This report is terrible! as I had too little time to write it, but gives some insights.

EOA Semestral Project - Image Compression

Libor Novak

January 9, 2017

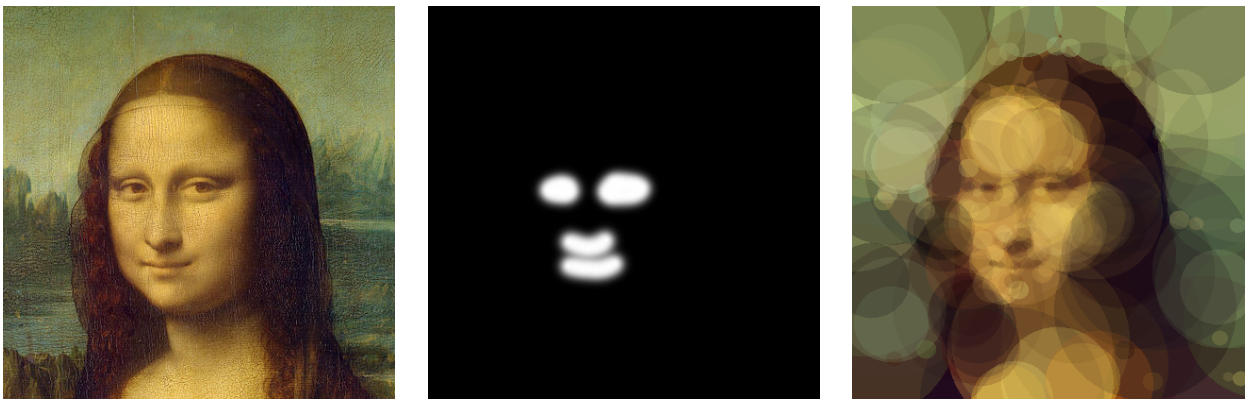
Abstract

This semestral project presents a solution to image compression. The goal is to suggest an optimization algorithm based on genetic algorithms, which minimizes the difference of the pixel colors of the prototype and the solution. The compressed images are represented as lists of basic geometric shapes, circles specifically. The programmed solution is easily extensible to support other types of shapes and compression algorithms (optimizers).

The code of the whole project is publicly available at `https://github.com/libornovax/eoa_image_compression`.

1 Introduction

The goal of image compression is to provide the human observer with the best image possible, while keeping restrictions on the data size of the image. The very straightforward way how to measure success of such algorithms is to simply compute the pixel-wise color difference between the original and the compressed image. However, the result of algorithms, which rely solely on this cost function was not satisfying for me because the result was lacking detail, which for me - as a human observer was crucial to recognize the displayed object. As described below, I decided to use extra human-defined weights on regions, which play a significant role for the human observer to identify the displayed object (see Fig. 1b).



(a) The 416×416 prototype image. (b) User-provided weights. (c) Best solution.

Figure 1: The prototype image used for testing the compression algorithm and the best approximation.

2 Algorithm Description

2.1 Representation - Chromosome

The compressed image is represented as a list of simple shapes - circles. The circles have RGB and α channels, position (x, y) and a radius. Their maximum radius is bounded to 1/4 of the image's smaller dimension. Also, the circle is required to intersect the image, which provides a bound on the position of the circle center when the radius is given. I used 200 circles throughout all my tests.

As the final representations tend to contain only large circles because they change the fitness function more than the small ones I created 3 groups of the shapes' sizes - SMALL, MEDIUM and LARGE. The radii of the circles are then forbidden to span out of their size group range.

2.2 Fitness functions

As given by the assignment, I used a fitness function, which computes the pixel-wise RGB differences between the prototype image and the approximation:

$$fitness = \sum_i \sum_{c \in \{R,G,B\}} (p_{ic}^O - p_{ic}^A)^2,$$

where i represents a pixel in an image and p_{ci}^* is the value of the c channel of the pixel i, O denotes the original (prototype) image and A denotes the approximation. However, this fitness function lacks description of image details, which are crucial for assessing the compression quality by a human observer. This inspired me to modify the fitness function in the following way: I ran Canny Edge Detector on the given image, filter the output by a Gaussian with a large σ and use the result as weights for the pixel-wise difference fitness function:

$$fitness = \sum_i \sum_{c \in \{R,G,B\}} w_i \cdot (p_{ic}^O - p_{ic}^A)^2,$$

where w_i is the weight of pixel i. This visually improved the result, but in the case of Mona Lisa (Fig. 1a) the image was still lacking detail in the mouth and eye region. The importance of these regions is however not possible to determine (fast) programatically. Therefore, I handcrafted a weight image (Fig. 1b), which I used in combination with the detected edges. This improved the visual result by a large margin.

It is also important to note that since the fitness is a difference between the two images the aim will be to minimize this fitness function.

2.3 Used optimization algorithms

I implemented three different optimization algorithms in total, which are listed below. I refer an interested reader to [1] for a detailed description of the algorithms. All of the algorithms use the same mutation operator - Sec. 2.3.1 to randomly perturb the chromosomes of the solutions. Genetic algorithms use also the crossover operator - Sec. 2.3.2, which I designed specifically for this task.

Steepest Ascent Hill Climber As the first optimization attempt I used the Steepest Ascent Hill Climb algorithm, which for this task worked really well. The reason is that the solution space contains many good local optima, which are very well found by the hill-climbing algorithm. I used pool size 10 from which the best solution was selected in each iteration. Inspiration was drawn from [2].

Steady-State Evolutionary Algorithm As the evolutionary algorithm I first tried to use the classic generational evolutionary algorithm, but problems with keeping diversity (many copies of almost exactly the same individual after only a couple of epochs) in the population made me change the replacement strategy to the steady-state replacement. In this case, each individual is replaced by its crossovered (with one another individual) and mutated version only if it has strictly better fitness than the parent. This results in exchanging small parts of chromosomes between individuals, but the diversity is kept because a large part of a chromosome can never be duplicated in the population.

Hybrid Evolutionary Algorithm Because the hill climbing optimization works really well for this task I decided to combine hill climbing epochs with the steady-state ones. The algorithm works in the following manner: For a given number of epochs I run the steady-state evolutionary algorithm, which provides the evolutionary algorithm with crossover in between the individuals. Then, the optimization switches to hill climbing and each individual is separately evolved for a given number of epochs. These two types of optimization are interleaving for the whole evolution period.

2.3.1 Mutation

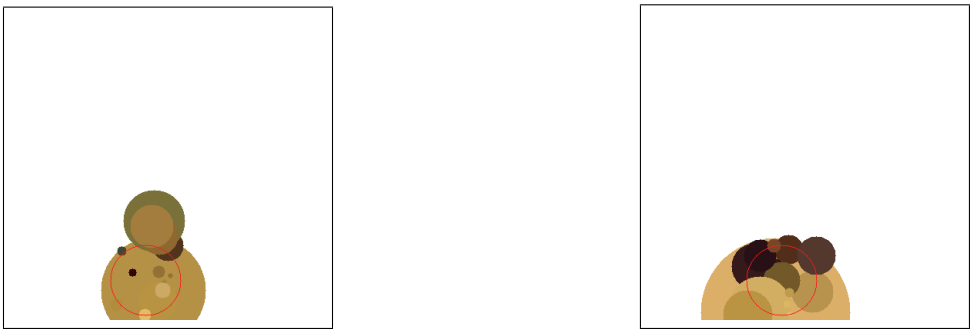
The same mutation operator was used in all optimizers. Each circle in the chromosome was mutated with a certain probability. Also, only one property of the circle (R, G, B, α, x, y, or radius) was mutated at a time to avoid excessive mutation. Each property also has its own mutation probability and a standard deviation of the mutation intensity. I also mutate the order of the circles in the chromosome - the earlier in the chromosome the circle is, the earlier it is rendered. These are sample parameters of the whole mutation.

```
# Probability of the whole shape being mutated at all
shape_mutation_prob: 0.05
# Probability of a shape to switch its position in the chromosome with another shape
shape_reorder_prob: 0.001

# Probability that a color (the alpha) channel will be mutated
color_mutation_prob: 0.1
alpha_mutation_prob: 0.1
# Standard deviation of the mutation for color and alpha channels
color_mutation_stddev: 10
alpha_mutation_stddev: 8

# Probability of a position (point) being mutated
position_mutation_prob: 0.1
# Standard deviation of the position mutation
position_mutation_stddev: 10
# Probability of a position (point) being completely randomly reinitialized
position_reinitialization_prob: 0.05

radius_mutation_prob: 0.1
radius_mutation_sdtddev: 5
```

2.3.2 Crossover

In this task we cannot just cut the chromosomes in random places because the circles in the chromosome are not ordered in any meaningful way, which would represent the solution and we

would be combining completely random circles, which together have no meaning to the final image. For this reason, I created a crossover operator, which is partly based on the phenotype of the individual. A random circle in the image is selected and all circles, which intersect it are exchanged, see Fig. 2. Therefore, the local properties from one individual can be transfered to another.



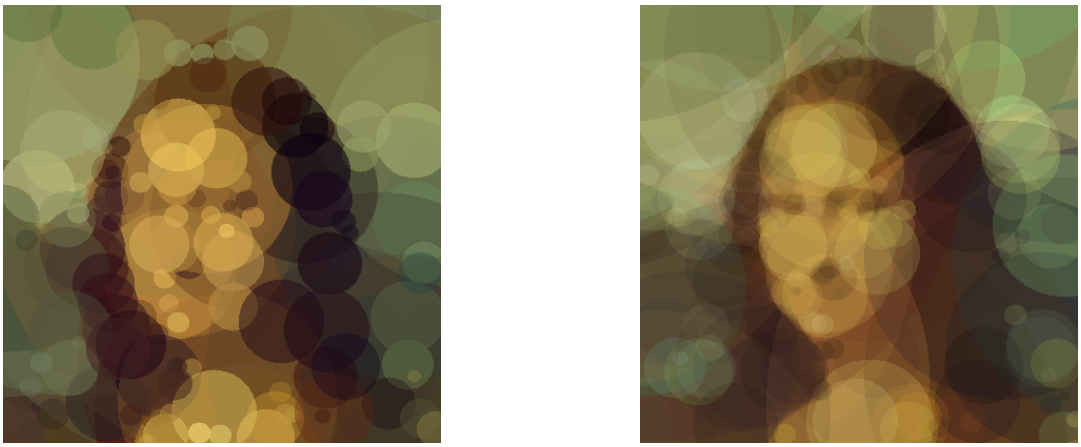
(a) Individual 1.

(b) Individual 2.

Figure 2: The crossover operation. All circles that intersect (or contain in case of LARGE circles) the "red" circle are exchanged from one individual to the other.

2.4 Rendering

A large problem of this task is that in order to evaluate each individual it has to be rendered to an image and then subtracted from the prototype. I programmed my own circle plotting functions because the ones in OpenCV were not suitable for plotting transparent circles. I experimented with two different types of rendering. One was plotting the circles one by one, layer by layer (see Fig. 3a). The other took all the circles, which intersect a pixel in the image and computed a weighted average over all of them - Fig. 3b. This results in more transparent circles. The space of available representations is much smaller because in this case the result does not depend on the order of the circles in the chromosome.



(a) Layer-by-layer rendering.

(b) Weighted averaging.

Figure 3: Different approaches to rendering of chromosomes - different phenotypes.

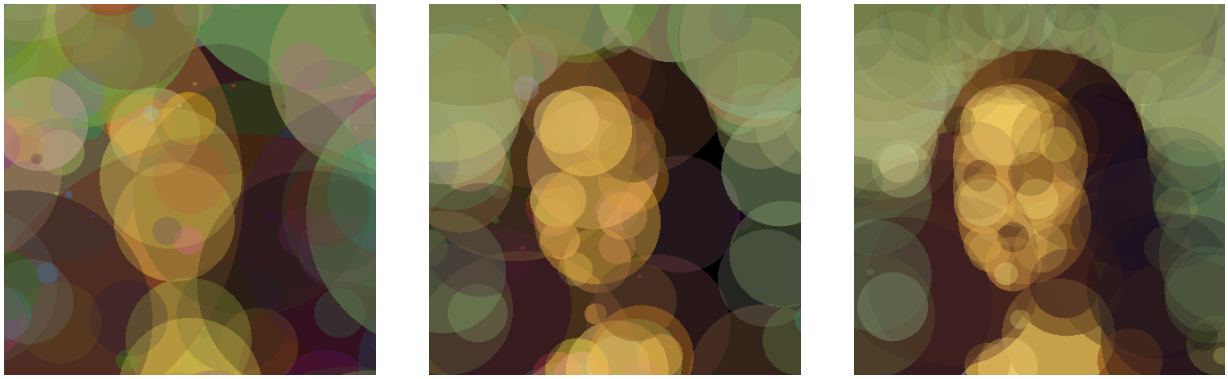
3 Discussion of Results

As this task is based on a visual outcome I evaluated results of many tests empirically. Therefore, exact numbers are not always available for the tests. Many improvements were also suggested to change the desired outcome of the optimization to better please the eye, therefore it also does not make so much sense to evaluate this task solely on the basic fitness function as the quality of the compression can be a subjective measure or even depend on the image that is being compressed.

3.1 Preliminary Results - Hill Climber

This section contains results of the very first tests of the hill climbing algorithm. The images in Fig. 4 are presented merely to demonstrate the evolution of the solutions from the very first attempts all the way to the best results. I do not provide settings nor fitness functions as the results can be very well rated empirically.

In all of the tests we can observe the lack of detail in the image. The "hair" part is very precisely rendered because there is a huge fitness loss on the edge. Also, the images bare a higher resemblance to the original image when viewed from a distance - when one cannot observe the separate circles, but the image "blends" together.



(a) First test, 50 circles, averaging, 1000000 iterations.

(b) 50 circles, layered rendering, 50000 iterations.

(c) 100 circles, averaging, 50000 iterations.

(d) 200 circles, 50000 iterations, layered rendering.

(e) 200 circles, 50000 iterations, averaging.

(f) 200 circles, 50000 iterations, averaging, restricted α, fitness weighted more on edges.

Figure 4: Results of the preliminary hill climbing algorithm.

4 Steepest Ascent Hill Climber

In this section I present the results of 3 different approaches on weighting the significant parts of the images in the fitness function. All approximations were created with 50000 iterations of the hill climbing algorithm with the following settings:

```
shape_mutation_prob: 0.05
shape_reorder_prob: 0.001
color_mutation_prob: 0.1
alpha_mutation_prob: 0.1
color_mutation_stddev: 10
alpha_mutation_stddev: 8
position_mutation_prob: 0.1
position_mutation_stddev: 10
position_reinitialization_prob: 0.05
radius_mutation_prob: 0.1
radius_mutation_sdtddev: 5
```

Fig. 5a shows the result of weight map, which was created by running Canny edge detector and Gaussian filtering of the edges. One can observe that the detail in the face improved when compared to images without fitness weighting, however the mouth, nose and eye part is still not rendered very well. Empirically we can see a person in the image, however it would be hard to determine that it is Mona Lisa.

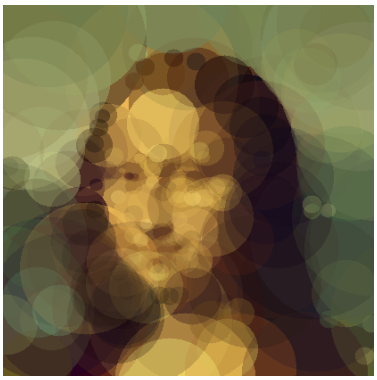
Fig. 5b and 5c show the run without the edge map. Instead the external user-provided weight map (Fig. 1b) was used. It is apparent that the mouth, nose and eye section detail improved vastly. We can even see the slight grin on Mona Lisa's face and both eyeballs. The problem now is that the detain in the hair section is a bit worse - we see a lot of imprecision in the circles in the top left of the face.

To approach these problems I combined the edge weights with the user-defined weights and created a combined weight map. The user-defined weights have the maximum of 1000, whereas the edge weights have the maximum of 100 in order to pronounce more the user-defined regions. The results in Fig. 5d and 5e show that I successfully kept a lot of detail on the user-defined regions, however the error on the problematic "hair" parts was suppressed.

Since I decided to use the combined weight map for all other tests I provide fitness evolution curves for the last two image approximations. In Fig. 6 we can see that after some 30000 iterations we are almost not improving on the fitness function. Time to generate 50000 epochs of the hill climber took approximately 40 minutes on a single CPU.

(a) Layered rendering, fitness weighted more on edges.

(b) Layered rendering, external weight map.

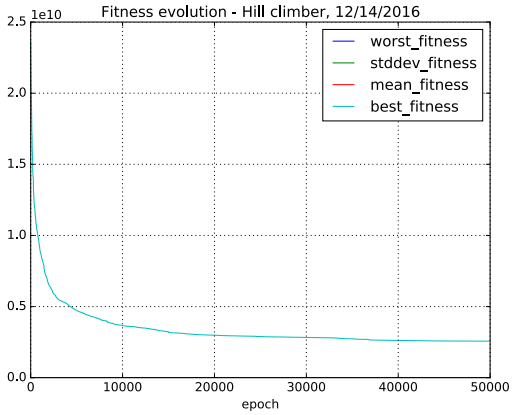


(c) Averaging, external weight map.

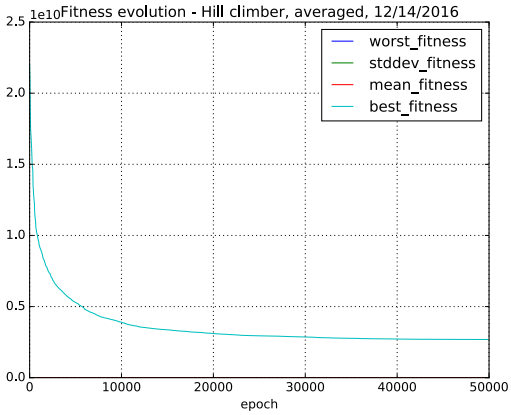
(d) Layered rendering, combined weight map. $fitness = 2.5694e{+}09$

(e) Averaging, combined weight map. $fitness = 2.68558e + 09$

Figure 5: Results of the preliminary hill climbing algorithm.



(a) Layered rendering. $fitness = 2.5694e + 09$



(b) Averaging. $fitness = 2.68558e + 09$

Figure 6: Fitness evolution of the best Hill-Climber runs with combined weight map.

4.1 Steady-State Evolutionary Algorithm

In the genetic algorithms I had a big problem of keeping the diversity of the population, however after the use of the steady-state replacement strategy this problem almost went away. To reduce the similarity of different solution in the population I introduced one more trick.

In the beginning of the optimization I randomly assign a region of interest to each individual (its area is 1/9 of the image), which in then weighted in the fitness function 10 times more than the rest of the image (Fig. 7). Because I use steady-state replacement I only replace an individual by a clone of the same individual, therefore these regions are kept unchanged throughout the evolution. After half of the epochs pases, the regions of interest are turned off and the evolution continues without them.

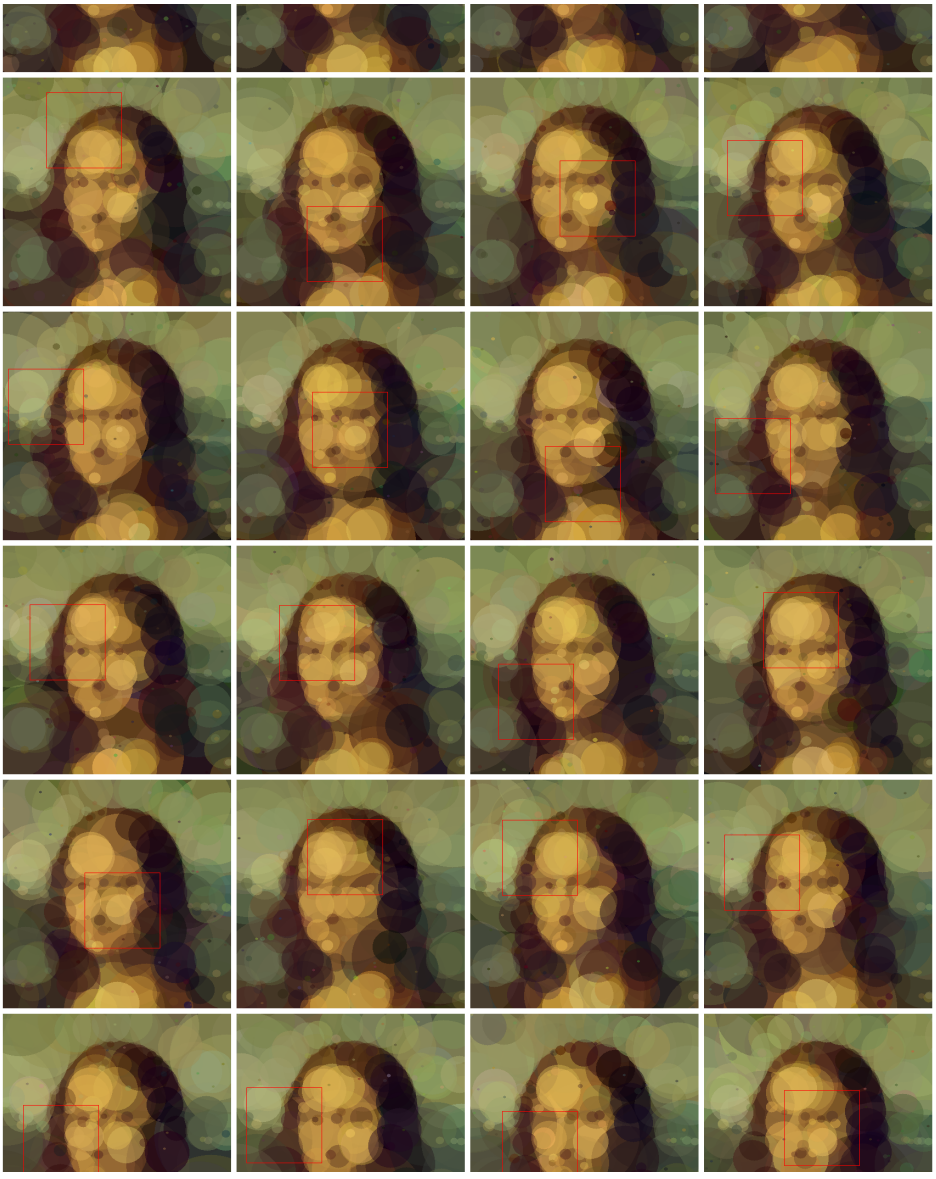


Figure 7: Regions of interest. Each image gets assigned a random region of interest, where the fitness is multiplied 10×. After a half of the iterations they are turned off.

After a bit of tweaking I ran a major test of the steady-state evolutionary algorithm. I ran it 3 times with exactly the same settings for 150000 epochs and the population of 225 individuals, tournament selection of size 2 and crossover probability 0.4. The parameters of the mutation were the following:

```
shape_mutation_prob: 0.1
shape_reorder_prob: 0.001
color_mutation_prob: 0.3
alpha_mutation_prob: 0.3
color_mutation_stddev: 10
alpha_mutation_stddev: 8
position_mutation_prob: 0.5
position_mutation_stddev: 10
position_reinitialization_prob: 0.1
radius_mutation_prob: 0.5
radius_mutation_sdtddev: 5
```

The best resulting image is in Fig. 8a. It is important to note that this test ran without the external weight file. I then ran another test with the external weight file with the same settings, but only 50000 epochs and the result is clearly visually superior as can be seen in Fig. 8b.

(a) Edge weighting only. 150000 epochs.

(b) Combined weights. 50000 epochs. $fitness = 3.02904e + 09$

Figure 8: Resulting images of the steady state evolutionary algorithm. Clearly, the image with combined weights looks superior.

In Fig. 9 we see exactly the moment, where the regions of interest were turned off as there is a significant step in fitness in the epoch 25000. In general the steady-state evolutionary algorithm did not perform so well and the evolution took about 100× longer than the hill climbing.

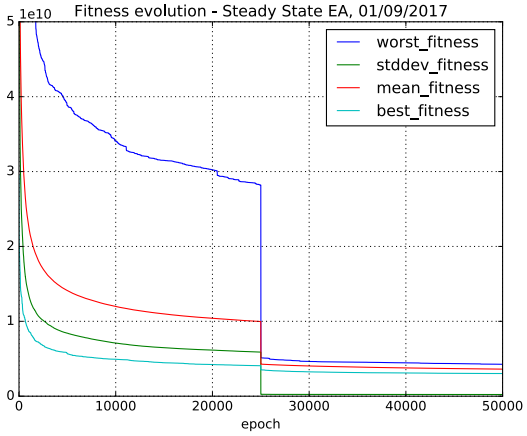


Figure 9: Fitness evolution of the steady-state evolutionary algorithm with regions of interest.

4.2 Hybrid (Interleaving) Evolutionary Algorithm

The final test that I ran on the hybrid evolutionary algorithm. I used the combined weight map. The evolution ran for 150000 epochs from which every 100th epoch was an epoch, when the hill-climbing algorithm ran. The hill climber had always just 25 iterations, pool size 10. Population size was 225, tournament selection of size 2 for the steady state epochs. Also, in the steady state epochs the probability of crossover was 100%. The mutation for the hill climber and the steady state evolution was the same:

```
shape_mutation_prob: 0.05
shape_reorder_prob: 0.001
color_mutation_prob: 0.1
alpha_mutation_prob: 0.1
color_mutation_stddev: 10
alpha_mutation_stddev: 8
position_mutation_prob: 0.1
position_mutation_stddev: 10
position_reinitialization_prob: 0.1
radius_mutation_prob: 0.1
radius_mutation_sdtddev: 5
```

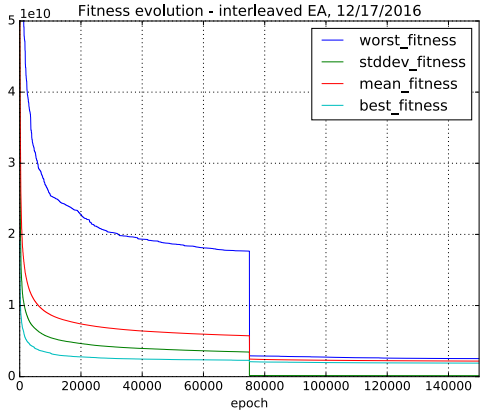
The resulting approximations are shown in Fig. 10. These approximations are superior to any other ones that I generated. They have very good detail in the mouth, nose and eye regions as well as the hair and even the background. The evolution of their fitness function can be seen in Fig. 11.

Also, it is important to say that even after 150000 epoch I was able to maintain population diversity. The whole population number 149900 is shown in Fig. 12. When zoomed in and carefully examined, one can see the differences in the genotypes (different positions, sizes and colors of the circles).

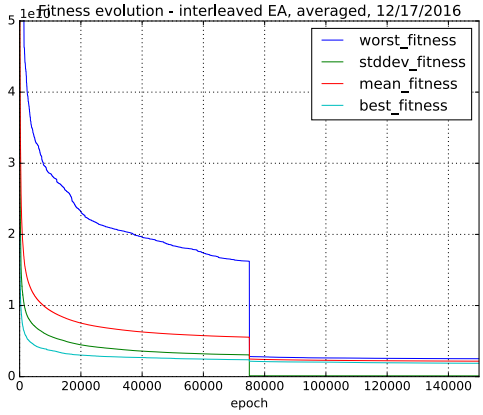
(a) Layered rendering. $fitness = 1.90332e + 09$

(b) Averaging. $fitness = 1.88273e + 09$

Figure 10: Results of the hybrid evolutionary algorithm.



(a) Layered rendering. $fitness = 1.90332e + 09$



(b) Averaging. $fitness = 1.88273e + 09$

Figure 11: Fitness evolution of the hybrid evolutionary algorithm.

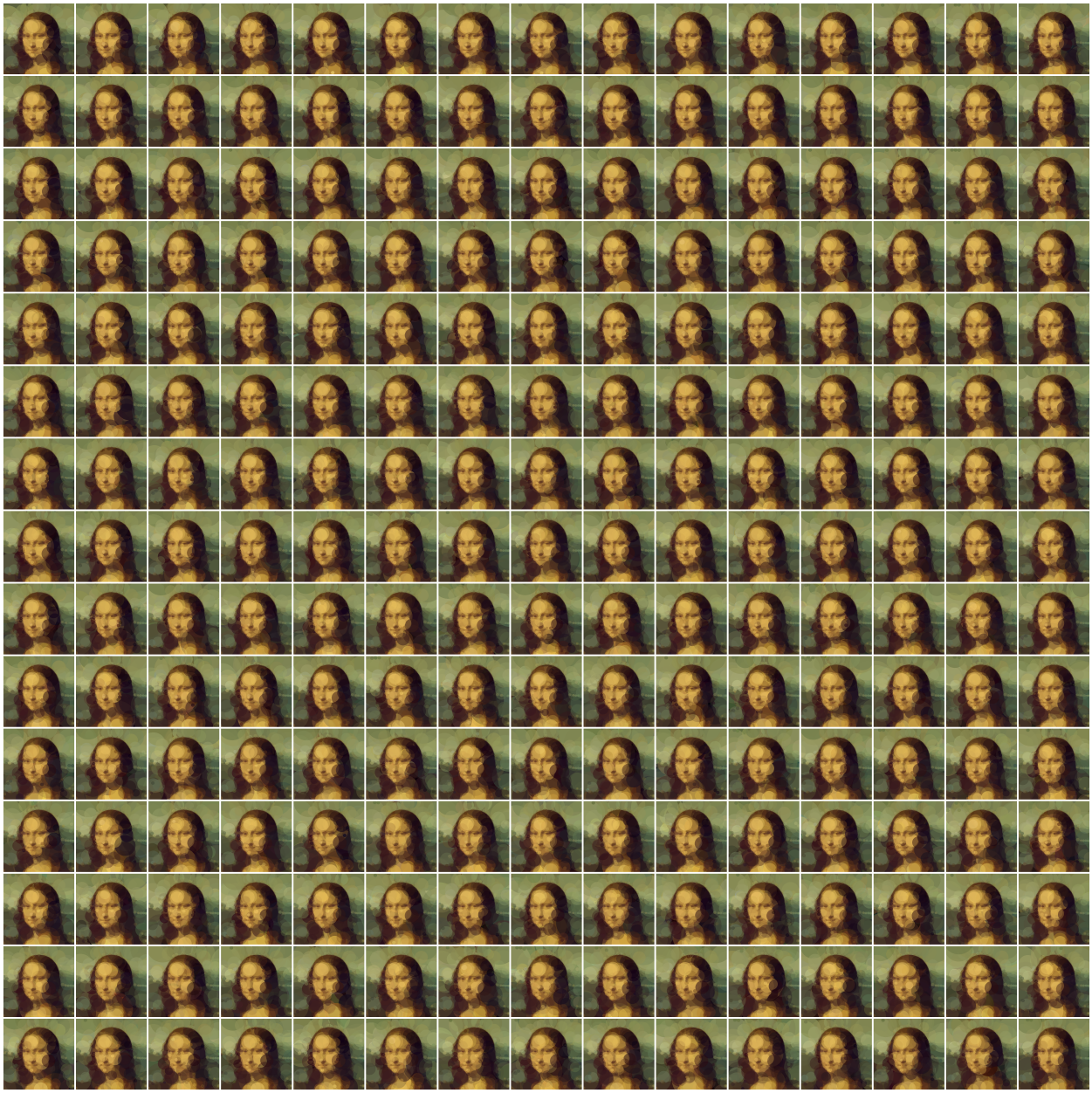


Figure 12: Whole population of 225 individuals in epoch 149900 of the hybrid evolutionary algorithm. Zoom in to see the details - differences between different individuals.

4.3 Comparison

When I compare the three implemented algorithms the winner would probably be the simplest one - hill climber (see Tab. 1). The reason is that it is about 100× faster than the evolutionary algorithms and provides only a slightly worse solution.

In this task, there are many good local optima, which are from the point of view of the fitness function equivalent. It is easy for the hill climbing algorithm to find such an optimum quickly. However, the search space is so big that the evolutionary algorithms struggle to find

global optima and instead get stuck in the same local optima. From the fitness point of view it is reasonable to use the hybrid version of the evolutionary algorithm because it converges faster due to the hill climbing capabilities and can search many candidate solutions at the same time and combine them in an intelligent way. The time price I paid for the hill climbing epochs was not so big because I implemented the hill climbing epoch to run in several threads.

	Hill Climber	Steady-State EA	Hybrid EA
Min. fitness	2.5694e+09	3.02904e+09	**1.88273e+09**
Time to solution	**1**	100	110

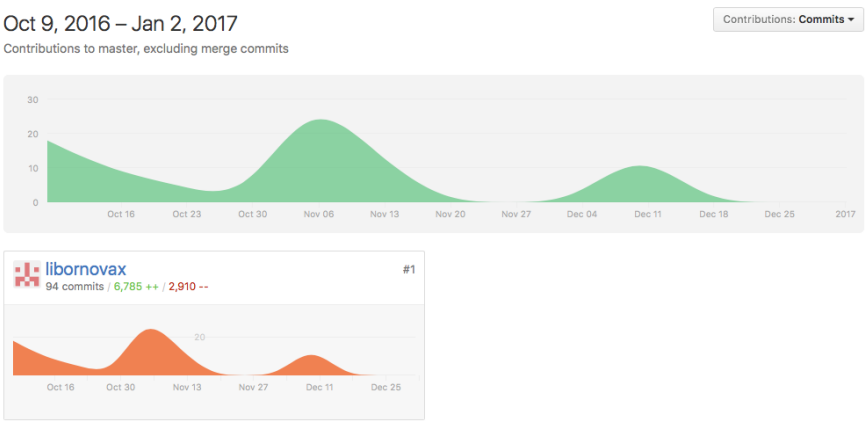
Table 1: Comparison of used optimization algorithms. Time is in relative units. Best achieved fitness with combined weights.

5 Conclusion

I created and tested several different optimization algorithms on the task of image compression. It was shown that for this particular task the hill climbing algorithm provides a sufficient and fast solution. Improvements to the given fitness functions were presented in order to improve the empirical evaluation of the results as the solution was lacking important details in the image. After the changes were implemented the result became very visually plausible and compares well to the original image.

Apart from implementing evolutionary algorithms I also learned ways to tweak them, how to fight population diversity, premature convergence and other assessment. Also I learned how to program a fast circle plotting algorithm and multithreading applications.

The code of the whole project is publicly available and can be found on my Github account `https://github.com/libornovax/eoa_image_compression`.



References

[1] S. Luke, *Essentials of metaheuristics*. Lulu Com, 2013.

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