Elad Ohayon

2/22/23

Arithmetic Puzzle Writeup

My algorithm uses a List<List<Character>>, to evolve a solution, meaning it uses a list of type character lists which represent digits. Using this data structure, I created randomness in my population by using a Random instance to generate a random index for each character and then placing it in its respective position. I specified my threshold as the double 1.0 which meant there was no difference between the sum and augend + addend; in other words, the chromosome is a solution. My fitness criterion was a ratio of how far the chromosome was from the solution measured by taking the distance of augend+ addend and the sum. My implementation performed its mutation on the winner of the tournament or roulette, if a random number selected between [0,1] was less than the mutation probability. I then randomly selected a gene in this "winner" and changed its value to another. Likewise, when implementing crossover, I first choose 2 winners from the tournament or roulette, and checked if a crossover will be performed by picking a random number between [0,1]. If a crossover was to be performed, I choose a random number between [0, geneCount] and divided the characters like so; I made the child inherit the characters in the first part from chromosomeA and the rest from chromosomeB.

To determine the selection type and mutation/crossover choices that produced the best results, I conduced an experiment where I held all constant except for what I was measuring, in this case the problem size. I recorded the cumulative number of generations and solutions when repeating the experiment 50 times to guarantee accuracy. I noticed very interesting results when looking at the best SelectionType. Both Roulette and Tournament arrived at the solution nearly identical number of times but the number of generations they took varied. Roulette consistently outperformed tournament when the disparity between the length of one term increased with respect to the others. For example, sum.length=3 and addend,augend.lengths =1. On the other hand, tournament outperformed roulette when the lengths of the input were more similar, for example (augend, addend, sum).lengths=2.

When checking for the best mutation and crossover probability, I found it was best to have both mutation and crossover probability as high as possible (between .7-1). This is because my algorithm either mutates or crosses over but does not do both to the same chromosome as this could "corrupt" a chromosome and drastically change its fitness for the worst. The rationale for this is that since the fitness of chosen chromosomes is high, it would be detrimental to overmodify them. A genetic algorithm is a good algorithm to solve this problem as there is a finite set of possible permutations and, although very large, by generating a random population and selecting the closest members and performing a mutation/crossover, we converge at a solution.