Genetic Programming vs Genetic Algorithms
ECE 657 A3 Q3

Group 49

Although both methodologies fall under the topic of “evolutionary” computation, genetic algorithms and genetic programming differ in various ways outside of just their naming. Both techniques use evolutionary operations on a population of possible solutions, that perform a biologically inspired search on the solution space to find an optimal answer to the problems they’re targeted to solve. Both techniques perform mutate, crossover and selection type operations during a search of the solution space but their granularity and types of structures they operate on is a key difference. Mutate randomly changes parts of the sequence of a solution introducing randomness into the overall search, while the crossover takes parts from two existing entries and recombines them to form a new potentially “better” outcome. Selection on the other hand evaluates and selects the best solutions in the newly generated search space before either terminating the search or moving to another evolution cycle. Both algorithms use a mechanism to measure “fitness” of their output solutions but work on different types of data for the given problem but genetic algorithms are more prone to producing invalid results due to the nature of the encoding space used for the data.

Genetic programming refers to the process of ordering and encoding a set of mathematical operations in a tree or graph data structure to produce the optimal functional result for a problem. The data structure used is the encoding used, and the moves within said structure become the algorithm’s population with the order of the nodes in the structure begin its state. With genetic programs, the correct output is a program with the optimal set of “moves” that generate a state path of operations that load to the optimal result measured by the fitness function. By crossing over, mutating and selecting is done on the branches between nodes to change the ordering as well as position of the nodes in the overall structure. The genetic program, searches across the possible set “moves” of the nodes and links, to obtain its result, which as the end result is a program. During the mutation process the branches within the structure of operations and their ordering is shifted around randomly, but the resulting operations do not yield invalid results. By storing the information in the specific data structure and tying each biological operation to modifying the structure, each item in the population whether mutates or crossed over, is potentially a valid state that can be evaluated.

A genetic algorithm similarly is used to solve an optimization problem. Its different from genetic programming as genetic algorithms first generate a population of potential solutions using numerically encoded strings called, chromosomes. Each chromosome is generated using an encoding o either binary, or any other chosen alphabet, string to represent a feature, known as a gene, with possible states called “alleles” and a given position or “locus” within the string itself. Borrowing from the biological sense, this encoding is then applied to every representation of initial solutions before the algorithm runs. The fitness function here is used to evaluate the distance to a possible optimal solution for a given set of constraints and outputs a resulting string data of the encoded states via chromosomes. The genetic algorithm performs the previously mentioned operations (mutations, crossover, selection) over various iterations until a “good enough” solution is found evaluating each new population using the fitness function. On a more nuanced take from this high level view the operations, mutation as well as crossover impose the greatest difference between both techniques as mutation, and even crossover can yield wildly different results, and sometimes invalid outcomes to the problem constraints, as the output is bound by the encoding string size, and not the problem’s constraints.

An example demonstrating of this outcome is requiring a solution to never contain values higher than a random value, say 257 on a 10 bit binary encoded chromosomes. Although a population can be generated to yield a result within the bounds of the problem, by only generating values under the 257 encoded value limit in this case, crossover and mutation operations of bit strings can potentially take binary values produce invalid results. Take for instance the following encoding (encoded in C in this case) :

0b0011111110 (decimal 254) and 0b010000000 (decimal 256)

If an crossover at the second from the left (second digit from the b) is performed, we yield the following result:

0b0111111110 (decimal 510) and 0b0000000000 (decimal zero)

Which produced one value outside the valid range of operations with a deterministic crossover operation. For a mutation should a random locus be selected there exists a non zero probability that the final encoded chromosome that is generated is an invalid result. Additionally binary mutations, such as random bit flips, in the top two bits (the most left) would also result in values also producing invalid constants for non zero, in positions 3 to 10. Unlike genetic programming, our goal, as well as constraints on the search space, need to be taken accounted for in the encoding used and can potentially yield invalid results.