Morgan Rogers

10-21-21

Program 2 – Genetic Algorithms Report

CS 461 Artificial Intelligence

This program uses a genetic algorithm to find a good selection of items to pack while staying within the weight guidelines, which is less than 500 pounds. The 'Program2Input.txt' gives 400 items with assigned weight and utility. My program assigns the number of items and bits in the genome to 20 and then picks 20 items out of the 400 given based on their utility. There are two classes, one called 'Thing' and another called 'Genome.' The 'Thing' class represents an item and has a weight and utility attribute. The 'Genome' class represents the genome or the string of bits in a population (or generation). 'Genome' has a binary string attribute which is a list of 20 (number of things picked) 1's and 0's and a fitness value attribute. Fitness is a value between 1 and 200 and this is calculated by adding up the utility values of the 'things' that correspond to the 1 values in the genome's binary string. The weights of the 'things' are also added up and if the total weight exceeds 500, a fitness value of 1 is assigned to that genome.

For the genetic algorithm part of the program, an initial population of genomes is created randomly based on the population size in the 'population_size_list'. Five iterations of the genetic algorithm are run with population sizes of 500, 1000, 1500, 2000, and 2500. The number of generations for each population size is 1000 and if the average fitness improves less than 1% across 50 generations, the iteration stops and a brand-new population is created with a size of the next population size in the 'population_size_list'. Each iteration is timed using the time python library. The average fitness of each generation is recorded and outputted to the 'average_fitness.txt' file. The genome with the highest fitness in each iteration of 1000 generations is calculated and its information is outputted to the 'max_fitness_genome.txt' file. Each generation of genomes is sorted by their fitness scores and the top two genomes with the highest fitness scores are added to the next generation. Two parent genomes of the current generation are selected using the 'selection_pair' function which selects parents based on the L2 probability distribution created using the 'L2' function I created using numpy. The parents are then split and recombined using the 'single_point_crossover' function which splits the two genomes at a random point and recombines them to create two new offspring which are then added to the next generation. Parents are chosen and offspring created ((length of the population) / 2) times.

The data structures used in this program are lists. Lists in python are easy to work with and I found them useful for the genome binary string. The random library was used for generating a random list of 'things' and creating the genome binary string with random 1's and 0's. I was not able to parallelize the program because I ran out of time, but I think parallelizing the program would have made it run a lot faster. It took around 10 minutes for the program to run (output shown in the pictures below) for all five populations. The max fitness out of all the iterations was 146.9 with the goal fitness being 200 (10 (max utility) * 20 items). For the population size of 500 and 1000, the iteration stopped short because there was not much

improvement of the average fitness between 50 generations. All 1000 generations were completed with the rest of the population sizes (1500, 2000, and 2500). The genome with the max fitness over all is shown in the second photo and it shows that the total weight for the genome is about 251.2 pounds which leaves 248.8 pounds free of space.

```
Starting population: 500
Max fitness after 700 generations: 129.6
Average fitness: 72.0
time: 12.23s

Starting population: 1000
Max fitness after 200 generations: 146.9
Average fitness: 95.97
time: 10.92s

Starting population: 1500
Max fitness after 1000 generations: 138.7
Average fitness: 94.8
time: 110.38s

Starting population: 2000
Max fitness after 1000 generations: 120.0
Average fitness: 68.8
time: 188.02s

Starting population: 2500
Max fitness after 1000 generations: 118.2
Average fitness: 44.9
time: 287.49s
```

```
max fitness genome:
    genome: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1] - fitness: 146.89999999999998
    total weight: 251.19999999999996
    total utility: 146.89999999999998

Items taken:
    Utility: 7.6 Weight: 20.2
    Utility: 6.3 Weight: 7.2
```