New Techniques for Inferring L-systems Using Genetic Algorithm

Jason Bernard(✉) and Ian McQuillan

Department of Computer Science, University of Saskatchewan, Saskatoon, Canada
jason.bernard@usask.ca, mcquillan@cs.usask.ca

Abstract. Lindenmayer systems (L-systems) are a formal grammar system that iteratively rewrites all symbols of a string, in parallel. When visualized with a graphical interpretation, the images have been particularly successful as a concise method for simulating plants. Creating L-systems to simulate a given plant manually by experts is limited by the availability of experts and time. This paper introduces the Plant Model Inference Tool (PMIT) that infers deterministic context-free L-systems from an initial sequence of strings generated by the system using a genetic algorithm. PMIT is able to infer more complex systems than existing approaches. Indeed, while existing approaches can infer D0L-Systems where the sum of production successors is 20, PMIT can infer those where the sum is 140. This was validated using a testbed of 28 known D0L-system models, in addition to models created artificially by bootstrapping larger models.

Keywords: L-systems · Inductive inference · Genetic algorithm
Plant modeling

1 Introduction

Lindenmayer systems (L-systems), introduced in [1], are a formal grammar system that produces self-similar patterns that appear frequently in nature, and especially in plants [2]. L-systems produce strings that get rewritten over time in parallel. Certain symbols can be interpreted as instructions to create sequential images, which can be visually simulated by software such as the "virtual laboratory" (vlab) [3]. Such simulations are useful as they can incorporate different geometries [2], environmental factors [4], and mechanistic controls [5], and are therefore of use to simulate and understand plants. L-systems often consist of small textual descriptions that require little storage compared to real imagery. Certainly also, they can produce a simulation extremely quickly with low cost computers in comparison to actually growing a plant.

An L-system is denoted by a tuple $G = (V, \omega, P)$, which consists of an alphabet V (a finite set of allowed symbols), an axiom ω that is a word over V, and

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

a finite set of productions, or rewriting rules, P. A deterministic context-free L-system or a D0L-system, has exactly one rule for each symbol in V of the form $A \rightarrow x$, where $A \in V$ (the predecessor) and x is a word over V (the successor, denoted by $succ(A)$). Words get rewritten according to a derivation relation, $\Rightarrow$, whereby $A_1 \cdots A_n \Rightarrow x_1 \cdots x_n$, where $A_i \in V, x_i$ is a word, and $A_i \rightarrow x_i$ is in P, for each i, $1 \leq i \leq n$. Normally, one is concerned with derivations starting at the axiom, $\omega \Rightarrow \omega_1 \Rightarrow \omega_2 \Rightarrow \cdots \Rightarrow \omega_n$. The sequence $(\omega_1, \ldots, \omega_n)$ is known as the developmental sequence of length n.

One common alphabet for visualization is the turtle graphics alphabet [2], so-called as it is imagined that each word generated contains a sequence of instructions that causes a turtle to draw an image with a pen attached. The turtle has a state consisting of a position on a (usually) 3D grid and an angle, and the common symbols that cause the turtle to change states and draw are: F (move forward with pen down), f (move forward with pen up), + (turn left), − (turn right), [(start a branch),] (end a branch), & (pitch up), $\wedge$ (pitch down), \(roll left), / (roll right), | (turn around 180°). For branching models, [causes the state to be pushed on a stack and] causes the state to be popped and the turtle switches to it. It is assumed that the right hand side of rewriting rules have parantheses that are properly nested. Additional symbols are added to the alphabet, such as A and B, to represent the underlying growth mechanics. The "Fractal Plant" L-system is inferred commonly [6,7] and so is shown here as an example: $G = (\{X, F\}, X, \{X \rightarrow F[+X]F[-X]+X, F \rightarrow FF\})$. After 7 generations, "Fractal Plant" can produce the image in Fig. 1 after 7 generations. More realistic 3D models may be produced with extensions of D0L-systems.

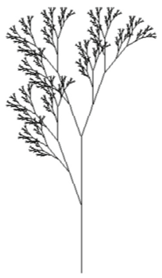

Fig. 1. Fractal plant after 7 generations [2].

A difficult challenge is to determine an L-system that can accurately simulate a plant. In practice, this often involves manual measurements over time, scientific knowledge, and is done by experts [8]. Although this approach has been successful, it does have notable drawbacks. Producing a system manually requires an expert that are in limited supply, and it does not scale to producing arbitrarily many models. Furthermore, the more complex plant models require a priori knowledge of the underlying mechanics of the plant, which are difficult and time consuming to acquire. To address this, semi-automated (used as an aide for the expert) [9,10], and fully automated approaches [6,7], have been introduced to find an L-system that matches observed data. This approach has the potential to scale to constructing thousands of models, and also has the potential to expose biomechanics rather than requiring its knowledge beforehand.

The ultimate goal of this research is to automatically determine a model from a sequence of plant images over time. An intermediate step is to infer the model from a sequence of strings used to draw the images. This is known as the inductive inference problem, defined as follows. Given a sequence of strings $\alpha = (\omega_1, \ldots, \omega_n)$, find a D0L-system, or if it exists, $G = (V, \omega, P)$ such that $\omega = \omega_0 \Rightarrow \omega_1 \Rightarrow \cdots \Rightarrow \omega_n$ where α is the developmental sequence of length n.

This paper introduces the Plant Model Inference Tool (PMIT) that aims to be a fully automated approach to inductive inference of L-systems. Towards that goal, PMIT uses a genetic algorithm (GA) to search for an L-system to match the words produced. This paper presents a different encoding scheme than previous approaches, and shows that it is more effective for inferring D0L-systems. Additionally, some logical rules based on necessary conditions are used as heuristics to shrink the solution space. Between these two techniques, it is determined that PMIT is able to infer L-systems where the sum of the production successors is approximately 140 symbols in length; whereas, other approaches are limited to about 20 symbols. Moreover, the testbed used to test PMIT is significantly larger than previous approaches. Indeed, 28 previously developed D0L-systems are used, and for these systems that PMIT properly inferred, it did so in an average of 17.762 s. Furthermore, additional (in some sense "artificial") models are created by combining the existing models where the combined length of the successors is longer than 140 symbols (which PMIT does not solve), and then randomly removing "F" symbols until it can solve them. This work can be seen as a step towards the goal of 3D scanning a plant over time, converting the images into strings that describe how to draw them, then inferring the L-system from the sequence of strings.

The remainder of this paper is structured as follows. Section 2 describes some existing automated approaches for inferring L-systems. Section 3 describes the logical rules used to shrink the solution space, and Sect. 4 discusses the genetic algorithm. Section 5 will discuss the methodology used to evaluate PMIT and the results. Finally, Sect. 6 concludes the work and discusses future directions. Some details are omitted due to space constraints, but appear online [11].

2 Background

This section briefly describes some notation used throughout the paper, contains a brief description of genetic algorithms since they are used as the search mechanism here, then describes some existing approaches to L-system inference.

An alphabet is a finite set of symbols. Given an alphabet V, a word over V is any sequence of letters written $a_1 a_2 \cdots a_n$, $a_i \in V, 1 \leq i \leq n$. The set of all words over V is denoted by V^*. Given a word $x \in V^*$, $|x|$ is the length of x, and $|x|_A$ is the number of A's in x, where $A \in V$. 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 $v \in V^*$, and x is a suffix of y if $y = ux$ for some $u \in V^*$.

The GA is an optimization algorithm, based on evolutionary principles, used to efficiently search N-dimensional (usually) bounded spaces [12]. In evolutionary biology, increasingly fit offspring are created over successive generations by intermixing the genes of parents. 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 and represents, in a problem specific way, an element of the solution to the problem. One common type of encoding is a real mapped encoding, where the genes have a real value from 0 and 1 and different ranges within are mapped contextually [12]. This encoding works best when the options at each step of the problem are unknown or dependent on prior choices.

The GA functions by first creating an initial population (P) of random solutions. Each member of the population is assessed using a problem specific fitness function. Then the GA, controlled by certain parameters, performs a selection, crossover, mutation, and survival step until a termination condition is reached. In the selection step, a set of pairs of genomes are selected from the population with odds in proportion to their fitness, i.e. preferring more fit genomes. During the crossover step, for each selected pair, a random selection of genes are copied between the two; thereby, producing two offspring. Each gene has a chance of being swapped equal to the control parameter *crossover weight*. The mutation step takes each offspring and randomly changes zero or more genes to a random value with each gene having a chance of being mutated equal to the *mutation weight*. Then each offspring is evaluated using 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). The termination condition may be based on such criteria as finding a solution with sufficient fitness, or hitting a pre-determined maximum number of generations.

Various approaches to L-system inference were surveyed in [13]. There are several different broad approaches towards the problem: building by hand [2,8], algebraic approaches [6,14], using logical rules [6], and search approaches [7]. Since PMIT is a hybrid approach incorporating a search algorithm, GA, together with logical rules to reduce intractability by shrinking the search space, this section will examine some existing logic-based and search-based approaches.

Inductive inference has been studied theoretically (without implementation) by several authors [13], e.g. Doucet [14]. He devised a method that uses solutions to Diophantine equations to, in many cases, find a D0L-system that starts by generating the input strings. A similar approach was implemented with a tool called LGIN [6] that infers L-systems from a single observed string ω. They devise a set of equations that relate the number of each symbol observed in ω to the linear combination of the production values in the growth matrix.

LGIN is limited to two symbol alphabets, which is still described as "immensely complicated" [6], and was evaluated on six variants of "Fractal Plant" [2] and had a peak execution time of four seconds.

Runqiang et al. [7] propose to infer an L-system from an image using a GA. Each gene is encoded to represent a symbol in each successor. The fitness function matches the candidate system to the observed data using image processing. Their approach is limited to an alphabet size of 2 and a maximum total length of all successors of 14. Their approach is 100% successful for a variant of "Fractal Plant" [2] with $|V| = 1$, and has a 66% success rate for a variant of "Fractal Plant" [2] with $|V| = 2$. Although they do not list timings, their GA converged after a maximum of 97 generations, which suggests a short runtime.

3 PMIT Methodology for Logically Deducing Facts About Successors

In this section, the methodology that is used by PMIT to reduce the size of the solution space with heuristics—all of which are based on necessary conditions for D0L-systems—will be described. Indeed, the success and efficiency of a search algorithm is generally tied to the size of the solution space. As all these conditions are mathematically true, this guarantees that a correct solution is in the remaining search space (if there is a D0L-system that can generate the input). In PMIT, logical rules are used to reduce the dimensional bounds in two contexts. The first context is to determine a lower bound ℓ and upper bound u on the number of each symbol B produced by each symbol A for each $A, B \in V$, henceforth called growth of B by A. Thus, two programming variables $(A, B)_{min}$ and $(A, B)_{max}$ are created that change such that $(A, B)_{min} \leq |succ(A)|_B \leq (A, B)_{max}$. A second context is a separate lower ℓ and upper bound u on the length of each successor for each $A \in V$. Then, two programming variables A_{min} and A_{max} are used such that $A_{min} \leq |succ(A)| \leq A_{max}$ and their values improve as the program runs. The bounds on growth and on lengths depend on each other, so all the rules are run in a loop until the bounds stop improving.

For this paper, it is assumed that if a turtle graphic symbol has an identity production (e.g. $+ \rightarrow +$), then this is known in advance. Typically, these symbols do have identity productions. There are some instances where "F" may not, (some variants of "Fractal Plant" [2]). In such a case, "F" is treated as a non-turtle graphics symbol for the purposes of inferring the L-system. Also, all successors are assumed to be non-empty, which are commonly used in practice when developing models [2]. This implies that A_{min} is initialized to 1 for each $A \in V$. For each turtle symbol $T \in V$, $T_{min} = T_{max} = 1$, $(T, T)_{min} = (T, T)_{max} = 1$ and $(T, A)_{min} = (T, A)_{max} = 0$ for every $A \in V, A \neq T$.

3.1 Deducing Growth

Consider input $\alpha = (\omega_0, \ldots, \omega_n)$, $\omega_i \in V^*$, $0 \leq i \leq n$ with alphabet V. Deduction of growth in PMIT is based on two mechanisms; the first being the determination of so-called *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 runs, it can determine additional successor fragments, which can help to deduce growth. 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 non-turtle graphics symbol is scanned (say, A, where the word scanned is αA). Then, in ω_2, PMIT skips over the graphical symbols in α (since each symbol in α has a known identity production), and the next A_{min} symbols, β, (the current value of the lower bound for $|succ(A)|$) must be an A-prefix fragment. Furthermore, since branching symbols must be paired and balanced 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 symbols, 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. Then, lower and upper bounds on the growth of B by A ($(A, B)_{min}$ and $(A, B)_{max}$) for each $B \in V$ can be found by counting the number of B symbols in any prefix and superstring fragments respectively and changing them if the bounds are improved. For a suffix fragment, the process is identical except from right to left starting at the end of ω_1. An example of this process appears in [11].

The second mechanism for deduction of growth is based on calculating the number of times each symbol $A \in V$ appears in word ω_i above the number implied from ω_{i-1} together with the current values of each lower bound $(B, A)_{min}$, for each $B \in V$. Formally, a programming variable for the *accounted for growth* of a symbol $A \in V$ for $1 \leq i \leq n$, denoted as $G_{acc}(i, A)$ is:

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

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

Then, $(B, A)_{max}$ 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 1 to n, where B occurs, as any of the n 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; also the floor function is used since $|succ(B)|_A$ is a positive integer) in the previous word, as computed by

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

An example is presented in [11].

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.

3.2 Deducing Successor Length

The deduction of A_{min} and A_{max} are found from two logical rules, one involving the sum of the minimum and maximum growth over all variables, and one by exploiting a technical mathematical property. The first rule simply states that A_{min} is at least the sum of $(A, B)_{min}$ for every $B \in V$ and similarly A_{max} is at most the sum of $(A, B)_{max}$ for every $B \in V$. The second rule is trickier but often improves the bounds for A_{max} and A_{min} for $A \in V$. This takes place in steps. First, the maximum number of symbols that can be produced by A in ω_i is computed by: $x := |\omega_i| - \sum_{B \in V, B \neq A}(B_{min} \cdot |\omega_{i-1}|_B)$. If $|\omega_{i-1}|_A > 0$, let:

$$A^i_{max} := \left\lfloor \frac{x}{|\omega_{i-1}|_A} \right\rfloor \tag{3}$$

if its value is improved. It follows that A_{max} can be set to $\min_{\substack{1 \leq i \leq n, \\ |\omega_{i-1}|_A > 0}} A^i_{max}$, if its value is improved. Next, now that these A^i_{max} values have been calculated, it is sometimes possible to further improve the A_{max} and A_{min} values. Let $Y^i \in V$, $1 \leq i \leq n$ be such that Y^i occurs the least frequently in ω_{i-1} with at least one copy. The current value of Y^i_{max} will be examined as computed by Eq. 3; note $Y^1, \ldots, Y^n$ can be different. Let $V^i_{max} := Y^i_{max} + \sum_{\substack{B \in V, \\ B \neq Y^i}} B_{min}$. Then, V^i_{max} can allow refinement of the upper bound for each successor, as A_{max} may be improved by assuming all other symbols produce their minimum and subtracting from V^i_{max}. Mathematically this is expressed as:

$$A_{max} := V^i_{max} - \sum_{\substack{B \in V, \\ B \neq A}} B_{min} \tag{4}$$

for $1 \leq i \leq n$, if A occurs in ω_{i-1}, and if the new value is smaller, which has the effect of the minimum over all i, $1 \leq i \leq n$. Although it is not immediately obvious that this formula is an upper bound on $|succ(A)|$, a mathematical proof has been completed (omitted due to space constraints), and appears in [11] along with an example of its use. Thus, A_{max} can be set in this fashion. Similarly, A_{min} can be set by taking Y^i that occurs most frequently.

4 Encoding for the L-system Inference Problem

In this section, the GA and encoding used by PMIT is described and contrasted with previous approaches.

The efficient search of a GA is controlled, in part, by the settings of the control parameters: population size, crossover weight, and mutation weight. The process of finding the optimal control parameter settings is called *hyperparameter search*. It was found via Random Search (details on methodology used in [11]) that the optimal parameter settings were 100 for population size, 0.85 crossover weight, and 0.10 for mutation weight. These parameters are henceforth used.

The fitness function for PMIT compares the symbols in the observed data to the symbols in the words produced by the candidate solution position by position. An error is counted if the symbols do not match or if the candidate solution is too long or short. The base fitness is the number of errors divided by total number of symbols. If the candidate solution produces more than double the number of symbols expected, it is not evaluated and assigned an extremely high fitness so that it will not pass the survival step. Since errors early on in the input words $\omega_0, \ldots, \omega_n$ will cause errors later, a word ω_i, is only assessed if there are no errors for each preceding word, and 1.0 is added to F for each unevaluated word. This encourages the GA to find solutions that incrementally match more of the observed words. PMIT is also evaluated using brute force, which is guaranteed to find the most fit solution and it was found that the solution found by the GA matches that found by brute force, showing that this fitness function is effective at finding an optimal solution.

PMIT uses three termination conditions to determine when to stop running. First, PMIT stops if a solution is found with a fitness of 0.0 as such a solution perfectly describes the observed data. Second, PMIT stops after 4 h of execution if no solution has been found. Third, PMIT stops when the population has converged and can no longer find better solutions. This is done by recording the current generation whenever a new best solution is found as Gen_{best}. If after an additional Gen_{best} generations, no better solutions are found, then PMIT terminates. To prevent PMIT from terminating early due to random chance, PMIT must perform at least 1,000 generations for the third condition only. This third condition is added to prevent the GA from becoming a random search post-convergence and finding an L-system by chance skewing the results.

The encoding scheme used most commonly in literature (e.g., [7,10]) is to have a gene represent each possible symbol in a successor. The number of genes for the approaches in literature varies due the specific method they use to decode the genome into an L-system, although they are approximately the total length of all successors combined. With this approach, each gene represents a variable from V (encoded as an integer from 1 to $|V|$). However, in some approaches (and PMIT) the decoding step needs to account for the possibility that a particular symbol in a successor *does not exist* (represented by $\oslash$). When the possibility of an $\oslash$ exists, such genes have a range from 1 to $|V| + 1$. As an example, assume $V = \{A, B\}$ and $A_{min} = 2, A_{max} = 3, B_{min} = 1, B_{max} = 3$. For A, it is certain to have at least two symbols in the successor and the third may or may not exist. So, the first three genes represent the symbols in $succ(A)$, where the first two genes have each possible values from $\{A, B\}$ and the third gene has $\{A, B, \oslash\}$.

Next the improvements made to the genomic structure defined by the basic encoding scheme will be described. Although they are discussed separately for ease of comprehension, all the improvements are used together.

PMIT uses the bounds and successor fragments to create a genomic structure. For example, if $V = \{A, B\}$, $A_{min} = 1$, and $A_{max} = 3$, then $succ(A)$ can be expressed as the genomic structure of $\{A, B\}, \{A, B, \oslash\}, \{A, B, \oslash\}$. If there is an A-prefix of B, then the genomic structure can change to $\{B\}, \{A, B, \oslash\}, \{A, B, \oslash\}$ since the first symbol in $succ(A)$ is B, essentially eliminating the need for the first gene. This is similar for an A-suffix. The second improvement to the basic encoding scheme further reduces the solution space by eliminating impossible solutions. When building the successor, PMIT first places the symbols known to be in $succ(A)$. For each successor, $\sum_{A,B \in V}(A,B)_{min}$ genes are created with a real value range between 0 and 1. Since these symbols must exist, the mapping selects an unused position within the successor. After these symbols are placed, if any additional symbols are needed up to the value of A_{max}, then PMIT selects the remainder allowing the option of using $\oslash$, ensuring that $(A,B)_{max}$ is not violated for any $A,B \in V$; i.e. the bounds computed by the heuristics shown in Sect. 3 ensure that the candidate solutions are always valid, and the genes' values are dynamically interpreted to ensure that the bounds are not violated. Further details and examples are in [11].

Lastly, it was determined that since new non-graphical symbols can only be produced by non-graphical symbols, it is possible to, at first, ignore the graphical symbols over a smaller alphabet V_{im}. Then, one can search for the successors over V_{im}^*, which is a simpler problem. For example, if $A \rightarrow F[+F]B$ and $B \rightarrow F[\text{-}F]A$, then with $V_{im} = \{A,B\}$ it is only necessary to find $A \rightarrow B$ and $B \rightarrow A$. Each graphical symbol can be added in one at a time. In the example above, the second step might add $+$ to V_{im} and find $A \rightarrow +B$ and $B \rightarrow A$. Solving these smaller problems is more efficient as the individual search spaces are smaller and when summed are smaller than the solution space when trying to find the full successor in one step. Additional details, including the use of successor fragments to further simplify the number of genes needed, are omitted and appear in [11].

5 Data, Evaluation, and Results

To evaluate PMIT's ability to infer D0L-systems, ten fractals, six plant-like fractal variants inferred by LGIN [2,6], and twelve other biological models were selected from the vlab online repository [3]. The biological models consist of ten algaes, apple twig with blossoms, and a "Fibonacci Bush". The dataset compares favourably to similar studies where only some variants of one or two models are considered [6,7]. The data set is also of greater complexity by considering models with alphabets from between 2 to 31 symbols compared to two symbol alphabets [6,7]. However, there remain gaps both in terms of successor lengths and alphabet size. Hence, additional L-systems are created by bootstrapping; that is, by combining successors from multiple L-systems to create new "fake" systems with every combination of alphabet size from 3 to 25 in increments of 2 and longest successor length from 5 to 25 in increments of 5. To get successors of the proper length some "F" symbols were trimmed from longer successors. These are called *generated L-systems.*

Two metrics are used to measure success. *Success rate* (SR) is the percentage of times PMIT can find any L-system that describes the observed data. *Mean time to solve* (MTTS) is the time taken to solve the models (measured using a single core of an Intel 4770 @ 3.4 GHz with 12 GB of RAM on Windows 10). PMIT stops execution at 4 h (14400 s) calling the search a failure, as more than this time is not practical relative to other tools in the literature.

5.1 Results

Three programs were evaluated. The first is PMIT (implemented in C++ using Windows 10), the second is a restriction of PMIT that uses a brute force algorithm without the GA or logical rules, and the existing program LGIN. No comparison is made to the work by Runqiang et al. [7] as LGIN is strictly better; indeed, LGIN is the best approach that could be found in literature making it the best algorithm to which PMIT can be compared. The comparison between brute force and GA shows the effects of using GA on MTTS. Results are shown in Table 1. No SR is shown for LGIN as it is not explicitly stated; however, it is implied to be 100% [6] for all rows where a time is

Table 1. Results for PMIT, Brute Force, and LGIN [6], on existing L-system models.

Model	PMIT			Brute force		LGIN [6]
	SR	MTTS (s)	Infer growth	SR	MTTS (s)	MTTS (s)
Algae [2]	100%	0.001	n/a	100%	0.001	-
Cantor Dust [2]	100%	0.001	n/a	100%	0.001	-
Dragon Curve [2]	100%	0.909	n/a	100%	4.181	-
E-Curve [2]	0%	14400	Yes	0%	14400	-
Fractal Plant v1 [2,6]	100%	33.680	n/a	100%	163.498	2.834
Fractal Plant v2 [2,6]	100%	0.021	n/a	100%	5.019	0.078
Fractal Plant v3 [2,6]	100%	0.023	n/a	100%	5.290	0.120
Fractal Plant v4 [2,6]	100%	0.042	n/a	100%	6.571	0.414
Fractal Plant v5 [2,6]	100%	34.952	n/a	100%	171.003	0.406
Fractal Plant v6 [2,6]	100%	31.107	n/a	100%	174.976	0.397
Gosper Curve [2]	100%	71.354	n/a	100%	921.911	-
Koch Curve [2]	100%	0.003	n/a	100%	0.023	-
Peano [2]	0%	14400	Yes	0%	14400	-
Pythagoras Tree [2]	100%	0.041	n/a	100%	2.894	-
Sierpenski Triangle v1 [2]	100%	2.628	n/a	100%	267.629	-
Sierpenski Triangle v2 [2]	100%	0.086	n/a	100%	128.043	-
Aphanocladia [3]	0%	54.044	Yes	0%	14400	-
Dipterosiphonia v1 [3]	0%	14400	No	0%	14400	-
Dipterosiphonia v2 [3]	0%	14400	Yes	0%	14400	-
Ditira Reptans [3]	100%	73.821	n/a	100%	6856.943	-
Ditira Zonaricola [3]	0%	74.006	Yes	0%	14400	-
Herpopteros [3]	0%	81.530	Yes	0%	14400	-
Herposiphonia [3]	0%	298.114	Yes	0%	14400	-
Metamorphe [3]	0%	14400	Yes	0%	14400	-
Pterocladellium [3]	0%	14400	No	0%	14400	-
Tenuissimum [3]	0%	14400	No	0%	14400	-
Apple Twig [3]	0%	14400	No	0%	14400	-
Fibonacci Bush [3]	0%	14400	Yes	0%	14400	-

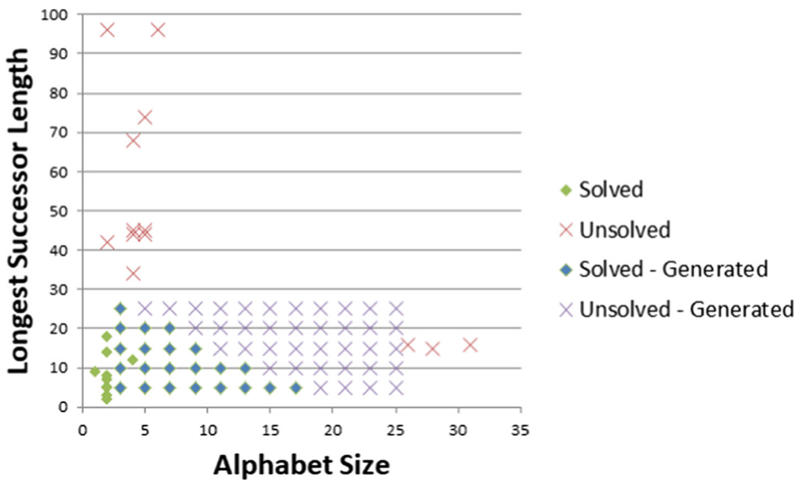

Fig. 2. L-systems solved with 100% SR by alphabet size and longest successor length.

written. The variants used by LGIN [2,6] are the six Fractal Plants. In general, PMIT is fairly successful at solving the fractals, the "Fractal Plant" variants, and also *Ditria reptans*. The success rates are all either 0% or 100%, indicating that a problem is either solved or not. It was observed that PMIT was able to solve many other models excluding the F and f symbols, as indicated in the "Infer Growth" column. For example, PMIT inferred for *Aphanocladia* that $A \rightarrow BA$, $B \rightarrow U[-C]UU[+/C/]U$; however, it was not able to then infer $C \rightarrow FFfFFfFFfFF[\text{-}F^4]fFFfFF[+F^3]fFFfFF[\text{-}FF]fFFf$. This is interesting as the growth mechanisms might be more complicated for a human to infer than the lines represented by the F and f symbols. Therefore, PMIT is a useful aide to human experts even when it cannot infer the complete L-system.

For the generated models, Fig. 2 gives one point for every L-system (generated or not) tested with PMIT. A model is considered *solved* if there is a 100% success rate and *unsolved* otherwise. It is evident that the figure shows a region described by alphabet size and longest successor length that PMIT can reliably solve. PMIT can infer L-systems with $|V| = 17$, if the successors are short (5) and can infer fairly long successors (25) when $|V| = 3$. Computing the *sum of successor words* $\sum_{A \in V} |succ(A)|$, then PMIT is able to infer L-systems where such a sum is less than 140, which compares favorably to approaches in literature where the sum is at most 20. Overall, in terms of MTTS, PMIT is generally slower than LGIN [6] for $|V| = 2$ although is still practically fast for these L-systems (less than 35 s); however, PMIT can reliably infer L-systems with larger alphabet sizes and successor lengths and still does so with an average of 17.762 s. Finally, the brute force algorithm required a MTTS of 621.998 s. Hence, the logical rules and the GA provide considerable improvement.

6 Conclusions and Future Directions

This paper introduced the Plant Model Inference Tool (PMIT) as a hybrid approach, combining GA and logical rules, to infer deterministic context-free L-systems. PMIT can infer systems where the sum of the successor lengths is less than or equal to 140 symbols. This compares favourably to existing approaches that are limited to one or two symbol alphabets, and a total successor length less than or equal to 20 [6,7]. Although PMIT is slower than existing approaches for "Fractal Plant" which has a small (2) alphabet [6,7] with a MTTS of 35 s or less compared to 2 s or less, PMIT is still practically fast. Furthermore, existing approaches are limited to 2 symbol alphabets while PMIT can infer some L-systems with up to 17 symbol alphabets with longer successors.

For future work, methods will be investigated to further extend the limits of alphabet size and successor length. Also, a main focus will be on the ability to properly infer the drawing pattern likely using image processing techniques, perhaps taking advantage of techniques devised here to sub-divide alphabets.

References

1. Lindenmayer, A.: Mathematical models for cellular interaction in development, parts I and II. J. Theor. Biol. **18**, 280–315 (1968)
2. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer, New York (1990). https://doi.org/10.1007/978-1-4613-8476-2
3. University of Calgary: Algorithmic Botany
4. Allen, M.T., Prusinkiewicz, P., DeJong, T.M.: Using L-systems for modeling source-sink interactions, architecture and physiology of growing trees: the L-PEACH model. New Phytol. **166**(3), 869–880 (2005)
5. Prusinkiewicz, P., Crawford, S., Smith, R., Ljung, K., Bennet, T., Ongaro, V., Leyser, O.: Control of bud activation by an auxin transport switch. Proc. Nat. Acad. Sci. **106**(41), 17431–17436 (2009)
6. Nakano, R., Yamada, N.: Number theory-based induction of deterministic context-free L-system grammar. In: International Conference on Knowledge Discovery and Information Retrieval, pp. 194–199. SCITEPRESS (2010)
7. Runqiang, B., Chen, P., Burrage, K., Hanan, J., Room, P., Belward, J.: Derivation of L-system models from measurements of biological branching structures using genetic algorithms. In: Hendtlass, T., Ali, M. (eds.) IEA/AIE 2002. LNCS (LNAI), vol. 2358, pp. 514–524. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-48035-8_50
8. Prusinkiewicz, P., Mündermann, L., Karwowski, R., Lane, B.: The use of positional information in the modeling of plants. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 289–300. ACM (2001)
9. Jacob, C.: Genetic L-system programming: breeding and evolving artificial flowers with Mathematica. In: Proceedings of the First International Mathematica Symposium, pp. 215–222 (1995)
10. Mock, K.J.: Wildwood: the evolution of L-system plants for virtual environments. In: Proceedings of the 1998 IEEE World Congress on Computational Intelligence, pp. 476–480. IEEE (1998)

11. Bernard, J., McQuillan, I.: New techniques for inferring L-systems using genetic algorithm (2017). https://arxiv.org/abs/1712.00180
12. Back, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford (1996)
13. Ben-Naoum, F.: A survey on L-system inference. INFOCOMP J. Comput. Sci. **8**(3), 29–39 (2009)
14. Doucet, P.G.: The syntactic inference problem for DOL-sequences. In: Rozenberg, G., Salomaa, A. (eds.) L Systems. LNCS, vol. 15, pp. 146–161. Springer, Heidelberg (1974). https://doi.org/10.1007/3-540-06867-8_12