NOVA IMS Information Management School

MDSAA

Master's Degree Program in

Data Science and Advanced Analytics

Computational Intelligence for Optimization

Project: Image Recreation

GitHub Link: https://github.com/evaraina/ProjectCIFO

Eva Raina 20231900
Fatma Boumelala 20230502

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

3rd June 2024

INDEX

1. OBJECTIVE

This project aims to explore the optimization concepts of Genetic Algorithms (GA) by applying them to the challenge of image recreation. The main objective is to use GA principles to recreate a given target image through a population of candidates that evolve over generations. This involves implementing key GA components such as fitness functions, and selection, crossover, and mutation operators.
The genetic algorithm (GA) starts from a randomly generated image of the same dimensions as the target image. This randomly generated image is evolved, using crossover and mutation, until it reproduces an image similar to the original one. The project aims to fine-tune the GA parameters to achieve the best possible image recreation, demonstrating the potential and limitations of genetic algorithms in optimizing complex visual tasks.

2. METHODOLOGY

2.1 Individual

The first step in the implementation of a genetic algorithm is defining the class Individual, in our approach each individual has a given length and width corresponding to the dimension of the target image and it's characterized by a representation; the representation can be given when the individual is defined or it's randomly generated by the class and it's unique to each different individual since it's a 3D array containing RGB values of each pixel.

RGB values represent the color model we based our image recreation on, that uses primary colors: Red, Green, Blue. In the RGB model each pixel is defined by triplet of integer values ranging from 0 to 255, the final color of the pixel is a combination of different intensity of the three main one: for example, an RGB value of (255,0,0) represents full intensity red, while the RGB value (255,255,255) represents the white color.

2.2 Fitness

Each individual added to the population is assigned a fitness value which helps determine how similar it is to the target image and also how likely it is to be crossover in the next generation.
In our approach the class Individual implements two different fitness functions, one based on the mean square error and another one based on the human visual perception.

MSE fitness function

This function takes as an input the target image representation and then computes the mean square error between the RGB values of the individual and the RGB values of the target image.

HVS fitness function

This function implements the human visualization system and computes a more realistic difference between the individual and the target image. The idea is that human perception of color doesn't always align with the RGB values: two colors that are considered similar by a

human eye may have a very different combination of primary color and appear very different in the RGB composition.
To overcome this problem, we transformed each pixel in a grayscale value and we computed the weighted sum to reflect the luminance of each pixel; this allows us to work on the difference of saturation and brightness instead of the RGB values and have a more realistic color representation.

The HVS function takes as an input the target image representation and computes the total sum of values of the difference between the grayscale values of the individual and the grayscale values of the target image.

2.3 Selection

The selection process decides which individuals of the current generation are chosen to possibly go through crossover and mutation and create the future generation.
In our case we implemented two different types of selection approaches in the class Population.

Tournament selection

The tournament selection function has the desired tournament size as input and it is divided in two parts: a first probabilistic part randomly chooses from the current generation as many individuals as the size of the tournament, a second deterministic part computes and confronts the fitness of the chosen individuals and selects the best fit as return.
Since for every tournament there is only a winner the function will be implemented twice to obtain the parents couple.

Fitness proportionate selection

Differently from the tournament selection, the fitness proportionate selection function computes the fitness for each individual in the population and assigns to them the probability of being chosen; this probability is proportionate to the fitness: since this is a minimization algorithm, lower is the fitness, higher is the probability of being selected.
At last, a casual number, between zero and the sum of all the fitness of the individuals in the population, is drawn; the individual correspondent to the value is selected.
Again the function will be implemented twice since it selects one individual each time.

2.4 Crossover

In genetic algorithms, the crossover process is an important operator used to combine the genetic information of two parent individuals to produce new offspring. The idea is to create new individuals that inherit characteristics from both parents, potentially leading to better solutions over successive generations. Every couple of parents has a probability p of either going to crossover or being directly copied to the next generation, in general p is a high value since the goal is to get better individuals every generation, we will use a baseline of $p=0.90$.

Since different crossover techniques lead to different results, in the algorithm there are three different crossover functions that will be later analyzed to decide the best option.

Single Point Crossover

The single point crossover is one of the most common functions implemented in the genetic algorithms. It randomly chooses a point in the representation array and two new arrays are then created: the first array takes values from the first parent up to the chosen point and the remaining values from the second parent, the second array does the opposite, taking values from the second parent up to the chosen point and the remaining values from the first parent. The function then returns the two offspring individuals with the corresponding representation array created.
This crossover approach allows to keep a good percentage of representation from each parent untouched in the offspring.

Uniform crossover

The uniform crossover function generates two representation arrays from two parents' one by assigning each pixel randomly from either of the parents with a 50% probability. It then returns the two offspring individuals corresponding to the created arrays. The process ensures that the genetic material from both parents is mixed evenly in the offsprings.

Row wise crossover

The row wise crossover function is similar to the uniform crossover but, as the name says, it works with a complete row instead of with pixels; each offspring row belongs either to parent one or parent two with probability 50%.
This approach allows to keep more of the genetic representation of the parents intact in the offspring but still ensure a uniform distribution of it.

2.5 Mutation

Mutations introduce new genetic structures into the population by randomly altering some of the genes of the individuals. This is essential to prevent the population from becoming too similar over time, which can lead to premature convergence.
Moreover, genetic algorithms can sometimes get stuck at a local optimum, where the population converges to a solution that is not the best possible. Mutation helps provide new generation variations that can lead the search out of the local optimum.

Since the main goal is not a random search but to keep the beneficial combinations of the best individuals so far, usually the mutation probability is low; we will use a baseline of probability $p=0.15$.

Inversion mutation

The inversion mutation approach creates an alteration in the representation by inverting a portion of it. For easier manipulation of the pixel the function starts by flattening the 3D array in 1D, then the function randomly selects two points in the individual array, it sorts them out by positional order, and then it inverts all the pixels inside the segment. Finally, it recovers the 3D shape of the representation and returns the final corresponding individual.

Swap mutation

The swap mutation function is similar to the inversion mutation, but it produces a smaller alteration by exchanging two bits, randomly selected, instead of the full segment between them.

3. EXPERIMENTS

By systematically varying the fitness functions, crossover methods, and population size we tried to identify the optimal configuration for achieving the best image recreation. The experiments were designed to measure not only the final quality of the recreated image but also the convergence rate and robustness of the algorithms.

Common Parameters:

- Number of generations: 20,000
- Crossover probability: 0.9
- Mutation probability: 0.15

The first experiment used a population of 100 individuals and evolved them over 20,000 generations. Tournament selection, uniform crossover, and swap mutation were employed, with the HVS fitness function on a circle image (Figure 1.a).

Since the image primarily used white and yellow colors, the algorithm wasn't able to distinguish between the two colors but was able to replicate the black circle. We hypothesized it's because the grayscale-based HVS fitness function was less effective in distinguishing subtle differences in contrast of these colors (Figure 1).

For the second test, the same parameters were maintained, but a geometric pattern was used as the target image (Figure 2.a). The distinct shapes and colors of the geometric pattern led to a noticeable improvement in the results, demonstrating the effectiveness of the HVS fitness function in handling images with more distinct visual features such as contrast between colors (Figure 2).

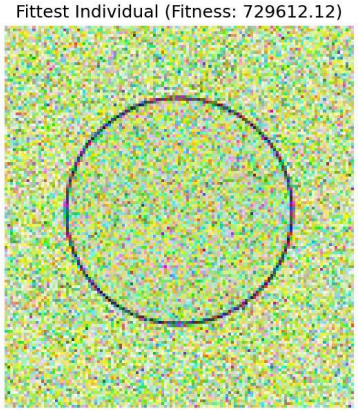

Figure 1: circle recreation

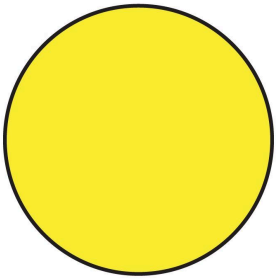

Figure 1.a: target circle image

Figure 2: geometric pattern recreation

Figure 2.a: target image geometric pattern

3.1 Comparison of Crossover

In the crossover comparison experiments, three crossover techniques were tested to understand their impact on the genetic algorithm's performance.
The functions were independently evaluated while keeping all other variables constant and each test was repeated five times to ensure consistency and reliability of the results.
Then we computed the mean of the fitnesses in the five different trials for each crossover and we plotted the results for a better comparison.

In the GA we also implemented a third algorithm, the **row-wise crossover**, but it was not included in the comparison because during the tests it was triggering the break code after a few hundred generations due to convergence of the fitness function.

In the comparison test we first implemented tournament selection, swap mutation, and **uniform crossover** *(Figure 3)*combined with the HVS fitness function and a population of 50 on the target image *(Figure 5)*.

Then we repeated the test maintaining the same parameters as in the previous test but with a **single-point crossover** instead *(Figure 4)*.

The graph showing the performance of the algorithms across 20,000 generations indicates that uniform crossover outperformed the single point crossover since in the first case the fitness level dropped to less than a million, whereas in the second case the fitness level only decreased to slightly under 1.3 million.

The difference between the two performances can also be visually seen in the images below, where the uniform crossover *(Figure 5.a)* was able to recreate a more detailed image compared to the single point crossover *(Figure 5.b)*.

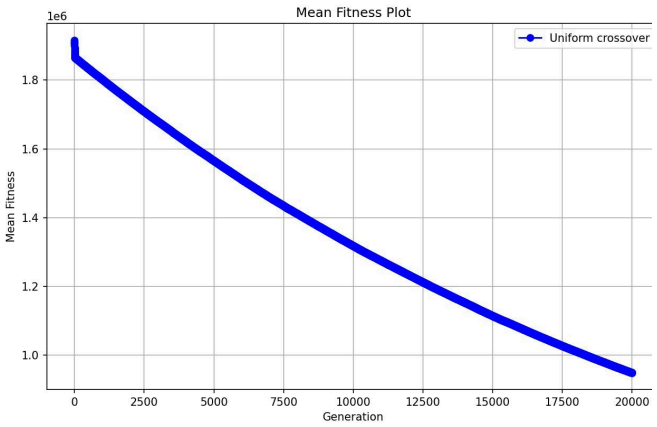

Figure 3: graph of uniform crossover

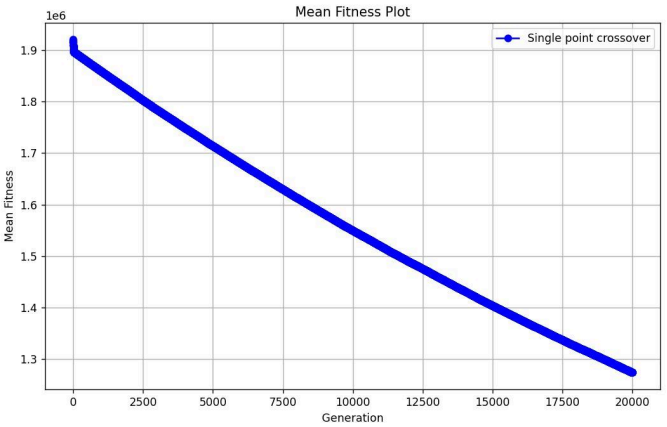

Figure 4: graph of single point crossover

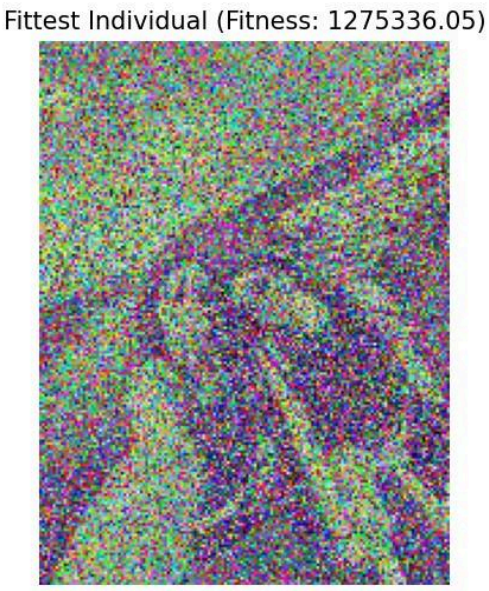

Figure 5.a : uniform crossover

Figure 5.b: single point crossover

3.2 Comparison of Fitnesses

To compare fitness functions, an ulterior test was implemented, where we used the same method as for the crossover comparison repeating the algorithm 5 times, and then selected the best fitness for each function.

For both tests done for this comparison, we used the same target image (Figure 5) and we implemented the same methods: tournament selection, uniform crossover and swap mutation and a population of 50 individuals. However, we applied the mean fitness function for the first one and the HVS fitness function for the second.

The results showed a significant decrease in performance with the mean fitness function that was not able to show any resemblance to the target image, indicating that the HVS fitness function is a lot more effective for our purposes.

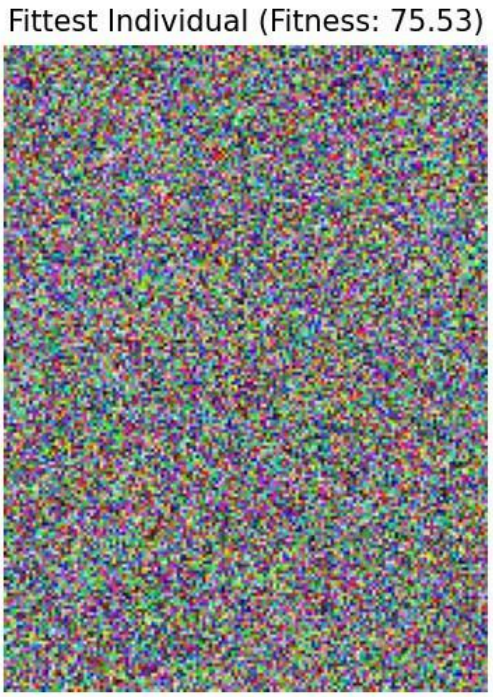

Figure 5.c: MSE fitness function

Figure 5.b: HVS fitness function

4. LIMITATION & CONCLUSION

By systematically varying these parameters and recording the performance, we aimed to identify the most effective combinations for achieving high-quality image recreation through genetic algorithms.

In our implementation the most functional implementation was with tournament selection, uniform crossover, swap mutation and HVS fitness function with a population of 50 individuals.

However, there are limitations to consider in this project. Achieving good results with this approach is computationally expensive. This high computational cost arises from the extensive processing needed to evolve the population and apply genetic operators over thousands of generations.

For example, to make the process faster and computationally cheaper, we had to reduce the size of the images compromising some details in the images for a lower number of pixels and we implemented the main tests with a population of 50 individuals; with a higher population may be possible to have even better results but it will be computationally too expensive: we try one execution of the code with 100 individuals (Figure 5.a) but it was computationally impossible to test it for a higher number of times.

Moreover, while the HVS function is able to imitate the human visualization of the color it still has some difficulty to recognize colors with similar contrast, as seen on the circle image (Figure 1), this results may results in a less effective recreation of the image or in a similar image with different color tones.

Figure 5.e: image recreation

Figure 5: target image

5. DIVISION AND REFERENCES

Image Recreation Project

https://medium.com/@sebastian.charmot/genetic-algorithm-for-image-recreation-4ca546454aaa

https://www.irjet.net/archives/V8/i4/IRJET-V8I4182.pdf

https://www.linkedin.com/pulse/reproducing-images-using-genetic-algorithm-python-ahmed-gad/

Division of Labor

Eva:

- code of the class Individual
- code of the visualization function
- code of the row-wise crossover method
- computation of the mean of fitness and graphs
- report

Fatma:

- code of evolve function
- code of selection functions
- code of uniform and single crossover functions
- code of the mutation functions
- report