Ministry of Education and Science of Ukraine
NATIONAL UNIVERSITY OF KYIV-MOHYLA ACADEMY
Department of Informatics Faculty of Informatics

Evolutionary art generation using genetic algorithms

Supervisor
Head of the Informatics Department,
associate professor, candidate of physical and mathematical sciences
Horokhovskyi S.S
(signature)
“____” _____ 2021 p.

By student 1st year
Master Degree Program
122 «Computer Science»
Moroz A.V
“____” _____ 2021 p.

Kyiv 2021

Ministry of Education and Science of Ukraine
NATIONAL UNIVERSITY OF KYIV-MOHYLA ACADEMY
Department of Informatics Faculty of Informatics

CERTIFY
Head of the Informatics Department, associate professor, candidate of physical and mathematical sciences
_____ Horokhovskyi S. S.
(signature)
„____”_____2021 p.

PERSONAL ASSIGNMENT
for course work

To student Moroz Andrii Viktorovich faculty of informatics 1-year master program
Thesis «Evolutionary art generation using genetic algorithms»

Output:
Text part content of coursework:
An individual task
Calendar plan
Annotation
Introduction
Chapter 1: Generative and evolutionary art
Chapter 2: Genetic Algorithms
Chapter 3: Image Generation using genetic algorithms
Chapter 4: Practice
Conclusion
References

Issue date „___” _____ 2021 p. Supervisor _____
(signature)
Task received _____
(signature)

Thesis: Euclidean Algorithms for Sound Generation

Calendar plan of coursework execution:

№	Stage name	Deadline	Note
1.	Assignment received	29.10.2020	
2.	Literature lookup	30.10.2020	
3.	Introduction into resources and following development of a draft intro	09.11.2020	
4.	Introduction part written	11.11.2020	
5.	Familiarising with generative and evolutionary art	17.12.2020	
6.	Familiarity with evolutionary art applications development history	8.01.2021	
7.	Development of second and third chapters	25.02.2021	
8.	Implementation of initial genetic algorithm	29.02.2021	
9.	Testing, debugging, and hyperparameter tuning for genetic algorithm	17.03.2020	
10.	Course work amendments according to the supervisor feedback	15.04.2021	
11.	Created final presentation and formatted layout of course work	9.05.2021	
12.	Presentation of the course work	17.05.2021	

Student Moroz A.V.

Supervisor Horokhovskyi S.S.

“_____” _____

Annotation

This course work explains the concept of generative art and especially concentrates on evolutionary art. Evolutionary art is a part of generative art that uses genetic

programming to produce art. Genetic programming for art generation works by creating an initial population, picks out ones that do not fit, and forms new ones combining images that went through selection. Also, this course work is going to explain how to generate images using genetic programming and show results from the developed program.

Introduction

History of the human art dates back to Late Stone Age. As first art forms date back to 73000 years ago. First arts were just an abstract form, for example, hunters getting to their prey. As humanity evolved the art evolved with it. Different branches of art appeared from sculpture to music. But one of the main forms of art still to this day's stays drawing art.

As told previously drawing art got from the form of abstract mammoth images to the beauty of Leonardo da Vinci's "Mona Lisa", back to me abstract forms like Malevich's "Black Square".

Development of the computing technologies is one of the most significant revolutions for humanity, and art could not have combined with computers. Art and science share a long and strong relationship. The best-known example, previously mentioned Leonardo da Vinci, was the greatest arts man of his time as well as scientist and developer. Currently, the bond between science and art is not as strong, but even the computer science pioneers were keen to join two things together. Two legendary personalities of the computer science community: Alan Turing and Ada Byron showed an interest in using computer devices for art-making purposes[1]. Unfortunately, the early interest of Turing and Byron did not find a lot of attention from computer science and artificial intelligence communities.

The main inspiration of artificial intelligence researchers was the human brain, but there is also an interest in biology-inspired computing techniques and evolutionary computation is one of them. It makes sense that nature-inspired models are to work

with art, as it inspired the first artist and inspirational spirit on nature went through the years.

1. Generative and Evolutionary Art

1.1. Start of History of Generative Art

Visual generative art is a subset of the visual arts that creates art using an autonomous framework that is usually described by code. Although it dates back to the 1950s, generative art has gained popularity in recent decades. This is a method for creating new concepts, forms, shapes, colors, or patterns using algorithms. To begin, you must establish rules that define the parameters of the creation process.

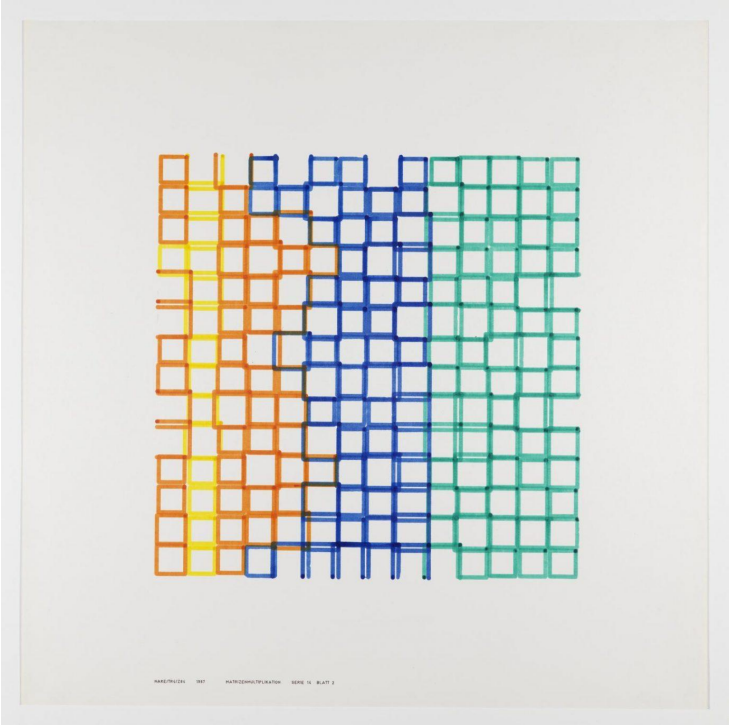
Artists and designers began experimenting with automated devices and analog computers in the late 1950s. This work served as a forerunner to the early digital pioneers' efforts in the 1960s. These early digital pioneers were engineers and scientists, rather than artists or designers, since they had access to more efficient computing tools at university science research labs. The first person to program a digital computer only for artistic purposes was Michael Doll, a technician, and professor at the University of Southern California.

One of the issues that early generative artists with computers faced were the lack of output devices. The plotter, a mechanical device that holds a pen or brush and has its movements controlled by a computer, was the primary source in use at the time. According to the instructions programmed, the computer guides the pen or brush across the drawing surface or alternately moves the paper underneath. The plotter had a linear output, and shading could only be done with crosshatching. As a result, much of the early generative art output focused on geometric forms and structures rather than more fluid content. Early practitioners valued the pure visual

form over content because they saw the computer as an autonomous machine that would allow them to conduct objective visual experiments. Plotter drawings were typically black on white paper, so the majority of early work was black and white, even after printers were introduced. Frieder Nake was one of the first artists to create color plotter drawings.

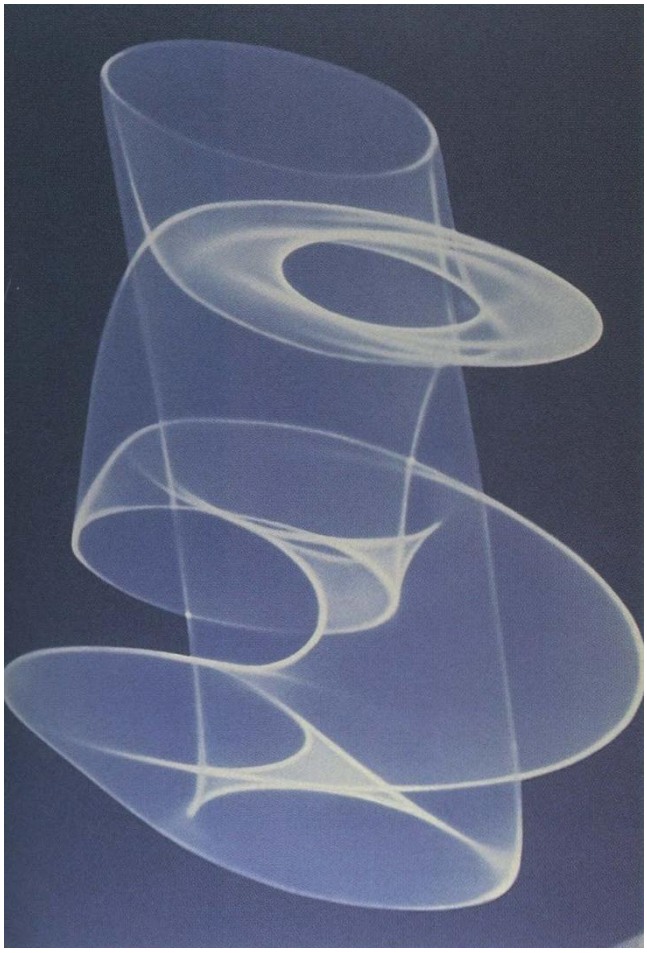
1.2. Generative Art Individuals

Frieder Nake is a mathematician and computer scientist in addition to being a computer art pioneer. Nake was one of the first to exhibit computer art, with his work on display at Stuttgart's Galerie Wendell Niedlich. While some of Nake's works were limited edition silkscreen prints, the majority of his work was done with China ink on paper using a Zuse Graphomat Z64 flatbed high-precision plotter. In the 1960s and 1970s, Nake took part in important group exhibitions, and his book "Ästhetik als Informationsverarbeitung" (1974) was one of the first to examine the connections between aesthetics, computing, and information theory, and has since become a seminal work in the multidisciplinary field of digital media.



Frieder Nake generative art work

Herbert Franke was another early proponent of generative art who came from a scientific background. Frank received his doctorate in theoretical physics in 1950 for a dissertation on electron optics, after studying subjects as diverse as physics, mathematics, chemistry, psychology, and philosophy. Franke experimented with computer art in this capacity, and from 1973 to 1997, he taught a course at Munich University called Cybernetical Aesthetic. Computer Graphics – Computer Art was the new name for the course. Franke was a founding member of Ars Electronica, a Linz-based cultural, educational, and scientific institute focused on new media art, in 1979.

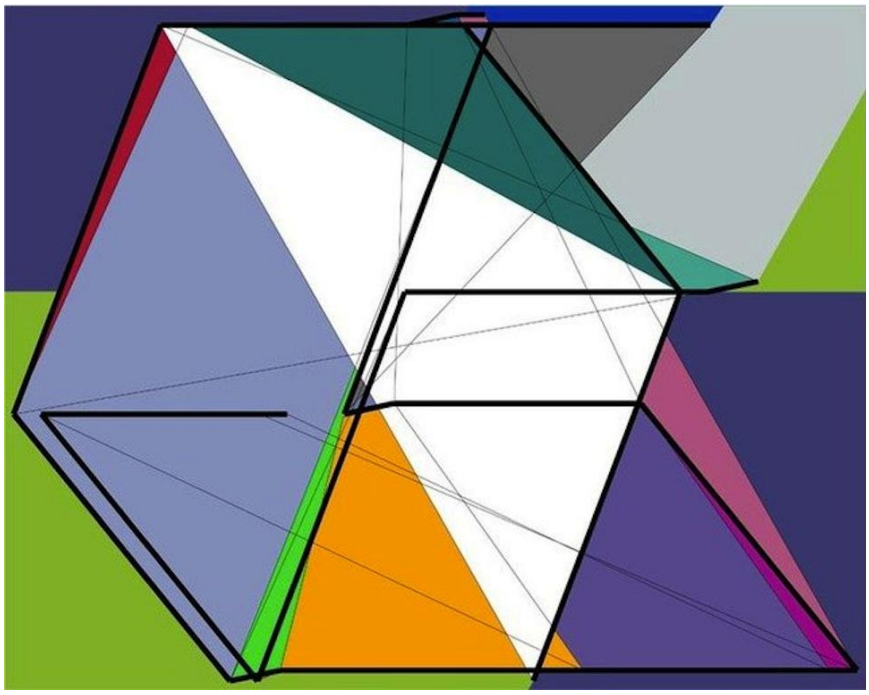


Herbert Franke generative art work

Though scientists made up the majority of early generative artists, this was not always the case. Vera Molnar received her education at the Budapest College of Fine Arts as a traditional artist. She discovered that her style leaned toward the non-representational, and she began experimenting with systematically determined abstract geometrical painting in the 1940s. Molnar set out to learn Fortran and Basic, two early programming languages, after founding the research group Art et Informatique in the 1960s, which investigated the relationship between art and computing. She also gained access to a computer at a research lab in Paris, where she made computer graphic drawings with a plotter.

Vera Molnar generative art work

Manfred Mohr was another pioneer of digital art who came from a fine art background. Mohr, an École des Beaux Arts graduate, was an abstract expressionist action painter who began experimenting with algorithmic art in 1969. His early computer works were based on drawings he had previously created, with a focus on rhythm and repetition.



Manfred Mohr generative art work

1.3. Generative Art Modernization

In the 1970s, generative art started to look outside of the computer laboratories where it was being done and into the wider art world. The term was soon applied to geometric abstract art in which simple elements were repeated, transformed, or varied to generate more complex forms. In this way, the work of Sol LeWitt, John Cage, and Ellsworth Kelly became generative art. The definition of generative art was being expanded not only within computer labs, but also within fine art institutions. The School of the Art Institute of Chicago established the Generative Systems department in 1970, which focused on art practices involving new technologies for image capture, inter-machine transfer, printing, and transmission. Theorists began to define generative art according to their own criteria. Henry Clauser, a theorist, identified the aspect of systemic autonomy as a critical element

in generative art in 1988, pointing out that process (or structuring) and change (or transformation) are among generative art's most defining features, and that these features, as well as the term "generative," imply dynamic development and motion. As people's perceptions of generative art evolved, so did the practice of generative art, and a new generation of artists emerged, bringing with them new ideas and modes of expression.

1.4. Evolutionary Art Early Ideas

The origins of evolutionary art can be traced back to Charles Darwin, a well-known scientist. Natural selection is the foundation of his evolutionary theory. If the population size is not to grow exponentially, selection becomes inevitable in an environment that can only accommodate a limited number of individuals and the basic instinct of individuals to reproduce. Natural selection favors individuals who effectively compete for available resources, i.e., those who are best adapted or fit to the surrounding conditions[2].

Turing came up with the idea of using Darwinian principles for automated problem-solving in 1948. By the 1960s, the first computer experiment on "optimization through evolution and recombination" had been completed. 1st This eventually led to Evolutionary Computing as we know it today[2].

1.5. Evolutionary Art Development

The idea for a program that could evolve "virtual creatures" or biomorphs was first mentioned in Richard Dawkins' book The Blind Watchmaker, published in 1986.[3] After that, Karl Sims created evolutionary art with procedural textures using evolutionary techniques and computer graphics. Genetic parameters were used to grow evolutionary 3D plant structures. Mutating symbolic lisp expressions were used to create evolutionary images, animations, and solid textures [10]. Todd

et al. applied biomorphs to 3D models, making them some of the earliest evolutionary artworks. The method was dubbed the interactive genetic algorithm, and it allowed people to choose mutant individuals manually [4].

1.6. Evolutionary Art Modern Era

With the increasing interest in an expression-based approach, the field of evolutionary art has expanded, providing a coherent and strong foundation for future study. Ibrahim created the Genshade system, which uses wavelet analysis to replicate the characteristics of a target image. The noise-function RenderMan shaders were developed and applied to the Genshade system. Tiago Silva used evolutionary art to create artificial life. An interactive board was used in particular to allow people to interact directly with the agents. Deep learning, neural networks, and convolutional neural networks are just a few examples of efficient and autonomous methods. Hart (2007) took an expression-based approach with the aim of creating a series of images that were distinct from previous works. The interface of this system allowed for more control over the colors and forms of the evolved images. Rooke (2006) used an expression-based approach influenced by aesthetic selection to create complex and informative images, with a focus on the evolution of color space. Unemi (2004, 2012) expanded on the expression-based approach to evolutionary art by developing photographs and animations toward the progression of color volumes and novel forms with a variety of variables. The photos have been viewed on the internet for decades using various iterations of the SBART system and have been developed based on aesthetic steps (Unemi, 1999). Machado and Cardoso (2002) released NEvAr, a computer-aided program. This is a genetic programming, user-guided evolution, and automated fitness assignment evolutionary art tool. The model employs a fitness feature that allows for visually appealing and complex pictures. Electric Sheep, a massive and ongoing

evolutionary art project based on collective human assessment, was initiated by Draves (2005). It was created as a distributed screen saver that allows users to approve or reject phenotypes in order to evolve artificial life, with an emphasis on the distributed system's long-term conduct. Greenfield (2006) describes simulated robots that are placed in an evolutionary environment and are used to create a new piece of art. The method uses optimization to classify robot paintings with higher aesthetic properties while also considering the actions of virtual robots.

Swarm painting has also been the subject of extensive study. Urbano (2006) looked at how a group of agents made decisions in order to create swarm art with fascinating random patterns.

To build data visualizations and to combine information aesthetics with data visualization, a swarm-based framework was used (Maças et al., 2015).

Boyd et al. (2004) collaborated on Swarm Art, a collaborative project that included a swarm-based simulation and projected the artwork onto a large screen. Greenfield (2005) studied the impact of different fitness measures on the generation of different artistic styles using a colony optimization model to evolve ant paintings. Urbano (2005) created collaborative creative work by embedding a pheromone substance for mass recruitment in ants' pattern behavior, which was inspired by natural phenomena.

2. Genetic Algorithms

2.1. How appeared and where genetic algorithms used

The genetic algorithm is a model or abstraction of biological evolution based on Charles Darwin's theory of natural selection, created by John Holland and his collaborators in the 1960s and 1970s (Holland, 1975). Holland is widely credited for being the first to apply crossover and recombination, mutation, and selection to the study of adaptive and artificial systems. The genetic algorithm as a problem-solving technique is incomplete without these genetic operators.

Since then, a variety of genetic algorithm variants have been developed and applied to a variety of optimization problems, ranging from graph coloring to pattern recognition, from discrete systems (such as the traveling salesman problem) to continuous systems (e.g., the efficient design of airfoils in aerospace engineering), and from financial markets to multi-objective engineering optimization.

In comparison to conventional optimization algorithms, genetic algorithms have many advantages. The ability to solve complex problems and parallelism are two of the most notable. If the objective (fitness) function is stationary or non-stationary (changes over time), linear or nonlinear, continuous or discontinuous, or with random noise, genetic algorithms may handle a variety of optimization problems. Since multiple offspring in a population act independently, the population (or any subgroup) will search in multiple directions at the same time. This property makes parallelizing algorithms for implementation a breeze. At

the same time, various parameters and even groups of encoded strings can be modified.

Genetic algorithms, on the other hand, have several drawbacks. The selection criteria for the new population, as well as the formulation of the fitness function, the use of population size, and the choice of important parameters such as the rate of mutation and crossover, should all be done with care. Any incorrect choice will make the algorithm difficult to converge or will simply generate useless results. Despite these disadvantages, genetic algorithms are still one of the most common nonlinear optimization algorithms.

2.2. What are genetic algorithms

Genetic algorithms are randomized search algorithms that were created in an attempt to mimic the mechanics of natural selection and genetics. Genetic algorithms work with string structures, similar to biological structures, that evolve over time using a randomized but structured information exchange to follow the rule of survival of the fittest.

As a result, a new set of strings is generated every generation, using sections of the old set's fittest members. The following are the key features of a genetic algorithm:

(1) Rather than the parameters themselves, the genetic algorithm operates with a coding of the parameter collection.

(2) Rather than starting with a single point, the genetic algorithm starts with a population of points.

(3) Payoff knowledge, not derivatives, is used by the genetic algorithm.

(4) The genetic algorithm employs probabilistic rather than deterministic transformation laws.

The coding that will be used must first be specified. After that, a random method is used to generate an initial population of strings. Then, using a series of operators, this initial population is used to produce subsequent populations, which should improve over time. Reproduction, crossover, and mutation are the three primary operators of genetic algorithms.

Reproduction is a method in which each string's objective function (fitness function) is used. This objective function determines the quality of a string. As a result, strings with a higher fitness value have a higher chance of producing offspring for the next generation.

Crossover occurs when members of the previous population are randomly mated in the mating pool. As a result, a pair of offspring is created, with elements from both parents (members), with the hope of improved fitness values. The mutation is the occurrence of a random change in the value of a string location with a small probability. The mutation is, in reality, a process of taking a random walk through the coded parameter space. Its aim is to prevent valuable data stored inside strings from being lost prematurely.

The application of genetic algorithms to the problem of optimizing operation and site-scale energy usage in manufacturing plants is expected to proceed along the lines outlined below.

The problem specifications and constraints will be specified first, using the pinch design process. The pinch's location will then be determined, and the required matches will be made. The problem can then be coded by creating the required

strings. These strings will include the problem's general characteristics as well as the parameters that influence it. A potential network configuration will be represented by each string. The objective function will be determined for each string, which is the value of energy and utility consumption as well as the number of units required in the represented network configuration, which is the total capital and operating cost. A random process will be used to construct a starting population of strings. The three key genetic algorithm operators will be used to increase the performance of the objective function, which is to construct network configurations with the lowest overall capital and operating costs. Advanced techniques would be applied to the final population of strings/possible networks in order to develop it further.

A genetic algorithm is a type of stochastic algorithm based on probability theory. The search mechanism is defined by a stochastic strategy when this approach is applied to a stagewise superstructure model. With a high degree of certainty, the global optimal solution for the synthesis of heat exchanger networks can be found. The “population” is a group of initial stochastic solutions that the search process starts with. Each solution is referred to as a "chromosome," which is made up of "gene," which stands for the optimal variables of heat exchanger networks, such as the mass flowrates of cold and hot streams.

In the genetic algorithm, there are two types of calculation operations: genetic and evolutionary. The genetic operation uses the probability transferring theory to pick some good chromosomes to spread with a certain probability and to let the other inferior chromosomes die, guiding the search path to the most promising area. They can explore various regions of the search space simultaneously with a stochastic search technique, and thus are less likely to end up in a local minimum.

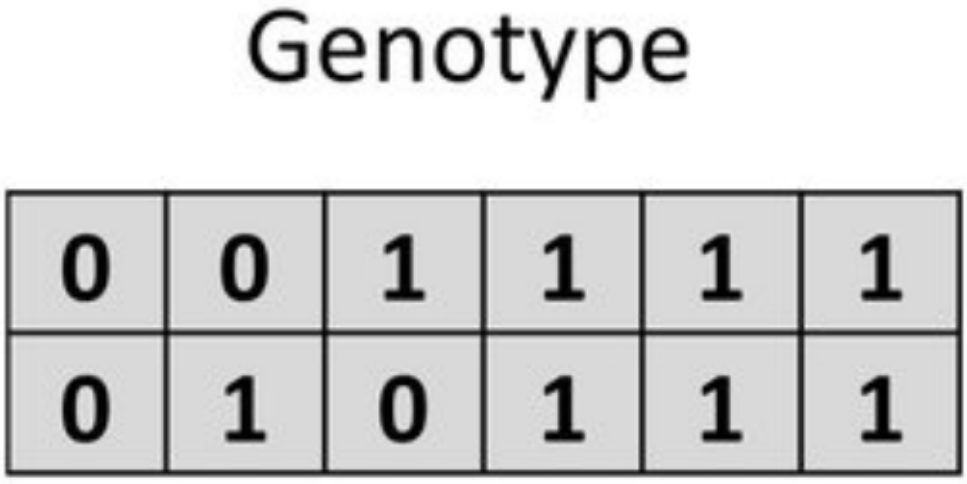
The genetic algorithm's strength is its ability to explore various regions of the search space in a relatively short amount of time. Many and nuanced goals can also

be easily included. However, the genetic algorithm only offers a broad basis for tackling complex optimization problems. The genetic operators are often problem-dependent and are important for efficient application in real-world problems. An approach for initial network generation, heat load determination of a match within the superstructure should be given to the synthesis problem of heat exchanger networks with multistream heat exchangers. Crossover, mutation, orthogonal crossover, and effective crowding operators are some of the operators that are specifically designed to adapt to the synthesis problem. The treatment of constraints is another challenge for genetic algorithm applications. The synthesis problem of heat exchanger networks with multistream heat exchangers should be given an approach for initial network generation, heat load determination of a match within the superstructure. Some of the operators that are explicitly designed to adapt to the synthesis problem include crossover, mutation, orthogonal crossover, and efficient crowding. Another problem for genetic algorithm applications is dealing with constraints.

2.3. Genotype

Breeding, reproduction, and mutation are all aided in nature by the genotype, which is a set of genes organized into chromosomes. If two specimens are bred to produce offspring, the offspring's chromosomes may contain a mixture of genes from both parents.

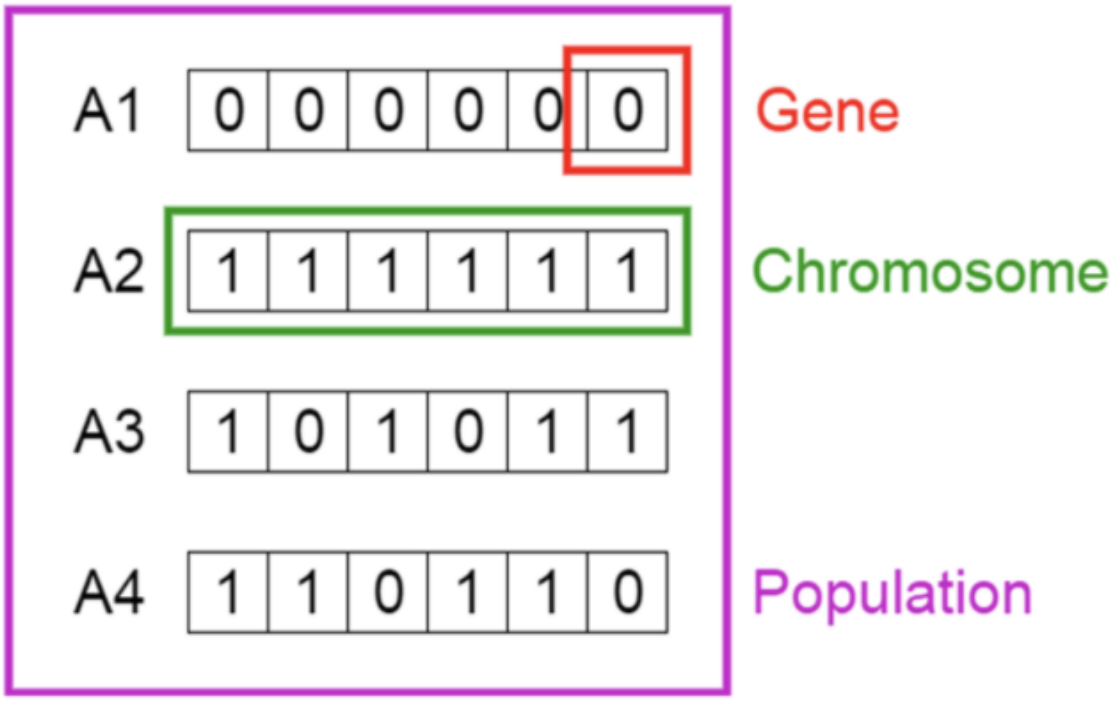
In genetic algorithms, each organism is characterized by a chromosome that contains a set of genes, which is similar to this definition. A chromosome, for example, can be represented as a binary string, with each bit representing a single gene[13]:



Sample of encoded chromosome

2.4. Population

Genetic algorithms keep track of a population of individuals – a list of potential solutions to the problem at hand – at any given time. Since each individual is represented by a chromosome, this population of individuals can be thought of as a set of chromosomes:



A population of individuals represented chromosomes

2.5. Fitness Function

Individuals are measured using a fitness function at each iteration of the algorithm (also called the target function). This is the function or dilemma that we are attempting to optimize.

Individuals with higher fitness scores provide better solutions and are more likely to be selected to replicate and represent the next generation. The consistency of the solutions increases over time, as do the fitness values, and the process can be stopped until a solution with a suitable fitness value is found.

The fitness function reflects the problem that we are attempting to solve. The aim of genetic algorithms is to find individuals that produce the highest score when this function is computed for them.

Genetic algorithms, unlike many conventional search algorithms, only consider the value obtained by the fitness function and do not depend on derivatives or any other details. As a result, they are well suited to dealing with functions that are difficult or impossible to distinguish mathematically.

2.6. Selection

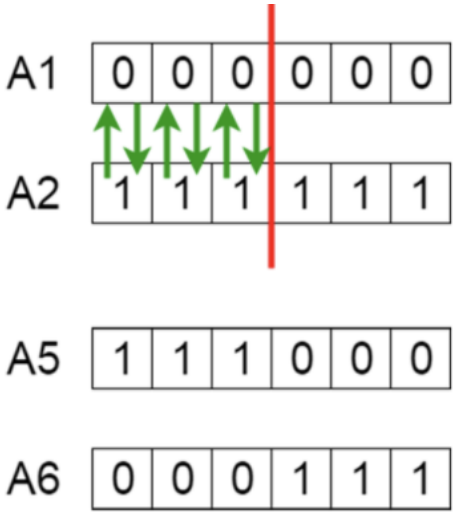
After determining the fitness of each individual in the population, a selection process is used to decide which individuals in the population will reproduce and produce offspring that will form the next generation.

The fitness score of the individuals is used in this selection process. Those with higher score values are more likely to be selected and to pass on their genetic material to the next generation.

Individuals with low fitness values may still be selected, but with a lower likelihood. As a result, their genetic material is not entirely removed.

2.7. Crossover

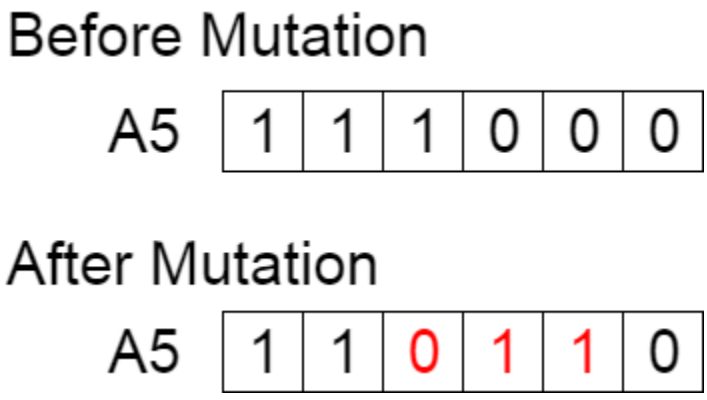
To produce a pair of new individuals, two parents from the current generation are normally selected, and parts of their chromosomes are swapped (crossed over) to create two new chromosomes representing the offspring. This process is known as crossover or recombination:



Crossover operation between two binary chromosomes

2.8. Mutation

The mutation operator's aim is to refresh the population on a regular and random basis, introduce new patterns into the chromosomes, and promote search in uncharted areas of the solution space. A mutation may appear as a random change in a gene. Mutations are carried out as random changes to one or more chromosome values, such as flipping a bit in a binary string:



Example of mutated binary encoded chromosome

2.9. The schema theorem

Holland's schema theorem, also known as the fundamental theorem of genetic algorithms, is a more formal expression of the building-block hypothesis.

This theorem is about schemata (plural of schema), which are patterns (or templates) contained in chromosomes. Each schema represents a subset of chromosomes that are related in some way.

If the set of chromosomes is expressed by binary strings of length four, the schema 1*01 represents all chromosomes with a 1 in the leftmost position, 01 in the rightmost two positions, and either a 1 or a 0 in the second from the left position, since the * represents a wildcard value.

There can be assigned two measurements to each schema in the following order:

- The total number of fixed digits (not wildcards)
- Determining length: The interval between the two most distant fixed digits

Schema	Order
***	0
101	3
*11	2
1**	1

Example of Holland's schema theorem

In the same way that a given string matches regular expressions, each chromosome in the population corresponds to multiple schemata. A chromosome with a higher score is more likely to survive the selection operation, as are all the schemata it

serves. As this chromosome crosses over with another or is mutated, some schemata will survive and others will perish. Low-order schemata with short defining lengths are more likely to survive.

As a result, the schema theorem states that in successive generations, the frequency of schemata of low order, short defining length, and above-average fitness increases exponentially. In other words, as the genetic algorithm advances, the smaller, simpler building blocks that represent the characteristics that make a solution better will become more prevalent in the population. In the following section, we will look at the differences between genetic and conventional algorithms.

2.10. Distinction from traditional algorithms

There are some main distinctions between genetic algorithms and conventional search and optimization algorithms such as gradient-based algorithms.

The following are the main characteristics that differentiate genetic algorithms from conventional algorithms:

- Keeping a population of solutions
- Using a genetic model of the solutions
- Using the results of a fitness test
- Displaying probabilistic action

2.10.1. Basing on population

Rather than a single candidate, the genetic quest is performed through a population of candidate solutions (individuals). At every point during the hunt, the algorithm

keeps a group of people who make up the current generation. The next generation of individuals is generated with each iteration of the genetic algorithm.
Most other search algorithms, on the other hand, retain a single solution and iteratively change it in search of the best solution. For example, the gradient descent algorithm iteratively moves the current solution in the direction of the steepest descent, defined by the negative of the given function's gradient.

2.10.2. Genetic model representation

Rather than directly operating on candidate solutions, genetic algorithms work on their representations (or coding), which are often referred to as chromosomes. A fixed-length binary string is an example of a simple chromosome.
The chromosomes allow us to speed up the genetic processes of crossover and mutation. Crossover occurs when chromosome parts from two parents are exchanged, while mutation occurs when chromosome parts are modified.
The use of genetic representation has the unintended consequence of decoupling the quest from the original problem domain. Genetic algorithms are oblivious to what the chromosomes mean and make no effort to interpret them.

2.10.3. Probabilistic behavior

Although many conventional algorithms are deterministic in nature, genetic algorithms use probabilistic rules to progress from one generation to the next.
When choosing the individuals that will be used to produce the next generation, for example, the likelihood of selecting a given individual increase with the individual's fitness, but there is still a random factor in making that choice. Individuals with low score values can still be selected, but with a lower likelihood.
The mutation is also a probability-driven process that results in shifts at random locations on the chromosome.

The crossover operator may also include a probabilistic component. Crossover can only occur with a certain probability in some variations of genetic algorithms. If there is no overlap, all parents are duplicated into the next generation without shift. Despite its probabilistic nature, a genetic algorithm-based search is not random; rather, it uses the random aspect to guide the search toward areas in the search space where there is a better chance of improving the results.

2.11. Benefits of genetic algorithms

2.11.1. Resilience to noise

Some issues exhibit raucous activity. This implies that, even for similar input parameter values, the output value can vary slightly from measurement to measurement. This can happen, for example, when the data is read from sensor outputs, or when the score is based on human opinion, as described in the previous section.

While this type of behavior can cause problems for many conventional search algorithms, genetic algorithms are typically resistant to it due to the repetitive operation of reassembling and reevaluating the individuals.

2.11.2. Parallelism

Parallelization and distributed computing are well suited to genetic algorithms. Since fitness is determined separately for each person, all individuals in the population may be measured simultaneously.

Furthermore, the operations of selection, crossover, and mutation can all be conducted on individuals and pairs of individuals in the population at the same time.

As a result, genetic algorithms are a perfect candidate for distributed as well as cloud-based implementation.

2.11.3. Continuous learning

Evolution is a never-ending process of nature. The population will adjust to changing environmental conditions. Similarly, genetic algorithms can run continuously in an ever-changing world, and the best current solution can be fetched and used at any point in time.

To be efficient, environmental changes must be slow in comparison to the generation turnaround rate of the genetic algorithm-based search.

2.12. Disadvantages of genetic algorithms

2.12.1. Hyperparameter tuning

A number of hyperparameters, such as population size and mutation rate, govern the behavior of genetic algorithms. There are no exact rules for making these choices when applying genetic algorithms to the problem at hand.

However, this is true of almost all search and optimization algorithms.

2.12.2. Computationally Intensive

Working with (potentially large) populations and the repetitive nature of genetic algorithms can be computationally intensive and time-consuming before achieving a successful result.

These can be mitigated by selecting appropriate hyperparameters, introducing parallel processing, and, in certain cases, caching intermediate performance.

2.12.3. Premature convergence

If the health of one person is significantly greater than that of the rest of the population, it can be duplicated to the point where it takes over the entire population. This can cause the genetic algorithm to get trapped in a local limit rather than the global one. To keep this from happening, it is important to keep the population diverse.

3. Genetic Algorithms for image generation

3.1. Genetic algorithm plan

- Data representation
- Initial Population
- Fitness Calculation
- Parent Selection
- Crossover
- Mutation

3.2. Data representation

For the purposes of the program created in this course work, we are using the 2-dimensional array, because the program works with 2D images. This can be easily done by converting the 2-dimensional array into one dimension. Matrix of image rows should be merged into one row. To return a 1-dimension array into 2-dimensional array it is essential to know the size of the image. By knowing the size of the image we can split a 1-dimension array back to the needed number of rows, in our case 3 because of the RGB.

3.3. Initial population

Genetic algorithms always begin with an initial population, which is a collection of solutions (chromosomes) to the problem at hand. So initial, the random population should be generated. Population numbers should be tuned by experimenting with genetic algorithms. The first population will give poor performance as it is totally random.

3.4. Fitness function

As mentioned previously genetic algorithms start with poor solutions from the initial population. Genetic algorithms are based on the constant evolution of the best individual, where the best individuals are chosen by the fitness function. In our case, we get deltas of colors at first and then measure the distance between the colors. The lower the result the better.

3.5. Crossover

Creating a new individual for our population by taking part of genes from the first parent and another part from the second in the idea of creating a better solution. We still should save the parents if offspring get only bad genes from parents.

3.6. Mutation

Operation of gene mutation in the offspring. Gene mutation rate should be tested and fitted not to create too much randomness in the process, but also to diversify the population.

4. Practice

4.1. The Application

The application developed in the process of creating the coursework is an evolutionary image generator. It takes an initial image as an input and uses polygon object and genetic algorithms to reproduce the initial image as close as possible.

4.2. Tools

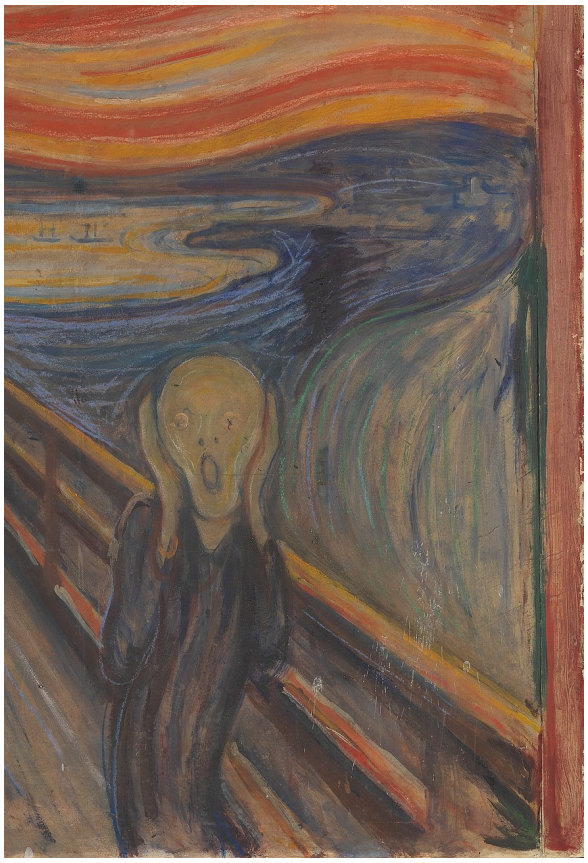
For development purposes as the main development language was chosen Python, for flexibility of the language, ease of development, and personal preference of the author. No external frameworks were used for the implementation of genetic algorithms. The author used mostly internal libraries of the language chosen but also used Pillow library for image processing purposes.

4.3. Application interface

The application has a simple command-line interface where the user should run the app using python and provide a path to the image that should be replicated.

4.4. Application use example

For demonstration purposes, the author is going to use the famous drawing of Edvard Munch “The scream”.



“The Scream” by Edvard Munch

The application should be provided by the image of the drawing.

```
→ python3 src/generate.py img/scream.webp
```

Then it creates an initial population.

Initial population of the image

As expected, the first population is just a group of polygons. As evolution continues, the population starts to replicate the forms of the image.

200th generation

Image replication continues and right now is limited by number of generations, but fitness function output can also be implemented as a stopper.

2000th generation

Conclusion

This course work gave the author deep knowledge of evolutionary and generative art, its history, development and modern time state. Influential thing was an interest of early pioneers of computer science like Alan Turing in the evolutionary art. The author got familiar with basic concepts of evolutionary programming and benefits and drawbacks of it.

While implementing the application author learned to implement genetic algorithms and experiment with them while tuning for hyperparameters of the algorithms of generation. As well, the author might use genetic algorithms in other spheres of computer science.

References

1. J. Romero, P. Machado , The Art of Artificial Evolution, Coimbra, August 2007
2. A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer-Verlag, Berlin, Heidelberg, 2003
3. Müller J., Evolutionary Art, 23 April 2012
4. Stephen T.; William, L., Mutator: A Subjective Human Interface for Evolution of Computer Sculptures; IBM United Kingdom Scientific Centre: Winchester, UK, 1991.
5. Ibrahim A.E.M. Genshade: An Evolutionary Approach to Automatic and Interactive Procedural Texture Generation; Texas A&M University: College Station, TX, USA, 1998.
6. Hart D., Toward greater artistic control for interactive evolution of images and animation. In Applications of Evolutionary Computing, 2007
7. Rooke S., The evolutionary art of Steven Rooke, 2006
8. Unemi T., SBART 2.4: Breeding 2D CG images and movies and creating a type of collage, 1999
9. Unemi T., Embedding movie into SBART—Breeding deformed movies, 2004
10. Unemi T., SBArt4 for an automatic evolutionary art, 2012
11. Machado P., Cardoso A., All the truth about NEvAr, 2002
12. Greenfield G., Robot paintings evolved using simulated robots, 2006
13. Wirsanky E., Hands-On Genetic Algorithms with Python, January 2020
14. Greenfield G., Evolutionary methods for ant colony paintings, 2005
15. Urbano, P., Playing in the pheromone playground: Experiences in swarm painting, 2005

16. Urbano, P., Consensual paintings, 2006

17. Maçãs C., Cruz P., Martins P., and Machado P., Swarm systems in the visualization of consumption patterns, 2015

18. Boyd J., Hushlak G., Jacob C., Swarmart: Interactive art from swarm intelligence, 2004

19. Holland, J., Adaptation in Natural and Artificial Systems, 1975