Universiteit Leiden

Computer Science

Dynamically evolving L-system generated plant visualizations

Name: Sander Ruijter S1415212
Date: 25/08/2016

1st supervisor: Dr. M.T.M. Emmerich
2nd supervisor: Dr. A.H. Deutz

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this thesis there will be a discussion about interactive evolutionary algorithms and if they are suited for the synthesis of Lindenmayer-systems. This is done by explaining both terms and how they can be combined into a Python program. That program is then used for a small scale research into the feasibility. In the end the Hausdorff and Levenshtein distances are given as ideas to speed up the process of the human selection.

Content

1. Introduction

The beauty of plants is something that has caught the attention of many researchers. Especially how to capture this beauty in a simple system has attracted the attention of computer scientists for a long time. Mathematicians also research the complex forms of symmetry in plants, because “Beauty is bound up with symmetry.” [9]

To try and capture this beauty digitally it is possible to use Lindenmayer-systems, or L-systems for short. L-systems were introduced by Aristid Lindenmayer in 1968. [7] L-systems are a set of rules which, in different iterations, can transform an axiom into a more complex string.

Each L-system consists of three parts: an alphabet, an axiom and a set of rules. The alphabet contains all the characters used to describe the L-system, typically F for branches, and +/- for changing the angle. The axiom is a starting string of characters from the alphabet. The rules are used to rewrite a string, starting with the axiom to get a more complex string. This can be done in different iterations to, normally, increase the size and complexity of the string.

After a string is generated a Turtle can be used to draw a graphical interpretation of the L-system. For this graphical interpretation the F will be used to move the turtle forward and the +/- to rotate the Turtle.

Although L-systems are a straightforward approach to create plants, getting such an L-system from (a picture of) a plant is not an easy task. This is because to fully automate this, you need to implement how to recognize the plant in the picture, compare it to the L-system generated picture and evolve it in such a way the pictures will be more similar.

To tackle two of these problems an interactive evolution method can be used. With this approach a person can select from a number of offspring’s evolved by a program and in this way select in such a way that an L-system will be created which looks like the plant the user of the program wants.

In this thesis there will be an explanation on interactive evolution of grammatical picture generation and how they can be mutated. There will also be an experiment with users concerning time, strategy, user experience and performance. In the end there will be a short discussion on recombination and selection.

1.1. Research Questions

This thesis will be about how you can synthesize an L-system of a given plant on a computer with the use of an interactive evolutionary algorithm. The first question that comes to mind is how can L-systems be implemented in a modern computer language, in this case Python?

The second question will be if it is possible to synthesize L-systems who resemble plants using an interactive evolutionary algorithm and how should you define the operators for mutation and, if implemented recombination?

The last question is how long would it take to accomplish evolutionary synthesis of an L-system by a human user, and is it within an acceptable timespan?

2. L-systems

L-systems were conceived as a mathematical theory of plant development, they are a type of formal grammar. They are used to describe how you can form strings which are part of the L-system. L-systems first appeared in *Mathematical models for cellular interaction in development. Journal of Theoretical Biology, 18:280–315, 1968* by A. Lindenmayer, hence the name Lindenmayer-systems. The first L-systems were not intended to be used for modelling complex plants, the emphasis was instead on the topology of the plants.

2.1. Types of L-systems

When working with L-systems there are different types you could use. For this thesis the types which will be discussed are Edge and Node rewriting L-systems, Bracketed L-systems, Stochastic L-systems and Context-sensitive L-systems. For the experiments the focus will be on bracketed L-systems with node rewriting.

With Edge rewriting systems the branches are replaced in multiple iterations. Most of the time there are multiple branch characters, for example F_l for a left orientated branch and F_r for a right orientated branch. In this way more complex systems are possible than with just one type of branch. In figure 2.1 an example of the turtle interpretation of an Edge rewriting L-system is given with 4 iterations, an angle of 60° and the axiom F_l.

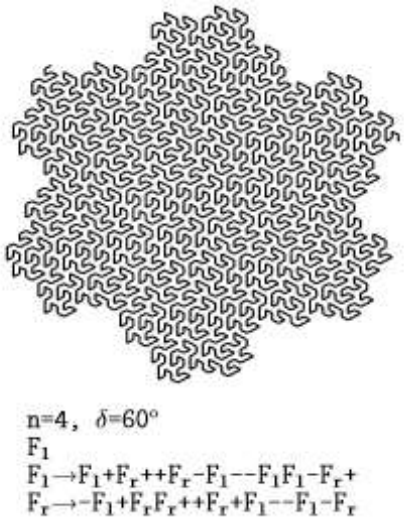

Figure2.1: An example of an Edge rewriting L-system. [8]

Node rewriting L-systems use a new character to expand the string, the character X will be used for this in this thesis. With these kind of L-systems typically the X is placed at the end of a branch, this way the generated picture will grow from one point. This type of rewriting is suited for generating simple plants because plants also grow from the end of their branches, just like these L-systems. Therefore in this thesis Node rewriting L-systems will be used to generate the pictures. In figure 2.2 an example of the turtle interpretation of an Node rewriting L-system is given with 2 iterations, an angle of 90° and the axiom L with L and R being used instead of X.

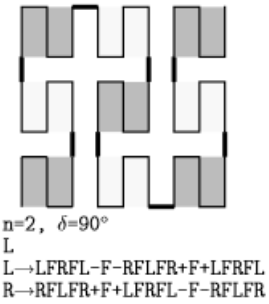

figure 2.2: an example of Node rewriting. [8]

To actually get the multiple ends of the branches to get a plant like structure, a way is needed to go back in the drawing phase to a previous point. To accomplish this square brackets are used, "[" and "]".

With each closing bracket the position is reverted to the corresponding opening bracket. This way a picture can be generated with multiple ends, so you can get a more plant like structure. Because of this the brackets are implement in the L-system used in this thesis. In figure 2.3 an example of the turtle interpretation of a Bracketed Node rewriting L-system is given with 5 iterations, an angle of 22.5° and the axiom X.

Figure 2.3: an example of Bracketed Node rewriting. [8]

The L-systems used in this thesis are all non-stochastic, which means they will always produce the same string and picture. With Stochastic L-systems it is possible to make some random selection between similar rewrite rules, this way with one L-system different variations of strings and pictures can be generated with the same overall look. However this does not add much value, the idea is to try and remake picture and if different L-systems can generate the same picture. Randomized L-systems will make this process more difficult because the same L-system can now make a different string/picture and even if two L-systems are now able to generate the same string/picture it can also generate a different string/picture and thus it will be harder to detect the similarities. In figure 2.4 an example of the turtle interpretation of a Stochastic Bracketed Edge rewriting L-system is given with 5 iterations, an angle of 45° and the axiom F.

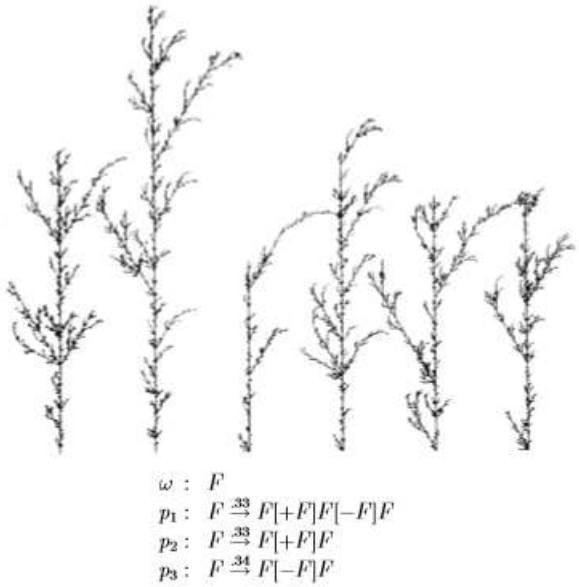

Figure 2.4: an example of Stochastic Bracketed Edge rewriting.[8]

The last variant of L-systems discussed is about the context sensitivity of the systems. Context sensitive L-systems can have different replacement rules depending on the characters before and after the replaced character. This will make the picture seem more diverse, but this will also make working with them more complicated. Also it adds a large set of rules which can be evolved but does not add extra value to the research and thus was left out for this thesis. If you want to know exactly how these kind of L-systems work, it is advised to read chapter 1.8 of The Algorithmic Beauty of Plants from P. Prusinkiewicz and A. Lindenmayer.

2.2. The Program

For this research a program was written in Python. The functions on how to draw an L-system was made with a tutorial from interactivepython.org, this tutorial included the basic uses of turtles and generating of standard Edge rewriting L-systems. This was modified to work with Node rewriting L-systems and to include Bracketed L-systems. To achieve the shift from Edge to Node rewriting was fairly easy. In the rules section a new character was used for the nodes, the character X.

To implement the brackets three stacks were used, one for the X-coordinates, one for the Y-coordinates and one for the direction. These stacks are local to the drawing function because they are only needed there. When there is an opening bracket “[” the position is pushed to the stacks and when there is a closing bracket “]” the position is popped from the stacks. Because opening and closing brackets always come in pairs there will be no problem when using this approach and at the end of the drawing phase the stack will be once again empty.

To save all other data of the L-systems a list variable is used. This list contains the replacement rules of the L-systems, the number of iterations needed to generate the picture, the starting axiom, the angle the turtle makes on turns and the length of the lines the turtle draws.

In figure 2.5 the control screen of the program is shown. This includes the parent-button for the L-system which was te last to mutate, the 6 children which have been mutated from the parent, a select button to mutate the last selected L-system and the done button, to show the given L-system and the numeber of times a mutation was selected to continue the process.

Figure 2.5: The control screen.

In figure 2.6 there is a picture of the drawing screen. In this screen the Turtle interpretation of the L-systems are shown. Every time you push the button of one of the L-systems on the control screen a that L-system will be drawn on the drawing screen.

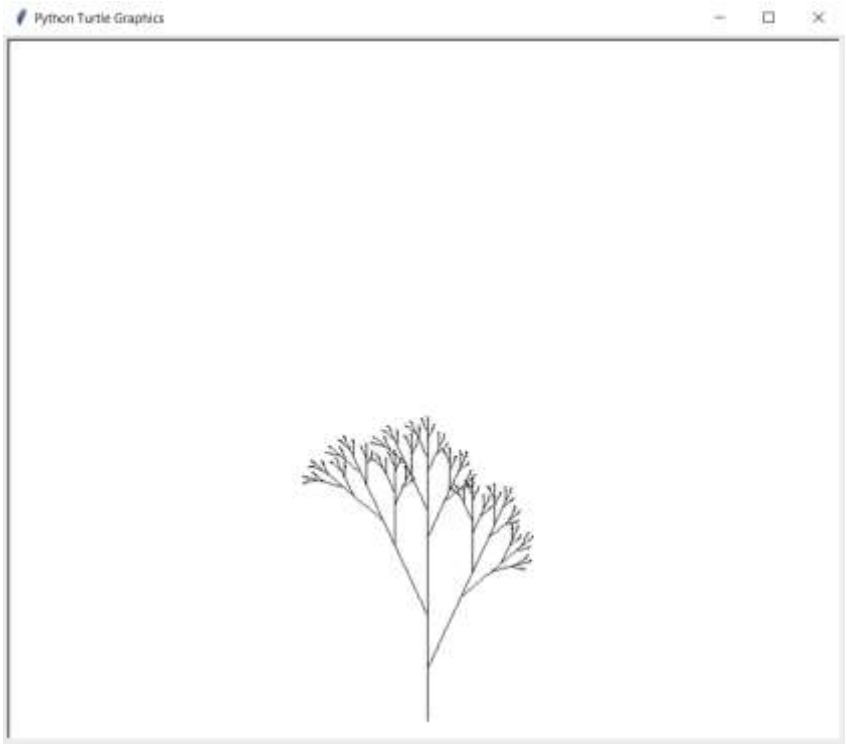

Figure 2.6: The drawing screen

The last screen the program opens is the screen in figure 2.7. This is the text screen. In the text screen, at the start of the program, is a small introduction to the program and what is expected during the experiments. When you push the select or the done button this is the screen which shows the information of what happens in the program. For someone who just uses the program for the experiments this is not that important. If you want to learn more about how the L-systems mutate and what rules they use, this screen is the most intresting one.

Figure 2.7: The text screen.

3. Evolutionary Algorithms

As the name implies, Evolutionary Algorithms (EA) are inspired by biological evolution. They are able to change a process, function or variables into slightly different ones. For this EA use tactics inspired by evolutionary processes like mutation, recombination and selection. By using these tactics they can slowly evolve the data given to them.

EA are best suited for finding reasonably good solutions. They evolve the data with small steps which makes it easier to guide the process. With this guidance it is more likely for a solution to be found. Guiding the EA can be done by the program itself or by a user, then it is called an interactive EA.

3.1. Interactive EA

For this thesis we will purely focus on interactive EA, that is when at certain points during the algorithm the user is able to give feedback to the algorithm. This can be done with reactive feedback or with proactive feedback.[2] The program uses reactive feedback because this is more natural for inexperienced users than proactive feedback.

Reactive feedback is when the algorithm asks for feedback from the user. This can be in the form of generating variants, selecting or evaluating the offspring for the next step in the algorithm.

Proactive feedback is when the user itself can interrupt the algorithm. Then they can adjust the parameters of the algorithm to get it out of stagnation or add new individuals into the population to increase the diversity and guide the algorithm to a solution based on the preference of the user.

The tactics used in Evolutionary algorithms are taken from the biology and adapted to be used in an algorithm. This makes the ideas behind the different parts and variants of EA easier to understand if you know how those processes work in the biology.

The first part of EA's is about how to generate offspring. In this thesis we will take a look at two ways to do this, mutation and recombination. Mutation is when one part of the offspring is changed. In an L-system this can be that one of the rules is changed, the axiom is changed or even the number of iterations or the angle. It is possible to do multiple mutations in one generation and in this way speed up the process at the cost of it being harder to get a specific result. For the program one mutation per generation is used.

Another way to generate offspring is by using recombination. With recombination there will be multiple parents selected which will then be combined in different ways to generate the new offspring. However, combining the parents is often not an easy task. For L-systems specifically because you cannot just take one role from one system and the other from another system to get an L-system which resembles them both. Further explanations on how you will get better results will be in paragraph 5.1.

3.2. The Program

In the program there are three functions which together make up the part about mutation. Those functions are *evolve*, *mutate* and *mutateRules*. From those three functions the evolve and mutate will always be used and mutateRules only when a rule or an axiom must be mutated.

The first function called when the Select button is pressed is evolve. This function gets the last L-system drawn as parameter, the only thing it does is set the last system as the parent of the new generation and call the function mutate six times to form the children of the new generation.

The function mutate, as the name implies, mutates the L-system that is given as a parameter to it. For this it generates a random number, which is used to select what to mutate. The choice is between the rules, the axiom, the number of iterations, the angle the turtle makes on a turn and the distance the turtle travels on a forward command. The rules have the highest chance to be mutated, followed by the axiom and lastly the iterations, angle and distance.

When the mutate function needs to mutate the rules or an axiom it calls the mutateRules function. This function once again takes a random number to make a decision on what to do. It can add or remove (if possible) an “F”, “X”, “+”, “-” or brackets. The added symbols are put randomly somewhere in the rule or axiom. If a character should be removed, the first one in the rule is removed.

Brackets are a harder to add and remove because they are always in pairs. When adding brackets the opening bracket must always be placed before the closing bracket as to not cause errors with trying to pop an empty stack. With removing the brackets there is a similar problem. Every closing bracket must have an opening bracket which precedes it. To cope with this the first opening bracket is removed and the first closing bracket. In this way there will be no errors with the brackets which cause stack problems.

4. Results

In this thesis there were three research questions:

How can L-systems be implemented on a computer, in this case with Python? This question has been answered in the second chapter by explaining how the first part of the program works.

Is it possible to synthesize L-systems who resemble plants using an interactive evolutionary algorithm and how should you define the operators for mutation and, if implemented recombination? This question can be cut into three pieces. The first part about the possibility of synthesizing L-systems who resemble plants will be explained and tested in this chapter. The tests will include multiple users trying to generate an L-system and making it as similar as possible to a picture of a real fern. The second part about the mutation operators has been answered in the third chapter by explaining the working of the mutations and the evolutionary algorithm. The last part of the question is about how to implement recombination, this is something not implemented during the thesis but will be further explained in chapter 5.1.

And how long would it take to accomplish evolutionary synthesis of an L-system by a human user, and is it within an acceptable timespan? The final question from this thesis will also be answered in this chapter. For this the same tests of the possibility of the synthesis of the L-systems are used. The time will be measured and the users are asked to fill in a form about how their experience with the program was.

4.1. Inverse problem

If there is a function with an input that is processed by the function and results into an output, the inverse problem would be how to find the input of a given output. In this thesis the inverse problem is how to create an L-system from a picture of a plant. The output is the picture and the input to create is a set of rules and parameter in the form of an L-system of the plant in the picture.

4.2. Experimental Setup

For the experiments a two variations of the program were used. The first one was to get the user familiar with how everything works. They start with the choice of 7 more or less complete L-systems and then are able to evolve them further into a form they prefer. The users are asked to continue until they think they know how the program works.

For the second program they start with one L-system which is just a few mutations away from the target picture, as seen in figure 4.1. Their task is to evolve it into the target L-system. This was chosen over letting them start from scratch because it may take an enormous amount of time to just generate a few profitable mutations, especially if you don’t know how L-systems work. To start from close to nothing just takes way too much time, as noticed after a few test experiments.

When the target L-system, or something close enough to it as interpreted by the user, is completed, they are asked to fill in a questionnaire. Here they were asked about their experience with the program, if they had any tips/complains and how long it took them to get a result in the second program and which strategy they used.

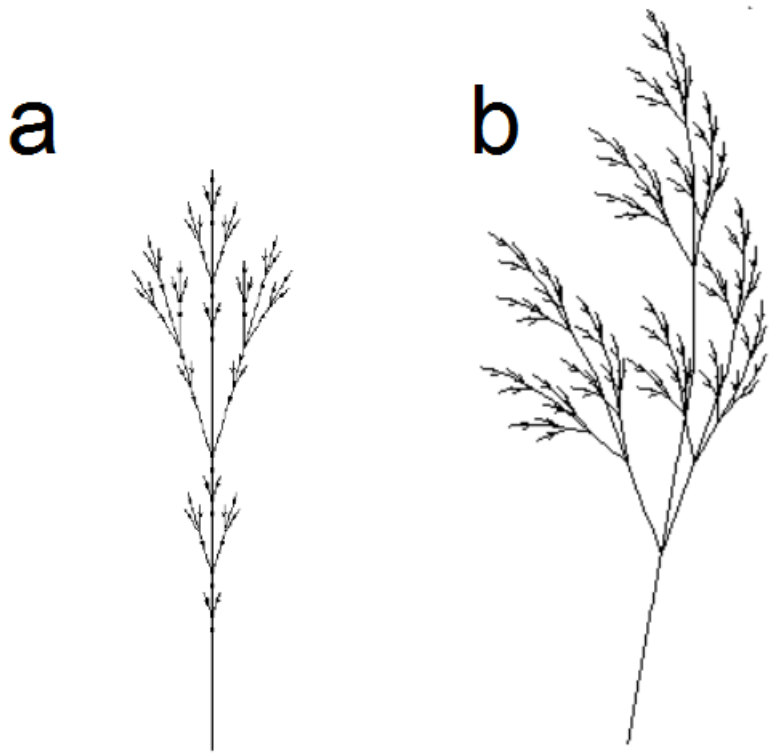

Figure 4.1: Starting and end point of the experiment.

The users of the program were all in the age of 18 till 23 years old. They had multiple levels of education ranging from MBO to WO. None of them exactly knew how L-systems worked and so during the first part of the experiment they were thought not only the basics of the program but also of L-systems in general.

4.3. Results

The experiments should answer the research questions if it is possible to achieve synthesis of an L-system which represents a given picture, how long it will take to do this, and if this is within an acceptable timespan.

To answer the first question, it is possible to achieve synthesis of an L-system of a given picture, but this turns out to be very hard with just visual feedback. In the final round of experiments the users were asked to change an L-system to a different L-system. This was possible in 6 mutations but the fastest done by a user was in 14 mutations. Also this was not with just the graphical feedback, but also by using the textual feedback of the mutations in combination with studying the target picture to get an idea about how the L-system should look like. Even for such a small amount of mutations to take place it took about half an hour to go from figure 4.1a to 4.1b.

So to actually start from scratch and build an L-system from an actual plant which is way harder to analyse will take much longer. Especially if the user of the program does not completely understand how L-systems work and when to choose which mutation. To completely synthesize the L-system of the experiment out of the basic L-system with axiom F and rules X→F and F→FF takes at least 24 mutations. So this will take approximately 2 hours but only if the same speed of mutations can be achieved as with the experiment, which is just the final part of the whole syntheses.

Because of the complexity of normal plants they are much harder to completely understand the underlying structure. This will also add to the time needed to synthesise the L-system. All together it is thus estimated to take well over 2 hours, which for a human is a long time to sit with the program and slowly mutate the L-system. Thus to actually make it a success the system has to be drastically sped up because as of now the time it takes is not acceptable.

The results of the experiments are shown in table 4.1. It includes the minimum, maximum and average value of how long the second part took and the amount of mutations it took to get perfect synthesis. The last row shows that 75% of the time perfect synthesis of the L-system is achieved. Thus 25% of the time the user of the program was happy with a not perfect representation, or thought the obtained L-system was as close as they could get.

	Minimum	Maximum	Average
Time taken	19 minutes	35 minutes	26 minutes
Mutations to target	14	23	18
Perfect synthesis	-	-	75%

Table 4.1: Results of the experiments.

4.4. Discussion

To actually get a test experiment which did not took too long that the users got bored was not an easy task. To test the feasibility of the tests they were first performed on a small test group of three people. Only at the third try was there a test that could be finished within 30 minutes. This was necessary because after 30 minutes without getting noticeably closer to the target the users started to get frustrated with the program. This was because although they were coming closer, the picture generated did not start to become similar to the target and thus there was no form of feedback they were doing it right.

The first set of small scale experiments asked the users of the program to remake the picture of a plant without any kind of head start. After approximately 45 minutes, and still no one getting close to the target, this experiment was cancelled due lack of results.

In the second set of small scale experiments the users were asked to remake an L-system from a picture. As a head start the rules needed already had five of the needed thirteen characters and the amount of iterations was already close too. This increased the visual feedback the users received but still after 45 minutes only one of the three managed to complete the experiment.

Halfway through the second experiment, the users actually started to make some use of the text screen to see what changed with every mutation. This allowed them to make their decisions faster and also with results closer to the target. Because of this an extra oral explanation was added on how the program and L-systems work. This helped to increase their understanding and efficiency in working with the program.

In the last set of small scale experiments the amount of mutations users had to make were further decreased by giving an extra head start in the rules and the angle needed in combination with the head starts from the second set of experiments. Only then was

it possible for most users to finish the experiment within 30 minutes and be fairly satisfied with the result.

Also the program used in the experiment is far from flawless. It is possible that it tries to delete characters which are needed or not even in the part it mutates. Also it is possible to do one mutation multiple times in one generation and thus doing less useful mutations. All these flaws were things that annoyed users. Thus for further improving the program guards can be installed which check against those conditions and select a new mutation.

Further criticism was about how most of the mutations did not increase the likeness of the L-system with the picture. This is because all mutations are completely random and the odds of a specific mutation appearing do not change throughout the program. The problem is that sometimes one specific mutation is needed and to actually get that one specific mutation wanted the parent generation should be mutated multiple times. This does not add to the number of mutations done, but takes a lot of time anyways.

Another point of critics is about the naming of the “done” button. This button shows the L-system in the text screen including all other variables of the program. It was originally intended to only be pressed when the users were finished. However it is now also used to see the L-system and other variables throughout the modification process. Because this helps the user gain insight into how the L-system works and what would be good steps how to mutate them it was advised to change the name to “check” or something similar.

5. Extra ideas

The program for this thesis, while finished, is not completed. The program itself has only the most basic of the evolution mechanisms, mutation. There are other evolution mechanisms that can be added to improve it. For further improvement selection methods can be implemented to assist the user in getting better results faster and change the feedback procedure from reactive to proactive.

5.1. Recombination

To implement recombination with L-systems is not something done easily. L-systems are made of a set of rules and, in this program, some extra variables to draw them. If you want to recombine two L-systems taking some rules from one and some from another will not give you an L-system which represents both. If you want to achieve that result the rules themselves have to be recombined.

To recombine the rules of Node rewriting L-systems there are lot of things which should be taken into account: the brackets, the rotations, the edges and the nodes. In the system of the program it also adds angle, iterations and length of the edges, but those are easier to recombine by taking a random value between the two values of the systems.

To get a L-system with similarities with both parents, the amount of brackets may not differ too much. If there are too many there will be more branches than with the parents and with too few there will be less. So to recombine them some bracketed parts of the L-systems need to be recombined, but which?

The a good option to recombine them is to also take a look at the rotation characters (“+” and “-”). This is because in this way you can differentiate between left and right branches and thus combine those specific branches.

Furthermore it is not advised to just add parts together, this will make the L-system much bigger than the parents. So to make an L-system about the same size as the parents you want to have about the same amount of edges (“F”) and nodes (“X”).

In figure 5.1 there is an example of how it can look if parents A and B are combined. All the systems have 3 iterations and an angle of 25 degrees. With the first and second child the rotation was not taken into account and thus one side of the pictures look empty. With the third and fourth child the characters before and after the brackets are different and thus make the pictures top or bottom heavy. With the fifth child all sets of brackets are combined into one child, this makes the picture look much fuller than both parents. On such a small picture this may not look like a problem but on an L-system with more iterations it will soon become crowded.

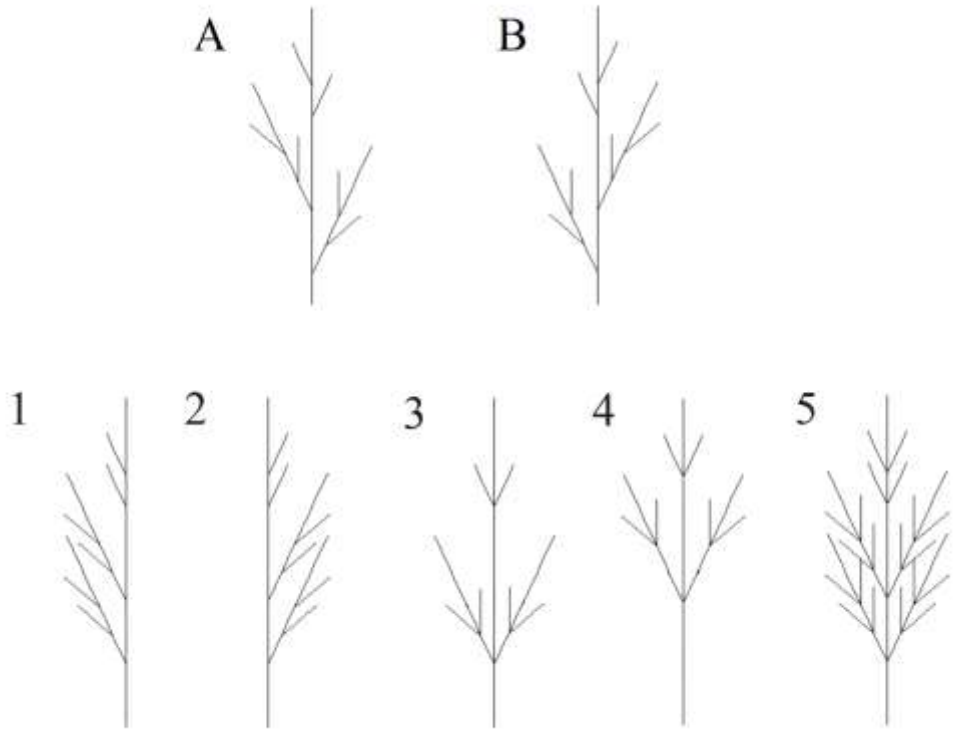

Figure 5.1: Seven turtle interpretations of L-systems with the following node rewriting rules:
A: X→F[+FX]F[-FX]FX
B: X→F[-FX]F[+FX]FX
1: X→F[-FX]F[-FX]FX
2: X→F[+FX]F[+FX]FX
3: X→F[-FX][+FX]FFX
4: X→FF[-FX][+FX]FX
5: X→F[-FX][+FX]F[-FX][+FX]FX

5.2. Hausdorff Distance

Another point in which the program can improve is the amount of work the users have to do. Now the user is in complete control of the program but it takes a lot of time to get some progress. To decrease the workload of the users a selection algorithm can be used. In this thesis two ways to calculate how much alike the L-systems are will be discussed, the Hausdorff distance and the Levenshtein distance.

The Hausdorff distance first appeared in the *Grundzüge der Mengenlehre* by Felix Hausdorff which was first published in 1914. The Hausdorff distance is a way to compare two pictures, or more precisely two sets of points in IR^2 (or in general in IR^d), and see how different they are. The Hausdorff distance is the largest distance from any point in one picture to the nearest point in another picture.[6]

By letting the computer calculate the Hausdorff distance and selecting the mutation with the smallest Hausdorff distance, an L-system which is more similar to the target will be created. If the Hausdorff distance becomes zero then the two pictures are the same. However getting to such a distance is not easy for a computer, it can get stuck with an L-system not too similar to the target. This is where the user of the program interrupts the algorithm and either makes some changes or select an L-system with a bigger distance which looks more promising.

The biggest problem however with implementing this into the program is with how to recognize the images in a way to compare them. The L-systems need to be converted to images and then again converted in a set of points so it can be given to the algorithm which can compare them to the target. For this rotation and translation may be needed to transform the picture without changing its essential shape. this way the Hausdorff distance will be minimalized

5.3. Levenshtein Distance

The Levenshtein distance or edit distance is, in contrary to the Hausdorff distance, not a distance between pictures but between strings. It is named after Vladimir Levenshtein who first came up with the idea for this distance in 1965. The Levenshtein distance is the amount of steps it takes to go from one string to another by using substitution, insertion or deletion.

L-systems are used to generate strings and thus the Levenshtein distance looks like a great match to compare a given L-system to a target. There is only one problem with using the Levenshtein distance, it also needs a string to compare to. This means that to actually use it there already has to be a string of either the output of the L-system or the rules of that L-system. If the program will be used to generate plants from pictures those strings are not available and thus the Levenshtein distance cannot be used as a selection method.

However in the case you want to recreate a given L-system it can be a useful selection method. An interesting question is if an output of an L-system is unique for that L-system or if there are others with the same output. By using the Levenshtein distance those outputs can be compared and selections can be made. Once an output with the Levenshtein distance of zero is generated the underlying L-systems can be compared and give some insights into this question. but how this was not the main subject of this thesis and due lack of time to implement this it was left out for now.

6. Conclusion

In summary, this thesis gave an answer to the research questions. It is possible to implement L-systems and L-system evolution in python. This was done for bracketed L systems with node replacement and evolving them through mutations. For recombination suggestions have been made but they were not implemented.

The thesis shows that it is difficult to recreate an observed L-system from scratch by means of interactive evolution, but if the solution is already close it might be possible, in particular if the user can directly suggest mutations.

Ideas of automated evolution loops have been suggested, and the Hausdorff distance seems to be a promising way to do this. The Levenshtein distance lends itself for studying some general aspects, such as uniqueness of solutions.

7. References

1. T. Bäck and H. Schwefel. An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation, 1* (1). 1-23.
2. R. Breukelaar, M. Emmerich, and T. Bäck. On Interactive Evolution Strategies.
3. R. Campbell. *Describing the shapes of fern leaves: A fractal geometrical approach.* Kluwer Academic Publishers, 1994.
4. S. Droste and D. Wiesmann. *Metric Based Evolutionary Algorithms,* Springer-Verlag, Berlin, 2000.
5. M. Emmerich, M. Grötzner and M.Schütz. Design of Graph-Based Evolutionary Algorithms: A Case Study for Chemical Process Networks, *Evolutionary Computation, 9* (3). 329-354.
6. D. Huttenlocher, G. Klanderman and W. Rucklidge. *IEEE Transactions on pattern analysis and machine intelligence, 15*(9). 850-863.
7. A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology, 18:280–315, 1968.
8. P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, Berlin, 2004.
9. H. Weyl. *Symmetry*. Princeton University Press, Princeton, New Jersey, 1982.