Zhixing Chang

Student ID: 104407793

Computer vision

January 26, 2019

GA report

Task 1

For the first task, we have made the generator of the strings. The goal is to let the program randomly generate certain size of population of strings, and generate child from all those parents that are somehow fitting to our target. There are several variables change our result.

1. First, if we change the length of the target phrase, the generations program takes gets larger, since it takes more turns to generate a string that fits all the characters in the target. For example, if the target is “to be or not to be”, it takes 169 generations to get the string that is exactly the same as the target. If the target is “to be or not to be and what else”, the total generation time increase to 627.
2. If we increase the population members, it takes fewer generations to get the result. If I increase the population size to 500, it takes only 27 generations to get the target string, since with the more population, the more easier program gets the target string. Also after the first generation, the generations with 500 population is like 2 times of the generations with only 200 population, so the target is easier to get.
3. If I increase the mutation rate from 1% to 3%, it is still possible to get target, but if the mutation rate is larger than 3, the program cannot find the target within 1000 generations. With the mutation rate as 3%, it takes more generation than 1% which is 198 generations, however, I have got the target with 3% mutation rate 55 generations one time. So this really depends on luck. But in general, with the increasing mutation rate, the target is harder to find.
4. With the larger possible characters being considered when generating population, it is harder to find the target string. When I include all the symbols in the population, it takes more than 500 generations to get the target. When we have more stuffs when generating population, it is possible to have more different characters comparing with the target.
5. The first method, the midpoint way is much slower than the second method. Since the first way takes many useless characters to child. And using the first method, it may take more than 1000 generations to get the target.
6. Increasing the mating factor, which is the maxticket variable in my code can find the target more easily with fewer generations. It makes the random selecting parents system is more likely selecting those strings with higher fitness, so the child can be more close to the target. I think the reasonable value for this is from 10 to 50. If this max ticket or factor gets too big, the parent we select may always be the same, and this will make the next generation having all the same strings which is not what we want.
7. If we increase the maximum generations, it doesn’t affect the result if we can get the target string within 1000 generations. But if we cannot get the target string, the increasing maximum generations can help us find closer string to the target. And after a certain, the whole fitness will converge to a certain point which may be close to the target but increases slowly. If we are lucky, we can get the target string.
8. There are 5 parts of my code:

|  |  |
| --- | --- |
| Calculate fitness | Elapsed time is 0.000018 seconds. |
| , build mating pool | Elapsed time is 0.000034 seconds. |
| , select parent from pool, | Elapsed time is 0.000058 seconds. |
| breed, | Elapsed time is 0.000019 seconds. |
| mutation | Elapsed time is 0.000023 seconds. |

This is the time from the last generation. We can see building mating pool takes some time and select parents from pool takes the most time. I think I should simplify selecting parents by directly getting the index from the pool and use the index to get the parents.

1. With the exponential factor, the total generation time reduces. However, if we have the exponential too large, those member with small fitness may have 0 tickets in the pool, which makes the whole program hard to run. So the reasonable exponential factor should be about 3.

Task 2.5

For 10X10 image

|  |  |
| --- | --- |
| Tol for first two calculating fitness method | 10 |
| Tol for second two calculating fitness method | 15 |
| Size of Population | 200 |
| Max generation | 2000 |
| Tickets | 20 |
| Mutation rate | 1 |
| Mutation range | 30 |
| Random mutation range | 25 |

Since all images are height \* width\* rgb(3), the size of the target matrix increases a lot comparing with the first task. So it takes a lot of time to get a close result. The two tolerance we used for calculating fitness are really important, since these two number decides the whole fitness group which affects our final result. If the tols are too small, it will take infinite time to get an image that is close to target. If the tols are too large, the final result we get will be really different from the target. So after several times of trying, I used 10 for calculating first two methods, and 15 for the second two methods. I make the population size to 200 to make the program run quicker, but if I am running 25X25 size, I will change the size to 500 or 1000. It is hard to find the result within 1000 generation for this large size of target, so I increase the max generation to 2000. I would do this for 5000 for 25X25. I set the others as default.

3. This project is a hard project, but it is a really useful project. First of all, I am really familiar to matlab now, and I am familiar with using vectorizations. I found out that vectorizations can sometime save a lot of time which means a lot for programming. I also learned that the images are 3D matrix consists of numbers from 0 to 255. It is really interesting to see the program auto generate different kinds of images which are nothing, but getting closer to the target image. After doing this project, it will be possible for me to draw a picture using matlab.

I also found out that this project is a really good and important start for this class. The image we take from the real world can be in pixels, so it is possible for us to find something from the image.

Last thing, it really need luck for this program to get the target image.