Inferring Stochastic L-systems Using a Hybrid Greedy Algorithm

* **

Jason Bernard
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
jason.bernard@usask.ca

Ian McQuillan
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
mcquillan@cs.usask.ca

***Abstract*—Stochastic context-free Lindenmayer systems (S0L-systems) are a formal grammar system that produce sequences of strings based on parallel rewriting rules over a probability distribution. The resulting words can be treated as symbolic instructions to create visual models by simulation software. S0L-system have been used to model different natural and engineered processes. One issue with S0L-systems is the difficulty in determining an S0L-systems to model a process. Current approaches either infer S0L-systems based on aesthetics or rely on a priori expert knowledge. This work introduces PMIT-S0L, a tool for inferring S0L-systems from a sequence of strings generated by a (hidden) L-system, using a greedy algorithm hybridized with search algorithms. PMIT-S0L was evaluated using 3600 procedurally generated S0L-systems and is able to infer the test set with 100% success so long as there are 12 or less rewriting rules in total in the L-system. This makes PMIT-S0L applicable for many practical applications.**

***Index Terms*—Lindenmayer Systems, Stochastic L-systems, Inductive Inference, Plant Modeling, Natural Process Modeling**

I. Introduction

Lindenmayer systems (L-systems) [1] are a well-known mechanism for modeling processes. They have been particularly successful at modeling plants [2, 3], but have also been applied to modeling other biological processes [4, 5, 6], and geological processes [7]. Modeling a process in this manner can allow for an improved understanding of the process' underlying mechanisms. Using plant modeling as an example, understanding the algorithm of how plants grow may allow plant physiologists to produce plants more suited to a variety of different environmental factors [2, 8]. In addition, simulating *in silico* saves time, money, and can even help improve crop yields [8].

This research was undertaken thanks in part to funding from the Canada First Research Excellence Fund and National Science, and from the Engineering Research Council.

Due to the importance of models, it is desirable to try to automate their creation rather than the current practice of experts creating them manually [5, 9]. This manual process has been described as requiring "tedious and intricate handwork" [5] that could be improved by an algorithm to "infer rules and parameters automatically from real ...images" [5]. Indeed, as an example, 3D imagery can be captured for a plant over time which could then be converted into sequences of symbolic instructions for a simulation software to reproduce the imagery. The final step is to infer the L-system that produces the sequence of strings, which is called the inductive inference problem and is the focus of this research.

L-systems [1] are a formal grammar system. Context-free L-systems (0L-systems) are defined by an alphabet V (a set of symbols), an axiom (a word over V), and a finite set of rewriting rules (or productions) P. Each rule specifies that a symbol in a string can be replaced by a string. For example, the rule $A \rightarrow AB$ states that any symbol A in a string can be replaced with an AB. In this example, A is called the predecessor and AB is the successor. A derivation step (denoted by $\Rightarrow$) involves rewriting in parallel each letter in a string using some rule with that letter as the predecessor and replacing it with the successor. When derivation steps are done iteratively, the resulting sequence of strings may then be interpreted as instructions, usually visualized, for simulation software such as the "virtual labratory" (vlab) [10], to act as a model for a process over time. If a 0L-system has a single replacement rule for each symbol in the alphabet, then it is called a deterministic context-free L-systems (D0L-system) [2]. A stochastic context-free L-system (S0L-system) is a 0L-system, where each rewriting rule has an associated probability [2, 3, 4, 5, 7]. To continue the previous example, the rules $A \rightarrow AB : 60\%$ and $A \rightarrow BB : 40\%$ indicates that for every copy of the symbol A in a word, there is a 60% probability of replacing it with AB and a 40% probability of replacing it with BB. With D0L-systems, the sequence of strings is well-defined for the axiom and the set of rules. For S0L-systems, different strings may result from different applications of the rules to the axiom.

For many processes where multiple outcomes are possible, a stochastic model is the most appropriate. As an example, for plant modeling, S0L-systems have been used as a mechanism to convert the stem of a plant into the apex [2, 3]. In this case, say a symbol S represents the stem and has two successors. One successor adds stem growth and other structural components, e.g. branches, with a probability p_1. The other successor has a symbol for the apex, e.g. A, with probability p_2. For flowering plants, the apex then has a successor to create the flower. All of the critical elements, the successors for S and A, and the probabilities for p_1 and p_2 are created by hand either experimentally by considering the aesthetics of the resulting system or using a priori knowledge of the modeled plant [2, 3]. Also, S0L-systems were used by Mech [3] to model the growth of trees under different environmental factors. Some examples outside of plant modeling, include the use of stochastic L-systems and a priori knowledge of environmental modifiers to model secondary structure of protein folds [4], and using stochastic rules to produce curves similar to facies' segment [7].

Another interesting potential use of stochastic models, is the ability to use the large number of output images to train computer vision problems. This is similar to the approach used in [11], where they used many images from an L-system model to augment a limited number of real images to train a deep learning network for the purposes of leaf counting. This worked better than using real images alone. Thus, stochastic L-systems and automatic inference of them are of interest towards this goal.

Given a sequence of input strings $\omega_0, \ldots, \omega_n$, the challenge is to find the "best" S0L-system that will initially generate those strings in order. Although there has been some research on inferring D0L-systems in this fashion [12, 13, 14, 15, 16], with [12] providing a recent successful approach as the authors were able to infer 28 of 28 D0L-systems correctly, there has been less work on inferring S0L-systems. Thus, it is now desirable to attempt to infer the strictly more difficult S0L-systems. Notice that this problem is more complicated as there is never a unique solution (some of the other challenges to inferring S0L-systems will be discussed in Section II).

This paper introduces the Plant Model Inference Tool for Stochastic Context-Free L-systems (PMIT-S0L) that infers S0L-systems based on a single observed sequence of strings. It is implemented in a domain agnostic fashion, and is not strictly tied to plant modeling. This paper presents an evaluation for four different implementations of PMIT-S0L, all of which are based conceptually on a greedy decision process. The four algorithms are: greedy algorithm, random forest, greedy algorithm hybridized with brute force search, and greedy algorithm hybridized with genetic algorithm. Although S0L-systems are used frequently in the literature [2, 3, 4, 5, 6, 7], specific S0L-systems are rarely published. So although it would be preferred to use a test set of known S0L-systems, PMIT-S0L is evaluated using procedurally generated S0L-systems with up to 9 symbols and up to 14 rewriting rules that are designed to be similar to existing known L-systems. A further challenge is defining a useful metric of success. For D0L-systems, success may be defined as finding an L-system that generates the strings (within a practical execution time). However, with S0L-systems there is a large number of L-systems that could generate the strings, so compatibility of an S0L-system to a sequence of strings is not so clear. Described in greater detail in Section II, for this paper, the argument is made that one mechanism of measuring success is to find an S0L-system with at least as high a probability of producing the sequence of strings as the "real" (original hidden) L-system that generates them.

PMIT-S0L is based on a greedy decision process; however, on initially implementing this process using a greedy algorithm alone, it was found to rarely infer S0L-systems with even as few as 3 rewriting rules. One approach to addressing the limitations of a greedy algorithm is to use random forest, and although slightly better it was still not practical algorithm for inferring S0L-systems. The next step was to hybridize the greedy algorithm with a search algorithm. Using a genetic algorithm as the search algorithm, the hybridized approach was 100% successful at inferring S0L-systems with 5 rewriting rules or less (97% at 6 rewriting rules). However, when using brute force search, PMIT-S0L was 100% successful at inferring all 3600 of the generated S0L-systems with at most 12 rewriting rules, in less than 4 hours.

The remainder of this paper is structured as follows. Section II provides a formal definition of stochastic L-systems, discusses some of the unique challenges associated with inferring S0L-systems, and then describes how a greedy algorithm may be used to infer S0L-systems. Section III will discuss the methodology used to evaluate PMIT-S0L at inferring S0L-systems. Section IV provides the results of the evaluation. Finally, Section V will conclude the paper and discuss the future of PMIT-S0L as a tool for inferring S0L-systems.

II. Inferring S0L-Systems using Greedy Algorithm

PMIT-D0L is an earlier approach for inferring D0L-systems. The first PMIT-D0L implementation in [17], searched for successors as an ordered sequence of symbols in a search space that was pruned using logic based on necessary conditions. A significant improvement was made when it was recognized that every successor must be a subword of the input strings, and that searching for an ordered sequence of symbols could be replaced by searching for successor lengths [12, 18]. The successor length-based search space was made easier to search by using diophantine equations to restrict the solution space to only those solutions that are possible [12, 18]. Critically, with a D0L-system, anything learnt about the successor of a symbol A using any method is true for every copy of A due to the deterministic nature of the D0L-system. For S0L-systems, this is not true as any facts determined about the successor for an instance of A are not necessarily true for other instances of A, because it is always possible that any instance of A might be rewritten to a different successor. Without making assumptions with respect to the number of successors for each symbol in the alphabet, inferring an S0L-system requires finding the successor for every instance of every symbol. This paper investigates inferring an S0L-system by scanning the words symbol-by-symbol and choosing each successor by preferring successors that have been previously selected using a *greedy algorithm*.

As previously described informally, S0L-systems will be formally defined, as in [19]. First, given an alphabet V, then V^* is the set of all words over V. Also, given a word $x \in V^*$, $|x|$ is the length of x. An S0L-system is a tuple $G = (V, P, p, \omega)$, where V is an alphabet, $\omega \in V^*$ is the axiom, P is a finite set of productions $a \to u, a \in V$, $u \in V^*$ (where a is called the predecessor and u is the successor) with at least one production for each letter, and p is a function from P to $(0, 1]$ such that, for all $A \in V$, $\sum\limits_{A \to \alpha \in P} p(A \to \alpha) = 1$. Some S0L-systems are defined as having multiple axioms with a probability of each occurring, but this is not necessary for inference from a single sequence.

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 $\omega_0, \ldots, \omega_n$ such that $x = \omega_0 \Rightarrow \cdots \Rightarrow \omega_n = y$
2) a function σ from the set of pairs $\{(i, j) \mid 0 \leq i < n, 1 \leq j \leq |\omega_i|\}$ into P such that, for i from 0 to $n-1$, if $\omega_i = a_1 \ldots a_m, a_j \in V$, then $\omega_{i+1} = \alpha_1 \ldots \alpha_m$ where $\sigma(i, j) = (a_j \to \alpha_j)$ for j from 0 to m.

Given such a derivation d, the probability of $\omega_i = a_1 \ldots a_m$ deriving ω_{i+1}, denoted $p(\omega_i \Rightarrow \omega_{i+1}, d)$ is $\prod_{j=1}^{m} p(\sigma(i, j))$. Further, the probability of d occurring is $p(d) = \prod_{i=0}^{n-1} p(\omega_i \Rightarrow \omega_{i+1}, d)$. The computation of the probability that an S0L-system produces a derivation is used as the criteria for the greedy algorithm to make a selection (described below), and as a fitness value for the hybrid of the greedy algorithm with the two search algorithms.

For inference, the input is a trace of the derivation, and the goal is to find a "good" L-system that has a derivation with this trace, and to determine the derivation. In fact, suppose an inference algorithm exists that can determine the derivation, by selecting a successor for every symbol of every string (except the last one). Then an S0L-system can be created by assigning a probability to each successor equal to the number of times it was selected divided by the total number of instances of that symbol (in all strings before the last string). For example, if the inference algorithm concludes that successor x_1 is used for 9 out of 10 instances of A, then the probability for $A \to x_1$ is 0.9. This is justified because if the successor has been used for 90% of the instances of a symbol, it is expected that the probability of selection should be 90%. However, this does not imply that the hidden S0L-system has exactly these probabilities. Table I shows two abstracted S0L-systems, where in parentheses is the number of times that each successor was selected to produce a sequence of strings. The odds column computes the probability of the derivation occurring. If both S0L-systems produce the same sequence of strings, then the first L-system is more likely

TABLE I
TWO ABSTRACTED S0L-SYSTEMS WITH ODDS THAT IT WOULD GENERATE A SPECIFIC SET OF STRINGS

Successors	Odds
$A \to x_1 : 90\%(9)$ $A \to x_2 : 10\%(1)$	$0.9^9 \times 0.1^1 = 3.87\%$
$A \to x_3 : 50\%(5)$ $A \to x_4 : 50\%(5)$	$0.5^5 \times 0.5^5 = 0.097\%$

to have produced the strings. Furthermore, all other things being equal, the probability will be higher if one or a few successors have a high probability, than if the probabilities are more evenly distributed, which can be seen in Table I. For example, suppose that the inference algorithm has selected x_1 as the successor for A eight times and x_2 once. It now must select a successor for the tenth instance of A, and it can pick either x_1 or x_2 (meaning that the next $|x_1|$ symbols match x_1 and the next $|x_2|$ also match x_2), then the best local choice is to select x_1 as it improves the odds that the resulting L-system produces the strings. Using the two concepts explained so far, a greedy algorithm can infer an S0L-system, and perhaps even the original system using the following process.

Consider a sequence of strings ($\omega_0 \Rightarrow \cdots \Rightarrow \omega_n$) of the form previously described. Furthermore, suppose that a search algorithm has produced a sequence $y_1, \ldots, y_N$ that represent candidate successor lengths, where N is the expected number of successors in the S0L-system (the process for selecting a value for N is described after the next paragraph), and let $z = 1$ (a programming variable used as in index for the sequence of successor lengths). For each $A \in V$, maintain a list of successors found so far. For each word ω_i, $0 \leq i \leq n-1$ in sequence, scan each symbol from left to right maintaining a pointer to the next symbol to be produced in ω_{i+1}. The successor selection process works by applying the following abstract rules in order described below:

1) Last symbol of current word,
2) Existing successor matches current symbol,
3) Build a successor of length y_z,
4) Iterate over all possible successor lengths. For each length, check if remaining symbols have matching successors; or,
5) Terminate with error.

For rule 1, if the algorithm is scanning the last symbol (a_m) in ω_i, the successor picked consists of all the remaining symbols in ω_{i+1} starting from the current pointer location. Rule 2 states that for the x^{th} symbol a_x in ω_i with $x \neq m$, choose a successor α from the list of successors of a_x that matches the next $|\alpha|$ symbols in ω_i starting from the current pointer location. If more than one successor matches, then a so-called *look ahead process* is used to consider the remaining symbols in ω_i to try to match them by hypothesizing each matching successor of a_x as the correct successor. The look ahead process stops for the first successor that results in all the remaining symbols in ω_i matching. Otherwise, it selects the the successor with the largest number of successive matches. In case of a tie, it selects the longest successor. If rule 2 fails to produce a successor, then under rule 3, the successor for a_x is made of the next y_z symbols in ω_{i+1} starting from the current pointer location, and z is incremented by 1. If $z > N$, then rule 4 uses the *look ahead process* from rule 2 by instead hypothesizing successor lengths up to 10 (the maximum length of a successor considered in this paper, which is discussed in Section III-B). The pointer is then advanced by the length of the successor picked. If no successor can be found for a symbol, then the algorithm terminates and reports to the operator that N should be increased (note, that is for purposes of controlling the experiments, in practice, this could be automated).

The process described above should be considered as the inner loop for a search algorithm. Thus, in essence, the search algorithm is finding the best list of choices for successor lengths to be made when the greedy algorithm is uncertain of what to pick. Two algorithms are used to implement the search: brute force and genetic algorithm. For the search, a literal encoding scheme [20] is used, with N integer values where the values represent successor lengths. A standard genetic algorithm is used with roulette wheel selection, uniform crossover, uniform mutation, and elite survival mechanisms [20]. As previously mentioned, the fitness value is the probability that a candidate solution produced the input strings. Using a hyperparameter search [21], a population size of 50, crossover weight of 0.9, and mutation weight of 0.01 were found to be optimal. N, the number of dimensions for the search, is an estimation of the number of successors believed to be in the S0L-systems, which is unknown. Since it is difficult to compute N precisely, there are three possible outcomes. First, N may be exactly right. Second, N may be too small in which case rule 4, the *look ahead process*, may find the successors. Third, N may be too large, which increases the size of the search space, so pruning techniques can be used to eliminate unneeded parts of the search space (pruning is still done for the other two conditions, but it is much more prevalent for this condition). For this paper, the evaluations for PMIT-S0L are reported for the first two conditions for N. The condition where N is too large was evaluated but was found to be very slow because the pruning could not reduce the space very much, as there are such a large number of possible S0L-systems that could produce the sequence of strings, i.e. there were not very many impossible sections of the search space. Further investigation should be done to look for better pruning methods; however, in the interim, in practice an acceptable solution is to use a single execution using the largest value for N that will execute in an acceptable time frame.

Example 1: Let $\omega_1 = AAA$ and $\omega_2 = AAAAAABBBB$. Furthermore, assume that $N = 1$, and the search algorithm has found the solution sequence of $y_1 = (3)$ and z is initialized to 1. An initially empty list of successors is created for A and B. A pointer is initialized pointing to the first A in ω_2. The scanning process finds the first A in ω_1. Since the list of successors is empty, it uses the y_z from the solution (3) and so assigns A the successor AAA and adds it to the list. The pointer is advanced 3 symbols to the 4^{th} symbol A in ω_2. The process now finds the second A in ω_1. This time it finds the successor AAA in the list matches the next three symbols in ω_2 so it picks it as the successor for the second A in ω_1. The pointer is advanced three positions to the first B in ω_2. The third A in ω_1 is the last symbol so it receives a successor composed of all remaining symbols in ω_2, which is $BBBB$.

This process has a logical limitation that it may make incorrect choices when for some symbol A, if it has two or more successor where one is the prefix of another. This limitation is demonstrated in the example below.

Example 2: Let $\omega_1 = AAA$ and $\omega_2 = AAAAAABBB$. Furthermore, suppose that the successors for A in the original system are $AAA\,(prob\,p_1); AAAB\,(prob\,p_2); BB\,(prob\,p_3)$. Finally, assume that the search algorithm has a candidate solution with the value 3. As before, the first A will be assigned the successor AAA based on the search algorithm's choice. The second A will be also assigned AAA based on the greedy choice, but this decision is incorrect as the desired choice is $AAAB$. Finally, the third A will also have the wrong successor of BBB. In this case, PMIT-S0L will not fail to find some S0L-system; however, there is an element of chance to whether the solution will be the original or best S0L-system depending on the exact successors, the probabilities of the successors, and the order in which they are encountered.

III. Methodology for Evaluation

This section describes how PMIT-S0L was evaluated and the results of the evaluation. First, the metrics used to evaluate PMIT-S0L are described. As discussed in Section 1, since there are very few specific S0L-systems to be found in the literature, the data used to evaluate PMIT-S0L was procedurally generated and this generation process is described.

A. Performance Metrics

When inferring D0L-systems in [12, 13, 14], the main measure of performance used is whether or not an L-system was found that is *compatible* with the input strings (meaning it initially generates them), called success rate (SR). With S0L-systems though, it is less clear what success means. Indeed, for a single sequence of strings given as input, there are a multitude of possible S0L-systems that could produce them. Therefore, an algorithm's performance for inferring S0L-systems cannot be based only on whether or not it produces the strings. For a single sequence of strings produced by a known S0L-system, the probability (p_{known}) that the known S0L-system produced that particular set of strings can be computed. Then for a candidate solution S0L-system, the probability ($p_{solution}$) can be computed that it produced the strings. If the algorithm's candidate solution has $p_{solution} \geq p_{known}$ then this is counted as a success, otherwise it is a failure. SR can then be computed as the percentage of executions for which the algorithm was successful. Although there may be other ways to measure success (such as quantifying the amount by which $p_{solution}$ is less than p_{known}), this measure can certainly indicate that an algorithm would be successful.

Measuring success in the manner described above does not capture how closely the candidate solution matches the hidden L-system, which is also of interest. Furthermore, as productions with high probability are more often used, inferring such productions should be given a higher priority. This can be done using a weighted generalizations of true positive, false positives, and false negatives that match the successors in the hidden system and the candidate solution. The four metrics are: weighted true positive - system to candidate (WTP-S2C), weighted true positive - candidate to system (WTP-C2S), weighted false positive (WFP), and weighted false negative (WFN). WTP-S2C is the sum of the successor probabilities averaged over $|V|$ from the hidden system for successors that are in both the hidden system and the solution. WTP-C2S is the sum of the successor probabilities averaged over $|V|$ from the candidate solution that are in both the candidate and hidden system. For example, suppose that an S0L-system has $V = \{A\}$, and the successors and associated probabilities of A are $AA\,(90\%), AAA\,(6\%)$, and $AAAA\,(4\%)$. If the solution finds an S0L-system with the successors and associated probabilities of A as $AA\,(80\%), AAA\,(9\%)$, and $AAAAA\,(11\%)$, then the $WTP\text{-}S2C = 0.90 + 0.06 = 0.96$ (0.04 is not included as the successor $AAAA$ is not found in the candidate solution) and $WTP\text{-}C2S = 0.80 + 0.09 = 0.89$ (0.11 is not included as the successor $AAAAA$ is not in the original system). WFP is the sum of the successor probabilities averaged over $|V|$ from the candidate solution for successors that are not in the hidden system. WFN is the sum of the successor probabilities averaged over $|V|$ from the hidden system for successors that are not in the candidate. For a given system, if the successors match perfectly, both WTP values should be 1, while both WFN and WFP should be 0.

In addition to measuring how often an algorithm is successful at finding a suitable S0L-system, the amount of time to find the S0L-system is an important performance metric for an algorithm to be considered a practical tool. Since the goal is for PMIT-S0L to be used as a tool by the research community, its performance is also measured using the mean time to solve (MTTS), which is the mean time until PMIT-S0L returns a successful or failed solution over all executions. A time limit is imposed on PMIT-S0L of 4 hours ($14,400$ seconds). The 4 hour limit is arbitrary but was selected to keep the overall experimental time reasonable given the large number of executions performed (described next in Section III-B). All timings were captured on a single core of an Intel 4770 @ 3.4 GHz with 12 GB of RAM on Windows 10.

B. Data

The S0L-systems used for test cases for PMIT-S0L are procedurally generated, meaning that they are produced randomly but certain techniques are used to make them somewhat realistic. The rules are based on observations of L-systems found in the University of Calgary's virtual laboratory (vlab) [22]. When simulating L-systems, typically some subset of the letters are interpreted graphically; for example, the commonly used "turtle graphics" are interpreted as instructions for moving a turtle in a 2D or 3D Euclidean space [2]. The authors [12] found that graphical symbols often make the L-system inference problem simpler to infer. To summarize their findings, graphical symbols may be used as "markers" to deduce successor relationships, e.g. showing that a specific symbol in ω_{i-1} must produce a specific symbol in ω_i. Markers provide a lot of information about the L-system by both eliminating specific symbols as possible members of successors and slicing up the words into subwords that may be processed separately. Moreover, the graphical symbols may be filtered out of the strings which are then solved as individual sub-problems [12, 17]. These sub-problems are easier to solve than the initial problem of the production of the non-graphical symbols, which must be solved together. In other words, the hardest problem is finding the non-graphical symbols of the productions. Since this is also true for S0L-systems, for this research all of the test cases will consist only of non-graphical symbols and the following observations are made with the graphical symbols filtered out from the original D0L-systems in vlab [22].

It is observed that most D0L-systems in vlab have an alphabet size from 1 to 6 symbols. Most successors are observed to have 1 to 10 symbols. The number of symbols in the successors is unevenly distributed with the majority of successors having 2 to 5 symbols but quite a few having as many as ten symbols. It is observed that most successors usually have less than four unique symbols and that successors only using one letter (e.g., $A \rightarrow BBB$) are almost never longer than 6 symbols. The axioms are found to almost always be one symbol, and almost never longer than three symbols.

Based on the observations above, the procedure for generating S0L-systems will be described (these are then hidden and PMIT-S0L is used to infer them). The alphabet V for the generated S0L-system consist of a random number of letters between 2 to 9 symbols corresponding to A to K (the symbol F is skipped as it is often used to mean draw a line). Alphabets with 1 symbol are not considered. After selecting an alphabet size, a number of successors for each symbol is chosen from 1 to 3. There is a 50% chance of a symbol having 1 successor, 40% chance of 2 successors, and 10% chance of having 3 successors. A control parameter U on the upper bound of total successors is imposed on the generated L-system. If a number of successors is selected for a symbol that would result in going over this limit, then the value is reduced to maintain the limit. So if $U = 10$ and 10 successors has been reached, any remaining symbols are eliminated as they will have 0 successors. Since this research is about inferring S0L-systems, each system generated is given at least one symbol with at least two successors. When there are multiple successors, the probability for each successor is picked randomly between a lower bound and upper bound, p_{min} and p_{max}, which are controlled experimentally (described in the next paragraph). The value for p_{max} is computed according to the number of successors and the probabilities selected so far. For example, if there are three successors, and $p_{min} = 15\%$, then the range of values for the first successor is between 15% and 70%, as the second and third successor need to have 15% (p_{min}) each reserved for them. The next step is to pick a successor length for every successor of each $A \in V$ that varies from 1 to 10, with a value from 2 to 6 preferred. The following probability distribution for successor lengths is used: length $1\colon 4\%, 2\colon 16\%, 3\colon 20\%, 4\colon 20\%, 5\colon 20\%, 6\colon 16\%, 7\colon 4\%,$ $8\colon 2\%, 9\colon 1\%, \text{and} 10\colon 1\%$. For each successor, a symbol list is created of up to five (not to exceed the successor length) unique symbols, with the probability evenly distributed (20% of 1 unique symbol, 20% of 2, etc.). Finally, symbols are picked up to the successor length for each successor, by sequentially picking from the symbol list until the desired length is reached, where each symbol in the list has equal probability of being selected; however, every symbol in the list is guaranteed to occur at least once. The axiom for the L-system is generated by picking symbols from V to a length of 1 to 4, which is also evenly distributed, with every symbol having equal probability of being selected.

To evaluate PMIT meaningfully, a systematic approach is used to ensure that PMIT experiences as many different scenarios as possible. To this end, two experimentally controlled parameters are used. The first is an upper bound on the number of successors, U. The lowest possible value for U is 3, then U was iteratively increased by 1 until PMIT-S0L timed out with brute force search (ultimately to a maximum of 14). For each value of U, five separate experiments were done to evaluate the effects of having less dimensions than the number of successors. A search space was constructed with a number of dimensions N incremented from $U - 4$ to U. For each of these four values of N, p_{min} was set to $1\%, 10\%, 20\%, 30\%, 40\%$ and 50% (as $p_{min} > 50\%$ is not logically possible). For each combination, ten S0L-systems were procedurally generated and each of these systems was executed twice, generating a different set of input strings for each of the two executions. As previously discussed, there is a certain element of chance when a symbol has two or more successors that have a common prefix; therefore, when such a situation occurred the experiments were not performed and a new system was generated (the effects of this limitation on PMIT-S0L were investigated in separate experiments, but not included here for reasons of space). In total, 3600 S0L-systems were generated resulting in 7200 executions of PMIT-S0L for each approach.

IV. Results

On completing the experiments for PMIT-S0L, it was observed that changing p_{min} had no statistical effect on any of the performance metrics (with Pearson's correlation coefficient $r = -0.0064$, $p < 0.05$); therefore, the tables below show the results averaged over the values of p_{min}, i.e. each row represents an average over 120 executions. Tables II and III show the results for brute force search and GA respectively when N equals the number of successors in the hidden L-system. WFP and WFN are not shown as they are equal to $1-$(WTC-C2S) and $1-$(WTP-S2C). Table IV shows the SR for the greedy algorithm and random forest (the SR for GA is repeated for ease of comparison). MTTS is not shown for greedy algorithm or random forest in Table IV since they are both extremely fast (sub-second).

It can be seen that with brute force search, a compatible S0L-system is always found in under 4 hours when there are 10 successors or less that has a $p_{solution} \geq p_{known}$. The GA is less successful, especially with 7 successors or more. The probable cause is that the fitness function is not effective for searching the solution space and so alternatives should be explored, since the GA is generally much faster (it is slower early on due to a minimum number of generations). However, it can be seen that enhancing the greedy algorithm with genetic algorithm greatly improved the SR over the greedy algorithm and random forest. Since brute force search did much better than the GA, the remainder of the discussion will focus only on the results for brute force search.

Table V shows the results (SR only) for brute force search when the number of dimensions in the search (N) is less than the actual number of successors. When N is one less than the actual number of successor ($N - 1$ column in Table V), then PMIT-S0L has a 100% SR, until the number of successors is greater than 11. In these cases, PMIT-S0L ran out of time. However, it would be expected based on the previous results, that if the time limit were increased that these would also be 100% and so they are marked with a "*". For $N-2$, PMIT-S0L is only 100% successful when the number of successors is between 6 and 12 (for 13 and more the same issue occurs as for $N - 1$). Similarly, for $N - 3$ and $N - 4$, it can be seen that as the number of successors increases, the SR increases. One possible explanation to explain the effect, is that when the number of successors is small, the ratio of N to the number of successors is larger, e.g. if the number of successors is 4, and $N = 2$, then ratio is 50%; however, when the number of successors increases to 10, then the ratio is 20%. Algorithmically, this means that PMIT-S0L is more reliant on using the *look ahead process* to find successors since there are not enough dimensions in the search space to find suitable successors. Furthermore, this implies that the *look ahead process* is not particularly effective. This warrants further investigation to find an improvement since estimating the number of successors is difficult.

With respect to finding the original system, based on the WTP, WFP, and WFN values, it can be seen that when the number of successors is less than 5 the original system is always found. When the number of successors grows, the original system is not always found but the differences are fairly minor as can be seen by the WTP values not exceeding 2%. In the cases where it might be important to find the original system exactly, some additional technique(s) may be required.

V. Conclusions

The paper has introduced the Plant Model Inference Tool for Stochastic Context-Free L-systems (PMIT-S0L). Conceptually,

TABLE II
PERFORMANCE METRICS FOR PMIT-S0L WHEN USING BRUTE FORCE SEARCH

$\#Succ$	SR	WTP S2C	WTP C2S	$MTTS$
3	100%	1.000	1.000	0 : 00 : 00.212
4	100%	1.000	1.000	0 : 00 : 00.192
5	100%	1.000	1.000	0 : 00 : 00.821
6	100%	0.985	0.984	0 : 00 : 01.672
7	100%	0.989	0.986	0 : 00 : 18.126
8	100%	0.997	0.997	0 : 02 : 18.747
9	100%	0.988	0.988	0 : 19 : 40.131
10	100%	0.989	0.988	3 : 48 : 48.530

TABLE III
PERFORMANCE METRICS FOR PMIT-S0L WHEN USING GENETIC ALGORITHM

$\#Succ$	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 IV
SUCCESS RATE COMPARISON FOR A GREEDY ALGORITHM, RANDOM FOREST AND GENETIC ALGORITHM

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

TABLE V
SUCCESS RATE FOR PMIT-S0L WHEN USING BRUTE FORCE SEARCH AND NUMBER OF DIMENSIONS IS UNDERESTIMATED

$\#Succ$	Number of Dimensions (N)			
	-1	-2	-3	-4
3	100%	0%	n/a	n/a
4	100%	0%	0%	n/a
5	100%	77%	0%	0%
6	100%	96%	21%	0%
7	100%	100%	56%	0%
8	100%	100%	62%	5%
9	100%	100%	59%	7%
10	100%	100%	73%	6%
11	100%	100%	71%	11%
12	0%*	100%	83%	12%
13	0%*	0%*	84%	16%
14	0%*	0%*	0%*	23%

PMIT-S0L uses a greedy decision process to find the S0L-system. Although there are other elements, the core concept of the algorithm is to scan each symbol in each input string and choose a successor from the list of successors found so far if it matches the next n symbols to be produced to maximize the likelihood that the solution S0L-system produces the input strings. Early on, when the list of known successors is empty or near empty, the greedy decision is incapable of making good choices due to a lack of information. Canonically, this is often resolved with a random forest, but this also did not perform well. So the greedy algorithm was hybridized with a search technique to help choose the successor.

PMIT-S0L was evaluated using brute force search and genetic algorithm as the search techniques. Brute force search was more successful than genetic algorithm, but much slower. Using a time limit of 4 hours, the combined algorithm with brute force search was able to reliably (with 100% success rate) find S0L-systems with up to 12 successors in total. With genetic algorithm, success rate dropped considerably with 7 total successors. It is probable that either the fitness function or genetic algorithm is not ideal for searching the solution space. However, enhancing a greedy algorithm with optimization greatly improved the results over the greedy algorithm alone or random forest. The greedy decision process also implies a limitation that no single symbol can have successors where one is a prefix of another. PMIT-S0L can still infer S0L-systems when this occurs but it will succeed by chance. Overall, PMIT-S0L with brute force search is a reliable, practical tool for inferring S0L-systems with the specified limitations. With brute force search, it was found that most of the time, the original L-system is found; however, when the original is not found the variations between the solution and the original were very minor.

S0L-systems have been shown to have applications in many different research domains to model processes. Prior to this research, using an S0L-systems to model such processes required taking a specific problem within the research domain and building the L-system by hand [2, 5], generally using a priori knowledge (e.g. [2, 4, 7]). Although that technique has been successful, it is inefficient as every problem within every domain would need to be investigated separately and a priori knowledge may not always be available. Indeed, PMIT-S0L allows for the possibility of revealing the mechanisms in such processes since it only requires a sequence of input strings. PMIT-S0L opens up the possibility of additional research communities to use S0L-systems for modeling.

For future work, the main issue is to address the limitation on successor prefixes. It is uncertain how often this occurs for real-world systems but regardless it should be investigated further. As it would be ideal to have a more efficient search algorithm that brute force search (although it can be trivially made massively parallel), and since genetic algorithm was not very successful, either new fitness functions or different search algorithms should be investigated. Finally, this research assumed that there is one observed sequence of strings but another possible scenario is to infer an S0L-system that best describes many observed sets of strings, e.g. obtained from a "field of plants".

REFERENCES

[1] A. Lindenmayer, "Mathematical models for cellular interaction in development, parts I and II," *Journal of Theoretical Biology*, vol. 18, no. 3, pp. 280–315, 1968.

[2] P. Prusinkiewicz and A. Lindenmayer, *The Algorithimic Beauty of Plants*. New York: Springer Verlag, 1990.

[3] R. Mech, "Modeling and simulation of the interaction of plants with the environment using L-systems and their extensions," Thesis, University of Calgary, 1997.

[4] G. Danks, S. Stepney, and L. Caves, "Protein folding with stochastic L-systems," in *11th International Conference on the Simulation and Synthesis of Living Systems*, 2008, Conference Proceedings, pp. 150–157.

[5] M. A. Galarreta-Valverde, M. M. Macedo, C. Mekkaoui, and M. Jackowski, "Three-dimensional synthetic blood vessel generation using stochastic L-systems," in *Medical Imaging: Image Processing*, 2013, Conference Proceedings, p. 86691I.

[6] G. S. Hornby and J. B. Pollack, "Evolving L-systems to generate virtual creatures," *Computers & Graphics*, vol. 25, no. 6, pp. 1041–1048, 2001.

[7] G. Rongier, P. Collon, and P. Renard, "Stochastic simulation of channelized sedimentary bodies using a constrained L-system," *Computers & Geosciences*, vol. 105, pp. 158–168, 2017.

[8] T. Watanabe, J. S. Hanan, P. M. Room, T. Hasegawa, H. Nakagawa, and W. Takahashi, "Rice morphogenesis and plant architecture: measurement, specification and the reconstruction of structural development by 3D architectural modelling," *Annals of Botany*, vol. 95, no. 7, pp. 1131–1143, 2005.

[9] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane, "The use of positional information in the modeling of plants," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 2001, Conference Proceedings, pp. 289–300.

[10] P. Prusinkiewicz, R. Karwowski, and B. Lane, "The L+C plant modelling language," *Functional-Structural Plant Modelling in Crop Production*, vol. 22, pp. 27–42, 2007.

[11] J. Ubbens, M. Cieslak, P. Prusinkiewicz, and I. Stavness, "The use of plant models in deep learning: an application to leaf counting in rosette plants," *Plant Methods*, vol. 14, no. 1, p. 6, 2018.

[12] J. Bernard and I. McQuillan, "A fast and reliable hybrid approach for inferring L-systems," in *Proceedings of the 2018 International Conference on Artificial Life*. MIT Press, 2018, Conference Proceedings, pp. 444–451.

[13] R. Nakano and N. Yamada, "Number theory-based induction of deterministic context-free L-system grammar," in *International Conference on Knowledge Discovery and Information Retrieval*. SCITEPRESS, 2010, Conference Proceedings, pp. 194–199.

[14] B. Runqiang, P. Chen, K. Burrage, J. Hanan, P. Room, and J. 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*. Springer, 2002, Conference Proceedings, pp. 514–524.

[15] K. J. Mock, "Wildwood: The evolution of L-system plants for virtual environments," in *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*. IEEE, 1998, Conference Proceedings, pp. 476–480.

[16] F. Ben-Naoum, "A survey on L-system inference," *INFOCOMP Journal of Computer Science*, vol. 8, no. 3, pp. 29–39, 2009.

[17] J. Bernard and I. McQuillan, "New techniques for inferring L-systems using genetic algorithm," in *Proceedings of the 8th International Conference on Bioinspired Optimization Methods and Applications*, ser. Lecture Notes in Computer Science, vol. 10835. Springer, 2018, Conference Proceedings, pp. 13–25.

[18] I. McQuillan, J. Bernard, and P. Prusinkiewicz, "Algorithms for inferring context-sensitive L-systems," in *17th International Conference on Unconventional Computation and Natural Computation*, ser. Lecture Notes in Computer Science, vol. 10867. Springer International Publishing, 2018, Conference Proceedings, pp. 117–130.

[19] P. Eichhorst and W. J. Savitch, "Growth functions of stochastic Lindenmayer systems," *Information and Control*, vol. 45, no. 3, pp. 217–228, 1980.

[20] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[21] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[22] University of Calgary. Algorithmic Botany. [Online]. Available: http://algorothmicbotany.org