A Fast and Reliable Hybrid Approach for Inferring L-systems*

Jason Bernard[1], Ian McQuillan[1]

[1]University of Saskatchewan, Saskatoon, SK
jason.bernard@usask.ca, mcquillan@cs.usask.ca

Abstract

Lindenmayer systems (L-systems) are a formal grammar system that iteratively create new strings from previous strings by rewriting each of its symbols in parallel according to a set of rewriting rules. The symbols in the string sequence produced can be taken as instructions to produce a visualization of a process over time. They have been especially useful for creating accurate simulations of plants. The L-system inductive inference problem is the problem of inferring an L-system that initially produces a given sequence of strings. Here, a new tool to solve this problem, PMIT-D0L is introduced, that combines projected solutions with linear diophantine equations, heuristics, and genetic algorithm. PMIT-D0L was validated using 28 previously developed deterministic context-free L-systems of different complexity, and it can infer every L-system in the testbed with 100% success rate in less than 4 seconds, a significant improvement over existing implemented tools.

Introduction

In 1968, Lindenmayer (Lindenmayer, 1968) proposed a formal grammar system, later called L-systems, to model cellular interactions. Since then, L-systems have been recognized as a robust modeling tool in many diverse research domains such as plant modeling (Lindenmayer, 1968; Prusinkiewicz and Lindenmayer, 1990), arterial branching (Zamir, 2001; Galarreta-Valverde et al., 2013), sedimentary channels (Rongier et al., 2017), and theoretical computer science (Prusinkiewicz and Hanan, 1992). Within a few years of the introduction of L-systems, researchers started laying the theoretical groundwork for the L-system inductive inference problem (e.g., (Feliciangeli and Herman, 1973; Herman and Walker, 1972)). One variant of inductive inference is, given a sequence of input strings, find an L-system that initially generates the sequence.

Currently, most L-systems models are found manually by experts one problem at a time, which can take a considerable amount of time and effort (Prusinkiewicz et al., 2001). Such models can then be used by domain experts to conduct experiments by simulating *in silico*, which is generally much less expensive than experimenting in the real world. Inferring an L-system algorithmically would be possibly much quicker, hopefully scaling to the creation of thousands of new models in custom scenarios from images over time.

There are many different types of L-systems; however, this work is focused on deterministic context-free L-systems (D0L-systems). A D0L-system is an ordered tuple $G = (V, \omega, P)$, consisting of an alphabet V, an axiom ω (a word using letters of V), and a finite set of productions P. A production (sometimes called a rewriting rule) is of the form $A \rightarrow u$, where $A \in V$ (called the predecessor) and u is a word over V (called the successor of A). In a deterministic L-system, only one successor exists for each predecessor, denoted by $succ(A)$. A derivation step, denoted by $\Rightarrow$, involves replacing every symbol in a string with its successor. In a context-free L-system, rewriting rules are applied solely based on each symbol, i.e. without considering neighbouring symbols. Usually, derivations start from the axiom, i.e. $\omega \Rightarrow \omega_1 \Rightarrow \omega_2 \Rightarrow \cdots$.

L-systems are often used for modeling, by interpreting some of the symbols in the alphabet as instructions for drawing an image, and each string as a sequence of instructions, and then sequences of strings correspond to a process. One common approach for visualization is the *turtle graphics* interpretation (Prusinkiewicz and Lindenmayer, 1990). Conceptually, a virtual turtle has a state consisting of coordinates in 2D or 3D Euclidian space, and an orientation, and the symbols can adjust these state variables. When the turtle's coordinates change, it may optionally draw a line between the new and old coordinates. For 2D systems, the turtle graphics are to turn left ($+$) or right ($-$) by a fixed angle, and move forward a fixed distance while drawing (F) or not drawing (f). Additional symbols for 3D allow for yaw, pitch, and rotational control (Prusinkiewicz and Lindenmayer, 1990). For processes that need branching, the symbols [and] start and end a branch by pushing and popping the turtle's state respectively. Other non-graphical symbols can also be used.

Several algorithmic approaches for inferring L-systems exist, and are surveyed in (Ben-Naoum, 2009). In (Mock,

*This research was supported in part by a grant from the Plant Phenotyping and Imaging Research Centre.

1998), L-systems are created that are based on aesthetics. In (Jacob, 1995), they are evaluated based on certain metrics (e.g., how many blooms and leaves exist). These types of methods do not infer an L-systems for a specific sequence of input strings, i.e. for a specific problem. Some methods algorithmically infer an L-system based on a sequence of inputs strings; e.g, (Nakano and Yamada, 2010; Bernard and McQuillan, 2018; Runqiang et al., 2002). This type of method is the focus of this research. It requires no ongoing interaction with the user, no a priori knowledge, and only a sequence of strings. This can be seen as an intermediate step towards inferring from a sequence of images, which has been attempted in a preliminary fashion by (Runqiang et al., 2002). They use a genetic algorithm (GA) to infer an L-system from an input image by using image processing techniques for their fitness function to match the resulting image from the candidate projected to the input image. This method was found to be 100% successful at inferring three variants of the "Fractal Plant" L-system (Prusinkiewicz and Lindenmayer, 1990) that have one symbol, and 66% successful the other three variants that have two symbols.

The Plant Model Inference Tool (PMIT) (Bernard and McQuillan, 2018) used a GA to search for an L-system compatible with a sequence of input strings after using heuristics based on necessary conditions to reduce the search space. It was successful compared to other existing tools that infer from strings (Runqiang et al., 2002; Nakano and Yamada, 2010) and could infer all systems tested where the total number of symbols added across all successors was at most 140, and those required a maximum of approximately 300 seconds. However, larger systems could not be solved in 4 hours of computation time. In all, PMIT solved 15 of 28 previously developed L-systems. By comparison, LGIN (Nakano and Yamada, 2010) attempts to infer an L-system from a single string. LGIN was evaluated on six "Fractal Plant" variants and was found to be able to infer all of them in, at most, a few seconds. Three of these variants have a one symbol alphabet and three have a two symbol alphabet. In their paper, they call the two symbol case "immensely complicated" (Nakano and Yamada, 2010), and they did not evaluate LGIN on any larger alphabets.

Here, a new method called PMIT-D0L is created that also uses a genetic algorithm with a different encoding scheme based on successor lengths instead of an ordered sequence of symbols. As before, heuristics (both new and some previously described in (Bernard and McQuillan, 2018)) are used to reduce the search space. Finally, linear diophantine equations are used to eliminate some possible length combinations, drastically reducing the overall search space. With these modifications, PMIT-D0L is able to infer all of the D0L-systems with 100% success rate in the testbed of 28 previously existing systems. These include systems with up to 31 symbols in the alphabet and a sum of successors of 282, and all are solved in at most 3.192 seconds. Hence, this is a significant improvement from previous approaches. In particular, 13 of 28 L-systems tested were immediately inferred in approximately 1 millisecond from the diophantine equations without the need for heuristics or searching.

Background

This section begins with some notation used throughout the paper. An alphabet, V, is a finite set of symbols. A word, ω, over V is any sequence of symbols $a_1 a_2 \cdots a_n$, $a_i \in V$, $1 \leq i \leq n$. The length of a word ω is denoted as $|\omega|$; further, the number of occurrences of letter a in ω is denoted as $|\omega|_a$. The set of words over V is denoted as V^*.

Given two words $x, y \in V^*$, x is a substring of y if $y = uxv$, for some $u, v \in V^*$; additionally, 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.

A genetic algorithm (GA) is an optimization algorithm described here only informally; see Back (Bäck, 1996) for definitions and more explanation. A GA is based on concepts from evolutionary biology, which states that the genes from parents are intermixed in offspring, and over subsequent generations the "species" will become more fit to their environment. With a GA, an encoding scheme is used to represent a solution to a problem as a set of values called a genome consisting of genes. Each gene represents a component of the solution to the problem, and the gene's value is mapped to an option for that component, and so is decoded in a problem specific fashion. For this work, a literal encoding is used, which means each gene's value directly represents an option in the problem solution (Bäck, 1996). Furthermore, PMIT-D0L uses the following simple standard operators for processing steps of the GA: roulette wheel selection, uniform crossover, uniform mutation, and elite survival (Bäck, 1996). The crossover and mutation steps are controlled by the weight parameters.

There have been various approaches towards inductive inference (Ben-Naoum, 2009). A technique used here is that of Doucet (Doucet, 1974), who recognized an algebraic relationship between the productions and the words generated by an L-system, that can be described as a set of linear equations. If $\omega_{i-1} \Rightarrow \omega_i$, then for every $A \in V$, $|\omega_i|_A$ is equal to the sum of the number of As produced by every symbol in ω_{i-1}. For example, if it is known that $ABA \Rightarrow ABABBBABA$, then $2 \times |succ(A)|_A + 1 \times |succ(B)|_A = 4$. Repeating this for every derivation step will give a set of linear equations. Given an alphabet of n letters, and a sequence of words $\varrho = (\omega_1, \ldots, \omega_m)$, consider the set of linear equations represented as a matrix equation

$$YM = Z, \tag{1}$$

where position (i, j) of Y is the number of the j^{th} letter in ω_{i-1}, and position (i, j) of Z is the number of the j^{th} letter in ω_i. For a D0L-system G, the matrix $M(G)$ where

entry (i, j) is the number of the j^{th} symbol in the successor of the i^{th} symbol, is called the *growth matrix* of G. If one is only given the sequence ϱ and not the grammar generating ϱ, then Y and Z are known while $M(G)$ is not. However, the growth matrix $M(G)$ of any D0L-system G generating ϱ is a solution to Equation 1. In general though, there can be additional solutions to Equation 1 that are not growth matrices of D0L-systems generating ϱ. If the matrix Y is invertible, then $M = Y^{-1}Z$ (substituting $M(G)$ for M is a solution, following the behavior of matrix multiplication). This process is described in more detail in (McQuillan et al., 2018) where inductive inference is shown to be solvable in polynomial time when Y is invertible. Furthermore, even in cases where Y is not invertible, one can use solutions to linear diophantine equations to obtain solutions. For example, if $ABA \Rightarrow ABABBBABA \Rightarrow ABABBBABABBBBBBBBBABABBBABA$, then the matrices would be appear as in Equation 2:

$$\begin{bmatrix} 2 & 1 \\ 4 & 5 \end{bmatrix} \times M = \begin{bmatrix} 4 & 5 \\ 8 & 19 \end{bmatrix} \tag{2}$$

where the growth matrix is a solution for M. The possible values in the growth matrix can be solved for by, for example, Gauss-Jordan Elimination. Each possible solution for M can be tested against the input strings to see if they represent the growth matrix of a D0L-system, and if so, it represents a compatible solution to the input strings. Doucet did not implement his method.

LGIN (Nakano and Yamada, 2010) uses a similar approach by creating equations describing the growth relationships for each symbol in the alphabet. LGIN then exhaustively tries to find the appropriate successor(s) to describe a single input string that satisfies the growth equations.

Inferring D0L-Systems

This section describes the process and techniques used by PMIT-D0L to infer D0L-systems. Conceptually, PMIT-D0L works by logically deducing three categories of facts about the successors of the (hidden) L-system to aid in a search for the L-system. First, for each $A \in V$, lower and upper bounds on $|succ(A)|$ are determined, called A_{min} and A_{max} respectively. Second, lower and upper bounds on $|succ(A)|_B$, are denoted by $(A, B)_{min}$ and $(A, B)_{max}$ respectively. These lower and upper bounds are treated as programming variables in this paper, so their values are updated as PMIT-D0L runs if an improved value is found. PMIT-D0L assumes that the D0L-systems are non-erasing so initially $A_{min} = 1$ for each $A \in V$. Furthermore, it is assumed that each turtle graphics symbol (T) except "F" has an identity production, e.g. + $\rightarrow$ +. Thus, $(T, T)_{min} = (T, T)_{max} = 1$ and $(T, A)_{min} = (T, A)_{max} = 0$ for each $A \in V, A \neq T$. For the "F" symbol, if it has an identity production this is assumed to be known and the previous statement applies; otherwise, it is treated as a non-graphical symbol for the purposes of inferring the L-system (it is also tested without this assumption). Additionally, branching symbols are paired in all successors, e.g. there is no "[" without a "]" in any successor. Third, successor relationships, of which there are five types, are defined as follows:

- A word ω is an A-complete if $\omega = succ(A)$.
- A word ω is an A-subword if ω is a subword of $succ(A)$.
- A word ω is an A-prefix if ω is a prefix of $succ(A)$.
- A word ω is an A-suffix if ω is a suffix of $succ(A)$.
- A word ω is an A-superstring if ω is a superstring of $succ(A)$.

Scanning for Successors

Previous attempts to infer L-systems by searching have focused on directly finding the correct symbols in the successor in the proper order (Mock, 1998; Runqiang et al., 2002; Bernard and McQuillan, 2018). Although intuitive, this approach is inefficient for two reasons. First, from an encoding perspective, every additional symbol in the successors requires another gene; thereby, causing the solution space to grow very quickly for even fairly short successors. Second, the search will find and assess many solutions that are not possible. For example, consider the strings $\omega_1 = ABA$ and $\omega_2 = ABABBBABA$ such that $\omega_1 \Rightarrow \omega_2$. With such strings, considering a solution with $A \rightarrow AAA$ is inefficient as the subword AAA does not exist in ω_2. Approaches that try to directly construct the successor as an ordered sequence of symbols struggle to avoid considering such successors. Logically, if from a subword u, $u \Rightarrow v$, and $|u|_A > 0$, then $succ(A)$ must be a subword of v. For this paper, the approach used to find successors is by searching for the lengths of $succ(A)$, i.e. $|succ(A)|$, for every $A \in V$ and then select a subword from the input strings of length $|succ(A)|$ for the first A encountered in the input strings.

Transforming a set of successor lengths into a set of successors (and hence the L-system) is indeed straightforward and efficient (McQuillan et al., 2018). It is done by scanning each pair of consecutive words $\omega_{i-1} = a_1 a_2 \cdots a_{|\omega_{i-1}|}$ and $\omega_i = b_1 b_2 \cdots b_{|\omega_i|}$ from left-to-right. If one possibility is $|succ(a_1)| = 3$, and $|succ(a_2)| = 2$, then this implies that $succ(a_1) = b_1 b_2 b_3$ and $succ(a_2) = b_4 b_5$. Proceeding under these assumptions finds the corresponding successor of each symbol, or finds an incompatibility. Hence, the goal is to search the space of possible successor lengths to find an L-system compatible with a sequence of strings.

The process flow for searching for a set of successor lengths is to first execute a series of heuristics (discussed next). The heuristics deduce facts about the successors relating to the upper and lower bounds of $|succ(A)|$, $|succ(A)|_B$ and finding A-subwords and A-superstrings of the successors. The heuristics are run in a loop until they no longer produce any new information. Afterwards, the problem is

divided into sub-problems by solving for the non-graphical symbols and then adding in graphical symbols one at a time. Finally, each sub-problem is solved by Gauss-Jordan elimination either providing a unique solution for the *length matrix*, or allowing for searching for a set of successor lengths that satisfy the diophantine equations.

Heuristics and Forming Independent Problems

If there is a solution to a D0L-system inference problem, then it must exist in the space of all possible D0L-systems. However, since this space is extremely large, heuristics based on necessary conditions are used to reduce its size. Three categories of heuristics are used to minimize the difference between A_{min} and A_{max} for every $A \in V$, based on growth, length, and successor relationships.

A summary of the heuristics used by the earlier version of PMIT are briefly described in this paragraph as they are still used by PMIT-D0L, with a detailed description available in (Bernard and McQuillan, 2018). The growth heuristics examine each pair of consecutive words, $\omega_{i-1} \Rightarrow \omega_i$, and counts the number of times A appears in ω_{i-1} and the number of times B appears in ω_i for every $A, B \in V$. By dividing, this gives a maximum number an upper bound on $|succ(A)_B|$. Then by assuming every symbol except one, B, produces their maximum, a minimum number of symbols B produced by A can be deduced. For the first non-turtle graphic symbol in each string, the successor relationship heuristic finds an A-prefix by using the first A_{min} symbols from the next string. Similarly, it is possible to find an A-suffix using the last non-graphical symbol, and A-superstrings using A_{max}.

More detail is given to the last technique of breaking down the inference process into independent sub-problems since it plays a significant role in the new heuristic using solutions (described later). A projection of a word $\omega \in V^*$ to a smaller alphabet $V' \subset V$ keeps all letters of $V^{'}$ and erases those in $V - V^{'}$. To determine the successor of each symbol A, first determine $succ(A)$ projected to the non-graphical symbols, i.e. find the successors consisting only of the non-graphical symbols. Then it is possible to independently determine where each graphical symbol should be placed. For example, if $V = \{A, B, C, [,], +, -\}$, then the first problem is to find each successor of A, B, C projected to $V^{'} = \{A, B, C\}$. Then there would be an independent problem for adding [and] together (as they exist in pairs) into the successors, then one for +, and -. This simplifies the inference problem by making the difference between A_{min} and A_{max} lower. Although more searches are needed, one for each sub-problem, they are each in a smaller search space and so decreases the size overall.

Symbols as Markers Any symbol for which the successor is known can drastically reduce possibilities for neighbouring symbols. Conceptually the idea is to line up every symbol A that has a known successor in a word ω_{i-1} with its successor as derived in ω_i. To illustrate this, Equation 3 shows a simple example. Since the + only ever derives the + symbol, it can be seen that the + symbol in ω_1 must produce the + in ω_2. This allows ω_1 to be separated into two parts, $\omega_{1,1} = A$ and $\omega_{1,2} = B$, and ω_2 into $\omega_{2,1} = ABA$ and $\omega_{2,2} = BBB$. Since $\omega_{1,1} \Rightarrow \omega_{2,1}$, this implies that $A \rightarrow ABA$ and $B \rightarrow BBB$ are productions.

$$\begin{array}{l} \omega_1\text{: } A{+}B \\ \qquad\quad \big\downarrow \\ \omega_2\text{: } \underbrace{ABA}_{\text{succ(A)}} + \underbrace{BBB}_{\text{succ(B)}} \end{array} \tag{3}$$

However, it is unusual for a single symbol to uniquely associate with one position of the next word except for any turtle graphics symbols at the beginning or end of the word. More commonly, there are a set of possible matches. For example if ω_2 had contained two +'s, $\omega_2 = \omega_{2,1}$+$\omega_{2,2}$+$\omega_{2,3}$, then either the + in ω_1 associates with the first + in ω_2, which would imply that $\omega_{1,1} \Rightarrow \omega_{2,1}$ and $\omega_{1,2} \Rightarrow \omega_{2,2}$+$\omega_{2,3}$, or the + associates with the second + in ω_2, which implies $\omega_{1,1} \Rightarrow \omega_{2,1}$+$\omega_{2,2}$ and $\omega_{1,2} \Rightarrow \omega_{2,3}$. The list of possible associations between a position of one word and a position of the next word is referred to as a *marker map*. If a position can be uniquely associated to a position in the next string, then this association is referred to as a *marker*. Both individual positions and sequences of positions are considered. In an example like, A[+B][-B]A[+[-C]][-[+D]], the individual symbols [,], +, and - alone might not uniquely associate; however, a sequence of symbols such as][- or]][-[+ are much more likely appear less often, drastically simplifying the problem. Hence, a *marker map* is built between each pair of consecutive strings, for every symbol that has a known successor referred to as a *candidate marker*. Of note, although graphical symbols are used in the example, non-graphical symbols may be used so long as their successor is known. Indeed, non-graphical symbols often make excellent markers as their successors tend to be more distinctive. Mapping a candidate marker onto its successor takes into account that a number of symbols N must be reserved for the successors of any symbols that follow the 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 $B_{min}+C_{min}+\text{-}_{min} = 3$ 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, after reserving symbols a candidate marker cannot be associated uniquely, then it is removed from the marker map.

Consider a pair $\omega_i = \omega_{i,1}A_1\omega_{i,2}\cdots A_n\omega_{i,n+1}$, and $\omega_{i+1} = \omega_{i+1,1}succ(A_1)\omega_{i+1,2}\cdots succ(A_n)\omega_{i+1,n+1}$, each A_j in ω_i is a marker which is associated to the annotated

successor in ω_{i+1}. It follows that $\omega_{i,j} \Rightarrow \omega_{i+1,j}$ for all j, $1 \leq j \leq n+1$. The process is then repeated for all j subwords, e.g. $\omega_{i,1} \Rightarrow \omega_{i+1,1}$. This process terminates when for some $\omega_{i,j}$ has only one symbol A with an unknown successor an A-complete is found, or, if no marker is found, then an A-prefix, A-suffix, and A-superstring may be found for the first and last symbols in the subword.

Successor Relationships from Projected Solutions As previously described, PMIT-D0L breaks down inference of successors first by using a projection to non-graphical symbols and then adding in remaining symbols one at a time. Let a solution to one of these sub-problems be called a *projected solution*, as it partly describes the final successors. After each sub-problem, each pair of consecutive strings ($\omega_{i-1} \Rightarrow \omega_i$) can be scanned using the *projected solution* to build successor relationships for the next sub-problem. This works very similarly to the marker process described above. Every symbol in ω_{i-1} is associated with its successor from the projected solution. If a symbol can be associated to only one successor location, then the symbol instance is called *certain*; otherwise, the instance is said to be *uncertain*. When an instance is uncertain, the match that results in the shortest successor can be assumed for the remaining subwords to be shown as A-subwords. Every non-graphical symbol must produce every symbol between beginning and end of its associated positions. If the symbols before and after are certain, then an A-prefix, A-suffix, or an A-complete can be found.

For example, assume that the first sub-problem is to solve the successors projected onto the non-graphical symbols resulting in the projected successor of A as ABA and of B as BA. Equations (4) to (6) show an example of the process of finding successor relationships from this projected solution. In Equation 4, 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). So, + is associated such the shortest successor for A is produced, which is to assume the + produces the first + of the pair. Then since from the projected solution of A ABA, $succ(A)$ must contain everything between the ABA, which is A[+B]A (α in Equation 4) and this is an A-prefix due to the uncertainity of the + symbol. In Equation 5, the + production is still uncertain so the association used is that which produces the shortest successor for B; i.e., that the + produces the second + of the pair allowing a B-subword to be identified. With respect to the -, at first glance it may appear uncertain; however, from the projected solution of B as BA, the - cannot produce the - between the B and [+A]. The subword B-[+A] (β) is a B-suffix due to the uncertainty of the preceding symbol. Finally, shown in Equation 6, 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 A-complete. Note, that this A-complete now makes the production of the first $+$ certain (in Equation 4), since it is now known that A did not produce it. Therefore the B-complete +B-[+A] could be produced. Since all of the heuristics together are executed in a loop until no new information is found, this would be found on the next pass.

$$\omega_1{:}A{+}B{-}A$$
$$\omega_2{:}\underbrace{A[{+}B]A}_{\alpha}{+}{+}B{-}[{+}A]{-}A[{+}B]A \tag{4}$$

$$\omega_1{:}A{+}B{-}A$$
$$\omega_2{:}A[{+}B]A{+}{+}\underbrace{B{-}[{+}A]}_{\beta}{-}A[{+}B]A \tag{5}$$

$$\omega_1{:}A{+}B{-}A$$
$$\omega_2{:}A[{+}B]A{+}{+}B{-}[{+}A]\underbrace{A[{+}B]A}_{\text{succ(A)}} \tag{6}$$

Diophantine Equations

As discussed in the previous section, Doucet (Doucet, 1974) recognized that the productions could be represented as a matrix equation, so as a step towards improving PMIT-D0L, a similar approach was implemented. In Doucet's original work, he solves for a growth matrix in Equation 1. In this equation, $M(G)$ can be replaced with the length of each production, called the *successor length matrix*, and Z replaced with the length of each word after the first. In the cases where Y is invertible, $Y^{-1}Z$ is still the unique solution, and the lengths of the successors are sufficient to assess compatibility and find the L-system.

Gauss-Jordan elimination cannot guarantee a unique solution for each successor length, often resulting in a set of linear diophantine equations, where the successor lengths are the variables, e.g. $5 \times X_1 + 3 \times X_2 = 24$. Each successor length only gets substituted for variables that appear in exactly one equation. When Gauss-Jordan elimination does not uniquely determine values for the variables 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 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. Using the equations means that only possible solutions need be checked as opposed to simply iterating over all possible length, thereby reducing the search space size. So, the value of the gene is dynamically changed, if the current value would result in a non-solution to the equations. 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.

Methodology

Data

The test suite used to evaluate PMIT-D0L consists of known L-systems and generated L-systems. Twenty-eight known D0L-systems were taken from the "virtual laboratory" (University of Calgary, 2017), which consists of 16 fractals (including the six "Fractal Plant" variants used to evaluate LGIN (Nakano and Yamada, 2010)), "Fibonnaci Bush" (a non-species specific but realistic 3D bush model), 10 algaes, and "Apple Twig with Blossoms". This test suite alone is larger than those used in literature, which tend to focus on a subset of the six "Fractal Plant" variants (Nakano and Yamada, 2010; Runqiang et al., 2002). These L-systems range from 2 to 32 symbols, with a sum of successor lengths from 3 to 282. To infer an L-system with a K letter alphabet, $K + 1$ strings were used as input.

Performance Metrics

PMIT-D0L is evaluated using two performance metrics. The first metric is *success rate* (SR), the percentage of 100 executions where PMIT successfully returns a compatible D0L-system for each of the 28 known L-systems in the test suite.

The second performance metric is *mean time to solve* (MTTS), which is the average time it takes for PMIT-D0L to return either a L-system that gives the input strings as its initial sequence, or to report that no L-system could be found over the 100 executions. In order to keep the overall experimental time practical, PMIT-D0L is only allowed to execute for a maximum of four hours (14400 seconds) for an single execution. After four hours, execution is stopped and it is counted as a failure. All times were calculated on a single core of an Intel 4770 @ 3.4 GHz with 12 GB of RAM on Windows 10. MTTS is used by LGIN (Nakano and Yamada, 2010) as a performance metric.

Parameter Optimization

The ease with which a GA will search a solution is, in part, determined by the value of the control parameters; however, optimal values for the control parameters are problem specific (Bergstra and Bengio, 2012). Previous studies have found that Random Search (RS) is an effective algorithm for optimizing a GA's control parameters (Bergstra and Bengio, 2012). RS works by assessing N randomly created parameter settings, called trials, within a neighbourhood of the current best configuration. The best trial is considered the new best configuration, and the process is repeated until none of the trials is better than the best configuration. In (Bergstra and Bengio, 2012), they found that $N = 16$, is effective for optimizing the control parameters for a wide variety of problems. The control parameters for PMIT-D0L were bound based on the suggestions by Grefenstette (Grefenstette, 1986). The population size was bound from 10 to 125 in increments of 5. Crossover weight was bound from 0.6 to 0.95 in increments of 0.05. Finally, mutation weight was bound from 0.01 to 0.20 in increments of 0.01, with the additional values of 0.001 and 0.0001 permitted. The optimal parameter settings for PMIT-D0L were found to be $P = 100$, $C = 0.85$, and $M = 0.1$.

Fitness Function

The fitness function for the GA assesses a genome as follows. The genome is transformed into a D0L-system, called the candidate system, as previously described. The first input string is treated as the axiom for the candidate system. The candidate system is used to produce a number of strings equal to the number of input strings. Sequentially, starting from the first input string, it is compared symbol by symbol to the corresponding string from the candidate system. Every symbol in the same position that does not match is counted as an error. The absolute difference in length between the two compared strings is added to the error count. Since errors early will compound into later errors, a pair of strings is only checked if the total error so far is zero. Finally, the error count is divided by the total number of symbols in the input strings, giving a real value between 0 and 1. Then 1 is added to the fitness for every string that was not compared. This encourages the GA to find solutions that incrementally solve more and more generations, while trying to focus on the earliest generations first as they are generally easier to solve due to having fewer symbols.

Results

Table 1 shows the MTTS for PMIT-D0L executed in three different ways to highlight the effect of the heuristics and GA. The first column shows the MTTS for the complete algorithm, the second column shows it without using diophantine equations, and the third column uses a brute force search instead of a GA (also without using diophantine equations). In the first column, systems solved without searching, i.e. solved uniquely using the diophantine equations without the need for searching, are marked with an "*".

With respect to SR, PMIT has a 100% success rate regardless of using diophantine equations or not for the 28 D0L-systems in the testset. However, in nearly every instance, using the diophantine equations makes PMIT-D0L faster. In particular for *Dipterosiphonia* v1, *Pterocladel-*

lium and *Tenuissimum*, the MTTS is significantly faster. These three L-systems have the largest alphabets and using diophantine equations sub-divides the alphabet into smaller sub-problems. Overall, the diophantine equations are beneficial since it lowers the peak MTTS to 3.192 seconds from 1565.095 seconds. When using a brute force search, the MTTS climbs considerably and the three L-systems with the largest alphabets fail to solve in the four hour time limit. Overall, the brute force search takes much longer even when 100% successful. Brute force search is quicker for the "Fibonacci Bush" L-system, which is simply a matter of chance that the correct solution happens to come early in the search space. In comparison to LGIN, PMIT-D0L is much quicker at solving the Fractal Plant models due to the lack of any searching.

Furthermore, it is observed that, on average, when a search is needed, approximately 80% of the time taken to infer an L-system is consumed by solving the sub-problem for the non-graphical symbols. Using the projected solution to find successor relationships makes all subsequent problems generally much easier, except when a successor is composed of only turtle graphics. Even in this case, effective projected solutions can be made by solving the turtle graphics in the following order: "[", "]" (no search is required here, as these symbols must be balanced), "+", "-", any 3D symbols, "F", "f". With such an order, the "F" and "f" symbols are generally found relatively easily. This is due to using a combination of using markers and projected solutions to find successor relationships. The projected solutions process eliminates the need to find any graphical symbols in the middle of the successor, as they are always found; thereby, leaving only those at the prefix and suffix of the successor. These in turn are found by the markers process, after which only a few uncertain cases remain. Ultimately, the difference between A_{min} and A_{max} is equal to the length of the uncertain section, which is usually just a few symbols. Finally, PMIT-D0L can still infer L-systems with 100% success rate if the assumption on known successors is relaxed to only assuming the identity production of [and], and assuming that any orientation changing symbols can produce themselves or their inverse; e.g., + $\rightarrow$ + or + $\rightarrow$ - are the only successors permitted for +. This increases the average MTTS by a factor of approximately 20.

With respect to using the diophantine equations, they only found a unique solution for some of the fractal systems. Since the biological models have larger alphabets it is more likely than any pair of symbols, A, B will be mathematically related, i.e. that as A increases by X, B increases by nX. In such a case, there will be no unique solution from Gauss-Jordan elimination. Additionally, for those L-systems with smaller alphabets, e.g. *Ditria Reptans*, the successors have symbol combinations that result in ambiguous strings, i.e. there are different possible successors to describe the sequence of strings. However, as mentioned above, the diophantine equations are removing combinations of successor lengths from consideration.

Model	PMIT	GA Only	Brute Force
Algae	0.001*	0.001	0.001
Cantor Dust	0.001*	0.001	0.001
Dragon Curve	0.001	0.001	0.001
E-Curve	0.029	0.001	0.078
Fractal Plant v1	0.001*	0.001	0.001
Fractal Plant v2	0.001*	0.001	0.001
Fractal Plant v3	0.001*	0.001	0.001
Fractal Plant v4	0.001*	0.001	0.002
Fractal Plant v5	0.001*	0.001	0.002
Fractal Plant v6	0.001*	0.001	0.002
Gosper Curve	0.001*	0.001	0.026
Koch Curve	0.001*	0.001	0.001
Peano	0.221	0.052	0.945
Pythagoras Tree	0.001*	0.001	2.894
Sierpenski Triangle v1	0.001*	0.001	0.002
Sierpenski Triangle v2	0.001*	0.001	0.001
Aphanocladia	0.007	0.006	0.047
Dipterosiphonia v1	1.639	664.206	14400.0
Dipterosiphonia v2	1.199	1.212	1.077
Ditira Reptans	0.003	0.001	0.002
Ditira Zonaricola	0.007	0.012	0.011
Herpopteros	0.006	0.017	0.079
Herposiphonia	0.015	0.383	2.492
Metamorphe	2.387	1.589	10.769
Pterocladellium	3.192	751.886	14400.0
Tenuissimum	1.141	1565.095	14400.0
Apple Twig	0.970	1.348	11.186
Fibonacci Bush	0.108	1.620	0.185

Table 1: Comparison of results for PMIT-D0L complete, using GA only, and using Brute Force

Conclusion

This paper has shown a hybrid approach for inferring deterministic context-free L-systems (D0L-systems) called PMIT-D0L. This has been a long-standing problem for which existing approaches have only solved the case where $|V| \leq 2$ and a sum of successors of 20 symbols. A previous version of PMIT-D0L (Bernard and McQuillan, 2018), raised this limit to alphabets with 17 symbols and a sum of 140 symbols across all successors. This new hybrid approach raises the limit further, as PMIT-D0L can infer D0L-systems with up to 31 symbols and a sum of 282 symbols. PMIT-D0L is able to infer the L-systems in the test suite in less than 4 seconds; therefore, PMIT-D0L is a fast, reliable tool for inferring L-systems.

The results from this work also show that the most difficult problem when inferring D0L-systems is finding the

non-turtle graphics in the successors. The use of projected solutions makes finding the graphical symbols fairly simple within the limits described. This should be used as guidance for future L-system inference algorithms.

Currently, L-systems are typically crafted by experts using scientific knowledge about the process to be modeled. This is time consuming for any single problem and requires every individual problem in a domain to be investigated separately. Algorithmically inferring L-systems is quicker and for a general-purpose algorithm, like PMIT-D0L, requires only a sequence of strings observed from the hidden L-system, i.e. PMIT-D0L is domain agnostic. Such an algorithm can reveal scientific knowledge about the mechanics of a process by inferring an appropriate L-system that can then be analyzed by an expert. This has the potential to have a large impact on multiple domains where L-systems are already used, such as plant modeling (Prusinkiewicz and Lindenmayer, 1990; Watanabe et al., 2005), anatomical modeling (Galarreta-Valverde et al., 2013; Zamir, 2001), and geological modeling (Rongier et al., 2017). Also, a fast, reliable tool such as PMIT-D0L opens up the possibility of investigating modeling applications of L-systems in other domains.

The future for PMIT-D0L will focus on other types of L-systems and to explore inferring L-systems under suboptimal conditions. This work assumes that the scanning process to produce the strings does so perfectly, so methods for inferring L-systems when there are errors in the strings or strings missing completely will be investigated. Finally, PMIT-D0L should be used to infer a D0L-system from strings produced by an actual hidden L-system.

References

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press.

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

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bernard, J. and McQuillan, I. (2018). New techniques for inferring L-systems using genetic algorithm. In *Proceedings of the 8th International Conference on Bioinspired Optimization Methods and Applications*, Lecture Notes in Computer Science, pages 13–25. Springer.

Doucet, P. (1974). The syntactic inference problem for D0L-sequences. *L Systems*, pages 146–161.

Feliciangeli, H. and Herman, G. T. (1973). Algorithms for producing grammars from sample derivations: a common problem of formal language theory and developmental biology. *Journal of Computer and System Sciences*, 7(1):97–118.

Galarreta-Valverde, M. A., Macedo, M. M., Mekkaoui, C., and Jackowski, M. (2013). Three-dimensional synthetic blood vessel generation using stochastic L-systems. In *Medical Imaging: Image Processing*, page 86691I.

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

Herman, G. and Walker, A. (1972). The syntactic inference problem as applied to biological systems. *Machine Intelligence*, 7:341–356.

Jacob, C. (1995). Genetic L-system programming: breeding and evolving artificial flowers with Mathematica. In *Proceedings of the First International Mathematica Symposium*, pages 215–222.

Lindenmayer, A. (1968). Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology*, 18(3):280–315.

McQuillan, I., Bernard, J., and Prusinkiewicz, P. (2018). Algorithms for inferring context-sensitive L-systems. In *17th International Conference on Unconventional Computation and Natural Computation*, Lecture Notes in Computer Science. Springer.

Mock, K. J. (1998). 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.

Nakano, R. and Yamada, N. (2010). Number theory-based induction of deterministic context-free L-system grammar. In *International Conference on Knowledge Discovery and Information Retrieval*, pages 194–199. SCITEPRESS.

Prusinkiewicz, P. and Hanan, J. (1992). L-systems: From formalism to programming languages. In *Lindenmayer Systems*, pages 193–211. Springer.

Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Algorithimic Beauty of Plants*. Springer Verlag, New York.

Prusinkiewicz, P., Mndermann, L., Karwowski, R., and Lane, B. (2001). 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.

Rongier, G., Collon, P., and Renard, P. (2017). Stochastic simulation of channelized sedimentary bodies using a constrained L-system. *Computers & Geosciences*, 105:158–168.

Runqiang, B., Chen, P., Burrage, K., Hanan, J., Room, P., and Belward, J. (2002). 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.

University of Calgary (2017). Algorithmic Botany. `http://algorothmicbotany.org`.

Watanabe, T., Hanan, J. S., Room, P. M., Hasegawa, T., Nakagawa, H., and Takahashi, W. (2005). Rice morphogenesis and plant architecture: measurement, specification and the reconstruction of structural development by 3d architectural modelling. *Annals of Botany*, 95(7):1131–1143.

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