Inferring Different Types of Lindenmayer Systems Using Artificial Intelligence

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Jason Bernard

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Lindenmayer systems (L-systems) are a formal grammar system which consist of a set of rewriting rules. Each rewriting rule is comprised of a symbol to replace (predecessor), a replacement string (successor), and an optional condition that is necessary for replacement. Starting with an initial string, every symbol in the string is replaced in parallel in accordance with the conditions on the rewriting rules, to produce a new string. The replacement process iterates as needed to produce a sequence of strings. There are different types of L-systems, which allow for different types of conditions, and methods of selecting the rules to apply. Some symbols of the alphabet can be interpreted as instructions for simulation software towards process modelling, where each string describes another step of the simulated process.

Typically, creating an L-system for a specific process is done by experts by making meticulous measurements and using *a priori* knowledge about the process. It would be desirable to have a method to automatically learn the L-systems (the simulation program) from data, such as from a temporal sequence of images. This thesis presents a suite of tools, collectively called the Plant Model Inference Tools or PMIT (despite the name, the tools are domain agnostic), for inferring different types of L-systems using only a sequence of strings describing the process over some initial time period. Variants of PMIT are created for deterministic context-free L-systems, stochastic L-systems, and parametric L-systems. They are each evaluated using existing known deterministic and parametric L-systems from the literature, and procedurally generated stochastic L-systems. Accuracy can be detected in various ways, such as checking whether the inferred L-system is equal to the original one. PMIT is able to correctly infer deterministic L-systems with up to 31 symbols in the alphabet compared to the previous state-of-the-art algorithm's limit of 2 symbols. Stochastic L-systems allow symbols in the alphabet to have multiple rewriting rules each with an associated probability of being selected. Evaluating stochastic L-system inference with 960 procedurally generated L-systems with multiple sequences of strings as input found the following: 1) when 3 input sequences are used, the inferred successors always matched the original successors for systems with up to 9 rewriting rules, 2) when 6 sequences of strings are used, the difference between the associated probabilities of the inferred and the original L-system is approximately 1%. Parametric L-systems allow symbols to have multiple rewriting rules with parameters that get passed during rewriting. Rule selection is based on an associated Boolean condition over the parameters that gets evaluated to choose the rule to be applied. Inference is done in two steps. In the first step, the successors are inferred, and in the second step, appropriate Boolean conditions are found. Parametric L-system inference was evaluated on 20 known parametric L-systems. For 18 of the 20 L-systems where all successors were non-empty, the successors were correctly identified, but the time taken was up to 26 days on a single core CPU for the largest L-system. The second step, inferring the Boolean conditions, was successful for all 20 systems in the test set. No previous algorithm from the literature had implemented stochastic or parametric L-system inference.

Inferring L-systems of greater complexity algorithmically can save considerable time and effort versus

constructing them manually; however, perhaps more importantly rather than relying on existing knowledge, inferring a simulation of a process from data can help reveal the underlying scientific principles of the process.

Acknowledgements

I have to begin by acknowledging Professor Ian McQuillan, my Ph.D. supervisor. Indeed, it can only be said that Ian is an excellent supervisor because he sincerely cares about the students in his lab, and he does his best to tailor their education to their own quirks. For me, this meant letting me run with the baseball (that's right, isn't it?), while always being available for advice and guidance. I certainly have a lot of fond memories of my time under Ian's mentorship, but I think one stands out in particular.

In the summer of 2018, Ian told me to stop researching new material as I had enough to support a thesis. I had delivered several papers to him at this point, so I'm sure he had visions (nightmares?) of spending much of his upcoming sabbatical revising my mediocre writing. So, when I was at the International Conference on Unconventional Computation and Natural Computation, and listening to Professor Julian Miller give a presentation on Cartesian genetic programming, when it hits me that this is what we needed to infer parametric L-systems. This is something Ian and I knew would be important for practical L-system inference, and hence would be a real crown jewel to this work. So I messaged Ian "Dude!!!![1] Cartesian genetic programming!! This is what we need to do parametric L-systems. I have to build this." I'm expecting to get a "Sounds good" because this is his default reply. Instead, I get a one word answer "No." I was honestly a little shocked for a moment, but then I messaged him back "No really, we need to do this." I can only imagine Ian was thinking "He's just going to do it anyway" before replying "Fine." Poor Ian.

Next, I have to acknowledge my M.Sc. supervisor, Professor Sabine Graf. In some sense, Ian should be thanking Sabine, because she taught me an incredible amount about how to actually do research properly. I can often hear Sabine's voice saying "Jason, you cannot do that. Research does not work that way." I just want Sabine to know that I was, in fact, listening, so her words were not in vain. Indeed, the success of this thesis is due in no small part to using proper research methods. Sabine has also continued to provide me with good career advice beyond what is required by a former supervisor, and I consider her a good friend.

This thesis would not exist if not for meeting Professor Przemyslaw Prusinkiewicz of the University of Calagry at the 1st Annual P^{2}IRC Symposium in 2016. During the symposium, I was trying to decide on a possible research direction, and had the thought to try to infer L-systems using artificial intelligence. On mentioning this to him, he gave me some very sound advice. He recommended that I not pursue this research as it "might be impossible." He could not have known that the *i-word* is an irresitable call to arms to me. At that moment, I knew I was going to research L-system inference. As testimant to his character, and excellence as a scientist, years later he would compliment me by saying that he was impressed with my progress, and noting that "Sometimes it takes a new set of eyes to solve a problem."

I would like to acknowledge my good friends in no particular order: Maurice "Flamin' Moe" Richard, Darlene "Darrior" Richard, Jim "Jimio the Socktrooper" Andrews, Trudy "The Usurper" Andrews, Louis "Almost Evil" Veniot, and Russell "Karl" Melanson for their friendship (axiom detected), and many game

[1] I call him dude when I'm excited.

nights. “It’s a Wondeful Life” can make the argument for quantity all it likes, I’ll take quality everytime, and these are truly excellent friends, who have always been there for me. Well, mainly they put up with me, and I’ll take what I can get. My two thesis cats, Catdaki and Paladin, provided companionship while working long days and nights. Finally, I’ll mention my aunt here, Kathy Pashko, because she asked to be included.

I never knew how much joy could be had from fatherhood until my children, Elijah and Morgan, came into the world. I remember so well how they could fit on my forearm. Now, they are adults, and I could not be happier for them. They have become such amazing people, and they should be proud of everything they do, and will accomplish. I appreciate that they feel they can turn to me for advice and love, and that relationship does keep me encouraged since I know I must try to be a good example. I would also like to thank Karen Bernard for 16 wonderful years of marriage, and for being such a great mother to our two children.

To finish these acknowledgements, I have to mention my mother and father, Lorraine and Charles. My mother was an accountant, and she worked very hard to make sure that I had a roof over my head, and food in my belly. She is a good mom, better than she realizes, and I have many fond memories of our time together. She has asked me to explain my work to her from time-to-time, which is useful for helping solidify what I’m doing in my own mind. When I was being bullied in junior high school, she told me to “just hit him”, and she would back me up with the school. So, I did the next time I was attacked. This helped me become more resilient, and always stand up for what is right. My father, an artist, died on August 11^{th} of 2017. Thinking back over my memories of him, I can see that he was trying to raise me to be tough and resilient. What I really learned from him was a sense of humour, an irreverence towards life, and how to bring joy to people by entertaining them. When he died, I decided that his influence should not go out from the world, so I picked up his banner, and I’ve done my best to bring laughs and wisdom to others as best as I can. I’m sure he would be happy that I’m finishing my degree, but I hope he would be proud of my efforts to represent his love of life.

“Life is an incomprehensible joke. Then you die, and are told the punchline.”
— Charles Edward Bernard a.k.a. The Woodchuck.
Image used with the permission of the CBC.

Dedication

This thesis is dedicated to the people who have provided me with a lifetime of bonuses to my attributes, skills, and several passive buffs.

- **Charles Bernard and Lorraine Hamilton**: +100% chance to exist. +1 to charisma, intelligence, and wisdom.
- **Elijah and Morgan Bernard**: +1 to all attributes. +2 bonus to all saving throws.
- **Karen Bernard**: +2 children. +1 to wisdom.
- **Maurice Richard and Jim Andrews**: +1 to dexterity and intelligence. +2 bonus to saving throws versus boredom.
- **Ian McQuillan, Sabine Graf and Charlie Carman**: Collectively, +3 to intelligence, and +3 to profession (scientist).
- **Derby, Spy, Ninja, Catdaki and Paladin**: +3 bonus to sense motive checks. +3 bonus to move silently checks. +1 bonus to saving throws versus insanity.
- **Ms. Sorel**: +1 to intelligence. +2 bonus to confidence.
- **Christine Belliveau and Erin Stusick**: Each provided +1 bonus to saving throws versus mental effects.
- **Danielle Leclerc, Shannon Josdal, and Sylvia Phung**: +6 to performance (flute), +3 to performance (vocal), and +1 to performance (piano) collectively.
- **Andre Coulombe and Guy Salter**: +2 to unarmed melee attack rolls.
- **Canadian Armed Forces**: +1 to strength and charisma. −1 to constitution. +1 to profession (officer).

Contents

List of Tables

List of Figures

List of Abbreviations

0L-system	Context-free L-system
CGP	Cartesian Genetic Programming
D0L-systems	Deterministic Context-free L-system
(k,l)-systems	Context-sensitive L-system (with a maximum context of k characters to the left and l characters to the right)
DFS	Depth First Search
ES	Exhaustive Search
GA	Genetic Algorithm
L-system	Lindenmayer System
MCMC	Monte Carlo Markov Chains
OSoS	Ordered Sequence of Symbols
PMIT	Plant Model Inference Tool
PMIT-D0L	Plant Model Inference Tool for Deterministic Context Free L-systems
PMIT-S0L	Plant Model Inference Tool for Stochastic Context Free L-systems
PMIT-S0L+PL	Plant Model Inference Tool for Stochastic Context Free L-systems with prefix limitation extension
PMIT-S0L-PL	Plant Model Inference Tool for Stochastic Context Free L-systems without prefix limitation extension
PMIT-PARAM	Plant Model Inference Tool for Parametric L-systems
SL-system	Stochastic L-system
S0L-system	Stochastic Context-free L-system
vlab	virtual laboratory

Preface

The following thesis presents an investigation into the inductive inference of Lindenmayer systems (L-systems). The main research goal is to develop tools to infer L-systems only from a sequence of strings given as input. As a result of this work, the following papers have been written, and either published or are under review. Additionally, the contributions to this work by myself, Jason Bernard, and my supervisor, Ian McQuillan, are provided below.

Citations

Chapter 3

- Jason Bernard and Ian McQuillan. Techniques for inferring context-free Lindenmayer systems with genetic algorithm. *Swarm and Evolutionary Computation*, 2020. (submission #SWEVO_2020_39): on arxiv at URL: : `https://arxiv.org/abs/1906.08860`

- Jason Bernard and Ian McQuillan. New techniques for inferring L-systems using genetic algorithm. In *Proceedings of the 8th International Conference on Bioinspired Methods and Their Applications*, volume 10835 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2018.

- Jason Bernard and Ian McQuillan. A fast and reliable hybrid approach for inferring L-systems. In *Proceedings of the 2018 International Conference on Artificial Life*, volume 30, pages 444–451. MIT Press, 2018.

Chapter 4

- Jason Bernard and Ian McQuillan. Stochastic L-system inference from multiple string sequence inputs. *Engineering Applications of Artificial Intelligence*, 2020. (submission #EAAI-20-235): on arxiv at URL: `https://arxiv.org/abs/2001.10922`

- Jason Bernard and Ian McQuillan. Inferring stochastic L-Systems using a hybrid greedy algorithm. In *Proceedings of the 30th International Conference on Tools with Artificial Intelligence*, pages 600–607. IEEE, 2018.

Chapter 5

- Jason Bernard and Ian McQuillan. Inferring temporal parametric L-systems using Cartesian genetic programming. 37*th International Conference on Machine Learning*, 2020. (under review)

Author Contributions

Jason Bernard performed the following tasks.

- Developed the overarching algorithms.
- Discovered and developed novel space reduction techniques.
- Adapted existing techniques from the literature as space reduction techniques.
- Developed the original concept that a greedy algorithm could be used to infer stochastic L-systems.
- Performed the statistical analysis of existing L-systems.
- Developed the procedural L-system generation tool based on the statistical analysis.
- Extended a concept found by Ian McQuillan (see below) that all L-systems that are not strictly context-free and deterministic appear initially to be stochastic.
- Developed the idea of using Cartesian Genetic Programming to transform a stochastic L-system into a parametric one.

Ian McQuillan made the initial observation that some types of L-systems appear initially to be stochastic. All papers were initially written by Jason Bernard, and subsequently revised by Ian McQuillan. Ian McQuillan supervised all of the research.

Chapter 1

Introduction

1.1 Lindenmayer systems

In 1968, Lindenmayer [41] proposed a formal grammar system, later called Lindenmayer systems (L-systems). He described how they could be used as a mathematical model for how multicellular biological systems can grow. While described in further detail in Chapter 2, L-systems use rewriting rules that replace symbols in a string with a replacement string. When this string replacement is applied to every letter of an entire string in parallel, and then this parallel rewriting iterates, reoccurring symbol patterns appear in the strings, along with self-similarity. Certain symbols of the alphabet can be given a visual and/or a mechanical meaning for use within simulation software. The most common method to provide a visual meaning is called the turtle interpretation [62], which gives a turtle a state consisting of a position (in 2D or 3D) and an orientation, and the letters of a string describes how an image can be drawn while the state changes (described in the next chapter). For example, the string $F+F$ could mean to draw a line of one unit long, turn left by a fixed number of degrees, and then draw another line of one unit long. The next string produced (by rewriting each letter in parallel) would describe the next step of the simulation. Other characters beyond those defined as part of the turtle interpretation can be used and do not directly get visualized, but they can have some mechanical interpretation. For example, if the symbol A is replaced by $F+F$, then this could imply that there is a mechanism within the process (represented by A) that produces the visualization ($F+F$) at the next time step. A deeper discussion on simulation and modelling is provided in the next chapter.

While Mandelbrot [43] and Wolfram [80] never used L-systems explicitly, they did describe ways in which self-similarity occurs frequently in nature, and they used related formalisms to represent them. Mandelbrot [43] described self-similarity as follows: "When each piece of a shape is geometrically similar to the whole, both the shape and the cascade that generate it are called self-similar." Mandelbrot described methods to represent fractals, and Wolfram investigated how such patterns can be produced by cellular automata. Later it was found that L-systems can also be used to represent fractals and some patterns that can be produced by cellular automata. Cellular automata are computational models that operate on a grid of cells, and by applying locally-defined rules to the cells, they can exhibit complex behaviours. But L-systems are particularly suited to describing branching models, and indeed, one of the main application areas of L-systems have been in algorithmic plant modeling [16, 48, 62, 65, 71]. However, they have also been used in

other diverse natural or human-engineered areas such as other biological systems [20,26,82], the modeling of buildings [50], and music [81]. Finally, L-systems were studied extensively within the theoretical computer science community, especially during the 1970s [69]. Many properties were studied, such as the computational power of various types of L-systems, and the various growth patterns that are possible. L-systems were also used successfully as a model for languages with parallel computation [70], and to design and program virtual robots [33–35].

1.2 Manual and Automatic L-systems Prediction

One challenge when using L-systems to model a specific process is finding an appropriate L-system. The most successful approach has been to build the L-system by hand (e.g., [2,16,50,53,55,60,62–65,82]) using detailed and extensive measurements, existing knowledge regarding the process, and possible external influences. This process has been described by Galarretta et al. as requiring "tedious and intricate handwork" that could be improved by an algorithm to "infer rules and parameters automatically from real ... images". Perhaps no work epitomizes this more so than Nishida's work on modeling Japanese Cypress trees [53]. Nishida took many images of Japanese Cypress branches (examples are shown in Figures 2.3a and 2.3c of Chapter 2), and then manually segmented the branches. An analysis of the different segments across all of the imagery revealed 42 distinct mechanisms involved in the tree's growth, which he used to construct a stochastic L-system, which Nishida used to produce a good but not exact simulation. After so much effort, it is possible that Nishida might have agreed with Galarretta's assessment that inferring the mechanisms algorithmically from imagery would be welcome.

The goal of automated L-system inference is to automatically find an L-system to simulate a process from data captured. That is, we would like to automatically learn the simulation program (or the developmental program) from data. In particular, a major goal is to infer an L-system from a time sequence of images. Ideally, high quality 2D or 3D imagery, as appropriate, of the process over time is captured. One method that could possibly be used to infer an L-system from a temporal sequence images of involves two steps: 1) segmentation into string descriptions and 2) inference of L-systems from the strings. Segmentation involves identifying the different elements of the physical structure (of a single plant at a single time point from images) and assigning each segment to a symbol. If done over an entire image, then a string can be produced that could reproduce (or approximate) the structure within a simulator. When done over a sequence of temporal images, then a sequence of strings can be produced where each string describes a step of the process. This process is similar to the methodology used manually by Nishida with the Japanese Cypress trees [53] (others have also used the process of taking measurements to produce L-systems [55]). There has been considerable success as of late on automatically segmenting imagery (e.g., [21, 72, 78]) in a similar fashion. In [78], a 3D sonic digitizer was used to identify leaves on *Oryza sativa* (japonica rice) as it was growing. They then used the identification of the leaf and tiller components (the two main components of rice grass), coupled

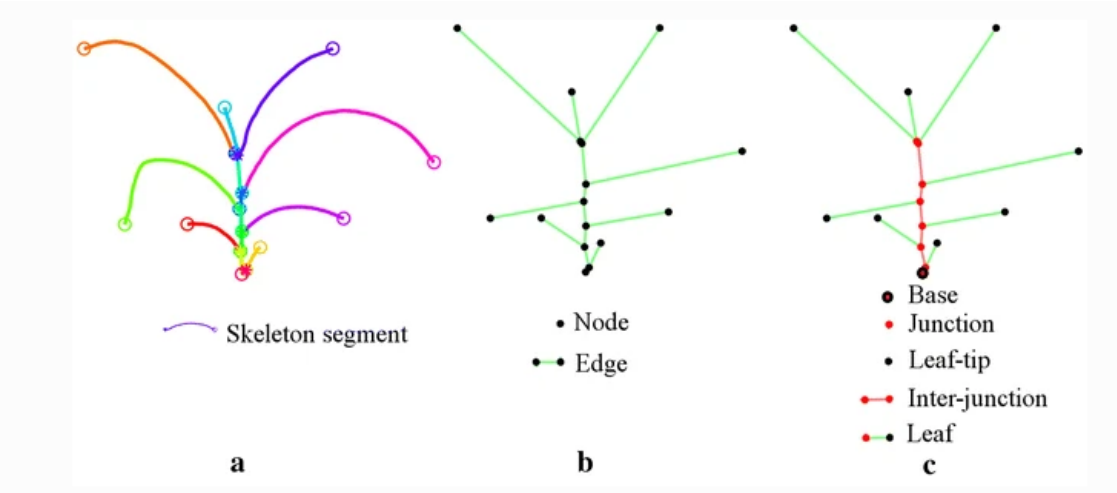


Figure 1.1: "Plant architecture determination: **a** plant skeleton with each leaf marked with different colors; **b** graphical representation of the plant with nodes and edges; and **c** plant body-part labeling" [21]. No modifications were made to this image. Image is used under CC-BY-4.0 license.

with extensive environmental data about the crops, to develop a parametric L-system as a model of the morphological responses to the environment. Choudury et al. [21] used RGB cameras to capture images of a plant growing over time from different angles. The imagery was then reduced to lines of 1 pixel width (skeletonization), the individual skeleton segments were found, and the segments were converted into a graph. The nodes and edges were mapped to the phenotypic components of the plant. This process is shown in Figure 1.1 and is an example of component-based phenotyping. After identifying the components, it is then straightforward to map them onto an approximate string of symbols that would draw the graph with a simulator. The second step is the focus of this research. Suppose n images captured for a process have been converted into a sequence of n strings. The next step is to infer an L-system that produces exactly those strings as its first n steps. This is known as inductive inference of L-systems (described in detail Chapter 2).

1.3 The Impact of L-systems

Here, we will discuss some of the advantages of inferring L-systems beyond the goals of component-based phenotyping. It is notable to point out that L-systems are highly compact compared to the images. It consists of only a short formula, it is reproducible, and it can be easily simulated. This allows imagery to be reproduced by simulation on demand instead of storing the imagery itself. Simple L-systems can provide a high-level mechanical model of a process; i.e, they can help identify the mechanisms that have an effect on the development. More complex L-systems, such as parametric L-systems, can also describe the phenotypic responses to environmental factors as was seen in [57, 78]. If a parametric L-system can be found that is a high-quality model of the process, then it is possible to simulate scenarios *in silico* that have never occurred. One can simulate and visualize the result of certain hypothetical environmental scenarios. While a single

L-system can be informational (as described above), if multiple L-systems are produced from multiple copies of a process (e.g., a population of plants), then it is possible to compare and contrast the L-systems. It is also possible to infer a single stochastic model that can describe an individual plant, or a population of plants, which could be derived from a single genotype, or a species, which can be compared or altered. Furthermore, the use of imagery produced from an L-system that operates in a stochastic manner has recently be shown to be effective at extending ground truth data sets for computer vision applications; e.g., leaf counting [76]. However, as has been stated, it can be difficult to even produce a single L-system for a process to say nothing of a large population; hence, to fully capitalize on the benefits of L-systems, it is proposed that L-systems need to be inferred algorithmically from data about the process.

1.4 Potential Impact for Automatic L-system Inference

Over the years, there has been some research into algorithmic L-system inference [18, 35, 37, 47, 51, 52, 71], and while successful in some sense, the approaches to date have serious limitations. Some of the tools are intended to function as an aide to the expert when making an L-system by hand [18,47]. In such approaches, typically, a population of L-systems is visualized to the operator who then picks ones that are aesthetically closest to the desired result. The tool then produces some variation on the selected L-system(s) until the user is satisfied with the result. While useful at reducing some of the effort involved in inferring an L-system, such tools are obviously not fully automated. The use of aesthetics means the tools are not ideal for finding an optimal matching L-system for a process. Some other automated tools are tied very closely to specific research domains [20, 50, 68]. Again, these tools tend to focus on producing aesthetically pleasing results as opposed to inferring an L-system for a target process. Additionally, these tools often use existing knowledge about the process. If suitable information about the process is available, then this can be successful, although even in such cases it adds the complication of deciding how to incorporate the knowledge into the inference tool. Algorithms that do not rely only on any existing knowledge or aesthetic-based outcomes, but rather require only a sequence of strings as input, can be called a generalized inference algorithm [52,71]. The benefit of functioning with only a sequence of strings as input is such an algorithm can be applied to any research domain for which a string sequence can be produced. Unfortunately, the existing algorithms [52, 71] in the literature only work for the simplest types of L-system (deterministic context-free L-system, described in the next chapter), and even then only for simple forms of those. However, this does suggest that generalized L-system inference is at least possible, although as stated by Nakano, it is "immensely complicated" when dealing with larger alphabets [52].

While discussed in greater detail in the next chapter, a brief definition is provided here for some L-system related concepts to allow an understanding of the motivation for this research provided in the next section. An L-system consists of an alphabet (V), an initial word (w), and a finite set of rewriting rules (P). The rewriting rules are of particular interest as there are many different variants that alter how a rewriting rule is

selected for an instance of a symbol A. The simplest is deterministic context-free rules, where each is of the form $A \rightarrow x$. A is the symbol being replaced (predecessor), and x is the replacement word over V (successor), and there is exactly one rule with each letter as predecessor. This research also investigates the more complex stochastic and parametric L-system inference. A stochastic L-system allows each symbol in V to have a list of one or more successors. Each successor has an associated probability of being selected such that the sum of probabilities for a symbol equals 100%. Successor selection is done according to the probabilities. Deterministic parametric L-systems allow each symbol to have a list of one or more successors, with a set of associated parameters. The parameters can get changed and passed to successors during rewriting. When there are two or more successors, each one has an associated Boolean condition using the symbol's parameters. Successor selection is done for an instance of a symbol, which has its own parametric state, by finding the associated Boolean condition that is true given the state. For all types of L-systems, they can generate a sequence of strings (which can then get visualized by a simulator) called a developmental sequence. The goal with inductive inference is to take only some initial portion of the developmental sequence as input (a description of some portion of the process), and predict an L-system that could generate it.

1.5 The Challenges and Goals of Practical Generalized Inductive L-system Inference

With those definitions, this thesis presents an investigation into the generalized inference of L-systems. The main research goal is to develop algorithms for the inductive inference problem; i.e., to find an L-system that produces a sequence of strings provided as input, which can be used by a simulator to model the process. This work does not: 1) investigate development of a simulator, as high-quality simulators already exist (e.g., virtual laboratory [39]), 2) model any specific process that does not already have a known L-system; the focus is on the inductive inference problem, or 3) investigate the segmentation problem (while useful to provide inputs for an inductive inference algorithm, it is a distinct problem). The motivation for this research is to take steps towards practical L-system inference and the several successful steps taken towards this goal are the main contribution of this work.

It might be sufficient to say that if an L-system can be inferred from a sequence of strings, then this represents successful generalized inference of an L-system. However, it is argued that some consideration must be given to how the strings would be produced in practice. De La Higuera [22] states that dealing with noisy data (strings with insertion and deletion errors) is a major limitation of inductive inference algorithms. Indeed, some reflection should also be taken on the expected underlying mechanisms for many processes. As mentioned above, existing algorithms can only infer the simple deterministic context-free L-systems with severe limitations [52, 71]. Suppose that an algorithm could infer any type of deterministic context-free L-system from a sequence of strings with no limitations. Such an algorithm is likely still insufficient to infer L-systems from imagery for several reasons: 1) processes are unlikely to have simple deterministic mecha-

nisms (deterministic given the information included in the L-system), 2) strings produced by a hypothetical segmentation algorithm will likely have imperfections, 3) not all parts of the strings are visualizable with images, 4) it is unclear how to match the right moment to acquire an image (and other data), with the developmental rate of the process, so as to not miss any critical developmental events. These obstacles are discussed next.

1.5.1 The Necessity of Complex L-systems

Many processes in nature do not use simple deterministic mechanisms at the level of abstraction modelled by L-systems. Consider a set of roses; while each are similar enough to be identified as a rose, they are not identical, which suggests that the process is not deterministic at a genetic level. Since differences are observed between different roses, it is possible that there are stochastic elements to the algorithm encoded within the rose, or there are elements that depend on factors not included in the model (such as unknown or not captured external factors). These additional factors could be modelled as parameters in an L-system. Additionally, a practical inference tool needs to be able to handle non-mechanical influences on the input strings. For example, while scanning a plant, it is possible that some leaves (or an entire branch) might be removed by accident from handling (or some other external force), which could be represented as a stochastic rule (e.g., there is some, hopefully small, chance a graduate student will drop the sample). It is argued in this thesis that there exists an algorithm for inferring more complex L-systems (either stochastic or parametric) as follows.

As a first step, such influences, whether internal mechanisms or completely external to the process, can be thought of as being stochastic. To continue the examples above, there is some chance that a rose may decide to produce a thorn at step x and some chance that it will not. Similarly, there is some chance that a branch in previous imagery is now simply gone due to mishandling. While a stochastic model may be sufficient for external influences, it is likely that the internal mechanisms for many processes are not stochastic, but are influenced by environmental and other factors. As an example, for plant growth, the influencing factors could be soil nutrients, water stress, temperature, amount of light over time, etc. Thus, it would be expected that for at least some processes, a parametric explanation of their mechanisms would produce a higher quality model. Hence, even when external influences are not known or captured, it is still possible to build a stochastic model that can describe part of its behaviour.

1.5.2 Assumptions on the Strings Used as Input

With respect to the relationship of segmentation algorithms to the inference tools presented herein, a significant limitation of the current work in this thesis must be acknowledged. The developmental sequence produced by an L-system will have all of the symbols visible. Some initial portion of a developmental sequence is used as input to infer an L-system. In contrast, strings produced by an image segmentation algorithm might only have some symbols visible, since only the external physical structure is (easily) observable, and

the unobservable mechanisms are effectively projected onto the structure. For example, consider the third image in Figure 1.1. If this image were generated by an L-system, then consider the next image generated. It may have a new leaf growing from the apex, but all that exists at the apex in the image of Figure 1.1 is an "inter junction". The segmentation algorithm would need to know that this is an apex, and also know that leaves can grow from the apex to place a mechanical symbol (A) at the correct location of the string. Recognizing an apex (and other similar structures from which growth can occur) is perhaps resolvable in some cases; however, the second issue defeats the purpose of not requiring advance knowledge about how the plant grows. Thus, it is prudent to assume instead that instead no A will be present in the strings, and it is essentially, and invisibly, projected onto the external structure.

For L-systems, this type of projection of an unobservable mechanism onto the structure could be accomplished via a mathematical operation on the developmental sequence called a homomorphism. A homomorphism maps two or more symbols onto the same observable symbols; e.g., the symbol A (a mechanism), and F (a line) could be mapped onto F (of the A could be mapped to the empty string); thereby, obfuscating the A. For the purposes of this thesis, no homomorphisms are assumed to exist; i.e., it is assumed that a hypothetical segmentation algorithm would produce every symbol present in the strings of the developmental sequence. This is recognized as being unrealistic; however, it is a reasonable first step to solve this easier (but not easy) problem with perfect strings, and the majority of existing approaches in the literature make this same assumption (with the exception of [51]). The extension of the inductive inference problem to take into account homomorphisms will be discussed in Chapter 6. That is, the inference of L-systems from only the obviously visible parts of the developmental sequence will be discussed.

1.6 Structure and Overview of the Thesis

This first chapter has presented a conceptual goal of working towards practical inductive L-system inference. The journey to that goal has been argued to first investigate the inference of deterministic context-free L-systems. Indeed, until such L-systems can be inferred without severe limitations, there is little point in moving on to more complex L-systems. Furthermore, more complex types of L-systems can start with successful techniques on the simplest type. The second subgoal is to investigate the inference of stochastic L-systems. Such L-system are both useful for practical modelling [57, 62] and extending data sets [75], but also take a step towards handling more realistic mechanisms. The final goal of this thesis is to investigate parametric L-system inference. Parametric L-systems can provide the highest quality models due to the expected controlling relationship of external factors to the mechanisms of a process (e.g., [62, 78]). These goals are divided into the following chapters.

Chapter 2 provides a more detailed discussion on existing concepts and theories on L-systems. This is followed by a discussion of the applications of different types of L-systems on modelling natural and human-engineered processes. Finally, the chapter provides a detailed look at different existing L-system inference

approaches from the literature.

Chapter 3 discusses the work done to infer deterministic context-free L-systems with less limitations than the current approaches in the literature. The aim was to be able to infer such L-systems of the greatest complexity that could be found in the literature[1]. Ultimately, the algorithm developed, called the Plant Model Inference Tool for Deterministic Context-Free L-systems (PMIT-D0L) is able to infer all of the L-systems in the test set in seconds, which corresponds to a maximum of 31 characters and a sum of 281 symbols in the rewriting rules. The state-of-the-art prior was 2 symbols in the alphabet and a maximum of about 20 total symbols in the rewriting rules [52, 71]. The improvement was accomplished by treating L-system inference as a search problem and by using a novel approach for encoding the L-system inference problem. Additionally, several techniques were found to reduce the search space size.

Chapter 4 describes an algorithm for inferring stochastic context free L-systems (S0L-systems), which as discussed above is an important step towards practical L-system inference. One of the main challenges was to find a way to pick a solution from the multitude of possible answers (this is a major difference between stochastic and deterministic L-systems). The solution presented in the chapter is to pick from the candidate L-systems examined using a search algorithm, the S0L-system with the greatest probability of producing the input strings under a parsimonious argument[2]. To find that S0L-system, the algorithm developed, called the Plant Model Inference Tool for Stochastic Context-Free L-systems (PMIT-S0L) uses a hybridization of a greedy algorithm with a search algorithm (exhaustive and genetic algorithm were evaluated). PMIT-S0L was mainly evaluated on procedurally generated L-systems as only one S0L-system could be found published in the literature, the aforementioned Japanese Cypress tree model [53]. It was found that PMIT-S0L was able to infer every L-system with up to 9 rewriting rules in less than a self-imposed time limit of one day (with some different assumptions it could infer S0L-systems with up to 12 rewriting rules). Additional experiments examined the effect of having more than one sequence of strings as input. It was found that having 3 string sequences was sufficient to always infer the original rewriting rules, but not the exact probabilities. For the associated probabilities, it was found that 6 input sequences approximately minimized the difference between the associated probabilities of the original and solution L-system at approximately 1%. There did not previously exist a generalized L-system inference tool for S0L-systems in the literature.

As might be expected from the discussion above, Chapter 5 describes a tool for inferring parametric L-systems as the next step towards practical L-system inference. The Plant Model Inference Tool for Parametric L-systems (PMIT-PARAM) is introduced. Is uses a combination of the PMIT-S0L algorithm described in Chapter 4 and Cartesian Genetic Programming (CGP) [46]. PMIT-PARAM functions by attempting to find the stochastic L-system with the greatest probability of producing the input strings (again, it selects that with the greatest probability from the candidates examined). Then, using CGP, the S0L-system is converted into

[1]Complexity could be measured in a number of different ways. The main focus herein was on the largest alphabet size and the sum of the number of symbols in the rewriting rules since, as will be seen, this has a direct effect on search space size.

[2]This is not necessarily the S0L-system with the greatest probability of producing the input strings, unless the search algorithm guarantees to find that L-system; e.g., an exhaustive, and quite probably intractable, search of all L-systems.

a parametric L-system by replacing the selection probabilities on the rewriting rules with Boolean conditions (stochastic and parametric L-systems are described more completely in the next chapter). PMIT-PARAM was found to be able to infer parametric L-systems with up to 9 rewriting rules from a test set of parametric L-systems in the literature, although the transformation (second) step was able to work on L-systems with up to 27 rewriting rules. As with S0L-systems, there did not previously exist a generalized algorithm for inferring parametric L-systems (an algorithm developed by Curry [18] infers parts of parametric L-systems so no comparison can be made).

Chapter 6 concludes the work with a summary of the findings in this thesis, and describes possible future directions for this research.

Chapter 2

Review of L-systems and L-System Inference

2.1 L-System Theory and Concepts

This section defines L-systems and describes concepts related to L-systems. The basic elements and function of an L-system are first described. Then, several different extensions to L-systems are discussed. This is followed by a description of how the strings created by L-systems may be interpreted.

First, some basic definitions are given. An alphabet V is a finite set of symbols, for example, an alphabet might consist of the set of symbols $\{A, B, C, +, -\}$. A word or string is any sequence of symbols over the alphabet. For example, $ABB+C$ is a string over V. The set of all words over V is denoted by V^*, which includes the so-called empty word denoted by λ. Also, given a word $x \in V^*$, $|x|$ is the length of x. All kinds of L-systems consist of an alphabet (V), a set of initial words, and a finite set of rewriting rules or productions (P, of different forms depending on the type).

2.1.1 Context-Free L-systems

We will start by defining the simplest types of L-system. A context-free L-system (0L-system) $G = (V, X, P)$ consists of three components: a finite alphabet (V), a set of initial words X called the axioms, and a finite set of rewriting rules or productions (P). The rewriting rules are of the form $A \to x$, where $A \in V$ is called the predecessor, and x is a word over V called a successor of A, and there must be at least one rule for each letter as predecessor. For example, a rule might be $A \to AB$, which means to take a symbol A in a word and rewrite it to an AB. When a symbol produces only itself, this is called an *identity production.* When writing L-systems, if no rule is specified for a symbol $A \in V$ then the *default production* is used, which is often taken to be the identity production $A \to A$.

The process of applying the rewriting rules is called a derivation step, denoted as $\Rightarrow$, and it involves simultaneously taking each symbol in a word, and replacing it with some successor. So $a_1 \cdots a_n \Rightarrow x_1 \cdots x_n$, where $a_i \in V$, and $a_i \to x_i \in P$ for each $1 \leq u \leq n$. So, a derived string can be thought of as the ordered concatenation of the successors of the symbols of the deriving word. Derivation steps are applied iteratively starting with an initial string, and then applied to each of the resulting strings. That is, $w = w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \cdots$, where $w \in X$. The first n such words $(w_1, \ldots, w_n)$ is called the length-n developmental sequence.

2.1.2 L-Systems Extensions

All L-systems share the common concept of parallel rewriting of symbols using rules; however, the method of selecting a rewriting rule for a symbol can vary. For example, the selection of a rewriting rule for a symbol may be deterministic, stochastic, or dependent on the value of a parameter. The following sub-sections describe the different extensions of L-systems that are most relevant to this thesis, and how the application of the rewriting rules vary between them. The different extensions are described as follows from Kari et al. [38] (unless otherwise noted) beginning with context-sensitivity, which is unique in that it can be used in conjunction with any other type of rule, and then describing the various other types of variants.

Context-free versus Context-Sensitive L-Systems

Context-sensitivity describes using a symbol's neighbours as a mechanism for selecting a rewriting rule. For example, in the word $ABCBA$, the first instance of B has an A as a left neighbour and C as a right neighbour, while the second instance of B has a C as a left neighbour and an A as a right neighbour. This difference in neighbours for each instance of the B symbols can provide a basis for selecting a different rewriting rule. When such context is used to select a rewriting rule, the L-system is called context-sensitive. This is contrasted by a context-free L-system where the rewriting rule for an instance of symbol is not effected by that symbol's neighbours. So, in a deterministic context-free L-system both B symbols would use the same rewriting rule, while in a deterministic context-sensitive L-system they might have different rewriting rules. However, if u and v are contexts for a symbol A, then in a deterministic context-sensitive L-system, all instances of A with context u and v would have the same successor. Table 2.1 shows examples of context-sensitive rules where the part before $<$ designates a left context and the part after $>$ designates the right context. Assuming that any unspecified rules have an identity production, the word $ABCBA$ becomes $ADBBE$. A context-sensitive L-system is called a (k, l)-system, where k indicates the maximum number of symbols used for left context, and l is the maximum number of symbols for the right context. So the example in Table 2.1 is a $(1, 1)$-system since it relies only on maximum of one character as the context to either side. Context-sensitive L-systems involving lengths greater than 1 do not seem to appear very commonly in the literature, with the only example found being used for describing a mechanism for cellular growth [62]. While speculative, this might be due to the difficulty for a human to look at long strings and observe relationships between context and productions. The example referenced for cellular growth used *a priori* scientific knowledge to expertly craft the L-system to have longer context lengths. Another possibility is context-sensitivity with length greater than one may be replaced by context-sensitive rules that essentially pass the context like a signal one symbol at a time over several derivation steps. While this would not be exactly the same as the original developmental sequence, it could provide a close enough approximation for the purposes of a simulation.

If C is right of a B, then replace B with D	$B > C \rightarrow D$
If B is left of an A, then replace A with E	$B < A \rightarrow E$
If B is left and right of C, then replace C with B	$B < C > B \rightarrow B$

Table 2.1: Examples of context-sensitive rewriting rules

S0L-system rule	$A \rightarrow AB : p = 0.5$ $A \rightarrow BA : p = 0.5$
stochastic (1,0)-system rule	$A < B \rightarrow AA : p = 0.1$ $A < B \rightarrow BB : p = 0.75$ $A < B \rightarrow AA : p = 0.15$
stochastic (1,1)-system rule	$A < A > B \rightarrow BB : p = 0.25$ $A < A > B \rightarrow AB : p = 0.75$

Table 2.2: Examples of stochastic rewriting rules.

Stochastic L-Systems

The following formal definition of S0L-systems is based on [24]. Formally, a stochastic context-free, or S0L-system, [24] is a quintuple $G = (V, X, P, p, I)$, where V is an alphabet, X a set of words over V, P is a finite set of productions $a \rightarrow u, a \in V$, $u \in V^*$ (where a is called the predecessor and u is the successor of the production) with at least one production for each letter, p is a function from P to $(0, 1]$ such that, for all $A \in V$, $\sum_{A \rightarrow \alpha \in P} p(A \rightarrow \alpha) = 1$, and I is a function from X to $(0, 1]$ such that $\sum_{x \in X} I(x) = 1$. I is used to select the axiom.

Given x, y as words over V, a derivation d of x to y of length n consists of two items:

1. a trace, which is a sequence of $n + 1$ words $w_0, \ldots, w_n$ such that $x = w_0 \Rightarrow \cdots \Rightarrow w_n = y$

2. a function σ from the set of pairs $\{(i, j) \mid 0 \leq i < n, 1 \leq j \leq |w_i|\}$ into P such that, for i from 0 to $n - 1$, if $w_i = a_1 \ldots a_m, a_j \in V$, then $w_{i+1} = \alpha_1 \ldots \alpha_m$ where $\sigma(i, j) = (a_j \rightarrow \alpha_j)$ for j from 0 to m.

It is also necessary to define concepts such as the probability of a derivation occurring, which will be done in Chapter 4. Stochastic L-systems appear in the literature in a variety of research areas such as plant modelling [17], protein folding [20], and generating music [81]. They can also be defined to be context-sensitive. Table 2.2 shows some examples of stochastic rewriting rules.

Parametric L-Systems

A parametric L-system has rewriting rules where a successor is selected based on a symbol's state; e.g., time and other environmental factors [62]. For example using time, a symbol may use one rewriting rule before

$A(T) : T < 3$	$\rightarrow A(T+1)A(T+1)$
$A(T) : T \geq 3$	$\rightarrow A(T+1)A(T+1)B(T)$
$B(T) : T < 4$	$\rightarrow B(T+1)$
$B(T) : T \geq 4$	$\rightarrow B(T+1)A(T-4)B(T+1)$

Table 2.3: Example of parametric rewriting rules with a single parameter time (T).

generation 3 and a different rule starting with generation 4. For parametric L-systems each rule may have its own parameters and conditions.

Each parametric rule consists of the predecessor, successor, one or more real number parameters and a logical condition to indicate when it should be selected. The conditions may consist of the parameters, arithmetic operators, and Boolean operators. For an instance of a symbol, the rule selected will be the one where the condition evaluates to true. Parametric L-systems most often have no identifying letter in the prefix and are simply called parametric L-systems [38, 40, 62]. Table 2.3 shows an example of a parametric D0L-system using a single parameter time (T), although as with other systems, parametric rewriting rules may be formed using other extensions, e.g. a rule may be stochastic and parametric.

2.1.3 Interpreting Symbols in L-Systems

Two broad categories of symbols exist: geometric and mechanical. A symbol can be geometric, mechanical, or both, although it is most common for a symbol to be one or the other. Geometric symbols are used to provide graphical instructions to the simulation software constructing the visualization of the L-system. L-systems are used quite often to produce visual simulations so first graphical interpretations will be discussed, starting with turtle graphics and then model-specific interpretations. This is followed by a discussion of how symbols may be interpreted mechanically for models.

Turtle Graphics Interpretation

The simplest turtle graphical system is imagined as a turtle on a 2D grid with a pen attached to it that then moves about and draws a curve along its path [54, 62]. This interpretation can be used to draw fractal curves [62] and has been used extensively for plant modelling [16, 17, 62, 63]. This interpretation uses four commands as its base: move forward with pen down (i.e. draw a line), move forward with pen up (i.e. do not draw a line), turn right and turn left. By commonly accepted convention [54, 62], the alphabet for these movements is $(F, f, +, -)$ respectively. The length of the movement desired for the symbols F and f along with the number of degrees to turn for the $+$ and $-$ symbols may be a globally defined constant, or defined by a parameter such as $f(2)+(20)$ with parametric L-systems, which would indicate to go forward two units, and then turn left by 20°.

Two major extensions are often used with the basic turtle interpretation described above. The first adds

Figure 2.1: A visual simulation produced by virtual laboratory [39] of an apple twig with blossoms [62]. Image used with permission of the copyright holder.

3D movement for the turtle [62] with the following commands (and symbols): pitch up (&), pitch down ($\wedge$), roll left (\), roll right (/) and turn around (|). As with the 2D turtle graphics commands, these may be of a fixed degree or permitted to vary by parameter, except for | that is always interpreted as a 180° turn. The second extension adds branching with the commands start branch ([) and end branch (]). To implement branching, a stack is used. A start branch command pushes the turtle's current position and orientation, i.e. state, onto the top of the stack. When an end branch command is reached, the state is popped from the top of the stack and is restored as the turtle's state. Using a stack in this fashion eases the implementation of nested branching as the top most state on the stack will always be the state just prior to the most recent branch start command.

Model Specific Graphical Interpretations

For some models, such as plants, certain elements are not easily drawn using turtle graphics as they would require a large number of turtle commands. For example, drawing a petal or leaf is certainly possible with turtle graphics but may require many commands, and it might be unnatural. To prevent words from becoming bloated by long command sequences, the turtle graphics interpretation is extended by symbols that represent model specific elements. When the symbol is encountered by the visualization process, the simulator (such as the virtual laboratory [39]) can use a customized graphical module to draw the element. This approach can be seen as a custom extension to turtle graphics; however, a model-specific graphical interpretation may be completely separated from turtle graphics as follows. Figure 2.1 shows an image produced by virtual laboratory (vlab) of an apple twig with blossoms. A subword of the string used to produced this image is $[!!))UUU((((/S////S////S////S////S////S]$. The S symbol is interpreted by a customized visualization module to draw the pink blossom petals; however, note that the turtle graphics symbols are still used to provide orientation in $3D$.

An additional example can be found in the work by Müller et al. [50] on using L-systems to model

buildings, they define an alphabet that represents the basic blocks that can comprise a building and then symbols for commands such as adding a floor, windows, types of facade, or selecting a type of roof. As above, during the visualization process, each symbol is drawn according to the desired interpretation rather than with classical turtle graphics. A final example of this can be seen in using L-systems to produce sheet music [81]. In this case, different symbols are translated into sequences of musical notes by extending the duration of a note or changing the pitch.

2.1.4 Mechanical Interpretations

A mechanical symbol represents a mechanism within the process being modelled. This is perhaps best understood by example. Mechanical symbols will always be model specific. In plant modelling, some mechanical symbol might represent the biological process that causes a branch to grow from a trunk [62], while in geological modelling a symbol might represent the forces that cause a curve to form in a river bed [68], and in protein modelling a symbol might represent the mechanisms that cause a fold to a particular secondary structure [20].

2.2 Applications of L-Systems

By giving the symbols in an alphabet a visual meaning in a simulator, L-systems are able to produce self-similar shapes that are recognized as underlying many natural processes [43, 62, 80]. Self-similarity occurs when a symbol, over one or more rewriting steps, produces itself and so allows the cycle to continue. For example, $A \Rightarrow BB \Rightarrow AAAA \Rightarrow BBBBBBBB$. Some degree of self-similarity can also occur in stochastic L-systems despite successors being selected by chance. This can be seen in the imagery (Figures 2.3a to 2.3d) produced by the stochastic systems developed by Nishida [53] for Japanese Cypress trees.

As previously mentioned, L-systems have had far reaching effects on many research areas [16, 20, 33–35, 48, 53, 62, 70, 71, 82]. The next subsection will provide a discussion on plant modelling as it certainly the area where L-systems have had the most impact. This will be followed by an overview of some of the other applications of L-systems to give a sense of their potential impact.

Consider the case of an L-system modelling a cellular automaton [34, 80]. In this case, the symbols 1 and 0 represent only whether a particular cell is on or off, and the rewriting rules define the automaton, or interaction between different cells. With cellular automata, the rules involve the state of the neighbours so a context-sensitive rule would be needed such as $0 < 1 > 1 \rightarrow 110$, which means that if a 1 has a 0 to the left ($<$) and a 1 to the right ($>$), then it produces a 110. Cellular automata may be visualized (an example of such a visualization is shown in Figure 2.2), so this example also shows how a symbol may be mechanical and graphical.

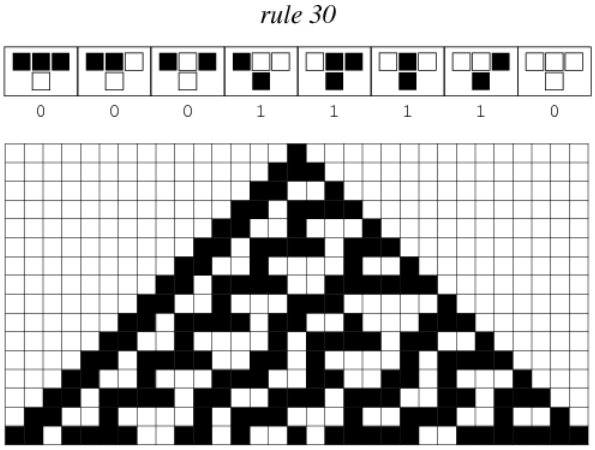
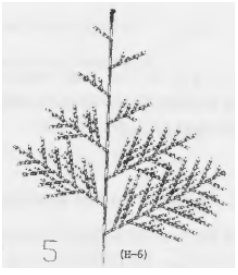


Figure 2.2: Cellular Automata Rule 30 [80]. Image used from [79] under CC BY 4.0.

(a) Observations H-4 of Japanese Cypress

(b) Image produced by an S0L-system similiar to observations H-4

(c) Observations H-6 of Japanese Cypress



(d) Image produced by an S0L-system similiar to observations H-6

Figure 2.3: Images from [53], reprinted with permission of Kyoto University.

(a) **(b)**

Figure 2.4: A real rose [6] used under CC-SA license (left), and three roses produced by a parametric 0L-system [62] (right, image reproduced with permission of the copyright holder).

2.2.1 Plant Modelling with L-systems

The earliest works on L-systems recognized the possibility that they could be useful for modelling cellular processes [41] and plant growth [42], although it would take a few years for the earliest plant models to emerge [25, 31]. These earliest models included a visualization component called CELIA [25] showing that the idea of visual simulation is an underlying conceptual motivation for L-systems. Since then, visualization software has grown more advanced. One leading tool is called the *virtual laboratory* (vlab) [39], which allows for L-systems to be described using two possible frameworks: cpfg [61] and lpfg [39], the latter provides a mechanism for L-systems to be enhanced by modules programmed in C.

For the most part, to visualize plant models, the words produced by the L-system are interpreted using the turtle graphics interpretation with some customization for leaves, petals, etc. Much of the earliest work in plant modelling used the basic 2D interpretation with the branching extension; however, more recently the extended 3D turtle graphics alphabet is used [62]. The resulting imagery produced can be remarkably realistic as seen in Figures 2.4a and 2.4b, which are are images of real and simulated roses. Such realistic imagery is made possible by understanding biological processes that govern plant growth and representing them in an L-system. There are many biological processes that have been successfully modelled such as auxin transport [15, 56, 62], flowering [49, 55, 62], phyllotaxis (how leaves appear on stems) [30, 55, 62], cellular layers [14, 62], and the phenotypic responses to environment [78]. As an example, approaches for modelling the flowering process in plants will be described next.

To model flowering plants several potential approaches were proposed [62] although they all share the idea of an apex node that produces growth until it flowers. This apex is represented by a rewriting rule such as $A \rightarrow FA$ (some variation occurs here with the possibility of some branching, e.g. $A \rightarrow F[B]A$). In this manner, starting from an axiom of A, an ever longer stem is created by the series of F symbols, which are interpreted by the turtle. The problem to overcome is how to stop the apex from creating growth and

change into a flower. One approach is to use a stochastic L-system where the apex node (the A symbol above) is transformed into a symbol representing the flower with some probably p. Although this works, the drawback is some stems will end up being much too short or long compared to their natural counterparts. Since "it is known that some plant species produce a fixed number of leaves before they start flowering" [62], an alternative is to use a parametric L-system to count the number of components that exist in the flowering plant. In this way, the plant will produce flowers similar to its natural counterpart; however, this approach only solves the issue when there exists such a component counting rule in the real plant.

The more comprehensive approach [62] is to model the biological signalling that occurs in plants by using parametric L-systems. In this approach, a signal is a parameter that is passed from symbol to symbol using L-systems. The conditions attached to the rewriting rules may be written to allow for very complex timing to occur throughout the model. This highlights that higher quality models can be produced using more complex types of L-systems as discussed in the previous chapter.

2.2.2 L-systems Applications in Other Domains

As mentioned previously, L-systems have grown from their roots in modelling biological processes to many research areas. Some of these areas, such as modelling arterial branching [26, 82], are very similar to plant modelling since their goal is to find the underlying mechanism of a biological process with a branching model. However, some applications, such as modelling buildings [50] and music [81], are very different and these will now be examined more closely to provide a contrast to plant modelling. L-systems as a model for cellular automata is also discussed since it is differs by being focused on the underlying processing of the automata and not for the objective of visualization.

The technique adopted by Müller et al. [50] uses parametric S0L-system in conjunction with creating their own alphabet to more closely represent building components. First, they define some basic building pieces, such as a cube, cylinder, and L, H, U, and T shaped buildings (the exact alphabet is not specified so for this explanation the symbols used will be $\{A, B, L, H, U, T\}$). These building blocks will be assembled, i.e. snapped together, to produce a building. So, for example, the word LB would place an L-shaped block and then snap a cylinder to the end of an L-shaped building. The alphabet is further enhanced with symbols (largely in the form of a function-like words, e.g. floor) that alters a block. So, for example, word $L\text{floor}B$ would place the L-shaped block but then add a floor prior to snapping on the cylindrical piece. Additional symbols in the alphabet allow for roof selection and texturing. All of these altering symbols, such as floor, take parameters that further define the visualization. So for example, floor takes parameters that define the number and placement of windows or entrances. The description of a single building is a subword of the word that describes an entire cityscape. They found that most types of L-system excessively favour a biological type of growth that is not observed in real cities. Therefore using a stochastic context-free JL-system[1], and

[1] A JL-system is one where some symbols are produced as a collection, and only one symbol from the collection is used as a predecessor in the next step. For example, $A \Rightarrow B\{C, D\}$, and then B plus either C or D are used in the next step.

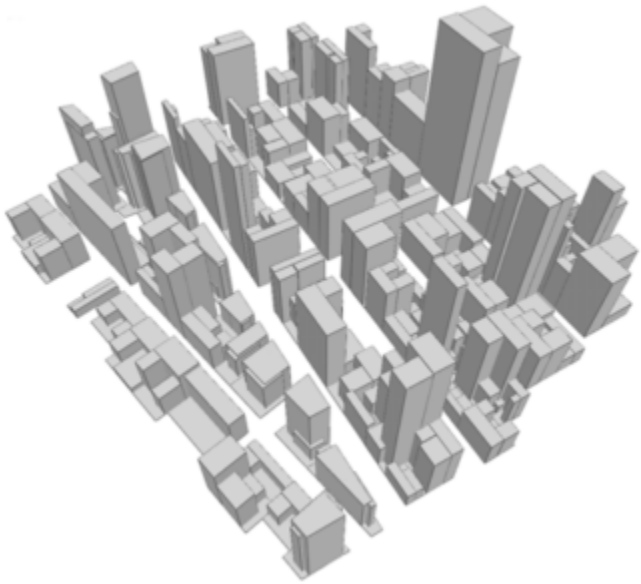


Figure 2.5: A modern city produced with a stochastic context-free JL-system [50]. Image used with permission of the authors.

starting from an axiom of a few buildings, they could produce entire cityscapes if each split in the production was allowed to grow separately. Figure 2.5 shows such a cityscape although for that particular simulation they did not use any of the texturing symbols.

In the work by Worth and Stepney [81], they investigated finding or creating music that was subjectively pleasing to the authors by using L-systems. Their approach uses the branching turtle graphics alphabet; however, they render the result musically. In their work, they propose two different renderings. In the first rendering, they interpret an F as an increase to the duration of the current note by a quarter note, a $+/-$ as ending the current note and changing the pitch of the next note, f is not used; however, they introduce an X symbol which is taken as null (do nothing). Branching symbols are interpreted as putting the definition of the current note on hold, defining a new note to be played by the contents of the branch, and then returning to the definition of the current note. So, word $FF[F+F]F$ would be interpreted as follows. A note definition would start with the first two Fs producing a half note but this is not played yet as the definition is not complete. The current note definition is put on hold and the contents of the branch are played, a quarter note followed a quarter note at the next higher pitch. Then the half note that was in progress is changed to a 3/4 note and played. For the second rendering, they work in reverse. They start with a few very long notes and interpreting F as a command to divide the long note in two. The symbols $+/-$ and the branching symbols are unchanged. Additionally they add a d symbol to represent halving the duration of a note. They evaluated S0L-systems and deterministic context-sensitive L-systems using the renderings and found that in either case the music sounded random, simple, but well-structured. Although not all of the music was pleasant, there were some pieces produced that sounded good to the authors.

Cellular automata are a model for parallel computation [80]. They consist of an array of boolean cells.

At each time step, the values in each cell are set to 0 or 1 based on the values of the cell and its neighbours at the current time. Every cell's new value is computed simultaneously. Based on this description, it can be seen that cellular automata can be modelled using a context-sensitive system, with Wolfram using deterministic context-sensitive L-systems to produce his famous cellular automata rule set [80]. With L-systems, productions are computed in a series of time steps and new values are produced in parallel, which are two of the requirements above for producing a cellular automaton. Context sensitivity allows for a cell's value to be produced through considering its value and its neighbour's values, the final requirement for cellular automata.

2.3 L-System Inference

2.3.1 Existing L-system Inference Approaches

Inferring L-systems is not a new problem, and over the decades since the discovery of L-systems, several different approaches have been explored to solve the problem from different perspectives. Although each of the different approaches have been successful in some way or another, each has a limitation or focuses on a specific inference sub-problem. This section will discuss some of the approaches found in the literature by describing how they work and what limitations they might have. A survey [7] describes many of these approaches.

Tools as an Aide to Experts

The oldest [25,62], and still common technique for inferring L-systems is to have human experts build them by hand. This approach has been successfully used extensively in multiple research areas [16,50,53,55,62–65,82]. This approaches uses human expertise to understand some of the underlying mechanisms and translates this knowledge into an initial L-system. The L-system is then modified iteratively until the result closely matches the desired output. The main drawback is the time and effort required to produce the L-system. Additionally, while it cannot be denied that building L-systems has been successful, it is possible that some problems may be too complex to be easily solved by hand either because the required L-system is extremely complex or the underlying mechanisms may not be well understood. An automated approach may, by inferring an L-system automatically for complex models, help to reveal the underlying natural phenomenon taking place, at least with respect to the high level mechanisms. For example, L-systems do not reveal the underlying biological mechanism that is taking place that causes a plant to produce a branch, but rather that such a mechanism exists in the first place (and with parametric L-systems what factors might be controlling the mechanism). The effort required to produce models by hand can be somewhat alleviated by having an automated approach to assess the output of the model, examples of which follows.

Curry [18] investigated using a genetic algorithm to find the parameter settings for parametric 0L-systems that describe plant models. In this approach, the alphabet, rewriting rules are assumed to be known.

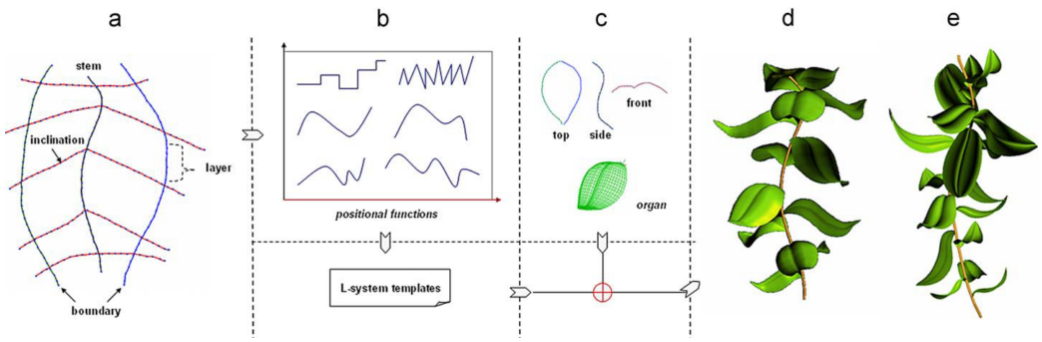


Figure 2.6: Converting a sketch into a parametric D0L-system [3]. Image reproduced with permission of the copyright holder.

Furthermore, the assignment of parameters to symbols and form of the conditions are assumed known. For example, they might have a rule $A(y) : y < p_1 \rightarrow A(2 \times y)B$, and then use the genetic algorithm to find the optimal setting for p_1. They use a genetic algorithm with single-point and multi-point crossover operators with a multi-parent mutation operator. The crossover operators are typical to most GA applications. Their mutation operator is atypical and selects two or more parent genomes, then for each gene the mean and standard deviation is computed across the parents. A random value is selected from the Gaussian distribution described by the mean and standard deviation and assigned to the child. The fitness function "is determined by the user according to aesthetic criteria" [18], so a human expert is required to monitor and assess the output. Additionally, the user selects which genetic operator to apply next. Although such a tool can be helpful, it does not alleviate much of the drawbacks to building a process by hand relative to a fully automated approach. And it certainly does not scale well to learning the model for a large population of plants.

More recently, Anastacio et al. [3] investigated using sketching as an aide to human crafted L-system design. In their approach, the operator draws a stem, plant organs at the desired inclination, and outer boundaries for the plant. The sketch is converted into functions to control the phyllotaxis of the organs. These functions are then used as the conditions and parameters to change a D0L-system into a parametric L-system that describes a desired plant. Finally, the operator draws an organ and chooses from three phyllotaxis templates (distichous, decussate, and Fibonacci spiral) and the visualization is produced. Some parts of this process are shown in Figure 2.6, namely the drawing, conversion to functions, organ drawing, and the final visualization. The authors capitalize on the understanding that plants display different types of phyllotaxis templates that can be selected by the operator; thereby, showing the importance of *a priori* biological knowledge to the human crafted approach. The authors observe that an approximation of a plant can be drawn rather quickly, although some time is required to then fine tune the L-system.

Inductive Inference

Doucet [23] examined the inductive inference problem (informally described in Chapter 1, and formally defined in Section 2.3.2) for D0L-systems from an algebraic perspective. He roughly describes two algorithms

neither of which were implemented, one for inductive inference using consecutive words and another for scattered words (i.e. some words are missing from the developmental sequence). His algorithms assume that the size of the alphabet is known, and the rank order of the words is known. His algorithms convert the words into Parikh vectors, which is simply the count of the number of each symbol in the word expressed as a vector. For example, the word $ABCA$ in Parikh vector form is $(2, 1, 1)$. We will describe the consecutive case as follows: each of the first k known words are converted into Parikh vectors and placed into a square $k \times k$ matrix where k is the number of letters in the alphabet. A second matrix is formed using the Parikh vectors of the first k words after the first word (both shown in Table 2.4). This defines a matrix equation, as in Table 2.4, where the variables in the matrix X are unknowns. In this case, if there is an L-system that gives this developmental sequence, then substituting the number of the j^{th} letter produced by the production from the i^{th} letter in for position (i, j) of X is a solution to this matrix equation. Therefore, if one examines all possible solutions to X, one of them must describe a correct solution. This gives a matrix problem in the form of $SX = T$, where S is the first matrix and T is the second matrix, which has a solution of $X = S^{-1}T$ if S is invertible. If S is not invertible, then X does not have a unique solution, but rather defines a set of linear Diophantine equations who solutions represent the relationships between the successors for each $A \in V$ and the number of each symbol in V in the successor. Mathematically, these equations have an infinite number of solutions but for the L-system inference problem, the values for the variables are bounded to the natural numbers since growth must be integral and non-negative, and also bounded by the lengths of the words in the developmental sequence. Being bounded does not mean that it is easy to find solutions to the equations as the number of combinations can be extremely large, but nonetheless, the equations define a bounded search space that can be searched. Doucet does not address how to convert the Parikh vectors into actual successors; however, a method is provided in Chapter 3, and was first published in [45] that converts the vectors for each $A \in V$ into the length of the successor of A.

In the scattered case, the missing words are inferred by considering the possible recurrence relationships described by the observed words. This is done by assigning the Parikh vectors of the observed words to variables and examining the polynomial equation that describes the recurrence relationship. For example, given the Parikh vectors for the first, third, fifth and ninth words as $w_1 = (0, 0, 0, 1), w_2 = (0, 1, 1, 1), w_5 = (1, 1, 2, 1)$, and $w_9 = (3, 2, 5, 4)$ then the recurrence relationship is $w_9 - 3w_5 + w_3 - 2w_1 = 0$ with a corresponding polynomial equation of $\psi = x^9 - 3x^5 + x^3 - 2x$. Reducing the polynomial to factors defines possible words. For example, one factor is $x^3 - x - 1$ which corresponds to $w_3 = w_1 + w_0$. Iterating with this relationship produces a candidate set of words that can then be used to determine the productions in a process similar to the one described above. The solutions to the matrices are not necessarily unique and different factors need to be tried exhaustively, thus making the effective solution space even larger. Doucet's approach though can determine if a scattered sequence is produced by a D0L-system, as the polynomial must have only integer coefficients after being reduced by the greatest common denominator for D0L-systems. We do not consider the scattered version of this problem in this thesis.

Developmental Sequence
A
$ABCA$
$ABCABBCABCA$
$ABCABBCABCABBBBCABCABBCABCA$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \\ 4 & 4 & 3 \end{bmatrix} \cdot X = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 4 & 3 \\ 8 & 12 & 7 \end{bmatrix}$$

Table 2.4: Words produced by a D0L-system and the corresponding matrices.

LGIN [52] uses number theory and logic to deduce a D0L-system and is probably the leading implemented approach to inferring L-systems. The problem investigated differs slightly in that it only takes one string of the developmental sequence as input. The method used is similar, but not identical, to Doucet [23], LGIN uses matrix operations to create equations describing the relationships of relationships between a symbol and the number of each symbol that appears in its successor. LGIN then exhaustively tries to find the appropriate successor(s) defined by the solutions that can generate a single input string. Since LGIN only uses a single string, it does not guarantee to find a unique solution. LGIN was evaluated on six "Fractal Plant" variants of D0L-systems, and it was found to be able to infer all of them in, at most, a few seconds. Three of these variants have one letter alphabets and three have two letter alphabets. In their paper, they call the case for alphabets with more than two symbols to be "immensely complicated" [52], and they did not evaluate LGIN on any larger alphabets.

Štava et al. [73] investigated an inductive inference approach similar to that used by Nishida to infer a model of Japanese Cypress trees. In effect, they combine segmentation of the image and rule generation; i.e., they look for shapes that appear frequently within the image and assign the shape to a rule. Using computer vision techniques, their algorithm is able to identify Bézier curves and produce an expression that would produce each curve. They call these curves the "elements" of the image. They observe that by adding new production steps, a production may be subdivided. The example they provide uses the production $A \rightarrow BCD$, which can then be subdivided into $A \rightarrow XY$, followed by $X \rightarrow BC$ and $Y \rightarrow D$ to effectively cause A to produce BCD over two derivation steps. They use this technique to divide the rules into the graphical element and the branching mechanism. Parameters are used to define the values required to produce the proper Bézier curve. As a final step, some minor manual modification is made to the produced L-system. They found that they are able to reproduce 2D vector images that contain repeating patterns.

Logical Approaches

Nakano [51] proposes an approach called LGIC2, where the words created by an unknown L-system are examined logically and an L-system is inferred that can produce, or come as close as possible to the observed words. LGIC2 focuses on inferring L-systems when the words are noisy, citing De La Higuera [22] that this is realistic for real unknown models. The algorithm works by scanning the words for commonly reoccurring collections of symbols, called candidates, and assumes that the successors for the symbols must by one of these candidates. The candidates are pruned based on three criteria. First, candidates are discarded if they occur below a threshold frequency with a threshold determined experimentally for a particular model. Second, candidates are rejected if they produce a statistically significantly number of additional symbols than in the original words. Third, for the fitness function they use the value from a least common subsequence (LCS) computation. However, since computing the LCS is computationally expensive, they first estimate the upper bound of the LCS calculation. They reject a candidate if the computed upper bound is less than the current best fitness value. Using this algorithm they are able to infer D0L-systems for some tree-like systems with errors. The main drawback to LGIC2 is it was only evaluated on a simple fractal-like trees with a two letter alphabet and short successors. On such systems, it found the appropriate L-system in 5 to 30 minutes; however, for even slightly more complex systems this time increased considerably as the number of candidates increase, and they rely on a brute force search. Additionally, LGIC2 is highly dependent on setting the frequency threshold properly, and although this can be done experimentally for a known model by trying different values, no insight is offered on how to algorithmically set this value for an unknown model. If this value is set and no L-system is returned, there is no way of knowing whether there is no such L-system or it failed to find one.

Evolving L-Systems

One of the most common approaches in the literature, other than building L-systems by hand, is to evolve L-systems using evolutionary operators, especially using genetic algorithm. This approach appears to be particularly popular for biological models (e.g., [35,37]). There is an appealing logic, although unproven, that because real plants and animals were created by evolutionary forces, then an underlying L-system describing how they are created could be evolved as well. This subsection will describe the different approaches towards evolving L-systems. Since genetic algorithm plays such a significant role in understanding these approaches, first the genetic algorithm will be briefly described followed by the discussion on some of the approaches to evolving L-systems.

One simple variant of the genetic algorithm is described as follows from Bäck [5] as an optimization algorithm, based on evolutionary principles, used to efficiently search N-dimensional (usually) bounded spaces. In evolutionary biology, increasingly fit offspring are created over successive generations by intermixing the genes of parents. Similarly, in the genetic algorithm, an encoding scheme is applied to convert a problem into a virtual genome consisting of N genes. Each gene is either a binary, integer, or real value which

represents an element of the solution in a problem-specific way. For example, in the travelling salesman problem one common encoding is to have each gene represent the ordered choices of what city the salesman should visit next, and each gene has a possible integer value from 1 to C, where C is the number of cities. The genetic algorithm functions by first creating an initial population P of random solutions. Each member of the population is assessed using a problem specific fitness function to assign a fitness value to the solution. Then, the genetic algorithm performs a selection, crossover, mutation, and survival step until a termination condition is reached. The termination condition may be based on such things as finding a solution with sufficient fitness, a pre-determined maximum number of generations, or a maximum period of time. In the selection step, a set of S (usually $P/2$) pairs of genomes are selected from the population with odds in proportion to their fitness, i.e. preferring more fit genomes. A genome may be selected for multiple pairings. During the crossover step, for each pair, a random selection of genes are copied between the two; thereby, producing two offspring. For each offspring, zero or more genes are randomly selected to be changed to a random value. Then each mutated offspring is assigned a fitness value from the fitness function. The offspring are placed into the population and genomes are culled until the population is of size P again. Usually, the most fit members are kept (elite survival) but other survival operators exist that can prefer to preserve some genetic diversity.

One of the earliest investigations into the evolving approach was by Jacob [37] who used a genetic algorithm to find D0L-systems that have the aesthetic of flowers. To do this, a template is created to give the rough structure desired. Within the template are variables with values selected by the genetic algorithm, i.e. each variable is represented by a gene. The variables represent either adding structural parts to the system (e.g. stalk, leaf, bloom) or graphical commands (e.g., turn left x degrees, turn up y degrees, etc.). The genetic algorithm had crossover and mutation operators that work in a typical fashion for genetic algorithms. The genetic algorithm was extended from the simple version with three additional operators: deletion, duplication, and permutation. The deletion operator allowed for the partial or total removal of rewriting rules. Duplication could copy an entire rule, or copy part of the successor for a single rule. Permutation rearranges the components within a pair of production rules. For example, if a solution has $A \rightarrow x$ and $B \rightarrow y$, then permute could swap the A and B. The use of templates helps to ensure that the result from applying any operator results is a valid rule and so no repair mechanism is required. Unlike with Curry's algorithm [18] that required the user to assess the aesthetics of the result, this approach uses a fitness function based on how much the plant grows in three dimensions, and how many bloom and leaves exists on the model for each generation. The main limitations of this approach are that it does not find an L-system for anything specific due to the use of an aesthetic fitness function; however, it does produce L-systems that look like non-species specific flowering plants. Furthermore, it cannot be easily extended to solve problems in general due to the use of templates. A template would be required to describe the unknown L-system but such *a priori* knowledge may not always be available.

Wildwood is an algorithm proposed by Mock [47] to evolve L-systems for plant models using genetic

algorithm. The main contribution of Wildwood is, unlike many other algorithms, it can be set to assess the fitness of an L-system based on the survivability of the plant. Wildwood assumes a fixed alphabet of the 2D turtle graphic alphabet with branches and an A ($\{A,F,f,+,-,[,]\}$). The genomes for the genetic algorithm consist of an ordered sequence of symbols. Wildwood uses a two-point crossover operator with a self-contained repair mechanism for branching structures, i.e. only valid crossover operations are permitted. This is accomplished by implementing two rules. First, if the selected subword has more [symbols than] symbols, the end point is extended until a balancing number of] symbols are found. The second rule states that if a selected subword has more] symbols than [symbols, then the endpoint is rolled backwards until they balance. A different start and end point is selected for each parent during crossover allowing the resulting child strings to shrink or grow. For mutation, a valid subword is selected using the same rules as for crossover and then changed into a random string of the same length. Wildwood possesses two fitness functions. The first fitness function focuses on aesthetics as determined by the user, i.e. the user decides which members of the population they like and which should be culled. The second fitness function is a preliminary attempt to find realistic plants based on the traits exhibited by the simulation. Equation 2.1 shows the formula for the fitness where W is the width of the plant, H is the height of the plant, and S is system parameter. The authors state that S was set to 30 for their experiments further noting that a higher S value rewards plants for being short. The main limitation of this approach is the use of a single non-terminal producing symbol.

$$Fitness = W + S/H. \tag{2.1}$$

Hornby and Pollack [35] use an evolutionary approach for finding parametric L-systems to model virtual creatures. In their approach, an initial population of random systems are created. Then over a series of generations, pairs of systems are selected with high fitness (the exact method is not specified but a roulette wheel or tournament selection method is implied), and a crossover operator is applied that swaps system components between them. Additionally, systems from the population are selected to be mutated with a system element changed at random. The new population members are then assessed by using a fitness function that simulates the creature in the real world and determines how far the centre of mass can move from its original position by rolling, flipping, etc. Creatures that move further are considered more fit and any creature that falls below a fitness threshold is discarded. Ashlock et al. [4] use a very similar evolutionary approach to produce plant models. The difference is their fitness function uses a bounded 2D area and looks for models which fill the shape without going outside the bounds. The requirement to have a different fitness function highlights the two issues with such approaches. First, the fitness function is specialized to the desired problem and cannot be carried over to another problem, i.e. the fitness function for generating creatures will not work for generating plant models. Second, it can be difficult to find a meaningful fitness function that do not have too much complexity. The fitness function must be executed for every candidate solution and so as the computational cost of the fitness function increases so does the cost of the overall algorithm. It would be preferential to have a standard fitness function that could cross over to different problems and would have a

low computational cost.

An approach for generating S0L-system using a combination of an evolutionary operator (mutation) and support vector machines (SVM) was proposed by Damaševičius [19] with the intended purpose of modelling DNA sequences. As a first step, a training data set is used to train an SVM to classify DNA into one of three taxa *Drosophila*, vertebrate, or monocot plants. Once trained, an S0L-system is created randomly for each taxa assuming a four letter alphabet of nucleotides, $\{A, C, T, G\}$ where each rewriting rule has a single successor and probability combination. For example, a rule may look like $A \rightarrow TATA$ with a probability of 0.85, meaning there is an 85% chance an A will produce a $TATA$, and a 15% chance it will produce an A. The following iterative process is used to refine the L-systems. String sequences were produced by the L-systems are then classified by the trained SVM. If the current L-system ruleset resulted in the most strings being successfully classified, then it was considered the best configuration, if not then the current ruleset was discarded. The best configuration of the L-systems was mutated using a directed random search based on the accuracy. The drawback to this approach is the requirement of some positive and negative examples to train the SVM, and this may not be possible in the case of trying to find an L-system for an unknown model. Even if such data is available, the ability to find a suitable system is only as good as the classifier. Although, it is notable that they had 93.40% accuracy for *Drosophila*, 92.10% accuracy for veterbrate, and 96.1% for monocot plants indicating it is a promising approach for this particular problem. The S0L-systems generated are limited to single successor and probability combination.

The last approach examined is by Runqiang et al. [71], in which they propose to infer a D0L-system from an existing model using a GA. Their approaches differ from other approaches in that they do not focus on aesthetics or plant traits but rather they try to make their L-system produce the same imagery as the known model. This technique allows it to be transferred from problem to problem as it is agnostic to the type of model, so long as there is a visualization as the use image processing techniques to match the source and produced imagery, i.e., it can be used equally well on trees, flowers, herbaceous plants, bushes, etc. In their approach, they encode the genome as a string of symbols. A two-point crossover operator is used without an embedded repair mechanism, i.e. invalid symbols strings may result. They use a uniform mutation operator with a repair mechanism for branching symbols where if the new symbol selected is a branching symbol, then another symbol is selected to be an opening/closing branching symbol to balance it. Additionally, the mutation operator changes the symbol order by selecting a subword of symbols and swaps them with another randomly selected subword. However, since illegal strings may still result, they use an unspecified repair mechanism to correct any invalid strings prior to assessment. The fitness function used to assess the candidate solutions computes the distance between the vertices along the trunk and branches. The main limitation of their approach is the limited alphabet size as they assume a maximum of 2 symbols (not including graphical symbols such +,-) and a total length of all successors combined of 14. Their approach failed 17 out of 50 times to find the L-system for "variant D fractal plant" [62]. The authors believe that the difficulty is two-fold: 1) searching for an L-system with "two variables makes the search space wider" [71],

2) and the total length of the successors at 14 make the "search less efficient." [71]

2.3.2 Inductive Inference and Computational Complexity

Many different approaches have been investigated to solve the problem of inferring L-systems. Other than producing systems by hand, the three approaches discussed in this review use either algebra, logical rules, or evolutionary operators. The algebraic approach to inductive inference discussed previously is limited by uncertainty with respect to how often it will produce Diophantine equations that describe large search spaces. The logical and evolutionary approaches existing in the literature can only infer L-systems that have small alphabets, which suggests that L-system inference results in might result in large search spaces. Nakano [52] calls the problem of alphabets larger than two symbols "immensely complicated", and Runqiang et al. [71] state that the search space was large enough to cause the algorithm to fail 17 out of 50 times. Hence, a first point of reflection is to consider the computational complexity for L-system inductive inference algorithms. Is it even possible to infer a deterministic L-system in a reasonable way?

During the research around this thesis, an L-system inductive inference algorithm was developed that functions as follows as described in [45]. Let ρ be a sequence of strings $(w_1, \ldots, w_n)$ produced by an unknown (hidden) D0L-system $G = (V, w, P)$; i.e., $w = w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$. Further, let $w_i = a_1 a_2 \cdots a_x$ and $w_{i+1} = b_1 b_2 \cdots b_y$, where each a_j and b_j is a letter. When the rewriting rules are applied to a word, all symbols are replaced in parallel; however, their order is not changed. Thus, if a derivation is applied to w_i, then the resulting word is $w_{i+1} = succ(a_1)succ(a_2)\cdots succ(a_x)$. Therefore, it follows that $succ(a_1)$ is the symbols b_1 to $b_{|succ(a_1)|}$, and $succ(a_2)$ is the symbols $b_{|succ(a_1)|+1}$ to $b_{|succ(a_1)|+|succ(a_2)|}$, and so on. As a result, if the successor lengths were known for each letter, then their successors can be easily determined. The successor lengths are, of course, not known; however, it is possible to search for them (a comparison of this encoding scheme to those used by existing approaches is provided in Chapter 3; however, in summary this represents an improvement in at least the number of dimensions (N) required). We refer to this as the scanning process. Let q be the longest word in ρ, and let S_ρ be the sum of the lengths of the developmental sequence. The algorithm can infer a D0L-system generating ρ with time complexity of $O(q^{|V|} \cdot S_\rho)$. If context-sensitivity is included, then the algorithm above is modified to include checking the context around a symbol with an upper bound of k symbols to the left and l symbols to the right using a sliding window. This modification means that every combination of context-lengths from zero to k (the maximum length of the left context), and zero to l (right context) must be examined meaning a generalized L-system inference algorithm can be developed with a worst case complexity of $O(q^{(|V|+1)^{k+l+1}} \cdot S_\rho)$. Although this is exponential; if the alphabet is of fixed size, then since k and l are always either 0 or 1 in the literature, this function is polynomial. So, if both the alphabet and context sizes can be reasonably taken as being of fixed size, the algorithm is of polynomial time. Thus, for both deterministic context-free and context-sensitive L-systems, the problem is quite practical with the scanning process if the alphabets are not too large.

The existence of a polynomial time algorithm gives some hope to being able to practically search for a

deterministic L-system given a sequence of strings as an input. The next chapter will present a practical deterministic L-system inference algorithm, and the following chapters will expand on this algorithm to infer stochastic L-systems, and then infer parametric L-systems.

Chapter 3

Techniques for Inferring Context-Free Lindenmayer Systems With Genetic Algorithm

Abstract

Lindenmayer systems (L-systems) are a formal grammar system, where the most notable feature is a set of rewriting rules that are used to replace every symbol in a string in parallel; by repeating this process, a sequence of strings is produced. Some symbols in the strings may be interpreted as instructions for simulation software. Thus, the sequence can be used to model the steps of a process. Currently, creating an L-system for a specific process is done by hand by experts through much effort. The inductive inference problem attempts to infer an L-system from such a sequence of strings generated by an unknown system; this can be thought of as an intermediate step to inferring from a sequence of images. This paper evaluates and analyzes different genetic algorithm encoding schemes and mathematical properties for the L-system inductive inference problem. A new tool, the Plant Model Inference Tool for Context-Free L-systems (PMIT-D0L) is implemented based on these techniques. PMIT-D0L has been successfully evaluated on 28 *known L-systems, with alphabets up to* 31 *symbols and a total sum of* 281 *symbols across the rewriting rules. PMIT-D0L can infer even the largest of these L-systems in less than a few seconds.*

3.1 Introduction

Lindenmayer systems (L-systems), introduced in [41], are a bio-inspired grammar system that produces self-similar patterns that appear frequently in nature and especially in plants [62]. L-systems produce a sequence of strings, where a word is obtained by the parallel application of rewriting rules to the previous word. Certain symbols can be interpreted as instructions to create images, and therefore a sequence of strings can describe a temporal process, which can be visually simulated by software such as the "virtual laboratory" (vlab) [77]. Such simulations can incorporate different geometries [27, 53, 62], environmental factors [2, 78], and mechanistic controls [53, 56], and as such are useful for simulating plants. L-systems often consist of small textual descriptions that require little storage compared to real imagery. Certainly also, they have a low cost in currency, time, and labor to simulate *in silico* compared to actually growing a plant, and realistic imagery can be produced with a well-constructed L-system.

(a) **(b)**

Figure 3.1: Fibonacci Bush after 7 generations (left), and apple twig with blossoms (right) as produced using vlab [62]. Images reproduced with permission of the copyright holder.

Formally, L-systems are described by an ordered tuple $G = (V, \omega, P)$ consisting of an alphabet V (a finite set of allowed symbols), an axiom ω that is a word over V, and a finite set of productions, or rewriting rules, P. A deterministic context-free L-system (D0L-system) has rules of the form $A \to x$, where $A \in V$ is called the predecessor, x is a word over V that is called the successor of A, with exactly one rule for each $A \in V$ as predecessor. Given a word $\omega_i = A_1 \cdots A_m$ where each $A_i \in V, 1 \leq i \leq m$, a derivation step $\Rightarrow$ is defined by $A_1 \cdots A_m \Rightarrow x_1 \cdots x_m$ where $A_i \to x_i$ is in P, for each i, $1 \leq i \leq m$. When this process is repeated $n-1$ times starting with the axiom ($\omega = \omega_1 \Rightarrow \omega_2 \Rightarrow \cdots \Rightarrow \omega_n$), the sequence $(\omega_1, \ldots, \omega_n)$ is called the length-n developmental sequence. By treating certain symbols as instructions for positioning and drawing in a 3D space (described in Section 3.2) temporal processes can be simulated using simulation software. Figures 3.1a and 3.1b show some structures built with D0L-systems in this fashion.

A difficult challenge is to determine an L-system that can accurately simulate a specific process, for example, modeling plant growth. In practice, this often involves manual measurements over time, scientific knowledge, and is done by hand by experts [27,53,62,64]. Although this approach has been successful, it does have notable drawbacks. Producing a system manually requires an expert, who are in limited supply, and it does not scale to producing arbitrarily many (perhaps closely related) models. Indeed, the manual process currently has been described as requiring "tedious and intricate handwork" [26] that could be improved if an automatic approach could "infer rules and parameters automatically from real ... images" [26]. Furthermore, when constructed manually, the more complex plant models require *a priori* knowledge of the underlying mechanics of the plant. In contrast, inferring L-systems automatically may be used to reveal scientific principles of the underlying process, or as stated by Godin and Ferraro, automatic inference "could be further exploited in combination with investigations at a biomolecular level to better understand plant development." [27]

The ultimate goal of this line of research is to automatically determine an L-system from a sequence of plant images over time. There have been simplified variants of this problem that have been attempted thus far in the literature. One approach is to develop a tool as an aide for the expert to reduce the work

load [36, 47]. With such approaches, the expert operator guides the tool towards a desirable model. Another approach is to build a fully automated method to find an L-system that produces the length-n developmental sequence given as an input [52, 71]. This is known as the inductive inference problem and it dates back to early work on L-systems studied from the perspective of decidability [32]. This can be thought of as an intermediate step of inferring an L-system from images, with an accurate segmentation of the images into descriptive strings being the second step. A similar type of plant image segmentation used on a temporal image sequence has been studied separately, e.g. [21]. However, for inductive inference to be truly crucial in combination with image segmentation, it would need to be both fast and accurate so that it could be expanded to work with realistic complications such as noisy images, imperfect data, and other mechanisms built into L-systems. Existing work on inference and inductive inference of L-systems is described in Section 3.2.3.

This paper creates the Plant Model Inference Tool for Deterministic Context-Free L-systems (PMIT-D0L) [9, 11] that aims to be an automated approach to solve the inductive inference problem for D0L-systems. Towards that goal, PMIT-D0L uses a genetic algorithm (GA) to search for an L-system that produces a sequence of strings provided as input. In general, GAs search solution spaces in accordance with the encoding scheme used for the problem, and to-date most existing approaches to L-system inference use similar encoding schemes. This paper presents and analyzes different encoding schemes, both new and old, to show which are most effective for inferring L-systems. Additionally, some mathematical properties are used to shrink the solution space.

Between the encoding schemes and the use of mathematical properties based on necessary conditions, PMIT-D0L is able to infer all L-systems in a test suite where the sum of production successor lengths is up to 281 symbols; whereas, other approaches implemented in the literature are limited to about 20 symbols as described in Section 3.2.3. Moreover, the test suite used to test PMIT-D0L is significantly larger than previous approaches, consisting of 28 previously developed D0L-systems. The best encoding scheme, based on the length of successors, took no longer than 3.192 seconds for each L-system, with an average of 0.391 seconds for them all. All L-systems were inferred with 100% accuracy with all encoding schemes. This is notable as the GA is being used to essentially learn the simulations from data. Some encoding schemes used were slower than others; however, the mathematical properties played a larger role in the speed. When multiple properties were combined together, they operated in a synergistic fashion to reduce the search space.

There are many future directions required in order to fully realize automatic inference of L-systems. Although many modern L-systems produced by experts use additional features, especially rules that have parameters, creating an inductive inference procedure for D0L-systems that is both fast and accurate is a big step forward. First, it shows that the concept has promise and its study is worth pursuing. Secondly, many uses of parameters in L-system rules behave like context-free L-systems during certain sections of their derivations (e.g. if the parameters are being used to incorporate a timing mechanism [59, 62]). The techniques developed in PMIT-D0L can also be used for these sections, and can also be used to detect deviations

corresponding to a change in the program via parameter. Therefore, studying D0L-system inference scientifically, and inferring D0L-systems in a fashion which can be extended into rules with parameters, is an important step towards the main long term objective. Indeed, PMIT-D0L provides both a fast and accurate implementation of inductive inference that is necessary for L-system inference from images, and is the first inductive inference implementation to do so.

The remainder of this paper is structured as follows. Section 3.2 will describe some existing automated approaches for inferring L-systems. Section 3.3 will discuss the different encoding schemes that can be used with PMIT-D0L, along with techniques for reducing the search space size, etc. Section 3.4 discusses the data set, performance metrics, and the results of the evaluation of PMIT-D0L. Finally, Section 3.5 concludes the work and discusses future directions.

3.2 Background

This section describes useful contextual and background information relevant to understanding this paper. It starts with describing some notation used. Since a GA is used as the search mechanism for this work, it contains a brief description of them. The section concludes with a discussion of some existing approaches to L-system inference.

3.2.1 Notation

An alphabet is a finite set of symbols. Given an alphabet V, a word over V is any finite sequence of letters $A_1A_2\cdots A_n$, $A_i \in V, 1 \leq i \leq n$. The set of all words over V is denoted by V^*, which contains the empty string denoted by λ. Given a word $x \in V^*$, $|x|$ is the length of x, and $|x|_B$ is the number of B's in x, where $B \in V$. Given $V = \{B_1, \ldots, B_k\}$, the Parikh vector of a string $x \in V^*$ is $(|x|_{B_1}, \ldots, |x|_{B_k})$.

Given two words $x, y \in V^*$, then x is a substring of y if $y = uxv$, for some $u, v \in V^*$ and in this case y is said to be a superstring of x. Also, x is a prefix of y if $y = xv$ for some v, and x is a suffix of y if $y = ux$ for some u.

Given a D0L-system $G = (V, \omega, P)$ as defined in Section 1, the successor of A is indicated by $succ(A)$. Given a rewriting rule $A \rightarrow succ(A)$, and $B \in V$, then the number of symbols B in $succ(A)$ is called the growth of B by A, denoted by $M(A, B)$. These values are stored in a $|V| \times |V|$ matrix called the growth matrix $M(G)$. Commonly, V includes symbols to provide simple graphical instructions to simulation software (such as vlab [77]). One commonly used such instruction set is the "Turtle Graphics" [54]. It is imagined as manipulating a turtle through a 2D or 3D space with a pen on its back. The turtle has a state consisting of its position and orientation. The symbols "F" and "f" move the turtle forward along its current orientation with the pen on or off respectively. In 2D, the symbols "+" and "−" turn the turtle a predefined number of degrees left or right. In 3D, additional symbols are needed for pitch ("&" down and "^" up), and roll ("\" left and "/" right) [54, 62]. For branching processes, the symbols "[" and "]" are used to start and stop a

branch, which is implemented as pushing and popping the turtle's state on a stack. It is usually the case that the symbols "[", "]", "+", "−" have identity productions. There are some instances where "F" may not have an identity production (e.g. some of the variants of "Fractal Plant" [62]). Given a sequence of n words ρ, G is said to be compatible with ρ if ρ is G's length-n developmental sequence. To differentiate the turtle graphic symbols "+", "−" from the corresponding mathematical operators + and −, the turtle graphics symbols will appear in bold as **+** and **−**.

3.2.2 Background on Genetic Algorithm

The GA is described as follows by Bäck [5]. The GA is an optimization algorithm based on evolutionary principles used to efficiently search N-dimensional (usually) bounded spaces. An encoding scheme is applied to convert a problem's solution space into one describable by a virtual genome consisting of N genes. Each gene can be either a binary, integer, or real value and represents, in a problem specific way, an element of the solution to the problem. While there exists several types of value encoding schemes, a literal encoding directly represents an element of the solution to a problem. An example of a literal encoding scheme uses gene values to represent the length of a successor, so a value of 3 indicates a successor length of 3. In contrast, a mapped encoding could instead use a real value from 0 to 1 subdivided into sections that represent the different possible solutions.

In evolutionary biology, increasingly fit offspring can be created over successive generations by intermixing the genes of parents. Similarly, a GA functions by iterating over the selection, crossover, mutation, and survival operators until at least one termination condition (e.g. a time limit) is met [5]. There exist different types of these operators; however, this paper will describe only those used here. The function of the GA is controlled by the parameters: population size (P), crossover weight (C), and mutation weight (M). Prior to the first iteration, a GA first produces an initial population of P random solutions. The selection operator chooses some number of pairs of genomes from the population using a selection technique. One such technique, a roulette wheel, is one where the chance of any option being selected (in this case a genome) is proportional to an associated value (in this case, the genome's fitness). For each pair of genomes, the crossover operator swaps a random selection of genes between them, resulting in P child genomes. The chance for any gene to be swapped is equal to C. The mutation operator changes a random selection of genes to a random valid value in each child genome. The chance of any individual gene being mutated is equal to M. The child genomes are added to the population, and the population is culled to size P, thereby keeping the most fit genomes (elite survival).

With PMIT-D0L, the following changes are made to the standard GA to encourage additional exploration. Although an individual genome may be selected for more than one pair, the same pair may not be selected more than once. If any genome has been modified by neither the crossover operator nor the mutation operator, then one gene is selected and mutated to ensure that at least one change has taken place. Where a mapped encoding is used, it is possible for two different genomes to map to the same solution. To prevent

such solutions from dominating the population, genomes that map to the same solution are automatically culled (similarly during initialization, duplicated solutions are not permitted in the population).

3.2.3 Existing Automated Approaches to L-system Inference

Various approaches to L-system inference were surveyed in [7]. Here, only certain works most closely related to PMIT-D0L are described. There are several different broad approaches towards the problem: building by hand [27, 53, 62, 64], algebraic approaches [23, 52], using mathematical properties [52], and search approaches [71].

Inductive inference was studied theoretically (without implementation) by Hermann and Rozenberg [32], and Doucet [23]. In [23], Doucet defined a set of equations that relate the number of symbols a_i observed in each string but the last in a sequence of strings ρ to all strings but the first. The (unknown) growth matrix must be a solution to these equations. Recently, this theoretical approach was extended to work for context-sensitive L-systems [45]. A somewhat similar approach [23] was implemented with a tool called LGIN [52] that infers L-systems from a single string. LGIN looks exhaustively at the successor combinations, extracted from a single string in a developmental sequence that fulfills these equations. Since only a single string is used, it does not guarantee to find a unique solution. LGIN is limited to two symbol alphabets (not including turtle graphic symbols), with larger alphabets described as "immensely complicated" [52]; however, LGIN was evaluated on six variants of "Fractal Plant" [62] and was very fast having a peak time to find the L-system of less than one second for 5 of the 6 variants, and four seconds for the remaining variant.

Runqiang et al. [71] investigated inferring an L-system directly from an image using a GA. In their approach, they encode each symbol within the successors as a gene. The fitness function attempts to match the candidate system to the observed data. As an input, they use an image of the shape produced by an L-system to be inferred, as seen for example in Figure 3.2. Their fitness function uses image processing to match the image produced by a candidate solution to the input image since their focus is on finding the proper angles and line lengths for drawing the image, in addition to the L-system itself. Their approach is limited to an alphabet size of 2 symbols and a maximum total length of all successors of 14. Their approach is 100% successful for variants of "Fractal Plant" [62] with $|V| = 1$, and a 66% success rate for variants of "Fractal Plant" [62] with $|V| = 2$. Although they do not list any timings, their GA converged after a maximum of 97 generations, which suggests a short runtime. While there are some superficial similarities between PMIT-D0L and their inference algorithm; i.e., both use a GA to infer an L-system, there are considerable differences. Their approach uses what we call an *ordered sequence of symbols* encoding scheme (discussed further in Section 3.3.1), while our work presents some more efficient encoding schemes for L-system inference. Also, their approach focuses on reproducing an image, while PMIT-D0L focuses on reproducing the input strings, allowing PMIT-D0L to be used for any process so long as a sequence of strings can be provided as input. Most importantly, this paper focuses heavily on search space reduction techniques that make PMIT-D0L successful.

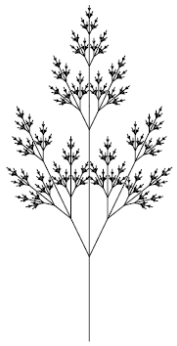


Figure 3.2: "Fractal Plant" variant #5 [62]. Image reproduced with permission of the copyright holder.

3.2.4 Scanning for Successors

A technique for finding productions based on the successor lengths for every $A \in V$ was previously described in [45], which we call the *scanning process*. This technique is used extensively in this research, and is described as follows. With L-systems, although the symbols are replaced in parallel, the sequence of successors in the new word is unchanged from the sequence of the original symbols; i.e., if $\omega_i = A_1 A_2 \dots A_m$ with $A_i \in V$, $2 \leq i \leq m$, then $\omega_{i+1} = succ(A_1)succ(A_2) \dots succ(A_m)$. Consider the case of A_1 in ω_i. To find $succ(A_1)$ it is only necessary to know the length of $succ(A_1)$. If $|succ(A_1)|$ is known (or different values for it are tested by searching), then the successor is the first $|succ(A_1)|$ characters of ω_{i+1}. The process of taking the next l symbols (where l is a hypothetical successor length) may be repeated for every new instance of a symbol encountered while scanning each word of a developmental sequence until every successor has been found, as described in [45]. With this fast algorithm, the goal is therefore to find a list of successor lengths that results in an L-system compatible with a developmental sequence; however, this may be done in a few ways. Most directly, a list of successor length values may be found by searching. Somewhat indirectly, the growth values for every $A, B \in V$ may instead be found, and a successor length for each $A \in V$ computed by summing the growth values for every $B \in V$.

3.3 Methodology

This section describes the design, development, and the process for evaluating PMIT-D0L. First, a high level overview of PMIT-D0L will be described, as this helps to contextualize the remainder of the section. This is followed by a description of the different encoding schemes used to define the space searched by PMIT-D0L to infer an L-system. The different techniques used to reduce the size of the defined search space are then discussed. The final two sub-sections describe the process used to optimize the control parameters of the GA, and finally, the fitness function and termination conditions.

Since many symbols, such as the turtle interpretation symbols, usually have identity productions, a set $\widehat{V} \subset V$ is provided as input where it is assumed that all symbols in $\widehat{V}$ have identity productions. Their

known values can speed up searches, but they can also be used to set up associations between the strings as described below (see Section 3.3.2). In addition, let $\overline{V} = V - \widehat{V}$ (those symbols with unknown productions).

Algorithm 3.3.1, which is implemented in C++ with CLR extensions, describes PMIT-D0L at a high level. It takes as input a sequence of strings $\rho = (\omega_1, \ldots, \omega_n), \omega_i \in V^*, 1 \leq i \leq n$, a desired encoding scheme En (described in Section 3.3.1), and the set of symbols $\widehat{V}$. PMIT-D0L will return either a D0L-system compatible with ρ or report that no D0L-system was found (as a GA cannot guarantee to find a solution) that is compatible with ρ. Due to the scanning process described in Section 3.2.4; deducing successor lengths and growth matrix values is particularly important. Thus, the first step is to initialize several programming variables. The current upper and lower bounds for the growth values of every pair of $A, B \in V$ is denoted by $(A, B)_{min}$ and $(A, B)_{max}$, and the bounds on successor length for each $A \in V$ is denoted by A_{min} and A_{max}. An alphabet V' is initially equal to $\widehat{V}$, and its symbols will be iteratively removed (as described below). Additionally, a prefix, suffix, and superstring of each successor A is calculated and stored in Pre_A, Suf_A, and Sup_A (these are called *successor fragments*) respectively. They are initially set to $NULL$ indicating that no successor fragment has yet been found.

Two assumptions are made regarding L-systems in this paper. First, the branching symbols "[" and "]" are assured to be properly nested within each production. Second, it is assumed that there are no erasing rules (i.e. a successor may not be λ). Most plant models in the literature are of this form. This implies that A_{min} is initialized to 1 for each $A \in V$. For each symbol $T \in \widehat{V}$, $T_{min} = T_{max} = 1$,$(T, T)_{min} = (T, T)_{max} = 1$ and $(T, A)_{min} = (T, A)_{max} = 0$ for every $A \in V, A \neq T$. We assume that all turtle interpretation symbols are in $\widehat{V}$ except possibly F, as this is most common. If $F \rightarrow F$ is a production, then we assume that $F \in \widehat{V}$.

Following the initialization, an initial analysis is done using the inputs V and ρ to refine the bounds on growth, the bounds on length, and to find any possible successor relationships by using a set of mathematical properties described in Section 3.3.2. This is done in a loop until no further refinement is found. The main processing then begins consisting of a nested loop. The outermost loop iterates over all symbols in V'; however, if V' is empty it still executes once (i.e. to account for the case where there are no symbols with already known identity productions). The main purpose of this loop is to search for a solution as if ρ contained only those symbols currently in $V - V'$ (as described in Section 3.3.2) which can simplify the problem. To this end, ρ' is constructed by filtering out all symbols in V' from ρ (i.e. that the symbols in $V - V'$ remain in ρ). This is followed by the innermost loop that refines the growth and length bounds and the successor fragments based on the same mathematical properties but using ρ' (Section 3.3.2). The innermost loop executes until no further refinement occurs during an iteration. The outermost loop then continues by defining a search space (*space*) in accordance with the desired encoding scheme En (as described in Section 3.3.1). The bounds on the search space are constructed from the growth and length bounds and the fragments as appropriate. A GA is used to search the defined space until a D0L-system is found that is compatible with ρ' or a termination condition is reached (as described in Section 3.3.4). Assuming a solution is found, then the solution found is a D0L-system compatible with ρ'; however, the successor fragments

can be refined by adding back the symbols of V' to the solution. The final step is to remove one symbol from V'. Conceptually, this last step can be thought of as iteratively solving simplified problems over smaller alphabets, then gradually re-introducing the symbols until an L-system compatible with ρ is found (described in Section 3.3.2).

Data: A sequence of strings ρ over V, an encoding scheme En, and a set of symbols with known identity productions $\widehat{V}$

Result: D0L-system compatible with ρ, or reports that no compatible D0L-system is found

$V' \longleftarrow \widehat{V}$ (Section 3.3.2);

// initialize length and growth bound variables;

// initialize fragment variables;

repeat

  refine growth bounds (Section 3.3.2);

  refine length bounds (Section 3.3.2);

  refine fragments (Section 3.3.2);

until *there are no changes*;

repeat

  $\rho' \longleftarrow$ filter ρ by removing all symbols in V' (Section 3.3.2);

  // using $V - V'$ and ρ';

  repeat

    refine growth bounds (Section 3.3.2);

    refine length bounds (Section 3.3.2);

    refine fragments (Section 3.3.2);

  until *there are no changes*;

  define a search space *space* based on En (Section 3.3.1);

  $solution' \longleftarrow$ search *space* using genetic algorithm (Section 3.3.1);

  add symbols of V' back to $solution'$ (Section 3.3.2);

  remove some symbol from V' if $V' \neq \emptyset$ (Section 3.3.2);

until $V' \neq \emptyset$;

Algorithm 3.3.1: D0L-system inference

3.3.1 Defining an L-system Search Space

This section describes the different encoding schemes used in this research, and in some cases existing approaches to using encoding schemes [23,52,71], for inferring D0L-systems. Broadly, the encoding schemes can be broken down into three categories: ordered sequence of symbols (OSoS), growth-based, and length-based. The OSoS approaches take the viewpoint that a successor is an ordered sequence of unknown symbols, and

so the search space is represented that way. Another approach investigated in this research is to instead attempt to determine successor lengths as the unknown, as an intermediate step, before determining the actual productions using the scanning process. Similarly, the growth values may be inferred first, and then simply summed for each $A \in V$ to produce a successor length followed by the scanning process.

Ordered Sequence of Symbols Encoding

While the technique of building a search space based on the idea of searching for the symbol in each position of each successor has been previously investigated [47, 71], PMIT-D0L creates this search space with additional requirements.

For every $A \in V$, a successor may be encoded as follows. For the remainder of this section, we create a special symbol $\overline{\emptyset}$. A number of genes equal to A_{max} is defined as this is the greatest number of symbols that may exist in the successor. The current values for Pre_A and Suf_A then identify a number of genes at the beginning and end equal to the length of the prefix and suffix respectively. For example, if $A_{max} = 7$, $Pre_A = A$, and $Suf_A = BB$, then the genome would appear as follows A _ _ _ _ B B, where _ represents an unknown symbol, which could be set to $\overline{\emptyset}$ if $|succ(A)| < 7$ (implying no symbol of V exists in that position of this successor considered). Each of the genes are permitted to have a real-value from 0 to 1. The most obvious way to decode a real-value is to map it onto $|V|$ equally sized ranges; however, with the information gathered by PMIT-D0L, a more effective approach can be used by dynamically altering the possible selections for each gene value. For each gene, a list of possible choices can be computed, a so-called *symbol pool*, based on the state of the successor at the time the choice is being made. To continue the previous example, if $(A, B)_{max} = 2$, then since there are already two occurrences of B symbols in the successor, B needs not be a choice for any of the remaining genes. The requirement $(A, B)_{min}$ can be similarly enforced. Continuing the example, if $(A, A)_{min} = 3$, $(A, C)_{min} = 2$, and the successor is A A _ _ _ B B, then the remaining genes must be either A or C regardless of how many symbols are in V. The lower bound on successor length is enforced by making $\overline{\emptyset}$ unable to be selected until $|succ(A)|_{min}$ symbols exist in the successor.

Furthermore, after a list of possible symbols for a gene is determined, instead of giving each symbol an equal chance of selection the ranges can be improved. For example, if the choices for a gene are $A, B, \overline{\emptyset}$, then one approach is to set the ranges from 0 to 0.33, 0.34 to 0.67, and 0.68 to 1.0 for A, B and $\overline{\emptyset}$ respectively. To further improve the searching with OSoS, it is possible to further refine the mapping with a lookahead in ρ. Consider the following string $AAAAAAAAAAAAABBBA$. If the state of a successor is A _, then the best choice (barring any additional restrictions or information) for the unknown symbol is A, since the symbol A is quite frequently the next symbol in the string following an A. This logic may be extended out to considering N characters. For example, with the string $AABAACAAB$, if the successor state is A A _, then A is impossible, B is 67% likely (occurs two out of three times), and C is 33% likely (occurs one of of three times). Note, these examples consider only a single string; however, in PMIT-D0L all of ρ is used to compute the associated probability. The $\overline{\emptyset}$ symbol is initially assigned a probability equal to

$1/(1 + (A_{max} - A_{min}))$, but since adding this probability with the probabilities for the other symbols in V results in a sum of probabilities greater than 1, the sum is then normalized. This encoding scheme is called OSoS(N), where N is the length of the lookahead. This paper evaluates both OSoS(1) and OSoS(2).

Growth Encoding

The growth-based approach, called PMIT-D0L(G), constructs bounds on each position of a possible growth matrix, of which there are $|V|^2$ values. PMIT-D0L(G) uses a literal encoding scheme and is similar to those seen in [23, 52], where the correct value of a dimension is each value in $M(A, B)$. The GA searches within the computed lower and upper bounds for $M(A, B)$. For each combination of growth matrix tested, the sum of each row is obtained to give a length and then the scanning process is used.

One limitation of this encoding is the possibility that a candidate does not satisfy the property that, for each $B \in V$, $\sum_{A \in V}(|\omega_i|_A M(A, B)) = |\omega_{i+1}|_B, 1 \leq i < n$ (total growth constraint). To avoid backtracking with this approach the GA is free to select any values within the lower and upper bounds for each $M(A, B)$. An alternate encoding scheme was also tested by changing the encoding scheme to use a real value from 0 to 1, and then it mapped ranges to the options that would permit the total growth constraint to be satisfied. The results of an evaluation were found to be very similar to those for the encoding scheme described above, and so results for this alternative approach are omitted.

Length Encoding

The length-based approach called PMIT-D0L(L) uses the scanning process that requires a successor length for each $A \in V$. Each dimension is ultimately mapped onto an integer value representing the length of a successor of a symbol in V. The values of each dimension represents the length of a successor, with the dimensions bounded by the computed upper and lower bounds for length. Compared to the growth-based approach, although the bounds on the individual dimensions are larger; i.e., typically, the lower bound $A_{min} \geq \sum_{B \in V}(A, B)_{min}$, and the upper bound $A_{max} \leq \sum_{B \in V}(A, B)_{max}$, with the length-based approach. The number of dimensions in the search space is $|V|$ with the length-based approach.

Thus, compared to PMIT-D0L(G), the main advantage is a reduced search space size in almost all but the most trivial of cases (formalized in the remark below). While, the search space size is reduced, a growth-based search space can still be more efficient because the fitness landscape may be more conducive to an effective search (i.e. a partially correct candidate may be a better stepping stone to a solution based on the manner in which a search algorithm exploits existing solutions).

Remark 1. With Algorithm 3.3.1, the search space size in the length-based approach is smaller than the search space size in the growth-based approach when $|V| \geq 2$ and the bounds of the growth-based approach are larger than 3, which can be seen as follows: Let S_A be the size of the dimension for A in the length-based paradigm, and $S_{(A,B)}$ be the size of the dimension for $M(A, B)$ in the growth-based paradigm. Algorithm 3.3.1 refines S_A such that it cannot be larger than $\sum_{B \in V} S_{(A,B)}$, therefore the size of the search space in the length-

based paradigm in the worst case is $\prod_{A \in V} \sum_{B \in V} S_{(A,B)}$; however, in the growth-based paradigm the search space is $\prod_{A,B \in V} S_{(A,B)}$. If $|V| \geq 2$, and $S_{(A,B)} \geq 3$ for all A, B, then $\prod_{A,B \in V} S_{(A,B)} > \prod_{A \in V} \sum_{B \in V} S_{(A,B)}$.

As with the growth-based approach, an alternative real-value encoding that enforced the constraint that $\sum_{A \in V}(|\omega_i|_A |succ(A)|) = |\omega_{i+1}|, 1 \leq i < n$ was implemented. The evaluation showed that the results were also not significantly different overall, and so this approach is not discussed further.

Row-Reduced Matrix Encoding

As discussed in Section 3.2, Doucet [23] recognized that the productions could be represented as a matrix equation, so as a step towards improving PMIT-D0L, a similar approach was implemented. Let Y be a matrix where each row i, from 1 to $n-1$ is the Parikh vector of ω_i (where ρ has n strings), and let Z be a matrix where each row is a Parikh vector of ω_2 to ω_n. In this case, if M is a growth matrix of an L-system with ρ as its length n developmental sequence, then $YM = Z$ is true. In Doucet's original work, he proposes to solve for M, and where Y is invertible, $Y^{-1}Z$ is a unique solution, and if the solution is not unique, to use linear Diophantine equations.

It is also possible to replace M with the length of each production, called the *successor length matrix*, and Z is replaced with the length of ω_2 to ω_n. M can then be solved for similarly to the growth matrix. The remainder of this discussion will be presented in the context of a length-based matrix; however, similar logic applies to a growth-based matrix by replacing growth values for successor lengths.

Gauss-Jordan elimination can be applied, and where there is not a unique solution, it results in a set of linear Diophantine equations, where the successor lengths are the variables, e.g. $5X_1 + 3X_2 = 24$. Each successor length only gets substituted for variables that appear in exactly one equation. In these cases, there are an infinite number of possible solutions. However, when inferring L-systems, the successor lengths are constrained to be natural numbers and within the bounds on the lengths provided by the lengths of the words in ρ, and it is therefore finite. For each equation, the encoding scheme used to search for a solution has N genes, where N is number of variables in an equation. The range of values for each gene is A_{min} to A_{max} for the symbol A the gene is representing (which can be more restricted than solutions to Diophantine equations due to the additional mathematical properties in Section 3.3.2 used that takes the sequences of the words into account). This encoding scheme is designated as PMIT-D0L(M+L) to indicate the addition of the matrix operation. For the matrix based on growth values, it is designated as PMIT-D0L(M+G).

Using the equations means that only possible solutions to the equations need to be checked as opposed to simply iterating over all possible lengths between the lower and upper bounds, thereby reducing the search space size. So, the value of the gene is dynamically changed. For example, say $A + B + C = 10$, and A, B, C have ranges 5 to 7, 1 to 5,1 to 5 respectively. If the GA picks $A = 7$, and $B = 4$ for the second gene, then B can be dynamically reduced to 2 allowing $C = 1$. This is a deterministic mapping and therefore permissible for a GA.

3.3.2 Reducing Search Space Size

In this section, the techniques that are used by PMIT-D0L to reduce the size of the solution space (with any of the encoding schemes described above) using mathematical properties of D0L-systems will be described. These are applied in Algorithm 3.3.1 everywhere that refine growth bounds, length bounds and refine fragments is performed. Being based on necessary conditions guarantees that all valid solutions are in the remaining search space (if there is a D0L-system that can generate the input strings).

Refining Successor Relationships

PMIT-D0L determines *successor fragments*, of which there are four types.

- A word ω is an A-subword fragment if ω must be a subword of $succ(A)$.
- A word ω is an A-prefix fragment if ω must be a prefix of $succ(A)$.
- A word ω is an A-suffix fragment if ω must be a suffix of $succ(A)$.
- A word ω is an A-superstring fragment if ω must be a superstring of $succ(A)$.

As PMIT-D0L runs, it can determine additional successor fragments, which can help in turn to reduce growth bounds. Certain prefix and suffix fragments can be found for the first and last symbols in each input word by the following process. Consider two words such that $\omega_1 \Rightarrow \omega_2$. It is possible to scan ω_1 from left-to-right until the first symbol from $\overline{V}$ is scanned (say, A, where the word scanned is αA). Then, in ω_2, PMIT-D0L skips over the symbols of $\widehat{V}$ in α (since each symbol in α has a known identity production), and the next A_{min} symbols (the current lower bound for $|succ(A)|$), β say, must be an A-prefix fragment. Furthermore, since branching symbols must be well-nested within a successor, if a [symbol is met, the prefix fragment must also contain all symbols until a matching] symbol is met. Similarly, an A-superstring fragment can be found by skipping α, then taking the next A_{max} symbols from ω_2 (the upper bound on $|succ(A)|$). If a superstring fragment contains a [symbol without the matching] symbol, then it is reduced to the symbol before the unmatched [symbol. The lower and upper bounds $(A, B)_{min}$ and $(A, B)_{max}$ for each $B \in V$ can be then possibly improved by counting the number of B symbols in any prefix and superstring fragments respectively. For a suffix fragment, the process is identical except the scan goes from right-to-left starting at the end of ω_1.

Example 1 *Consider input strings:*

$$\omega_1 = +++A[-FF][+F]BF$$

$$\omega_2 = ++++A[-FF][-FF][+F][+F]BFF.$$

Assume thus far PMIT-D0L has calculated that $A_{min} = 2$ *and* $A_{max} = 8$*. It can scan* ω_1 *until* A *is found and record that* $\alpha = +++$*. An A-prefix fragment is* $+A$ *as those are the first two* (A_{min}) *symbols of* ω_2 *after*

skipping α. An A-superstring fragment is $+A[-FF][$ as those are the first eight (A_{max}) symbols of ω_2 after skipping α, which can be reduced to $+A[-FF]$ due to the unbalanced $[$ symbol. By counting within the prefix fragment, lower bounds on the growth for A are $(A,+)_{min} := 1$ and $(A,A)_{min} := 1$, while upper bounds can be found from the superstring fragment to be $(A,+)_{max} := 1, (A,-)_{max} := 1$, $(A,A)_{max} := 1$,$(A,[)_{max} := 1$, $(A,])_{max} := 1$ and $(A,F)_{max} := 2$.

Refining Successor Relationship Using Symbols as Markers

If a symbol has a known successor, then it may be possible under certain circumstances to "line it up" with the symbol(s) it produces. In such circumstances, the derivation is sliced into two parts, and this reduces the possible productions for the symbols in each part. To illustrate the concept, consider the simple example:

$$\begin{array}{rl} \omega_1\text{: } & A{+}B \\ & \Big\downarrow \\ \omega_2\text{: } & \underbrace{ABA}_{\text{succ(A)}} + \underbrace{BBB}_{\text{succ(B)}} \end{array}$$

If $+ \in \widehat{V}$, then PMIT-D0L assumes that $+ \to +$, and the $+$ symbol in ω_1 may only produce the $+$ in ω_2. This splits the derivation such that everything to the left of the $+$ in ω_1 must produce everything to the left of the $+$ in ω_2, i.e. $A \to ABA$ is a production. Using similar logic except to the right of the $+$ implies $B \to BBB$ is a production.

In practice, it is unusual for a single position of one string to be uniquely associated with a position in the next string, as seen in the example above. More often, any individual position may be associated to multiple positions of the next word. However, a sequence of symbols, each with known successor, may be unique. For example, in the string $A[+B][-B]A[+[-C]][-[+D]]$, the individual symbols [,], +, and − alone might not uniquely associate to their successors; however, a sequence of symbols such as][− or]][−[+ are potentially unique, and as such may be associated to a unique location. To make use of this observation, for each word, a list of possible associations between every position of a symbol in $\widehat{V}$ to positions of the next word is constructed. This list of associations is called a *marker map*. A marker map is constructed based on both individual symbols and sequences of symbols, which are referred to as *candidate markers*. Associating a candidate marker to potential successors takes into account that a number of symbols must be reserved for symbols that appear before and after the candidate marker. For example, if $\omega_1 = A{+}BC{-}, \omega_2 = A{+}BC{+}C{-}$, $B_{min} = C_{min} = 1$, and $+$ associates with both $+$'s in ω_2, both are candidate markers. But since $B_{min} + C_{min} + -_{min} = 3$ the final 3 symbols of ω_1 produce at least the $+C-$ of ω_2. This eliminates the second $+$ in ω_2 as being produced by the $+$ in ω_1, and the $+$ in ω_1 can only be associated to the first $+$. If, following the construction of the marker map, a candidate marker is not uniquely associated with its successor, then it is removed from the marker map.

Consider a derivation $\omega_i \Rightarrow \omega_{i+1}$ expanded as

$$\omega_{i,1}A_1\omega_{i,2}\cdots A_m \Rightarrow \omega_{i+1,1}succ(A_1)\omega_{i+1,2}\cdots succ(A_m)\omega_{i+1,m+1},$$

where each A_j in ω_i is associated to the annotated successor in ω_{i+1} forming a marker. It follows that $\omega_{i,j} \Rightarrow \omega_{i+1,j}$ for all j, $1 \leq j \leq m+1$. From this, improved successor fragments, growth and length bounds may be found.

Refining Growth and Length Bounds

Here the bounds on $(A,B)_{min}$ and $(A,B)_{max}$ are improved. As all properties are run in a loop, these bounds are also influenced by successor fragments, and markers as described above, which can result in significant improved bounds versus just examining what can be datamined from Parikh vectors alone. A programming variable for the *accounted for growth* of a symbol $A \in V$ for $2 \leq i \leq n$, denoted as $G_{acc}(i,A)$ is:

$$G_{acc}(i,A) := \sum_{B \in V} (|\omega_{i-1}|_B \cdot (B,A)_{min}).$$

The *unaccounted for growth* for a symbol A, denoted as $G_{ua}(i,A)$, is computed as $G_{ua}(i,A) := |\omega_i|_A - G_{acc}(i,A)$.

The unaccounted for growth can be used to improve the growth bounds. In particular, $(B,A)_{max}$ is set (if it can be reduced) under the assumption that all unaccounted for A symbols are produced by B symbols. Furthermore, $(B,A)_{max}$ is set to be the lowest such value computed for any word from 2 to n, where B occurs, as any of the $n-1$ words can be used to improve the maximum. And, $|succ(B)|_A$ must be less than or equal to $(B,A)_{min}$ plus the additional unaccounted for growth of A divided by the number of B symbols (if there is at least one) in the previous word, as computed by

$$(B,A)_{max} := \min_{\substack{2 \leq i \leq n, \\ |\omega_{i-1}|_B > 0}} \left((B,A)_{min} + \left\lfloor \frac{G_{ua}(i,A)}{|\omega_{i-1}|_B} \right\rfloor \right).$$

Indeed, the accounted for growth of A is always updated whenever values of $(B,A)_{min}$ change, and the floor function is used since $|succ(B)|_A$ is a non-negative integer. For example, if $\omega_{i-1} = ABA$, $\omega_i = ABABBBABA$, $(A,A)_{min} = 1$, and $(B,A)_{min} = 0$, then the accounted for growth of A in ω_i is computed by $G_{acc}(i,A) = (A,A)_{min} \cdot |\omega_{i-1}|_A + (B,A)_{min} \cdot |\omega_{i-1}|_B = 1 \cdot 2 + 0 \cdot 1 = 2$. This leaves two A's in ω_i unaccounted for. An upper bound on the value of $|succ(A)|_A$ is set when the A's in ω_{i-1} produce all of the unaccounted for growth in ω_i. So A produces its minimum ($(A,A)_{min} = 1$) plus the unaccounted for growth of A in ω_i (2) divided by the number of A's in ω_{i-1} ($|\omega_{i-1}|_A = 2$), hence $(A,A)_{max} := 2$. Similarly, $(B,A)_{max}$ is achieved when only B's produce all unaccounted for growth of A; this sets $(B,A)_{max}$ to $(B,A)_{min} = 0$ plus the unaccounted for growth (2) divided by the number of B's in ω_{i-1} (1), which is 2.

Once $(B,A)_{max}$ has been determined for every $A, B \in V$, the observed words are re-processed to compute possibly improved values for $(B,A)_{min}$. Indeed for each (B,A), if $x := \sum_{\substack{C \in V \\ C \neq B}} (C,A)_{max}$, and $x < |\omega_i|_A$, then this means that $|succ(B)|_A$ must be at least $\left\lceil \frac{|\omega_i|_A - x}{|\omega_{i-1}|_B} \right\rceil$, and then $(B,A)_{min}$ can be set to this value if its bound is improved. For example, if ω_{i-1} has 2 A's and 1 B, and ω_i has 10 A's, and $(A,A)_{max} = 4$, then at most two A's produce eight A's, thus one B produces at least two A's (10 total minus 8 produced at most by A), and $(B,A)_{min}$ can be set to 2.

In a similar fashion, the length bounds A_{min} and A_{max} can be set using unaccounted for length.

Refining Successor Relationships from Solution Projection

As previously defined, $\widehat{V} \subset V$ where all symbols in $\widehat{V}$ have a known identity production, and $\overline{V} = V - \widehat{V}$. Since a symbol in $\overline{V}$ cannot be produced by a symbol in $\widehat{V}$, in the nested loop of Algorithm 3.3.1, it is possible to first infer an L-system over $V - V'$ (V' is a programming variable initially set to $\widehat{V}$). For example, if $V = \{A, B, C, [,], +, -\}$ and $V' = \{[,], +, -\}$, then the first problem is to find each successor of A, B, C projected to $\{A, B, C\}$. After solving this initial problem, then a series of problems are solved for each symbol of $\widehat{V}$ to determine where it belongs in each successor. Note that "[" and "]" may be completed together due to the assumption that they are properly nested. Overall, symbol filtering simplifies the inference problem by making the difference between A_{min} and A_{max} smaller. Although more searches are needed, up to one additional search for each symbol in $\widehat{V}$, they are each in a smaller search space.

As described above, PMIT-D0L removes the symbols of $\widehat{V}$ temporarily by projecting ρ onto a reduced alphabet, and then iteratively adding each symbol of $\widehat{V}$ back into the problem one at a time. Additional information about the successors of an L-system, i.e. the successor relationships, that produces ρ as its length-n developmental sequence may be gleaned by analyzing the solution to each sub-problem. Let a solution to one of these sub-problems be called a *projected solution*, as it partly describes the final successors. The process for using the projected solutions is conceptually similar to that used for markers as positions in pairs of consecutive words will be "lined up", and from there successor relationships deduced. To begin some terminology for this process is provided. For every derivation step $\omega_{i-1} \Rightarrow \omega_i$, every position in ω_{i-1} is scanned, and associated to the possible locations in ω_i that it may produce. If a position may be associated to only one locale it is called *certain*; otherwise, the position is called *uncertain*. Uncertain positions are excluded from any subwords that are produced (described in detail by example next); thereby, resulting in the longest possible subword produced only of certain positions. If the positions before and/or after are certain, then an A-prefix (before certain and after uncertain), A-suffix (before uncertain and after certain), or $succ(A)$ (before and after certain) can be found.

For example, assume that the first sub-problem is to solve the successors projected onto the symbols of $V - V'$ resulting in the projected successor of A as BAB and of B as AB. Equations 3.1 to 3.3 show an example of the process of finding successor relationships from this projected solution. In Equation 3.1, it can be seen that the $+$ is uncertain; however, it must produce one of the two annotated $+$ symbols (it cannot

produce the first + as there are no surrounding [and] symbols). Since + is uncertain, it cannot be said definitively whether A produces the first annotated + in ω_2 or not. It could be that A produced it, or it could be that + produced it. However, an A-subword can be properly calculated, but the + cannot be included in it. Since the projected solution of A is BAB, then $succ(A)$ must contain everything between the BAB, which is $B[+A]B$ (α in Equation 3.1) and this is declared an A-prefix due to the uncertainty of the + symbol. In Equation 3.2, the + production is uncertain so the association used is that which ensures that the B-subword is valid; i.e., that the + produces the second annotated + of the pair. With respect to the −, at first glance it may appear uncertain; however, from the projected solution of B as AB, the − cannot produce the − between the A and $[+B]$. The subword $A-[+B]$ (β) is a B-suffix due to the uncertainty of the preceding symbol. Finally, shown in Equation 3.3, an A-subword is formed for A based on the projected solution; however, in this case, both the preceding symbol is certain and there are no following symbols; therefore, this is $succ(A)$. Note, that since $succ(A)$ is now known, this makes the production of the first + certain (in Equation 3.1), since it is now known that A did not produce it. Therefore $succ(B)$ can be deduced as $+A-[+B]$ could be produced. Since all of the properties are computed in a loop until no new information is found, this would be found on the next pass.

$$\begin{array}{l}\omega_1{:}A+B-A\\ \omega_2{:}\underbrace{B[+A]B}_{\alpha}++A-[+B]-B[+A]B\end{array} \tag{3.1}$$

$$\begin{array}{l}\omega_1{:}A+B-A\\ \omega_2{:}B[+A]B++\underbrace{A-[+B]}_{\beta}-B[+A]B\end{array} \tag{3.2}$$

$$\begin{array}{l}\omega_1{:}A+B-A\\ \omega_2{:}B[+A]B++A-[+B]-\underbrace{B[+A]B}_{\text{succ(A)}}\end{array} \tag{3.3}$$

3.3.3 Parameter Optimization

As described in Section 3.2.2, the function of the GA is controlled by the population size (P), crossover weight (C), and mutation weight (M) parameters. Optimizing these parameters is difficult based on general

Parameter	PMIT-D0L					
	OSoS(1)	OSoS(2)	G	M+G	L	M+L
P	110	105	90	95	105	100
C	0.80	0.80	0.85	0.90	0.85	0.85
M	0.17	0.14	0.07	0.09	0.10	0.10

Table 3.1: Optimized parameters for each variant of PMIT-D0L

principles, since the optimal settings will depend on the characteristics of the fitness landscape, which is problem specific [8]. As such, typically, the parameters are set by doing a *hyperparameter search*. Bergstra and Bengio [8] found that using a Random Search provides an effective means to optimize the GA's parameter settings. Using Random Search works as follows. A range of good values is selected for each control parameter. In this case, based on the work by Grefenstette [29], the ranges were set to $10 \leq P \leq 125$ in increments of 5, $0.6 \leq C \leq 0.95$ in increments of 0.05, and $0.01 \leq M \leq 0.20$ in increments of 0.01, with additional values of 0.001 and 0.0001 permitted. An initial mid-range value is selected for each parameter ($P = 60$,$C = 0.8$,$M = 0.10$), and this is considered the current parameter value set. Iteratively, sixteen trials of PMIT-D0L are executed with a random variant of the current parameter value set. Each parameter is randomly modified up or down by no more than two increments, i.e. P may be modified by -10, -5, 0, $+5$, $+10$, while also remaining within the ranges above. The variant parameter set that provides the best fitness value is considered the new current parameter value set. In the case of a tie, which was quite common with PMIT-D0L, the fastest execution time is used. When none of the sixteen trials provide an improvement over the current parameter value set, the hyperparameter search terminates. The resulting parameter value sets for each variant of PMIT-D0L is shown in Table 3.1.

3.3.4 Fitness Function and Termination Conditions

After a candidate L-system G is produced from the solution S, the following process is used to evaluate fitness. To begin, any G which produces more than double the expected number of symbols is assigned the maximum fitness value so it (practically) guaranteed to be culled in the survival step. Starting with ω_1, a developmental sequence of length n is produced using G denoted as $\overline{\rho}$. For each $\overline{\omega}_i \in \overline{\rho}$, or until it terminates (see below), the symbol in each position of ω_i is compared to the corresponding position in $\overline{\omega}_i$. An error is counted if the symbol does not match (like Hamming distance), or if there is no corresponding symbol (i.e., one of the strings is longer or shorter than the other). For example, when comparing $\omega_i = XYXXXY$ to $\overline{\omega}_i = XYYX$, there are four errors. The third and fourth symbols differ, and additionally ω_i has six symbols, while $\overline{\omega}_i$ has only four. This process terminates when the number of errors for the i^{th} derivation is greater than zero, as any errors will cascade forward. The fitness value is computed as the number of errors divided by the number of expected symbols (e.g., 4/6), plus the number of unchecked derivations $(n - i)$. This encourages the GA to find solutions that incrementally match ρ.

PMIT-D0L uses a three-part termination condition to determine when to stop running. Ideally, PMIT-D0L terminates when a solution is found with a fitness value of 0.0 as this corresponds to an L-system that produces ρ as its length n developmental sequence. PMIT-D0L will also terminate when the population is considered to have converged to prevent the GA from acting as a random search and skewing the results. First, the current generation Gen_{best} is recorded whenever a new best solution is found. If an additional Gen_{best} generations pass without finding a new best solution, the population is considered converged. To prevent random chance from causing early termination, PMIT-D0L must process at least $1,000$ generations. PMIT-D0L also terminates after a time limit is reached. For this paper, the time limit was set to four hours; however, this was mainly used to control the overall experimental time. In practice, a user may be willing to wait less or more time to find an L-system.

3.4 Evaluation

3.4.1 Data Set

To evaluate PMIT-D0L's ability to infer D0L-systems, ten fractals, plus the six plant-like fractal variants (one shown in Figure 3.2) inferred by the existing program LGIN [52,62], and twelve other biological models were selected from the vlab online repository [77]. The biological models consist of ten algaes, apple twig with blossoms (shown in Figure 3.1b), and a "Fibonacci Bush" (shown in Figure 3.1a). The dataset compares favourably to similar studies where only some variants of one or two models are considered [52,71]. The data set is also of greater complexity by considering models with alphabets from between 2 to 31 (non-identity) symbols compared to two symbol alphabets [52,71].

3.4.2 Performance Metrics

Two performance metrics are used to measure how well PMIT-D0L can infer D0L-systems. The first metric is *success rate* (SR) which is defined as the percentage of times PMIT-D0L can find any L-system compatible with the input sequence. The second metric is *mean time to solve* (MTTS) measured to the millisecond level since some models solve in sub-second time. Time was measured using a single core of an Intel 4770 @ 3.4 GHz with 12 GB of RAM on Windows 10. PMIT-D0L is only allowed to execute for at most 4 hours (14400 seconds) and reaching this limit is considered a failure. These metrics are consistent with those found in literature [52,71].

3.4.3 Results

There are three results discussed in this section. The first set of results is the performance metrics shown in Table 3.2. For PMIT-D0L(M+G) and PMIT-D0L(M+L), the systems where the matrix was invertible are marked with "*" as no searching was required. A raw average for each encoding technique is also provided.

Brute force was also implemented to highlight the effect of the GA over brute force, Table 3.3 shows a comparison of MTTS between these two search algorithms using the PMIT-D0L(M+L) (selected as it was overall the best algorithm for the brute force implementation). For brute force, the mathematical properties were still used, but to emphasize the effect of the search algorithm, the mathematical properties were only computed once instead of in a loop. The MTTS results are shown explicitly in seconds with the best result for a system bolded. SR is always 100% with the GA, and was either 0% or 100% for brute force. In cases where it is 0%, the MTTS result is bold, and the MTTS is 14000 (the max value). The third set of results, shown in Table 3.4, are based on examining the effect of different search space reduction techniques on search space size. A discussion is provided on a fitness landscape analysis of the different encoding schemes. The fitness landscape analysis examined the manner in which the GA traversed the search space, and examined the local neighborhood of candidate solutions.

3.4.4 Discussion

It is evident from Table 3.2 and the average row that OSoS(1) and OSoS(2) are worse than the other encoding schemes, and therefore using some form of length with the scanning process seems to be best. However, OSoS(2) is faster overall than OSoS(1) (especially for *Metamorphe*), and therefore additional lookahead is helping with OSoS. Overall, PMIT-D0L is 100% successful at inferring a diverse range of D0L-systems. The L-systems in the data set have different numbers of successors, with vastly different lengths, and structures. With respect to using Doucet's [23] approach to find a unique solution, the matrix is found to be invertible for a little less than half of the L-systems in the test set, but never for any of the biological models. However, for both PMIT-D0L(M+G) and PMIT-D0L(M+L) the addition of the matrix operation to reduce the search space provides a benefit over PMIT-D0L(G) and PMIT-D0L(L) respectively. It is not so clear cut as to which encoding scheme is best, although PMIT-D0L(M+L) is the fastest overall, finishing in an average of 0.391 seconds. Certainly, it can be seen that PMIT-D0L(M+G) and PMIT-D0L(M+L) tend to be better than those without the matrix operations, but the timings tend to be close. However, for Fibonacci Bush, PMIT-D0L(M+L) performed much better than PMIT-D0L(M+G). Overall, both are quite fast, and the same can be said for all of the growth-based and length-based encoding schemes. This leads perhaps to the conclusion that choosing between a growth-based or length-based encoding scheme is not so important, but rather the success of PMIT-D0L (of any type) is largely attributed to the space reduction techniques. This observation is further reinforced by the results in Table 3.3. Except where there are ties at 0.001 seconds, the data indicates not only that a GA provides substantial improvement over brute force, but also a single pass of computing growth and length bounds does not provide as much improvement when compared to PMIT-D0L(M+L) (*Dipterosiphonia* v1, *Pterocladellium*, and *Tenuissimum*). The conclusion that the success is tied mainly to search space reduction led to evaluating the effect more explicitly (described in the next paragraph). Additionally, this conclusion suggests as a future work to incrementally increase $|V|$ (or make other systematic changes to the test data) to see if a difference emerges between growth-based and

Model	$\|\Sigma\|$	PMIT					
		OSoS(1)	OSoS(2)	G	M+G	L	M+L
Algae	2	**0.001**	**0.001**	**0.001**	**0.001***	**0.001**	**0.001***
Cantor Dust	2	**0.001**	**0.001**	**0.001**	**0.001***	**0.001**	**0.001***
Dragon Curve	2	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**
E-Curve	2	24.312	23.075	0.026	**0.025**	0.088	0.029
Fractal Plant v1	2	**0.001**	**0.001**	**0.001**	**0.001***	**0.001**	**0.001***
Fractal Plant v2	2	**0.001**	**0.001**	**0.001**	**0.001***	**0.001**	**0.001***
Fractal Plant v3	2	**0.001**	**0.001**	**0.001**	**0.001***	**0.001**	**0.001***
Fractal Plant v4	2	**0.001**	**0.001**	**0.001**	**0.001***	0.002	**0.001***
Fractal Plant v5	2	**0.001**	**0.001**	**0.001**	**0.001***	0.003	**0.001***
Fractal Plant v6	2	**0.001**	**0.001**	**0.001**	**0.001***	0.002	**0.001***
Gosper Curve	2	0.022	0.023	**0.001**	**0.001***	0.006	**0.001***
Koch Curve	1	**0.001**	**0.001**	**0.001**	**0.001***	**0.001**	**0.001***
Peano	2	0.236	0.235	**0.202**	0.210	5.916	0.221
Pythagoras Tree	2	**0.001**	**0.001**	**0.001**	**0.001***	**0.001**	**0.001***
Sierpenski Triangle v1	2	**0.001**	**0.001**	**0.001**	**0.001***	0.010	**0.001***
Sierpenski Triangle v2	2	**0.001**	**0.001**	**0.001**	**0.001***	0.003	**0.001***
Aphanocladia	4	0.004	0.004	0.005	**0.001**	0.206	0.007
Dipterosiphonia v1	28	11.885	10.871	123.008	3.820	178.718	**1.639**
Dipterosiphonia v2	5	0.278	**0.236**	0.348	1.114	1.055	1.199
Ditira Reptans	4	0.009	0.009	0.004	**0.003**	0.039	**0.003**
Ditira Zonaricola	4	0.012	0.010	0.006	**0.003**	0.161	0.007
Herpopteros	4	0.019	0.017	0.007	**0.004**	0.070	0.006
Herposiphonia	5	0.026	0.022	0.016	**0.013**	0.190	0.015
Metamorphe	5	7732.255	512.040	1.381	3.793	**0.632**	2.387
Pterocladellium	30	22.631	8.805	0.944	**0.881**	4.120	3.192
Tenuissimum	31	0.851	0.871	0.603	**0.520**	120.619	1.141
Apple Twig	17	1.012	0.963	**0.914**	**0.914**	0.957	0.970
Fibonacci Bush	8	500.262	112.332	4.095	37.525	8.663	**0.108**
Average	n/a	296.208	23.912	4.699	1.744	11.481	0.391

Table 3.2: Comparison of MTTS in seconds for different encoding schemes for PMIT-D0L with the best MTTS bolded. $|\Sigma|$ indicates the number of non-identity symbols in the L-system. SR is 100% for all executions. Results with "*" indicate an invertible matrix.

L-system	GA Only	Brute Force
Dragon Curve	0.001	0.001
E-Curve	0.001	0.078
Peano	0.052	0.945
Aphanocladia	0.006	0.047
Dipterosiphonia v1	664.206	**14400**
Dipterosiphonia v2	1.212	1.077
Ditira Reptans	0.001	0.002
Ditira Zonaricola	0.012	0.011
Herpopteros	0.017	0.079
Herposiphonia	0.383	2.492
Metamorphe	1.589	10.769
Pterocladellium	751.886	**14400**
Tenuissimum	1565.095	**14400**
Apple Twig	1.348	11.186
Fibonacci Bush	1.620	0.185

Table 3.3: Comparison of MTTS in seconds for using GA and using brute force with single execution of bound reduction techniques for L-systems with a non-invertible matrix. For GA, SR is 100%, for brute force SR is 0% where bold, and 100% otherwise.

length-based approaches.

Table 3.4 shows the length-based search space size for each L-system in the test set projected onto $\overline{V}$ with a different set of space reduction techniques applied (in the columns). In Table 3.4, the "Trivial" column shows the search space for a single execution of the length bounds reduction technique (Section 3.3.2) as this is the least amount of analysis possible with PMIT-D0L while still allowing it to function (this is a necessary first step to initialize the A_{max} bounds). The "All" columns shows the search space size when all techniques are applied. The "Only Growth and Length Reduction" columns shows the search space size when only the Parikh vectors and word lengths are examined (Section 3.3.2). The "Only Successor Relationships" column shows the search space size when the techniques related to successor relationships are used alone (Sections 3.3.2 and 3.3.2), after a single execution of the length reduction techniques.

For this table, only the $\overline{V}$ alphabet was used as the search spaces for each additional problem where a symbol from $\widehat{V}$ is added to determine to which successor they belong are often small (≤ 16) in comparison after applying the solution projection technique (Section 3.3.2), with some exceptions. In particular, *Dipterosiphonia* v1, *Pterocladellium* and *Tenuissimum* had exceptional cases where the search spaces were larger for symbols from $\widehat{V}$ than the problem for $\overline{V}$, although even in the worst case the search spaces were still on the order of 10^4 or less when all of the techniques were applied. This was due to the solution projection

L-system	Trivial	All	Only Growth and Length Reduction	Only Successor Relationships
Algae	1	1	1	1
Cantor Dust	9	1	4	9
Dragon Curve	4	1	4	4
E-Curve	6557	4	2070	4
Fractal Plant v1	4	1	1	1
Fractal Plant v2	4	1	1	1
Fractal Plant v3	4	1	1	1
Fractal Plant v4	4	1	1	1
Fractal Plant v5	4	1	1	1
Fractal Plant v6	4	1	1	1
Gosper Curve	525	4	90	1368
Koch Curve	1	1	1	1
Peano	35144	1	2352	3×10^5
Pythagoras Tree	1	1	1	1
Sierpenski Triangle v1	4	1	1	12
Sierpenski Triangle v2	3	1	1	20
Aphanocladia	2×10^5	36	392	45
Dipterosiphonia v1	1×10^{18}	16	1×10^{18}	4×10^{12}
Dipterosiphonia v2	2×10^5	81	12615	1×10^5
Ditira Reptans	6084	49	196	49
Ditira Zonaricola	1×10^5	49	196	49
Herpopteros	6×10^5	25	1350	74
Herposiphonia	6×10^6	1	39520	3×10^6
Metamorphe	2×10^8	576	85698	4×10^8
Pterocladellium	6×10^{17}	81	1×10^{14}	5×10^{11}
Tenuissimum	1×10^{20}	96	2×10^{16}	1×10^{11}
Apple Twig	1×10^{11}	1	3×10^5	2×10^7
Fibonacci Bush	1×10^6	576	7200	7200

Table 3.4: Comparison of search space size for different space reduction techniques for PMIT-D0L. All values larger than $100,000$ shown in scientific notation.

technique being unable to mark the "f" and "F" symbols as certain. An examination of the three L-systems shows that more so than other L-systems in the test set, these have many symbols that have a prefix or suffix with a sequence of "f" and "F" symbols (the productions for Dipterosiphonia [48] are displayed in Table 3.5). Such a scenario is the worst case for the solution projection technique since it mainly excels at finding symbols that lay between the symbols from $\overline{V}$. This also explains why these L-systems have a higher MTTS as seen in Table 3.2.

From this table, it is evident that there exists a synergistic effect between the growth and length reduction, and the techniques for refining successor fragments. The resulting search space is much lower when all of the techniques are applied than when only one category of techniques is applied. The difference between these columns highlight the synergy between the various space reduction techniques. The information from refining the growth and length values helps to define some of the successor relationships by describing the necessary symbols in the successor. The techniques that use the successor relationships, in turn, help to refine information about the growth and length values of the successors by find a superstring of the successor.

While all of the techniques are essential overall, the use of successor relationships extracts information by utilizing the sequence in which the symbols appear in ρ. This works by capitalizing on the fact that, even though the symbols are replaced in parallel, the order of the successors is the same as the symbols; i.e, if $w_n = a_1 a_2 \cdots a_m$ then $w_{n+1} = succ(a_1)succ(a_2) \cdots succ(a_m)$. Thus, if the relationship between a_i and $succ(a_i)$ is known (or even partially known), much can be deduced about the location of $succ(a_i)$ in w_{n+1} based on the location of a_i in w_n. This in turn allows for the deduction of symbols near a_i. This is one of the main differences between PMIT-D0L and other existing approaches. Capturing information from the symbol sequence of strings in ρ should provide guidance towards future investigations on inferring L-systems.

The fitness landscape analysis was conducted in two parts. First, a number (r) of random points were selected in the search space. The fitness function was applied to the selected point and to each point in a local neighborhood around it within ± 3 in each dimension. The count of each unique fitness value was collected for the neighborhood. Second, the best and worst fitness values collected from the executions were examined from the perspective of how they change as the GA converges towards the solution. Although the analysis was done for each encoding scheme, and across all of the systems, ultimately the observations can be made mainly collectively. Any observations that were made for a specific encoding scheme will be specified.

The fitness values are extremely stratified, i.e. there are relatively few unique fitness values for any system. There are almost always less strata for the length-based encoding schemes versus the growth-based encoding schemes, and the growth-based schemes in turn have less than OSoS(2). This means that it can be difficult to find a better solution since so many solutions have the same fitness value and the GA will get stuck in a local optima (this can be seen in the figures as a plateau). The main difference is that the length-based schemes tend to have a local optima with relatively high fitness. OSoS and growth-based encoding schemes, by contrast, have a local optima with a low fitness. This suggests that there is a tighter relationship, or more information revealed, in the growth-based and OSoS encoding schemes with respect to the actual

Productions
$c \rightarrow FFFz[+k][-r]FFfd$
$z \rightarrow Fz$
$k \rightarrow lmfF$
$r \rightarrow stfF$
$d \rightarrow FFFz[+k][-r]FFfe$
$l \rightarrow fF$
$m \rightarrow n$
$s \rightarrow fF$
$t \rightarrow u$
$e \rightarrow FFFz[+fj]FFfg$
$n \rightarrow fFF[-A]Fo$
$u \rightarrow fFF[+A]Fv$
$j \rightarrow abF$
$g \rightarrow FFFz[+k][-r]FFfh$
$A \rightarrow fFB$
$o \rightarrow fFF[-B]Fp$
$v \rightarrow fFF[+B]Fw$
$a \rightarrow Ff$
$b \rightarrow c$
$h \rightarrow FFFz[+k][-r]FFfi$
$B \rightarrow fFC$
$p \rightarrow fFF[-C]Fq$
$w \rightarrow fFF[+C]Fx$
$i \rightarrow FFFz[-fj]FFfc$
$C \rightarrow fFD$
$q \rightarrow fFF[-D]F$
$x \rightarrow fFF[+D]F$
$D \rightarrow fF$

Table 3.5: L-system for *Dipterosiphonia* v1 [48].

solution, and that minor variations from the solution in length-based approaches generates much more error. Especially with respect to OSoS encoding, even though it was not as good with respect to MTTS, maybe there is a way to use the information revealed to help guide the other encoding schemes more quickly towards a solution using hybridization. Finally, the stratified nature of the search space suggests a probable reason why the mutation weight for the GA tends to be high; exploration is more important than exploitation for these encoding schemes and the L-system inductive inference problem.

In examining the local neighborhood, it was observed that it is rarely very uniform. Even small variations to gene values can have an impact on fitness values. This lack of uniformity suggests that global/local hybrid search, or a swarm-based search that focuses on searching the neighborhood of local optima (e.g. particle swarm optimization), should be effective choices. Alternatively, differential evolution may be a good choice since it has been shown to be effective on search spaces that are similar to the one found in this research [67].

3.5 Conclusions

This paper presented an evaluation and analysis of different encoding schemes for the Plant Model Inference Tool for deterministic context-free L-systems (PMIT-D0L) to infer L-systems from a sequence of strings. Some of the encoding schemes are based on or modifications of earlier works, while some are novel. The classical encoding schemes look at the problem of inferring successors as finding an ordered sequence of symbols for the successors [71]. Alternatively, considering the input strings and successors as Parikh vectors, a so-called growth-based or length-based approach can be used.

The evaluation of the different encoding schemes does not indicate a clear best encoding, however length-based approaches are the fastest for this test set. Much of this paper focused on techniques for reducing the search space size, mainly using necessary conditions. While all of the different techniques were found to reduce the search space, there was an observed synergistic effect when used in combination. The techniques are effective to the degree that the choice between growth-based and length-based is not particularly critical for this particular test set. Additional investigation should be done on larger or more complex L-systems to determine if this still holds true.

A fitness landscape analysis was also conducted and it was observed that the landscape is very stratified and very non-uniform in the local neighborhood of an arbitrary point. This suggests that exploration, especially of a local neighborhood, might be an opportunity for future improvement over genetic algorithm. A global/local hybrid or a swarm algorithm may be good choices, alternatively differential evolution has been previously shown to be a good choice for the types of fitness landscapes observed [67].

The inductive inference of L-systems from input strings allows for much more rapid development of models than the current approach of building models by hand [27, 53, 62, 64], which can take considerable effort. Additionally, by going directly from observation (strings) to a model, allows for mechanistic principles to be possibly revealed, as opposed to requiring expert knowledge to build the model.

Since PMIT-D0L seems capable of inferring L-systems with fairly large alphabets (at least 31 symbols in the test set in a fast manner), this work will be used as a base for investigating the inference of other, more complex, L-system extensions such as stochastic L-systems, parametric L-systems, and for using images as input. While inferring parametric L-systems is more complex than D0L-systems, some of the principles and techniques from this paper may be applicable, especially if these L-systems behave like D0L-systems in certain parts of their derivation, such as when parameters are used as a timing mechanism. Also, the use of markers and solution projection should still be applicable as well to parametric L-systems. As a final note, the speed of PMIT-D0L can be further enhanced using parallel processing, which will be explored in the future.

3.6 Acknowledgements

This research was undertaken thanks in part to funding from the Canada First Research Excellence Fund, and the National Science Engineering Research Council (I. McQuillan grant 2016-06172, J. Bernard scholarship).

Chapter 4

Stochastic L-system Inference from Multiple String Sequence Inputs

Abstract

Lindenmayer systems (L-systems) are a grammar system that consist of string rewriting rules. The rules replace every symbol in a string in parallel with a successor to produce the next string, and this procedure iterates. In a stochastic context-free L-system (S0L-system), every symbol may have one or more rewriting rule, each with an associated probability of selection. Properly constructed rewriting rules have been found to be useful for modeling and simulating some natural and human engineered processes where each derived string describes a step in the simulation. Typically, processes are modeled by experts who meticulously construct the rules based on measurements or domain knowledge of the process. This paper presents an automated approach to finding stochastic L-systems, given a set of string sequences as input produced by a hidden system. The implemented tool is called the Plant Model Inference Tool for S0L-systems (PMIT-S0L). PMIT-S0L is evaluated on $3,770$ *procedurally generated S0L-systems in the test suite under different experimental scenarios. The evaluation shows that PMIT-S0L correctly infers S0L-systems with up to* 9 *rewriting rules each in under* 12 *hours. Additionally, it is found that* 3 *sequences of strings is sufficient to find the original system in all cases in the test suite.*

4.1 Introduction

In 1968, Lindenmayer [41] proposed a grammar system later called Lindenmayer systems (L-systems) to model multicellular growth in plants. They consist of rewriting rules that are used to replace, in parallel, every symbol in a string with a word. The process of replacing all symbols with words in this manner is called a derivation step. Starting from an initial word, derivation steps iterate, thereby producing a sequence of strings. The strings can produce a visual simulation or model by interpreting the symbols as instructions, such as "draw a line", or "turn left/right" (e.g., [26,62,64,68]) with each string being one step of the temporal simulation. L-systems can be deterministic, implying that the simulation always proceeds the same way, with each string uniquely determined by the previous string; or stochastic, where the rules and derivations have a probability of occurring.

L-systems of different types have been particularly successful at modelling plant growth [7,62]. Stochastic models have successfully modeled different plant structures and processes [1,62], including Japanese Cypress trees [53]. As will be discussed, the algorithm presented in this paper is domain agnostic, and so may have applications in other diverse research domains where stochastic L-systems have been successfully used. These other domains include modelling protein secondary structure [20], arterial branching [26] and sedimentary formations [68]. The process for finding a stochastic L-system has been described by Galarreta et al. [26] as requiring "tedious and intricate handwork" that could be improved by an algorithm to "infer rules and parameters automatically from real ... images".

Currently, models using S0L-systems are generated predominantly by hand by experts (e.g., [26,53,62,68]), which is time consuming. In [20], automatic inference of S0L-systems was discussed specifically for the application of modeling protein folding. Their approach is domain specific as they use *a priori* scientific knowledge regarding proteins as the basis for constructing the L-system's rules. Additionally, often the resulting models from existing approaches are assessed aesthetically [20,53,68], which further reinforces that such approaches are domain specific as an aesthetic assessment can rarely (if ever) be transferred to another process. These two drawbacks hinder automatic inference.

One approach for inferring an L-system from one or more temporal sequences of images is to divide the problem into two steps: 1) a segmentation of each image into the letters such that they would approximately draw the image in a simulator, 2) the inference of an L-system from the sequences of strings. This work is concerned with the second of these steps, which is called the inductive inference of an L-system. This paper presents a generalized algorithm for inductively inferring S0L-systems from one or more sequences of strings, called the Plant Model Inference Tool for Stochastic Context-free L-systems (PMIT-S0L), which, despite the name, is domain independent. PMIT-S0L could possibly reduce the time and effort required to produce a model of a process from imagery. Furthermore, since PMIT-S0L uses no *a priori* information, it can also help reveal the mechanisms underlying a process (perhaps hidden from the images themselves), and thereby have additional scientific impact.

Inductive inference was defined in the 1970's, and studied briefly in the theoretical computer science community [23,32]. It has recently been studied for deterministic context-free L-systems [9,45] where it was found that all systems in a test suite of 30 L-systems found in the literature could be inferred accurately, each in under 4 seconds. Inference of L-systems generally was surveyed in [7].

One of the challenges with S0L-system inference, in comparison to deterministic L-systems, is the multitude of systems that may produce the sequences of strings. For PMIT-S0L, it is argued within that given a set of L-systems that can produce the string sequences, the best choice (in absence of any additional information) is an S0L-system that has the greatest probability of having produced the input strings. As such, the core concept for PMIT-S0L's algorithm is a greedy selection process that chooses rewriting rules such that after each choice, the result would be the S0L-system that locally maximizes the probability of producing the input strings.

In this paper, the implementation of PMIT-S0L is described. It uses a greedy algorithm hybridized with a search algorithm (exhaustive search and genetic algorithm are evaluated as the search algorithms). Then, an evaluation is presented for PMIT-S0L at inferring S0L-systems using 960 procedurally generated S0L-systems. These systems were generated using statistical properties of existing L-systems in order to be equally as complex as L-systems created by experts. Procedurally generated L-systems were used since only one S0L-system could be found explicitly in the literature [53] (other papers [20, 26, 68] created them but do not include it in their respective paper). The limited number of stochastic L-systems in the literature is due to the difficulty in constructing them ([53] took an entire lengthy paper to create and justify theirs as will be described in Section 4.2), which would be dramatically improved by an automated approach. PMIT-S0L is also evaluated on this published S0L-system for Japanese Cypress [53]. Additionally, the effects of having $M > 1$ sequences of strings as input for different values of M is investigated with respect to differences between the true S0L-system and the S0L-system provided as a solution by PMIT-S0L.

Scenarios where more than one sequence are used are important as, for example, it would be possible to have imagery from many plants of the same genotype of a species (or different genotypes of a species), and to desire a stochastic model to describe the population of plants. Furthermore, it would be particularly useful to compare populations of plants by comparing the models. Additionally, stochastic L-system models that can describe a diversity of plants are useful indirectly to improve image recognition tasks. For example, Ubbens et al. [75] use a large set of synthetic L-system images of *Arabidopsis thaliana* rosettes to augment the size of a data set used to train a deep convolutional neural network for the purposes of leaf counting. This was found to improve leaf counting on real rosette images versus only training using real images.

This work is an extension of the conference paper [10] that had the additional restrictions of only using one sequence of strings as input, and it had a limitation with respect to not having two successors of the same symbol with one being a prefix of the other — the so-called *prefix limitation*. This paper additionally presents methods for removing both limitations. PMIT-S0L is evaluated using a variety of performance metrics. Detailed in Section 4.3, the success rate describes the percentage of times that an S0L-system is found that has an equal or greater probability of producing the input than the original S0L-system has of producing the input. Thus, if the original L-system was defined as the correct L-system, then it can be possible to find L-systems that are more likely to have generated the input, and are therefore better than the L-system which actually generated the strings. The time taken to find a solution (or report that none exists) is also measured. Additional metrics are used to capture the differences between the predicted solution and the original system, both in terms of rewriting rules and probabilities.

The remainder of this paper is structured as follows. Section 4.2 provides some background information on L-systems and some applications of S0L-systems. Section 4.3 discusses some of the unique challenges of inferring stochastic L-systems, and how PMIT-S0L functions. Section 4.4 provides the methodology for evaluating PMIT-S0L, focusing on the methods used to evaluate multiple sequences of strings as input. Section 4.5 provides the results of the evaluation. Finally, Section 4.6 concludes the paper and discusses

future directions of PMIT-S0L and inferring L-systems.

4.2 Background and Preliminaries

To start, some basic formal notation is required. An alphabet is a finite set of symbols. Given an alphabet V, V^* is the set of all words over V. For a word x, the length of x is denoted by $|x|$. And for a finite set Y, the number of elements in Y is denoted by $|Y|$.

Formally, context-free L-systems are described by an ordered tuple $G = (V, X, P)$ where V is an alphabet, X is a finite set of strings $\{x_1, \ldots, x_q\}$ where each x_i, $1 \leq i \leq q$ is a word in V^* called an axiom (some definitions have only one axiom), and P is a finite set of productions (also referred to as rewriting rules, although the term productions will be used herein). Each production is of the form $A \to \alpha$ where $A \in V$ is called the predecessor and $\alpha \in V^*$ is called the successor of the production (or a successor of A). The system is said to be non-erasing (also called propagating in the L-systems literature) if α is not the empty word for any production. The term context-free here means that the neighboring symbols in the string do not affect the selection of a successor. A derivation step is defined by, $u \Rightarrow v$, if $u = A_1 \cdots A_n$, $v = \alpha_1 \cdots \alpha_n$, and $A_l \to \alpha_l \in P$, for $1 \leq l \leq n$. The system is called deterministic if X only contains one string, and P contains exactly one rule with each symbol in V; with deterministic systems, a derivation step on a string involves taking, in parallel, every symbol in the string and replacing it with its unique successor. In a stochastic L-system, every $A \in V$ has a set of one or more successors each with an associated probability of being selected, such that the sum of the associated probabilities for each $A \in V$ equals 100%. When performing a derivation step with a stochastic L-system, for each symbol in a string, a successor is chosen from the set of corresponding successors with the associated probability. Formally, a stochastic context-free, or S0L-system, [24] is a quintuple $G = (V, X, P, p, I)$, where (V, X, P) is a context-free L-system, p is a function from P to $(0, 1]$ such that, for all $A \in V$, $\sum_{A \to \alpha \in P} p(A \to \alpha) = 1$ (that describes the probability of applying a production), and I is a function from X to $(0, 1]$ such that $\sum_{x \in X} I(x) = 1$ (that describes the probability of starting with an axiom).

In [24], the authors continue with the following definitions for derivations of an S0L-system. Given x, y as words in V^* where $x \in X$, a derivation d of x to y of length m consists of two items:

1. a trace, which is a sequence of $m + 1$ words $(w_0, \ldots, w_m)$ such that $x = w_0 \Rightarrow \cdots \Rightarrow w_m = y$,

2. a function σ from the set of pairs $\{(j, l) \mid 0 \leq j < m, 1 \leq l \leq |w_j|\}$ into P such that, for j from 0 to $m - 1$, if $w_j = A_1 \cdots A_{|w_j|}, A_l \in V$, then $w_{j+1} = \alpha_1 \cdots \alpha_{|w_j|}$ where $\sigma(j, l) = (A_l \to \alpha_l)$ for l from 1 to $|w_j|$.

Thus, the function σ describes the productions applied to each letter in the derivation. Given such a derivation d, the probability of w_j deriving w_{j+1}, $P(w_j \Rightarrow w_{j+1}, d)$ is $\prod_{l=1}^{|w_j|} P(\sigma(j, l))$. Further, the probability of d

occurring is $P(d) = I(x) \cdot \prod_{j=0}^{m-1} P(w_j \Rightarrow w_{j+1}, d)$. Lastly given a finite set of derivations $\{d_1, \ldots, d_M\}$, the probability of them occurring is

$$\prod_{i=1}^{M} P(d_i). \tag{4.1}$$

As mentioned in Section 4.1, modeling with L-systems is done by associating a meaning to each symbol (or a subset of symbols) in V; typically the meaning is an instruction for simulation software such as the "virtual laboratory" [77]. So, a string of symbols is then taken as a sequence of instructions, and each derived word is taken as the next step in a temporal process. Symbols may have graphical and/or mechanical functions within the resulting model. A common graphical interpretation is the so-called *turtle graphics* [62], which imagines a turtle in a 2D or 3D space with a state consisting of a position and orientation. The graphical symbols then modify the turtle's state. When the turtle moves forward, it may optionally simultaneously draw a line. For branching models, two graphical symbols ("[" and "]") will push and pop the turtle's state onto a stack and switch to that state. Other symbols are used to represent components or an underlying mechanism in the model. For example, a symbol may be used to represent the apex of a plant where the stem will continue to grow at the next derivation step, until it flowers; this can be modeled stochastically [62].

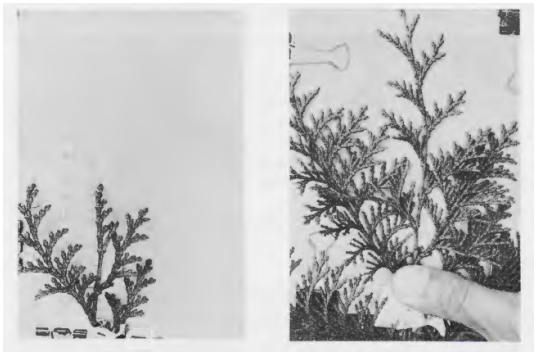
Nishida [53] investigated using S0L-systems to model Japanese Cypress trees. He did not use any *a priori* biological knowledge of Japanese Cypress, and instead used a process of converting imagery to segments, and then to an L-system. Importantly, this is the same process as our main goal (images to segments to L-systems, with PMIT-S0L being useful for the second step), except their work was done completely manually. In the paper, the meticulous process is described of segmenting images of Japanese Cypress by hand. The segments are then mapped to letters of an alphabet, with successors and associated selection probabilities picked for the segments such that they would reproduce the appropriate segmentation in the next image. The result is an S0L-system with 23 symbols and a total of 42 productions shown in Table 4.1. The system produces imagery similar to the photos of the Japanese Cypress, as seen in Figures 4.1a to 4.1d [53]. This shows that the goal of automatic inference in exactly this fashion, but in a completely automated manner, is an exciting opportunity, as doing so manually requires a huge amount of effort.

4.3 Inferring S0L-Systems using Greedy Algorithm

This section will describe the methodology of PMIT-S0L. For the remainder of this paper let $\rho = \{\rho_1, \ldots, \rho_M\}$ be the input, where each ρ_i is a finite sequence of strings over V. Furthermore, let $\rho_i = (w_{i,1}, \ldots, w_{i,m_i})$, $1 \leq i \leq M$ (each $w_{i,j}$ is a string in V^*). In this paper, we assume $m_1 = \cdots = m_M$ (all sequences can be truncated to have the same number of strings), and we denote this size by m. Inferring an L-system can be stated as outputting some L-system G (or reporting that none exists) that could produce all the sequences in ρ; that is, G has a derivation with a trace of ρ_i, for each i, $1 \leq i \leq M$. In this case, G is said to be compatible with ρ. To do this, the algorithm scans each sequence ρ_i in ρ and attempts to determine a derivation, which

(a) Observations H-4 of Japanese Cypress

(b) Image produced by an S0L-system similiar to observations H-4



(c) Observations H-6 of Japanese Cypress

(d) Image produced by an S0L-system similiar to observations H-6

Figure 4.1: Images from [53], reprinted with permission of Kyoto University.

$A \to CB, 0.61$	$A \to A, 0.39$	$B \to DA, 0.61$
$B \to B, 0.39$	$C \to E{+}F/, 1.00$	$D \to E{-}G/, 1.00$
$E \to E, 1.00$	$F \to JH, 0.70$	$F \to JF, 0.30$
$G \to KI, 0.70$	$G \to KG, 0.30$	$H \to JI, 1.00$
$I \to KH, 1.00$	$J \to V{-}L/, 0.99$	$J \to V{-}L/{+}M/, 0.01$
$K \to V{+}M/, 0.99$	$K \to V{-}L/{+}M/, 0.01$	$L \to NP, 0.60$
$L \to NL, 0.40$	$M \to OQ, 0.60$	$M \to OM, 0.40$
$N \to N, 0.60$	$N \to V{+}R/, 0.39$	$N \to V{-}S/{+}R/, 0.01$
$O \to O, 0.60$	$O \to V{-}S/, 0.39$	$O \to V{-}S/{+}R/, 0.01$
$P \to P, 0.50$	$P \to NT, 0.50$	$Q \to Q, 0.50$
$Q \to OU, 0.50$	$R \to WR, 0.23$	$R \to OR, 0.02$
$R \to R, 0.75$	$S \to WS, 0.23$	$S \to NS, 0.02$
$S \to S, 0.75$	$T \to T, 0.65$	$T \to NU, 0.35$
$U \to U, 0.65$	$U \to OT, 0.35$	$V \to V, 1.00$

Table 4.1: S0L-system for Japanese Cypress [8].

has a trace of ρ_i in a word-by-word fashion. Each time it determines part of a derivation (e.g., that some specific occurrence of a letter in $w_{i,j}$ produces the subword between two positions of $w_{i,j+1}$), it adds this production to the current list of productions if it has not already been added.

4.3.1 Complications with Stochastic Inference

If a compatible L-system does exist, then it must be found in the space of all L-systems, and hence it is at least possible to search for one. PMIT-D0L [9, 11] is an existing tool for inferring deterministic context-free L-systems (D0L-systems). The first PMIT-D0L implementation [11] used genetic algorithm, and searched for successors as an ordered sequence of symbols in a search space that was pruned using mathematical properties based on necessary conditions of L-systems. A significant improvement was made when it was recognized that every successor of a symbol $A \in V$ must be a subword of every word directly after one where A occurs (for a deterministic L-system $M = 1$ is enough since the word produced at each step is uniquely determined), and that searching for an ordered sequence of symbols could be replaced by searching for successor lengths for each letter of the alphabet [9, 45]. With each possible solution consisting of a successor length for each $A \in V$, it is possible to scan every symbol in each word in ρ starting from the first word, and take the subword of the associated length as the successor. When a scan can be completed without any inconsistencies, then the resulting successors are compatible with ρ. This is referred to as the *scanning process*. This approach works in the deterministic case because any information deduced about the successor for any instance of A in ρ must be true for every A that appears in every word in ρ, and therefore this search space has only $|V|$

dimensions.

For stochastic L-systems, while it still true that every $A \in V$ must produce a subword of the next string and hence some kind of encoding scheme with successor lengths is possible, it is no longer true that deducing a fact about some instance of A is true for all other instances of A. Indeed, with S0L-systems, different instances of each A (of each word of each sequence) in ρ can produce different successors. For $\rho_i = (w_{i,1}, \ldots, w_{i,m})$, then since every instance of a symbol can produce a different successor, the most intuitive solution space defines at least one dimension (for example, using the successor length encoding scheme from [9, 45]) for every symbol in each word of each sequence to produce a search space with N dimensions, where $N = \sum_{i=1}^{M} \sum_{j=1}^{m} |w_{i,j}|$. This is clearly intractable as an extra dimension is needed for every additional symbol in ρ. Furthermore, reducing the bounds for the dimensions (e.g., upper and lower bounds on successor length) is very difficult since there is little information upon which to deduce such bounds (due to the aforementioned issue that every instance of a symbol can produce a new successor). The lower bounds of each successor length is 1 (productions are assumed to be non-erasing in this paper) and the upper bound for an instance of A would be the length of the next word minus the number of symbols in the previous word (since every symbol produces at least one symbol) directly after one where that A occurs. In summary, to search for an S0L-system with the entire search space in a similar fashion to what has been done for D0L-systems, requires both a number and scale to the dimensions that would be too large to allow for a practical search time.

A further complication when inferring S0L-systems is there exist a multitude of possible S0L-systems that are compatible with ρ. By taking each pair of consecutive words $w_{i,j}$,$w_{i,j+1}$ in every ρ_i of ρ, then every way of dividing $w_{i,j+1}$ into $|w_{i,j}|$ subwords can be used to create productions that lead to a compatible S0L-system. However, one crucial and distinguishing property for any of these candidate solutions is the probability that it produces ρ. A best possible solution compatible with ρ is desired rather than an arbitrary solution.

4.3.2 Methodology with PMIT-S0L

PMIT-S0L infers an S0L-system based on ρ by selecting successors, such that each choice locally maximizes the probability of producing the words of ρ scanned so far. This paper investigates inferring an S0L-system by scanning the words symbol-by-symbol, and choosing each successor based on a successor length (similarly to PMIT-D0L in [9, 45]) by preferring successors that have been previously selected using a *greedy algorithm*.

To determine an S0L-system, an alphabet, axioms, and rewriting rules with associated probabilities need to be predicted. The alphabet V is found easily by recording every unique symbol in ρ, and so we henceforth assume it is known. For the S0L-system, multiple axioms are assumed. In particular, the set of all axioms X is assumed to be the set of all of the first words of sequences in ρ. We do not attempt to infer the probability of starting with each axiom (which could just be calculated as the number of times each axiom is the first word of a sequence in ρ divided by M). Also, computing a reduced number or a single axiom is an area for future investigation. The process for inferring the rewriting rules and their probabilities is more involved and is described in remainder of this section.

Successors	Odds
$A \rightarrow \alpha_1 : 90\%(9)$ $A \rightarrow \alpha_2 : 10\%(1)$	$0.9^9 \times 0.1^1 = 3.87\%$
$A \rightarrow \alpha_3 : 50\%(5)$ $A \rightarrow \alpha_4 : 50\%(5)$	$0.5^5 \times 0.5^5 = 0.097\%$

Table 4.2: Two abstracted S0L-systems with odds that it would generate a specific set of strings.

As mentioned earlier, for any ρ, there are a multitude of possible compatible S0L-systems, but a best solution is a S0L-system with the greatest probability of producing ρ [10]. In absence of any additional information (e.g., *a priori* knowledge that might lead to a different choice), the S0L-system with the greatest computed probability is said to have maximum parsimony. For example, Table 4.2 shows two abstracted S0L-systems, where in parentheses is the number of times that each successor was applied to produce a sequence of strings in the two different derivations. The "Odds" column shows the computed probability of the derivation occurring. It can be seen that the first S0L-system has a greater probability of producing the sequence of strings, and so should be preferred as the solution. The probability that an S0L-systems produced ρ is increased when one or a few successors have a high probability, as opposed to an equal distribution across all of the successors. This mathematical property provides the guidance for a greedy algorithm to infer an S0L-system using the process described next.

Conceptually, the core greedy algorithm component is relatively straightforward. Suppose that a list of successors for each $A \in V$ is being maintained including a count of how often each successor has been selected. For each word $w_{i,j}$ with j from 1 to m, the algorithm will partition $w_{i,j+1}$ into $|w_{i,j}|$ subwords by scanning each letter A in order from left-to-right as follows: If A is the rightmost symbol of $w_{i,j}$, then pick the successor to produce everything that remains in $w_{i,j+1}$ (the parts of $w_{i,j+1}$ that have not been matched in the derivation). If A is not the rightmost symbol, then pick the successor, if one exists, from the current list for A out of those that match the next symbols of $w_{i,j+1}$ that has been selected most often so far. One issue with this algorithm is that early on, especially for the first few symbols scanned, the list of successors for each $A \in V$ is (or is near) empty or none match, and so the greedy process is not able to make a choice from the existing list. Sometimes in the literature, this type of problem is resolved using a random forest algorithm [13], but this was found to not work well in this instance (discussed below). A search algorithm was used in these instances to pick successors when nothing in the list is matched.

For this, the algorithm keeps a vector y of N non-negative integers. When there is no successor in the current list for A that matches the next symbols in $w_{i,j+1}$, the algorithm retrieves the next unused integer from y, k say, and then a successor is selected using the next k symbols from $w_{i,j+1}$. To find y (a single sequence for all letters), a search algorithm is used (this search process will be described in Subsection 4.4.2). However, this process raises a new question: how many successor length choices are needed; i.e., what should be N?

The best value for N is the number of distinct successors in the best S0L-system, which is difficult to accurately predict in advance. However, if a guess at N is made, then either the guessed N will be exactly right, too low, or too high. If N is exactly right, then there are no issues. If N is too high, then so too will be the execution time, i.e., the search space becomes larger than necessary. If N is too low, then it is possible that the algorithm may reach a point where the greedy algorithm cannot make an appropriate choice and there is no element left in y. In this case, then either the search can be restarted with a higher value of N or y could be extended by one dimension. However, some solutions will encounter this issue repeatedly, thus requiring many extensions leading to an increased execution time. In practice, simply extending each time as needed is probably intractable; e.g., there would be candidate solutions that would required hundreds or thousands of additional dimensions. A balance can be achieved by having the implementation extend y by a limited additional number of times per word of ρ (PMIT-S0L uses a limit of one), which can help find correct solutions while keeping the expansion of the search space reasonable. If more than the chosen limit is needed, then a higher value of N is required.

Lastly, the process will terminate with error when none of the methods above can produce a successor for the current symbol A (this only occurred when N was too small in our tests). In this case, N needs to be incremented; however, it is unclear what is an optimal value for the increment. A small increment for N minimizes the growth of the search space, but increases the chance that the search will fail again. A large increment for N increases the chance that the subsequent search will succeed, but will possibly make the search space larger than necessary (certainly an increment of 1 will eventually find the correct N). In our experiments, a value of N that is too low is simply considered a failed state in order to better understand its effect on the time taken to complete. The value of N can be increased manually or automatically if desired.

Procedure 1 *In summary, for a given integer sequence y, the successor selection process works by choosing the first applicable rule when scanning each character in ρ. Let $z := 1$ be a programming variable.*

1. *If scanning the last symbol of the current word, then select all remaining symbols in the produced word,*

2. *An existing successor in the list for A matches the next symbols in $w_{i,j+1}$; the most frequently chosen thus far is selected greedily,*

3. *Build a successor of length y_z, and increase z by one,*

4. *Terminate with error.*

Given an appropriately chosen y, this process can determine a compatible S0L-system. Furthermore, notice, that this procedure is determining the derivations associated with traces in ρ. However, the solution need not be optimal (the greedy choice might not lead to the best solution).

4.3.3 Searching for Successor Lengths

Next, the method for determining the integer sequences y is described. Two algorithms, standard genetic algorithm (SGA) and exhaustive search (ES), are evaluated. For both, a literal encoding [5] is used, so a search space is constructed of N integer dimensions with bounds from 1 to 10 (as discussed and justified later in Section 4.4.2, the maximum number of symbols in each successor is assumed to be 10 although this could be increased). For each sequence y searched, Procedure 1 is used to predict an S0L-system G compatible with ρ, and the derivations corresponding to each trace in ρ. The fitness value of y is defined to be the probability of these derivations occurring with the traces in ρ, which is given in Equation (4.1). Algorithm 4.3.1 shows the pseudocode for PMIT-S0L.

Data: A set of M sequences of strings over V called ρ, a search algorithm S
Result: B, the S0L-system with the greatest probability of producing ρ of those searched; or reports that no compatible S0L-system was found
$T := false$ // variable to track if the algorithm should terminate;
B // stores the best solution found thus far;
$F(B) := 0$ // stores the fitness of B;
repeat
  $S.iterate()$ // performs one iteration appropriate for the search algorithm;
  foreach $y \in S.population$ **do**
    // Produce an S0L-systems G using the rules in Procedure 1;
    $(G, fitness) := Produce(y)$;
    if $fitness > F(B)$ **then**
      $B := G$;
      $F(B) := fitness$;
  end
  $T := S.terminate()$ // determines if S should terminate in an algorithm specific way;
until $T = true$;

Algorithm 4.3.1: This pseudocode describes the process for searching for an S0L-system with a higher probability of generating ρ.

When the search technique is set to SGA, it uses roulette wheel selection, uniform crossover, uniform mutation, and elite survival operators [5]. An SGA with simple operators is selected as little is known about the search space and an SGA provides easily tunable mechanisms for balancing exploration and exploitation. Briefly, an SGA works by the following steps [5]. It consists of a population of P genomes, where each genome in this case is a candidate vector y consisting of N genes. The initial population is produced randomly. The main processing loop of an SGA performs four steps: selection, crossover, mutation, and survival. In the selection step, $P/2$ pairs of genomes are selected from the population. The crossover step takes each pair

and randomly swaps zero or more genes between them, and since there are $P/2$ pairs, this results in P new genomes. Each new genome is then mutated by randomly changing zero or more genes to a new value. The survival step merges the initial and new populations. Each member of the population is assessed a fitness value as described above (using Procedure 1 with y from the population), and the population is sorted by fitness value. The top P genomes are preserved and the remainder are culled. With respect to Algorithm 4.3.1, *S.iterate*() for an SGA corresponds to performing the selection, crossover, mutation, and survival steps. *S.population* is the entire population of genomes. The SGA terminates by convergence detection (*S.terminate*() in Algorithm 4.3.1), which functions as follows. Termination occurs as follows: every time a new best solution is found, a variable *Gen* records the number of the current generation. If *Gen* additional iterations occur without a new best solution being found, then the SGA terminates.

To optimize the control parameters of the SGA, a hyperparameter search was done [8] using a random search with 16 trials. As a result of the hyperparameter search, the control parameters were set as follows: population size of 50, crossover weight of 0.9, and a mutation weight of 0.01.

In contrast, the ES simply iterates through all possibilities until it terminates, keeping track of the S0L-system with the highest fitness value. Since later dimensions are often unused, it would be preferred for these dimensions to be searched last, i.e., to search deeply into the leading dimensions, hence a depth first search is used. *S.iterate* corresponds to searching one step deeper. *S.population* is the number of candidates solutions in this iteration, which is always 1 with ES. *S.terminate* returns *true* when there are no more nodes to search.

4.3.4 The Prefix Limitation

In the conference paper [10], this earlier version of PMIT-S0L had an identified limitation, when for at least one $A \in V$, there are two or more successors, and one successor is a prefix of the other [10]. Let u, v be two successors of A, such that $u = vx$, and u, v, x are words over V. In this case, if the shorter successor (u) is encountered first in ρ, then assuming all other successors are correct, for all future instances of A, the greedy choice (rule #2 of Procedure 1) will always select u as the successor as the next $|u|$ symbols can be positively associated to A. This effect is shown in Example 2 below. Since the introduction of PMIT-S0L [10], a technique has been found to often correct this limitation, described next, although it is only used with ES and not with the SGA (due to complications to be discussed).

Example 2 *Let $w_1 = AAA$ and $w_2 = AAAAAABBB$. Suppose that the successors for A in the original system are $AAA\,; AAAB\,; BB$ (with some associated probabilities that are not important for the example). Finally, assume $N = 2$ (the length of y), and the search algorithm has a candidate solution with the value $3, 4$. The first A will be assigned the successor AAA based on the value found by the search algorithm (rule #3). The second A will be also assigned AAA based on the greedy choice (rule #2), but this decision is incorrect as the desired choice is $AAAB$. Finally, the third A will also have the wrong successor of BBB (chosen by rule #1).*

In this example, in order to find the correct successor, the greedy choice (rule #2) needs to be ignored and instead to use the 2^{nd} value in the search space (4) should be used to find the successor $AAAB$. Instead, y is modified to be of the form $y = (t_1, y_1, t_2, y_2, \ldots, t_N, y_N)$. Procedure 1 is modified so that after using y_z for the length in rule 3, it only allows at most t_{z+1} greedy choices before forcing it to not apply any more greedy choices (skip rule 2 and apply rule 3). Consider the following example:

Example 3 *Suppose the first few values of* $y = (t_1, y_1, t_2, y_2, t_3, y_3, \ldots)$ *are* $(0, 3, 3, 4, 1, 5, \ldots)$. *Then, the first five successors have been found by:*

1. *Build a successor of length* 3 *since* $y_1 = 3$
2. *A greedy choice*
3. *A greedy choice*
4. *A greedy choice*
5. *Stop allowing greedy choices since* $t_2 = 3$
6. *Build a successor of length* 4 *since* $y_2 = 4$
7. *A greedy choice*
8. *Stop allowing greedy choices since* $t_3 = 1$
9. *Build a successor of length* 5 *since* $y_3 = 5$

In this way, the search procedure used to produce y also dictates exactly when new productions should be created according to y, even if a greedy choice can be applied. Furthermore, the modified search space can be pruned if the end of ρ is reached and not all elements in y have been used up to y_N (other vectors y with different values in the unused parts do not need to be considered). Similarly, if the t values leads to an incompatibility, then certain vectors can be pruned. Since it is easier to prune these values with ES, adjusting PMIT-S0L to remove the prefix limitation requires the use of ES.

Algorithmically, PMIT-S0L has a Boolean control parameter called *prefix limitation* (henceforth, PL) that controls whether it uses this alternate procedure to address the limitation. Where relevant for discussing differences (e.g., Results) PMIT-S0L+PL indicates that the prefix limitation variable it set to `true`, while PMIT-S0L-PL indicates that it is set to `false`.

It is acknowledged that while this technique can address the prefix limitation, it is somewhat inefficient (shown in Section 4.5). Finding a more efficient technique to address this limitation is an area for future investigation.

4.4 Evaluation Methodology

This section describes the experimentation used to evaluate PMIT-S0L at successfully inferring (parsimonious) compatible S0L-systems for one or more sequences of strings. This section starts by discussing the metrics used to measure the success of PMIT-S0L. This is followed by a description of the procedural generation process used to produce the test set.

4.4.1 Performance Metrics

The metrics used to evaluate PMIT-S0L are described next. In these metrics, the *original system* is the hidden L-system that generated the input strings, and the *candidate* is the predicted L-system. While the original L-system can be thought of as the ground truth, it is actually possible to find an L-system that is more likely to generate the input ρ than the original L-system. Therefore, the main goal is to find one with the highest probability of generating ρ rather than the original. This can occur when the original L-system produces ρ by chance that is unlikely for it. As the number of string sequences in ρ increases, the less likely this should be. Hence, the main research question of this work is to investigate the effect of inferring a candidate L-system when multiple sequences of strings are used as input.

Several measures are used to assess accuracy, which is necessary in order to properly capture different ways in which S0L-systems can differ: success rate (SR), mean time to solve (MTTS), weighted true positive - system to candidate (WTP-H2C), weighted true positive - candidate to system (WTP-C2S), probability error (e), maximum successor difference (*diffmax*), and successor difference rate (*diffrate*), which will be described next.

Success for PMIT-S0L is defined as inferring an S0L-system that is compatible with ρ that has equal to or greater probability of producing the sequence(s) than the original system (finding an L-system that is slightly worse than the original would therefore be classified as not successful with this stringent measure). Thus, success rate is the percentage of experiments for a set of control parameters that successfully finds such an S0L-system.

MTTS is the amount of time required by PMIT-S0L to find a candidate system, whether successful or not. Execution time is limited to 12 hours to keep overall experimental times practical (in practice, a user may be willing to wait longer for a result). All timings were captured using a single core of an Intel 4770 @ 3.4 GHz with 12 GB of RAM on Windows 10.

Within the context of this work, a true positive is defined as a successor that is in both the original L-system and the candidate L-system regardless of any difference in their associated probability. One could also certainly define the following: a false positive as a successor that is in the candidate L-system but not in the original, and a false negative is the opposite. It would be possible to simply count the true positives, false positives and false negatives with these definitions. However, there are two issues: first, the terms false positive and false negative implies that original L-system is the better L-system, which might not be the case.

Second, only counting successors ignores the probabilities and successors with a higher associated probability are more important toward reproducing a process *in silico*. Hence, the measures listed next are used instead, and they are weighted to favor those with a higher associated probability.

1. Weighted True Positive - System to Candidate (WTP-S2C) is the sum of associated probabilities for true positive successors in the original system divided by $|V|$.

2. Weighted True Positive - Candidate to System (WTP-C2S) is the sum of associated probabilities for true positive successors in the candidate system divided by $|V|$.

Ideally, the weighted true positive values above should be 1.00 indicating a perfect match. When a match between the hidden and candidate L-systems is imperfect, greater values generally indicate that successors with higher associated probabilities have been matched.

Probability error (e) is calculated by taking each production in the original system; if that production is also in the predicted system, then add the absolute difference of the two probabilities; if the production is not present, then add the probability of it occurring (in the original system). Once all productions have been examined, the total is divided by the number of symbols in the alphabet. Thus, this metric is measuring how different the predicted L-system is from the original even taking into account the differences in probabilities of the rewriting rules. It is an important metric for the accuracy of inferring S0L-systems.

While it is argued that successors with a higher associated probability are more important towards successfully simulating a process, the measures maximum successor difference (*diffmax*), and successor difference rate (*diffrate*) provide an alternative viewpoint to the weighted metrics above on how well the candidate and hidden system match. The maximum successor difference is the largest count of the number of successors that are in the hidden L-system but are not in the candidate L-system across the experiments. While this could be averaged over the number of productions, the intent of this measure is to examine if the number of extra or missing successors varies with M (as opposed to by number of productions and M). The successor difference rate is the percentage of experiments where the successor difference was not zero.

4.4.2 Data

Since there are very few published S0L-systems in the literature, the test cases for PMIT-S0L are mainly procedurally generated, using the following procedure. A statistical analysis was done of known L-systems to find realistic distributions for the successor rule length, the number of distinct letters in successors, and the number of successors per symbol (using parametric L-systems). The distribution is created by counting the number of times a value is found (for each metric listed in the preceding sentence), and dividing it by an appropriate divisor. The total number of successors examined is used as the divisor for computing the distributions for the successor rule length and the number of distinct letters in the successors. For the number of successors per symbol, the divisor is the total number of symbols examined in the analysis set. For example, the percent chance of choosing a successor length of 5 is found by counting the number

of times a successor has a length of 5 in the analysis set and dividing by the total number of successors examined in the analysis set. This analysis was done to ensure that the generated L-systems have distributions similar to known L-systems. The statistical analysis was done using L-systems found in the University of Calgary's virtual laboratory [77]. Graphical symbols (that are part of the turtle interpretation) are not used explicitly. Therefore, our experimental results are reflective of the symbols being totally arbitrary without any assumptions on the alphabet.

Alphabet size is not explicitly picked, but instead is based on a value S that is the total number of successors. Symbols are then iteratively added from $(A, B, C, \ldots, K)$ with a randomly selected number of successors for each symbol added until S is reached. Each symbol has a 50% chance of having 1 successor, a 40% chance to have 2 successors, and a 10% chance to have 3 successors. If a number of successors for a symbol is selected such that S would be exceeded, then it is reduced to ensure that this does not happen. It is possible by chance that all symbols would have 1 successor, which would be a deterministic L-system. If this occurs, then two symbols from V are picked randomly; e.g. A and B. The symbol A is given another successor, and B is removed ensuring the total number of successors remains correct, and that all produced systems are stochastic, but not deterministic.

When a symbol has 2 or more successors, the associated probability (p) for each successor is found by iterating over the number of successors, and selecting a random value between an upper u and lower bound l, which are programming variables. The upper and lower bounds are initialized such that $u := 100\% - (n-1)$ and $l := 1\%$. After each iteration, $u := u - p$. The last successor has $p := u$. The next step is to construct a word for each successor over V.

To determine the length of the successor, the following probability distribution is used to determine each successor's length (expressed as p - l, where p is the chance the successor has length l): 4% - 1, 16% - 2, 20% - 3, 20% - 4,20% - 5, 16% - 6, 4% - 7, 2% - 1, 1% - 1, and 1% - 10. Finally, the probability distribution for the number of distinct symbols is evenly divided from 1 to 5 (20% chance each). Random symbols are then selected from V until the successor length is reached.

Using the procedure above, three data sets are generated to evaluate PMIT-S0L.

The first and second data set are called DS_{PL} (data set prefix limitation) and DS_{nPL} (data set no prefix limitation) respectively and both enforce that $M = 1$ to isolate any effects from higher values of M. DS_{PL} furthermore enforces that all produced systems have no cases where $A \rightarrow uv$ and $A \rightarrow u$ for any $A \in V$, where u and v are words over V; i.e., no common prefix for two successors of a symbol A. DS_{nPL} has no restriction on prefixes. The intent of these data sets is to allow an evaluation of the effect of addressing the prefix limitation, so when PMIT-S0L is executed on DS_{PL} the vector y produced contains only successor length values. For DS_{PL}, S was iterated from 3 to 10, 60 L-systems were generated for each value of S, and each experiment was conducted twice. For DS_{nPL}, S was iterated from 3 to 9 (only because it was clear that PMIT-S0L would timeout with $S = 10$), 60 L-systems were generated for each value of S, and each experiment was conducted twice.

To test the effect of using different numbers of sequences of strings M, an S0L-system G is generated using the process described above with the following parameters. S (total number of successors) is selected as a random value from 3 to 9. The lower bound of 3 is the lowest possible value for and S0L-system with $|V| \geq 2$, and $|V| = 1$ is not considered as it is uninteresting. The upper bound was determined by the experiments with SD_{nPL} described above. For each generated S0L-system, it generates input sequences for each M iterated from 1 to 10. For any single experiment, when $M > 1$, the exact same sequence of strings is not permitted to be generated; if this occurs, a new sequence of strings is generated until it differs from all of the sequences produced so far. Sixty S0L-systems were generated in this fashion, and for each combination of G and M, the experiment was done twice (for a total of $1,200$ experiments). The dataset for these experiments is denoted as DS_{vM} (data set varying M).

4.5 Results and Discussion

This section provides the results of the evaluation of PMIT-S0L. First, the evaluation of PMIT-S0L on the procedurally generated S0L-systems (the main result) is discussed, including the effects of addressing the so-called *prefix limitation* (previously described in Section 4.3.4). Afterwards, observations regarding the inference of the L-system for Japanese Cypress trees are described. The variations of PMIT-S0L executing with and without the prefix limitation process are denoted as PMIT-S0L+PL and PMIT-S0L-PL respectively when needed for clarity.

A preliminary investigation was done using only greedy algorithm, then random forest, before building a hybrid greedy algorithm with a simple genetic algorithm (SGA), and exhaustive search (ES). As they were preliminary, the results of these initial experiments can be found in Table A2; however, in summary both the greedy-only algorithm and the random forest were found to be ineffective relative to the hybrid algorithm. Therefore, the remainder of this section focuses on the hybrid algorithms.

It was found that PMIT-S0L-PL is able to reliably infer S0L-systems with up to 10 successors when $M = 1$ (shown in Figure 4.2). However, some minor variations were noticed between the candidate solutions and the original S0L-systems shown on the two WTP lines. While SGA was not as successful as exhaustive search, it is much faster, peaking at about 2 minutes for 10 successors (see Table A1, time for ES described below). A fitness landscape analysis was completed; however, there were no characteristics to the search space that suggested a particular way forward. Thus, an avenue for future investigation is to investigate the search space more deeply to find ways to avoid the use of ES for reasons of performance.

The effect on MTTS on inferring S0L-systems with and without the prefix limitation process enabled is shown Figure 4.3 (the other metrics were not significantly effected so these are included in the Table A3). It shows that PMIT-S0L+PL is slower than the other variants. It is difficult to objectively say whether the time increase is worth the ability to infer S0L-systems where a symbol A has a successor that is a prefix of another, as it is unclear how often this occurs in practice. Subjectively, considering the possibility that the

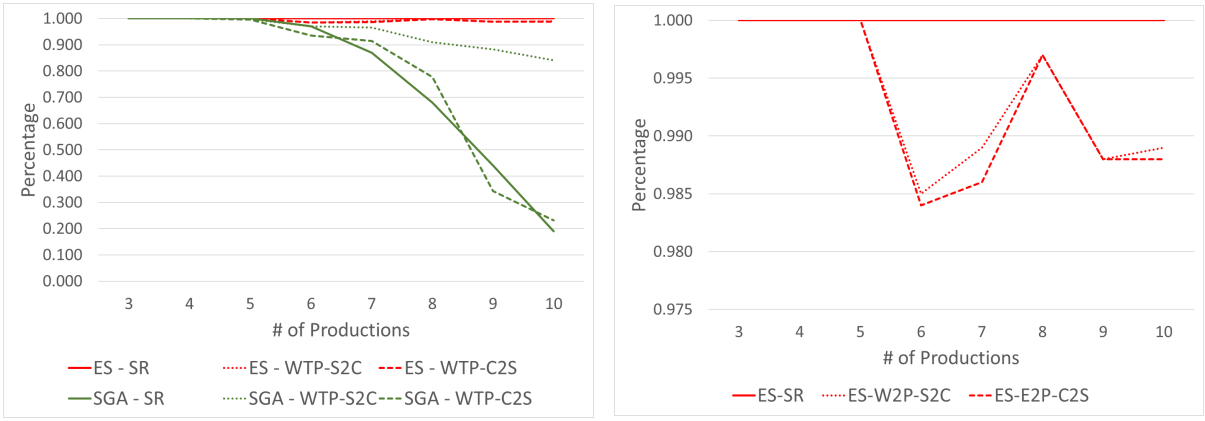


Figure 4.2: On the left, the measures of accuracy SR, WT2-S2C, WTP-2CS for PMIT-S0L-PL using ES (red) and SGA (green) with data set DS_{PL} are plotted against the number of productions. As it is difficult to see the three lines for ES, a zoomed view is shown on the right.

encoding scheme described herein to address the prefix limitation had the potential to be extremely large, the time increase seems reasonable. This suggests that, at least for the procedurally generated data sets, the method for removing the prefix limitation with pruning is effective.

The main research goal of this paper is investigating the effect of inferring an S0L-system when using various numbers of sequences of strings as an input. Figure 4.4 shows how the probability error, WTP-C2S and WTP-S2C change as M increases. This raw data is shown in Table 4.3, which also shows the mean time to solve, the maximum successor difference, and successor difference rate. SR is not shown as it was always 100%, for each value of M with PMIT-S0L+PL and dataset DS_{vM}. It can been seen that from $M \geq 3$, PMIT-S0L+PL infers the original system for all test cases, as the WTP values are all 1.000. With respect to error in the probability distribution, this becomes subjectively reasonable at $M = 3$ with 6.3% average error. Error seems to be close to minimal at $M = 6$ (at approximately 1%), although it continues to decline at a small rate as M increases. Increasing M has a negligible effect on MTTS since ρ_2 to ρ_M are only scanned when an S0L-system has been found to be compatible with ρ_1, and this is similar to the scanning process which takes sub-millisecond time in all tests cases. So, adding additional sequences of strings is generally beneficial in practice.

With respect to differences between the candidate system and the original system, it can be seen that when $M = 1$ there can be up to two successor differences. These non-matching productions tend to be for low probability successors as can be seen from the values for WTP. When $M = 2$, there was a single instance where PMIT-S0L+PL did not exactly match the original system. Again, the difference was a very low probability successor. In examining this experiment more closely, it was found that the successor in the original system had only been selected once and the symbols lined up in such a fashion that the candidate system found by PMIT-S0L+PL was reasonable given the data. Still, overall, in practice it would be recommended to have at least 3 sequences of strings to infer an S0L-system that closely models an underlying process reliably.

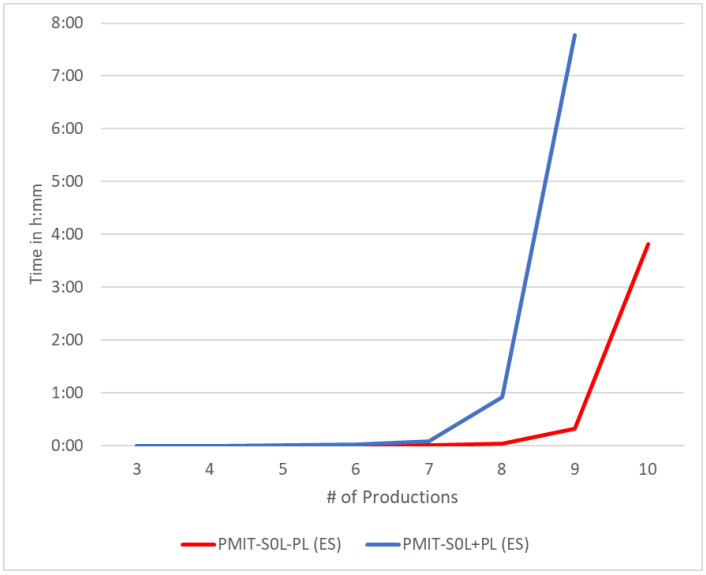


Figure 4.3: A comparison of MTTS versus the number of productions for PMIT-S0L-PL (red) and PMIT-S0L+PL (blue).

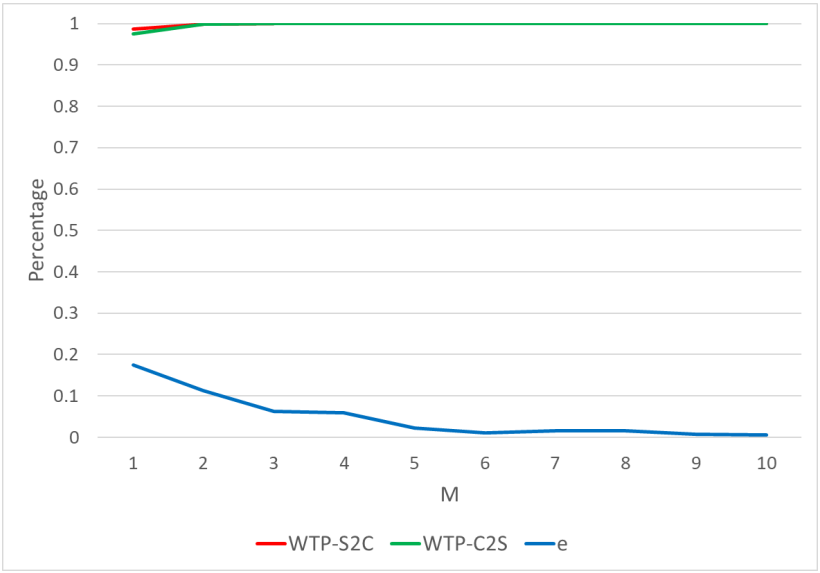


Figure 4.4: The measures of accuracy WTP-C2S, WTP-S2C and e plotted against different values of M for PMIT-S0L+PL.

M	WTP-S2C	WTP-C2S	e	*diffmax*	*diffrate*	MTTS
1	0.987	0.975	0.174	2	20%	03:48.228
2	0.998	0.999	0.113	1	5%	01:35.192
3	1.000	1.000	0.063	0	0%	01:30.821
4	1.000	1.000	0.060	0	0%	03:55.821
5	1.000	1.000	0.023	0	0%	02:30.821
6	1.000	1.000	0.010	0	0%	04:00.821
7	1.000	1.000	0.016	0	0%	02:16.131
8	1.000	1.000	0.016	0	0%	02:16.131
9	1.000	1.000	0.008	0	0%	02:46.496
10	1.000	1.000	0.006	0	0%	03:00.065

Table 4.3: Performance metrics for PMIT-S0L+PL with M sequences of strings from 1 to 10, $N = S$, and the dataset DS_{vM}. SR = 100% for all experiments and is therefore not in the table.

With respect to inferring the S0L-system found by Nishida [53] for modeling Japanese Cypress, PMIT-S0L+PL was not able to infer it in a practical amount of time. The first experiment was to set $N = 42$, which is much greater than the approximate maximum of 9 successors in the other experiments. After several hours, this was terminated and it was estimated that PMIT-S0L+PL would take at least 10^9 hours to complete using exhaustive search in a sequential fashion.

4.6 Conclusions and Future Directions

This paper presents an investigation into inferring stochastic context-free L-systems (S0L-systems) when using different numbers of sequences of strings with the Plant Model Inference Tool for S0L-systems (PMIT-S0L). PMIT-S0L is a generalized algorithm for inferring S0L-system and requires no *a priori* scientific knowledge when compared to existing approaches for inferring S0L-systems algorithmically (e.g. [20, 62, 68]). Being generalized means that PMIT-S0L may be used for any problem as opposed to requiring a specific algorithm for each individual problem in a specific research domain. PMIT-S0L opens up the possibility of inferring S0L-systems in research domains where none have been found to date.

PMIT-S0L is primarily evaluated on procedurally generated S0L-systems due a shortage of specific systems published in the literature. An analysis was done of existing L-systems to create realistic procedural generation rules. Three data sets, for a total of 960 L-systems (a total of 3,000 experiments were conducted across these systems), were generated to evaluate different aspects of PMIT-S0L.

PMIT-S0L has two different modes of operation controlled by a *prefix limitation* Boolean parameter (denoted as PMIT-S0L+PL and PMIT-S0L-PL). The *prefix limitation* parameter controls if it searches all L-systems, or only those without multiple successors of the same letter, where one is a prefix of the

other. The results show PMIT-S0L-PL is quite a bit faster than PMIT-S0L+PL. For an S0L-system with 9 successors, PMIT-S0L-PL will succeed on average in about 20 minutes, while PMIT-S0L+PL takes several hours. However, PMIT-S0L+PL is still practical considering the effort required to infer an S0L-system by hand.

PMIT-S0L-PL was found be an extremely accurate tool for inferring a compatible S0L-system that is at least as probable as the original system. This paper shows that PMIT-S0L+PL was always able to find the original system when at least three sequences of strings were used as inputs. Additionally, the evaluation showed that when six sequences were used, the error in the solution's probability distribution compared to the original system becomes low (about 1% or less), and there is not much improvement for adding additional sequences of strings. Therefore, it is recommended that in practice at least 3 sequences of strings should be used to infer S0L-systems, but 6 or more is ideal to minimize the error in probabilities. Finally, there is essentially no penalty to execution time for adding additional sequences of strings, so there is little reason to avoid using all of the string sequences that are available.

PMIT-S0L (in either mode) can be used with an exhaustive search, which is not an efficient searching algorithm. However, with normal branch-and-bound pruning, it could infer all L-systems in a test suite of 420 L-systems with at most 9 productions in about 8 hours. Genetic algorithm was also evaluated, and was much faster; however, it was less accurate.

There are several directions that can be taken with this research in the future. Perhaps most importantly comes from applying this research to the practical problem of inferring L-systems from segmented images. De La Higuera [22] argues that to be realistic, L-system inference algorithms should handle errors in the strings; e.g., insertions or deletions of symbols. These errors can be, at least initially, considered stochastic productions; i.e., if A has the production $A \rightarrow xyz$, but periodically is subject to a deletion error such that $A \rightarrow xz$, then this can be thought of as a stochastic production. With a set of stochastic productions inferred, it is then a matter of detecting and fixing any errors.

Additionally, parallelism will be used to speed up PMIT-S0L; however, it still would be useful to investigate more efficient searching techniques to allow PMIT-S0L to infer S0L-systems with larger number of productions at practical speeds. Additionally, the main issue with using PMIT-S0L in a practical fashion is the need to select a reasonable value for the size of the vector used for searching (which roughly corresponds to the number of productions). An algorithm that does not require this parameter would be ideal, or alternatively finding a good way to compute or estimate the vector size.

4.7 Acknowledgments

This research was undertaken thanks in part to funding from the Canada First Research Excellence Fund, National Science Engineering Research Council grant #2016-06172, and the Alexander Graham Bell scholarship for Jason Bernard. Additionally, the authors would like to thank Dr. Farhad Maleki for his assistance

in acquiring the images for this paper.

Appendix A

A1 Different Techniques for Inferring Stochastic L-systems

With DS_{PL} and PMIT-S0L-PL, a comparison of success rate of these four broad approaches is shown in Table A1. The comparison reveals that greedy algorithm, random forest, and the hybrid with SGA were not as effective as a hybrid with exhaustive search at inferring S0L-systems; hence, justifying the decision to only the two hybrid algorithms to investigate the research question of this paper.

#Succ	*Greedy Algorithm*	*Random Forest*	*Greedy* +SGA	*Greedy* +ES
3	22%	52%	100%	100%
4	16%	35%	100%	100%
5	3%	11%	100%	100%
6	0%	5%	97%	100%
7	0%	2%	87%	100%
8	0%	0%	68%	100%
9	0%	0%	44%	100%
10	0%	0%	19%	100%

Table A1: Success rate comparison between a greedy algorithm, random forest, hybrid greedy algorithm with simple genetic algorithm and hybrid greedy algorithm with exhaustive search, $M = 1$, $N = S$, and the DS_{PL}

A2 Inferring Stochastic L-systems with Greedy Algorithm Hybridized with a Simple Genetic Algorithm

It is found that PMIT-S0L using a SGA as the search technique is not as successful as PMIT-S0L using ES; however, it is much faster.

A3 Predicting Values Lower than the Actual Number of Successors

The effects of allowing for a two or more successors of a symbol A to have a common prefix has little effect on performance metrics except for MTTS.

$\#Successors$	SR	WTP-S2C	WTP-C2S	MTTS
3	100%	1.000	1.000	0:00:00.306
4	100%	1.000	1.000	0:00:00.400
5	100%	0.997	0.996	0:00:00.486
6	97%	0.971	0.936	0:00:01.206
7	87%	0.965	0.915	0:00:04.069
8	68%	0.909	0.778	0:00:09.598
9	44%	0.882	0.343	0:00:54.388
10	19%	0.841	0.232	0:02:04.130

Table A2: Performance metrics for PMIT-S0L-PL when using genetic algorithm, $M = 1$, $N = S$, DS_{PL}

$\#Successors$	SR	WTP - S2C	WTP - C2S	MTTS
3	100%	0.973	0.947	0:00:00.085
4	100%	0.971	0.981	0:00:00.557
5	100%	0.929	0.856	0:00:08.505
6	100%	0.923	0.906	0:01:20.776
7	100%	0.967	0.941	0:04:55.114
8	100%	0.910	0.952	0:55:10.900
9	100%	0.974	0.973	7:46:16:355

Table A3: Success rate and successor existence comparisons for PMIT-S0L+PL when inferring an S0L-system where $A \to uv$ and $A \to u$ for some $A \in V$, $M = 1$, $N = S$, DS_{PL}

Chapter 5

Inferring Temporal Parametric L-systems Using Cartesian Genetic Programming

Abstract

Lindenmayer Systems (L-systems) are formal grammars that use rewriting rules to replace, in parallel, every symbol in a string with a replacement string, and this procedure iterates. This produces a sequence of strings whose symbols may be interpreted as simulation instructions. This has been found to be useful for modeling natural temporal processes. There are many different types of L-systems, with parametric L-systems being among the most useful for creating simulations as they allow the mechanisms (represented by rewriting rules) to change based on different influences as the parameters change, such as environmental factors. Typically, L-systems are found by experts based on taking precise measurements and using a priori knowledge. As a step toward reducing the effort around finding an appropriate L-system, this paper presents the Plant Model Inference Tool for Parametric L-systems (PMIT-PARAM), an algorithm for automatically learning parametric L-systems from a sequence of strings generated, where at least one parameter represents time. PMIT-PARAM is evaluated on a test suite of 20 known parametric L-systems, and is found to be able to infer the correct rewriting rules for the 18 L-systems containing only non-erasing productions; however, it can find appropriate parametric equations for all 20 *of the L-systems. Inferring L-systems algorithmically not only can automatically learn models and simulations of a process with potentially less effort than doing so by hand, but it may also help reveal the scientific principles governing how the process' mechanisms change over time.*

5.1 Introduction

Lindenmayer systems (L-systems) [41] are formal grammars consisting of an alphabet V, an axiom w (a starting word over V), and a finite set of rewriting rules P (also called productions). While the process for applying the productions is described in greater detail below, in short all letters of a string are rewritten in parallel, and the rewriting is applied to the strings iteratively starting with w, and so a sequence of strings is produced. Commonly, the letters in the alphabet are given an associated interpretation for a simulator; e.g., draw a line, or turn left by 30° degrees, etc., in either 2D or 3D as described in [62]. In this way, a

string can describe a fixed image, and a sequence of strings can reproduce a temporal process by a simulator. Hence, an L-system that produces the sequence of strings can produce the simulation, and the L-system is a representation of the program controlling the model.

The main goal is to automatically infer an L-system from data; i.e., to learn the simulator program. It is often possible to acquire a sequence of images over time of a plant growing [72]. One approach would be to divide inference into two steps. The first would be to segment the images into corresponding strings that would approximately draw them with a simulator [21]. The second is to infer the L-system from the strings. This last step is known as inductive inference and is the focus of this research.

The challenge is therefore to find an L-system that produces the strings that simulates a desired process. Usually this is done by experts via possibly time consuming measurements and *a priori* knowledge (e.g. from developmental biology for biological models) [64]. Such models can be successful at simulating natural processes; for example, Figures 5.1 and 5.2 show a visualization from a simulation of *Helianthus annuus*, and *Mycelis muralis*, that were made using parametric L-systems created in the "virtual laboratory" [77]. Our goal is to do this automatically. This can possibly save much effort, and may even help reveal principles underlying the process; as stated by Godin and Ferraro "as a byproduct of such an analysis, we show that the method enables us to identify putative hierarchies of meristem states that could be further exploited in combination with investigations at a biomolecular level to better understand plant development." [27].

Figure 5.1: Visual simulation of *Helianthus annuus* as produced by vlab [62]. Image used with permission of the copyright holder.

L-systems can take several different forms typically by changing the method used to select a rewriting rule for an instance of a symbol. The simplest form of an L-system is a deterministic context-free L-system (D0L-system), which has rewriting rules of the form $A \to x$, where $A \in V$ is called the predecessor and $x \in V^*$ (V^* is the set of all strings over V) is called the successor. For each letter $A \in V$, there is exactly one rule in P with the letter as predecessor, and its successor is denoted by $succ(A)$. In a deterministic context-sensitive L-system ((k,l)-system), the selection of a successor depends on the surrounding k symbols to the left, and the

Figure 5.2: Visual simulation of the growth of *Mycelis muralis* as produced by vlab [62]. Image used with permission of the copyright holder.

l symbols to the right of the letter in the string. In this case, the productions have the form $u < A > v \rightarrow x$ where $u, v, x \in V^*$, and $|u| \leq k$ and $|v| \leq l$. As the L-system is deterministic, there is still only one successor possible for any given A in a string. A stochastic context-free (or context-sensitive) L-system allows for each A (and u, v for context-sensitive L-systems) to have one or more successors. Each successor has an associated probability of being selected such that the sum of the probabilities for a combination of symbol plus context equals 100%. Finally, a parametric L-system adds a set of parameters to each rewriting rule, and a Boolean condition is evaluated based on any associated parameters. Although, we will only consider parametric L-systems that are deterministic in this paper, there can exist some $A \in V$ with more than one rewriting rule (or A, u, v if it is context-sensitive), but the one selected for any instance of a symbol (and context) is the one for which the associated Boolean condition is true (if none are true, then an identity production, $A \rightarrow A$, is usually assumed). For example, there could be an axiom $A(1)$ and two rewriting rules with a parameter t as follows: $A(t) : t \leq 5 \rightarrow A(t+1)$ and $A(t) : t > 5 \rightarrow B(t+1)$. This can be interpreted as A rewriting to itself for the first 5 derivation steps while incrementing the parameter before rewriting to a B. One common use of parameters is to describe time, as t is being used above. It is indeed quite natural to use a parameter for time for temporal processes. Although other uses for parameters are possible as well, time-based rules are the focus of this research as they are frequently used with parametric (deterministic) L-systems in the literature [62]. This also allows for the construction of a test suite of known L-systems to test successfulness of the inference procedure on existing L-systems that were constructed for some practical purpose.

Applying the rewriting rules to each letter of a string is called a derivation step, and this proceeds iteratively in a similar fashion regardless of the type of L-system, differing only in how a successor is selected. A derivation step on a string $s = a_1 a_2 \cdots a_n$, replaces each letter a_i in parallel with a selected successor x_i (determined as described above, either deterministically, stochastically or based on the associated parametric values), denoted by $a_1 a_2 \cdots a_n \Rightarrow x_1 x_2 \cdots x_n$. Starting with the axiom w, if an $n-1$ step derivation is applied

$w = w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$, then $\rho = (w_1, \ldots, w_n)$ is said to be a length-n developmental sequence. For a deterministic system, this sequence is unique.

This research focuses on inferring parametric L-systems, that are deterministic but may be context-free or context-sensitive with one symbol for the left and right context (the most commonly used context length). Towards this goal, this paper introduces the Plant Model Inference Tool for Parametric L-systems (PMIT-PARAM) to infer an L-system including rewriting rules together with Boolean conditions, from a sequence of strings ρ as input, where ρ is the developmental sequence of the predicted L-system. The set of parameters and the context lengths for each symbol in the alphabet are also provided as input. In practice, the context lengths would not be known; however, providing them allows the research to focus exclusively on the problem of inferring a parametric L-system, and not also on the issue of inferring context lengths. The algorithm could be easily extended by techniques provided in [45] to search for the proper context lengths, which would increase execution time.

The method used by PMIT-PARAM is divided into two main steps: 1) to find an intermediate stochastic L-system (SL-system) that ignores the Boolean conditions and parameters, but that generates the input strings, and 2) convert that system into a parametric L-system with deterministic rules. The first step makes use of the algorithm presented in [12] (an extended version of [10], and described in Section 5.3) that attempts to return an intermediate SL-system with the greatest probability of producing ρ (this has been argued as being the best choice to produce a sequence of strings from a hidden L-system [12]). Cartesian Genetic Programming (CGP) [46] is then used to find the Boolean conditions for each rewriting rule in the SL-system. CGP has previously been found to be a good algorithm for finding a set of equations to satisfy data sets, and it supports finding Boolean expressions (such as those used by parametric L-systems) without modifying the algorithm [46].

PMIT-PARAM is evaluated using 20 known parametric L-systems, some context-free and some context-sensitive. Each L-system is used to generate a developmental sequence, and the algorithm attempts to infer the L-system from only the inputs — i.e., without any domain knowledge — thereby, making PMIT-PARAM domain agnostic (while mainly motivated for simulating plants). PMIT-PARAM was found to successfully infer the set of successors for 18 of the 20 systems, while PMIT-PARAM was 100% successful at inferring appropriate Boolean conditions for all 20 systems. Thus, it found a correct L-system in 18 of the cases, which essentially means it was able to determine the simulation programs from data.

This paper is formatted as follows. The next section briefly discusses the state of the literature on inferring L-systems, and describes CGP. Section 5.3 discusses how PMIT-PARAM infers the successors, and details how PMIT-PARAM finds appropriate Boolean conditions from the intermediate inferred stochastic successors. Section 5.4 describes the performance metrics and data used to evaluate PMIT-PARAM. Section 5.5 provides the results of the evaluations, and discusses observations made during and from the evaluation. Section 5.6 concludes the work and discusses future directions for this research.

5.2 Background

There has been several works on inferring the simplest type of L-systems, D0L-systems [9,23,52,71]. A survey for some of these and other approaches, which use string or other types of data as input, can be found in a survey by Ben-Naoum [7]. Existing algorithmic approaches for inferring L-systems have focused mainly on two methods. One approach is to infer the L-system algebraically to determine relationships between the strings in the sequence [9, 23, 52]. The other approach is to search a space of L-systems using, for example, genetic algorithm [9,71]. Since stochastic L-system inference is so fundamental to this research it is discussed in greater detail in Section 5.3 below.

Only a few papers could be found in the literature on the practice of inferring some parts of parametric L-systems semi-autonomously (e.g., [18, 73, 74]), which is likely due to the fact that they are significantly more complex than other types — yet widely used in practice. In [18] uses genetic programming to infer parameters for the length and angles of segments. For example, such parameters can be used for statements such as $+(45)F(3)$ that means to turn 45° left and then draw a line of 3 units long (using parameters in this fashion is commonplace [62]). In their work, the L-system's successors and Boolean conditions are assumed to be already known. For example, in the rewriting rule

$$A(s,q) : s \geq MINSIZE \rightarrow !(q)F(s)!(q \cdot b);$$
$$[+(ANG1)/(DIV1)A(0.5 + RATE1, qr^e)]$$
$$[+(ANG2)/(DIV2)A(0.5 + RATE2, q(1-r)^e)$$

their algorithm finds reasonable values for $MINSIZE$, $ANG1$, $ANG2$, $RATE1$, $RATE2$, $DIV1$, and $DIV2$ [18] to produce the imagery desired by the user.

It presents the user with a small visualized population of L-systems. The user selects two members based on aesthetics, and a new solution is evolved using well-known genetic operators (e.g., crossover and mutation). The user repeats the process as desired until a suitable solution is found. The algorithm was developed as an aide for refining an L-system, and it requires the productions to be developed by hand. Our contribution to the study of parametric L-system inference is that PMIT-PARAM not only infers parametric values, but also the successors and associated Boolean conditions, and there is not a manual aesthetic step.

Štava et al. [74] investigated an alternative approach to evolving parametric settings. In their work, they use Monte Carlo Markov Chains to search a space defining parameter values for a known parametric context-free L-system. The similarity of the resulting imagery is done by segmenting the original and produced images, and converting them into an acyclical graph. The graphs may then be compared providing a "distance" measure as reasonable measure of similarity of the images (this technique was first proposed by Godin and Ferraro [27], although the authors suggest some extensions). Their work was evaluated on 3D tree images. They found their approach could produce suitable parameters to approximate, but not exactly match, the

input image in 12 to 270 minutes depending on the number of nodes in the resulting graph.

5.2.1 Cartesian Genetic Programming

The following background on CGP is as described by Miller [46]. CGP originated as a technique for evolving circuit designs by using a grid of nodes to represent logical gates. A search component, for example the 1+4 evolutionary strategy [66], would modify the types of gates, the connections between the inputs, gates, and outputs. CGP uses a genome structure consisting of a list of function genes and output genes. Each function gene is made of a list of integers representing the data addresses for the function inputs (this could be input data or the output of another function), and an integer representing the function from the list of permitted functions. Later, it was found that CGP could be used to find a set of equations by modifying the logical gates to mathematical operators. In [62], it recommends that the Boolean conditions of parametric L-systems may consist of "numeric constants, combined using the arithmetic operators ... exponentiation operator ... relational operators ... logical operators, and parentheses". Thus, for this research the following operators are mapped from integral values for the function genes: $+$, $-$, $\times$, $\div$, $\hat{}$, $\neg$, $\wedge$, $\vee$, $=$, $\leq$, $<$, $\geq$, $>$. For this research, the function arity is set to two (as recommended by [46]), so each function gene consists of three integers total. The upper and lower bound of the data address genes are computed such that a cyclic graph may not be produced. The output genes consist of a single integer which is a data address, which again may be input data or an output from a function node. This configuration is called the "CGP General Form". A population of genomes is evolved using evolutionary operators (1+4 evolutionary strategy is recommended by Miller [46] and is used for this research), and the resulting expression is evaluated using a problem specific fitness function. Although both crossover and mutation operators may be used with CGP, it is recommended to use mutation only [28], which is where a subset of one or more genes are randomly modified to different permitted values.

Of note, tracing a path from the output(s) to the input(s) will not pass through every function node (indeed, typically less than 5% of nodes are used [46]). Thus, many of the function genes are so-called *non-coding genes*. This allows the genome length to be very long without adding too much difficulty in finding a suitable solution; however, it has been found that longer genome lengths result in redundant operations. This occurs because a simple mutation operator is likely to only mutate non-coding genes, and thus a child decodes to the same expression as the parent. The recommended method to resolve this is Goldman mutation [28], which mutates genes at random until at least one known coding gene is mutated.

5.3 Inferring Parametric L-Systems

This section describes how PMIT-PARAM infers a parametric L-system, which is fully automatic and infers the full L-system including productions and conditions from a developmental sequence. First, PMIT-PARAM infers the successors without parameters by treating the input strings as if they were produced by an SL-

system. After, a set of Boolean conditions for each $A \in V$ that has two or more successors identified by the first step is determined and used to replace the associated probabilities on the rewriting rules. In all cases, it is assumed that no production rewrites to the empty word; i.e., they are non-erasing. A preliminary investigation into changing PMIT-PARAM to consider erasing productions found it would be significant and challenging issue.

5.3.1 Inferring the Successors

Consider the case where there exists a hidden parametric L-system $G_{hidden} = (V, w, P_{hidden})$ that has produced a sequence of strings $\rho = (w_1, \ldots, w_n)$. We are only going to consider cases where at least one parameter represents time, and where the parameters are actually used; therefore, within ρ, there will be at least two instances of some symbol $A \in V$ that produce a different successor.

The inference problem we are trying to solve considers ρ as input, and it will attempt to infer an L-system $G_{predicted}$ that gives ρ as a developmental sequence (which may or may not match G_{hidden}). The parametric data is assumed to be part of the input. For example, $F(3)$ might appear in the developmental sequence and the 3 value is present in it. If one thinks of segmenting images into strings, graphical parameter values might be evident, as might some time-based parameters, but for some types of parameters, it is less obvious. But this assumption is made to keep the research focused on parametric L-system inference, and not on the difficulties in capturing parametric data. A future investigation should examine the effects where the parametric data is incomplete, or has errors, etc.

Suppose first that the parametric data is ignored; i.e., let $\bar{\rho}$ be obtained from ρ by removing all parameter values, and let $\bar{\rho} = (\bar{w}_1, \ldots, \bar{w}_n)$. If two occurrences of A in ρ are rewritten differently, then it would appear that A is stochastically switching productions to produce $\bar{\rho}$. Thus, PMIT-PARAM starts by inferring a stochastic L-system $G_s = (V, \bar{w}, P_s)$ that produces $\bar{\rho}$ within the same limits found in [12]. If the successors in P_{hidden} are the same (without the parameters and conditions) as those in P_s, then it may be possible to convert from stochastic to parametric productions by replacing the associated probabilities on each production with appropriate parameters and with a Boolean condition. Note, if the successors are not the same between G_{hidden} and G_s, then it is still possible the final predicted parametric L-system will produce ρ, although $G_{predicted}$ would be different than G_{hidden}. The process for finding an intermediate SL-system is briefly described as follows from [12].

To begin, for a given $\bar{\rho}$ produced by a hidden L-system, in the deterministic case there may be several L-systems that can be inferred, although quite often there is only one possible solution. In the stochastic case, there exists a multitude of SL-systems that can produce $\bar{\rho}$. Consider the case of a pair of words $\bar{w}_i = a_1 a_2 \cdots a_j$ and $\bar{w}_{i+1} = b_1 b_2 \cdots b_m$, where $\bar{w}_i \Rightarrow \bar{w}_{i+1}$. It is possible that a_1 produces all but $m-1$ symbols in $\bar{w}_{i+1}$, and a_2 through a_j produce one symbol each. It is also possible that a_1 produces all but $m-2$ symbols, and a_2 produces two symbols, and the remainder each produce one symbol, and so on. What separates each of these candidate solutions is the probability that they would produce the developmental

sequence. From the multitude of solutions, it is argued that the best solution is the SL-system that has the greatest probability of producing $\bar{\rho}$ [12].

The probability that a candidate L-system produced $\bar{\rho}$ may be computed as the product of the individual events, which for an SL-system is the product of the probabilities of the successors being selected. For example, if $\bar{\rho} = (\bar{w}_1, \bar{w}_2)$, $\bar{w}_1 = ABBA$, $\bar{w}_2 = AABBCCDD$, where the productions are $A \rightarrow AA : 0.75$, $A \rightarrow DD : 0.25$, $B \rightarrow BB : 0.9$, and $B \rightarrow CC : 0.1$, then the probability that this L-system produced $\bar{\rho}$ is $0.75 \cdot 0.9 \cdot 0.1 \cdot 0.25 = 0.0169$ (1.69%). However, with alternate probabilities on the rewriting rules $A \rightarrow AA : 0.5$, $A \rightarrow DD : 0.5$, $B \rightarrow BB : 0.5$, and $B \rightarrow CC : 0.5$, the probability rises to $0.5 \cdot 0.5 \cdot 0.5 \cdot 0.5 = 0.0625$ (6.25%), and so of the two, this SL-system would be preferred as a solution.

With the intent to find the L-system with the greatest probability, a greedy algorithm is used to prefer successors of a letter that have already been chosen. This is done by maintaining a list of successors for each $A \in V$ and picking a successor from the list that maximizes the local probability that the resulting SL-system produces $\bar{\rho}$. So as $\bar{\rho}$ is scanned symbol-by-symbol starting with the first word, the following process is followed. Let $\bar{w}_i = a_1 a_2 \ldots a_j$, $\bar{w}_{i+1} = b_1 b_2 \ldots b_m$; then a successor is selected from the list of successors for a_1 if it matches $b_1 \ldots b_q$ where q is the length of the successor of a_1. This is repeated for a_2 starting from position $q + 1$ of $\bar{w}_{i+1}$, and so on.

While this works after the lists are already populated with the successors, in practice, especially for the first few times a symbol A is encountered, there will be no successor in the corresponding list to select. Such a scenario can be resolved if the greedy algorithm is hybridized with a search algorithm that produces a list of successor lengths from which to select when a greedy choice could not be made. For example, suppose that $\bar{w}_i = a_1 a_2 \ldots a_j$, $\bar{w}_{i+1} = b_1 b_2 \ldots b_m$, and a_1 had never before been encountered (or no successor in the list matched the starting symbols in $\bar{w}_{i+1}$). Furthermore, suppose that the search algorithm has produced a vector of integer successor lengths (x, y, z) from which no values had yet been selected. In this case, the successor of a_1 would be assumed to be the first x symbols from $\bar{w}_{i+1}$, i.e. $b_1 \ldots b_x$, and this would be added to the list of successors for a_1. The vector of integers is found by trying many combinations using an exhaustive search algorithm with branch-and-bound pruning to reduce the search space size (detailed in [12]). The probability of it generating $\bar{\rho}$ is used as the fitness value.

5.3.2 Parametric Conditions

To convert the SL-system to a parametric L-system $G_{predicted} = (V, w, P_{predicted})$ (with deterministic context-free or context-sensitive rules), the probabilities associated with each successor must be replaced with an appropriate Boolean condition. For each A, denote the productions from A in $P_{predicted}$ as follows, where S_A is the number of productions from A:

$$A(X_i) : \Theta_i \rightarrow succ(A)_i$$

for i from 1 to S_A, where X_i are the parameters and Θ_i are the Boolean conditions. The Boolean conditions in $P_{predicted}$ are considered *appropriate* if for each $A \in V$, and for each value of i from 1 to S_A, every instance

of A in ρ where $A \rightarrow succ(A)_i$ is applied, the associated Boolean condition Θ_i is true, and for all $j \neq i$, Θ_j is false. Note, a trivial Boolean condition may be found for the last Boolean condition of A by taking the inverse of all previous conditions and then taking their logical conjunction; i.e. if A has three successors and Θ_1 and Θ_2 are the Boolean conditions for $succ(A)_1$ and $succ(A)_2$ respectively, then a logically valid condition for $succ(A)_3$ is $\neg\Theta_1 \wedge \neg\Theta_2$. For this research, this trivial condition is not used as it is considered desirable to find a condition based on the environmental factors; however, for some problems it may be acceptable to use it. While CGP is certainly capable of finding S_A Boolean conditions simultaneously [46] the problem can be simplified considerably into S_A individual searches.

PMIT-PARAM works by looping through each $A \in V$, and if A has two or more productions ($S_A \geq 2$), then it performs S_A separate searches using CGP to find a set of Boolean conditions for A (this implies this part could be easily parallelized; however, this was not done for this research).

For this research, CGP was configured as follows. The number of inputs was set to the maximum number of parameters in the symbol's state. For the problems in the test set, this was one to three parameters. Additionally, two special inputs were defined to produce random integer values from 1 to 100, and real values from $(0, 10]$. A single row of 1000 function nodes was used. A single output node is defined. As recommended by Miller [46], the $1 + 4$ evolutionary strategy [66] using Goldman mutation [28] searches the space for the Boolean conditions. For each condition, all of the parametric states for every instance of A are run through the condition. The search terminates when the Boolean condition Θ_i evaluates to true for every instance $A \rightarrow succ(A)_i$ is applied, and evaluates to false otherwise.

5.4 Evaluation

The evaluation methodology for inferring parametric L-systems is relatively straightforward. Consideration is first given to the performance metrics necessary to properly evaluate PMIT-PARAM. Since the algorithm consists of two distinct steps, PMIT-PARAM is evaluated at both steps separately. In practice, to successfully infer a parametric L-system, PMIT-PARAM must succeed in both steps; however, this allows a more granular analysis of PMIT-PARAM's performance to identify limitations and make observations. Afterwards, a test set of existing parametric L-system is described.

PMIT-PARAM is evaluated against the test set at successfully inferring the correct successors. That is to say given a hidden parametric L-system $G_{hidden} = (V, w, P_{hidden})$ generating ρ, PMIT-PARAM infers a stochastic L-system $G_s = (V, w, P_s)$, and checks if P_{hidden} (without parameters and conditions) is equal to P_s. As will be shown in Section 5.5, in some cases PMIT-PARAM times out while inferring G_s. For the purpose of evaluating the inference of $G_{predicted}$ from G_s, even if the first step failed, the successors from G_{hidden} are used as input (without the parameters). Since the procedure to predict G_s is a deterministic algorithm, it is executed once for each L-system in the test set, while the second step using CGP is executed 5 times.

5.4.1 Performance Metrics

Two metrics are used to evaluate PMIT-PARAM: success rate (SR) and time to solve (TTS). Success rate is the percentage of executions where the correct L-system was found — i.e., it matches G_{hidden} as described below. It is calculated separately for both step 1 and step 2. TTS is the time taken for PMIT-PARAM to return a solution or time out, and is also computed separately for both steps. These two performance metrics have been used previously to evaluate different versions of PMIT [9, 12], and LGIN [52]. SR has also been used by Runqiang et al. [71].

Since there is only a single execution of the first step, the success rate must be 0% or 100%. For the second step, SR is a measurement of the percentage of executions for which a set of parametric conditions are found from either the predicted stochastic system, or from the productions of the hidden system without the conditions and parameters if the first step timed out. The Boolean conditions found need not match the exact syntactic conditions as P_{hidden}, but they need to be true if and only if the corresponding conditions are true when being applied in ρ.

TTS is measured separately for each step of PMIT-PARAM. For the first step, TTS is the amount of time taken to return a compatible set of rewriting rules for the input strings. In the second step, TTS measures the time taken to return a compatible set of parametric conditions for the candidate L-system's rewriting rules. Unlike the inference of the successors, the minimum, maximum and average TTS values are recorded over the five executions (as shown in Section 5.5). All TTS values were recorded based on using a single core of an Intel 7700 @ 3.6 GHz CPU with 12 GB of RAM on Windows 10.

5.4.2 Data

The data used to evaluate PMIT-PARAM are 20 known parametric L-systems, all of which have deterministic context-free or context-sensitive rules. All of the L-systems were taken from the L-system repository at the University of Calgary called the virtual laboratory (vlab) [77]. Nine of the systems are primarily intended as learning material to teach students about the principles involved in plant modeling using parametric L-systems. Nine are biological models of different plant species. The last two are a model of a non-species specific plant growing, while being consumed by an insect. 18 of the 20 L-systems are non-erasing, while the remaining two have an erasing production. Due to space constraints, the L-systems themselves are omitted, but their names appear in Table 5.2.

5.5 Results

This section provides the results and discusses any observations made during the evaluation. The results with respect to the performance metrics are shown in Tables 5.2 and 5.3.

PMIT-PARAM was able to infer the successors of the original system in step 1 for all 18 non-erasing

L-systems. PMIT-PARAM is not algorithmically capable of inferring the rewriting rules for the two "insect" systems, as all successors must be non-erasing and these two systems have erasing successors used to represent the insect eating the plant (the models are included anyways for step 2). PMIT-PARAM was able to infer the successors for 12 of the 18 in approximately 12 hours or less. The longest was an L-system model of *Mycelis muralis* [62] that took 26 days. An analysis was done to determine the factor(s) controlling the execution time by computing the Pearson correlation coefficient (and p-value) versus different variables. It was found that the number of successors versus time using a cubic regression had an $R^2 = 0.7355$ with $p = 0.03$, which indicates a relatively strong correlation. This is reasonably expected as the number of successors defines the number of dimensions in the search space. However, a similar regression with the total number of symbols summed across all successors had an $R^2 = 0.8626$ with $p = 0.03$. This stronger correlation is also reasonable as when the successors are quite small the strings do not grow very large. Figures 5.3 and 5.4 show graphs of the data used for these calculations including a 3^{rd} order polynomial trend line with corresponding correlation coefficients.

While some of the L-systems TTS values are quite long in comparison to some of the other results, they are still fairly reasonable in a practical sense for inferring an L-system model for some unknown process. Additionally, all of the executions were done using a single core of a CPU; however, as the first step of PMIT-PARAM uses an exhaustive search, it may be trivially improved by parallel processing. With respect to the associated probabilities, they are always exactly equal to the number of times a successor is selected for a symbol A divided by the total number of A symbols. This makes sense as really the underlying selection is deterministic. For example, Table 5.4 shows an L-system for simulating crocuses. For the symbol a there are two parametric successors; however, to the stochastic L-system these appear to be seven successors. Let Q be used instead of $F(1)[\&(30)\ L(0)]/(137.5)$ for simplification, and the successors are $Qa(2), Qa(3), Qa(4), Qa(5), Qa(6), Qa(7)$, and $F(20)A$. These occur one time each, and so the associated probability for each successor is $1/7$.

PMIT-PARAM (the second step) was 100% successful at inferring parametric conditions for all of the systems in the test set. With respect to TTS, the timing varied considerably from one second to nearly 18 hours between members of the test set. Even for a specific system, the minimum and maximum times could be considerably different. Three factors were found to positively relate to an increase in TTS. The first factor was an increase in the number of parametric conditions required, as this causes more searches to be required. The second factor was the number of parameters in the parametric condition, as they increased the time for that search, and hence the overall time. Neither of these two factors were very surprising. However, the third factor was less expected. It was found for the *Helianthus annuus* system that finding the conditions ran for a long time due to the wide range of values for each condition, and the narrow gaps between them. For example, the rewriting rules for the symbol $C(m)$ require a condition from $m \leq 440$, one from $440 < m < 565$, and one from $565 < m < 580$, etc. The suggests the possibility of using a global/local hybrid search to enhance finding specific values by periodically searching only in a space defined

Step #	Boolean Expression
1	$(((t/7) \cdot (89 \geq 52)) < ((82 \geq 0.139897) = (12 < 79)))$
2	$(((t/7) \cdot 1) < (1 = 1))$
3	$((t/7) < 1)$
4	$(t/7) < 1$

Table 5.1: Shows the step-by-step reduction of a sample Boolean expression found by PMIT-PARAM.

by modifying values in the equations. Additionally, there may be the possibility of using logic to refine the values. For example, if an equation has been found where $m < 411$, it might be possible to recognize that this is true for most of the samples, and specifically false for samples where the values are between 411 and 440, hence suggesting a higher value is needed. The conditions found by PMIT-PARAM are rarely as simple as the original conditions; however, they could be mathematically reduced either by hand, or more likely using software such as Maple [44]. Table 5.5 shows the parametric conditions that were found for the crocuses L-system [62]. Note, programming variables with values of `true` are taken as 1, and values of `false` are taken as 0. Also, $\vee$ is implemented as returning 1 if either operand equals 1; and 0 otherwise, so $45 \vee 83$ is 0 and not undefined. Finally, $\wedge$ is implemented as returning 1 if both operands are 1, and 0 otherwise. This was done to make the search faster since many of the operations tend to be such logical operators, and it could take a long time to find suitable operands. It can be seen that many of the operations become a 0 or 1, although typically somewhere in the condition is an operation that is highly related to the original. For example, consider the rule $a(t) : (((t/7) \cdot (89 \geq 52)) < ((82 \geq 0.139897) = (12 < 79)))$. Table 5.1 shows the step-by-step reduction, which ultimately becomes $t/7 < 1$, which is true when $t < 7$. And, $a(t) : ((7 = t) = ((((45 \vee 83) \cdot ((0.109226 \vee 0.120853) \vee (88 \geq 11))))$ has $t = 7$ in the expression itself, while the remainder reduces to $= 1$, or $(t = 7) = 1)$. The reason these strange expressions occur is because there are many nodes in the CGP grid, and by chance, some combination of nodes will often produce a 0 or 1 due to the operators that return Boolean values, e.g. $m \leq n$. These combinations of nodes can then get applied via a multiplication or division operation to essentially cause no change. It may seem that the process could be simplified to only look for simply relational operations like $t < 7$ or $t = 7$, and certainly that could be attempted first; however, ultimately all of the operations listed in [62] are required. For example, an L-system to model auxin transport called "basipetal" [63] contains Boolean conditions such as $t >= 1 \wedge n < 3$ (indicating the necessity of the logical operators), and $i = u + v$ (indicating the need for mathematical operators).

System	SR	TTS
acropetal-signal*	0.00	7:16:19:45
acrotonic*	1.00	1:33:04
basipetal*	1.00	3:21:12
basipetal-signal*	0.00	1:02:48:29
dibotryoid*	1.00	0:27:05
mesotonic-signal*	0.00	5:09:03:02
florigen v1*	1.00	0:01:12
florigen v2*	1.00	0:00:58
florigen+auxin*	1.00	0:01:24
insect (schematic) [A]	n/a	n/a
insect (realistic) [A]	n/a	n/a
Asplenium resiliens	1.00	0:00:03
Capsella bursa-pastoris	1.00	13:09:38
Convallaria majalis	0.00	12:21:08:50
Crocus	1.00	0:45:16
Helianthus annuus	1.00	4:19:51
Leucanthemum vulgare	1.00	0:01:02
Lychnis coronaria	1.00	0:03:27
Mycelis muralis v1*	0.00	26:06:15:28
Mycelis muralis v2*	0.00	20:11:37:09

*: Indicates the system is context-sensitive.

A: Erasing productions may not be inferred.

Table 5.2: Success rate and time to solve for PMIT-PARAM at inferring successors in step 1.

System	TTS		
	Min	Max	Avg
acropetal-signal*	0:00:07	0:00:14	0:00:11
acrotonic*	0:00:01	0:00:04	0:00:02
basipetal*	0:00:40	0:10:46	0:05:32
basipetal-signal*	0:00:17	0:04:18	0:01:51
dibotryoid*	0:00:02	0:00:15	0:00:06
mesotonic-signal*	1:50:45	6:34:09	3:42:15
florigen v1*	0:00:01	0:00:02	0:00:01
florigen v2*	0:00:01	0:00:11	0:00:04
florigen+auxin*	0:00:01	0:00:06	0:00:03
insect (schematic)	0:00:01	0:00:01	0:00:01
insect (realistic)	0:00:02	0:00:07	0:00:04
Asplenium resiliens	0:00:02	0:00:08	0:00:04
Capsella bursa-pastoris	0:00:01	0:00:01	0:00:01
Convallaria majalis	0:00:01	0:00:01	0:00:01
Crocus	0:00:01	0:00:02	0:00:01
Helianthus annuus	5:13:07	17:51:49	11:15:38
Leucanthemum vulgare	3:18:54	8:15:49	5:10:18
Lychnis coronaria	0:00:01	0:00:01	0:00:01
Mycelis muralis v1*	0:00:01	0:00:01	0:00:01
Mycelis muralis v2*	0:00:04	0:00:27	0:00:11

*: Indicates the system is context-sensitive.

Table 5.3: Minimum, maximum and average time to solve for PMIT-PARAM at inferring parametric conditions for step 2. SR is 100% for all systems, and so it is not shown.

$A : * \rightarrow \ K(0)$
$a(t) : t < 7 \rightarrow F(1)[\&(30)\ L(0)]/(137.5)a(t+1)$
$a(t) : t = 7 \rightarrow F(20)A$
$L(t) : t < 9 \rightarrow L(t+1)$
$K(t) : t < 5 \rightarrow K(t+1)$
$F(l) : l < 2 \rightarrow F(l+0.2)$

Table 5.4: L-system for crocuses as made manually [62]. Default identify productions are not shown.

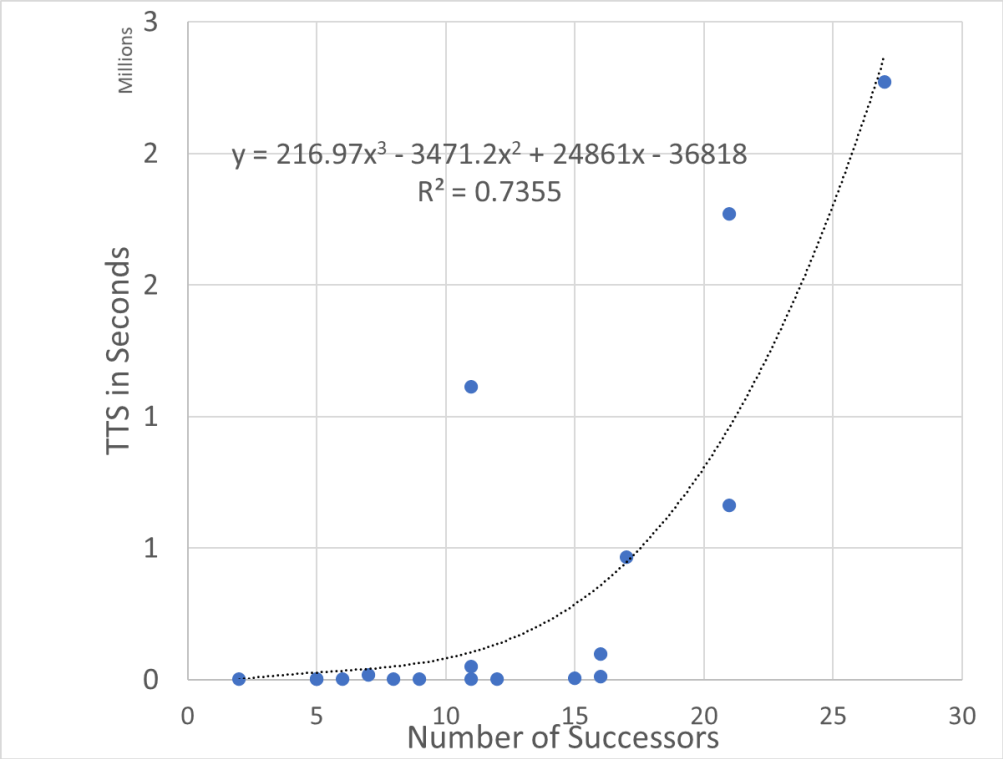


Figure 5.3: The number of successors in each L-system versus the time to infer the successors. A 3^{rd} order polynomial trend line is shown with corresponding correlation coefficient.

$A : *$
$a(t) : (((t/7) \cdot (89 \geq 52)) < ((82 \geq 0.139897) = (12 < 79)))$
$a(t) : ((7 = t) = ((((45 \vee 83) \cdot ((0.109226 \vee 0.120853) \vee (88 \geq 11)))$
$L(t) : ((t/4) > (((0.648579 \leq 48) \wedge (14 < 29)) + (0.477615 \leq 0.999725)))$
$K(t) : ((t + 1)) \leq (9 - t))$
$F(l) : (0.364055 \leq 7) > (p0 - (0.018342 \leq 36))$

Table 5.5: The Boolean conditions found by PMIT-PARAM after step 2. Successors are not shown as they are identical to Table 5.4.

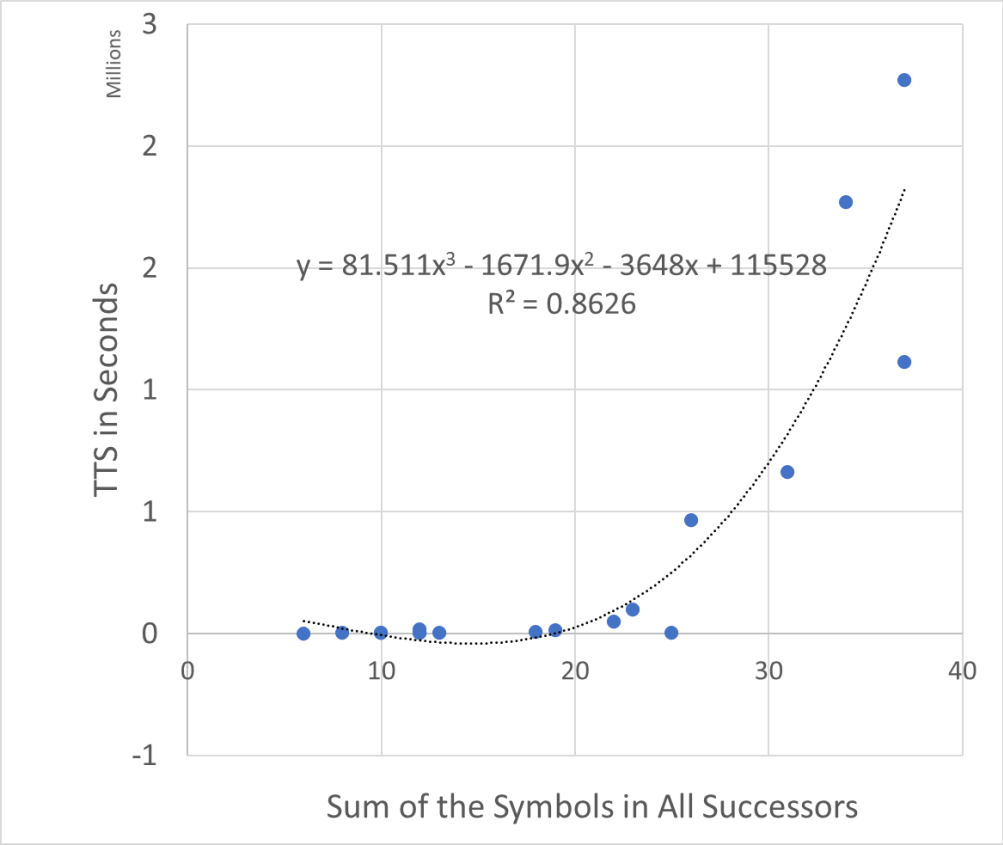


Figure 5.4: The total number symbols summed across all successors in each L-system versus the time to infer the successors. A 3^{rd} order polynomial trend line is shown with corresponding correlation coefficient.

Figure 5.5: A simulation produced in vlab of a population of crocuses using the original L-system in Table 5.4 [62]. Image used with the permission of the copyright holder.

5.6 Conclusions

This paper has introduced the Plant Model Inference Tool for parametric L-systems (PMIT-PARAM), which despite the name is domain agnostic but motivated towards inferring simulations of plants from data. PMIT-PARAM functions by inferring an intermediate stochastic L-system from a sequence of strings, and then converts the stochastic L-system into a parametric L-system with deterministic context-free or context-sensitive rules. The first step is justified by recognizing that when the parametric data is ignored on symbols in a sequence of strings, then a parametric L-systems acts similarly to a stochastic system. The second step uses Cartesian genetic programming (CGP) [46] to find parametric conditions for rewriting rules to replace the associate selection probabilities in the stochastic L-system. It was found that PMIT-PARAM was reasonably successful at inferring the successors as it inferred 18 of 20 systems. Two could not be inferred because the underlying systems contained erasing rewriting rules, which violates the assumptions of PMIT-PARAM. While the successors for 12 of the L-systems took less than 12 hours to be inferred, the longest took 26 days. This is still practical for inferring an L-system compared to the effort needed to find them by hand over a long period of time. Additionally, the intermediate stochastic system inference uses an exhaustive search, and so it could be trivially improved with parallelism. PMIT-PARAM was 100% successful at inferring the parametric conditions for all 20 systems in the test set, taking from a sub-second to 17 hours depending on the system.

Parametric L-systems are considered particularly apt for modeling natural processes as they can be used to model the environmental factors that influence the underlying processes. Parametric L-systems are particularly difficult to produce by hand as they typically rely on *a priori* scientific knowledge about the effects of the environmental factors on the process. Inductively inferring a parametric L-system algorithmically, as shown in this paper, requires only a sequence of strings. As such, the algorithm herein not only infers L-systems but has the potential to reveal scientific knowledge by inferring the relationships between environmental factors and a process.

For PMIT-PARAM, the future work will focus on adjusting it so that it can infer an L-system with an erasing rewriting rule since some processes have such underlying mechanisms. With respect to inferring Boolean conditions, PMIT-PARAM may be improved by means of a global/local search hybrid since it was recognized that one challenge experienced by PMIT-PARAM was in refining the specific values found by CGP into the simplest possible condition. Additionally, a future investigation will examine the effect of adding extraneous parameters to the state that are unrelated to the parametric conditions.

5.7 Acknowledgments

The authors would like to acknowledge Professor Julian Miller for his kind assistance with optimizing the parameters for CGP. This research was undertaken thanks in part to funding from the Canada First Re-

search Excellence Fund and the National Sciences and Engineering Research Council, grant #2016-06172 and Alexander Graham Bell scholarship for Jason Bernard.

Chapter 6

Summary and Future Research

This chapter summarizes the work done in support of this thesis, and the potential impact of the work. It begins by providing a broad overview of the contributions to the study of the inductive inference problem. This is followed by a summary of the work itself. Additionally, some of the challenges faced when developing the PMIT algorithms are discussed, including some of the (potential) limitations that exist as a result. This chapter, and the thesis, conclude with a discussion on the future directions of this research.

6.1 Contributions

The overarching goal of this research was to investigate questions surrounding the L-system inference problem, with a particular focus towards making a practical tool that is research domain agnostic; i.e., requiring only a sequence of strings as input[1]. Currently, developing an L-system for a specific process is typically done by hand, which may require considerable time and effort depending on the expertise of the builder, and availability of *a priori* information [26,53,64]. Galarreta et al. describes the process as "tedious and intricate handwork" that could be improved by an algorithm to "infer rules and parameters automatically from real ... images", and the intricate methodology of Nishida [53] taking many measurements of Japanese Cypress trees to produce 42 rewriting rules reinforces this. This work represents a significant part of what is needed for an end-to-end process for algorithmic L-system inference from imagery. A seperate component is needed to capture suitable imagery, and convert the imagery to strings via segmentation. The potential impact is to reduce the time and effort required to produce high quality models of temporal processes. However, Godin and Ferraro state that model inference "could be further exploited in combination with investigations at a biomolecular level to better understand plant development." [27]. In other words, developing a model algorithmically has the potential to reveal the mechanisms and factors that control a process; thereby, expanding the potential impact of this work to include making additional scientific discoveries.

The main motivation of this research was to take steps towards practical generalized L-system inference from data obtained from a process. It was argued that based on the literature the most commonly used L-systems are: 1) deterministic with alphabet of 5 or more symbols (often 12 or more), 2) stochastic L-systems,

[1]Originally, it was thought that *a priori* knowledge of plant biology would be needed to infer L-systems for plant models. Hence the algorithm's name Plant Model Inference Tool. However, early success allowed the algorithm to be focused towards domain agnosticism.

and 3) parametric L-systems. Prior to this work, the state-of-the-art for deterministic L-system inference was limited to context-free systems with a maximum of 2 symbols in the alphabet, and no generalized inference algorithms existed for stochastic or parametric L-systems, while there exists some for specific problems (e.g., [20, 26, 33, 35, 73]) or that can infer parts of a parametric L-system [18, 74]. This primary contribution to generalized inductive inference of deterministic, stochastic and parametric L-systems is the following:

- The Plant Model Inference Tool[2] for Deterministic Context-Free L-systems (PMIT-D0L) can infer D0L-systems with at least up to 31 symbols in a few seconds.
- The Plant Model Inference Tool for Stochastic L-systems can infer stochastic L-systems (PMIT-S0L) with at least up to 9 rewriting rules (12 with a few restrictions lifted) in under a self-imposed one day time limit.
  - PMIT-S0L was not able to infer Nishida's Japanese Cypress tree models due to its extremely large size [53].
- The Plant Model Inference Tool for Stochastic L-systems can infer parametric L-systems (PMIT-PARAM) with the following limitations:
  - Up to 9 rewriting rules (12 with a few restrictions lifted) in under a self-imposed one day time limit.
  - Up to 25 Boolean conditions in under a self-imposed one day time limit.

The following additional discoveries were made that can provide guidance towards the future development of L-system inference tools.

- Discovered novel search reduction techniques primarily based on examination of the mathematical properties of the input strings.
- Discovered novel search space definitions to describe a space of L-systems.
- Discovered that a stochastic L-system may be found from the multitude of possible solutions by preferring the one found with the greatest probability of producing the input strings.
- Developed new metrics for assessing accuracy of stochastic L-system inference.
- Discovered that all context-sensitive and non-deterministic L-systems initially appear to be stochastic (based on an observation by Ian McQuillan, see Preface).

[2]While the name "Plant Model Inference Tool" might suggest that the work is limited to plant models, this is simply a holdover from the earliest days of this research where the focus was on plant modelling. All of the tools are domain agnostic as they require only a sequence of strings as input.

It has been shown that the earliest work on improving the inference of D0L-systems was fundamental to inferring S0L-systems. In turn, the work on inferring S0L-systems made it possible to infer parametric L-systems. The goal of this work was to take steps towards practical L-system inference for real-world processes, which was taken to mean being able to infer L-systems more complex than D0L-systems as it is expected that more complex forms are needed to produce high-quality models for at least some real-world processes. This goal has been accomplished albeit with some limitation on the complexity of an L-system that can be inferred based on the number of successors (although the upper limit, 12, is still higher than the limit of state-of-the-art inference algorithms on inferring only D0L-systems prior to this work). It is acknowledged that while these contributions are a promising step forward there are further challenges that remain to have truly practical inductive inference of an L-system (some of which are discussed Section 6.3 below).

6.2 Summary

In Chapter 2, this thesis provided a background on the theory and concepts surrounding L-systems. That chapter also discussed the various applications of L-systems by showing how they are used to model natural processes (e.g., [16, 26, 62, 68]), human-engineered processes (e.g., [50, 81]), and processes in theoretical computer science (e.g., [58, 70, 80]). Chapter 2 also discusses the state of L-system inference. Prior to this work, L-system inference had been investigated quite a bit with some success albeit with serious limitations. L-system inference could be categorized into two broad groups. In one group are the tools intended to be aides to experts as they develop L-systems (e.g., [3, 36, 47]. Such tools tend to focus on presenting the user with a visualization of a population of L-systems. The user selects one or more L-systems that are aesthetically pleasing and the software iterates until the user is content with the result. The main limitation is that such a tool is difficult to use for a specific process as it requires the user to seed the program with an L-system that is approximately correct. Additionally, these tools tend to use small alphabets, likely due to the aforementioned need for a seed system from the operator. The other group are automatic L-system inference algorithms that infer an L-system from data obtained from the process itself; e.g., a sequence of strings (e.g., [23, 52, 71]). As argued above, such algorithms have a greater potential to make an impact; however, the state-of-the-art algorithms were limited to deterministic context-free L-systems (D0L-systems) with an alphabet of 2 symbols. L-systems for real-world processes almost always have 5 or more symbols, with many having 12 or more.

Based on the literature review, the first research goal was to develop an L-system inference algorithm that can infer L-systems with many more symbols in the alphabet. After all, if it is not possible to infer D0L-systems with larger alphabets then it is certainly impossible to infer more complex L-systems as that is a strictly harder problem. Chapter 3 presented the work on investigating this goal. PMIT-D0L was successful due to the use of a novel encoding scheme that recognizes that every successor is a subword of the input strings; therefore, it is only necessary to find the length of the successor for each symbol in the alphabet

(other discoveries were made on other different encoding schemes as detailed in the chapter). The successors are found by scanning the strings symbol-by-symbol, and choosing successors of the length taken from a candidate solution found by a search algorithm (GA and brute force were evaluated). This encoding scheme reduces the dimensions of the search space over other encoding schemes used in the literature, as only one dimension is needed for each symbol in the alphabet. While reducing the number of dimensions may help to improve search speed, it is not necessarily sufficient on its own; e.g., if the bounds on the dimensions were to be very large. However, in this case, using a variety of techniques, the bounds are kept very reasonable, and in many cases were actually smaller than the alphabet size (the typical dimension size in other approaches). These findings were fundamental to moving towards inferring more complex types of L-systems.

The next research goal, discussed in Chapter 4, was to develop an algorithm for inferring stochastic context-free L-systems (S0L-systems). There exists two primary challenges to overcome for developing an algorithm to infer S0L-systems. There are a multitude of S0L-systems that can produce a given sequence of strings; therefore, the first step is to determine a method for selecting a solution. The method developed determines the probability that a candidate solution produced the input strings. The solution returned is the candidate L-system found with the greatest computed probability of producing the strings of those searched. The second challenge is that the analytical techniques for determining facts about the successors in deterministic L-systems, do not work in stochastic L-systems as every instance of a symbol may have an entirely new successor. This challenge directly influences the search space as it effects the number of successors in the S0L-systems (dimensions) and the bounds on the successor lengths (scope). No good solution was found to this problem and as such this remains an area of future research (discussed below). For this research, some reasonable assumptions were made on the number of successors and the upper bounds on the successor lengths (lower bound assumed to be 1) based on a statistical analysis of known L-systems. The Plant Model Inference Tool for Stochastic Context-free L-systems (PMIT-S0L) uses a greedy algorithm (hybridized with a search) that when making a local choice maximizes the probability a resulting solution will produce the input sequence of strings. PMIT-S0L was evaluated primarily using procedurally generated L-systems (3770 of them) as only one S0L-system could be found that is explicitly provided in the literature [53]. PMIT-S0L was also evaluated on this solitary published L-system as well.

Of note, since PMIT-S0L uses an exhaustive search, the speed could be trivially improved used parallelism, and PMIT-S0L was given a significant disadvantage compared to PMIT-D0L as it did not assume assume any identity productions (such as those that usually occur for graphical symbols) — they were filtered out from the Japenese Cypress S0L-system. This was done as graphical symbols allow the possibility of some analysis of the strings using the same techniques as in Chapter 3. The loss of branching symbols is a particular loss since these provide essentially free gains. This harder variant of the inference problem was intentionally selected so as to eliminate any effects from the analytical techniques, and clearly establish the success and limitations of the algorithm itself; however, in practice, it would be unusual for any physical process to have no such symbols. Thus, PMIT-S0L's performance in practice would be expected to be greatly improved.

As mentioned a few times in this thesis, given a sequence of strings that are produced by an L-system that is more complex than a D0L-system, then they can be initially assumed to be produced stochastically as they have such an appearance; i.e, different instances of a symbol A are producing different successors[3].Therefore, if a stochastic L-system can be found that produces a sequence of strings, then there may exist a transformation algorithm to alter the stochastic L-system into another form. Chapter 5 presents the Plant Model Inference Tool for Parametric Deterministic L-systems that first finds a stochastic L-system, using the algorithm from Chapter 4, and then it uses Cartesean Genetic Programming to replace the associated probabilities on the successors of the stochastic L-system with Boolean conditions using a set of parameters assumed to be provided as input, along with a sequence of strings. PMIT-PARAM was evaluated in two stages on 20 known parametric L-systems. First, it was evaluated at finding a stochastic L-system with the same successors as the parametric L-system in the test set.

6.3 Future Work

The future of PMIT, or L-system inference, lays in two primary directions, which will be expanded upon in the subsections that follow. While this work has shown that L-system inference can be successful for D0L-systems, S0L-systems, and parametric L-systems, there exist many more different variants of L-systems as partially described in Chapter 2. So, it would be useful to investigate techniques for inferring these other variants. That being said, this thesis has described three algorithms (although PMIT-PARAM shares much with PMIT-S0L) for inferring three different L-system variants. This imposes the practical problem when inferring an L-system of trying to decide which algorithm should be run. It would be more practical to have a single algorithm that can infer all types of L-systems.

6.3.1 Inferring Other Types of L-systems

Chapter 2 describes many, but not all, variants of L-systems. Of those variants, there are three types that seem particularly important for being able to practically model a process from imagery (or from a sequence of strings produced by some method for non-physical processes), so it would be useful in the future to investigate at least the following two types of L-systems: 1) context-sensitive L-systems, and 2) homomorphic L-systems.

Context-sensitivity has been shown to be a useful mechanism for modeling signaling in a process [62]. While parametric L-systems can sometimes be used for as well for signalling, context-sensitivity is sometimes more appropriate, and may be easier to infer a context-sensitive L-system than a parametric one as the data that describe the signals may be difficult to capture as detailed in Chapter 2.

[3]The techniques in Chapter 3 will confirm that the strings cannot be made by a D0L-system in many cases. Consider the case where a symbol A has a set of successors $S = \{S_1, \ldots, S_n\}$, and let S_l be the longest successor. If for any $S_i \in S$, $1 \leq i \leq n$, $i \neq l$, is not a subword of S_l, then the techniques will deduce that the strings cannot be produced by a D0L-system; otherwise, it may be inconclusive.

Even for a physical process, due to the scale or time frame involved[4], it may not always be possible for RGB (or other) imagery to capture some aspects important for constructing an L-system from a process in action. As a result, any strings produced from imagery for such processes will not have a symbol that corresponds to such a difficult to capture mechanism. Either the symbol will be missing or, it will in effect be projected onto the physical structure that is visible. In either case, when the alphabet is obfuscated in such a manner this is usually simulated through an additional visualization method called a homomorphism. A homomorphism is a function that is defined on the words in terms of how it maps letters. For example, a homomorphism could map two letters A and B onto the letter F (the turtle interpretation symbol for a line). This would have the effect of both A and B in a word of a developmental sequence causing the simulator to draw a line. But if one were using segmentation only and F would be detectable, and not the A's and B's but the L-system would rely on having A's and B's. Similarly, the homomorphism can map onto the empty word (corresponding to some element that is not detectable on the image). Hence, it is imperative to be able to infer L-systems with homomorphisms, and this is future work.

6.3.2 Data Issues with Inferring L-systems

With respect to parametric L-systems, the tool presented in this thesis assumes that at least one of the symbols is temporal, and this is a reasonable assumption especially for temporal processes. However, as seen in [78], the more desirable method would find the relationships between environmental data and the phenotypic components of the plant. Other environmental data that are factors in determining how the process functions may also be captured using various devices such as multispectral cameras, GPSs, thermometers, etc. While not discussed in detail, the non-temporal parameters for L-systems in the test set we used for evaluating PMIT-PARAM are generally non-environmental, but represent signals. Hence, it is uncertain how well PMIT-PARAM could infer L-systems using environmental data, and while algorithmically it should be able to find Boolean conditions for such data, this should be investigated.

For processes that have no physical form; e.g, a computer program, then some other data would be captured, such as outputs, that would allow a string representation to be produced for each time step. The string production step involves taking the data captured and converting it into a string representation. It is unclear what sort of data is necessary to infer such non-physical processes as all of the known L-systems represent at least a hypothetical physical structure. Even the fractals, which are mathematical constructs, could be made physical. While the PMIT tools should be capable of inferring L-systems for such processes, it should not taken as a given.

Finally, as discussed in Chapter 1, noisy data is a recognized existing major limitation for grammar inference [22]. This thesis justified examining perfect strings as a good first step towards improving inductive L-system inference. Noise in the data can come in different forms, and from different sources. Errors in the

[4]For example, with plants much of the physical process is happening at a microscopic level. While for geological models, the time frames tend to be very long

string data can take the form: 1) insertion errors (a symbol is in the string that should not there), 2) deletion errors (a symbol is missing that should be present), and 3) modification errors (a symbol is misidentified as a different symbol; e.g., an A is present in the string as a B. Such errors may be thought of as an insertion error of a B and deletion error of the A). Errors can occur in different ways; however, the four main sources of error are likely to be from: 1) natural process variation, 2) lack of detectability, 3) string conversion, and 4) external non-systematic error. While the goal of L-system inference is to perfectly capture the mechanical relationship between a process and a model, especially with parametric L-systems, this is likely very difficult due to the inability to perfectly capture every event and all of the corresponding data for the event. Hence, there will appear to be some unexplained natural variation in the process which may appear as insertion or deletion errors. As previously discussed, for many processes the mechanisms may not be readily visible due to scale, time frame, or type of imagery used. Such mechanisms are likely to be projected onto the physical structure, hence appearing as deletion errors. Certainly the goal of component-based phenotyping is to perfectly skeletonize plant imagery, and identify the individual phenotypic elements; however, it is possible there may exist some error in such algorithms, or certain components may be occluded from the images used. Finally, there exists the possibility that a non-systematic error may occur while capturing the data for a process (e.g., a branch falling off a plant during scanning).

Now that some progress has been made, especially for D0L-systems, it would be desirable to investigate L-system inference with noisy input data. As with other challenges, it is clear that noisy data also causes the input string sequence to appear to have stochastic characteristics. It may be possible to remove errors by first finding a stochastic L-system and then correcting the errors via analysis. However, there is one issue that cannot be addressed this way. While this approach could be reasonably expected to work for detecting deleted symbols in a successor, it is unclear how to determine that a deleted symbol produced its successor symbols. Let $w_i \Rightarrow w_{i+1} \Rightarrow w_{i+2}$, $w_i = a_1 a_2 \cdots a_n$, $w_{i+1} = b_1 b_2 \cdots b_m$, and $w_{i+1} = c_1 c_2 \cdots c_o$. If there is a deletion error in the successor of a_1 such that b_2 is missing, then the symbols in w_{i+2} that correspond to the successor of b_2 now seem to be produced by other symbol(s); e.g., b_1 and/or b_3. Resolving this issue is an area of future work.

6.3.3 Universal L-system Inference

Suppose that there exists some technique for transforming an S0L-system to every other known type of L-system similar to the methodology of PMIT-PARAM. This could be used as an intermediate step for all other types. For example, for context-sensitivity, do all of the successors for a symbol A match a context-sensitive selection? If so, then the associated probabilities can be replaced with context-sensitive rules for A. Thus, if an algorithm examines every possible S0L-system for a sequence of strings, and applies the hypothetical transformation algorithm on each S0L-system, then it will instead find an L-system for a process [5]. If

[5] This assumes there can be some reasonable hierarchy for L-systems; e.g., D0L-system is best, followed by D2L-system, etc. The actual best solution may be problem specific.

the greedy aspect is removed from the PMIT-S0L algorithm, then within the bounds specified by PMIT-S0L examines all possible S0L-systems for a sequence of strings. However, PMIT-S0L without the greedy algorithm is intractable. Thus, it seems that thesis has presented some groundwork for a single algorithm that can infer all known types of L-systems; however, it will require some additional investigation.

References

[1] Masahiro Agu and Yota Yokoi. A stochastic description of branching structures of trees. *Journal of Theoretical Biology*, 112(4):667–676, 1985.

[2] M. Allen, Przemyslaw Prusinkiewicz, and Theodore DeJong. Using L-systems for modeling source-sink interactions, architecture and physiology of growing trees: The L-PEACH model. *New Phytologist*, 166(3):869–880, 2005.

[3] Fabricio Anastacio, Przemyslaw Prusinkiewicz, and Mario Costa Sousa. Sketch-based parameterization of L-systems using illustration-inspired construction lines and depth modulation. *Computers & Graphics*, 33(4):440–451, 2009.

[4] Daniel Ashlock, Kenneth Bryden, and Stephen Gent. Simultaneous evolution of bracketed L-system rules and interpretation. In *2006 IEEE International Conference on Evolutionary Computation*, pages 2050–2057. IEEE, 2006.

[5] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Orogramming, Genetic Algorithms*. Oxford University Press, 1996.

[6] A. Barra. Rosa "Mrs John Liang". URL: `https://commons.wikimedia.org/wiki/Rosa_%27Mrs._John_Laing%27#/media/File:Rosa_'Mrs_John_Laing'.jpg`.

[7] Farah Ben-Naoum. A survey on L-system inference. *INFOCOMP Journal of Computer Science*, 8(3):29–39, 2009.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[9] Jason Bernard and Ian McQuillan. A fast and reliable hybrid approach for inferring L-systems. In *Proceedings of the 2018 International Conference on Artificial Life*, volume 30, pages 444–451. MIT Press, 2018.

[10] Jason Bernard and Ian McQuillan. Inferring stochastic L-systems using a hybrid greedy algorithm. In *Proceedings of the 30th International Conference on Tools with Artificial Intelligence*, pages 600–607. IEEE, 2018.

[11] Jason Bernard and Ian McQuillan. New techniques for inferring L-systems using genetic algorithm. In *Proceedings of the 8th International Conference on Bioinspired Optimization Methods and Applications*, volume 10835 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2018.

[12] Jason Bernard and Ian McQuillan. Stochastic L-system inference from multiple string sequence inputs, 2020. URL: `https://arxiv.org/abs/2001.10922`, `arXiv:2001.10922`.

[13] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[14] Mikolaj Cieslak and Przemyslaw Prusinkiewicz. Gillespie-Lindenmayer systems for stochastic simulation of morphogenesis. *in silico Plants*, 1(1):diz009, 2019.

[15] Mikolaj Cieslak, Adam Runions, and Przemyslaw Prusinkiewicz. Auxin-driven patterning with unidirectional fluxes. *Journal of Experimental Botany*, 66(16):5083–5102, 2015.

[16] John Corbit and David Garbary. Computer simulation of the morphology and development of several species of seaweed using Lindenmayer systems. *Computers & Graphics*, 17(1):85–88, 1993.

[17] Paul Henry Cournède and Cedric Loi. Generating functions of stochastic L-systems and application to models of plant development. *Discrete Mathematics & Theoretical Computer Science*, 2008.

[18] Roger Curry. On the evolution of parametric L-systems. Technical report, University of Calgary, 2000.

[19] Robertas Damaševičius. Structural analysis of regulatory DNA sequences using grammar inference and support vector machine. *Neurocomputing*, 73(4-6):633–638, 2010.

[20] Gemma Danks, Susan Stepney, and Leo Caves. Protein folding with stochastic L-systems. In *11th International Conference on the Simulation and Synthesis of Living Systems*, pages 150–157. MIT Press, 2008.

[21] Sruti Das Choudhury, Srinidhi Bashyam, Yumou Qiu, Ashok Samal, and Tala Awada. Holistic and component plant phenotyping using temporal image sequence. *Plant Methods*, 14(1):35, 2018.

[22] Colin De La Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[23] P. G. Doucet. The syntactic inference problem for D0L-sequences. *L Systems*, pages 146–161, 1974.

[24] Peter Eichhorst and Walter J Savitch. Growth functions of stochastic Lindenmayer systems. *Information and Control*, 45(3):217–228, 1980.

[25] Dinnus Frijters and Aristid Lindenmayer. A model for the growth and flowering of *Aster novae-angliae* on the basis of table $< 1, 0 >$ L-systems. In *L systems*, pages 24–52. Springer, 1974.

[26] Miguel A Galarreta-Valverde, Maysa MG Macedo, Choukri Mekkaoui, and Marcel Jackowski. Three-dimensional synthetic blood vessel generation using stochastic L-systems. In *Medical Imaging: Image Processing*, page 86691I, 2013.

[27] Christophe Godin and Pascal Ferraro. Quantifying the degree of self-nestedness of trees: application to the structural analysis of plants. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(4):688–703, 2010.

[28] Brian W. Goldman and William F. Punch. Analysis of Cartesian genetic programming's evolutionary mechanisms. *IEEE Transactions on Evolutionary Computation*, 19(3):359–373, 2015.

[29] John Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):122–128, 1986.

[30] Félix P Hartmann, Pierre Barbier De Reuille, and Cris Kuhlemeier. Toward a 3D model of phyllotaxis based on a biochemically plausible auxin-transport mechanism. *PLoS Computational Biology*, 15(4):e1006896, 2019.

[31] P. H. Hellendoorn and Astrid Lindenmayer. Phyllotaxis in *Bryophyllum tubiflorum*: morphogenetic studies and computer simulations. *Acta Botanica Neerlandica*, 23(4):473–492, 1974.

[32] Gabor Herman and Grzegorz Rozenberg. *Developmental Systems and Languages*. North-Holland, Amsterdam, 1975.

[33] Gregory S Hornby, Hod Lipson, and Jordan B Pollack. Evolution of generative design systems for modular physical robots. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, volume 4, pages 4146–4151. IEEE, 2001.

[34] Gregory S Hornby and Jordan B Pollack. Body-brain co-evolution using L-systems as a generative encoding. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 868–875. Morgan Kaufmann Publishers Inc., 2001.

[35] Gregory S Hornby and Jordan B Pollack. Evolving L-systems to generate virtual creatures. *Computers & Graphics*, 25(6):1041–1048, 2001.

[36] Christian Jacob. Genetic L-system programming: breeding and evolving artificial flowers with Mathematica. In *Proceedings of the 1st International Mathematica Symposium*, pages 215–222, 1995.

[37] Christian Jacob. Evolving evolution programs: Genetic programming and L-systems. In *Proceedings of the 1st Annual Conference on Genetic Programming*, pages 107–115. MIT Press, 1996.

[38] Lila Kari, Grzegorz Rozenberg, and Arto Salomaa. L systems. In *Handbook of Formal Languages: Word, Language, Grammar*, volume 1, pages 253–328. Springer, 1997.

[39] Radoslaw Karwowski and Przemyslaw Prusinkiewicz. The L-system-based plant-modeling environment L-studio 4.0. In *Proceedings of the 4th International Workshop on Functional-Structural Plant Models*, pages 403–405. UMR AMAP Montpellier, France, 2004.

[40] Aristid Lindenmayer. Adding continuous components to L-systems. In *L systems*, pages 53–68. Springer, 1974.

[41] Astrid Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology*, 18(3):280–315, 1968.

[42] Bernd Lintermann and Oliver Deussen. Interactive modeling of plants. *IEEE Computer Graphics and Applications*, 19(1):56–65, 1999.

[43] Benoit Mandelbrot. *The Fractal Geometry of Nature*, volume 173. WH freeman New York, 1983.

[44] MapleSoft. solve. URL: `https://www.maplesoft.com/support/help/maple/view.aspx?path=Task/SolveInequality`.

[45] Ian McQuillan, Jason Bernard, and Przemyslaw Prusinkiewicz. Algorithms for inferring context-sensitive L-systems. In *17th International Conference on Unconventional Computation and Natural Computation*, volume 10867 of *Lecture Notes in Computer Science*, pages 117–130. Springer, 2018.

[46] Julian Miller. *Cartesian Genetic Programming*, volume 43. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[47] Kenrick J. Mock. Wildwood: The evolution of L-system plants for virtual environments. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 476–480. IEEE, 1998.

[48] Ralph Morelli, Ralph Walde, E. Akstin, and Craig Schneider. L-system representation of speciation in the red algal genus *Dipterosiphonia* (*Ceramiales*, *Rhodomelaceae*). *Journal of Theoretical Biology*, 149(4):453–465, 1991.

[49] Richard J Morris. *Mathematical Modelling in Plant Biology*. Springer, 2018.

[50] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Transactions On Graphics*, 25(3):614–623, 2006.

[51] Ryohei Nakano. Emergent induction of deterministic context-free L-system grammar. In *Innovations in Bio-inspired Computing and Applications*, pages 75–84. Springer International Publishing, 2014.

[52] Ryohei Nakano and Naoya Yamada. Number theory-based induction of deterministic context-free L-system grammar. In *International Conference on Knowledge Discovery and Information Retrieval*, pages 194–199. SCITEPRESS, 2010.

[53] Taishin Nishida. K0L-system simulating almost but not exactly the same development-case of Japanese Cypress. *Memoirs of the Faculty of Science, Kyoto University, Series B*, 8(1):97–122, 1980.

[54] Przemyslaw Prusinkiewicz. Graphical applications of L-systems. In *Proceedings of Graphics Interface*, volume 86, pages 247–253, 1986.

[55] Przemyslaw Prusinkiewicz, Mikolaj Cieslak, Pascal Ferraro, and Jim Hanan. Modeling plant development with L-systems. In *Mathematical Modelling in Plant Biology*, pages 139–169. Springer, 2018.

[56] Przemyslaw Prusinkiewicz, Scott Crawford, Richard Smith, Karin Ljung, Tom Bennett, Veronica Ongaro, and Ottoline Leyser. Control of bud activation by an auxin transport switch. *Proceedings of the National Academy of Sciences*, 106(41):17431–17436, 2009.

[57] Przemyslaw Prusinkiewicz, Yvette Erasmus, Brendan Lane, Lawrence D Harder, and Enrico Coen. Evolution and development of inflorescence architectures. *Science*, 316(5830):1452–1456, 2007.

[58] Przemyslaw Prusinkiewicz and James Hanan. L-systems: From formalism to programming languages. In *Lindenmayer Systems*, pages 193–211. Springer, 1992.

[59] Przemyslaw Prusinkiewicz and Jim Hanan. Visualization of botanical structures and processes using parametric L-systems. In *Scientific Visualization and Graphics Simulation*, pages 183–201. John Wiley & Sons, Inc., 1990.

[60] Przemyslaw Prusinkiewicz, Radoslaw Karwowski, and Brendan Lane. The L+C plant modelling language. *Functional-Structural Plant Modelling in Crop Production*, pages 27–42, 2007.

[61] Przemyslaw Prusinkiewicz, Radoslaw Karwowski, Radomír Měch, and Jim Hanan. L-studio/cpfg: a software system for modeling plants. In *International Workshop on Applications of Graph Transformations with Industrial Relevance*, pages 457–464. Springer, 1999.

[62] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Science & Business Media, 2012.

[63] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental models of herbaceous plants for computer imagery purposes. *Computer Graphics*, 22(4):141–150, 1988.

[64] Przemyslaw Prusinkiewicz, Lars Mündermann, Radoslaw Karwowski, and Brendan Lane. The use of positional information in the modeling of plants. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 289–300. ACM, 2001.

[65] Przemyslaw Prusinkiewicz and Adam Runions. Computational models of plant development and form. *New Phytologist*, 193(3):549–569, 2012.

[66] Ingo Rechenberg. *Evolutionsstrategie'94*. Frommann-Holzboog, Stuttgart, Germany, 1994.

[67] Paolo Rocca, Giacomo Oliveri, and Andrea Massa. Differential evolution as applied to electromagnetics. *IEEE Antennas and Propagation Magazine*, 53(1):38–49, 2011.

[68] Guillaume Rongier, Pauline Collon, and Philippe Renard. Stochastic simulation of channelized sedimentary bodies using a constrained L-system. *Computers & Geosciences*, 105:158–168, 2017.

[69] Grzegorz Rozenberg and Arto Salomaa. *The Mathematical Theory of L-systems*. Academic press, 1980.

[70] Grzegorz Rozenberg and Arto Salomaa. *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*. Springer Science & Business Media, 2012.

[71] Bian Runqiang, Phoebe Chen, Kevin Burrage, Jim Hanan, Peter Room, and John Belward. Derivation of L-system models from measurements of biological branching structures using genetic algorithms. In *Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 514–524. Springer, 2002.

[72] Xavier Sirault, Jurgen Fripp, Anthony Paproki, Peter Kuffner, Chuong Nguyen, Rongxin Li, Helen Daily, Jianming Guo, and Robert Furbank. PlantScan: a three-dimensional phenotyping platform for capturing the structural dynamic of plant development and growth. In *Proceedings of the 7th International Conference on Functional-Structural Plant Models*, pages 45–48, 2013.

[73] Ondrej Šťava, Bedrich Beneš, Radomir Měch, Daniel G Aliaga, and Peter Krištof. Inverse procedural modeling by automatic generation of L-systems. *Computer Graphics Forum*, 29(2):665–674, 2010.

[74] Ondrej Šťava, Sören Pirk, Julian Kratt, Baoquan Chen, Radomír Měch, Oliver Deussen, and Bedrich Beneš. Inverse procedural modelling of trees. *Computer Graphics Forum*, 33(6):118–131, 2014.

[75] Jordan Ubbens, Mikolaj Cieslak, Przemyslaw Prusinkiewicz, and Ian Stavness. The use of plant models in deep learning: an application to leaf counting in rosette plants. *Plant Methods*, 14(1):6, 2018.

[76] Jordan R Ubbens and Ian Stavness. Deep plant phenomics: a deep learning platform for complex plant phenotyping tasks. *Frontiers in plant science*, 8:1190, 2017.

[77] University of Calgary. Algorithmic Botany. URL: `http://algorothmicbotany.org`.

[78] Tomonari Watanabe, Jim Hanan, Peter Room, Toshihiro Hasegawa, Hiroshi Nakagawa, and Wataru Takahashi. Rice morphogenesis and plant architecture: measurement, specification and the reconstruction of structural development by 3D architectural modelling. *Annals of Botany*, 95(7):1131–1143, 2005.

[79] Eric Weisstein. "cellular automaton" From MathWorld–A Wolfram Web Resource. URL: `http://mathworld.wolfram.com/CellularAutomaton.html`.

[80] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601, 1983.

[81] Peter Worth and Susan Stepney. Growing music: musical interpretations of L-systems. In *Workshops on Applications of Evolutionary Computation*, volume 3449 of *Lecture Notes in Computer Science*, pages 545–550. Springer, 2005.

[82] Mair Zamir. Arterial branching within the confines of fractal L-system formalism. *The Journal of General Physiology*, 118(3):267–276, 2001.