search heuristic.

## 3 Steps Of a Generic GA

The following may help you structure the implementation of a “generic” GA.

1. Create an initial population of random chromosomes for the first generation of the algorithm.
2. Measure the *fitness* of each chromosome in this generation of the population. If any exceeds the threshold, return it: you’re done!
3. Randomly select some individuals to reproduce, biasing the selection towards individuals with higher fitness values.
4. Randomly decide to *crossover* (combine traits of two individuals) to create children that represent the population of the next generation.
5. Randomly decide to *mutate* some chromosomes.
6. The population of the new generation is now complete: the GA replaces the previous generation’s population with the new generation’s population.
7. If the “maximum number of generations” threshold is reached, return with the best solution found so far.
8. Return to *step 2*.

The picture below depicts a GA’s life-cycle.


Measure
If fitness is above some threshold, we're done.
Select
For reproduction, with a higher probability of selecting fitter chromosomes
Create
Start of a new generation
Mutate
Some chromosomes have random changes.
Crossover
Combine some of the selected chromosomes.

Many "selection" techniques can be used: two common ones are

- "Roulette-wheel" selection in which every chromosome's chance of being selected is proportional to its fitness
- "Tournament" selection in which a number of randomly chosen chromosomes challenge one another; the one with the best fitness is selected.

## 4 Implementing a GA

The fun part of implementing a GA is figuring out how to translate the selection, crossover, and mutation metaphors into your problem domain. Then you decide what threshold will suffice for your needs. Finally, throw in some randomness, start the GA, and see if a good solution "evolves" from your "primordial soup" (first generation's population)[2].

[2]Note that you can significantly influence the GA's behavior by seeding the initial population with a specific set of "good" or "bad" traits.

See the `GeneticAlgorithms` directory in our classes's git repository for (excerpted) presentations that I found on the Internet on the topic of GAs. Here is an Introduction to Genetic Algorithms — Including Example Code. (I'm OK with your looking at that code because the challenge is to apply these ideas to a specific problem. IMHO, phrases such "In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s)." are not that helpful. YMMV of course.)