Genetic Algorithm: Evolution of Mona Lisa

Adrià Fenoy
January 22, 2019

CONTENTS

1 Introduction **2**

2 Genetic Algorithm **3**
2.1 Population 3
2.2 Fitness 4
2.3 Selection 6
2.4 Crossover 7
2.5 Mutation 8

3 Hill Climbing Algorithm **8**

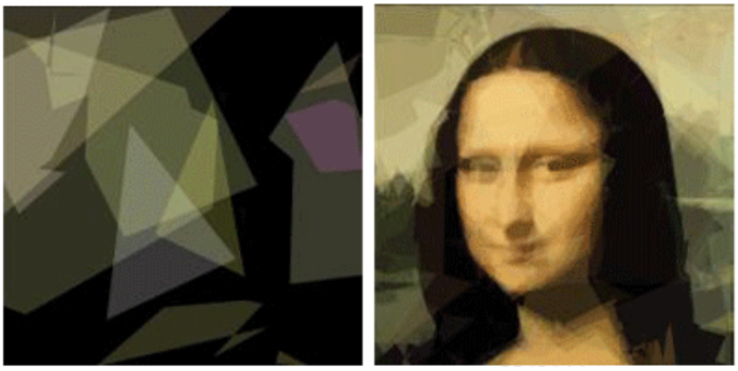
4 Results **9**
4.1 Genetic Algorithm 10
4.2 Hill Climbing Algorithm 11
4.3 Fitness convergence 12

5 Conclusions **13**

References **14**

1 INTRODUCTION

The goal of this work is to design a genetic algorithm that tries to reproduce Mona Lisa with the superposition of several semitransparent polygons with different colors. This is an original idea published in 2008 on Roger Johansson's blog [1].



(a) Generation 2022 (b) Generation 904314

Figure 1.1: Mona Lisa comparison in two different stages of the algorithm (Roger Johansson's solutions).

Because of the previous approach consists in a parent and a slightly mutated son competing to be the best on each generation, what he is doing is not exactly a genetic algorithm, but a hill climbing or evolutionary algorithm instead. In this report both approaches will be implemented and compared in order to see which is the algorithm that reproduces Mona Lisa the best.

2 Genetic Algorithm

A genetic algorithm is an algorithm inspired by natural selection which purpose is to find an optimal solution using analogs to mutation, crossover and selection processes occurring during evolution [2]. The usual flowchart of a genetic algorithm is as follows.

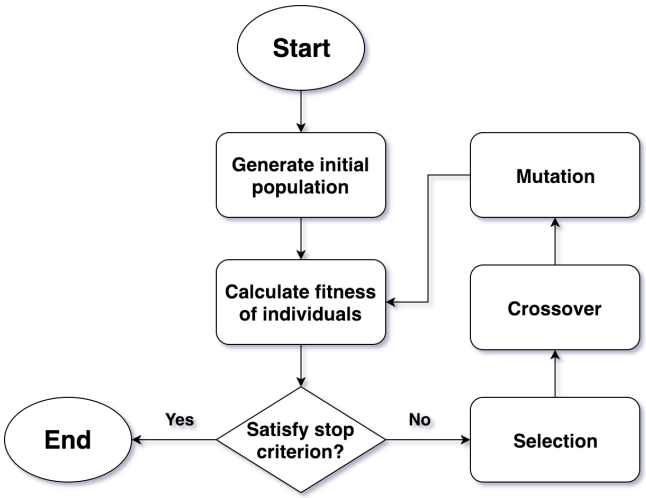


Figure 2.1: Genetic algorithm flowchart

The genetic algorithm that is going to be presented follows this schema, but it has some peculiarities in each of the steps (selection, crossover and mutation), which will be explained in the following sections. The hill climbing evolutionary algorithm also follows this schema, with the only difference of avoiding crossover and using different selection criteria. The mutation part is the same in both versions.

2.1 Population

The population for this problem consists in many individuals, each one a different imitation of Mona Lisa. Each of these individuals contains a given number of polygons with a given number of vertices. Each vertex is a point in a 2D space limited by the size of the original picture. Each color is represented in RGBA, this means that it can be represented by a vector of four components corresponding to red, green, blue and the transparency. This can be pictured in the following figure.

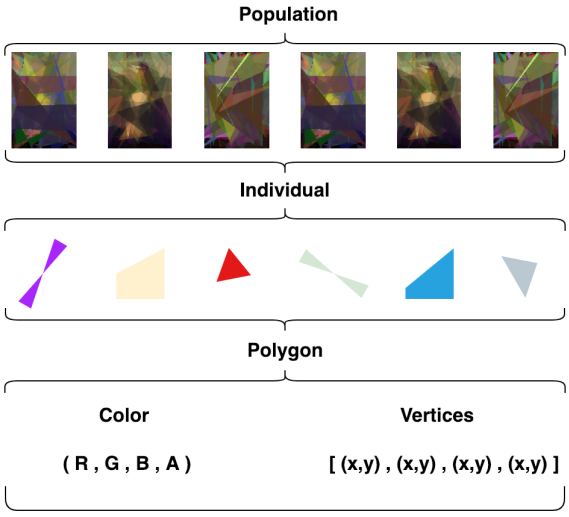


Figure 2.2: Schema of the population and its constituent parts.

The drawing of the polygons, image display and other operations that involve images are done with Python 3 using the library Pillow[3].

2.2 Fitness

Before explaining the algorithm, it needs to be described how fitness will be computed for this problem. Since we are comparing an original goal image to another candidate image, we can perform a pixel per pixel color difference, but there is not one unique way to compute the color difference. Three different color differences have been tested[4]:

- Euclidean distance:

$$\Delta C = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2} \tag{2.1}$$

  It has been tested with and without the squared root in order to avoid unnecessary computation. There were not appreciable differences in the results that might indicate that one provides a better images than the other.

- Absolute distance:

$$\Delta C = |R_2 - R_1| + |G_2 - G_1| + |B_2 - B_1| \tag{2.2}$$

This version tries to reduce more computational cost. Again, there were not appreciable differences in the results with the use of other distances.

- HSV distance:

  This one is a little bit more complicated, but it tries to provide an homogeneous color difference based on human perception:

$$\Delta C = \sqrt{2 \cdot \Delta R^2 + 4 \cdot \Delta G^2 + 3 \cdot \Delta B^2 + \frac{\bar{r} \cdot (\Delta R^2 - \Delta B^2)}{256}} \tag{2.3}$$

  where

$$\Delta R = R_2 - R_1 \,, \tag{2.4a}$$

$$\Delta G = G_2 - G_1 \,, \tag{2.4b}$$

$$\Delta B = B_2 - B_1 \,, \tag{2.4c}$$

$$\bar{r} = \frac{R_1 + R_2}{2} \,. \tag{2.4d}$$

  Again, this one did not seem to improve results or even make the algorithm converge faster.

Despite of the fact that same results were obtained with all fitness functions, it is possible that doing a deeper analysis of each one of them, differences would actually be found.

The final fitness of one individual corresponds to the sum of all pixel by pixel fitness. In addition, the fitness function has been normalized in order to obtain a more readable result, that is easy to compare when we compare results for different models. Taking these two facts into account, and using the squared Euclidean distance, the final fitness is computed like follows:

$$f = 1 - \frac{\sum_{r=1}^{n_r} \sum_{c=1}^{n_c} \Delta C_{r,c}}{3 \cdot n_r \cdot n_c \cdot 255^2} \tag{2.5}$$

where n_c and n_r are the number of columns and number of rows. Note that color difference was calculated with the squared Euclidean distance. If another distance were used, the parameters on the denominator might be changed. The 3 comes from each of R, G and B; and 255 comes form the color range, which goes from 0 to 255.

2.3 SELECTION

For the genetic algorithm it has been used two different selection methods:

- Crossover Selection:

  The first one is regular crossover selection. This method consists on picking a given number of individuals (tournament size) from the whole population and, from these individuals, pick the best one (according to their fitness).

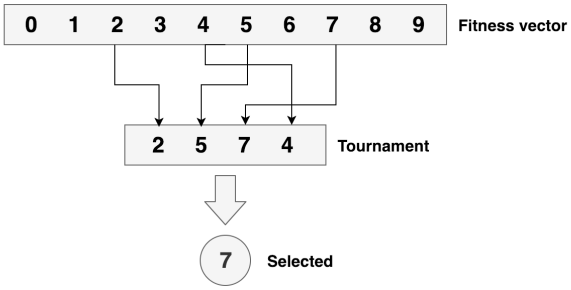
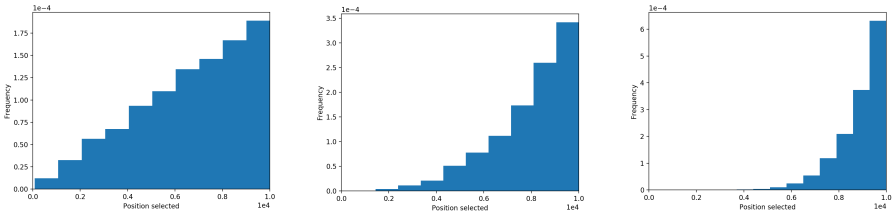


Figure 2.3: Tournament selection example (Higher fitness is better for this example).

  This provides a set of parents where parents with better fitness are more likely to be selected. This method picks individuals according to their position in the sorted list of fittest individuals, this means that concrete values of the fitness are not relevant. This can be better appreciated in the figures below.



(a) Tournament size = 2 (b) Tournament size = 4 (c) Tournament size = 8

Figure 2.4: Frequency of selection for each position in the fitness vector of length 10000 for different tournament size.

  As it can be appreciated in the previous histograms, increasing the tournament size makes the algorithm more exploitative, meanwhile reducing it makes the opposite effect. So this is the mechanism to control the greed of the algorithm during selection.

- Elitist Selection:

  The second selection method is an elitist selector. What it does is to pick the best individuals of one generation and make sons identical to their parents for the next generation. This is a very greedy component of the algorithm, and not many parents should be sent directly to the next generation. Despite of this fact, it is useful because it allows to be more exploratory in the mutation stage without the risk of loosing the best solution.

  In fact, using an elitist selection was inspired by the version of the algorithm used originally by Roger Johansson. Despite his version do not perform crossover, it always chooses the best individual between the mutated son and the parent, which is analogous to elitism without performing crossover afterwards.

2.4 Crossover

For crossover, since there are no restrictions in how a individual must be constructed with a given set of polygons (in contrast to other problems like traveling salesman[5]), most of the crossover methods can be used successfully for this problem.

Uniform crossover is the crossover method implemented in the code. This method consists in swapping some of the polygons from one parent to the other with a certain probability.

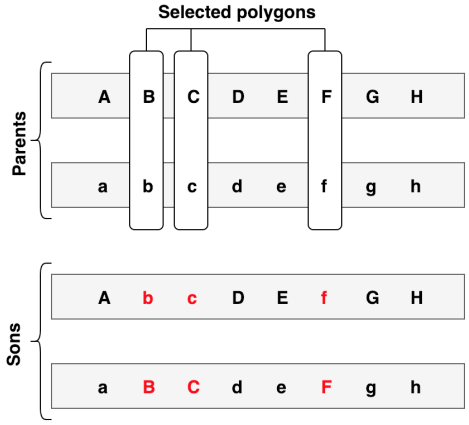


Figure 2.5: Uniform crossover example.

An example of uniform crossover can be seen in the previous figure. Polygons are swapped with other polygons in the same position, this is done this way because the order of polygons is important due to the fact that the final color is computed using alpha composition which

is not commutative [6]. For the same reason, none of the mutations performed will swap position of the polygons, which ensures that each polygon stays in his position during the whole evolution process.

2.5 Mutation

Mutation is the most important part of a genetic algorithm. Thanks to it, the individuals keep improving and getting closer to the optimal solution. It is also the part of the algorithm where the designer of it can be more creative, as possible mutations are very different according to the problem that is being faced. That said, it is important to use this creativity to follow the main goal of the problem which, in this case, it is reproducing Mona Lisa. It is also important to make only those mutations that are essential to let the individuals evolve, or otherwise the problem can have too many free parameters which will make more difficult to find the right ones that solve the problem.

In the code, we limited the amount of mutations to two:

- Vertex: Changing one vertex randomly chosen in the polygon.
- Color: Changing one of the four components of the colour randomly chosen in the polygon. The four components are red, green, blue and transparency, all of them taking values from 0 to 255.

This mutations happen once per polygon in each individual, and it only happens one or the other according to a certain probability. Moreover, the strength of the mutation (which is the range of values a given value can take after mutating) is limited. This strength varies during the execution of the program according to the following expression:

$$s = s_0 \sqrt[4]{1 - f} \tag{2.6}$$

where s_0 is the starting strength. This reduces the strength of mutations as fitness keeps improving in order to achieve very small mutations when we are very close to the optimal solution.

3 Hill Climbing Algorithm

Another version of the program has been implemented. This is a similar version to the one proposed by Roger Johansson in his blog. The main difference with the genetic version is that in this case crossover operations are not performed. For this reason, this would not be considered a genetic algorithm but a hill climbing evolutionary algorithm instead.

The other big difference is in the selection method. In this case the tournament selection is avoided and it only has been used an elitist selection with the parent and the mutated child, only the best of these two is surviving to the next generation. The mutation part of the algorithm remains the same, but the values for the parameters related to mutation are adapted to perform better for this algorithm.

The advantage of this version respect to the genetic algorithm is time. Meanwhile in the genetic algorithm it has been used a population of 21 individuals, in this version only one individual was used. This means that ha fitness calculation (which is by far the most time expensive part of the algorithm) only has to be performed once. This means that, in the end, the time per generation for the genetic algorithm will be 21 times higher. This may seem a good reason to think that the hill climbing algorithm is a better choice, but the genetic algorithm also has his own advantages. The first thing to notice is that, because we have 21 individuals mutating, we are exploring the space of solutions in 21 different ways, meanwhile hill climbing algorithm is only exploring one mutation per generation.

4 Results

In this section, the results obtained with the previous algorithms will be presented and compared between them. The figure below shows the reference image used to compare each individual with. Its size is 97x145 pixels, which was a good balance between resolution and computation time (remember that the fitness was computed pixel by pixel).

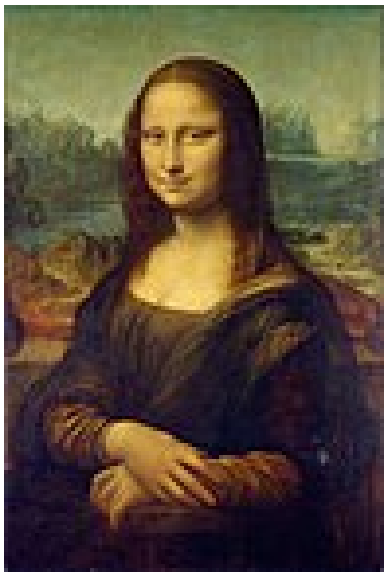
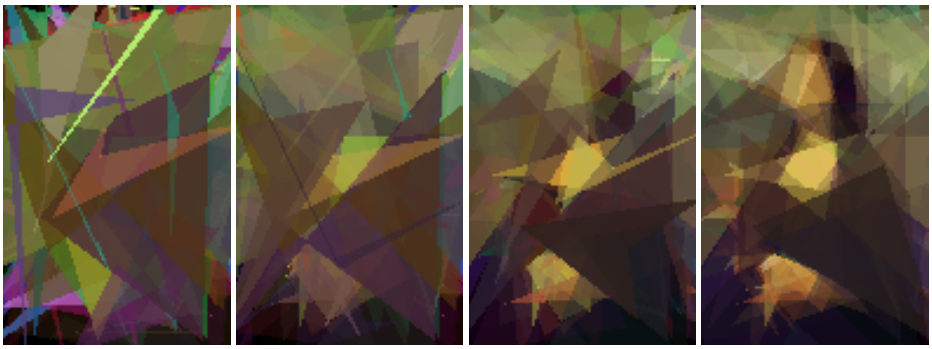


Figure 4.1: Image of Mona Lisa used as reference image for the program.

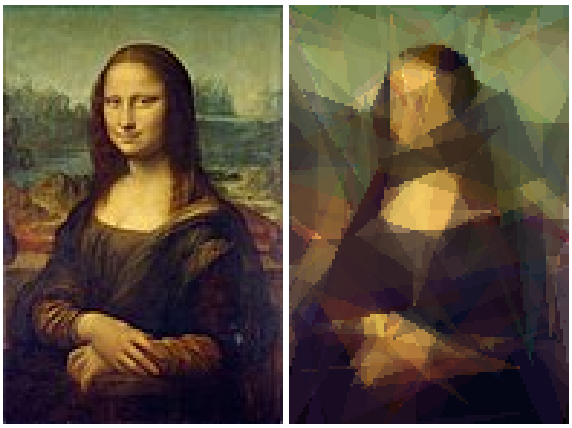
In the following figures we can observe the evolution for the first generations of the evolution process for the genetic algorithm. As it can be seen, in just a few generations an image that reminds to Mona Lisa can be obtained. From this point, it is a challenge to achieve a better resolution in the details of the image. It can also be appreciated that it is much more easy to find the right color (which is done is just a few generations), than finding the right shapes for the polygons.



(a) Generation 100 (b) Generation 300 (c) Generation 900 (d) Generation 2700

Figure 4.2: Mona Lisa comparison different stages of the genetic algorithm.

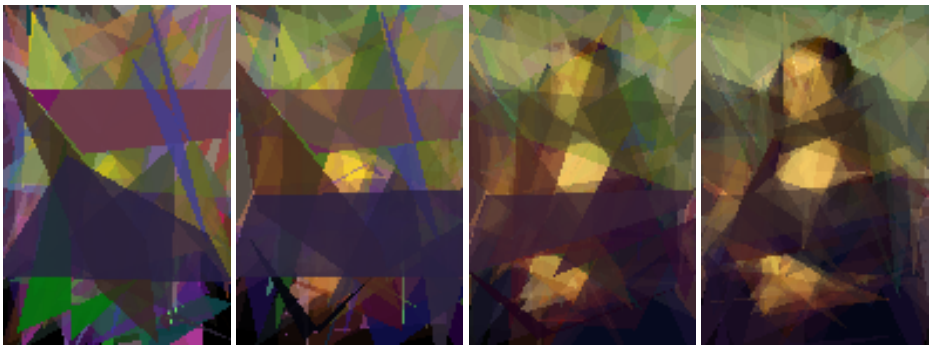
In the next two figures a comparison between the best image obtained with the genetic algorithm and the original image. We can observe that the image, despite being very precise in the colors used, it is not optimized in the way polygons are placed.



(a) Reference Image (b) Generation 27000

Figure 4.3: Comparison between the reference image and the final result of the genetic algorithm.

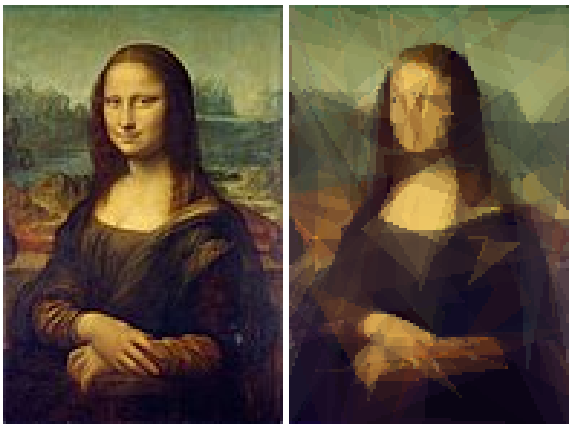
As it can be observed in the figures below, for the hill climbing algorithm, there is no big change between generations 1000 and 3000. It seems like this version needs more time to converge than the other. Despite of this, when it converges it seems that it finds a better solution.



(a) Generation 1000 (b) Generation 3000 (c) Generation 9000 (d) Generation 27000

Figure 4.4: Mona Lisa comparison different stages of the hill climbing algorithm.

In the figures below a comparison between the best approximation to Mona Lisa and the reference image can be observed. It seems like, in this case, a much better resolution is achieved, but it is still not able to achieve a good resolution in the face, which is the hardest part due to its details.



(a) Reference Image (b) Generation 550000

Figure 4.5: Comparison between the reference image and the final result of the genetic algorithm.

4.3 Fitness convergence

Studying the fitness convergence is important when studying the results of a genetic algorithm. It helps understanding the behaviour of the algorithm as well as it provides information about if the solutions are improving or not.

In the following picture a comparison between the genetic algorithm and the hill climbing algorithm is presented. The figure represents the average fitness for each generation. As it can be appreciated, the genetic algorithm converges faster in that sense because on each generations it is exploring more possible solutions than the hill climbing algorithm. Nevertheless, the hill climbing algorithm keeps getting closer to the genetic algorithm until it eventually improves the solution provided by it.

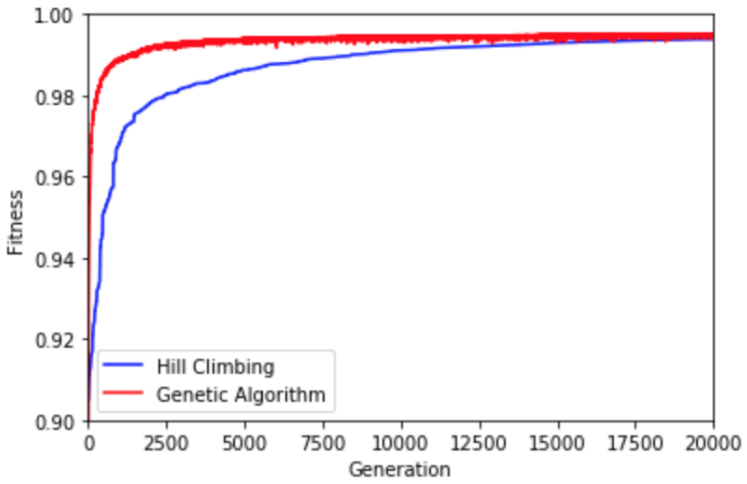


Figure 4.6: Comparison of the fitness by generation for the hill climbing and genetic algorithms.

The following picture provides another point of view. It represents the average fitness in terms of the computation time. As it was already pointed out, a generation for the genetic algorithm is 21 times slower than the hill climbing generation. For this reason, a comparison from that point of view must be done. As it can be observed, the hill climbing algorithm is the best one in that sense.

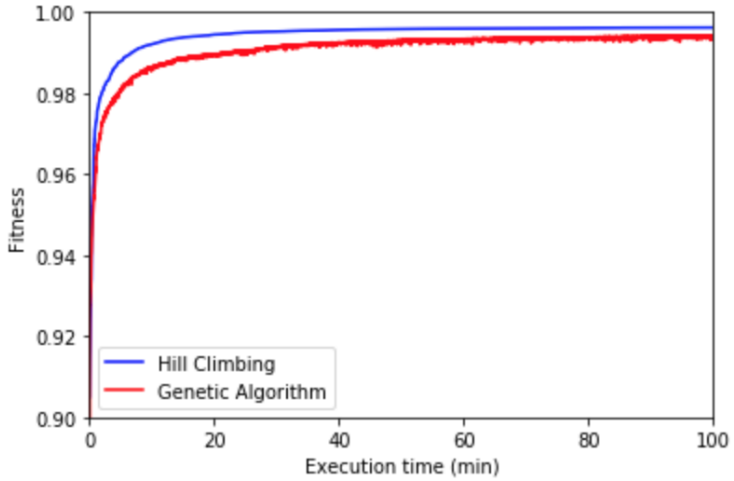


Figure 4.7: Comparison of the fitness by execution time for the hill climbing and genetic algorithms.

One last comment is that the fitness seem to converge to a value which is not exactly 1, which would be a exact replica of the Mona Lisa. It would be interesting to study how this fitness convergence value could be increased using different number of polygons, different number of vertices, or any other tweak on the algorithm parameters.

5 Conclusions

In this report, it has been presented two different algorithms to reproduce Mona Lisa with polygons. The first one was a genetic algorithm and, the second one, a hill climbing evolutionary algorithm.

It has been explained with detail the selection, crossover and mutation methods as well as the fitness calculation used in the algorithm. Many color distances have been tested, but not many differences were observed in terms of results accuracy.

The genetic algorithm was performing better when a generation by generation analysis was made, nevertheless, the hill climbing was performing faster generations which, in the end, it was finding a solution faster from that point of view. The hill climbing algorithm also achieved a solution which was more similar to the original Mona Lisa.

REFERENCES

[1] Genetic Programming: Evolution of Mona Lisa. https://rogerjohansson.blog/2008/12/07/genetic-programming-evolution-of-mona-lisa/.

[2] Genetic Algorithm - Wikipedia. https://en.wikipedia.org/wiki/Genetic_algorithm.

[3] Pillow. https://pillow.readthedocs.io/en/stable/.

[4] Color Difference - Wikipedia. https://en.wikipedia.org/wiki/Color_difference.

[5] I. Oliver, D. Smith, and J. R. Holland, "Study of permutation crossover operators on the traveling salesman problem," 1987.

[6] Alpha Compositioning - Wikipedia. https://en.wikipedia.org/wiki/Alpha_compositing.