CS4300Spring 2018 Hauschild

Project 3
Due Date: 4/5/2018

Submission: Please submit your completed source files and any documentation through MyGateway submission system. **Please do not email them to me.**

[100pt] In this project you will implement a simple genetic algorithm that only uses crossover and mutation which attempts to solve the onemax problem

Assuming a function is given the size of the population as N and the size of the string n:

Initialization:

Create a population N of randomly generated binary strings of the specified size n. These strings should be generated with the probability of each bit being 1 of 0.5.

Selection:
You will select sets of parents using binary tournament selection of k=2. So, given a population of size N, select a random individual. Then select another random individual. Select the better one (in fitness) as parent one. Then selection a random individual and another random individual. The more fit one because parent two. These are the first two parents for recombination.

Recombination:
For recombination (the method to produce children from parents) we will use uniform crossover. Your crossover operator that you should use is uniform crossover. Uniform crossover generates a child by randomly picking from which parent to get each bit. Uniform crossover should generate two children, that are opposite of each other. This crossover operator should only be applied with probability of $p_c = 0.6$. What this means is, 40% of the time, your uniform crossover operator should simply copy the parents as the children, without modification. 60% of the time, it should perform the crossover that generates new children by randomly taking bits from each.

You should also use a mutation operator that, after you generate a child using crossover or copy, you mutate an individual bit with probability of 1/n where n is the size of the string. If a bit is selected for mutation, simply flip its value to 0 if it is 1 or 1 if it is 0. Note that with a probability of 1/n, this means that we would expect on average to flip only one bit, but it could flip more than one.

Replacement:
Once the children are generated, we must determine how much of the parent population to replace. In this case, for a population of size N, we will generate N-2 children (so find (N-2)/2 pairs of parents that produce N-2 children). Then, replace all but the best 2 parents with this set of children. This is called elitism replacement, as we are keeping the very best individuals around each generation.

Fitness function:
Your fitness function is the onemax function, which is the sum of 1s in the binary string. The global optimum is simply the size of the string, with higher fitness being desired. So for example, the string {01101101} would have a fitness of 5 and the string {11111111} would be the global optimum. While this is a fairly simple function to optimize, it will let you easily test your algorithm as the improvement of the population should be easy to see.

Overall procedure:

Assuming we have a population of size N, initialize the population of N individuals according to the initialization step.

Then we enter our generational loop:

> Start by finding the average fitness, highest fitness and lowest fitness individuals found in the population, display in particular the highest and lowest fitness individuals, then output the average fitness, highest fitness and lowest fitness values.
>
> Then using your *selection* technique, select two parents. Produce two children and add them to the pool of new children. Repeat this process of producing children until you have N-2 children. Then, using our *replacement* scheme, create a new population for the next generation by adding the two most fit individuals in the old generation to the set of children.

You should repeat this process until the average fitness has not increased for 3 generations. In that case, terminate, showing the most fit individual, its fitness and the average fitness at that time, along with the number of generations it took.

NOTE: Due to the algorithm being simplified (Using a simplified replacement strategy and only using elitism with two individuals), it is possible that the results can be poor depending on the initial population and the settings. Given reasonable values, it should get very close to maximum fitness. However, do not assume it will find the global optimum every time.

Overall Task:

Using the above technique, your program should prompt the user for the size of the string (n). Your algorithm should then, starting with a population of N=10, do 5 runs of the above procedure, counting up how many times it succeeded in finding the global optimum before terminating. If it fails to find it 5 of 5 times, it should repeat it with N=20. If that fails, double the population size again and repeat. Continue to do this until it succeeds (or reaches N=81920, at which point you should just output that the algorithm failed). When it succeeds, you should now have an upper bound where it succeeded and a lower bound where it failed. Calculate the midpoints between these two populations and try to use this to solve the problem. If it fails, that is your new lower bound. If it succeeds, that population is your new upper bound. Continue this procedure until you have found the smallest population size that will succeed 5/5 times. This procedure of finding the minimum population size to solve a problem is called *bisection*.

Along with your submission, I want a README file. In this readme I want you to tell me the population you got for a string size of 20 and a string size of 50.

Note: Make sure that your GA works before wrapping it up in bisection or trying to 5 times. Manually inspect the results with very small N and n to verify. Make sure your algorithm does *selection* correctly, as that is often what people stumble over.