Stochastic L-system inference from multiple string sequence inputs

Jason Bernard1 · Ian McQuillan2

Accepted: 18 November 2022 / Published online: 8 December 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract
Lindenmayer systems (L-systems) are a grammar system that consists of string rewriting rules. The rules replace every symbol in a string in parallel with a successor to produce the next string, and this procedure iterates. In a stochastic context-free L-system (S0L-system), every symbol may have one or more rewriting rule, each with an associated probability of selection. Properly constructed rewriting rules have been found to be useful for modeling and simulating some natural and human engineered processes where each derived string describes a step in the simulation. Typically, processes are modeled by experts who meticulously construct the rules based on measurements or domain knowledge of the process. This paper presents an automated approach to finding stochastic L-systems, given a set of string sequences as input. The implemented tool is called the Plant Model Inference Tool for S0L-systems or PMIT-S0L. PMIT-S0L is evaluated using 960 procedurally generated S0L-systems in a test suite, which are each used to generate input strings, and PMIT-S0L is then used to infer the system from only the sequences. The evaluation shows that PMIT-S0L infers S0L-systems with up to 9 rewriting rules each in under 12 hours. Additionally, it is found that 3 sequences of strings are sufficient to find the correct original rewriting rules in 100% of the cases in the test suite, and 6 sequences of strings reduce the difference in the associated probabilities to approximately 1% or less.

Keywords Lindenmayer systems · Plant modelling · Model inference · Stochastic simulations · Hybrid search algorithm

1 Introduction

In 1968, Lindenmayer (1968) proposed a grammar system later called Lindenmayer systems (L-systems) to model multicellular growth in various organisms including plants. They consist of rewriting rules that are used to replace, in parallel, every symbol in a string with a word. The process of replacing all symbols with words in this manner is called a derivation step. Starting from an initial word, derivation steps iterate, thereby producing a sequence of strings. L-systems can be deterministic, implying that the simulation always proceeds the same way, with each string uniquely determined by the previous string; or stochastic, where the rules and derivations have a probability of occurring.

In this paper, stochastic context-free L-systems (S0L-systems) are studied. In such an L-system, all rules are of the form,

$$A → α : p,$$

where $A$ is some letter, $α$ is some word called a successor of $A$, and $p$ is an associated probability. Starting with a string $x$, a derivation occurs by taking each occurrence of every letter and replacing it with one of its successors with the associated probability to produce a new word, and the process iterates to create a sequence of strings. Thus, stochastic context-free L-systems where all successors are of length 1 can be seen as finite discrete-time Markov chains, as the current state only depends on the previous state. However, by replacing each letter in parallel, and then iterating, it is easy to obtain string patterns of exponential growth. These strings are important for visual simulations or models of processes by interpreting the symbols as instructions, such as “draw a line”, or “turn left/right” (e.g. Prusinkiewicz et al. 2001; Prusinkiewicz and Lindenmayer 1990; Galarreta-Valverde et al. 2013; Rongier et al. 2017) with each string being one step of the temporal simulation. They are also related
to stochastic context-free grammars (Lari and Young 1990), although with those, the rewriting is sequential rather than parallel, making it impossible to enforce that strings grow in a nonlinear fashion (Parikh 1966). Indeed, L-systems are particularly useful for modeling nonlinear processes (see, e.g. Section 1.9 of Prusinkiewicz and Lindenmayer 1990).

L-systems of different types have been particularly successful at modeling plant growth (Prusinkiewicz et al. 2018). In addition to simulations being visually accurate (Mündermann et al. 2005), they can also model mechanisms of plant development (Prusinkiewicz and Runions 2012), environmental factors (Mêch and Prusinkiewicz 1996), and genetic factors (Prusinkiewicz et al. 2018). Stochastic models have successfully been used to model different plant structures and processes (Prusinkiewicz and Lindenmayer 1990; Agu and Yokoi 1985), including Japanese Cypress trees (Nishida 1980). Furthermore, theoretical analysis has described the potential for L-systems more broadly as computational or algorithm models (Rozenberg and Salomaa 2012). The algorithm presented in this paper is domain agnostic, so may have applications in research domains where stochastic L-systems can be applied. They have already been applied to the modeling of protein secondary structure (Danks et al. 2008), arterial branching (Galarreta-Valverde et al. 2013), and sedimentary formations (Rongier et al. 2017).

Currently, models using SOL-systems, and other types of L-systems, are generated predominantly by hand by experts (e.g. Prusinkiewicz and Lindenmayer 1990; Nishida 1980; Rongier et al. 2017; Galarreta-Valverde et al. 2013), which is time-consuming. The process for finding a stochastic L-system has been described by Galarreta-Valverde et al. (2013) as requiring “tedious and intricate handwork” that could be improved by an algorithm to “infer rules and parameters automatically from real …images”. In Danks et al. (2008), automatic inference of SOL-systems was discussed specifically for the application of modelling protein folding. Their approach is domain specific as they use a priori scientific knowledge regarding proteins as the basis for constructing the L-system’s rules. Additionally, often the resulting models from existing approaches are assessed aesthetically (Danks et al. 2008; Rongier et al. 2017; Nishida 1980), which further reinforces that such approaches are domain specific as an aesthetic assessment can rarely (if ever) be transferred to another process. These two drawbacks hinder automatic inference. There have been a handful of attempts to infer branching structures from images (Runqiang et al. 2002; Št’ava et al. 2010; Guo et al. 2020; Holliday and Samal 1995), with the most recent using deep neural networks to discover the branching structures of an object transformed into a skeletonized structure with a greedy selection (Guo et al. 2020).

One approach for inferring an L-system from one or more temporal sequences of images is to divide the problem into two steps: (1) image processing (e.g. segmentation and skeletonization) on each image into the letters such that they would approximately draw the image in a simulator (e.g. Das Choudhury et al. 2018; Khan et al. 2020) and (2) the inference of an L-system from the sequences of strings. This basic two-stage approach was used successfully in Holliday and Samal (1995), but it was only tested on individual synthetic images. However, the approach has promise especially with optimized and robust solutions for both steps (1) and (2). This work is concerned with the second of these steps, which is called the inductive inference of an L-system. This paper presents a generalized algorithm for inductively inferring SOL-systems from one or more sequences of strings, called the Plant Model Inference Tool for Stochastic Context-Free L-systems (PMIT-SOL), which despite the name is domain independent. PMIT-SOL could possibly reduce the time and effort required to produce a model of a process from imagery. Furthermore, since PMIT-SOL uses no a priori information, it can also help reveal the mechanisms underlying a process (perhaps hidden from the images themselves) and thereby have additional scientific impact.

Inductive inference of different types of grammars is an important problem generally (De La Higuera 2010). Inductive inference of L-systems was studied mainly in the 1970s (Doucet 1974; Feliciangeli and Herman 1973; Ben-Naoum 2009). Deterministic context-free L-systems inference has recently been studied (Bernard and McQuillan 2021; McQuillan et al. 2018) where it was found that inference can be done optimally in polynomial time (with fixed alphabets), and all systems in a test suite of 30 L-systems in the literature could be inferred 100% accurately, each in under 4 seconds. Inference of L-systems generally was surveyed in Ben-Naoum (2009).

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

In this paper, the implementation of PMIT-SOL is described using greedy algorithm hybridized with search algorithms. Hybridizing greedy and genetic algorithm has been used previously for other problems (e.g. González et al. 2005; Al-Khiaty and Ahmed 2017); however, it has not been previously used or proposed for inductive inference of L-systems. Greedy algorithm is frequently used for grammar inference generally (De La Higuera 2010) and has been used for inferring L-systems from images (Guo et al. 2020). Hence, it is
important to assess their use on inductively inferring stochastic L-systems. In preliminary tests (not reported in this work), a greedy algorithm alone was found to be ineffective except with small alphabets. This work evaluates a hybrid of greedy algorithm with search algorithms (genetic and exhaustive search) as a practical technique to infer stochastic L-systems, which has not been explored in the literature. It is worth noting that it is easy to infer a stochastic L-system that can produce a given set of sequences as input (in linear time), by simply creating an L-system where the first letter of a string has a rule to produce the entire next string, and all other letters produce the empty word. However, this L-system is unlikely to be accurate in a sense that we will describe shortly. It is important to be able to reuse productions as much as possible. Greedy algorithms are natural for this purpose, but not guaranteed to be optimal. Other algorithmic strategies will be discussed in Sect. 6.

The evaluation is presented for PMIT-S0L at inferring SOL-systems with 960 procedurally generated SOL-systems. Some examples of the generated systems are provided in Appendix 1. These systems were generated using statistical properties of existing L-systems in order to be equally as complex as L-systems created by experts. Procedurally generated L-systems were used since only one SOL-system could be found explicitly in the literature (Nishida 1980) (other papers Danks et al. 2008; Galarrreta-Valverde et al. 2013; Rongier et al. 2017 created them but do not include it in their respective paper). The limited number of stochastic L-systems in the literature is due to the difficulty in constructing them (Nishida 1980 took an entire lengthy paper to create and justify theirs as will be described in Sect. 2), which would be dramatically improved by an automated approach. PMIT-S0L is also evaluated on this published SOL-system for Japanese Cypress (Nishida 1980). Additionally, the effects of having $M > 1$ sequences of strings as input for different values of $M$ are investigated with respect to differences between the true SOL-system and the SOL-system provided as a solution by PMIT-S0L. Scenarios where more than one sequence are used are important as, for example, it would be possible to have imagery from many plants of the same genotype of a species (or different genotypes of a species) and to desire a stochastic model to describe the population of plants. Furthermore, it would be particularly useful to compare populations of plants by comparing the models. Additionally, stochastic L-system models that can describe a diversity of plants are useful indirectly to improve image recognition tasks. For example, Ubbens et al. (2018) used a large set of synthetic images produced from an L-system model of Arabidopsis thaliana rosettes to augment the size of a data set used to train a deep convolutional neural network for the purposes of leaf counting. This was found to improve leaf counting on real rosette images versus only training using real images.

This work is an extension of the conference paper (Bernard and McQuillan 2018) that had the additional restrictions of only using one sequence of strings as input, and it had a limitation with respect to not having two successors of the same symbol with one being a prefix of the other—the so-called prefix limitation. This paper additionally presents methods for removing both limitations. PMIT-S0L is evaluated using a variety of performance metrics. Detailed in Sect. 3, the success rate describes the percentage of times that an SOL-system is found that has an equal or greater probability of producing the input than the original SOL-system has of producing the input. Thus, if the original L-system is defined as the correct L-system, then it can be possible to find L-systems that are more likely to have generated the input and are therefore better than the L-system which actually generated the strings. The time taken to find a solution (or report that none exists) is also measured. Additional metrics are defined to capture the differences between the predicted solution and the original system, both in terms of rewriting rules and probabilities.

The remainder of this paper is structured as follows: Section 2 provides some background information on L-systems and some applications of SOL-systems. Section 3 discusses some of the challenges of inferring stochastic L-systems, and how PMIT-S0L functions. The methodology for evaluating PMIT-S0L, focusing on the methods used to evaluate multiple sequences of strings as input, is provided in Sect. 4. Section 5 provides the results of the evaluation. Finally, Sect. 6 concludes the paper and discusses future directions of PMIT-S0L and inferring L-systems.

2 Background and preliminaries

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

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

In Eichhorst and Savitch (1980), the authors continue with the following definitions for derivations of an S0L-system. Given \( x, y \) as words in \( V^* \) where \( x \in X \), a derivation \( d \) of \( x \) to \( y \) of length \( m \) consists of two items:

1. a trace, which is a sequence of \( m+1 \) words \((w_0, \ldots, w_m)\) such that \( x = w_0 \Rightarrow \cdots \Rightarrow w_m = y \),
2. a function \( \sigma \) from the set of pairs \( \{(j, l) \mid 0 \leq j < m, 1 \leq l \leq |w_j| \} \) into \( P \) such that, for \( j \) from 0 to \( m-1 \), if \( w_j = A_1 \cdots A_{|w_j|}, A_j \in V \), then \( w_{j+1} = \alpha_1 \cdots \alpha_{|w_j|} \) where \( \sigma(j, l) = (A_l \rightarrow \alpha_l) \) for \( l \) from 1 to \( |w_j| \).

Thus, the function \( \sigma \) describes the productions applied to each letter in the derivation. Given such a derivation \( d \), the probability of \( w_j \) deriving \( w_{j+1} \) is

\[
p(w_j \Rightarrow w_{j+1}, d) = \prod_{l=1}^{|w_j|} p(\sigma(j, l)).
\]

Further, the probability of \( d \) occurring is

\[
p(d) = I(x) \cdot \prod_{j=0}^{m-1} p(w_j \Rightarrow w_{j+1}, d).
\]

Lastly given a finite set of derivations \( D = \{d_1, \ldots, d_M\} \), the probability of \( D \) occurring is

\[
p(D) = \prod_{i=1}^M p(d_i).
\]

As mentioned in Sect. 1, modelling with L-systems is done by associating a meaning to each symbol (or a subset of symbols) in \( V \); typically the meaning is an instruction for simulation software such as the “virtual laboratory” (Prusinkiewicz et al. 2007). So, a string of symbols is then taken as a sequence of instructions, and each derived word is taken as the next step in a temporal process. Symbols may have graphical and/or mechanical functions within the resulting model. A common graphical interpretation is the so-called turtle graphics (Prusinkiewicz and Lindenmayer 1990), which imagines a turtle in a 2D or 3D space with a state consisting of a position and orientation. The graphical symbols then modify the turtle’s state. When the turtle moves forward, it may optionally simultaneously draw a line. For branching models, two graphical symbols (“[” and “]”) will push and pop the turtle’s state onto a stack and switch to that state. Other symbols are used to represent components or an underlying mechanism in the model. For example, a symbol may be used to represent the apex of a plant where the stem will continue to grow at the next derivation step, until it flowers; this can be modelled stochastically (Prusinkiewicz and Lindenmayer 1990).

Nishida (1980) investigated using S0L-systems to model Japanese Cypress trees. He did not use any \textit{a priori} biological knowledge of Japanese Cypress and instead used a process of converting imagery to segments and then to an L-system. Importantly, this is the same process as our main goal (images to segments to L-systems, with PMIT-S0L being useful for the second step), except their work was done completely manually. In the paper, the meticulous process is described of segmenting images of Japanese Cypress by hand. The segments are then mapped to letters of an alphabet, with successors and associated selection probabilities picked for the segments such that they would reproduce the appropriate segmentation in the next image. The result is an S0L-system with 23 symbols and a total of 42 productions (shown in Table SD1 in the supplementary materials). The system produces imagery similar to the photos of the Japanese Cypress, as seen in Figures 1, 2, 3, 4 (Nishida 1980) (all images reprinted from Nishida (1980) with permission of Kyoto University). This shows that the goal of automatic inference in exactly this fashion is an exciting opportunity, as doing so manually requires a huge amount of effort.

### 3 Inferring S0L-systems using Greedy algorithm

This section will describe the methodology of PMIT-S0L. For the remainder of this paper let \( \rho = \{\rho_1, \ldots, \rho_M\} \) be the input, where each \( \rho_i \) is a finite sequence of strings over \( V \), and let \( \rho_i = (w_{i,1}, \ldots, w_{i,m_i}) \), \( 1 \leq i \leq M \) (each \( w_{i,j} \) is a string in \( V^* \)). In this paper, we assume \( m_1 = \cdots = m_M \) (sequences can be truncated to have the same number of strings), and we denote this size by \( m \). Inferring an L-system
can be stated as outputting some L-system $G$ (or reporting that none exists) that could produce all the sequences in $\rho$; that is, $G$ has a derivation with a trace of $\rho_i$, for each $i$, $1 \leq i \leq M$. In this case, $G$ is said to be compatible with $\rho$. To do this, the algorithm scans each sequence $\rho_i$ in $\rho$ and attempts to determine a derivation, which has a trace of $\rho_i$ in a word-by-word fashion. Each time it determines part of a derivation (e.g. that some specific occurrence of a letter in $w_{i,j}$ produces the subword between two positions of $w_{i,j+1}$), it adds this production to the current list of productions if it has not already been added. Hence, PMIT-S0L produces derivations $D = \{d_1, \ldots, d_M\}$ such that $d_i$ produces $\rho_i$ for $1 \leq i \leq M$, and from the derivations, the L-system can be easily calculated.

### 3.1 Complications with stochastic inference

If a compatible L-system does exist, then it must be found in the space of all L-systems, and hence, it is at least possible to search for one. PMIT-D0L (Bernard and McQuillan 2021) is an existing tool for inferring deterministic context-free L-systems (D0L-systems). The first PMIT-D0L implementation (Bernard and McQuillan 2018) used genetic algorithm, and searched for successors as an ordered sequence of symbols in a search space that was pruned using mathematical properties based on necessary conditions of L-systems. A significant improvement was made when it was recognized that every successor of a symbol $A \in V$ must be a subword of every word directly after one where $A$ occurs (for a deterministic L-system $M = 1$ is enough since the word produced at each step is uniquely determined) and that searching for an ordered sequence of symbols could be replaced by searching for successor lengths for each letter of the alphabet (Bernard and McQuillan 2021; McQuillan et al. 2018). With each possible solution consisting of a successor length for each $A \in V$, it is possible to scan every symbol in each
word in $\rho$ starting from the first word and take the subword of the associated length as the successor. When a scan can be completed without any inconsistencies, then the resulting successors are compatible with $\rho$. This is referred to as the scanning process. This approach works in the deterministic case because any information deduced about the successor for any instance of $A$ in $\rho$ must be true for every $A$ that appears in every word in $\rho$, and therefore, this search space has only $|V|$ dimensions.

For stochastic L-systems, while it still true that every $A \in V$ must produce a subword of the next string and hence some kind of encoding scheme with successor lengths is possible, it is no longer true that deducing a fact about some instance of $A$ is true for all other instances of $A$. Indeed, with SOL-systems, different instances of each $A$ (of each word of each sequence) in $\rho$ can produce different successors. For $\rho_i = (w_{i,1}, \ldots, w_{i,m})$, since every instance of a symbol can produce a different successor, the most intuitive solution space defines at least one dimension (for example, using the successor length encoding scheme from Bernard and McQuillan (2021); McQuillan et al. (2018)) by preferring successors that have been previously selected using a greedy algorithm. Hence, it is inferring a set of derivations that can produce $\rho$, and from this, the scanning process is used to find the L-system.

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

As mentioned earlier, for any $\rho$, there are a multitude of possible derivations (and hence SOL-systems) that can produce $\rho$, but a best solution is a set of derivations with the greatest probability of producing $\rho$ (Bernard and McQuillan 2018). In absence of any additional information (e.g. a priori knowledge that might lead to a different choice), a set of derivations with the greatest computed probability is said to have maximum parsimony. For example, Table 1 summarizes two derivations where in parentheses is the number of times that each successor was applied to produce a sequence of strings in the two different derivations. The “Odds” column shows the computed probability of the derivation occurring. The first derivation has a greater probability of generating the sequence, and so should be preferred as the solution. The probability that a set of derivations produced $\rho$ is increased when one or a few successors have a high probability, as opposed to an equal distribution across all of the successors. This mathematical property provides the guidance for a greedy algorithm to infer a stochastic system using the process described next.

Conceptually, the core greedy algorithm component is relatively straightforward. Suppose that a list of successors for each $A \in V$ is being maintained including a count of how often each successor has been selected. For each word $w_{i,j}$ with $j$ from 1 to $m$, the algorithm will partition $w_{i,j+1}$ into $|w_{i,j}|$ subwords by scanning each letter $A$ in order from left-

### 3.2 Methodology with PMIT-SOL

PMIT-SOL infers an SOL-system based on $\rho$ by selecting successors, such that each local choice maximizes the probability of producing the words of $\rho$ scanned so far. This paper investigates inferring an SOL-system by scanning the words symbol-by-symbol, and choosing, based on successor length, each successor (similar to PMIT-DOL in Bernard and McQuillan 2021; McQuillan et al. 2018) by preferring successors that have been previously selected using a greedy algorithm. Hence, it is inferring a set of derivations that can produce $\rho$, and from this, the scanning process is used to find the L-system.

For stochastic L-systems, while it still true that every $A \in V$ must produce a subword of the next string and hence some kind of encoding scheme with successor lengths is possible, it is no longer true that deducing a fact about some instance of $A$ is true for all other instances of $A$. Indeed, with SOL-systems, different instances of each $A$ (of each word of each sequence) in $\rho$ can produce different successors. For $\rho_i = (w_{i,1}, \ldots, w_{i,m})$, since every instance of a symbol can produce a different successor, the most intuitive solution space defines at least one dimension (for example, using the successor length encoding scheme from Bernard and McQuillan (2021); McQuillan et al. (2018)) by preferring successors that have been previously selected using a greedy algorithm. Hence, it is inferring a set of derivations that can produce $\rho$, and from this, the scanning process is used to find the L-system.

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

As mentioned earlier, for any $\rho$, there are a multitude of possible derivations (and hence SOL-systems) that can produce $\rho$, but a best solution is a set of derivations with the greatest probability of producing $\rho$ (Bernard and McQuillan 2018). In absence of any additional information (e.g. a priori knowledge that might lead to a different choice), a set of derivations with the greatest computed probability is said to have maximum parsimony. For example, Table 1 summarizes two derivations where in parentheses is the number of times that each successor was applied to produce a sequence of strings in the two different derivations. The “Odds” column shows the computed probability of the derivation occurring. The first derivation has a greater probability of generating the sequence, and so should be preferred as the solution. The probability that a set of derivations produced $\rho$ is increased when one or a few successors have a high probability, as opposed to an equal distribution across all of the successors. This mathematical property provides the guidance for a greedy algorithm to infer a stochastic system using the process described next.

Conceptually, the core greedy algorithm component is relatively straightforward. Suppose that a list of successors for each $A \in V$ is being maintained including a count of how often each successor has been selected. For each word $w_{i,j}$ with $j$ from 1 to $m$, the algorithm will partition $w_{i,j+1}$ into $|w_{i,j}|$ subwords by scanning each letter $A$ in order from left-

\[ w_{i,j} = w_{i,j,1}w_{i,j,2} \ldots w_{i,j,|w_{i,j}|} \]

\[ w_{i,j+1} = w_{i,j+1,1}w_{i,j+1,2} \ldots w_{i,j+1,|w_{i,j+1}|} \]

For stochastic L-systems, while it still true that every $A \in V$ must produce a subword of the next string and hence some kind of encoding scheme with successor lengths is possible, it is no longer true that deducing a fact about some instance of $A$ is true for all other instances of $A$. Indeed, with SOL-systems, different instances of each $A$ (of each word of each sequence) in $\rho$ can produce different successors. For $\rho_i = (w_{i,1}, \ldots, w_{i,m})$, since every instance of a symbol can produce a different successor, the most intuitive solution space defines at least one dimension (for example, using the successor length encoding scheme from Bernard and McQuillan (2021); McQuillan et al. (2018)) by preferring successors that have been previously selected using a greedy algorithm. Hence, it is inferring a set of derivations that can produce $\rho$, and from this, the scanning process is used to find the L-system.

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

As mentioned earlier, for any $\rho$, there are a multitude of possible derivations (and hence SOL-systems) that can produce $\rho$, but a best solution is a set of derivations with the greatest probability of producing $\rho$ (Bernard and McQuillan 2018). In absence of any additional information (e.g. a priori knowledge that might lead to a different choice), a set of derivations with the greatest computed probability is said to have maximum parsimony. For example, Table 1 summarizes two derivations where in parentheses is the number of times that each successor was applied to produce a sequence of strings in the two different derivations. The “Odds” column shows the computed probability of the derivation occurring. The first derivation has a greater probability of generating the sequence, and so should be preferred as the solution. The probability that a set of derivations produced $\rho$ is increased when one or a few successors have a high probability, as opposed to an equal distribution across all of the successors. This mathematical property provides the guidance for a greedy algorithm to infer a stochastic system using the process described next.

Conceptually, the core greedy algorithm component is relatively straightforward. Suppose that a list of successors for each $A \in V$ is being maintained including a count of how often each successor has been selected. For each word $w_{i,j}$ with $j$ from 1 to $m$, the algorithm will partition $w_{i,j+1}$ into $|w_{i,j}|$ subwords by scanning each letter $A$ in order from left-

\[ w_{i,j} = w_{i,j,1}w_{i,j,2} \ldots w_{i,j,|w_{i,j}|} \]

\[ w_{i,j+1} = w_{i,j+1,1}w_{i,j+1,2} \ldots w_{i,j+1,|w_{i,j+1}|} \]
to-right as follows: If $A$ is the rightmost symbol of $w_{i,j}$, then pick the successor to produce everything that remains in $w_{i,j+1}$ (the parts of $w_{i,j+1}$ that have not been matched in the derivation). If $A$ is not the rightmost symbol, then pick the successor, if one exists, from the current list for $A$ out of those that match the next symbols of $w_{i,j+1}$ that has been selected most often so far. One issue with this algorithm is that early on, especially for the first few symbols scanned, the list of successors for each $A \in V$ is (or is near) empty or none match, and so the greedy process is not able to make a choice from the existing list. In the literature, this type of problem is sometimes resolved using a random forest algorithm (Breiman 2001), but this was found to not work well in this instance (discussed below). A search algorithm was used in these instances to pick successors when nothing in the list is matched.

For this, the algorithm keeps a vector $y$ of $N$ nonnegative integers. When there is no successor in the current list for $A$ that matches the next symbols in $w_{i,j+1}$, the algorithm retrieves the next unused integer from $y$, $k$ say, and then a successor is selected using the next $k$ symbols from $w_{i,j+1}$. To find $y$ (a single sequence for all letters), a search algorithm is used (this search process will be described in Sect. 4.2). However, this process raises a new question: how many successor length choices are needed, i.e. what should be $N$?

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

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

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

1. If scanning the last symbol of the current word, then select all remaining symbols in the produced word.
2. An existing successor in the list for $A$ matches the next symbols in $w_{i,j+1}$; the most frequently chosen thus far is selected greedily.
3. Build a successor of length $y_z$, and increase $z$ by one.
4. Terminate with error.

Given an appropriately chosen $y$, this process can determine derivations associated with traces in $\rho$. However, the solution need not be optimal (the greedy choice might not lead to the best solution).

### 3.3 Searching for successor lengths

Next, the method for determining the integer sequences $y$ is described. A standard genetic algorithm (SGA) and exhaustive search (ES) are evaluated. For both, a literal encoding (Back 1996) is used, so a search space is constructed of $N$ integer dimensions with bounds from 1 to 10 (as discussed and justified later in Sect. 4.2, the maximum number of symbols in each successor is assumed to be 10 although this could be increased). For each sequence $y$ searched, Procedure 1 is used to predict a set of derivations that have $\rho$ as traces. The fitness value of $y$ is defined to be the probability of these derivations occurring with the traces in $\rho$, which is given in Eq. (2). Algorithm 3.1 shows the pseudocode for PMIT-S0L.
When the search technique is set to SGA, it uses roulette wheel selection, uniform crossover, uniform mutation, and elite survival operators (Back 1996). An SGA with simple operators is selected as little is known about the search space of $Q$ genomes, where each genome in this case is a candidate vector $y$ consisting of $N$ genes. The initial population is produced randomly. The main processing loop of an SGA consists of a population and an SGA provides easily tunable mechanisms for balancing exploration and exploitation. Briefly, an SGA works by the following steps (Back 1996). It consists of a population of $Q$ genomes, where each genome in this case is a candidate vector $y$ consisting of $N$ genes. The initial population is produced randomly. The main processing loop of an SGA performs four steps: selection, crossover, mutation, and survival. In the selection step, $Q/2$ pairs of genomes are selected from the population. The crossover step takes each pair and randomly swaps zero or more genes between them, and since there are $Q/2$ pairs, this results in $Q$ new genomes. Each new genome is then mutated by randomly changing zero or more genes to a new value. The survival step merges the initial and new populations. Each member of the population is assessed a fitness value as described above (using Procedure 1 with $y$ from the population), and the population is sorted by fitness value. The top $Q$ genomes are preserved and the remainder are culled. With respect to Algorithm 3.1, $S.iterate()$ for an SGA corresponds to performing the selection, crossover, mutation, and survival steps. $S.population$ is the entire population of genomes. The SGA terminates by convergence detection ($S.terminate()$ in Algorithm 3.1), which functions as follows. Termination occurs as follows: every time a new best solution is found, a variable Gen records the number of the current generation. If Gen additional iterations occur without a new best solution being found, then the SGA terminates.

To optimize the control parameters of the SGA, a hyperparameter search was done (Bergstra and Bengio 2012) using a random search with 16 trials. As a result of the hyperparameter search, the control parameters were set as follows: population size of 50, crossover weight of 0.9, and a mutation weight of 0.01.

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

### 3.4 The prefix limitation

In the conference paper (Bernard and McQuillan 2018), this earlier version of PMIT-S0L had an identified limitation, when for at least one $A \in V$, there are two or more successors, and one successor is a prefix of the other (Bernard and McQuillan 2018). Let $u, v$ be two successors of $A$, such that $u = vx$, and $u, v, x$ are words over $V$. In this case, if the shorter successor ($u$) is encountered first in $\rho$, then assuming all other successors are correct, for all future instances of $A$, the greedy choice (rule #2 of Procedure 1) will always select $u$ as the successor as the next $|u|$ symbols can be positively associated with $A$. This effect is shown in Example 1. Since the introduction of PMIT-S0L (Bernard and McQuillan 2018), a technique has been found to often correct this limitation, described next, although it is only used with ES and not with the SGA (due to complications to be discussed).

**Example 1** Let $w_1 = AAA$ and $w_2 = AAAAAABBB$. Suppose that the successors for $A$ in the original system are:

$$AAA; AAAB; BB$$

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

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

\begin{quote}
\textbf{Example 2} Suppose the first few values of
\[ y = (t_1, y_1, t_2, y_2, t_3, y_3, \ldots) \]
are \((0, 3, 4, 1, 5, \ldots)\). Then, the first five successors have been found by:
\begin{enumerate}
\item Build a successor of length 3 since \(y_1 = 3\)
\item A greedy choice
\item A greedy choice
\item A greedy choice
\item Stop allowing greedy choices since \(t_2 = 3\)
\item Build a successor of length 4 since \(y_2 = 4\)
\item A greedy choice
\item Stop allowing greedy choices since \(t_3 = 1\)
\item Build a successor of length 5 since \(y_3 = 5\)
\end{enumerate}
\end{quote}

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

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

It is acknowledged that while this technique can address the prefix limitation, it is somewhat inefficient (shown in Sect. 5). Finding a more efficient technique to address this limitation is an area for future investigation.

\section{4 Evaluation methodology}

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

\subsection{4.1 Performance metrics}

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

Several measures are used to assess accuracy, which is necessary in order to properly capture different ways that SOL-systems can differ: success rate (SR), mean time to solve (MTTS), weighted true positive - system to candidate (WTPS2C), weighted true positive - candidate to system (WTP-C2S), and probability error (e). There will be defined next.

Inference of an SOL-system with derivations \(D\) that can produce \(\rho\) is deemed \textit{successful} if the probability of \(D\) produces \(\rho\) calculated using Eq. (2) is greater than or equal to the probability of the original derivations \(\overline{D}\) to generate \(\rho\). Thus, finding an L-system that is slightly worse than the original would therefore be classified as not successful with this stringent measure. The success rate is the percentage of experiments that successfully finds such an SOL-system and derivations.

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

In order to assess the similarity between the predicted L-system and the original, a true positive is defined as a successor that is in both the original L-system and the candidate L-system regardless of any difference in their associated probability. One could also certainly define the following: a false positive as a successor that is in the candidate L-system but not in the original, and a false negative is the opposite. It would be possible to simply count the true positives, false positives and false negatives with these definitions. How-
ever, there are two issues: first, the terms false positive and false negative imply that the original L-system is the better L-system, which might not be the case. Second, focusing on successors with higher associated probabilities are probably more important to accurate reproducing a process in silico since they occur more frequently. Hence, the measures listed next are used instead, and they are weighted to favour those with a higher associated probability.

1. Weighted True Positive - System to Candidate (WTP-S2C) is the sum of associated probabilities for successors common to the candidate (C) and the original system (S) divided by \( |V| \).
2. Weighted True Positive - Candidate to System (WTP-C2S) is the sum of associated probabilities for successors common to the candidate (C) and the original system (S) divided by \( |V| \).

Ideally, the weighted true positive values above should be 1.00 indicating a perfect match. When a match between the hidden and candidate L-systems is imperfect, greater values by the metrics generally indicate that successors with higher associated probabilities have been matched. A simple example of the computation is provided in Table 2. Critically, these two metrics do not indicate how well the associated probabilities match, which is captured with the probability error metric below. If all of the successors match, then regardless of the associated probabilities, the WTP-C2S and WTP-S2C metrics will be computed as 1.00.

Probability error (\( e \)) is calculated by taking each production in the original system: if that production is also in the predicted system, then add the absolute difference of the two probabilities; if the production is not present, then add the probability of it occurring (in the original system). Once all productions have been examined, the total is divided by the number of symbols in the alphabet. Thus, this metric is measuring how different the predicted L-system is from the original even taking into account the differences in probabilities of the rewriting rules. It is an important metric for the accuracy of inferring SOL-systems. Using values from the example in Table 2, the \( e \) metric would be computed as \( e = (|0.67 - 0.60| + |0.80 - 0.85| + |0.19 - 0.15| + 0.33 + 0.01) / 2 = 0.25 \). If all of the productions had matched, the error would simply be the sum of the absolute differences of the associated probabilities.

### 4.2 Data

Since there are very few published SOL-systems in the literature, the test cases for PMIT-SOL are mainly procedurally generated via a random process described as follows.

The procedural generation is based on observations of L-systems found in the University of Calgary’s virtual laboratory (http://algorothmicbotany.org). For the analysis, 72 known deterministic L-systems were taken from the repository. The L-systems extracted were a mix of fractals, L-systems for teaching their principles, and models of plants. For each L-system, the number of symbols in the alphabet (only when there were two of more symbols), the length of the successors (with the graphical symbols removed), number of distinct non-graphical letters in each successor, and the number of successors associated with a symbol (only for any L-systems that have at least one symbol with two or more successors) were captured. The observations showed that the most common values were as follows: (1) the alphabet size ranged from between 2 and 9 symbols, (2) the successor length varied from 1 to 10, (3) each symbol may have between 1 and 3 successors, and (4) each successor has 1–5 distinct symbols. A potential issue using procedural generation would be an intentional bias towards producing L-systems that were easy to infer. Hence, these observations and the distribution of the values are not explicitly used to create the procedural generation method described below, but rather to produce some approximate guidelines that err towards making L-systems that should be harder to infer. For example, the odds of a symbol having a single successor is very high, but is not conducive to producing SOL-systems. Similarly, many successors had a length of 1 or 2, but this does not tend to create L-systems that are hard to infer as the resulting strings in \( \rho \) are very short.

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

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

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

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

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

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

### 5 Results and discussion

This section provides the results of the evaluation of PMIT-S0L. First, the evaluation of PMIT-S0L on the procedurally generated S0L-systems (the main result) is discussed, including the effects of addressing the so-called *prefix limitation* (previously described in Sect. 3.4). Afterwards, observations regarding the inference of the Japanese Cypress tree L-system are described. The variations of PMIT-S0L executing with and without the prefix limitation process are denoted as PMIT-S0L+PL and PMIT-S0L-PL, respectively, when needed for clarity.

A preliminary investigation was done using only greedy algorithm, then random forest, before building a hybrid algorithm combining a greedy with a simple genetic algorithm (SGA), and exhaustive search (ES). As they were preliminary, the results of these experiments can be found in Table SD2 of the supplementary information; however, in summary both the greedy-only algorithm and the random forest were found to be ineffective relative to the hybrid algorithm. Therefore, the remainder of this section focuses on the hybrid algorithms.

It was found that PMIT-S0L-PL with exhaustive search had 100\% success rate with up to 10 successors when \( M = 1 \) (shown in Fig. 5). With greedy plus exhaustive search, PMIT-S0L had 100\% success rate in every single case. However, WT2-S2C and WTP-2CS were slightly less than 100\% in some cases due to minor variations between the candidate productions. SGA was not as successful as exhaustive search as is evident from the green line showing success rate which decreased as the size of the L-system increased. However, it is much faster, peaking at about 2 minutes for 10 successors (see Table SD3, time for ES described below). A fitness landscape analysis was completed; however, there were no characteristics to the search space that suggested a particular way forward. Thus, an avenue for future investigation is to investigate the search space more deeply to find ways to avoid the use of ES for reasons of performance.

For all further experiments, only exhaustive search plus greedy algorithm were evaluated, and not SGA. The effect on MTTS on inferring S0L-systems with and without the prefix limitation process enabled is shown in Fig. 6 (as the other

| Original System (S) | Candidate (C) |
|---------------------|---------------|
| \( A \rightarrow AB, p = 0.67 \) | \( A \rightarrow AB, p = 0.60 \) |
| \( A \rightarrow BA, p = 0.33 \) | \( A \rightarrow AA, p = 0.40 \) |
| \( B \rightarrow BB, p = 0.80 \) | \( B \rightarrow BB, p = 0.85 \) |
| \( B \rightarrow AA, p = 0.19 \) | \( B \rightarrow AA, p = 0.15 \) |
| \( B \rightarrow AB, p = 0.01 \) | n/a |
| WTP-S2C = \((0.67 + 0.8 + 0.19)/2 = 0.83 \) | WTP-C2S = \((0.60 + 0.85 + 0.15)/2 = 0.80 \) |

Rows 2 and 5 are not included in the sum to compute WTP-S2C and WTP-C2S as the successors do not match.
metrics were not significantly effected, they are included in Table S6). It shows that PMIT-S0L+PL is slower than the other variants. It is difficult to say whether the time increase is worth the ability to infer S0L-systems where a symbol $A$ has a successor that is a prefix of another, as it is unclear how often this occurs in practice. Subjectively, considering the possibility that the encoding scheme described herein to address the prefix limitation had the potential to be extremely large, the time increase seems reasonable. This suggests that, at least for the procedurally generated data sets, the method for removing the prefix limitation is effective.

The next goal is to investigate the effect of inferring an S0L-system when using various numbers of sequences of strings as an input. Figure 7 shows how the probability error, WTP-C2S and WTP-S2C change as $M$ increases. This raw data is shown in Table 3, which also shows the mean time to solve, the maximum successor difference, and successor difference rate. SR is not shown as it was always 100%, for each value of $M$ with PMIT-S0L+PL and dataset $D_{S_{v,M}}$. It can be seen that from $M \geq 3$, PMIT-S0L+PL infers the original system for all test cases, as the WTP values are all 1.000. With respect to error in the probability distribution, this becomes subjectively reasonable at $M = 3$ with 6.3% average error. Error seems to be close to minimal at $M = 6$ (at approximately 1%), although it continues to decline at a small rate as $M$ increases. Increasing $M$ has a negligible effect on MTTS since $\rho_2$ to $\rho_M$ are only scanned when an S0L-system has been found to be compatible with $\rho_1$, and this is similar to the scanning process which takes sub-millisecond time in all test cases. So, adding additional sequences of strings is generally beneficial in practice.

With respect to inferring the Nishida’s S0L-system for modeling Japanese Cypress (Nishida 1980), PMIT-S0L+PL was not able to infer it in a practical amount of time. The first experiment was to set $N = 42$, which is much greater
than the approximate maximum of 9 successors in the other experiments. After several hours, this was terminated and it was estimated that PMIT-SOL+PL would take at least $10^9$ hours to complete using exhaustive search in a sequential fashion.

### 6 Conclusions and future directions

This paper presents an investigation into inferring stochastic context-free L-systems (SOL-systems) when using different numbers of sequences of strings with the Plant Model Inference Tool for SOL-systems (PMIT-SOL). PMIT-SOL is a generalized algorithm for inferring SOL-system and requires no a priori scientific knowledge when compared to existing approaches for inferring SOL-systems algorithmically (e.g. Danks et al. 2008; Prusinkiewicz and Lindenmayer 1990; Rongier et al. 2017). Being generalized means that PMIT-SOL may be used for any problem as opposed to requiring a specific algorithm for each individual problem in a specific research domain. PMIT-SOL opens the possibility of inferring SOL-systems in research domains where none have been found to date.

PMIT-SOL is primarily evaluated on procedurally generated SOL-systems due a shortage of specific systems published in the literature. An analysis was done of existing L-systems to create realistic procedural generation rules. Three data sets, for a total of 960 L-systems (a total of 3000 experiments were conducted across these systems), were generated to evaluate different aspects of PMIT-SOL.

PMIT-SOL has two different modes of operation controlled by a prefix limitation Boolean parameter (denoted as PMIT-SOL+PL and PMIT-SOL-PL). The prefix limitation parameter controls if it searches all L-systems, or only those without multiple successors of the same letter, where one is a prefix of the other. The results show PMIT-SOL-PL is quite a bit faster than PMIT-SOL+PL. For an SOL-system with 9 successors, PMIT-SOL-PL will succeed on average in about 20 minutes, while PMIT-SOL+PL takes several hours. However, PMIT-SOL+PL is still practical considering the effort required to infer an SOL-system by hand.

PMIT-SOL-PL was found be extremely accurate for inferring a compatible SOL-system that is at least as probable as the original system. This paper shows that PMIT-SOL+PL was always able to find the original systems in our data set when at least three sequences of strings were used as inputs. Hence, the success rate was 100% using greedy algorithm and exhaustive search. Additionally, the evaluation showed that when six sequences were used, the error in the solution’s probability distribution compared to the original system becomes low (about 1% or less), and there is not much improvement for adding additional sequences of strings. Therefore, it is recommended that in practice at least 3 sequences of strings should be used to infer SOL-systems, but 6 or more is ideal to minimize the error in probabilities. Finally, there is essentially no penalty to execution time for adding additional sequences of strings, so there is little reason to avoid using all of the string sequences that are available. However, the time increases significantly as the number of productions increases.

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

There are several directions that can be taken with this research in the future. Perhaps most importantly comes from applying this research to the practical problem of inferring L-systems from segmented images. De La Higuera (2005) argues that to be realistic, grammar inference algorithms need to handle errors in the strings, e.g. insertions or deletions of symbols.

While the algorithm presented herein is 100% successful at inferring L-systems with greedy algorithm and exhaustive search, its utility is limited as the number of productions increases due to the execution time. All tests were done with a single CPU core, and therefore parallelism would allow for inference of larger L-systems; however, simply adding more processing capability imposes limits, based on cost, availability, etc. It is more desirable to have efficient algorithms; hence, other approaches may be worth considering. Neural network-based approaches have already been shown to have potential with a system that may be visualized (Guo et al. 2020). Natural language processing (NLP) has been increasingly effective at parsing in a variety of languages (Edwards 2021). While some of the earliest works in

---

### Table 3 Performance metrics for PMIT-SOL+PL with exhaustive search and $M$ sequences of strings from 1 to 10, $N = S$, and the dataset $D_{S,M}$

| $M$ | WTP-S2C | WTP-C2S | $\epsilon$ | MTITS |
|-----|---------|---------|----------|-------|
| 1   | 0.987   | 0.975   | 0.174    | 03:48:228 |
| 2   | 0.998   | 0.999   | 0.113    | 01:35:192 |
| 3   | 1.000   | 1.000   | 0.063    | 01:30:821 |
| 4   | 1.000   | 1.000   | 0.060    | 03:55:821 |
| 5   | 1.000   | 1.000   | 0.023    | 02:30:821 |
| 6   | 1.000   | 1.000   | 0.010    | 04:00:821 |
| 7   | 1.000   | 1.000   | 0.016    | 02:16:131 |
| 8   | 1.000   | 1.000   | 0.016    | 02:16:131 |
| 9   | 1.000   | 1.000   | 0.008    | 02:46:496 |
| 10  | 1.000   | 1.000   | 0.006    | 03:00:065 |

Success rate is 100% for all experiments and is therefore not in the table.
grammar inference used NLP techniques (Nevill-Manning and Witten 1997), these approaches should be reinvestigated using more modern NLP techniques. The approach herein, of course, should not be simply abandoned. While genetic algorithm was not effective at searching the space, other search algorithms beyond exhaustive search would be of interest [the “No Free Lunch Theorem” states that no algorithm can be optimal for all search problems, and the expectation is for an algorithm to be adjusted to the search space (Ho and Pepyne 2002)]. It is recommended to conduct a formal search landscape analysis to understand the characteristics of the search space more completely. Finally, the main issue with using PMIT-S0L in a practical fashion is the need to select a reasonable value for the size of the vector used for searching (which roughly corresponds to the number of productions). An algorithm that does not require this parameter would be ideal, or alternatively finding a good way to compute or estimate the vector size.

Acknowledgements The authors thank Dr. Farhad Maleki for his assistance in acquiring the images for this paper.

Author Contributions JB had the original idea for the algorithm, designed, developed, evaluated PMIT-S0L, wrote the first draft of the paper, and helped with the revision process. IM supervised the research, designed, developed, evaluated PMIT-S0L, wrote the first draft of the paper, and helped with the revision process. IM supervised the research.

Funding This research was conducted as part of the Plant Phenotyping and Imaging Research Center, funded in part by the Canada First Research Fund, by National Science Engineering Research Council grant #2016-06172 (IM), and the Alexander Graham Bell Scholarship (JB).

Data availability statement One of the L-systems used in the data set is published, and appropriate citations are provided. The remainder are made by procedural generation, and so really belong as part of the code. The L-systems used as the basis for making the procedural generation routine are stored in the University of Calgary’s L-system repository, which we accessed with their permission.

Code Availability PMIT-S0L is open source and available at https://github.com/jbernard-McMaster/PMIT.

Declarations

Conflict of interest We have no conflicts of interest.

Appendix A

This appendix contains some S0L-systems procedurally generated for this research and the resulting L-systems predicted by PMIT-S0L for a different number of input strings. Turtle graphics symbols were not included as it has been shown that including turtle graphics symbols makes inferring L-systems inductively easier (Bernard and McQuillan 2021).

| Table 4 Three sample procedurally generated S0L-systems and solution with | Original system (S) | Candidate (C) |
|-----------------------------|----------------------|----------------|
| A → AABA, p = 0.31         | A → AABA, p = 0.30   |
| A → AA, p = 0.69           | A → AA, p = 0.70     |
| B → BABAB, p = 1.00        | B → BABAB, p = 1.00  |
| A → BBA, p = 0.08          | A → BBA, p = 0.06    |
| A → AABA, p = 0.92         | A → AABA, p = 0.94   |
| B → AAAAA, p = 0.28        | B → AAAAA, p = 0.31  |
| B → BBAABAB, p = 0.72      | B → BBAABAB, p = 0.69|
| A → BBAABAAA, p = 0.30     | A → BBAABAA, p = 0.29|
| A → AA, p = 0.11           | A → AA, p = 0.11     |
| A → BBA, p = 0.59          | A → BBA, p = 0.60    |
| B → BB, p = 0.56           | B → BB, p = 0.52     |
| B → AAAABB, p = 0.17       | B → AAAABB, p = 0.21 |
| B → AAA, p = 0.27          | B → AAA, p = 0.27    |

All solutions generated with M = 3. The axiom in all cases is A. L-systems separated by double horizontal line

| Table 5 Three sample procedurally generated S0L-systems and solution with | Original system (S) | Candidate (C) |
|-----------------------------|----------------------|----------------|
| A → CA, p = 0.54           | A → CA, p = 0.51     |
| A → CCBCCCCC, p = 0.46     | A → CCBCCCCC, p = 0.49|
| B → CCAABAAACBA, p = 0.63  | B → CCAABAAACBA, p = 0.64|
| C → ACCCC, p = 1.00        | B → ACCCC, p = 1.00  |
| A → CBCBC, p = 0.42        | A → CBCBC, p = 0.51  |
| A → CBB, p = 0.41          | A → CBB, p = 0.38    |
| A → CB, p = 0.17           | A → CB, p = 0.11     |
| B → BBC, p = 0.28          | B → BBC, p = 0.28    |
| B → BBBCA, p = 0.72        | B → BBBCA, p = 0.72  |
| B → BCC, p = 0.28          | B → BCC, p = 0.31    |
| C → ACABB, p = 1.00        | C → ACABB, p = 1.00  |
| A → CB, p = 0.72           | A → CB, p = 0.65     |
| A → BBBBBCBBC, p = 0.28    | A → BBBBBCBBC, p = 0.35|
| B → BBBBBCBB, p = 1.00     | A → BBBBBCBB, p = 1.00|
| C → CA, p = 0.33           | C → CA, p = 0.32     |
| C → BBB, p = 0.67          | C → BBB, p = 0.68    |

All solutions generated with M = 3. The axiom in all cases is A. L-systems separated by double horizontal line

In all cases, the axiom for each system was set to “A”, and M = 3. The solution for each system was created by executing PMIT-S0L once. See Tables 4, 5, 6, 7, and 8.
Table 6 One sample procedurally generated S0L-systems and solution with $|V| = 5$

| Original system (S) | Candidate (C) |
|---------------------|---------------|
| $A \rightarrow DDAACD$, $p = 1.00$ | $A \rightarrow DDAACD$, $p = 1.00$ |
| $B \rightarrow CDDBC$, $p = 1.00$ | $B \rightarrow CDDBC$, $p = 1.00$ |
| $C \rightarrow CCD$, $p = 1.00$ | $C \rightarrow CCD$, $p = 1.00$ |
| $D \rightarrow DD$, $p = 0.43$ | $D \rightarrow DD$, $p = 0.40$ |
| $D \rightarrow DBD$, $p = 0.24$ | $D \rightarrow DBD$, $p = 0.25$ |
| $D \rightarrow BBEEDDEED$, $p = 0.33$ | $D \rightarrow BBEEDDEED$, $p = 0.35$ |
| $E \rightarrow ABBC$, $p = 0.25$ | $E \rightarrow ABBC$, $p = 0.29$ |
| $E \rightarrow BC$, $p = 0.75$ | $E \rightarrow BC$, $p = 0.71$ |

All solutions generated with $M = 3$. The axiom in all cases is $A$.

Table 7 One sample procedurally generated S0L-system and solution with $|V| = 6$

| Original system (S) | Candidate (C) |
|---------------------|---------------|
| $A \rightarrow CCCBA$, $p = 0.15$ | $A \rightarrow CCCBA$, $p = 0.20$ |
| $A \rightarrow BBBBBA$, $p = 0.37$ | $A \rightarrow BBBBBA$, $p = 0.33$ |
| $A \rightarrow DBDBBEEE$, $p = 0.48$ | $A \rightarrow DBDBBEEE$, $p = 0.47$ |
| $B \rightarrow DDC$, $p = 0.84$ | $B \rightarrow DDC$, $p = 0.86$ |
| $B \rightarrow CFCFCCFC$, $p = 0.16$ | $B \rightarrow CFCFCCFC$, $p = 0.14$ |
| $C \rightarrow DD$, $p = 0.57$ | $C \rightarrow DD$, $p = 0.58$ |
| $C \rightarrow DDDDDE$, $p = 0.43$ | $C \rightarrow DDDDDE$, $p = 0.42$ |
| $D \rightarrow CECE$, $p = 0.51$ | $D \rightarrow CECE$, $p = 0.50$ |
| $D \rightarrow EECCE$, $p = 0.49$ | $D \rightarrow EECCE$, $p = 0.50$ |
| $E \rightarrow DDGF$, $p = 1.00$ | $E \rightarrow DDGF$, $p = 1.00$ |
| $F \rightarrow AABA$, $p = 1.00$ | $F \rightarrow AABA$, $p = 1.00$ |

The solution was generated with $M = 3$. The axiom in all cases is $A$.

Table 8 One sample procedurally generated S0L-system and solution with $|V| = 7$

| Original system (S) | Candidate (C) |
|---------------------|---------------|
| $A \rightarrow CB$, $p = 0.34$ | $A \rightarrow CB$, $p = 0.32$ |
| $A \rightarrow EA$, $p = 0.59$ | $A \rightarrow EA$, $p = 0.59$ |
| $A \rightarrow DGDD$, $p = 0.07$ | $A \rightarrow DGDD$, $p = 0.09$ |
| $B \rightarrow BCCCC$, $p = 0.50$ | $B \rightarrow BCCCC$, $p = 0.55$ |
| $B \rightarrow CCGG$, $p = 0.50$ | $B \rightarrow CCGG$, $p = 0.45$ |
| $C \rightarrow EEDDEE$, $p = 0.33$ | $C \rightarrow EEDDEE$, $p = 0.34$ |
| $C \rightarrow DDH$, $p = 0.67$ | $C \rightarrow DDH$, $p = 0.66$ |
| $D \rightarrow BBEEB$, $p = 1.00$ | $D \rightarrow BBEEB$, $p = 1.00$ |
| $E \rightarrow AAG$, $p = 1.00$ | $E \rightarrow AAG$, $p = 1.00$ |
| $F \rightarrow BHH$, $p = 1.00$ | $F \rightarrow BHH$, $p = 1.00$ |
| $G \rightarrow DA$, $p = 1.00$ | $G \rightarrow DA$, $p = 1.00$ |

The solution was generated with $M = 3$. The axiom in all cases is $A$.

References

Agu M, Yokoi Y (1985) A stochastic description of branching structures of trees. J Theor Biol 112(4):667–676

Al-Khityat MAR, Ahmed M (2017) Matching UML class diagrams using a hybridized greedy-genetic algorithm. In: 2017 12th International scientific and technical conference on computer sciences and information technologies (CSIT), vol. 1, pp. 161–166. IEEE

Back T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming. Oxford University Press, Genetic Algorithms

Ben-Naoum F (2009) A survey on L-system inference. INFOCOMP J Comput Sci 8(3):29–39

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13:281–305

Bernard J, McQuillan I (2018) Inferring stochastic L-systems using a hybrid greedy algorithm. In: Proceedings of the 30th international conference on tools with artificial intelligence, pp. 600–607. IEEE

Bernard J, 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, vol. 10835, pp. 13–25. Springer

Bernard J, McQuillan I (2021) Techniques for inferring context-free Lindenmayer systems with genetic algorithm. Swarm Evol Comput 64:100893

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Danks G, Stepney S, Caves L (2008) Protein folding with stochastic L-systems. In: 11th International conference on the simulation and synthesis of living systems, pp. 150–157

Das Choudhury S, Bashyam S, Qiu Y, Samal A, Awada T (2018) Holistic and component plant phenotyping using temporal image sequence. Plant Methods 14(1):35

De La Higuera C (2005) A bibliographical study of grammatical inference. Pattern Recogn 38(9):1332–1348

De La Higuera C (2010) Grammatical inference: learning automata and grammars. Cambridge University Press

Doucet P (1974) The syntactic inference problem for DOL-sequences. L Systems pp. 146–161

Edwards C (2021) The best of NLP. Commun ACM 64(4):9–11

Eichhorst P, Savitch WJ (1980) Growth functions of stochastic Lindenmayer systems. Inf Control 45(3):217–228

Feliciangeli H, Herman GT (1973) Algorithms for producing grammars from sample derivations: a common problem of formal language theory and developmental biology. J Comput Syst Sci 7(1):97–118

Galarrreta-Valverde MA, Macedo MM, Mekkaoui C, Jackowski M (2013) Three-dimensional synthetic blood vessel generation using stochastic L-systems. In: Medical imaging: image processing, p. 866911

González MA, Sierra M, Vela CR, Varela R (2005) Genetic algorithms hybridized with greedy algorithms and local search over the spaces of active and semi-active schedules. In: Conference of the Spanish association for artificial intelligence, pp. 231–240. Springer

Guo J, Jiang H, Benes B, Deussen O, Zhang X, Lischinski D, Huang H (2020) Inverse procedural modeling of branching structures by inferring L-systems. ACM Transact Gr 39(5):1–13

Ho YC, Pepyne DL (2002) Simple explanation of the no-free-lunch theorem and its implications. J Optim Theory Appl 115(3):549–570

Holliday DJ, Samal A (1995) Object recognition using L-system frac-tals. Pattern Recogn Lett 16:33–42

Khan NA, Lyon OA, Eramian M, McQuillan I (2020) A novel technique combining image processing, plant development properties, and the Hungarian algorithm, to improve leaf detection in maize. In:
Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops, pp. 74–75

Lari K, Young SJ (1990) The estimation of stochastic context-free grammars using the inside-outside algorithm. Comput Speech Lang. 4(1):35–56

Lindenmayer A (1968) Mathematical models for cellular interaction in development, parts I and II. J Theor Biol 18(3):280–315

McQuillan I, Bernard J, 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, vol. 10867, pp. 117–130. Springer International Publishing

Mündermann L, Erasmus Y, Lane B, Coen E, Prusinkiewicz P (2005) Quantitative modeling of Arabidopsis development. Plant Physiol 139:960–968

Méch R, Prusinkiewicz P (1996) Visual models of plants interacting with their environment. Proceedings of SIGGRAPH 1996:397–410

Nevill-Manning CG, Witten IH (1997) Linear-time, incremental hierarchy inference for compression. In: Proceedings DCC’97. Data compression conference, pp. 3–11. IEEE

Nishida T (1980) K0L-system simulating almost but not exactly the same development-case of japanese cypress. Mem Fac Sci 8(1):97–122

Parikh RJ (1966) On context-free languages. J ACM 13(4):570–581

Prusinkiewicz P, Cieslak M, Ferraro P, Hanan J (2018) Mathematical modelling in plant biology, chap. Modeling Plant Development with L-Systems, pp. 139–169. Springer

Prusinkiewicz P, Karwowski R, Lane B (2007) The L+C plant modelling language. Funct-Struct Plant Modell Crop Prod 22:27–42

Prusinkiewicz P, Lindenmayer A (1990) The algorithmic beauty of plants. Springer Verlag, New York

Prusinkiewicz P, Mündermann L, Karwowski R, 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, pp. 289–300. ACM

Prusinkiewicz P, Runions A (2012) Computational models of plant development and form. New Phytol 193:549–569

Rongier G, Collon P, Renard P (2017) Stochastic simulation of channelized sedimentary bodies using a constrained L-system. Comput Geosci 105:158–168

Rozenberg G, Salomaa A (2012) Lindenmayer systems: impacts on theoretical computer science, computer graphics, and developmental biology. Springer Science & Business Media

Runqiang B, Chen P, Burrage K, Hanan J, Room P, 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, pp. 514–524. Springer

Št’ava O, Beneš B, Méch R, Aliaga DG, Krištof P (2010) Inverse procedural modeling by automatic generation of L-systems. In: Computer graphics forum, vol. 29, pp. 665–674. Wiley Online Library

Ubbens J, Cieslak M, Prusinkiewicz P, Stavness I (2018) The use of plant models in deep learning: an application to leaf counting in rosette plants. Plant Methods 14(1):6

University of Calgary: Algorithmic Botany. http://algorothmicbotany.org

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.