Genetic Algorithm for Image Reconstruction

Bhavik Bhagat
St. Francis Xavier University
Department of Computer Science
Antigonish, Nova Scotia, Canada
x2020coq@stfx.ca

Greeshma Raju
St. Francis Xavier University
Department of Computer Science
Antigonish, Nova Scotia, Canada
x2020bgu@stfx.ca

***Abstract*—A genetic algorithm is a search heuristic based on Charles Darwin's [2] theory of *natural selection* that belongs to the larger class *Evolutionary Algorithm* [3]. This algorithm is modeled after the natural selection process, in which the fittest individuals are chosen for reproduction to produce offspring for the next generation.**

Genetic algorithms [4] are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection. This study uses Genetic Algorithm to generate a target image. These algorithms do not ensure optimal solutions; however, they give good approximations usually in time. In this work, a genetic algorithm is implemented to solve Image reconstruction and results are compared over generations by varying control parameters.

***Index Terms*—Evolutionary Computation; Genetic Algorithms; PyGAD, mutation, fitness function.**

I. Introduction

Genetic Algorithms use the stochastic approach based on biological evolutionary processes proposed by Holland [5]. In nature, the most suitable individuals survive and mate to produce the next generation of offspring. Environments are filled with populations of individuals that strive for survival and reproduction. In genetic algorithms, a population of candidate solutions (called individuals) to an optimization problem is evolved towards a better solution. Every candidate solution in the population has properties that can be mutated or recombined. Usually, the candidate solutions are encoded using bits (0s and 1s), but other types of encoding exist. The Evolved Antenna 1 was generated using a similar evolutionary technique to create the best radiation pattern, even though the design was pretty complicated and humans wouldn't imagine creating such a design. That is why these algorithms can be an asset in solving some problems.

Genetic algorithms are based on the ideas of natural selection and genetics, based on the "survival of the fittest". In the method section III algorithm is explained in detail to generate the target image.

II. Data

Since this solely relies on the target image we want to reconstruct, we need the images as input data. We are trying to generate three images with varying dimensions to see which image is reconstructed with the least noise.

1) flower 2

Fig. 1: The 2006 NASA ST5 spacecraft's Evolved Antenna

2) fruit 3
3) mickey 3
4) smiley 4

III. Methods

This project uses PyGAD, an open-source Python library to execute the genetic algorithm and training machine learning algorithms. PyGAD supports 19 different parameters for customizing the genetic algorithm for various applications.

A. Genetic Algorithm

The basic steps of Algorithms are as follows:

1) Create Individual and Generate Population

Fig. 2: target: flower

Fig. 4: target: fruit

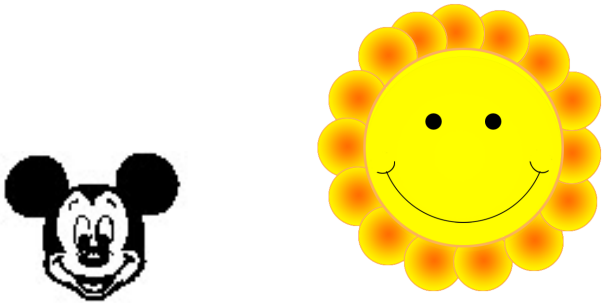

Fig. 3: target: mickey, smiley

2) Evaluate Fitness
3) Selection
4) Crossover
5) Mutation
6) Next Generation

B. Detailed Explanation

Environments are filled with *populations of individuals* that strive for survival and reproduction. Individuals are subjected to selection which evaluates how well they exist in the environment relative to the rest of the population. We will generate a *fitness function* once the population is initialized, then calculate the fitness of the individuals in the population. And then for *selection*, based on the different types of selection techniques, fit individuals will seed the next generation/population. Fitness is usually the value of the objective function in the optimization problem being solved. The individuals with more fitness values are randomly selected from the current population. Then based on the *crossover strategy* selected individuals are recombined to create a new individual, followed by a *mutation*. Both crossover and mutation are based on probability. There are various mutation strategies can be applied. This mutation creates diversity in the population, helping to overcome local minima or maxima depending on the search-space of the problem. This new individual will be populated in a *new generation for next iteration*. This process is repeated for several generations or until we meet the criteria to stop. This criterion could be a particular fitness value reached that we desired. These different parameters can be tuned to get a better solution.

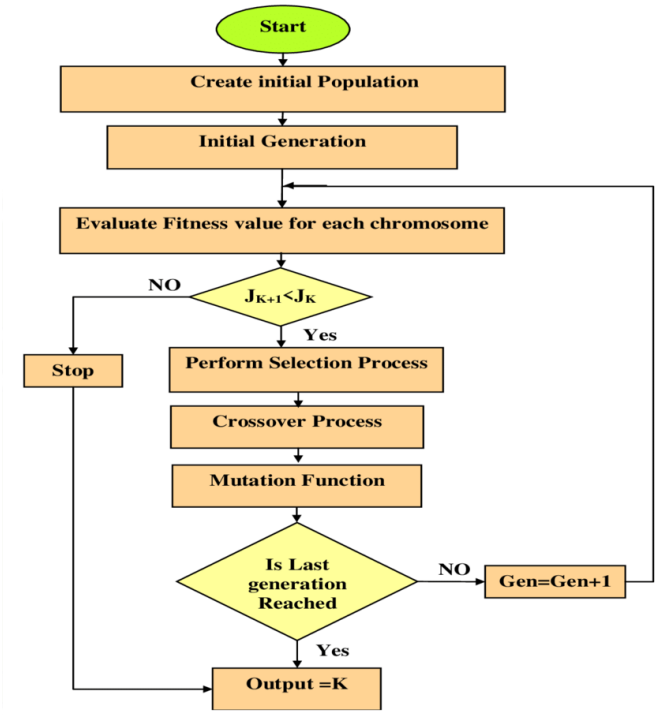

Fig. 5: Genetic Algorithm

This flowchart gives a better idea of the workflow of the basic genetic algorithm.

C. Genetic Algorithm Program settings

The constructor of the PyGAD.GA class has 19 parameters, of which 16 are optional. The three required parameters are: num-generations - Number of generations numparents-mating: Number of solutions to be selected as parents fitness-func: The fitness function that calculates the fitness value for the solutions.

fitness-func parameter allows the genetic algorithm to be customized for different problems, and we have defined this function that calculates the fitness value for a single solution We have mainly concentrated on and explored the parameters best suited for image processing.GA class in PyGAD module accepts several parameters that allow us to customize the Genetic algorithm depending on the application requirements. Tested the GA class multiple times by varying these parameters and identified the parameters listed in TABLE I as the ones most suited for image reconstruction.

TABLE I: GA class parameters

Population	20000
Fitnessfunction	fitnessfunction
Genes	Depends on image size
Solution Per Population	20
Parents Mating	10
Generations	20000
Mutation Type	random,adaptive
Mutation Probability	0.01
crossover	single point crossover
Minimum Mutation	0.0
Maximum Mutation	1

Along with the above parameters for the Genetic Algorithm class, we tested the program with different types of mutations, selections, and crossover strategies. To accommodate these options in the code, the ”parameters” variable was created to identify the tuning parameters for each iteration, and it is of the form :

```
parameters = {
    "crossover_prob": crpb,
    "mutation_prob": mtpb,
    "num_generations": gen,
    "mutation": mutations_strategy,
    "parent_num": num_parents,  }
```

Varied the values of the above parameters to account for the modifications discussed in section IV.

IV. Modifications

For this problem, we have tried several combinations of modifications for genetic operators to generate an optimum result. These strategies are included in the PyGAD [1] library.

A. Selection Modifications

1) Steady State Selection
2) Roulette Wheel Selection
3) Stochastic Universal Selection
4) Rank Selection
5) Random Selection
6) Tournament Selection

B. Crossover Modifications

1) Single Point Crossover
2) Two Points Crossover
3) Uniform Crossover
4) Scattered Crossover

C. Mutation Modifications

1) Random Mutation
2) Inversion Mutation
3) Swap Mutation
4) Scramble Mutation

The effect of each method on the fitness and overall solution will be discussed in the result V section. Some modifications may work better than others relying on the problem we are trying to solve. In addition, hyper-parameters like the number of generation, population size, crossover rate and mutation rate, selection pressure, etc. also impacts the results. The tuning of hyper-parameters is a tedious task. Practically, the solution that works using fewer resources is preferred over having the best solution. Hence, often we see a stopping criteria as a particular fitness achieved or in a general number of generations.

Apart from the modifications discussed above. We tried the adaptive mutation technique. Usually, when we do mutation based on a random probability, we apply that to all candidate solutions regardless of their fitness value. This is why regardless of the solution's fitness, the same number of genes are mutated each cycle. Instead, we used this adaptive mutation technique, which adapts according to the fitness solution. It works as follows:

1) Determine the fitness of the population f_{avg}
2) Find the fitness value of each chromosome f
3) If $f < f_{avg}$ then the solution is low quality, hence the mutation rate for that should be high to increase the quality of the solution
4) If $f > f_{avg}$ then the solution is high quality, so the mutation rate should be low to maintain the quality high, thus avoiding degradation in the quality.

We need two values for the mutation rate (high and low), where a high mutation rate should be used for the low-quality images. Likewise, a low mutation rate should be used for the high-quality images we are trying to reproduce.

For the tournament selection, we have to pass an extra parameter, the number of tournaments to select the best parents.

V. Results and Discussion

This Genetic algorithm starts from a randomly generated image of the same shape as the input image. Then it tried to reproduce an image similar to the original image using crossover, mutation, selection, and other hyper-parameters

modifications. From the available mutations, ”random” produced the best image reconstruction results on the input images. As mentioned in Section II, we used four different input images for testing. We calculated fitness statistics across images for several parameters and tabulated the results in the TABLE II. Fitness for the best solution and index of the best solution is displayed on the terminal after a predefined number of generations complete successfully. Among all the input images tested so far, the flower has produced the least accurate output image after reconstruction, which could be related to its large size. Modifications related to crossover and selection did not help to improve the image reconstruction accuracy to a great extent. On the other hand, it caused the execution time to increase drastically and produced low-quality images. Plot comparing the evolution of the Mickey image from 1000 generations to 20000 generations are shown in the below figure :

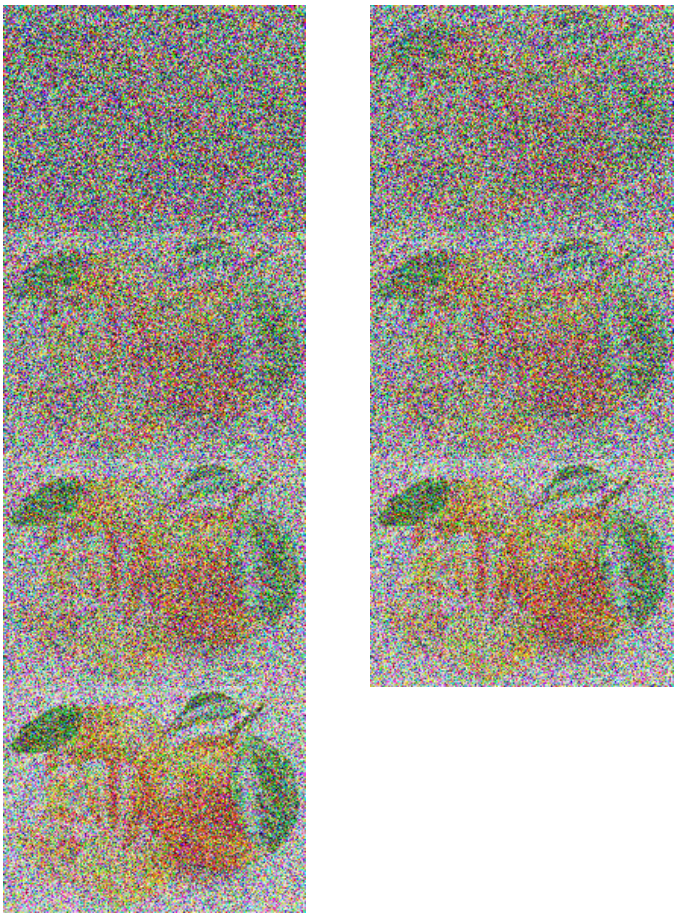

Fig. 6: Reconstruction results for Orange image evolved over generations: 1000,4000,7000,8000,12000,16000 and 20000

A plot comparing the evolution of the Mickey image from 1000 generations to 20000 generations is shown in Figure 7. Similar images were generated for smiley and flower input as well. Tested the program with different mutations on the same image and observed that ”random” mutation gave a linear increase in the fitness over the generations while swap and

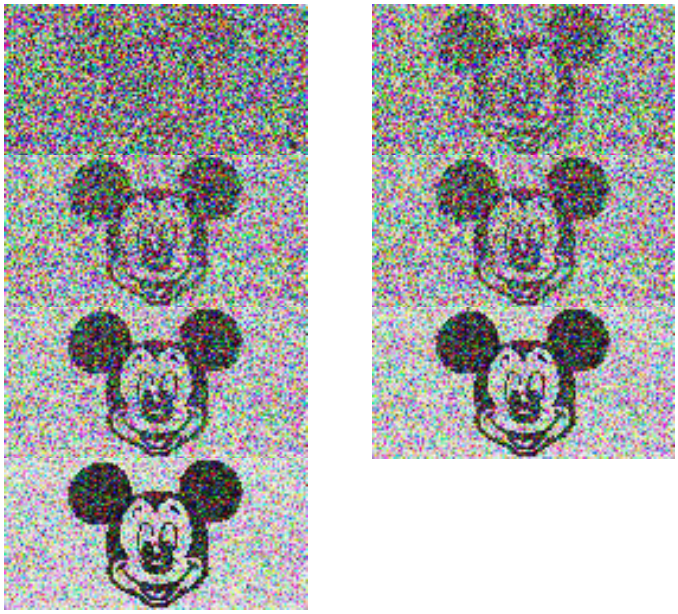

Fig. 7: Reconstruction results for Mickey image evolved over generations:1000, 4000,7000,8000,12000,16000 and 20000

inversion mutations showed a step-wise growth in the fitness value as shown in Figure 8.

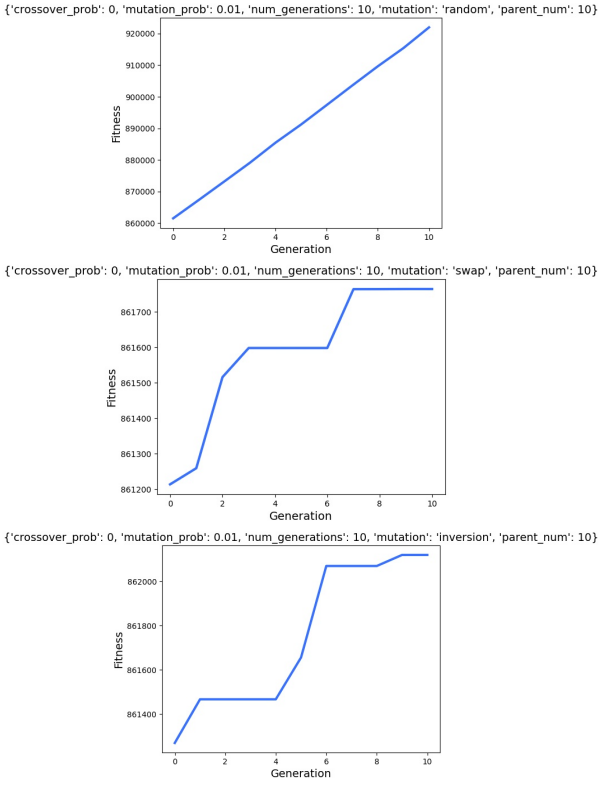

Fig. 8: Comparison of Mutation modifications for small generations

The fitness function for random mutation has converged

TABLE II: Summary statistics of the Fitness Results

Input Image	Mutation Type	Generation	Fitness
Orange	random	1500	31612.55
		3500	32467.28
		4500	33386.63
		6000	35328.48
		7500	36254.21
		10200	37846.02
		14460	39792.18
		16000	40391.48
		20000	41748.40
Smiley	random	3500	412914.64
		4000	414723.78
		6000	421920.93
		8000	428204.74
		10000	433757.91
Mickey	random	3500	88267.32
		4000	89471.63
		8000	10240.31
		12000	11057.52
		16000	12650.62
		20000	13031.23

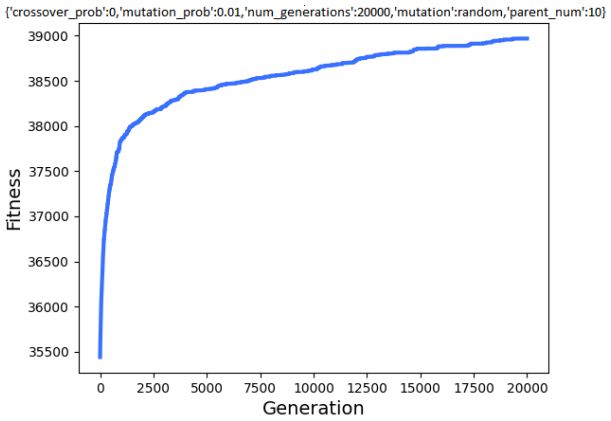

Fig. 9: Fitness function for random mutation

after 8000 generations as shown in Figure 9.

VI. Conclusions and Future Work

During the image reconstruction using the PyGAD library, we observed that the program works with gray and RGB images. Testing the program on different input images led to the conclusion that reconstruction quality mainly depends on the input image's pixel values and the mutation type. Image reconstruction produced accurate results with a random mutation that adds a random value to each gene selected for mutation. As per this study, the random value range for the image reconstruction was 0.0 to 1.0. Pixel values of the image play a significant role while choosing the range of values used in the initial population. We reconstructed some small images (i.e., Mickey) with relatively high accuracy than bigger images. There is scope to run the algorithm for more than 20000 generations using adaptive mutation as part of future work. We could not run the algorithm for more than 20000 generations due to our systems' hardware limitations.

VII. Appendix

Contents

List of Figures

References

[1] Ahmed Fawzy Gad. Pygad: An intuitive genetic algorithm python library, 2021.

[2] Wikipedia contributors. Charles darwin — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Charles_Darwin&oldid=1059767812, 2021. [Online; accessed 12-December-2021].

[3] Wikipedia contributors. Evolutionary algorithm — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Evolutionary_algorithm&oldid=1055165596, 2021. [Online; accessed 12-December-2021].

[4] Wikipedia contributors. Genetic algorithm — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=1051184365, 2021. [Online; accessed 12-December-2021].

[5] Wikipedia contributors. John henry holland — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=John_Henry_Holland&oldid=1041115860, 2021. [Online; accessed 12-December-2021].