YunQi Bang

COMS516X Homework 2

How to run

To run the program with command prompt, simply navigate to its directory and run the following command.

```
python ga.py
```

To change the parameters, open the file with any text editor and navigate to the main function (line 170 – 180) to modify the values accordingly.

Part I: Initial Parameters

A program is written to demonstrate how genetic algorithm works in finding an optimal solution with a random initial population.

For this assignment, the string "IowaState_Cyclones" is used as the optimal solution. Each of the chromosomes in this algorithm will have 18 genes (which is the number of characters in the optimal string). An initial population with 32 chromosomes will be generate randomly, and every generation after the first generation will have its population size at 16. At each iteration of the genetic algorithm, the top 8 chromosomes (the rest are killed off) will be selected to mate with each other, producing 8 new chromosomes. Only these top 8 chromosomes and the 8 new chromosomes will make up the population for the next generation. One-point crossover is used in the mating process, and the pairing of the parent chromosomes is done by alternating the top 8 chromosomes such that the evens and odds are matched, i.e. the best mates with the third best, the second best mates with the fourth, and so on. A mutation rate of 0.03 is used where the mutation operator is just randomly picking a valid allele for each gene.

Fitness Function

Two different fitness functions are tested, (i) distance of each gene from the optimal solution by considering their ASCII values, **(ii) binary decision** on if the gene matches the optimal solution. The search space of both functions are shown in Figure 1 and we can see that the fitness function that uses binary decision has a very flat surfaces, meaning a wide local optimal space, whereas the search space of the other fitness function is less linear. Table 1 shows the results from 10 runs of each approach and we can see that, surprisingly, **approach (ii) works better with 56% less generations needed** to reach the optimal solution compared to approach (i). We will look into possible explanations for this outcome in Part II of this report.

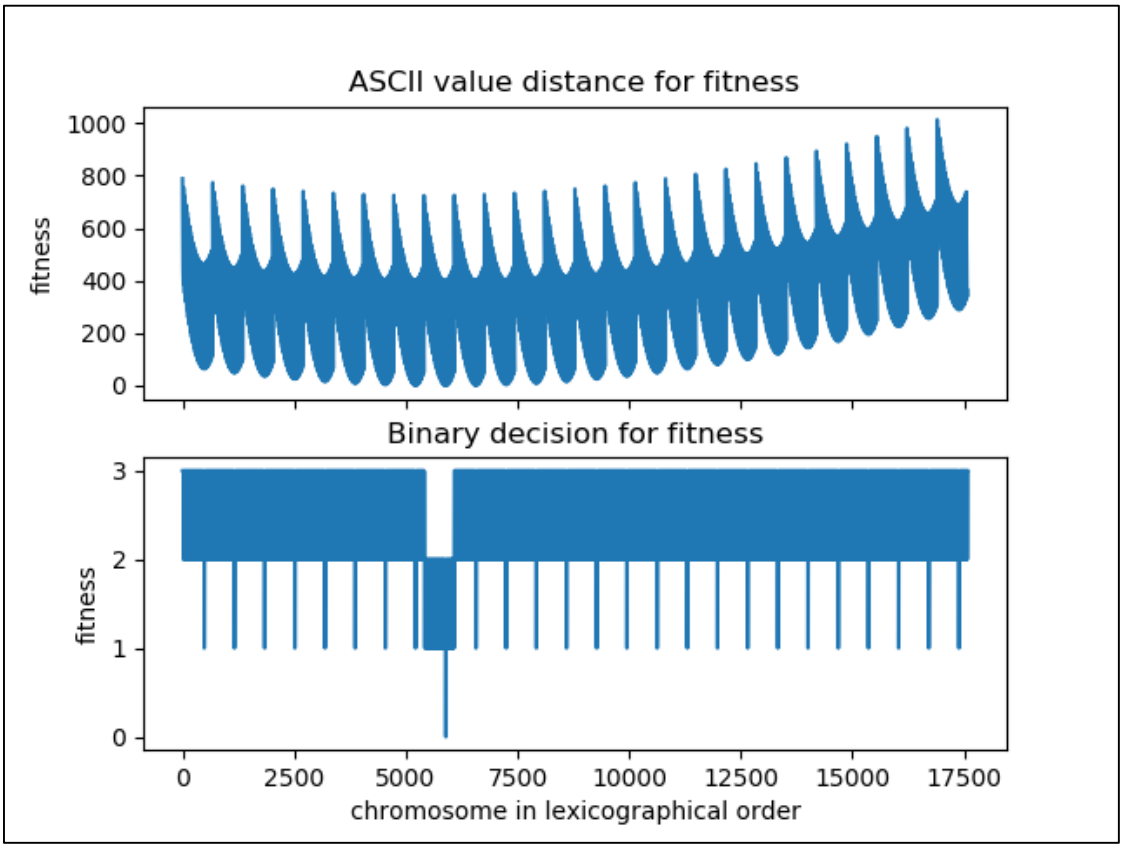

Figure 1 Example search space of different fitness functions. Optimal solution for this example is "ISU". Alleles are ASCII values from 'A' to 'Z' and chromosome length is 3. These parameters are used as the cardinality of the set of possible chromosomes with default parameters exceeds the capability for matplotlib to visualize.

Table 1 Results from 10 run of Genetic Algorithm with different fitness functions and default parameters

Fitness\No. of Generation	Min	Max	Std Dev	Mean
1. ASCII distance	1081	2050	314	1381
2. Binary decision	416	939	152	596

Mutation

We also observed the outcome with the absence of mutation for both fitness functions. Table 2 shows the number of generation needed for the algorithm to converge. Note that **none of the runs were able to reach the optimal solution**, and the population will **always converge to all the chromosomes being identical**, thus stuck in a local optimum point. An interesting finding is that while approach (i) always quickly converge into having identical population, approach (ii) often leads to converging into population of similar chromosomes with same fitness values for generations until it eventually (after a lot more generations) converges into all identical chromosomes. The * in Table 2 means that the mean is computed by also uses the number of generation needed to converge into identical fitness.

Table 2 Results from 10 runs of Genetic Algorithm with different fitness function and no mutation. Note that none of the runs found the optimal solution.

Fitness\No. of Generation	Min	Max	Std Dev	Mean
1. ASCII distance	5	10	2	7
2. Binary decision	4	15	3	7*

Random Generation

We experimented with random generation of chromosomes for each generation. The result is as expected, that all the runs **could not find the optimal solution nor show signs of converging** even after 500000 generations. The result does not change even if we were to use a different mutation rate or population size (except that the probability of randomly generating the solution would be higher if we have a much bigger population size). This is because random generation does not have the goal to minimize the fitness function, as it does not even consider the fitness, thus the fitness values would not converge over generations. If we look at the search space of this problem, there are 58 possible alleles for the gene to take and there are 18 genes in a chromosome, resulting in 58^{18} possible solutions. The probability for random generation to find the optimal solution is 58^{-18} = 1.81e-32, which is near to impossible. Theoretically speaking, random generation will always find the optimal solution, given we have enough time, luck, and patience, since the probability of finding the solution is not 0.

Part II: Manipulating the Parameters

N-point Crossover

In previous part, 1-point crossover is chosen as the mating process. Here we have chosen to experiment with different crossover to test if more crossover points would cause the algorithm to converge to an optimal solution slower as it introduces more changes, or that it would help the algorithm to converge to an optimal solution faster as the offspring would have a higher chance to have more informational genes being passed on to them. Results can be seen in Table 3 but the performance with different number of crossover are with slight improvement but highly similar. There is **no significant improvement or degradation in the performance** with more crossover points. To further test the effect of crossover points, the mutation rate is set to 0.003, a smaller number than default, to decrease the effect of mutation. However, the performance with different number of crossover points is also highly similar, thus being omitted. One possible explanation for this is that the algorithm always causes the population to converge into identical chromosomes and **what leads the algorithm to reach the optimal solution is actually the mutation**. This means that the crossover only affects how fast the algorithm converges to a local optimum fitness instead of the optimal solution.

Table 3 Results from 10 runs of Genetic Algorithm with different crossover points in mating process. The numbers represent the number of generation needed to reach the optimal solution.

No. of Crossover Points	Fitness Function	Min	Max	Std Dev	Mean
1	ASCII distance	1081	2050	314	1381
	Binary decision	416	939	152	596
2	ASCII distance	965	2608	520	1514
	Binary decision	401	853	151	594
5	ASCII distance	898	1924	309	1320
	Binary decision	264	815	196	490
9	ASCII distance	784	2079	403	1238
	Binary decision	272	826	161	486
16	ASCII distance	702	1983	359	1160
	Binary decision	252	846	209	575

Population Size

We have also tested the effect of population size on the performance. Result is as showed in Table 4 and we can observe that the algorithm **benefits from a larger population size**, while a smaller population size really hurts the performance of the algorithm, especially the one with ASCII value distance as its fitness function where it was unable to find the optimal solution in 500000 generations. Bigger population size would lead to bigger variety of good candidates thus the mating process would have a higher probability of producing good offspring while smaller population size would quickly converge to identical chromosomes and eventually found the solution with mutation.

*Table 4 Results from 10 runs of Genetic Algorithm with different population size. The numbers represent the number of generation needed to reach the optimal solution. * indicates that the value does not include a failed run that couldn't find the solution in 500000 generations.*

Population Size	Fitness Function	Min	Max	Std Dev	Mean
Halved	ASCII distance	10251	470081*	142601	178973
	Binary decision	741	2683	554	1332
Default	ASCII distance	1081	2050	314	1381
	Binary decision	416	939	152	506
Doubled	ASCII distance	527	903	121	698
	Binary decision	197	409	68	264

Other experiments/findings

Mutation rate is important for the algorithm to converge. For this particular optimal solution, it is found that mutation rate of 0.003 works the best for the algorithm with ASCII distance as

its fitness function while the default mutation rate of 0.03 benefits the approach with binary decision. A lower mutation rate assures that that algorithm will always converge as it introduces less turbulence, thus the algorithm works better with a curved fitness graph as seen in Figure 1. For binary decision fitness function, a higher mutation rate (0.03) would help in finding the optimal solution because of the wide area of local optimum as seen in the same figure, since the algorithm will have no motivation to move towards the global optimum.

From Part I, we have seen that binary decision fitness function has better performance compared to ASCII distance**. To better visualize the algorithm, the optimal solution is shortened and changed to all 'A's**. All the generations are displayed as it goes. The binary decision fitness function only considers a chromosome as a better chromosome as it gets 1 more gene correct while ASCII distance fitness takes the square of the distance for each gene. As we can see in Figure 2, **both of the algorithms has one 'A' appeared in their first generation, binary decision quickly sees it and starts propagating the correct allele in the second generation, while ASCII distance considers 'A`X' as a worse chromosome than 'JCI', therefore discarding this correct gene and preventing it from being passed on to the next generation**. This shows how using the distance as fitness function would hurt the performance.

As the mutation section in Part I and the crossover experiment in Part II suggest, both of these algorithm converge to identical population quickly and eventually depend on mutation to reach the optimal solution. Therefore, having the correct gene being passed on in early stages can boost the performance drastically.

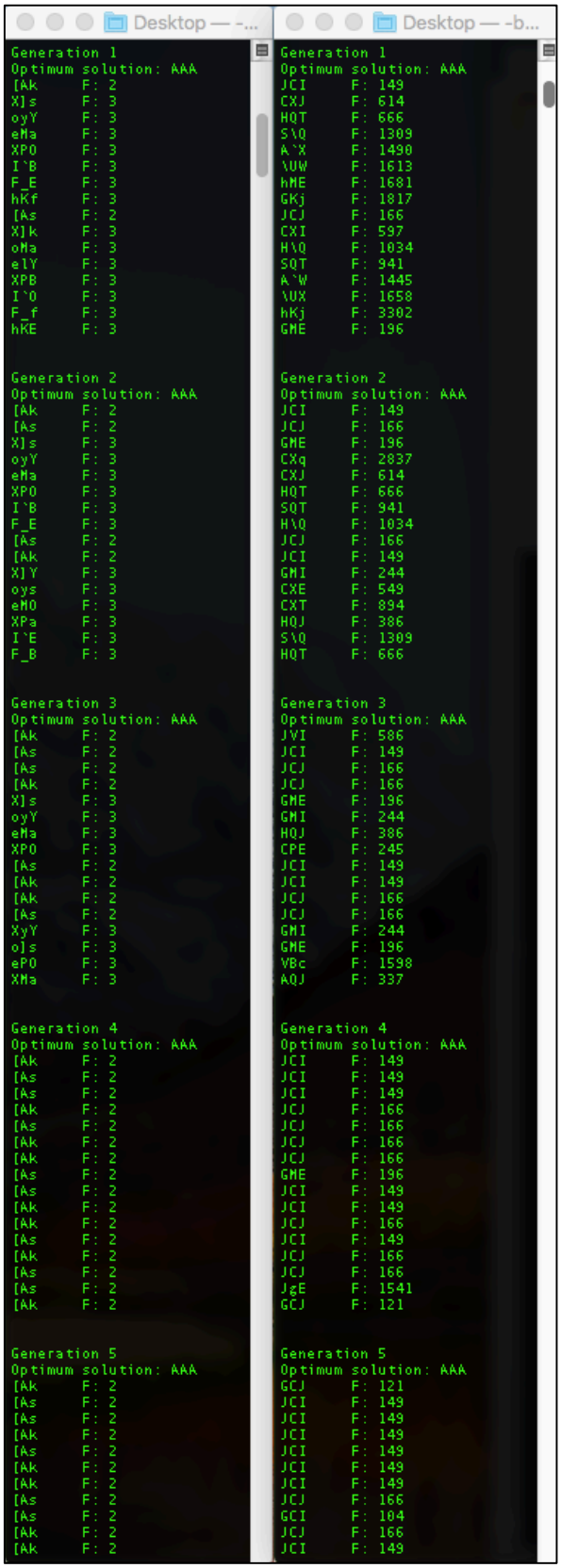

Figure 2 First 5 generations from GA with binary decision as fitness function (left) and with ASCII distance as fitness function (right) for 'AAA' as the optimal solution