

# Evaluation of a genetic algorithm based image approximation method

Gergely Vakulya

Alba Regia Technical Faculty

Óbuda University

Székesfehérvár, Hungary

*vakulya.gergely@amk.uni-obuda.hu*

**Abstract**—In this paper a genetic algorithm (GA) based image approximation algorithm is presented for black and white images. The approximation uses a limited amount of triangles, where the GA is used to optimize their shape and position. The formal definitions, implementation details and test results are provided. The algorithm can produce recognizable approximations in less than 139 bytes.

**Keywords**—genetic algorithm, image approximation

## I. INTRODUCTION

Genetic algorithms (GA) [1] offer good approximations to various problems, where no fast algorithm is known to calculate the optimal solutions. It models the natural selection, where each member of a population (called specimen) represents one possible solution.

GA starts with an initial population, which is a set of possible solutions (e.g. empty or randomly generated). In each cycle GA produces new specimens by applying *mutation* and *crossover* operations on the existing ones. Then the population is evaluated by the *selection* operator based on the *fitness* function.

In this paper a GA-based lossy image compression method is presented. A number of triangles are used to approximate a monochromatic image; the position of the triangles are optimized with GA. The chromosome representation, the mutation and crossover operators and different selection strategies are evaluated. The algorithm is capable to give recognizable approximation of images in the 100 byte range, which means an extreme compression in exchange for the lower quality output.

## II. RELATED WORK

Genetic algorithms are suitable to problems, where there is no efficient solution available to calculate the optimal (or, in many cases, any) solution. An advantage of GA is the iterative operation, making it possible to stop the optimization process in any time and use the current best solution.

Scheduling is an everyday problem in industry [6] and transportation planning [3], where genetic algorithms can offer a good alternative to e.g. linear programming.

A similar area covers the Vehicle Routing Problems (VRP). A practical genetic algorithm based solution can be found in [4]. It is closely related to the traveling salesman problem, which can also be approximated with GA [5].

A good application of genetic algorithm is calculating the synaptic weights of neural networks. MarIO [2] is an AI player to the Super Mario game.

Genetic algorithms are also suitable to generate program code (genetic programming) to determine the behavior of e.g. autonomous robots. A good example is [7], where a specially tailored programming language (Robocode) is used to program virtual battle robots.

In economics portfolio optimization [8] and automatic trading [9] can be implemented with genetic algorithms.

In the area of image processing GA can be efficiently used in combination with neural networks for e.g. image segmentation [10] or image enhancement [12]. Image compression tends to use fast algorithms to give an indistinguishable (lossless compression) or an indistinguishable (lossy compression) representation of the input image, in significantly less space [11].

## III. THE GA-BASED IMAGE COMPRESSION

### A. Problem statement

An image *IMG* is given as square ( $n \times n$ ) matrix, where each pixel is either black or white:

$$IMG = (a_{ij}) \in \{0, 1\}^{n \times n} \quad (1)$$

An approximation is given as a set of  $n$  triangles. The vertices of the triangles are placed in the target image, in integer positions, and can have either black or white color. The  $k^{\text{th}}$  triangle  $t_k$  can be given as a quad:

$$t_k = (A, B, C, c), A, B, C \in 1 \dots n^2, c \in \{0, 1\} \quad (2)$$

Time  $I(t_k)$  image of triangle  $t_k$  is a set of points, which are geometrically between the vertices:

$$\begin{aligned} I(t_k) &= a_{ij}, \\ \exists \alpha, \beta &\in [0, 1] : \\ i &= \beta(\alpha A_1 + (1 - \alpha)B_1) + (1 - \beta)C_1 \\ j &= \beta(\alpha A_2 + (1 - \alpha)B_2) + (1 - \beta)C_2 \end{aligned} \quad (3)$$

### B. Chromosome representation

The chromosome representation of an approximation  $Q$  is a set of triangles, represented by a triplet  $(k, T, C)$ , where

- $k \in \mathbb{Z}$  is the number of triangles used in the approximation,
- $T \in \mathbb{Z}^{k \times 3 \times 2}$  is the set of triangles, given with 3 coordinate pairs,
- $C \in 0, 1^k$  is the set of triangle colors.

### C. Fitness function

The image form  $R$  of an approximation  $(k, T, C)$  is defined as follows:

$$R_{i,j}(k, T, C) = \begin{cases} C_v, & \exists v : (i, j) \in I(T_v), \\ & \nexists w : w > v, (i, j) \in I(T_w), \\ & 1 \leq v, w \leq k \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Instead of the fitness function, where the more suitable specimen have larger fitness, an  $E$  error function is used, representing the proportion of the mismatching area between the target image and the triangle based approximation, as a percentage.

$$E = \frac{\sum_{i,j} \left( \frac{|IMG_{i,j} - R_{i,j}|}{mn} \right)}{n^2} \cdot 100 \quad (5)$$

$1 \leq i, j \leq n$

### D. Genetic operators

The three genetic operators (mutation, crossover and selection) are implemented as follows. Three simple mutation types was built into the program.

- $M1$  removes a randomly selected triangle
- $M2$  adds a triangle with random coordinates
- $M3$  modifies the position of 1, 2 or 3 vertices of a triangle

The crossover operator takes triangles from two selected triangle-set and takes triangles from both, randomly.

During the selection the fitness function is calculated and a defined amount of the population is eliminated and overwritten by newly generated specimen (by crossover).

### E. Program flow

The flow of the program goes through the following steps.

Step 1. Loading the reference image.

Step 2. Generating the initial population, where each specimen consist of one randomly generated triangle.

Step 3. Applying *mutation* operators for each specimen with predefined probabilities.

Step 4. Calculating the *error* for each specimen.

Step 5. Writing out statistical information.

Step 6. Applying the *selection* operator, i.e. eliminating the specimen with the highest error.

Step 7. Generating new specimen with *crossover* in place of the eliminated ones.

Step 8. Repeating the procedure from Step 3 until achieving the desired number of generations.

### F. Software architecture

The image approximation algorithm is implemented in C, without using external libraries (besides glibc). This approach makes the code easily portable, but on the other hand it makes handling of different image formats more difficult. Thus, only loading and saving of BMP files with predefined resolution and color depth are supported.

The test program prints a short information line after each generation, which contains the generation number, the error of the best matching approximation and the current number of polygons. The best matching image is saved after each generation, to give more visual feedback.

With the actual settings each approximated image takes

- 8 bits to store the number of polygons,
  - $3 \times 2 \times 9 \times k$  bits to store the triangle coordinates,
  - and  $k$  bits to store the colors of the triangles,
- sum up to  $1 + \frac{55}{8}k$  bytes.

## IV. TEST RESULTS

To test the performance of the proposed image approximation method, two moderately complex test images was selected, as shown in Fig. 1. Both images have a resolution of  $512 \times 512$  pixels.

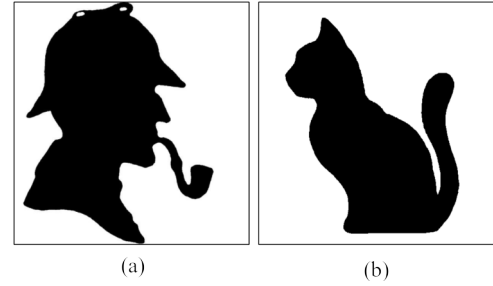


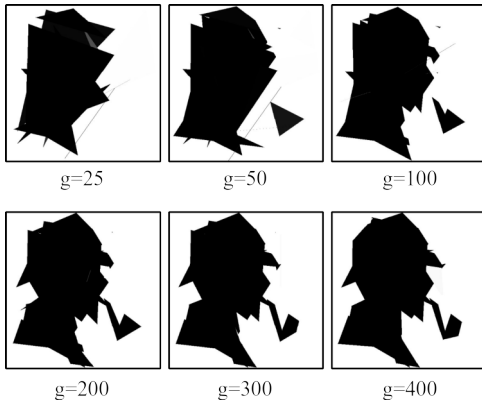
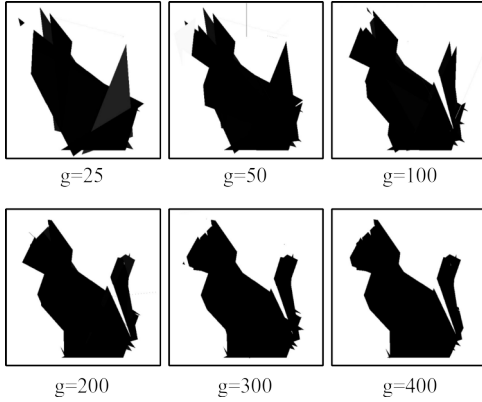
Fig. 1. The selected test images. (a) Silhouette of Sherlock Holmes, (b) Silhouette of a cat.

The approximation algorithm was tested with the selected images with the following settings. The size of the population is 2048. In each generation each specimen is mutated with the  $M1$ ,  $M2$  and  $M3$  operators with 10%, 50% and 80% probability, respectively. The 1800 worst specimen was overwritten with the crossover of randomly selected ones. Note that this is an elitist setup, which can lead to a fast convergence, at a risk of a less accurate final result.

The polygon limit was set to  $k_{max} = 5, 10, 15, 20, 30, 40, 50, 60, 70$  and  $80$ . Each different setup was run through 1000 generations.

Figures 2 and 3 show the image with the best fitness after 25, 50, 100, 200, 300 and 500 generations with  $k_{max} = 20$  polygons. As can be clearly seen, the accuracy of the approximation quickly increases first; both images are recognizable after 200 generations, but only slight improvements can be observed afterwards.

Figures 4 and 5 show the change of best observed fitness after each generation for the Sherlock and the cat images. Different colors mean different maximal triangle numbers. With

Fig. 2. The best matching Sherlock image after  $g$  generations.Fig. 3. The best matching cat image after  $g$  generations.

extremely low polygon count the approximation remains very rough. Adding more than approx. 30 triangles does not make any quantitative improvement, some extra detail was observed up to 60 triangles. A recognizable image approximated with 20 triangles takes 139 bytes of data.

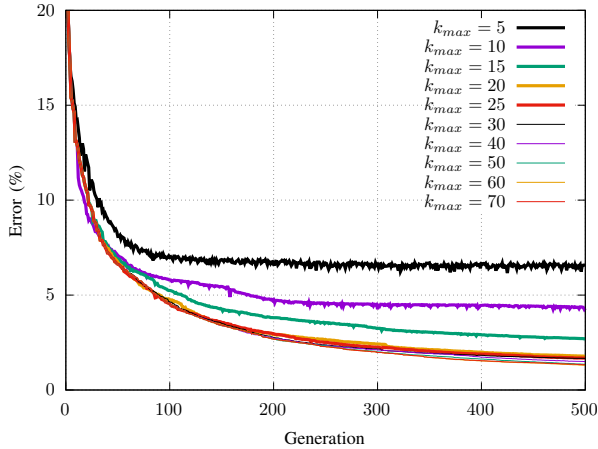


Fig. 4. Change of the fitness function during the approximation of the Sherlock image. The smallest error after each generation is shown for different polygon limits.

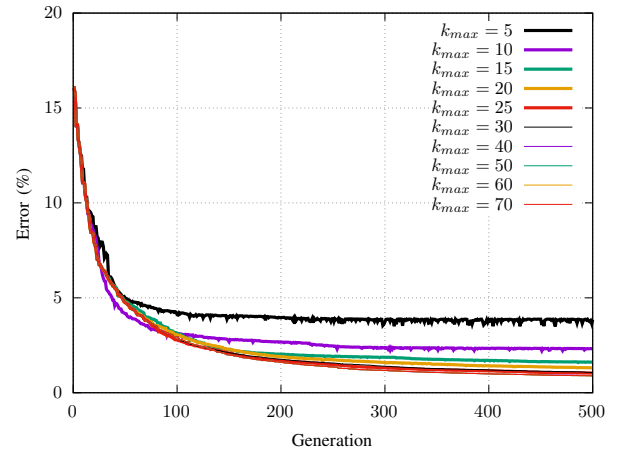


Fig. 5. Change of the fitness function during the approximation of the cat image. The smallest error after each generation is shown for different polygon limits.

TABLE I  
THE ERROR AFTER 1000 GENERATIONS WITH DIFFERENT  $k_{max}$  VALUES

| Image name | $k_{max}$ | Final error (%) | Size (bytes) |
|------------|-----------|-----------------|--------------|
| Sherlock   | 5         | 6.338           | 36           |
|            | 10        | 4.218           | 70           |
|            | 15        | 2.401           | 105          |
|            | 20        | 1.439           | 139          |
|            | 25        | 1.220           | 173          |
|            | 30        | 1.301           | 207          |
|            | 40        | 1.123           | 276          |
|            | 50        | 1.006           | 345          |
|            | 60        | 0.940           | 414          |
|            | 70        | 0.898           | 483          |
| Cat        | 5         | 3.775           | 36           |
|            | 10        | 2.260           | 70           |
|            | 15        | 1.320           | 105          |
|            | 20        | 1.199           | 139          |
|            | 25        | 0.837           | 173          |
|            | 30        | 0.807           | 207          |
|            | 40        | 0.532           | 276          |
|            | 50        | 0.612           | 345          |
|            | 60        | 0.569           | 414          |
|            | 70        | 0.562           | 483          |

## V. COMPARISON TO COMPRESSION ALGORITHMS

Images can be compressed with (lossless and lossy) image compression algorithms and with general data compression software as well. To provide a comparison in term of size the test images were compressed with the two most well-known image compression methods (PNG and JPEG) and two general data compression solutions (rar and bzip2).

The compression level of PNG was set to the highest value and JPEG is tuned to worst quality (smallest file size) setting. Storing of extra data (i.e. EXIF, colorspace information) was turned off for both formats. Rar was used with `-m5 -md4096` settings, and the `-9` switch was used with bzip2. Both settings means the best possible compression. The proposed approximation algorithm was used with  $k_{max} = 20$  and  $k_{max} = 50$ . The results are summarized in Table II.

TABLE II

THE RESULTING DATA AMOUNTS WITH IMAGE AND GENERAL DATA COMPRESSION ALGORITHMS AND WITH THE PROPOSED METHOD.

| Compression method    | Size (bytes) |      |
|-----------------------|--------------|------|
|                       | Sherlock     | cat  |
| JPEG                  | 2443         | 2521 |
| PNG                   | 2688         | 2247 |
| rar                   | 2059         | 1553 |
| bzip2                 | 1859         | 1690 |
| GA ( $k_{max} = 20$ ) | 139          | 139  |
| GA ( $k_{max} = 50$ ) | 345          | 345  |

## VI. SUMMARY

A genetic algorithm based image approximation method was proposed. The algorithm can approximate black and white images with a limited (definable) number of black and white triangles. The GA is implemented with 3 mutation and a crossover operator. Selection operator uses an elitist approach. Formal definition was given to the fitness function computation.

The proposed algorithm was implemented as a C language program and tested with different input images and with several maximal number of triangles.

The algorithm can generate a recognizable quality approximation of images with moderate complexity in less than 150 bytes with computation time under 10 minutes.

## REFERENCES

- [1] Sivanandam, S. N. and Deepa, S. N., "Introduction to Genetic Algorithms", Springer-Verlag Berlin Heidelberg, 2008, ISBN 978-3-540-73189-4
- [2] Baldominos A., Saez Y., Recio G., Calle J. "Learning Levels of Mario AI Using Genetic Algorithms", In: Advances in Artificial Intelligence. CAEPIA 2015. Lecture Notes in Computer Science, vol 9422. Springer, Cham. doi: 10.1007/978-3-319-24598-0\_24
- [3] C. Lin, J. Yu, J. Liu and C. Lee, "Genetic Algorithm for Shortest Driving Time in Intelligent Transportation Systems", 2008 International Conference on Multimedia and Ubiquitous Engineering, 2008, pp. 402-406, doi: 10.1109/MUE.2008.16.
- [4] Zhu, K. Q., "A new genetic algorithm for VRPTW", Proceedings of the international conference on artificial intelligence. 2000.
- [5] Ahmed, Z. H. "Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator", International Journal of Biometrics & Bioinformatics (IJBB) 3.6 (2010): 96.
- [6] Lim, C., and Eoksu S., "Production planning in manufacturing/remanufacturing environment using genetic algorithm", Proceedings of the 7th annual conference on Genetic and evolutionary computation. 2005.
- [7] Shichel Y., Ziserman E., Sipper M., "Using Genetic Programming to Evolve Robocode Players" In: Genetic Programming. EuroGP 2005. Lecture Notes in Computer Science, vol 3447. Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-540-31989-4\\_13](https://doi.org/10.1007/978-3-540-31989-4_13)
- [8] S. Slimane and M. Benbouziane, "Portfolio Selection Using Genetic Algorithm" In: Journal of Applied Finance & Banking , Vol. 2, No. 4 (2012): pp. 143-154.
- [9] R.J. Kuo, C.H. Chen, Y.C. Hwang, "An intelligent stock trading decision support system through integration of genetic algorithm based fuzzy neural network and artificial neural network", Fuzzy Sets and Systems, Volume 118, Issue 1, 2001, Pages 21-45, ISSN 0165-0114, 10.1016/S0165-0114(98)00399-6.
- [10] Pare, S., Bhandari, A. K., Kumar, A., Singh, G. K., Khare, S. "Satellite image segmentation based on different objective functions using genetic algorithm: a comparative study." In 2015 IEEE international conference on digital signal processing (DSP) (pp. 730-734). IEEE.
- [11] Jayaraman, S., S. Esakkirajan, and T. Veerakumar. "Digital image processing", Mc Graw Hill India, 2009, ISBN 978-0070144798
- [12] Deborah, H., Arymurthy, A. M., "Image enhancement and image restoration for old document image using genetic algorithm", In 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies (pp. 108-112). IEEE.