Lindenmayer System Optimization: An Evolutionary Algorithm Framework

Dr. Saleha Raza
Associate Professor, ECE
Habib University
Karachi, Pakistan
saleha.raza@sse.habib.edu.pk

Lyeba Abid
Computer Engineering
Habib University
Karachi, Pakistan
la07309@st.habib.edu.pk

Ali Muhammad Asad
Computer Science
Habib University
Karachi, Pakistan
aa07190@st.habib.edu.pk

Sadiqah Mushtaq
Computer Engineering
Habib University
Karachi, Pakistan
sm07152@st.habib.edu.pk

Abstract–**This paper presents a novel approach to simulating the evolution of artificial 2D plant morphologies using L-systems, focusing on expanding beyond traditional tree structures to explore alternative forms such as serpinski and dragon curves. Inspired by evolutionary hypotheses on plant development, we employ a genetic algorithm with a fitness function tailored to simulate the factors influencing plant evolution. Our system allows interactive selection, empowering users to guide simulated evolution toward desired phenotypes. Through experimentation, we demonstrate the efficacy of evolutionary algorithms in optimizing L-systems, showcasing their potential in diverse applications such as gaming and architectural design. By extending the scope of L-system exploration, our research contributes to both the understanding of natural phenomena and the generation of novel structures, underscoring the versatility of L-systems in artificial evolution studies.**

I. Introduction

The field of computational intelligence has seen significant advancements in recent years, with applications ranging from data analysis to artificial intelligence, and even various engineering feilds [1] [2]. Computational techniques inspired from nature can play a pivotal role in various fields such as animations, computer graphics, games, and virtual reality. Several techniques effectively model not only real world structures, but also living beings and real life lower organisms [3] [4] [5].

L-systems, or Lindenmayer Systems, based on the work of Aristid Lindenmayer [6] provide mathematical models for the development of growth of various complex organisms such as fungi, plants, and other natural systems. First defined as linear arrays of finite automata, later gained momentum and developed into a branch of formal language theory [7], followed by their applications for modeling plants and plant-like structures using rule-based techniques [8].

Our research aims at using Evolutionary Algorithms to simulate the evolution of artificial 2D plant morphologies. Based on various parameters, those structures can be evolved, each time on a random rule, to provide the optimally stable structure for a given set of parameters. The main goal of this research is to optimize the structure of the plant by evolving the rules of the L-systems using Evolutionary Algorithms.

II. Related Work

Little work can be found in the field of evolutionary algorithms being applied in the context of plants, such as the generation and optimization of L-systems. In the context of games, work has been done to utilize evolutionary computation to make games more immersive, and in the creation of buildings, environmental impacted flora, and dynamic weapons [9] [10] [11]. However, there is a lack of such a method for the generation of realistic flora [12].

The first of the few works that has been conducted on evolutionary algorithms for plants was done by Niklas who used three parameters, along with a constrained evolutionary technique implementing the nearest neighbor heuristic [13]. However, another paper by Jacob restricts the production to a set number of branches, but proves that the use of genetic algorithms for the generation of L-systems is a viable method in the production of 3-dimensional plants [14].

Other works used 2-dimensional DOL-Systems (Deterministic and Context free) that are the simplest L-Systems using an advanced fitness function in order mimic the natural growth of plants [15], and implement an interactive process by selecting the best individuals as the selection process. In 1998, Mock worked on the evolution of Wildwood's plants, which are based on L-systems, by generating a random initial population, incorporating both computational evolution and hand-bred human based phenotypes in his fitness function, and selecting the best individuals as the selection process [16]. This approach helped beginners to test and play around with plant-like structures and their evolutions based on certain features as per their requirements.

III. Lindenmayer System

L-systems, short for Lindenmayer systems, are mathematical models used to describe the growth processes of plants, algae, and other natural structures. They were introduced by biologist Aristid Lindenmayer in 1968 as a way to formalize the study of the development of simple multicellular organisms.

L-systems consist of an alphabet of symbols, a collection of production rules, an initial axiom, and an iteration process. The symbols in the alphabet represent various elements of the system, such as the components of a plant or the instructions for drawing a fractal. The production rules describe how the symbols are replaced or modified in each iteration of the system.

Typically, L-systems are used to generate complex and intricate structures through iterative application of simple rules. They have found applications in computer graphics, where they are used to model the growth of trees, the formation of branching patterns, and the generation of realistic plant-like shapes. They are also used in artificial life simulations and in the study of biological pattern formation.

There are two main things that L systems introduced:

1) **Parallelism**: This means that when parts of a living thing are growing, they can change at the same time. In real life, different parts of a plant might be growing or changing all at once, rather than one after the other. L systems allow us to describe this kind of simultaneous growth [7].
2) **Dynamic Grammar**: Normally, when we think of grammar, we think of rules for putting words together to form sentences. But in the context of L systems, grammar is seen as a set of rules that describe how something changes over time. It is not just static rules but rules that happen in a sequence.

A. *Rule Generation*

The process of generating strings in L-systems involves substitutions based on defined rules, starting from an initial axiom. These elements play a crucial role in determining the structure of the resulting strings and, consequently, the visual interpretation by the turtle.

Variables: In L-systems, variables represent symbols or characters that undergo substitution or transformation according to predefined rules. These variables define the alphabet or set of symbols used in the L-system. We have used the following variables to generate rules: {F, +, -, [,]} Each variable represents how drawing is done. We leveraged the Turtle Library of python to view the drawing of the tree structure. The Turtle navigates the x-y plane, adjusting its position and orientation based on encountered symbols within the L-system string. The turtle's state is defined by a triplet of coordinates (x, y, α), where (x, y) represent its Cartesian position, and α denotes its direction. Here is how each character in the Rule string is interpreted by the Turtle:

- "F" symbol: Upon encountering the "F" symbol, the turtle advances forward with a step size of δ, updating its position accordingly.
  - This movement involves incrementing the angle by θ whenever the turtle encounters the "+" symbol, resulting in a left turn.
  - Conversely, encountering the "-" symbol prompts a right turn, adjusting the turtle's orientation.
  - These simple rules facilitate the generation of intricate shapes and patterns.
- "[ ]" symbol: The approach maintains a stack to manage branching structures denoted by "[" and "]" symbols.
  - Upon encountering "[," the current turtle state is pushed onto the stack, indicating the start of a branch.
  - Conversely, "]" signifies the end of a branch, prompting the retrieval of the topmost state from the stack to reset the turtle's position.

Axioms: The axiom of an L-system serves as the starting point for string generation. It represents the initial configuration from which subsequent strings are derived through iterations. In our case the axiom is $'F'$

Substitutions: Substitutions in L-systems entail replacing symbols in a string according to predefined rules. Substitution for our L-system is dynamic to ensure that an evolutionary algorithm can be used to evolve the plant instead of using an already optimized rule. Our substitution rule is: Let the first randomly generated rule be:

$$F[-F]F[+F][F]$$

Then,

$$\{w : F, p : F \rightarrow F[-F]F[+F][F]\}$$

where,

- w: Represents the axiom of the L-system, which is the initial symbol or starting point of the string generation process.
- p: Represents the production rule or substitution rule of the L-system.

The symbol F is substituted with the sequence $F[-F]F[+F][F]$. This substitution rule specifies how the symbol F evolves in subsequent iterations of the L-system derivation. This keeps on changing based on the generated population of strings.

Validations Checks: We applied some validation checks on the rule to ensure the L-system was sustainable. The Validation checks were as follows:

- Each occurrence of "[" signifies the opening of a branch, and to ensure proper closure, we maintain a count of branches.
- To determine the maximum characters that can be added within each branch, we establish a branch length equal to 30% of the chromosome's length. Thus, every branch contains at least one character, preventing any empty branches.
- If the chromosome contains a "+" symbol, the subsequent character cannot be "-", as it would negate the turtle's previous turn. Similarly, after encountering "-", the next character cannot be "+". Therefore, each "+" or "-" must be followed by either "F" or another branch.

- When multiple branches are present, each one initiates only after the completion of the previous branch, ensuring that no branch is nested within another.

IV. Evolutionary Algorithm

The evolutionary algorithm (EA) stands as one of the oldest and most recognized optimization techniques, drawing inspiration from natural processes. In an EA, the exploration of solution space mirrors the dynamics of natural environments, incorporating principles from Darwinian evolution [2]. Within EA, a population of individuals exists; each individual, referred to as a chromosome, embodies a potential solution to the problem at hand. The problem's definition stems from the fitness function. Based on the degree of alignment with the fitness function, each individual is assigned a quality measure, known as fitness. Fitness serves as a primary criterion for evaluation. Individuals with higher fitness values stand a better chance of being selected for the next generation of the population. EAs encompass three key operators: Survivor Selection (where a new population is formed based on the fitness values of individuals from the previous generation), crossover (which typically involves exchanging parts of individuals between selected pairs), and mutation (where specific gene values are randomly altered). The parameters of the evolutionary algorithm (EA) require careful fine-tuning to achieve a balance between exploration and exploitation within the search space, facilitating convergence. In our application of the EA to optimize plant structures generated by L-systems, we have focused on ensuring both exploration and exploitation by meticulously adjusting parameters and experimenting with various selection schemes.

A. *Population*

In an Evolutionary Algorithm, the population represents a collection of candidate solutions for the optimization problem. For our problem, a chromosome consists of rules of the L-System. Thus, the population comprises an array of L-system rules, where each string corresponds to a chromosome. These chromosomes translate into paths taken by the turtle, represented as an array of Cartesian coordinates (x, y, θ).

The length of each chromosome ranges from 5 to 30 characters initially. However, as the algorithm proceeds and substitutions occur, the chromosome length may increase with each generation. Unlike in traditional Evolutionary Algorithms, we cannot impose strict constraints on the chromosome length during evolution, as doing so could impede the functionality of the L-system.

B. *Fitness Functions*

In order to evolve the structures to make them look like real trees, we looked at past research into botany and what factors or genes really contribute to a well evolved tree structure. We realised that natural selection operates on phenotypic effects rather than genes directly. Human perception excels at selecting phenotypic patterns, but creating computer programs to do so is complex.

To achieve a higher degree of automation, it was necessary to design an effective fitness function. Formulating hypotheses regarding the factors influencing plant evolution was crucial for guiding simulated evolution toward plant-like structures. We came up with 5 behaviors demontrated by plants: Vertical Tropism, Bilateral Symmetry, Stability, structural stability, Photon Harvesting Ability and Branching Proliferation [13] [15].

Consequently, we have based our fitness functions on that. These components are quantified using simple algorithmic techniques based on geometric information derived from the L-system's interpretation in a 2D Cartesian coordinate system.

- **Vertical Tropism** Vertical Tropism assesses the upward growth potential of plants. Taller plants are often more adept at accessing sunlight for photosynthesis and spreading seeds, thus promoting their growth and propagation [13].
- **Bilateral Symmetry** Structural Symmetry assesses the balance and proportionality of plant structures. The 'weight' balance of the structure is estimated by summing the absolute values of vertices' x-coordinates on both sides of the vertical axis. Higher fitness is assigned to structures whose left-to-right ratio is closer to one, indicating better balance and symmetry. This ensures that the plant exhibits a well-balanced branching pattern on both sides of the trunk, enhancing stability and aesthetic appeal.
- **Stability** Stability measures the structural integrity of plants by analyzing the number of branches emerging from branching points. Plants with fewer branches emanating from each point are considered more stable, as excessive branching can lead to structural weakness.
- **Photon Harvesting Ability** Photon Harvesting Ability quantifies the plant's efficiency in capturing sunlight for photosynthesis. This is assessed by determining the maximum absolute value of the x-coordinate along the plant structure. Greater spread or surface area of branches results in higher photon harvesting ability, as it allows the plant to capture more sunlight for energy production.
- **Branching Proliferation** Branching Proliferation examines the density and abundance of branches in plant structures. Plants with a higher proportion of branching points and multiple branches demonstrate enhanced capacity for resource acquisition and reproduction.

Overall Fitness Formula: In order to calculate the combined fitness we used the following formula:

$$\frac{(vt \times w_{vt}) + (ss \times w_{bs}) + (s \times w_s) + (pha \times w_{pha}) + (bp \times w_{bp})}{w_{vt} + w_{bs} + w_s + w_{pha} + w_{bp}} \quad (1)$$

Where:

- w_{vt}: Weight representing the importance of Vertical Tropism
- w_{bs}: Weight representing the importance of Bilateral Symmetry
- w_s: Weight representing the importance of Stability
- w_{pha}: Weight representing the importance of Photon Harvesting Ability
- w_{bp}: Weight representing the importance of Branching Proliferation
- vt: Fitness value for Vertical Tropism
- ss: Fitness value for Structural Symmetry
- s: Fitness value for Stability
- pha: Fitness value for Photon Harvesting Ability
- bp: Fitness value for Branching Proliferation

C. Crossover

In evolutionary algorithms, crossover is pivotal for creating offspring with diverse genetic traits, mimicking natural genetic recombination. Our implementation utilizes a two-point crossover approach, where segments between randomly chosen points in parent chromosomes are exchanged. This process fosters the generation of offspring that inherit a combination of parental characteristics, potentially leading to novel and promising solutions. Valid offspring are iteratively produced until two suitable candidates emerge, contributing to the population's genetic diversity and the algorithm's exploration of solution space. By integrating crossover, our algorithm enhances the search for optimal solutions by leveraging the genetic information encoded in the parent chromosomes.

Consider two parent chromosomes:
Parent 1: 'F+F-[F-]'
Parent 2: 'F-F+[F+]'
Crossover points are randomly selected within each parent chromosome, and genetic material is swapped between them.

Offspring 1 inherits segments from Parent 1 and Parent 2, resulting in 'F+F-[FF-]'. Offspring 2 inherits segments from Parent 1 and Parent 2, resulting in 'F[F+]F+'.

This 2-point crossover introduces genetic diversity, potentially leading to the discovery of novel solutions.

D. Mutation

In our mutation operation, we've implemented two approaches to introduce variability within the offspring chromosomes: Symbol Mutation and Block Mutation.

Symbol Mutation:

Symbol Mutation involves randomly selecting a symbol from the chromosome, typically from the set $\{F, +, -\}$, and substituting it with another random but syntactically correct symbol. For instance, in the chromosome 'F+F-[-FF+]', a symbol mutation might replace the second occurrence of '+' with '-', resulting in 'F-F-[-FF+]'.

Block Mutation:

Block Mutation selects a contiguous block of genetic material within the chromosome and replaces it with a new sequence of characters. This chosen block represents a segment of the chromosome undergoing alteration, thereby potentially leading to the exploration of new genetic configurations. For example, in the same chromosome 'F+F-[-FF+]', a block mutation might replace the block '[-FF+]' with a new sequence, such as 'FFF', resulting in 'F+F-FFF'.

E. Parent Selection Schemes

Parent selection is a crucial step in evolutionary algorithms where individuals from the population are chosen to become parents for the next generation. Here, we explored different parent selection schemes utilized in our system:

- **Truncation Selection**: Truncation selection involves selecting the top individuals from the population based on their fitness scores. In our implementation, we select parents from the top 10% of the population. This method aims to maintain genetic diversity by favoring individuals with higher fitness, potentially leading to faster convergence towards optimal solutions.
- **Fitness Proportionate Selection**: Fitness proportionate selection, also known as roulette wheel selection, selects parents with probabilities proportional to their fitness values. Individuals with higher fitness have a higher chance of being chosen as parents. This method aims to strike a balance between exploitation of high-fitness individuals and exploration of the search space.
- **Rank-Based Selection**: Rank-based selection assigns probabilities to individuals based on their ranks in the population. Higher-ranked individuals have higher probabilities of being selected as parents. This approach is less sensitive to outliers in fitness values and helps maintain diversity by ensuring that lower-ranked individuals still have a chance of being selected.
- **Tournament Selection**: Tournament selection involves randomly selecting a subset of individuals from the population and choosing the fittest individual from that subset as a parent. This process is repeated to select multiple parents. Tournament selection is robust and less prone to premature convergence, making it suitable for maintaining diversity in the population.
- **Random Selection**: Random selection randomly chooses individuals from the population to be parents. In this scheme, each individual has an equal chance of being selected. While simple, this method may not necessarily prioritize fitter individuals, potentially leading to slower convergence or premature convergence to suboptimal solutions.

F. Survivor Selection Schemes

These are the survivor schemes used to determine which individuals from the current population will survive to the next generation:

- **Binary Tournament**: In our implementation, binary tournament selection was utilized, where two individuals are randomly selected from the population, and the fitter individual is chosen as a survivor. This process is repeated until the desired number of survivors is reached. Binary tournament selection provides a simple and efficient way to balance exploration and exploitation.

- **Truncation Selection**: Truncation selection involves keeping the top individuals from the population based on their fitness scores. In our implementation, we retained all individuals, ensuring that the fittest individuals continued to contribute to the population in the next generation. Truncation selection proved effective for maintaining high-quality solutions over multiple generations.
- **Fitness Proportional Selection**: We used fitness proportional selection, also known as roulette wheel selection, to select survivors with probabilities proportional to their fitness values. Individuals with higher fitness had a higher chance of being chosen as survivors. This method maintained diversity in the population while favoring individuals with higher fitness.
- **Rank-Based Selection**: Rank-based selection assigns ranks to individuals based on their fitness values and selects survivors based on their ranks. Higher-ranked individuals had a higher chance of being chosen as survivors. Rank-based selection was robust and less sensitive to outliers in fitness values, ensuring a fair representation of individuals in the next generation.

Algorithm 1: Evolutionary Algorithm for L-System

```
Data: Parameters: population_size, generations,
      mutation_rate, offspring_count,
      substitute_order, seed, weight_a, weight_b,
      weight_c, weight_d, weight_e, PatternType
Result: Final population with their fitness values
1 Initialize:
    – Generate initial population of L-systems.
    – Perform substitution on initial population.
    – Generate turtle interpretation for each L-system in the
      population.
    – Calculate fitness for each L-system based on its turtle
      interpretation using Equation 1.
  for each gen_num in generations do
      – Perform parent selection based on fitness proportion.
      – Perform crossover between selected parents to
        generate offspring.
      – Apply mutation to the offspring.
      – Combine offspring with the current population.
      – Perform substitution on the new population.
      – Calculate fitness for the new population.
      – Perform survivor selection to keep the best individuals.
      – Calculate and record average fitness for the generation.
  return the final population with their fitness values;
  Draw the final L-system using the best individual
   from the final population;
```

a

V. Results and Analysis

A. Observing Convergence per Fitness Function

Initially, we conducted individual tests of our algorithms for each of the fitness functions we had devised. While these tests revealed some aesthetically pleasing structures that aligned with the intended fitness criteria, the resulting forms did not consistently resemble tree-like structures. Nevertheless, this testing phase served to validate the functionality of our fitness functions, confirming their correctness.

1) Vertical Tropism: '

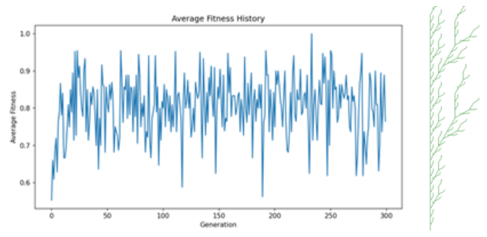

Figure 1: $w_{vt}, w_{bs}, w_{s}, w_{pha}, w_{bp} = 100, 0, 0, 0, 0$

2) Bilateral Symmetry: '

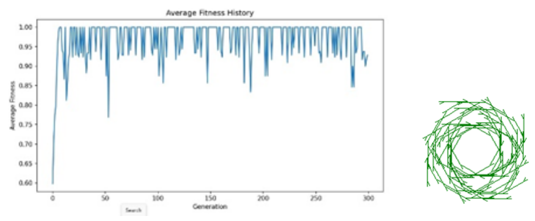

Figure 2: $w_{vt}, w_{bs}, w_{s}, w_{pha}, w_{bp} = 0, 100, 0, 0, 0$

3) Photon Gathering Ability: '

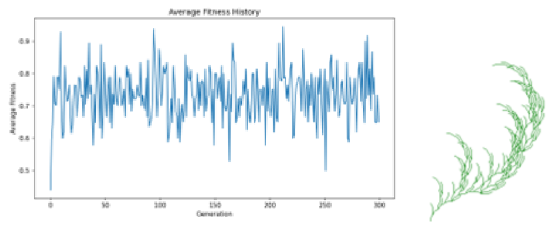

Figure 3: $w_{vt}, w_{bs}, w_{s}, w_{pha}, w_{bp} = 0, 0, 0, 100, 0$

4) Stability: '

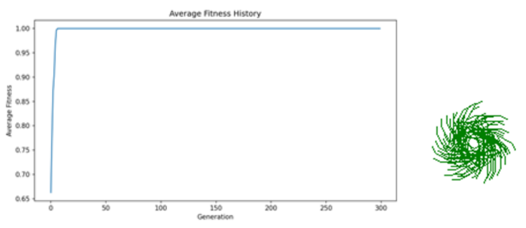

Figure 4: $w_{vt}, w_{bs}, w_s, w_{pha}, w_{bp} = 0, 0, 100, 0, 0$

5) Braching Proliferation: '

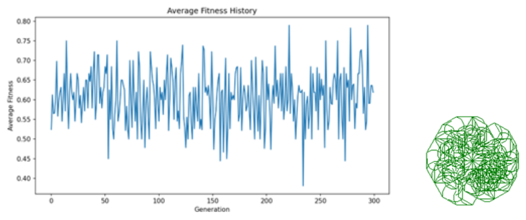

Figure 5: $w_{vt}, w_{bs}, w_s, w_{pha}, w_{bp} = 0, 0, 100, 0, 0$

B. Analyzing Evolutionary Dynamics

1) Comparison between Block Mutation and Symbol Mutation: We implemented our system utilizing both block mutation and symbol mutation techniques. The most favorable outcomes were achieved with the weight parameters depicted in Figure 6 for block mutation and Figure 7 for symbol mutation. We have plotted a graph illustrating the average fitness over successive generations.

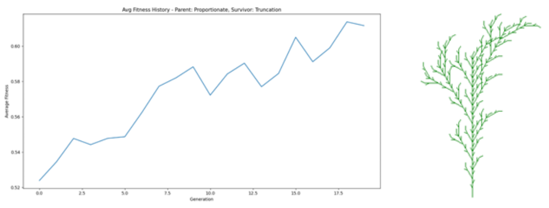

Figure 6: $w_{vt}, w_{bs}, w_s, w_{pha}, w_{bp} = 100, 90, 40, 0, 80$

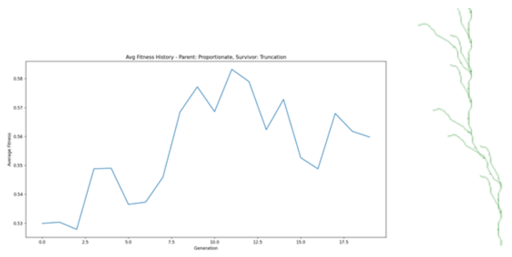

Figure 7: $w_{vt}, w_{bs}, w_s, w_{pha}, w_{bp} = 100, 90, 40, 0, 80$

We can observe from the figures that when we implemented the system with block mutation, it yielded better results compared to symbol mutation. This can be attributed to the fact that block mutation replaces a block of symbols rather than just a single symbol, allowing for larger-scale changes in the chromosome, which makes the population more diverse.

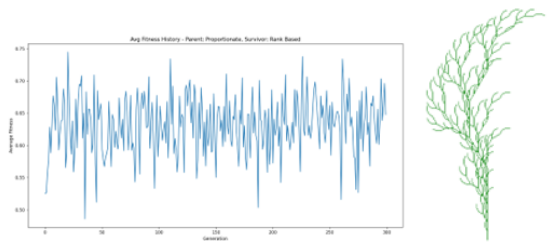

Figure 8: $w_{vt}, w_{bs}, w_s, w_{pha}, w_{bp} = 100, 90, 40, 0, 80$

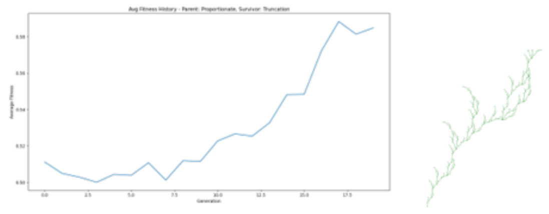

Figure 9: $w_{vt}, w_{bs}, w_s, w_{pha}, w_{bp} = 100, 90, 40, 0, 80$

2) Comparing Survivor Selection Strategies: Rank-Based vs Truncation: Implementing truncation in our system posed challenges, notably in the considerable time required for figure generation due to the significant increase in string length. Unlike truncation, this issue was not observed with the rank-based or any other survivor scheme. Due to limitations in computational resources, we could only implement truncation for 20 generations, compared to 300 generations in the case of rank-based selection. Despite this limitation, the figures clearly demonstrate that truncation outperforms rank-based selection by consistently selecting the fittest individuals, thus ensuring a higher overall performance.

3) Evaluating Parent Selection Methods: Fitness Proportion vs Tournament: '

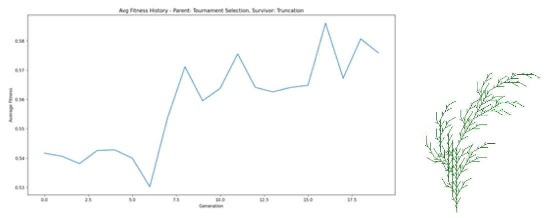

Figure 10: $w_{vt}, w_{bs}, w_s, w_{pha}, w_{bp} = 100, 90, 40, 0, 80$

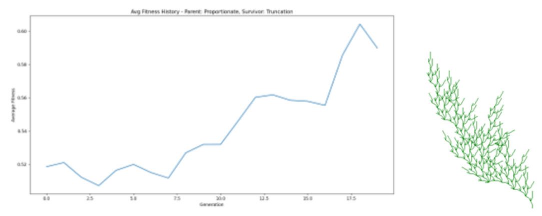

Figure 11: $w_{vt}, w_{bs}, w_s, w_{pha}, w_{bp} = 100, 90, 40, 0, 80$

In our system, we employed various parent selection schemes. However, when comparing tournament selection with fitness proportionate selection, the latter outperformed. This is because fitness proportionate selection consistently emphasizes choosing individuals based on their fitness, leading to a more directed evolutionary process. Conversely, tournament selection introduces randomness, potentially leading to suboptimal parent choices and slower convergence towards optimal solutions.

VI. Testing on Alternative Rules

We experimented with various rule sets of L-systems, adjusting parameters such as the angle increment from 30° to 120° for generating the Sierpinski triangle and 90° for the Dragon curve. While the traditional rules typically vary primarily in the angle and substitution rule, we did not vary the substitution rule. Instead, we used a dynamic substitution approach that remained consistent across all patterns. This method ensured that the substitution process adapted dynamically with the population, contributing to the evolution of both Sierpinski and Dragon curves. The results of this approach revealed captivating evolutionary patterns as generations progressed.

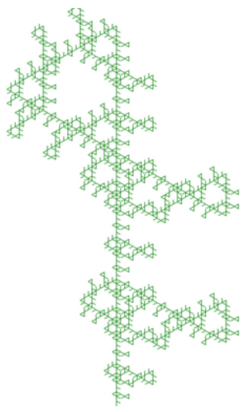

Figure 12: Sierpinski using Photon Gathering Ability only

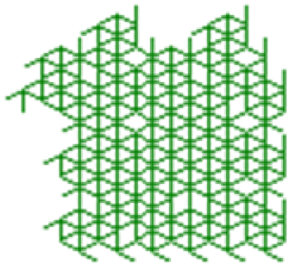

Figure 13: Sierpinski using Symmetry only

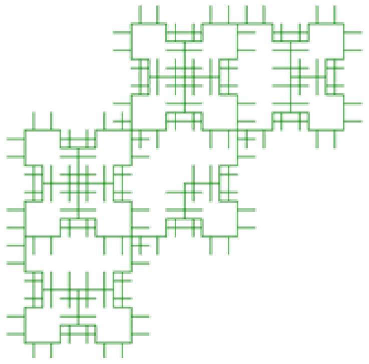

Figure 14: Dragon using Branching Proliferation only

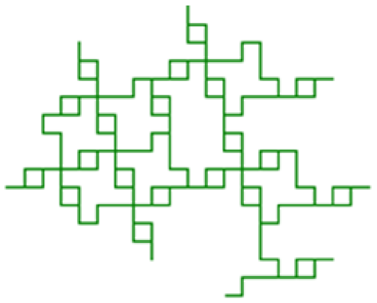

Figure 15: Dragon using Stability only

VII. Future Work

In our implementation, a chromosome represents a rule, which is dynamically updated through substitution in the genetic algorithm a certain number of times. As mentioned earlier, this leads to a significant increase in the length of the updated string within a short period. Consequently, parent selection schemes such as the truncation scheme show significant performance degradation after only about 20 generations due to the increased string length and memory issues. A similar degradation is observed in the fitness proportionate selection scheme, which also necessitates calculating the fitness of the entire population. As a result, we were unable to run the truncation scheme for more than 20 generations, in contrast to 300 generations for other selection schemes. Access to additional computational resources beyond our laptops, such as a high-performance computer or computing cluster, would enable us to run the truncation scheme for a longer period and potentially observe better results.

Furthermore, most of our testing for the new rules was conducted for the tree-based structure, and the fitness functions were designed to optimize this structure. The same approach is being applied to the Sierpinski triangle and the Dragon Curve. Moving forward, future work could involve specifically testing the new rules for these structures and designing new and improved fitness functions tailored to their exhibited patterns. This could undoubtedly yield even better results for these structures.

Finally, color can also be considered when designing the fitness function. Using color as a parameter and evolving it could lead to the creation of structures that are more aesthetically pleasing, visually appealing, beautiful, and even realistic. This could represent a significant step towards the generation of realistic flora.

References

[1] A. Ethan and E. Edward, "Artificial intelligence in bioinformatics: Advancements and applications," 08 2023.

[2] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Computing and Applications*, vol. 32, pp. 12363–12379, 2020.

[3] R. Dawkins, *The Blind Watchmaker*, vol. 58. 01 1986.

[4] P. Oppenheimer, "Real time design and animation of fractal plants and trees," vol. 20, pp. 55–64, 08 1986.

[5] K. Sims, "Evolving virtual creatures," vol. 8, pp. 15–22, 01 1994.

[6] A. Lindenmayer, "Mathematical models for cellular interactions in development i. filaments with one-sided inputs," *Journal of Theoretical Biology*, vol. 18, no. 3, pp. 280–299, 1968.

[7] G. Rozenberg and A. Salomaa, *The mathematical theory of L systems*. Academic press, 1980.

[8] P. Prusinkiewiczy, P. Hananz, M. Hammely, and R. Mech, "L-systems: From the theory to visual models of plants," 01 2002.

[9] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," *ACM Trans. Graph.*, vol. 25, pp. 614–623, 07 2006.

[10] N. Shaker, J. Togelius, and M. Nelson, *Procedural Content Generation in Games*. 01 2016.

[11] R. Měch and P. Prusinkiewicz, "Visual models of plants interacting with their environment," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, (New York, NY, USA), p. 397–410, Association for Computing Machinery, 1996.

[12] B. Fitch, P. Parslow, and K. Lundqvist, *Evolving Complete L-Systems: Using Genetic Algorithms for the Generation of Realistic Plants*, pp. 16–23. 06 2018.

[13] K. J. Niklas, "Computer-simulated plant evolution," *Scientific American*, vol. 254, no. 3, pp. 78–87, 1986.

[14] H. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel, and C. Jacob, "Evolution programs evolved," 10 1996.

[15] G. Ochoa, "On genetic algorithms and lindenmayer systems," in *International Conference on Parallel Problem Solving from Nature*, pp. 335–344, Springer, 1998.

[16] K. Mock, "Wildwood: the evolution of l-system plants for virtual environments," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 476–480, 1998.