Genetic algorithm

Graded assignment 01 - report by Frédéric Charon

Task 1)

We want to solve the given optimization problem using a genetic algorithm. For the task we have a formula for the thrust and different variables, each of them having a specific value range. The problem is solved by using these variables as features for the chromosome class and the formula to evaluate the outcome. We know that the optimal outcome is a thrust of 870N, so a chromosome is very fit if the evaluation is close to that value.

Task 2)

1. We implement the features as binary bitstrings. Feature 1, 2 and 4 are just transformed to bits, for the mapping we use the map_bin_to_int() method. To make it easier handling the bitstrings for feature 3 and 5 we first multiply them by 10 and 100, so the map function includes the division.
2. Crossover – the crossover uses a crossover function and applies it to each feature of two different chromosomes.
   Arithmetic crossover – for each feature the values are converted to integers, added and divided by two, then they're converted back to bit strings.
   Single-point – if the bit strings don't have the same length, we add 0s at the beginning of the smaller bit string. Then we choose a random crossover point for each feature. The part before that point is inherited from parent1, the other part from parent2.
   Multi-point – for each bit in every feature it is randomized from which parent the bit is coming from.
3. Mutation – the mutation rate determines the probability of the features to mutate. For the high-level mutation, we add an offset to every feature, while the low-level mutation flips one bit of every feature. After a feature has changed, we check that the values are still valid.
4. We implement the evaluation function to see how fit the chromosome is, here we use the formula given for thrust. The closer the result is to the optimal thrust, the better the fitness of our chromosome.
5. In the __str__() function we print the resulting thrust as well as the value for each feature.

Task 3)

Elitist selection – Here we sort our population list of chromosomes and choose the top 20 (the fittest) chromosomes to use the crossover function on and replace the 20 last elements (the least fit) with these new made chromosomes.

Roulette wheel selection – Here we generate a random number for each chromosome. If the number is higher than the fitness, the chromosome stays. If it is not, the chromosome is going to be replaced by a new one, for the new one we chose two random parents of our old population. Since the random number's maximum is set to 500, each chromosome having a worse fitness than 500 is guaranteed to be replaced.

Task 4)

Implementing the run_model() function, which calculates and sets the fitness of each chromosome, sorts the population list by fitness and then removing and generating chromosomes from that list depending on the chosen selection type.

Task 5)

It is interesting to see how the model behaves using different inputs. A problem is that it is possible for the algorithm to be stuck in a loop when the entire population's features are too similar, using a bigger population size minimizes the risk of this to happen. For the same reason it can be helpful to use a maximum number of generations instead of a desired average fitness, this way we prevent the algorithm of getting stuck, but we might not get values as precise as for the average fitness as termination criterium. A higher mutation rate also helps to get more genetic diversity and might help the model to not get stuck, however if it is set too high the model behaves more random and needs longer to find the optimal feature values. Using Roulette wheel over Elitist selection is another option to add more diversity, here even the fittest chromosomes can be replaced but the odds are lower as for less fit chromosomes.