Universiteit Leiden
The Netherlands

Opleiding Informatica

Automating the process of defining the aesthetic quality of an image in a numerical way

Luuk van den Nouweland

Supervisors:
Dr. B. van Stein
Dr. A.V. Kononova
D.L. Vermetten

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

September 20, 2021

Abstract

The goal of this research is to automate the process of determining the aesthetic quality of an image numerically. Subsequently, this numerical value will be used in an unsupervised evolutionary algorithm to generate visually pleasing images. To achieve this, an objective function has to be created that is able to score images based on their qualities. These images are generated using a dataset of elements that are extracted from ornaments. Each image is represented in a tree-like structure, which the algorithm uses to score each image. The objective function looks at seven different aspects of a generated image. Several experiments are performed to see how scoring images based on these aspects influence the end results. From these experiments, the conclusion can be drawn that the symmetry detection function used in the objective function is not performing as desired. Therefore, the results are below expectations. More experimentation needs to be done with different symmetry heuristics.

Contents

1 Introduction

An Evolutionary Algorithm (EA) is an algorithm based on Darwin's evolution theory [5] [11]. The goal of an EA is to simulate a population of individuals whom all try to survive to create offspring. However, only the strongest individual will be able to do this. This is also known as survival of the fittest. Since only the strongest individuals survive in each generation, gradually the population will be more capable of surviving.
In this research, we want to find out if it is possible to generate aesthetically pleasing images using an EA. There are two types of EA's that can be used to achieve this goal. These are called supervised and unsupervised algorithms. A supervised algorithm makes use of a human as the objective function. This means that a person scores each image generated by the EA to slowly get a better result. The end result will depend on the person's values for aesthetic and unaesthetic images. This is both an advantage as well as a disadvantage of a supervised algorithm. The advantage is that the algorithm can be controlled very precisely. However, the disadvantage is that the evolutionary process will be subjective. Another disadvantage is the size of the experiments performed. Since each image has to be manually evaluated, the algorithm is limited in the number of images per generation and also in the total number of generations.
The goal of this research is to create an EA that makes use of an unsupervised evolutionary algorithm. In an unsupervised algorithm, no people are involved. Therefore, the process of generating images will be fully automated. To accomplish this, an objective function needs to be created which can differentiate between images with a "low" and "high" aesthetic quality.
To generate images, a dataset of "smaller images" is provided by Ornamika; [17]. In this dataset, images are deconstructed into smaller segments. These segments can be used to generate new images. For example, an image that consists of three segments; a triangle, a circle and a square, will be divided into three separate segments in the dataset. The EA will use this dataset when generating images. The objective function will also be specifically designed to take these types of images into account.
For this paper there are two questions we want to answer:

- Is it possible for a computer to assess the aesthetic quality of an image in a numerical way?
- Is it possible to use the previously mentioned numerical value to create an evolutionary algorithm that generates images?

1.1 Thesis overview

The thesis is structured as follows. Section 2 contains the related work. Section 3 talks about the implementation of the objective function and the evolutionary algorithm. Section 4 describes how and why the experiments are performed and what the results of the experiments are. Section 5 will discuss the results from the experiments. Last, in section 6 the paper is concluded and further research will be discussed.

2 Related work

Richard Dawkins is the first person to generate shapes using an evolutionary algorithm [7]. To support his arguments in the book and show the process of evolution, he has created a program that draws 2-dimensional "biomorphs". These biomorphs consist of straight black lines with a variable length, angle and position. A user can select which biomorphs are allowed to create offspring. Next random mutation would occur. Then, from the mutated offspring, the user can select which one will be the basis for the new generation. This way the user can influence the evolutionary process of the biomorphs. Another early example of an evolutionary algorithm used to generate structures and textures is by Karl Sims [18]. In his research, he gives several examples of his implementation. For instance, 3-dimensional plant structures are generated using fractal limits. Also, textures are generated by mutating symbolic expressions. Equations that calculate the color value for each pixel are evolved using a specific set of rules.

The concept of interactively generating images using an algorithm led to the creation of evolutionary art (EvoArt). One of the big advantages (and disadvantages) of interactively evolving images is the fact that humans control the evolution process. However, as mentioned previously, user fatigue is a big limitation. Therefore, research has been done to automate the selection process. For example, Research by DiPaola and Gabora discusses "how computer generated art can become more creatively human-like with respect to both process and outcome" [8]. To achieve this, an automatic fitness function is created to evolve abstract images that look like Darwin.

However, automating a fitness function to score the aesthetics of an image is not an easy task. Art is very subjective. Trying to differentiate between "good" and "bad" art is already complex. And even if such a distinction can be found, telling a computer to use the found distinction to score images isn't easy either. Al-Rifaie et al. [1] discusses this problem. In it, symmetry is discussed and its significance in the aesthetics of an image. Also, a swarm intelligence technique is introduced to detect symmetry. The authors believe that there still is a long way to go before human appreciation can be translated into a computational model. The research did show promising results, however.

Another possibility in the automation process of scoring aesthetically pleasing images is to let the computer learn the difference between good and bad, instead of defining the differentiation yourself. For example, Ciesielski et al. [3] uses machine learning on two image databases (one containing photos, the other containing abstract images) rated by humans to learn what features of images are associated with a high aesthetic value. There are several more examples where the computer tries to learn the users preference to automate the image generation process [14], [9], [2], [15]. But to accomplish this, a database of human-rated images is necessary.

A research paper that has much overlap with this paper is called "Studying Aesthetics in Photographic Images Using a Computational Approach" by Datta et al. [6]. In this paper, machine learning is used to automatically infer the aesthetic quality of photographic pictures. Both a classification and a regression model are created. To train this network, an online peer-rated photographic database is used. Several features are extracted from the images to train the models on. Similarly, our evolutionary algorithm will also look at different features of the generated images to determine their aesthetic quality. The biggest difference between these two papers is the type of images that are used in the algorithms. Very different features need to be judged in photographic images compared to the abstract images used in this research. Also, the machine learning model is being trained on a dataset, while the evolutionary algorithm is not. This paper concludes that a

significant correlation can be found between the visual properties of a photograph and its aesthetic ratings.
As has already been said, art is very subjective. Therefore, the question if it is even possible to objectively assign the aesthetic quality numerically is very valid. Datta et al. [6] compare a classification model against a regression model. They claim that since the measurements are highly subjective, absolute scores are less meaningful. For classification, they only have to differentiate between a "low" and "high" aesthetic quality and not between absolute scores. Li et al. claim that personality is highly related to personal preference [13]. Therefore, a personality-assisted multi task deep learning framework is created. Cui et al. [4] regard the users' behavior as "the reflection of their perception of images and harness these additional clues to improve image aesthetics assessment." Many examples are available of researchers trying to objectively score images based on user preference. In our research, personal preference is not taken into account. For the objective function, personal preference is generalized to calculate aesthetic quality and create a baseline. The images are not generated based on user preference.

3 Implementation

In this section, a summary will be given on how the algorithm, including the objective function, is implemented and how an image is represented.

3.1 Structure of an image

In this research, we do not want to generate images by modifying the values of the pixels themselves. Instead, we have a database of sub-images[1] provided by Ornamika [17]. Hence, the images generated by the EA will consist of these sub-images. A representation for these type of images will need to be constructed so that the EA will be able to modify them. This representation will also be used by the objective function to assess the quality of the image.
The decision has been made to represent the images in a tree-like structure. This structure can be found in figure 1. In the figure, it can be seen that one image consists of a list of units, and one unit consists of a list of elements. An element is an object which stores all the information about a specific artwork inside a unit. (e.g. (relative) position, rotation, size, which sub-image is used, etc...). A unit is a grouping of elements. In the unit information is stored about which elements are part of the unit and where this unit is located in the image. All the information of each element is stored relative to the information of the unit. For example, the position of an element is stored relative to the position of the unit. This way, the image can be easily modified. In figure 1, circles are the objects that store information and rectangles are variables that are stored in the objects. These variables can be modified by the algorithm.
An example of the representation of an image will be given using figure 2c. The list of units currently consists of five objects. Four units are copies of each other. The fifth one is located in the middle. The middle unit will be used to help clarify the construction of a unit. This unit consists of five elements. Four repeating elements and one unique element of a horse. Since this unit is positioned in the middle, the relative position will be $(x : 50\%, y : 50\%)$. The relative width of the unit is also 50 percent. However, the elements inside the unit do not use the entire width that is provided. The

[1] A sub-image is an image that has been extracted from an artwork [17].

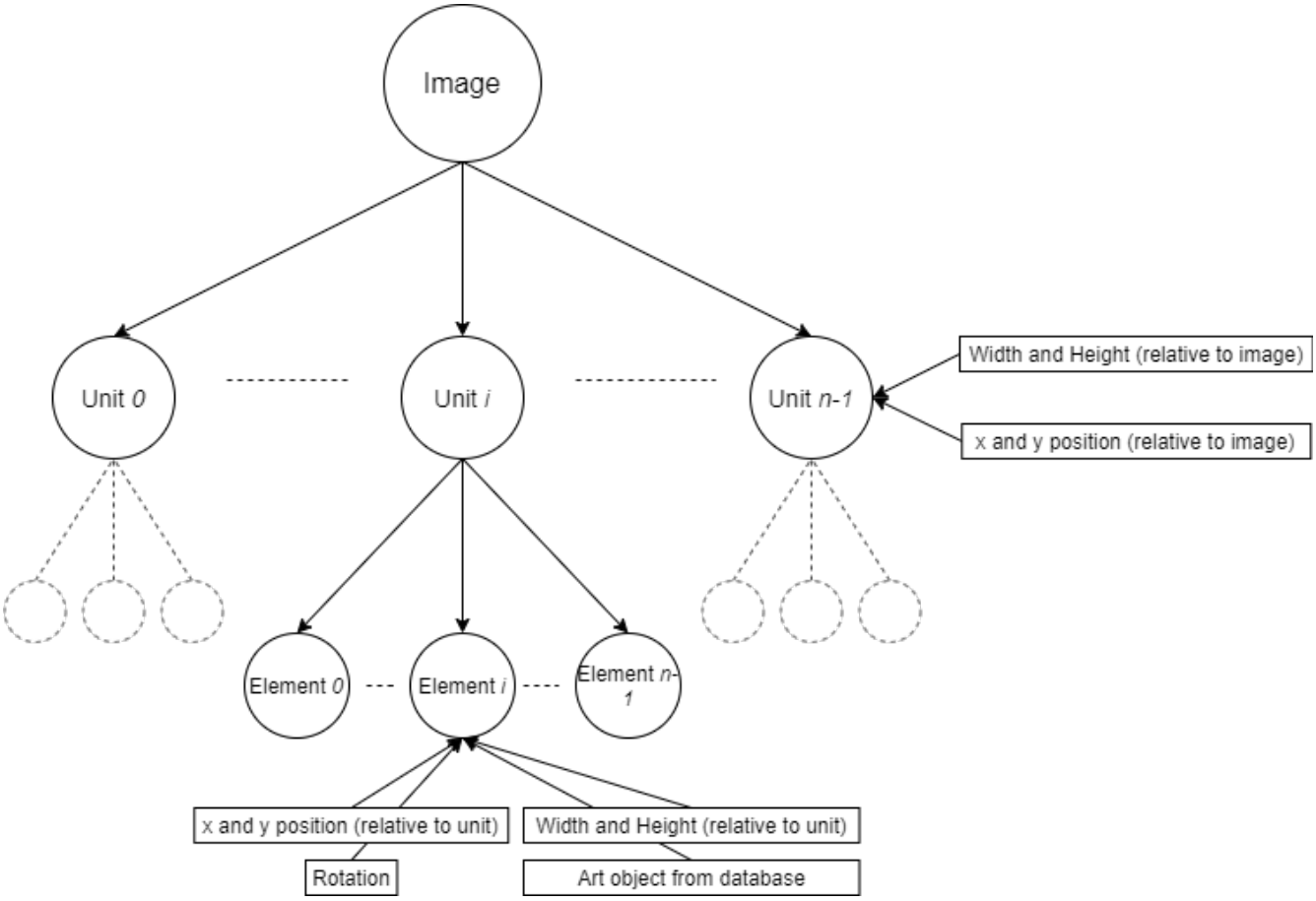


Figure 1: Representation of an image. Circles are objects that are part of the image. Rectangles are components of objects.

relative height is also 50 percent. Even though the relative width and height are the same, the actual height of the unit is smaller than the width. This is caused by the fact that the image is a rectangle instead of a square and the unit width and height are stored relative to the image width and height.
An element is constructed similarly to a unit. The middle element in the middle unit (*the horse*) will be used to help clarify the construction of an element. The element is centered inside the unit. Therefore, the position of the element is $(x : 50\%, y : 50\%)$. The relative width is 24 percent. The rotation factor is relative to how the image is stored. In this case, the image of the horse is stored in the position that can be seen in the image. Therefore the rotation factor is zero degrees.

3.2 Initialisation

The process of initialisation will be described using an enumeration. The definition of all variables can be found in table 1.

1. First, n images are generated where n is the population size
2. For each image, a random number of units will be initialised. The number of units will be limited by a user-defined range; *minUnits*, *maxUnits*
3. Each unit will have a random size and position. The width and height and x and y position will be stored relative to the image. This means that these variables will be stored as percentages
    3.1. The width, *unitWidth*, of a unit is generated randomly in a user-defined range; *minUnitWidth*, *maxUnitWidth*
    3.2. The height, *unitHeight*, is calculated using the width of the unit image ratio
    3.3. The position, *unitX* and *unitY*, is generated randomly. The only limitation on generating the positions is that units can not be placed too close to the boundaries of the image. Therefore, units can not go out of bounds
4. For each unit, a random number of elements will be initialised. The number of elements is limited by a user-defined range; *minElements*, *maxElements*
5. First, when generating elements, a sub-image from the database is chosen
6. Each element has a random size, position and rotation. These variables will all be stored relative to the unit the element is part of. This means that these variables, except rotation, will be stored as percentages
    6.1. Each element has a random width and height, *elementWidth*, *elementHeight*, that are initialised using a lower and upper bound; *minElementWidth*, *maxElementWidth*. The height of the element depends on the image ratio of the sub-image that this element represents
    6.2. Each element has a random position; *elementX*, *elementY*. It is made sure that the element will not exceed the boundaries of the unit.
    6.3. The rotation of an element is generated randomly in steps of 45 degrees

Variable name	Description	Default value
n	Population size	10
minUnits	The lower bound for the initialisation of the number of units in an image	3
maxUnits	The upper bound for the initialisation of the number of units in an image	10
unitWidth	The width of a unit	-
unitHeight	The height of a unit	-
minUnitWidth	The lower bound for the initialisation of the width of a unit	20
maxUnitWidth	The upper bound for the initialisation of the width of a unit	45
unitX	The x position of a unit	-
unitY	The y position of a unit	-
minElements	The lower bound for the initialisation of the number of elements in a unit	3
maxElements	The upper bound for the initialisation of the number of elements in a unit	8
elementWidth	The width an element	-
elementHeight	The height of an element	-
minElementWidth	The lower bound for the initialisation of the width of an element	10
maxElementWidth	The upper bound for the initialisation of the width of an element	25
elementX	The x position of an element	-
elementY	The y position of an element	-
Rotation	The rotation of an element	0

Table 1: All variables used for the initialisation of the algorithm as described in section 3.2. The default value is given for user defined variables. Default values are chosen from experimentation during the development of the algorithm

3.3 Objective function

The objective function is one of the main contributors to this research. The objective function helps to answer the research question: *"Is it possible to assess the aesthetic quality of an image in a numerical way?"* The second research question; *"Is it possible to use the previously mentioned numerical value to create an evolutionary algorithm that generates images?"*, is heavily dependent on the first research question. Therefore, it is essential for the algorithm that the objective function works correctly.
The goal of the objective function is to determine the "quality" of the image in a numerical way. The evolutionary algorithm can use the numerical value to differentiate between the quality of the images. Therefore, if the EA and objective function are implemented correctly, the quality of the images should slowly increase. In this subsection, the implementation of the objective function will be discussed.
The objective function looks at several different factors to determine the quality of an image. Each subsection will discuss a separate part of the objective function. Every part of the objective function

will have its own weight associated with it: $\alpha_i,\ i = 1, \ldots, 7$. All the scores of the separate parts of the function will be stored in $\beta_i,\ i = 1, \ldots, 7$. All mentioned variables will be summarised in table 2.

3.3.1 Overlap

One of the components the objective function looks at is the number of overlapping elements. Each element is placed randomly on the image. However, this can cause elements to overlap with each other. This is undesirable because if elements are overlapping, the image will look crowded and unstructured. We associate this with a low aesthetic value. To prevent this, a lower score will be given when elements overlap with each other. For this, the function will look at the number of elements that are overlapping and the number of pixels in the image that are overlapping.
For the number of overlaps, a score of 1000 will be given when there are no overlaps. For every overlap, 250 points will be deducted. See equation 1.
For the number of pixels, a percentage is used. See equation 2.
The total score for the overlap will take the average of these two formulas. See equation 3.

$$score_1 = 1000 - (numberOfOverlaps \cdot 250) \tag{1}$$

$$score_2 = 10 \cdot (100 - percentage) \tag{2}$$

$$\beta_1 = \frac{score_1 + score_2}{2} \tag{3}$$

3.3.2 Emptiness

The objective function also looks at the number of empty pixels on the screen. An image that is filled with very small elements and therefore looks very empty is unsatisfactory. The same can be said for images that are too crowded with bigger elements. Therefore, the objective function rewards the image if the screen is filled to a percentage the user desires.
The formula that calculates the fitness for the emptiness can be found in formula 4. t is a user-defined parameter for the preferable percentage of the screen filled (in pixels). $t = prefFilled$ in table 2. This function will reward the image for being close to the desired value but will punish it when it exceeds this value. x is the actual percentage of the screen filled.

$$\beta_2 = \begin{cases} -\dfrac{100}{t^2}(x-t)^2 + 100, & \text{if } \{0 < x \leq t\} \\ -\dfrac{100}{t-100}(x-100), & \text{if } \{t < x < 100\} \end{cases} \tag{4}$$

3.3.3 Symmetry

One of the most important parts of this objective function is symmetry detection. We define symmetric images as images with high aesthetic quality. In a symmetric image, the position, color, shape, size and rotation of all elements are related and compared to each other. Therefore, this part of the objective function is very important in answering the two research questions. Since symmetry

detection is difficult to implement ourselves, the symmetry detection algorithm from [10] is used. This algorithm calculates a score based on how symmetric an image is. The algorithm can predict multiple symmetries and will return a numeric value on how certain it is per symmetry found. The higher the score, the more symmetric the image is. Elawady et al. use edge-based feature extraction using Log-Gabor filters with a special voting scheme [10]. The paper mentions that this method provides a great improvement over other proposed methods. However, it does not mention how the score is calculated. Therefore, an assumption is made that the score will returned will be between 0 and 1 where 0 means no symmetry at all and 1 means that the algorithm is 100 percent confident that there is symmetry. Since the algorithm can return multiple scores the user can define how many found symmetries will be used. The average of these scores will be taken. The score returned by the symmetry function, *symmetryScore*, is multiplied by 1000. Therefore the output will be:

$$\beta_3 = symmetryScore \cdot 1000 \tag{5}$$

3.3.4 Number of units

A unit is a collection of elements. Having too many or too few units can make an image feel either too crowded or too empty. Therefore, the objective function also looks at the total number of units; *numUnits (nU)*. When the number of units are inside a predefined range, *minPrefUnits (minPU)*, *maxPrefUnits (maxPU)*, maximum points will be given. The bigger the difference between the predefined range and the actual value, the lower the score. 500 points will be deducted from the total score for every unit too little or too much. *min(a,b)* is a function that returns the minimum value of a and b.

$$\beta_4 = \begin{cases} 1000, & \text{if } minPU \leq nU \leq maxPU \\ 1000 - (500 \cdot min(|nU - minPU|, |nU - maxPU|)), & \text{otherwise} \end{cases} \tag{6}$$

3.3.5 Number of elements

Similarly to the previous subsection, having too many or too few elements inside a unit can make the unit feel too crowded or too empty. Thus, the objective function also looks at the number of elements *numElements (nE)* inside each unit. If the average of elements per unit is inside the user-defined range, *minPrefElements (minPE)*, *maxPrefElements (maxPE)*, the maximum score is given. The further away the average is from the user-defined range, the lower the score will be. For every single unit, a score is calculated. If the number of elements in the unit are inside the preferred range, 1000 points are given for this unit. 500 points are deducted for every element outside the preferred range. The average over all units is taken. u is the number of units. *min(a,b)* is a function that returns the minimum value of a and b.

$$\forall units\ score_i = \begin{cases} 1000, & \text{if } minPE \leq nE \leq maxPE \\ 1000 - (500 \cdot min(|nE - minPE|, |nE - maxPE|)), & \text{otherwise} \end{cases} \tag{7}$$

$$\beta_5 = avg(score_1, \ldots, score_u) \tag{8}$$

3.3.6 Recurring elements

The objective function will give a higher score for repeating the same elements. Repeating the same elements can cause a pattern to occur. We associate an image with a pattern as an image with high aesthetic quality. The function will count how many elements are repeated, and once a user-defined value is reached, $prefRepElements = k$, close to maximum points will be given. x is the number of repeating elements. The elements do not have to be inside the same unit for the repeating bonus to count.

$$\beta_6 = 1000 + \frac{-k \cdot 100}{x + \frac{t}{10}} \tag{9}$$

3.3.7 Variance

The objective function will also give a higher score for having a varied image. We associate an image with very few unique elements as monotonous. Similarly, an image with too many unique elements can be seen as too cluttered. The function will count how many different unique elements, *numUniqueElements (nUE)*, there are in the image. If the number of unique elements is inside a user-defined range, *minPrefVar (minPV)*, *maxPrefVar (maxPV)* maximum score will be given. 500 points will be deducted for every unique element outside the preferred range. *min(a,b)* is a function that returns the minimum value of a and b.

$$\beta_7 = \begin{cases} 1000, & \text{if } minPV \leq nUE \leq maxPV \\ 1000 - (500 \cdot min(|nUE - minPV|, |nUE - maxPV|)), & \text{otherwise} \end{cases} \tag{10}$$

3.3.8 Total fitness

The total fitness is the sum of all previously calculated values multiplied by their corresponding weights.

$$fitness = \sum_{i=1}^{7} \alpha_i \cdot \beta_i \tag{11}$$

Variable name	**Description**	Default value
$\alpha_i, i = 1, \ldots, 7$	Weights for each part of the objective function	1
prefFilled	The preferred percentage of empty pixels on the screen	20
minPrefUnits	The minimum number of the preferred number of units in an image	4
maxPrefUnits	The maximum number of the preferred number of units in an image	8
minPrefElements	The minimum number of the preferred number of elements in a unit	3
maxPrefElements	The maximum number of the preferred number of elements in a unit	8
prefRepElements	The preferred number of repeating elements, which means how many elements are repeated	7
minPrefVar	The minimum for the preferred number of variance, which means how many unique elements there are	4
maxPrefVar	The maximum for the preferred number of variance, which means how many unique elements there are	10
numUnits	The number of units in an image	-
numElements	The number of elements in a unit	-
numUniqueElements	The number of unique elements in an image	-
numberOfOverlaps	The number of elements that are overlapping with each other	-

Table 2: All variables used for the objective function of the algorithm as described in section 3.3. The default values are given for user defined variables

3.4 Selection

For the selection step of the algorithm, tournament selection is used. Tournament selection is a common method for selection in evolutionary algorithms [16]. It is a stochastic operator. This means that two individuals will be picked at random. These two individuals will fight each other in a "tournament". The individual with the highest fitness wins the tournament and gets selected for crossover. In the case of this algorithm, an individual represents an image. Therefore, the fitness of the images will be compared and the one with the highest aesthetic quality will be selected.
There are two types of selection methods: proportional and elitist [12]. In proportional methods, the chance of selection is proportional to the fitness of the individuals. In elitist methods, the selection method prefers the individuals with the highest fitness. The first method promotes genetic diversity, but a slower convergence to the global optimum. The second method promotes faster convergence, but a higher chance of getting stuck in local optima.
For this algorithm, tournament selection without elitism is chosen. This is done to prevent the algorithm from getting stuck in local optima. This means that there is no guaranteed chance for the individual with the highest fitness to be selected for the next generation. Also, according to Miller and Goldberg; "tournament selection is easy to implement and works well on parallel and nonparallel architectures" [16].

3.5 Mutation

The mutation step is supposed to add some randomness to the images in the hope that the quality of the image will improve. Below is a list of all possible mutations on an image. The list has been split into unit-specific mutations and elements-specific mutations

- Mutations on units
  - Move unit: move a unit randomly in the x/y direction for a user-defined minimum and maximum distance; *minUnitDistance*, *maxUnitdistance*.
  - Resize unit: increase or decrease the size of a unit randomly in a user-defined range; *minResize*, *maxResize*
  - Delete unit: delete a random unit from an image
  - Add new unit: add a new unit to an image. This unit is generated randomly as described in section 3.2
  - Copy unit: copy a unit and place it randomly inside the image
- Mutations on elements
  - Move element: move an element randomly in the x/y direction for a user-defined minimum and maximum distance; *minElementDistance*, *maxElementDistance*. These variables are stored relative
  - Resize element: increase or decrease the size of an element randomly in a user-defined range. This uses the same variables as resizing a unit
  - Delete element: delete a random element from a random unit
  - Add new element: add a new element to a random unit. This element is generated randomly as described in section 3.2
  - Change element: change an element into another element
  - Copy element: copy an element and place it randomly inside the unit
  - Rotate element: randomly rotate an element clockwise or counterclockwise in a step size of 45 degrees. This means that an element can be rotated 0, 45, 90, or 135 degrees

Variable name	**Description**	Default value
minUnitDistance	The minimum distance a unit can be moved	0.5
maxUnitdistance	The maximum distance a unit can be moved	20
minElementDistance	The minimum distance an element can be moved	0.1
maxElementDistance	The maximum distance an element can be moved	20
minResize	The minimum factor for resizing a unit or an element	0.1
maxResize	The maximum factor for resizing a unit or an element	3

Table 3: All variables used for the mutation step of the algorithm as described in section 3.5

3.6 Crossover

For crossover, single point crossover is used. This is done the keep the algorithm relatively simple. Since all the units are independent of each other and the position (in the genome) and order of the units are irrelevant, more complicated crossovers (e.g. partially mapped crossover, cycle crossover) would only make the algorithm itself more difficult to implement, without having any relevant impact. Therefore, single point crossover is used.
In this implementation, crossover always happens at the center of the genome for the number of units. For example, when combining two images, if the first image has five units and the second image has three, the first image will give two units to the second image and receive only one unit from the second image. Both images end up with 4 units. If an odd number of units are present, the number of units given will either be rounded up or down, depending on if the image that receives the units has more or less units.

4 Experiments & Results

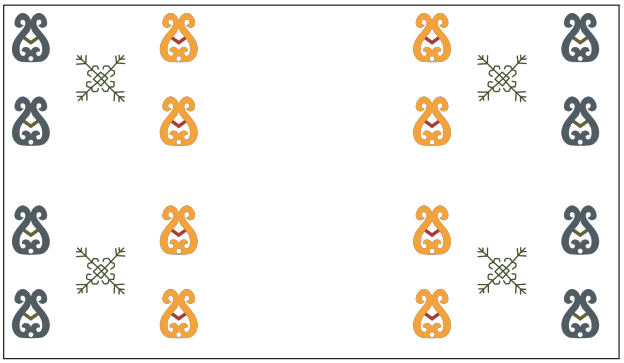
4.1 Testing the objective function

This experiment is performed to evaluate the performance of the objective function. This is closely related to the first research question: *"Is it possible for a computer to assess the aesthetic quality of an image in a numerical way?"*
For the first experiment, an image will be created that is supposed to have a high aesthetic quality. Next, random mutations will occur on this image. Using these random mutations, we can see how the objective function reacts. This image and the images where mutation has occurred can be found in images 2a through 5c. Images 2b through 5d show the corresponding scores given by the objective function. The image on which all mutations will occur is figure 2c. The value of each variable used by the objective function can be found in table 4.
Images 2a and 2c are created by hand to represent a simple aesthetically pleasing image. Figure 2a is perfectly symmetric. Figure 2c is also symmetric apart from the asymmetric horse in the middle. When looking at figure 2b and 2d, it can be seen that both images get a maximum score for the number of units, number of elements and preferred number of repeating elements. However, figure 2a does not reach the maximum score for the variation of the elements in the image. This is because the image only has 3 unique elements, while the preferred minimum value is 4. Even though there are no overlapping elements present in the image, the scores given by the objective function are not maximum on this subject. The reason for this is that in the code an imperfect method is used to check for overlap. This method, however, speeds up the process itself. Since the maximum score for overlap in all images is close together, this small difference from the actual total maximum score can be neglected. There is a small difference in the score for the percentage of the screen filled. Since figure 2c has an extra unit, it is closer to the desired value of the number of pixels on the screen. The most interesting difference can be found in the symmetry scores of the images. This will be discussed in more detail in section 4.1.1.
Images 2e and 3a are very similar in how the mutation has affected the scores given by the objective function. Figure 2e has copied a random unit and pasted it somewhere on the image. In this particular case, the copy has been pasted on top of the top left unit. Also, random rotations have taken place. 3a has moved the bottom right unit in the negative x and y-direction. When looking

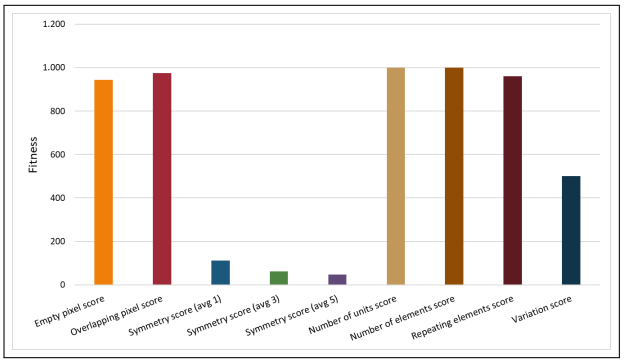
at figure 2f and 3b, it can be seen that the score given for the number of overlapping elements is significantly lower in figure 2e than in figure 2c. The same can be said about figure 3a. This is to be expected because the mutations caused overlap to occur. Also, the score for overlapping elements in figure 3a is almost double that of figure 2e. This is because the former image has 2 units overlapping entirely and the latter image only has partial overlap.
In figure 5a a random unit is added to the image. This unit causes overlap to happen. So much overlap is happening that the score for overlapping elements is negative.
Figure 5e is an image purposely designed to have low aesthetic quality. It consists of two different units which both have been copied and moved once.

4.1.1 Symmetry

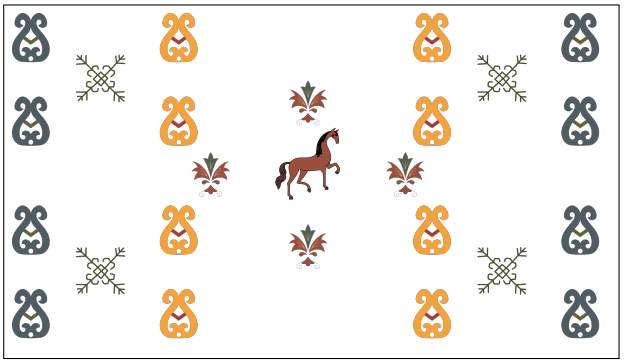
In all images, the biggest difference can be found in the symmetry score. There are three different symmetry scores shown. The best symmetry score, the average of the top three symmetry scores and the average of the top five symmetry scores. A comparison of these scores can be found in figure 7. The first thing that stands out is the fact that the score given for symmetry is significantly lower than the other scores. See figure 2d for example. There are a few possible causes for this. In section 3.3.3 the assumption is made that the score returned by the symmetry detection function will be between 0 and 1. This assumption is most likely incorrect. However, we do know that a higher score means that the algorithm is more confident in having detected symmetry. Therefore, a solution to this problem would be to increase the weight of this segment of the objective function to equalize the scores.
However, when comparing image 2a to image 2c we can see that image 2c has a higher symmetry score. This can also be seen in figure 7. This is odd since image 2a is perfectly symmetric and image 2c has an asymmetric element in the center. One would expect that image 2a would receive a higher score. Elawady et al. use edge detection to make an educated guess [10] about the symmetry. Even though image 2c is less symmetric, more "information" is present in the image. Apart from the horse, four other elements are added that are symmetric. Therefore, more data is present in the image for the algorithm to make an educated guess. Since more data is present, the algorithm can be more confident in saying that this image appears to be symmetric. This theory is supported even more when comparing image 2a to an image used for testing in the research done by Elawady et al. [10]. This comparison can be found in image 6. When looking at figure 6a, we can see that the algorithm does find the correct symmetry axis. The red line is the most confident guess. The score given for this red line is ~0.115. But when looking at image 6b, we can see that the score for its most confident guess is ~0.322. This is almost triple the score for image 6a.
Therefore, it is plausible that when the images are generated, the symmetry detection algorithm will cause the generated images to be more crowded and full. However, the segment of the objective function that looks at the percentage of the screen filled (section 3.3.2) should combat this.
Also, even though the symmetry scores of images 2a and 2c are illogical, they are significantly higher than the symmetry score for image 5e. Image 5e is an image created to not be aesthetically pleasing.
When looking at figure 7, a similar pattern can be seen between the best score, average of top 3 scores and average of top 5 scores. The ratios of the scores returned by the algorithm remain the same. Therefore no significant advantage can be found in using for example the average of the top 5 scores over the using only the best score.



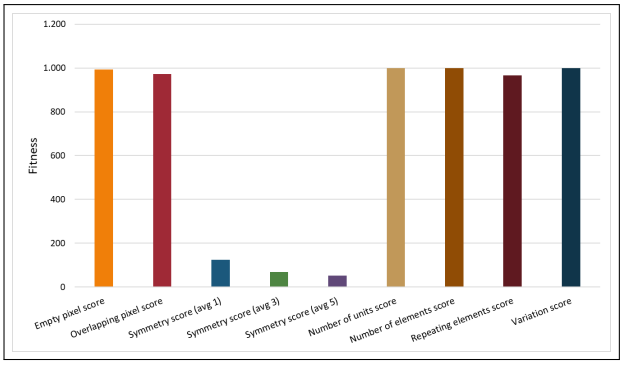
(a) Manually created symmetric figure



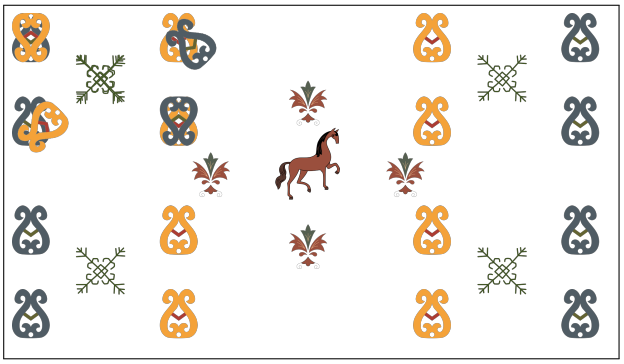
(b) Scores of the components of the objective function



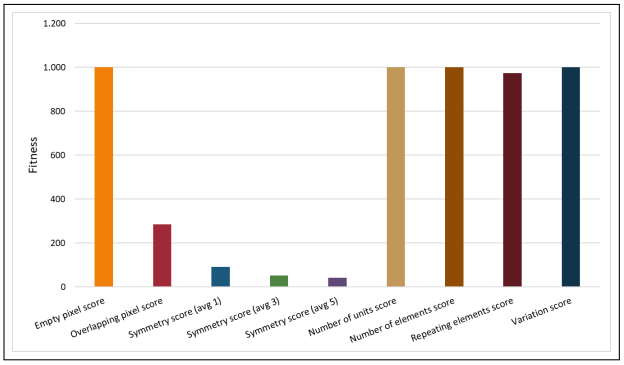
(c) Symmetric figure with an extra unit inside. This extra unit has been manually created



(d) Scores of the components of the objective function



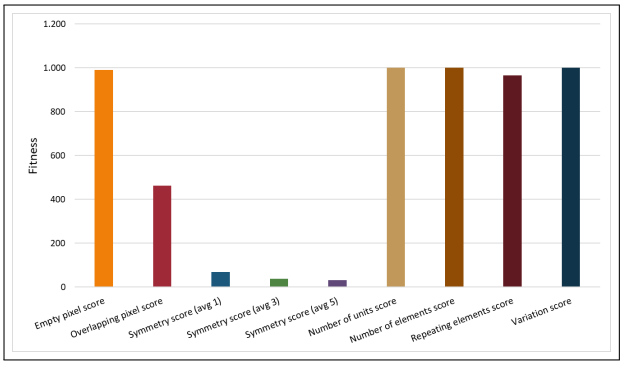
(e) Mutation where a unit is copied, the elements are randomly rotated and placed on top of another unit



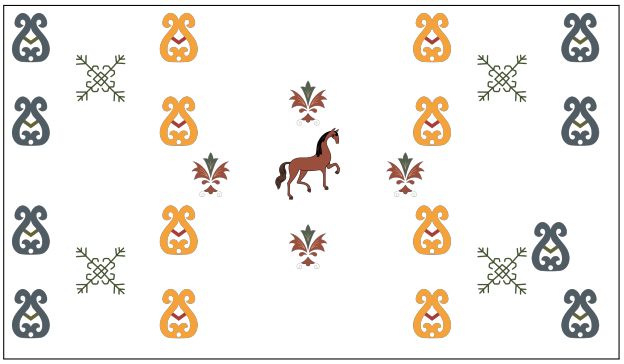
(f) Scores of the components of the objective function

Figure 2: Figures 2b, 2d, 2f are respectively the objective scores of figures 2a, 2c, 2e. Figures 2a and 2c are manually created images. They have a transparent background. Subsequently, all mutations on these images also have a transparent background. Figure 2e is a mutation on figure 2c

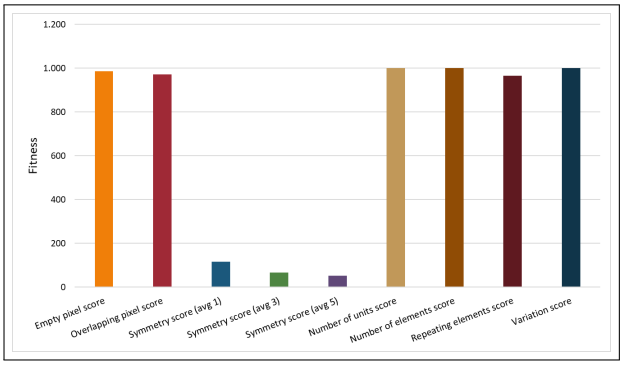
(a) Mutation on the bottom right unit where it is moved randomly



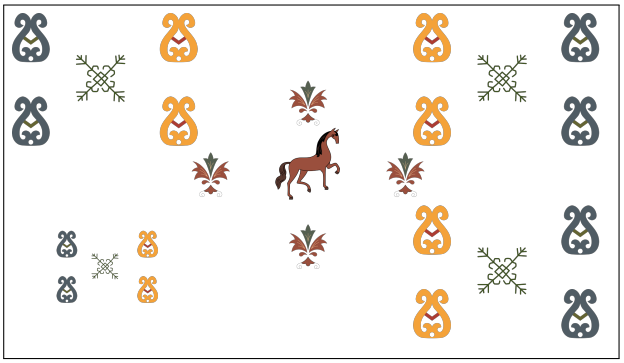
(b) Scores of the components of the objective function



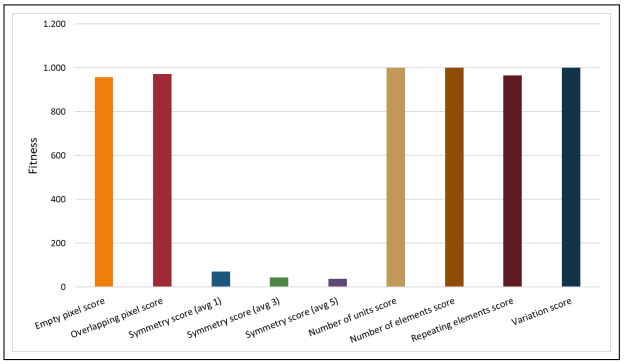
(c) Mutation on an element in the bottom right unit. This element is moved randomly



(d) Scores of the components of the objective function

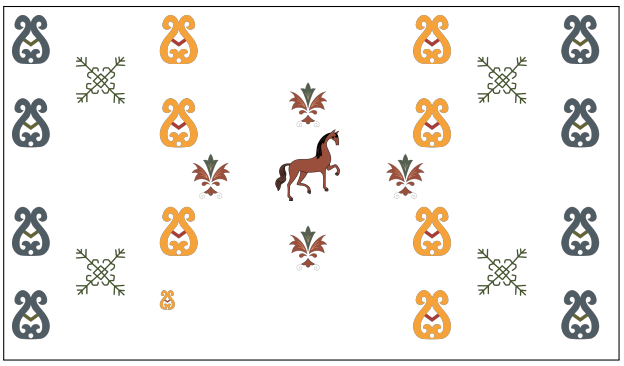


(e) Mutation on the bottom left unit where the size of the unit is decreased

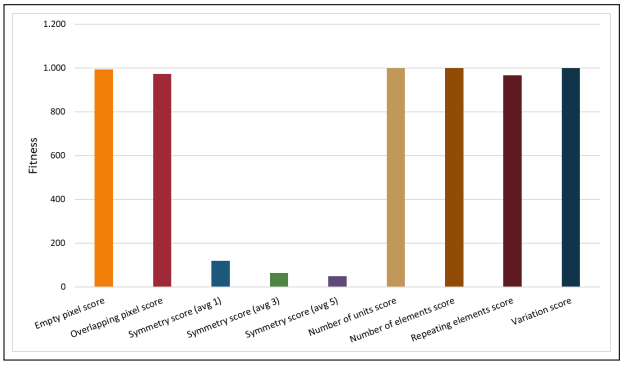


(f) Scores of the components of the objective function

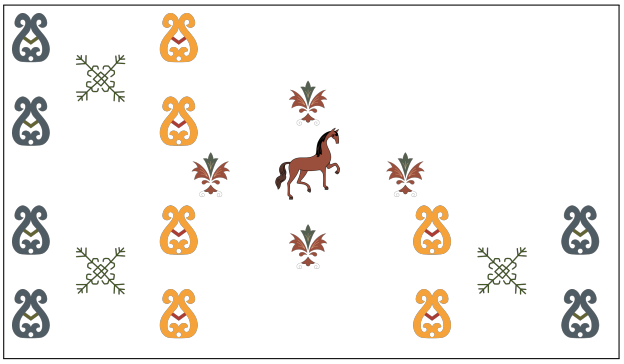
Figure 3: Figures 3b, 3d, 3f are respectively the objective scores of figures 3a, 3c, 3e. Figures 3a, 3c and 3e are mutations on figure 2c



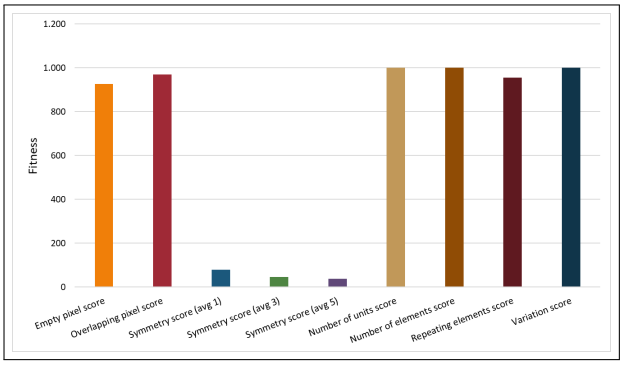
(a) Mutation on an element in the bottom left unit where the size of the element is decreased



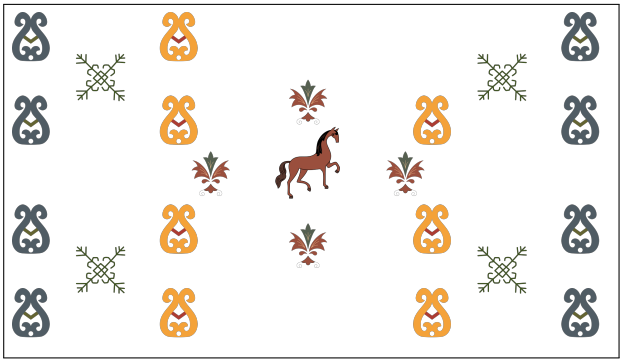
(b) Scores of the components of the objective function



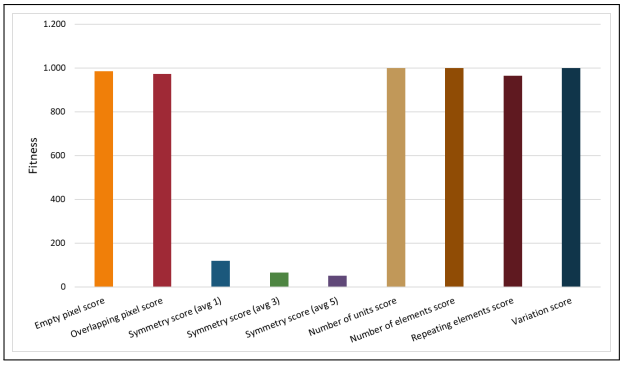
(c) Mutation where the top right unit is deleted from the image



(d) Scores of the components of the objective function

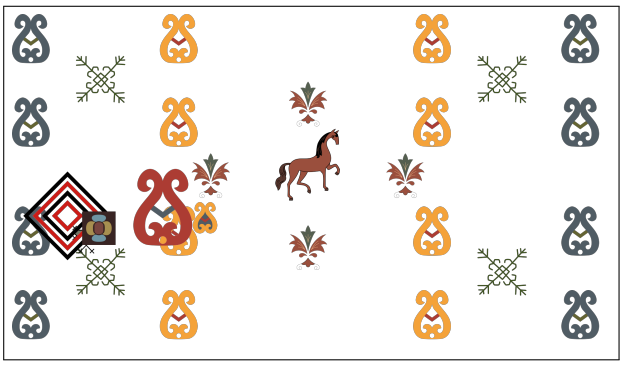


(e) Mutation on the top right unit where a random element is deleted

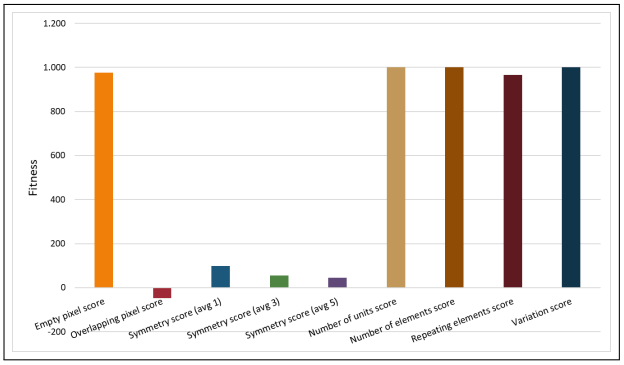


(f) Scores of the components of the objective function

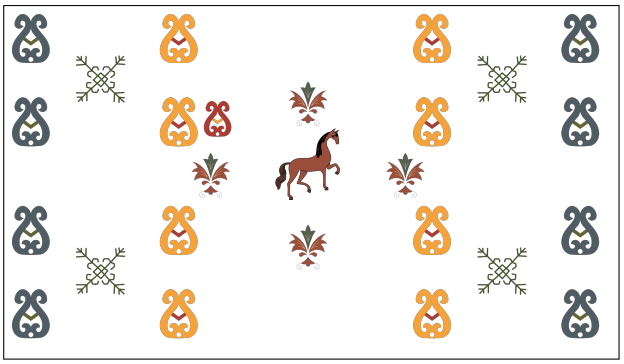
Figure 4: Figures 4b, 4d, 4f are respectively the objective scores of figures 4a, 4c, 4e. Figures 4a, 4c and 4e are mutations on figure 2c



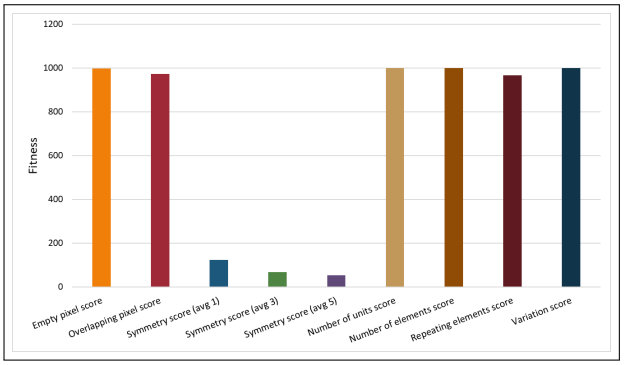
(a) Mutation where a random unit is added to the image



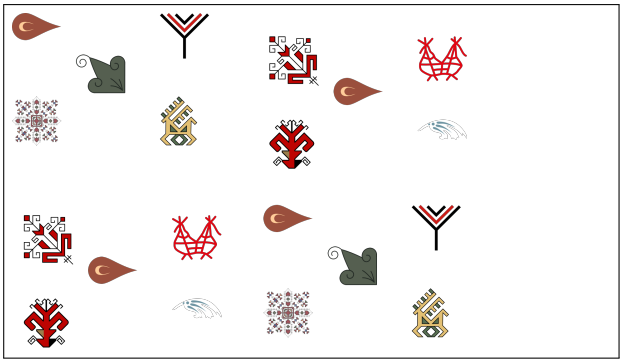
(b) Scores of the components of the objective function



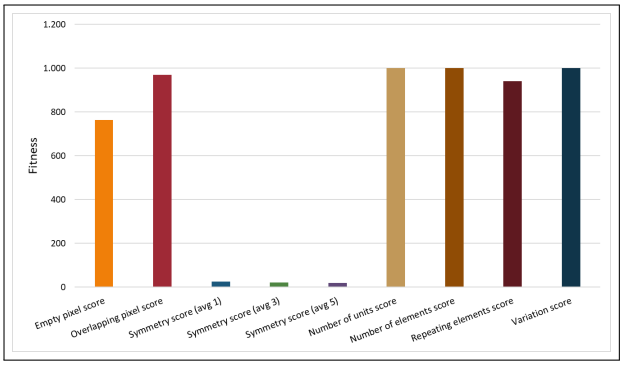
(c) Mutation where a random element is added to the middle unit



(d) Scores of the components of the objective function

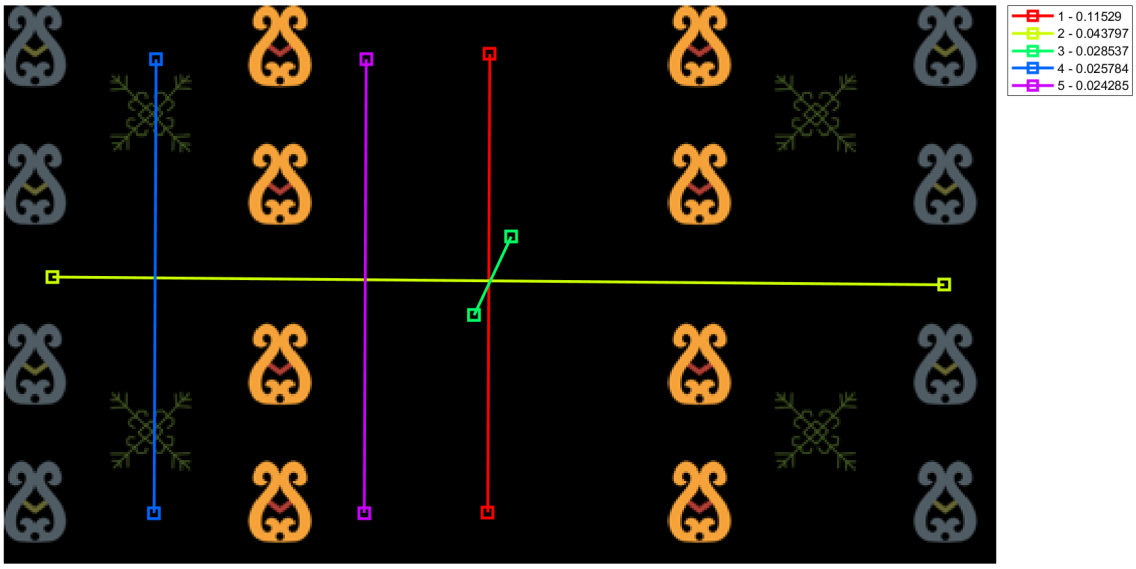


(e) Example of an image with low aesthetic quality

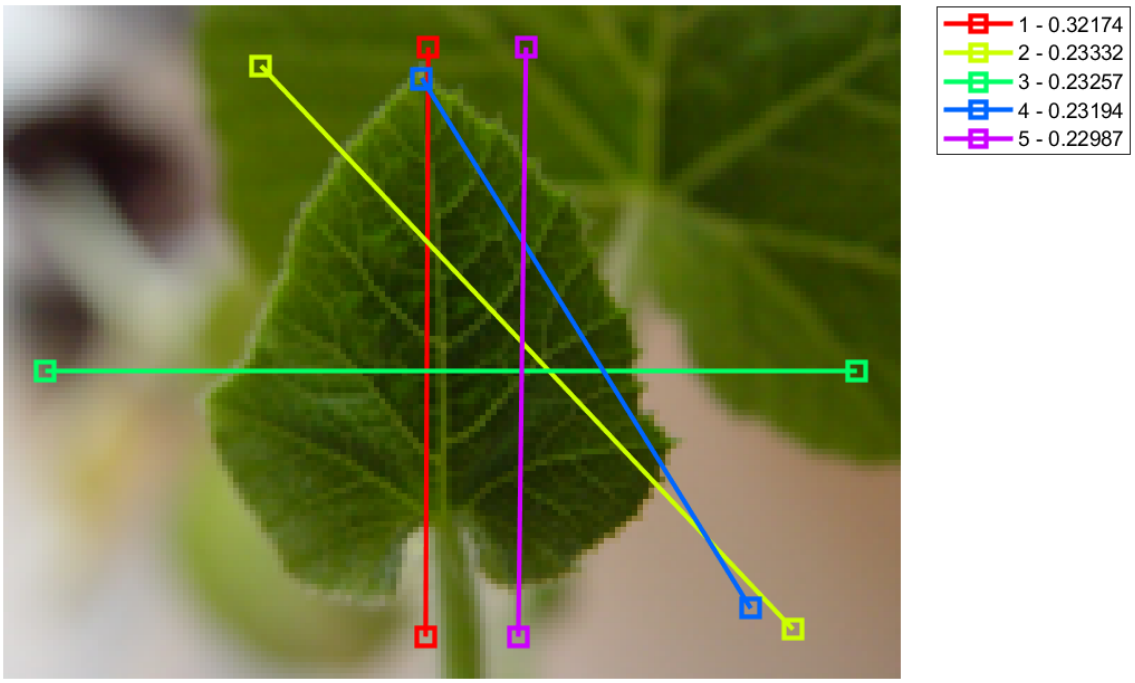


(f) Scores of the components of the objective function

Figure 5: Figures 5b, 5d, 5f are respectively the objective scores of figures 5a, 5c, 5e. Figures 5a and 5c are mutations on figure 2c. Figure 5e is purposely created to not be aesthetically pleasing



(a) Symmetry axis on figure 2a. The background of this image is transparent, even though it is shown as black.



(b) Symmetry axis on an image provided by [10]

Figure 6: The symmetry axes with scores generated by the symmetry algorithm as described in section 3.3.3

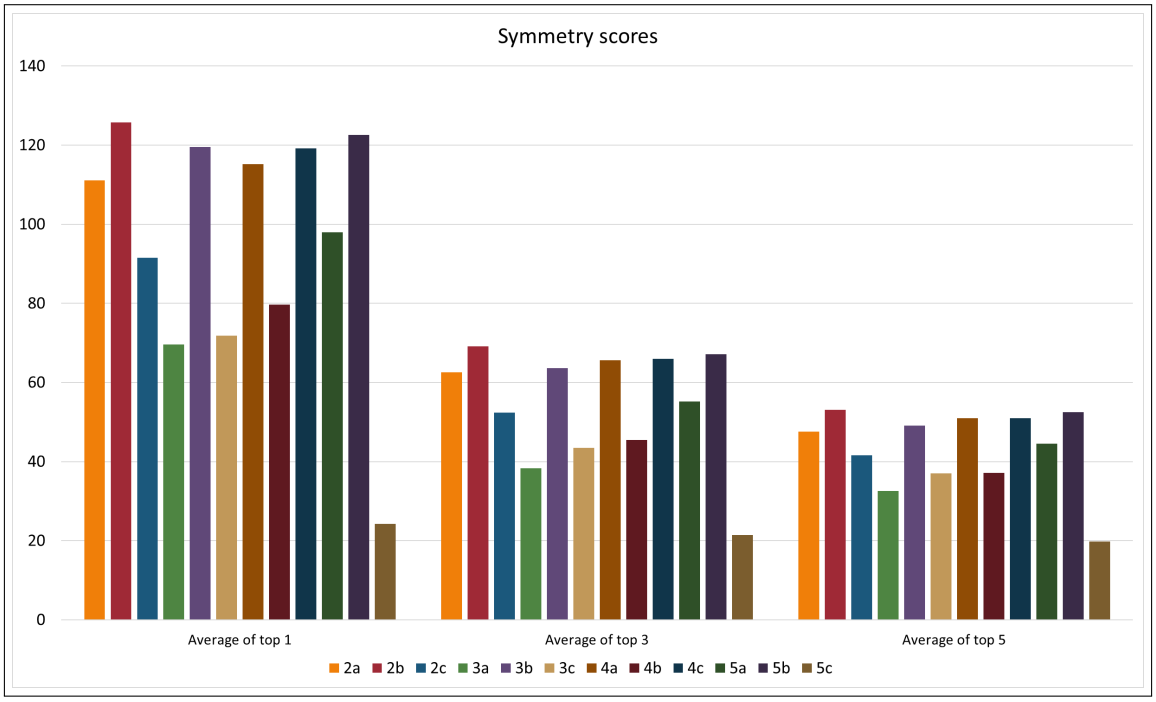


Figure 7: Symmetry scores of each image where the average of the top 1, 3 and 5 found symmetries is given

Variable name	Default value
$\alpha_i, i = 1, \ldots, 7$	1
prefFilled	20
minPrefUnits	4
maxPrefUnits	8
minPrefElements	3
maxPrefElements	8
prefRepElements	7
minPrefVar	4
maxPrefVar	10

Table 4: Values of the parameters used by the objective function corresponding to the figures 2 through 5 and figure 7 in section 4.1. The definition of the variables can be found in table 2

4.2 Measuring the performance of the evolutionary algorithm

In this subsection, the evolutionary algorithm itself will be executed. From the results of this experiment, it is possible to conclude if the method used to develop the algorithm works or if other implementations need to be used. First, the values of the parameters need to be defined. If a parameter is not mentioned, the default value will be used. The changed parameter can also be found in appendix A. The default values can be found in their respective table. The definition of which weight corresponds to which part of the objective function can be found in table 5.

Weight	Part of objective function
α_1	Overlap
α_2	Emptiness
α_3	Symmetry
α_4	Number of units
α_5	Number of elements
α_6	Recurring elements
α_7	Variance

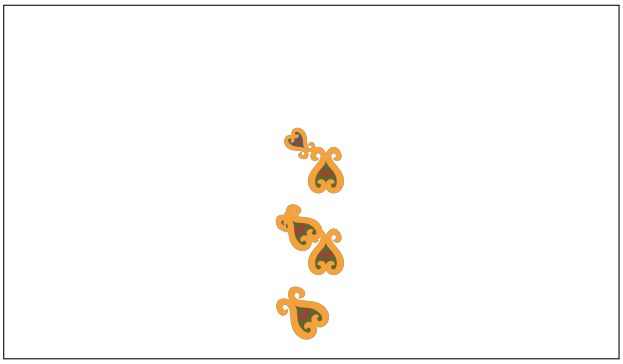
Table 5: Weights with their respective parts of the objective function

The values of the parameters will be defined based on the findings in the previous section. There we noticed that the α_1, α_2 and α_3 had significant impact on the aesthetic value of an image. Therefore these weights will be increased. α_1 and α_2 will be increased to a value of 2. Because in the previous section the discovery was made that the value returned by the symmetry function was significantly lower, α_3 will get a value of 20. $\alpha_4, \alpha_5, \alpha_6$ and α_7 were considered to have a lower impact. Therefore the values for the respective weights will be lowered to 0.5. The result of the first experiment can be found in figure 8a and the objective scores can be found in figure 10a. When looking at the total fitness, it can be seen that the score is almost 50000. Moreover, the total fitness exists almost entirely out of the symmetry score. The other functions have close to zero influence on the total fitness.
Therefore, for the second experiment, the weight for the symmetry score is lowered from 20 to 2. All other values remain the same as in the previous experiment. The results of the second experiment are more balanced than the previous experiment. However, the symmetry score is close to zero and does not influence the total fitness. Thus for the next two experiments, the weight for the symmetry score is slowly increased to increase its influence. The third experiment increases the weight for symmetry score to 10 and the fourth experiment increases the weight to 15.
The results of the first, third and fourth experiment (respectively figure 8a, 8c, 8d) look somewhat unfilled and empty. Therefore, for the fifth experiment, the weight for the emptiness of the image will be increased from 2 to 3. Also, to make sure that the end result of the image contains the preferred number of units α_4 will be increased to 1.
For the sixth experiment, the weight for the symmetry score is again increased. This time from 15 to 18. In experiment five (figure 10e), the symmetry score still has no relevant impact on the total fitness. For the seventh experiment, a similar thought pattern is followed. The symmetry score is found to still not have a relevant impact on the total fitness. Therefore the weight is increased to 20. This is the same value as in experiment one. However, considering that in the previous experiments the symmetry score did not "explode" as much as in experiment one, it is very likely to say that that was an anomaly. This statement is supported even more when looking at the results of experiment seven in figure 11a. Experiment eight has the same parameters as experiment seven. However, the subset of available images for the algorithm is changed.
When looking at the total fitness of the first eight experiments (Figures 10a through 10b), it can be seen that the total fitness fluctuates significantly. Therefore, the best result of a generation can be worse than the previous generation. For example, in the last 200 generations of experiment seven (Figure 11a), the total fitness score diminishes significantly. For these experiments, the population size is kept relatively low since the algorithm is computationally heavy. However, with

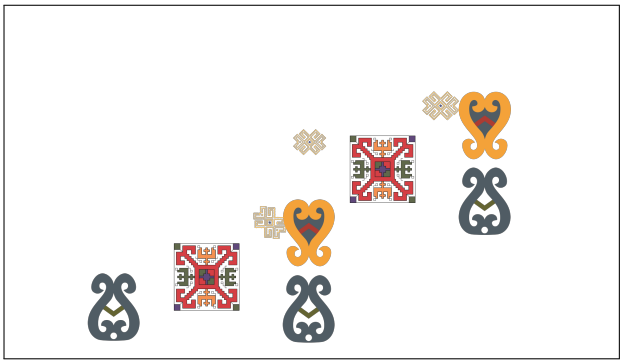
a low population size, there is a bigger chance that none of the generated images in the next generation have a higher fitness score. Thus, for experiment nine the population size is doubled from 10 to 20 and the total number of generations is halved from 1000 to 500. Hopefully, starting from this experiment and forward the total fitness score fluctuates to a lower degree.
Experiment ten follows a similar pattern of doubling the population size and halving the total number of generations. The population size is increased to 40 and the number of generations is decreased to 250. Experiment eleven has the same parameters as experiment ten (including the same population size). The number of generations is increased to 1000.

5 Discussion & Limitations

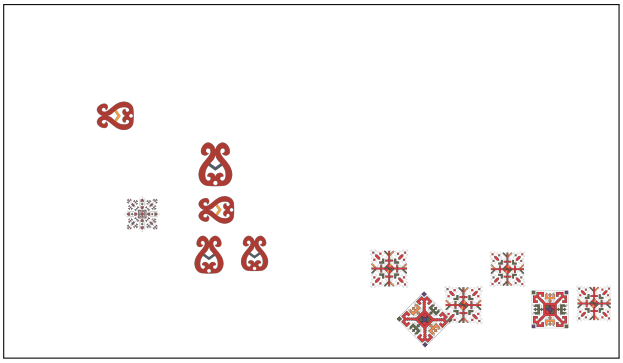
In this research we set out to answer two questions: *"Can a computer define the aesthetic value of an image in a numerical way?"* and *"Can we use the previously defined numerical value to generate images using an evolutionary algorithm?"* To help answer these questions, we will discuss the results given in section 4.
First, the question *"Can a computer define the aesthetic value of an image in a numerical way?"* will be discussed. Defining the aesthetic value of an image is a complicated task. For the images used in this research, the objective function has been split into seven parts. Each part independently judges a characteristic of the image and returns a score based on if the characteristic is (partly) present in the image. The total of these scores is the numerical value of the aesthetic quality of the image. In figure 2d the numerical values corresponding to image 2c returned by the 7 parts of the objective function can be found. Figure 2c is an example of a simple aesthetically pleasing image. The maximum that each part can return (except for the symmetry part) is 1000. In the research done by Elawady et al. [10] no maximum return value is defined for the symmetry function. Because of this the return value can be significantly higher than other values, as can be seen in figure 10a. However, this can be solved by adjusting the weights correctly.
When looking at figure 2d, we can see that the scores for this image are maximal on most parts of the objective function. However, the symmetry score returned is extremely low. In fact, this is the case for all images discussed in section 4.1. When comparing the symmetry scores from section 4.1, something strange stands out. The symmetry score for figure 2c is higher than the score for figure 2a. This comparison can best be seen in figure 7. This anomaly has also been discussed in section 4.1.1. Figure 2a is perfectly symmetric and therefore the highest score from the symmetry function is expected. Figure 2c is an exact copy of figure 2a with an asymmetric unit added in the center. Subsequently, a lower score is expected in comparison to figure 2a. However, this is not the case. Also, when testing the symmetry function on a photographic image of a leaf, a higher score is returned compared to figure 2a even though the leaf is definitely not perfectly symmetric. This comparison can be found in figure 6. From these findings, we can already tell that the symmetry function can be unreliable and may cause problems when implemented in an evolutionary algorithm. The symmetry function is a key part of the objective function and is supposed to do most of the heavy lifting. When looking for symmetry; shape, size, color, rotation, position and more are compared to figure out if an image is symmetric. A perfectly symmetric image where all elements are positioned aesthetically and another asymmetric image using the same elements in random positions (without overlap) would get the same fitness score if the symmetry part is left out of the objective function. That is why an inconsistent symmetry function can cause problems.



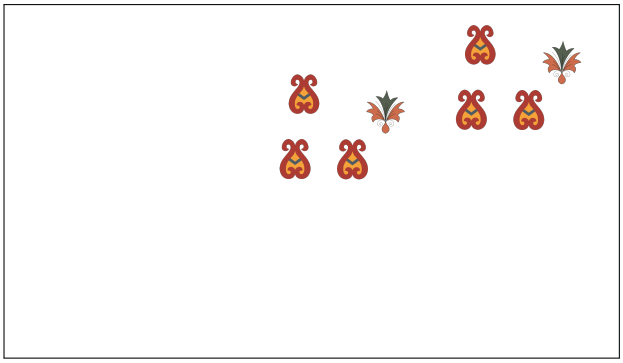
(a) Result of experiment 1



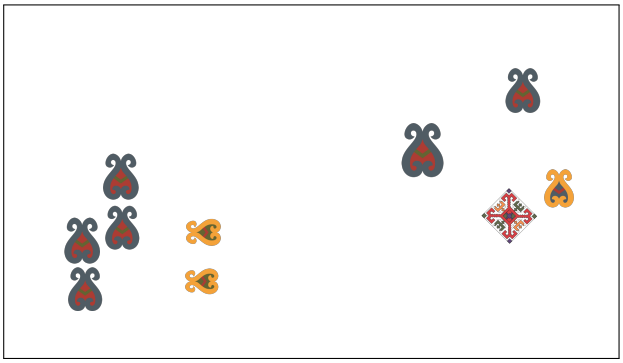
(b) Result of experiment 2



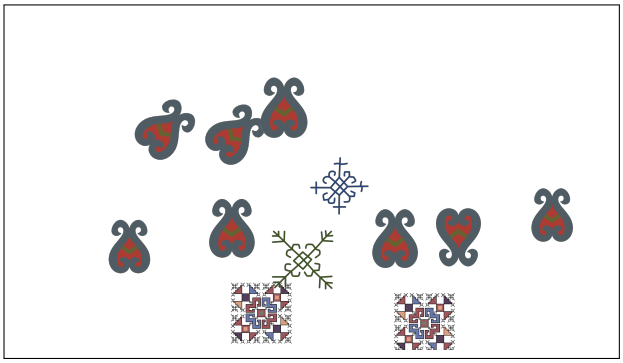
(c) Result of experiment 3



(d) Result of experiment 4

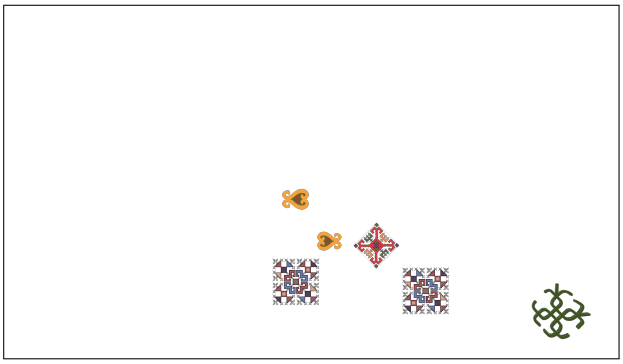


(e) Result of experiment 5

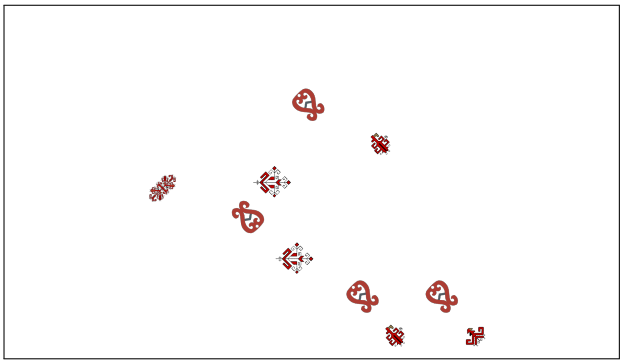


(f) Result of experiment 6

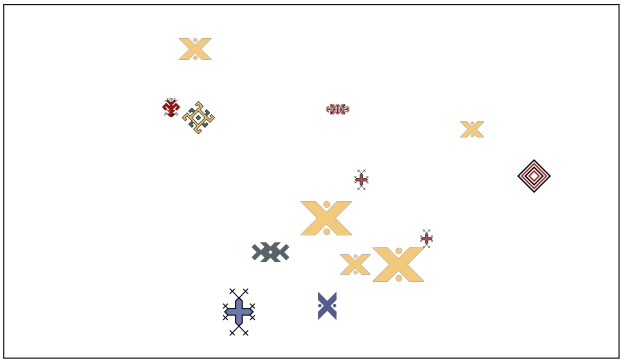
Figure 8: Results of the first six experiments relating to section 4.2. The best image of the last generation is shown



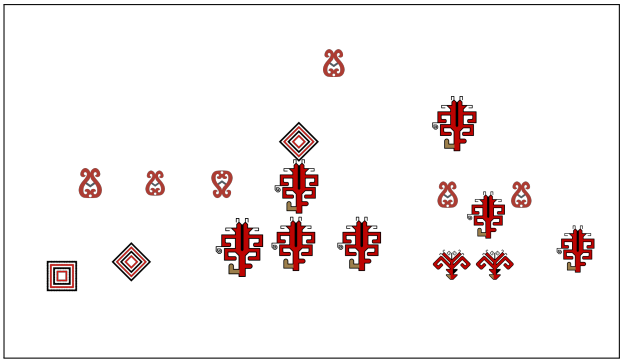
(a) Result of experiment 7



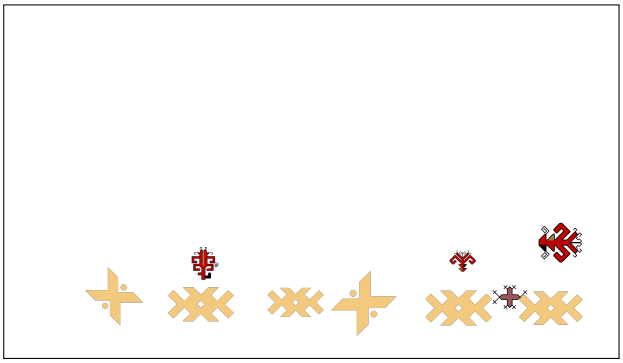
(b) Result of experiment 8



(c) Result of experiment 9

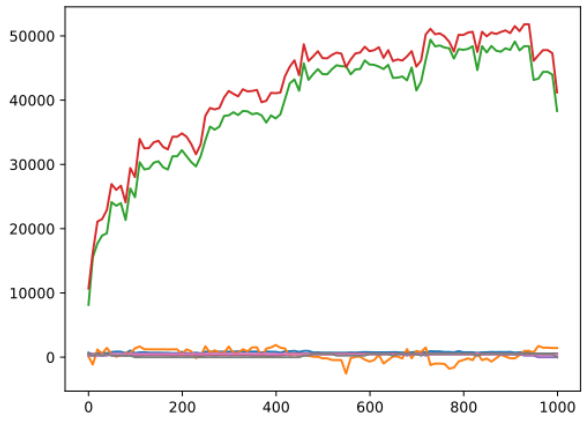


(d) Result of experiment 10

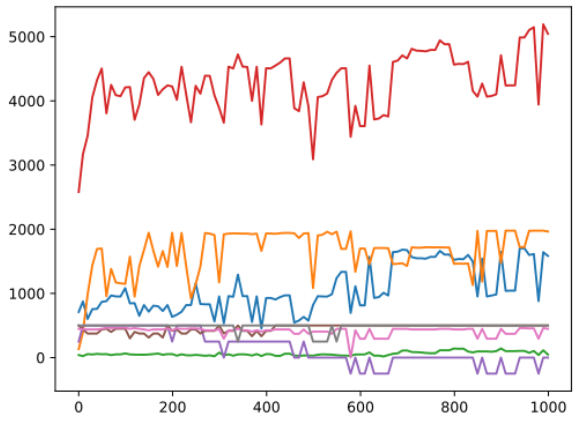


(e) Result of experiment 11

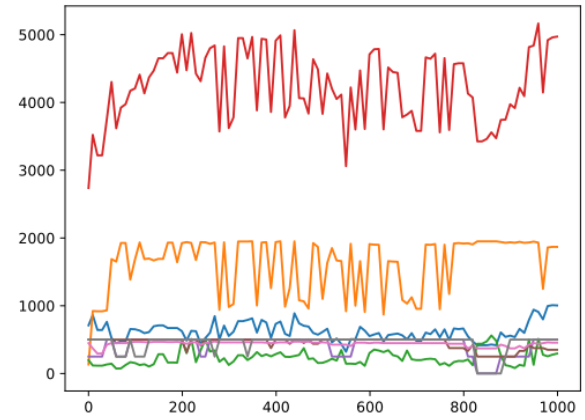
Figure 9: Results of the last five experiments relating to section 4.2. The best image of the last generation is shown



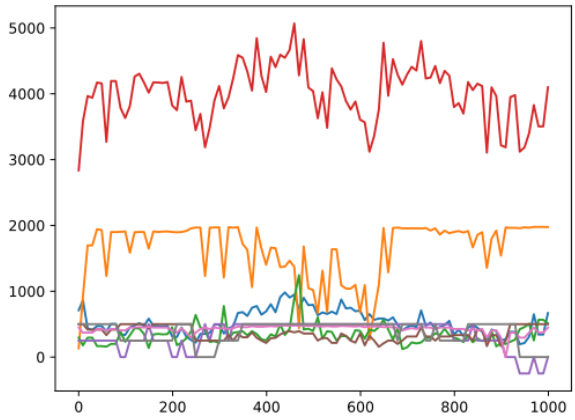
(a) Objective scores of experiment 1



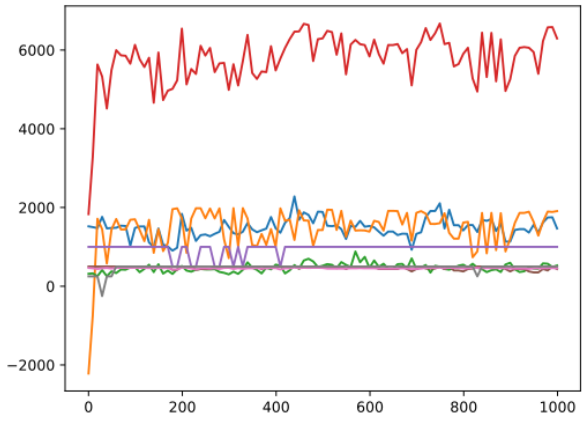
(b) Objective scores of experiment 2



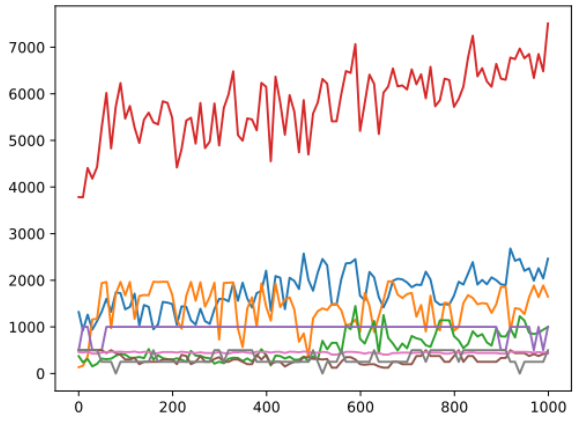
(c) Objective scores of experiment 3



(d) Objective scores of experiment 4

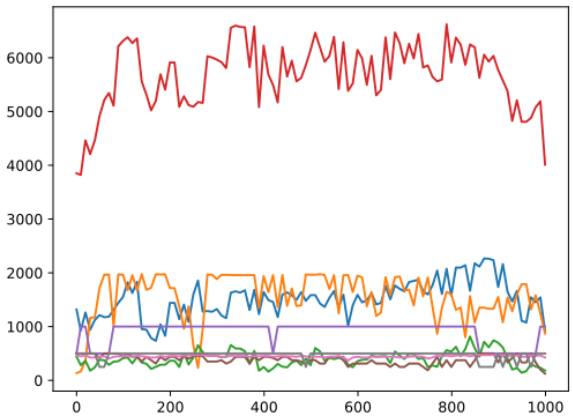


(e) Objective scores of experiment 5

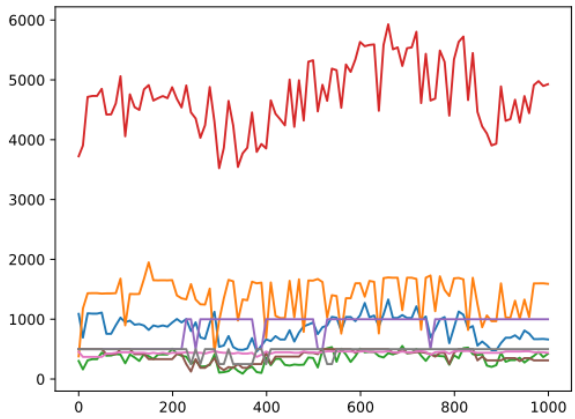


(f) Objective scores of experiment 6

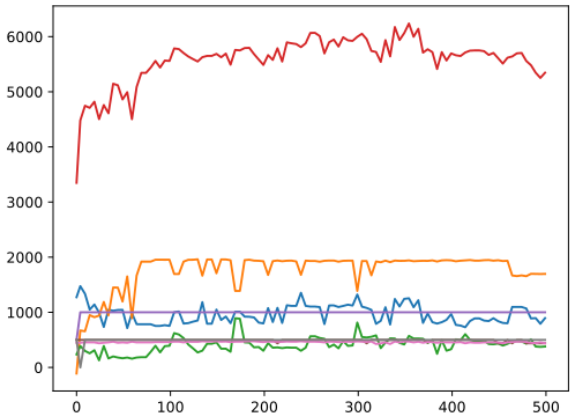
Figure 10: Objective scores of the first six experiments relating to section 4.2. The legend of the graphs can be found in figure 11f



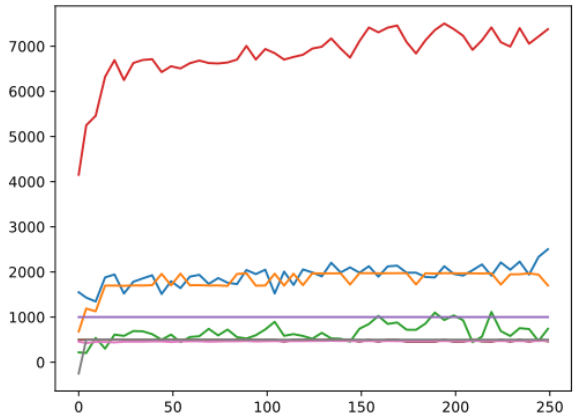
(a) Objective scores of experiment 7



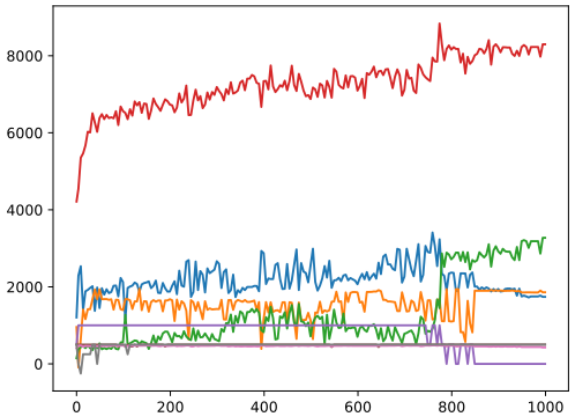
(b) Objective scores of experiment 8



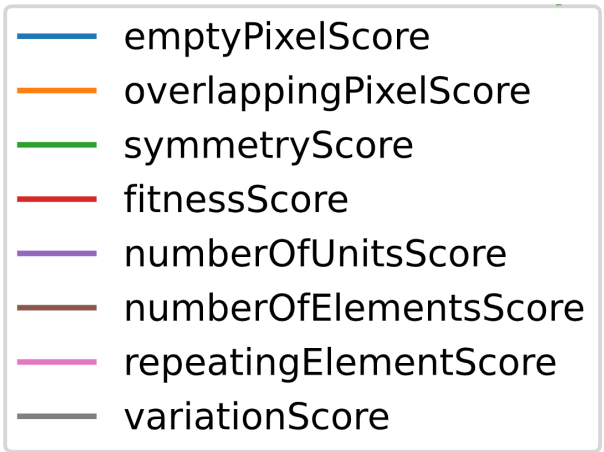
(c) Objective scores of experiment 9



(d) Objective scores of experiment 10



(e) Objective scores of experiment 11



(f) Legend corresponding to all graphs present in figure 10 and figure 11

Figure 11: Objective scores of the last five experiments relating to section 4.2

The difficult part of the symmetry detection function is that we want to be able to detect partial symmetry. We want to reward images that are more symmetric than other images, even though neither are perfectly symmetric. Furthermore, the function can not be computationally heavy, since it is applied in an evolutionary algorithm. One possible solution to an inconsistent symmetry function could be to force symmetry on the generated images. This can be done by creating templates where the positions of the elements and units are forced. This, however, limits the algorithm substantially and would decrease the number of possible outcomes depending on the number of available templates. Also, the research question if a computer could generate an aesthetically pleasing image would only be partially answered, since the end results would be influenced by human preference.
Another possibility would be for the algorithm to focus only on generating units and scoring these units independently. Detecting symmetry would be significantly easier with only four or five elements in an image instead of twenty to twenty-five elements. These generated units can then be used to construct an image. The algorithm would effectively be split into two parts. To see if this would be possible, the symmetry detection algorithm needs to be tested to see if it performs better on less complex images. If not, research needs to be done in other partial symmetry detection heuristics. It would also be possible to create a very simple symmetry function yourself, that only looks at position and shape. If an own symmetry detection algorithm is created, characteristics like size and rotation would possibly need to be removed to keep the algorithm relatively simple. Other characteristics like color would need their own segment in the objective function. When researching into other partial symmetry detection heuristics, it needs to be taken into account that the algorithm can not be computationally heavy. A deep learning method could be used, but for that a database with training data needs to be created.
To answer the second question: *"Can we use the previously defined numerical value to generate images using an evolutionary algorithm?"* an evolutionary algorithm is created. Images are represented in a tree-like structure for easy modification. For selection, tournament selection is used and for crossover, one point crossover is used. When the initial images are generated, the algorithm is given complete freedom and is limited by almost nothing. This is done in the hopes that the objective function will correct faults in the image itself, instead of limiting the algorithm and therefore potentially limit the end result. For example, this means that in the initial generation a lot of overlapping elements can be present. Subsequently, this should result in a low fitness score. A low fitness score means that it is more difficult for this image to survive the selection step of the algorithm. The results of the evolutionary algorithm can be found in figures 8 and 9 with corresponding fitness scores in figures 10 and 11. When looking at the images, it can be observed that most images are not very symmetric. Only figures 8a and 9e have a high symmetry score. For the first image, something strange happened in the objective function causing the symmetry score to be significantly higher than the other scores. The scores can be seen in figure 10a. From this, we would expect the image to be (very) symmetric. However, when looking at figure 8a, this does not seem the case. The elements do seem to be centered along the middle y-axis of the image. Also, all the elements are of similar shape, size and color. But this image is not symmetric. For the next four experiments, the weight for symmetry is kept relatively low and therefore symmetry has close to zero influence on the total score. It seems like most images focus on minimizing the overlap between elements. However, as said before, the symmetry function also influences other characteristics of the images. The positioning of the elements for example. If the symmetry detection function is not working properly, then the only thing that influences the positions of the elements is the

check if the elements are overlapping. Therefore, most positions will be random and the position of elements will have very little correlation to each other. For experiment 6, the objective score for symmetry increases slightly in the second half of the experiment, as can be seen in figure 10f. But when looking at figure 8f, the image shows no trace of symmetry. Most elements are blue, but this could very well be a coincidence. One common similarity between these first six experiments is that the total fitness score fluctuates a lot. When looking at experiment 7, figure 11a, this can be seen even better. The final fitness score is almost the same as the fitness score the algorithm started with. Since no elitism is used in selection, this can happen. Therefore, in experiments 9, 10 and 11 the population size is increased. Another possibility would be to introduce elitism, as discussed in section 3.4. However, this has not been tried for this set of experiments. Increasing the population also causes the execution time of the algorithm to increase. To counter this, the number of generations has been decreased for experiments 9 and 10. These experiments did show less fluctuation in the total fitness and therefore for experiment 11 the number of generations was increased to its original value. When comparing the total fitness score of experiment 11 (figure 11e) to for example experiment 8 (figure 11b) we can see that the fluctuation has decreased significantly. However, the execution time of experiment 11 was significantly longer. Even though experiment 11 was able to reach a higher total fitness than the previous experiments, the end result (figure 9e) of the algorithm is not very aesthetically pleasing. We do see some sort of a pattern emerge with a yellow cross followed by two other yellow shapes and a red shape above the middle yellow shape. However, the rotation of the yellow crosses is already different from each other. Also, to call this pattern aesthetically pleasing would be a stretch. If we look at generation 774 of experiment 11 (figure 12), this pattern can be seen more clearly. One potential reason for the symmetry function to delete the top two units of the pattern is that it is easier for the function to find symmetries in less complex images. Therefore, it will automatically tend to give a higher score to less filled and less cluttered images. This theory is also supported by figure 11e. Just before generation 800, there is a significant increase in the symmetry score. At the same time, as the symmetry score increases, the score for the number of empty pixels on the screen decreases. This theory does contradict the previously mentioned theory that the symmetry function prefers more filled images, since more information is available for the edge detection itself.
In short, the symmetry function seems to be the limiting factor here. Since the symmetry function is not working as desired, each element is not fully judged in relation to the other elements. If two elements are positioned perfectly symmetric, the symmetry function will not reward it maximally. The current way the evolutionary algorithm is set up does allow for potential patterns to emerge, as can be seen in figure 12, but not much more than that. This image has the potential to be aesthetically pleasing but is not quite there yet. Further research needs to be done in other symmetry heuristics to compare performances. Also, it could be interesting to introduce elitism.

5.1 Limitations

Time constraints and Computational resources: Because of COVID-19 this study has been performed from a home office. Therefore, all experiments have also been run from the home office. The computer in the home office lacks computational power in comparison to the computers at Leiden University. Also, the code for the symmetry function that was provided by Elawady et al. [10] was inefficient when combined with the code written for this research. The lack of computational power and inefficient code caused the experiments to take a long time. For example, experiment 11

(figure 9e) took a little over 39 hours to complete. Also, when the experiment was running the home computer could not be used for other purposes. This limited the length and number of experiments substantially.

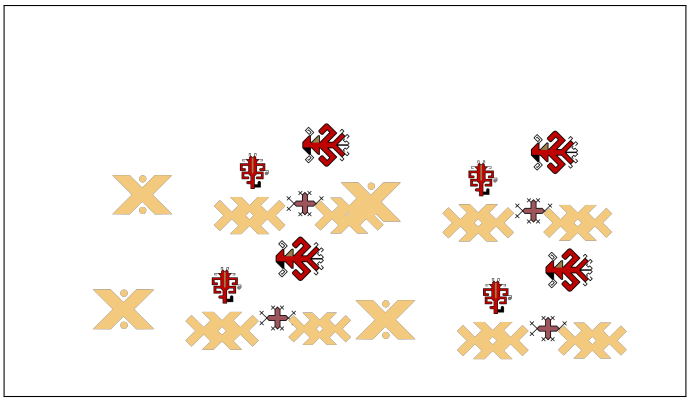


Figure 12: Best result of generation 774 of experiment 11

6 Conclusions and Further Research

This research was conducted to answer two questions: "*Is it possible for a computer to assess the aesthetic quality of an image in a numerical way?*" and "*Is it possible to use the previously mentioned numerical value to create an evolutionary algorithm that generates images?*" For this, an image database from Ornamika [17] was used and images were represented in a tree-like structure. To assess the quality of images, an objective function was created that looks at seven different aspects of the image. A score is given for each different aspect and the total of these scores is the numerical value for each image. Next, this objective function was used to create an evolutionary algorithm that generates these images.
When experimenting with this setup, we noticed that this setup does allow for some sort of patterns to emerge. Even though this happens, we are not yet able to call the generated images aesthetically pleasing. This is mostly caused by the symmetry function 3.3.3 not working as desired. This caused the results of the experiment to be below expectations. Further research needs to be done to use this setup to generate visually appealing images.

6.1 Further research

The algorithm does show potential, but more experimentation needs to be done to generate aesthetically pleasing images. Currently, the objective function is heavily dependent on the symmetry function. However, as said before, the symmetry function performs subpar. It would be interesting to see how the algorithm would perform with a different symmetry function. For this, more research needs to be done on other partial symmetry detection algorithms. Also, it might be possible to split the symmetry function into smaller functions and add them to the objective function. For example, the algorithm could divide the image in a grid-like structure and check for simple symmetry this way. It would also be interesting to look more at the colors used in an image. Color is an important aspect of the aesthetic quality of an image. But currently, there is not a specialized part of the objective

function that judges this aspect. It would be interesting to experiment more on potential interesting qualities of an image and expand the objective function. Moreover, the currently used symmetry function is very inefficient. Splitting this function into multiple smaller parts and integrating that in the code should speed up the algorithm significantly. Therefore, it should be easier to run larger experiments. It would also be interesting to introduce elitism to speed to execution time of the algorithm.

References

[1] Mohammad Majid al Rifaie, Anna Ursyn, Robert Zimmer, and Mohammad Ali Javaheri Javid. On symmetry, aesthetics and quantifying symmetrical complexity. In *International Conference on Evolutionary and Biologically Inspired Music and Art*, pages 17–32. Springer, 2017.

[2] Shumeet Baluja, Dean Pomerleau, and Todd Jochem. Towards automated artificial evolution for computer-generated images. *Connection Science*, 6(2-3):325–354, 1994.

[3] Vic Ciesielski, Perry Barile, and Karen Trist. Finding image features associated with high aesthetic value by machine learning. In *International Conference on Evolutionary and Biologically Inspired Music and Art*, pages 47–58. Springer, 2013.

[4] Chaoran Cui, Peiguang Lin, Xiushan Nie, Muwei Jian, and Yilong Yin. Social-sensed Image Aesthetics Assessment. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 16(3s):1–19, January 2021.

[5] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859. or the Preservation of Favored Races in the Struggle for Life.

[6] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. Studying aesthetics in photographic images using a computational approach. *Computer Vision – ECCV 2006 Lecture Notes in Computer Science*, page 288–301, 2006.

[7] Richard Dawkins. *The blind watchmaker: why the evidence of evolution reveals a universe without design*. Norton, New York, 1996.

[8] Steve DiPaola and Liane Gabora. Incorporating characteristics of human creativity into an evolutionary art algorithm. *Genetic Programming and Evolvable Machines*, 10(2):97–110, June 2009.

[9] Anikó Ekárt, András Joó, Divya Sharma, and Stayko Chalakov. Modelling the underlying principles of human aesthetic preference in evolutionary art. *Journal of Mathematics and the Arts*, 6(2-3):107–124, 2012.

[10] Mohamed Elawady, Christophe Ducottet, Olivier Alata, Cecile Barat, and Philippe Colantoni. Wavelet-based reflection symmetry detection via textural and color histograms: Algorithm and results. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[11] David B. Fogel, editor. *Evolutionary computation: the fossil record*. IEEE Press, New York, 1998.

[12] Khalid Jebari. Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3:333–344, 12 2013.

[13] Leida Li, Hancheng Zhu, Sicheng Zhao, Guiguang Ding, and Weisi Lin. Personality-assisted multi-task learning for generic and personalized image aesthetics assessment. *IEEE Transactions on Image Processing*, 29:3898–3910, 2020.

[14] Yang Li, Changjun Hu, Leandro L Minku, and Haolei Zuo. Learning aesthetic judgements in evolutionary art systems. *Genetic Programming and Evolvable Machines*, 14(3):315–337, 2013.

[15] Jon Mccormack and Andy Lomas. Deep learning of individual aesthetics. *Neural Computing and Applications*, 33(1):3–17, 2020.

[16] Brad L. Miller and David E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113—131, 1996.

[17] Ornamika. Ornamika. https://ornamika.com/, 2021.

[18] Karl Sims. Artificial evolution for computer graphics. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques - SIGGRAPH '91*, pages 319–328, Not Known, 1991. ACM Press.

A Appendix

A.1 Parameter values for the experiments

Experiment	α_1	α_2	α_3	α_4	α_5	α_6	α_7	n	*Number of generations*
Experiment 1	2	2	20	0.5	0.5	0.5	0.5	10	1000
Experiment 2	2	2	**2**	0.5	0.5	0.5	0.5	10	1000
Experiment 3	2	2	**10**	0.5	0.5	0.5	0.5	10	1000
Experiment 4	2	2	**15**	0.5	0.5	0.5	0.5	10	1000
Experiment 5	2	**3**	15	**1**	0.5	0.5	0.5	10	1000
Experiment 6	2	3	**18**	1	0.5	0.5	0.5	10	1000
Experiment 7	2	3	**20**	1	0.5	0.5	0.5	10	1000
Experiment 8	2	3	20	1	0.5	0.5	0.5	10	1000
Experiment 9	2	3	20	1	0.5	0.5	0.5	**20**	**500**
Experiment 10	2	3	20	1	0.5	0.5	0.5	**40**	**250**
Experiment 11	2	3	20	1	0.5	0.5	0.5	40	**1000**

Table 6: All parameters of each experiment described in section 4.2. Bold faced values have changed in comparison to the previous experiment. Unmentioned parameters use default values.