Assignment 2

Genetic Algorithm

Task - 1 Report

Name - OM SINGH

Roll - 2105634

Section - AI_CS-38

Github Link - https://github.com/singhomIndia/2105634_AI.git

Task - Implement a program that takes an image as its input and generates the same image using N number of squares using Genetic Algorithm.

Here is a report on how I implemented crossover, mutation, and selection in the given task :

Crossover:

I implemented two different crossover operators:

- **Slice crossover: This operator selects a random midpoint in the parent DNA sequences and exchanges the genes after that midpoint.**
- **Random crossover: This operator randomly chooses a gene from one parent and then copies it to the child. It then repeats this process for the remaining genes, alternating between parents.**

I chose to implement these two crossover operators because they are both commonly used and have been shown to be effective in a variety of genetic algorithm applications.

Mutation:

I implemented a simple mutation operator that randomly selects a gene in the child DNA sequence and changes its value to a random number between 0 and 255.

I chose to implement this simple mutation operator because it is easy to understand and implement. It also has the advantage of being able to introduce a wide range of mutations into the population, which can help to prevent the algorithm from becoming stuck in a local optimum.

Selection:

I implemented a fitness-proportionate selection operator. This operator assigns a selection probability to each individual in the population based on

its fitness score. The higher an individual's fitness score, the higher its selection probability.

I chose to implement this selection operator because it is a simple and effective way to select the fittest individuals in the population for reproduction.

Here is a more detailed explanation of how the crossover, mutation, and selection operators are used in the code:

1. **The `perform_natural_selection()` function takes the current population and a selection probability array as input and returns a new population of individuals. This function works by first selecting two parents from the current population using the selection probability array. The `crossover_slice()` or `crossover_random()` function is then used to create a new child individual from the selected parents. The `mutation()` function is then used to introduce a small amount of mutation into the child individual. The new child individual is then added to the new population.**
2. **The `save_fittest_dna()` function takes the fittest individual in the current population as input and saves its DNA sequence to a file. This function is used to keep track of the best individual found so far during the optimization process.**
3. **The `setup()` function initializes the genetic algorithm parameters, including the population size, mutation rate, and crossover operator. It also creates the initial population of individuals. The `setup()` function then enters a loop where it repeatedly performs the following steps:**
    - **Calculates the fitness score of each individual in the population.**
    - **Performs natural selection to create a new population of individuals.**
    - **Saves the fittest individual in the population to a file.**

- **Displays the current generation number and the fitness score of the fittest individual.**
- **Terminates the loop if the fittest individual has reached the target fitness score or a maximum number of generations has been reached.**

Overall, the code implements a simple but effective genetic algorithm for image regeneration. The crossover, mutation, and selection operators are all commonly used and have been shown to be effective in a variety of genetic algorithm applications.