Unexpected Patterns Generated while Attempting the Design of an Algorithm for Generating Natural Fractal Objects Using Evolutionary Computation

Habiba Akter [1], Rupert Young [2]
Department of Engineering and Design,
School of Engineering and Informatics,
University of Sussex,
United Kingdom
[1] h.akter@sussex.ac.uk
[2] r.c.d.young@sussex.ac.uk

Abstract

This paper presents results and observations from initial experiments with an aim to implement Evolutionary Computation to generate realistic fractal objects. The target fractals are both regular and irregular, commonly observed in nature.

Keywords: *Fractal Objects, Evolutionary computation, Genetic Algorithm, Iterated Function Systems.*

Introduction

Despite being efficient in explaining different patterns, the classic geometry cannot explain the complex structures (Frame, Michael and Urry, Amelia, 2016). In this case, the fractal geometry plays an important role to categorise and explain them mathematically (Mandelbrot and Mandelbrot, 1982; Campbell and Abhyankar, 1978). Unlike the simple structures, the dimension of fractals are fractional numbers (Uthayakumar and Prabakar, 2012). Fractal structures are very often seen in the biological organism, specially in plants. Widely used examples are the Barnsley fern, the roughness of the coastline, strokes of clouds, mountain ranges, mammalian lungs etc (Brambila, 2017). The fractal objects can be mainly divided into the following two categories (Bayırlı et al., 2014; Hutchinson, 1981):

1. **Regular fractals:** This category includes the objects which are self-similar. If they are zoomed in, at every scale, they look similar to their original shapes. They are also known as geometric fractals.

2. **Irregular fractals:** These structures do not have the feature of self-similarity property. They are also known as non-geometric fractals.

We propose to design a tool using an Evolutionary Algorithm (EA) to generate both regular and irregular fractals.

Main Objectives

The main objectives of this work are set as follows:

- To generate natural fractals using a GA, without relying on the popular reverse-calculation methods.

- To use the tool for generating both regular and irregular fractals which resemble complex patterns observed in biological organisms.

Selection of Algorithm

The interest in the paradigm of "Evo-Devo" i.e., the evolutionary developmental biology has been the main motivation of this work. The iterative nature of fractals inspired the use of evolutionary computation to generate images of fractal structures (Lauwerier and Lauwerier, 1991; Collet et al., 2000). The development of computers has helped generating fractal structures. One of the very first successful computer-generated fractal image is the Mandelbrot's set as shown in Figure 1(Mandelbrot et al., 2004).

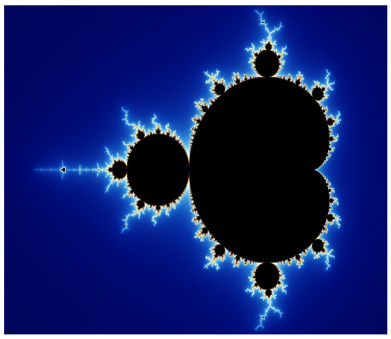

Figure 1: Computer-generated Mandelbrot's Set

The idea of using the evolutionary computation in studying fractals is not new (Lutton, 1999; Collet et al., 2000). The work in (Collet et al., 2000) discusses a new technique for treating the inverse problem for Iterated Functions Systems (IFS) for generating fractals using Genetic Programming (GP). This paper is based on the idea of implementing a Genetic Algorithm (GA) for searching the parameters of natural fractal objects. Originally developed by John Holland, the GA is one of the most frequently used EAs by the researchers (Mitchell, 1998; Sarker et al., 2002; Slowik and Kwasnicka, 2020; Katoch et al., 2021). Researchers have also justify the idea of implementing Genetic Algorithm (GA) and fractals together (Véhel and Lutton, 1993; Lutton, 1999; Bossard et al., 2016). We are not only interested in the regular fractals, but also irregular fractals which are strikingly similar as the ones observed in real life.

Proposed Design

We aim to design a fractal-generating tool that outputs the parameters by exploring the search space. These are the GA-tuned parameters to be used in fractal functions. Figure 2 presents a flowchart of the steps involved in the GA (Mitchell, 1998).

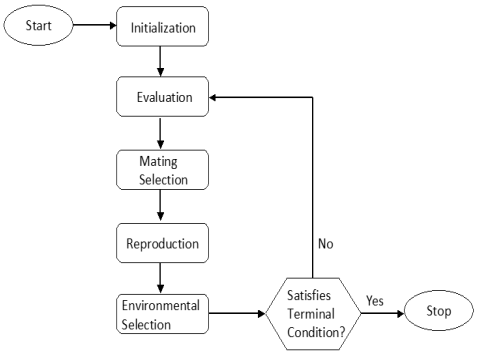

Figure 2: The steps in a Genetic Algorithm

However, the implementation is not simple as at each step, the parameters are to be selected very carefully. This is mainly because we wish to generate realistic patterns. At the first step, a set of population, P is generated. The members of P are known as "chromosomes" which are made with genes. For our algorithm, the genes represent the values to be used as the coefficients of mathematical functions to generate fractals. If n is the number of genes in a chromosome, then a chromosome of the population P can be represented using Figure 3.

g_1	g_2	g_3	g_4	.....	$g_{(n-1)}$	g_n

Figure 3: The steps in a Genetic Algorithm

Each chromosome is then evaluated using a fitness function. To do so, we have used "fractal dimension" which is a well-known method for measuring the complexity of fractal structures. There are different methods for calculating the fractal dimension of an image (Fernández-Martínez and Sánchez-Granero, 2014a,b; Falconer, 2004; Husain et al., 2021). We have used the Box-Counting Dimension method for evaluation.

Equation 1 calculates the fractal dimension FD, which represents the fitness of a chromosome, F:

$$F = \lim_{\epsilon \to 0} \frac{log\, n}{log\, \frac{1}{\epsilon}} \tag{1}$$

Here, n is the number of boxes covering the points of an image and ϵ is the size of boxes. The range of ϵ needs to be pre-selected.

The fitter chromosomes from P are selected for reproduction, which undergo the process of crossover for a certain probability, ρ_c and produce offspring. We implemented single-point crossover, where two parent chromosomes are selected and crossed over.

Similarly, for mutation probability of ρ_m, a certain number of chromosomes undergo mutation to produce offspring. A gene from the parent is randomly selected randomly and its value is altered.

The offspring from both crossover and mutation are then combined together to be sent as the population set, P for the next iteration. This goes on until the algorithm meets the terminating condition. We have run the tests setting up a maximum number of iterations as the terminating condition.

Parameters	Notation
Size of Population	N
Crossover probability	ρ_c
Mutation probability	ρ_m
Crossover rate	r_c
Mutation rate	r_m
Terminating condition	i_{max}

Table 1: The parameters to be selected for the Genetic Algorithm to develop the proposed algorithm to generated fractals

Experiments and Observations

We took the example of the Barnsley fern to compare with. Barnsley came up with the Iterated Function Systems (IFS) based on his "Collage Theorem" and an iteration algorithm "decodes" the data back to images again. IFS generates fractals using affine transformations (Barnsley and Demko, 1985; Barnsley, 2014; Barnsley et al., 2003). These affine transformations for two dimensional (2-D) fractals can be presented by the form shown in Equation 2.

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{2}$$

For better explanation, if we denote the coefficients a through d as v_i, then Equations 3 through 6 calculates the affine transformation functions, f_1 to f_4:

$$x_{n+1} = v_0 \times x_n + v_1 \times y_n + e$$
$$y_{n+1} = v_2 \times x_n + v_3 \times y_n + f \quad (3)$$

$$x_{n+1} = v_4 \times x_n + v_5 \times y_n + e$$
$$y_{n+1} = v_6 \times x_n + v_7 \times y_n + f \quad (4)$$

$$x_{n+1} = v_8 \times x_n + v_9 \times y_n + e$$
$$y_{n+1} = v_{10} \times x_n + v_{11} \times y_n + f \quad (5)$$

$$x_{n+1} = v_{12} \times x_n + v_{13} \times y_n + e$$
$$y_{n+1} = v_{14} \times x_n + v_{15} \times y_n + f \quad (6)$$

Each equation is selected for a certain percentage. Table 2

Functions	Percentage
f_1 (Equation 3)	1%
f_2 (Equation 4)	85%
f_3 (Equation 5)	7%
f_4 (Equation 6)	7%

Table 2: Probability of choosing each of the four affine transformations in the IFS to generate the Barnsley fern

The proposed GA evolves the values of the coefficients, v_i. The lower limit and upper limit of v_i are denoted as v_{min} and v_{max} in the rest of the paper.

Experiments with Different Parameters

At first, we paid attention to the initial population set to start running the tests. Two different scenarios have been considered for this.

Table 3 lists the parameters set for these tests.

Parameters	Values
Size of initial Population, N	100
Crossover probability, ρ_c	0.7
Mutation probability, ρ_m	0.2
Crossover rate, r_c	0.5
Mutation rate, r_m	0.02
Terminating condition	i_{max}

Table 3: The parameters to be selected for the Genetic Algorithm to develop the proposed algorithm to generated fractals

Evolving four Coefficients, a, b, c and d: The first experiment is set to generate the coefficients a to d for the four affine transformation, f_1 to f_4. The variable are within the range of $v_{min} = -1$ to $v_{max} = 1$.

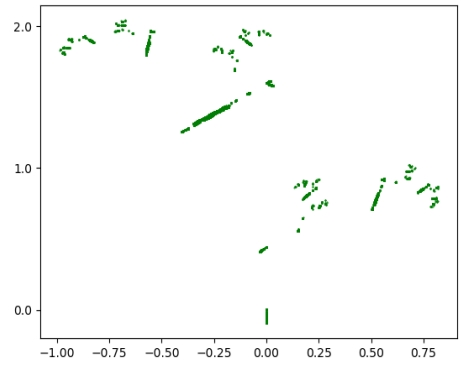

Figure 4: Image generated after the 200th iteration

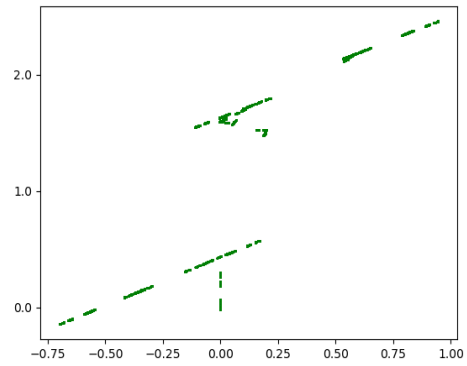

Figure 5: Image generated after the 500th iteration

It is clear that the idea of generating all the coefficients within the same range is not feasible. As we can see even if we increase the number of iterations, the images are far from realistic.

Evolving two coefficients, c and d: The genes in each chromosome are randomly generated within the range of -1 to 1. The image generated after 500 iterations is shown in Figure 6.

We continued the iterations since, with the increase in the number of iterations, the parameters are evolved and may give better results. However, no improvement was noticed and hence we stopped the rum after the 1000th iteration. Figure 7 includes the result image.

It is clear that even after a thousand iterations, the GA still cannot generate a real life-like fractal fern when the range of

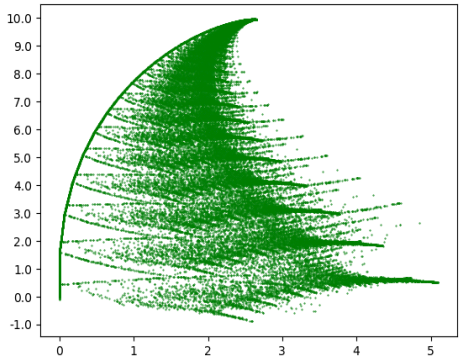

Figure 6: Image generated after the 500$^{\text{th}}$ iteration

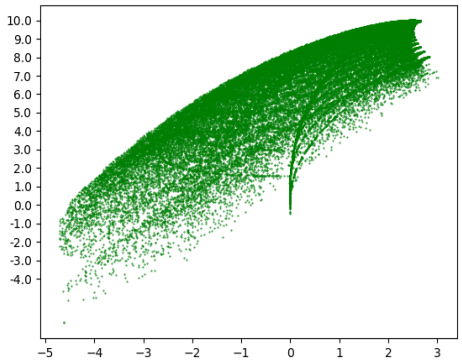

Figure 7: Image generated after the 1000$^{\text{th}}$ iteration

the values for each gene is set within -1 and 1.

Experiments with Altering the Probability of using the Transformation Functions

We decided to run some tests with different probability of choosing the affine transformations. Our plan was to generate that randomly as well. But before that, we had run the IFS only by altering the probabilities of choosing f_1, f_2, f_3 and f_4. Table 4 has the percentages for this set of experiments.

Functions	Percentage
f_1 (Equation 3)	1%
f_2 (Equation 4)	8%
f_3 (Equation 5)	12%
f_4 (Equation 6)	79%

Table 4: Probability of choosing each of the four affine transformations in the IFS to run the experiments of Section to generate the Barnsley fern

Figure **??** has the image generated using the functions in the IFS code for 70000 iterations.

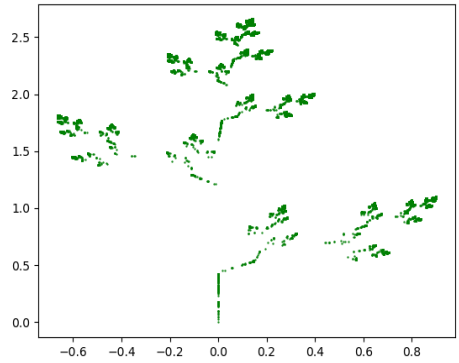

Figure 8: Image generated after the 70000$^{\text{th}}$ iterations of the IFS for fern

From this experiment it is clear that the probability of choosing the transformation functions of an IFS to generate a self-similar fractal has to be chosen very carefully. If not, the output will still be a self-similar pattern, but not the "target" image.

Experiments with Fractal Optimisation Method

The box-counting dimension is optimised in order to generate an accurate fractal. At first, we selected a wide range for the scaling of the boxes. We selected the range to be within -0.8 to 1.

Figure 9 and Figure 10 include the images generated from the 25$^{\text{th}}$ and 50$^{\text{th}}$ iteration.

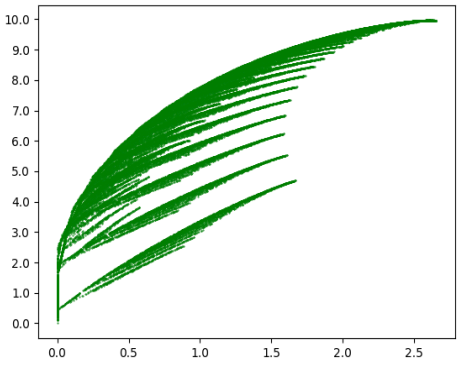

Figure 9: Image generated after the 25$^{\text{th}}$ iteration

To see if the increment of iterations outputs better results, we continued the run until it reaches the iteration of 100. Figure 11 includes the final output image.

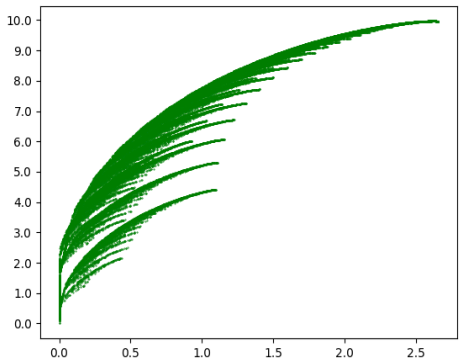

Figure 10: Image generated after the 50th iteration

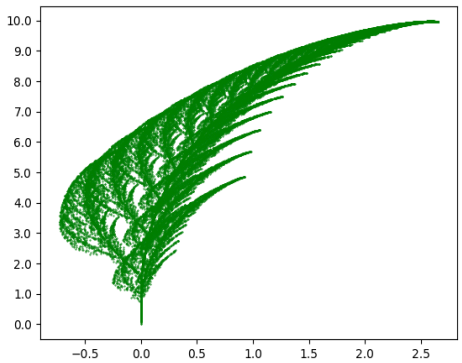

Figure 11: Image generated after the 100th iteration

This set of tests proves that with a big range of size of boxes ϵ, even after hundreds of iterations, the fractals are not optimised. There is some self-similarity, but the images do not resemble the real fern.

The graph in Figure 12 includes the box counting dimension calculated for all the 100 images generated.

It is clear from the graph that the values are not optimised with the increase in the iterations. The values are also changed in random order rather than ascending as assumed.

Conclusions and Discussion

Objectives Revisited

- To generate natural fractals using a GA, without relying on the popular reverse-calculation methods.

- To use the tool for generating both regular and irregular fractals which resemble complex patterns observed in biological organisms.

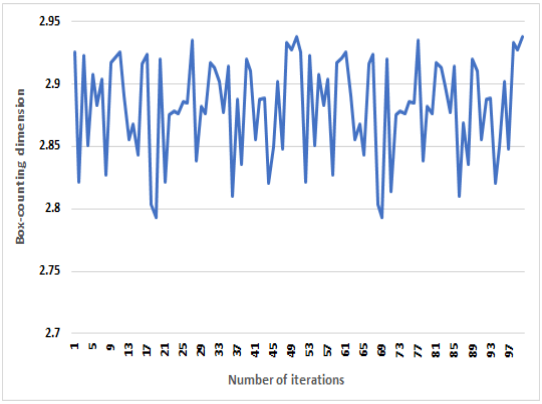

Figure 12: graph showing the box counting dimension of attempted 100 images of fern

Concluding Comments

This work presents the results of some initial experiments to achieve the objectives mentioned. After careful observations, we have come to a conclusion the Genetic Algorithm requires a minimum estimation of ranges before the initialisation of population. A large range of values for the coefficients of an Iterated Function System will only generate some random patterns without any fractal characteristics. A completely random choice of the probability of using these functions in the code of the Iterated Function System will generate a self-similar pattern but not resembling a natural fractal. The method of calculating the fractal dimension from the values evolved by the Genetic Algorithm is also not very simple. A bigger size of box gives a fractional value of the dimension. Although this is a criteria of a fractal object, the images generated are not real-life like.

Future Works

The experiments described in this paper have been helpful to revisit the initial objectives. Later we have been able to design an algorithm using a standard Genetic Algorithm. But it was clear from these experiments that we need to select the parameters very carefully if we want to generated a fractal object that resembles a biological organisms.

Acknowledgements

This work was funded by the Leverhulme Trust Research Project Grant RPG- 2019-269 which the authors gratefully acknowledge.

References

Barnsley, M., Hutchinson, J. E., and Stenflo, Ö. (2003). V-variable fractals and superfractals. *arXiv preprint math/0312314*.

Barnsley, M. F. (2014). *Fractals everywhere*. Academic press, Boston.

Barnsley, M. F. and Demko, S. (1985). Iterated function systems and the global construction of fractals. *Proceedings of the*

Royal Society of London. A. Mathematical and Physical Sciences, 399(1817):243–275.

Bayırlı, M., Selvi, S., and Çakılcıoğlu, U. (2014). Determining different plant leaves' fractal dimensions: a new approach to taxonomical study of plants.

Bossard, J. A., Lin, L., and Werner, D. H. (2016). Evolving random fractal cantor superlattices for the infrared using a genetic algorithm. *Journal of the Royal Society Interface*, 13(114):20150975.

Brambila, F. (2017). *Fractal analysis: applications in physics, engineering and technology*. BoD–Books on Demand.

Campbell, P. and Abhyankar, S. (1978). Fractals, form, chance and dimension.

Collet, P., Lutton, E., Raynal, F., and Schoenauer, M. (2000). Polar ifs+ parisian genetic programming= efficient ifs inverse problem solving. *Genetic Programming and Evolvable Machines*, 1(4):339–361.

Falconer, K. (2004). *Fractal geometry: mathematical foundations and applications*. John Wiley & Sons.

Fernández-Martínez, M. and Sánchez-Granero, M. (2014a). Fractal dimension for fractal structures. *Topology and its Applications*, 163:93–111.

Fernández-Martínez, M. and Sánchez-Granero, M. (2014b). Fractal dimension for fractal structures: A hausdorff approach revisited. *Journal of Mathematical Analysis and Applications*, 409(1):321–330.

Frame, Michael and Urry, Amelia (2016). *Fractal worlds: Grown, built, and imagined*. Yale University Press, New Heaven and London.

Husain, A., Reddy, J., Bisht, D., and Sajid, M. (2021). Fractal dimension of coastline of australia. *Scientific Reports*, 11(1):1–10.

Hutchinson, J. E. (1981). Fractals and self similarity. *Indiana University Mathematics Journal*, 30(5):713–747.

Katoch, S., Chauhan, S. S., and Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126.

Lauwerier, H. and Lauwerier, H. A. (1991). *Fractals: endlessly repeated geometrical figures*. ICON Group International.

Lutton, E. (1999). Genetic algorithms and fractals. *Evolutionary Algorithms in Engineering and Computer Science, Ed. K. Miettinen, P. Neittaanmaki, MM Makela, J, Periaux, John Wiley & Sons*.

Mandelbrot, B. B., Evertsz, C. J., and Gutzwiller, M. C. (2004). *Fractals and chaos: the Mandelbrot set and beyond*, volume 3. Springer.

Mandelbrot, B. B. and Mandelbrot, B. B. (1982). *The fractal geometry of nature*, volume 1. WH freeman and Co., New York.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT press.

Sarker, R., Mohammadian, M., and Yao, X. (2002). *Evolutionary Optimization*, volume 48. Springer Science & Business Media.

Slowik, A. and Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 32(16):12363–12379.

Uthayakumar, R. and Prabakar, G. A. (2012). Creation of fractal objects by using iterated function system. In *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)*, pages 1–7. IEEE.

Véhel, J. L. and Lutton, E. (1993). *Optimization of fractal: function using genetic algorithms*. PhD thesis, INRIA.