Algorithms for Inferring Context-Sensitive L-systems

Ian McQuillan[1], Jason Bernard[1], and Przemyslaw Prusinkiewicz[2]

[1] Department of Computer Science, University of Saskatchewan
Saskatoon, SK, Canada
`mcquillan@cs.usask.ca, jason.bernard@usask.ca`

[2] Department of Computer Science, University of Calgary
Calgary, AB, Canada
`pwp@ucalgary.ca`

Abstract. Lindenmayer systems (L-systems) are parallel string rewriting systems (grammars). By attaching a graphical interpretation to the symbols in the derived strings, they can be applied to create simulations of temporal processes, and have been especially successful in the modeling of plants. With the objective of automatically inferring L-system models in mind, here we study the inductive inference problem: the inference of models from observed strings. Exact algorithms are given for inferring L-systems that can generate input strings for both deterministic context-free and deterministic context-sensitive L-systems. The algorithms run in polynomial time assuming a fixed number of alphabet symbols and fixed context size. Furthermore, if a specific matrix calculated from the input words is invertible, then a context-sensitive L-system can be automatically created (if it exists) in polynomial time without assuming any fixed parameters.

1 Introduction

Lindenmayer systems (L-systems) are formal grammars that repeatedly rewrite strings. By definition, L-system rules are applied to each letter of a string in parallel to produce a new string, and the process is repeated on the new string. L-systems can be deterministic or non-deterministic, context-free or context-sensitive, and parameterized or non-parameterized. Theoretical properties of L-systems have been reviewed in [1,2].

By attaching a graphical interpretation to the symbols, L-systems can generate geometric objects (models). This is typically done via *turtle interpretation* wherein the turtle has a state consisting of its position and orientation, and specific symbols of the L-system provide instructions for moving, drawing, and turning in 2D [3] and 3D [2]. L-systems with turtle interpretation have been especially successful in the modelling of plants [2,4,5].

There has been relatively less work on methods for inferring L-systems. There are certain useful techniques for manually inferring models from images and real plants by experts [5], but existing approaches to automatically infer models from data have limitations; see survey [6]. One could imagine automatically inferring models from sequences of images over time, and this has been attempted in a preliminary fashion [7]. Acquiring such images digitally is now quite practical, from cameras in fields taking pictures periodically, to more complicated camera and sensor setups used in greenhouses, which can create point clouds representations of the plants [8]. Inference of models would be a useful step towards digitally characterizing plants, understanding the differences between them, and even breeding or designing new (real) plants from models. An intermediate step is to infer L-system models from strings that describe the plant structure. That is, given a sequence of strings produced by an unknown L-system, can the L-system itself be inferred? This problem is known as *inductive inference*. In [9], an algorithm was provided that used letter occurrence arithmetic and matrix inversion to infer a deterministic context-free L-system (D0L-system) from an initial sequence of strings. A related technique was implemented in [10], which infers D0L-systems for alphabets with at most two symbols, while calling the problem "immensely complicated" for alphabets of larger size. Inference of D0L-systems from branching structures was investigated in [11]. There has also been some investigation on inference where the given strings are non-consecutive [6,9,11,12]. In [13], an algorithm is given that infers hierarchical structure from a string by replacing repeated phrases with D0L-system rules.

In this paper, we review and extend selected algorithms for inductive inference of L-systems, and analyze their time complexity. Fixed-parameter-tractable algorithms are those that run in polynomial time if one assumes that a parameter is fixed [14]. We propose a fixed-parameter-tractable algorithm that can always infer a deterministic context-sensitive (or context-free) L-system from sequential data, if such a system exists. This algorithm runs in polynomial time for context-free systems, assuming the alphabet is of fixed size. For context-sensitive systems, it runs in polynomial time assuming that the context size is also fixed (ie. there are constants k and l such that each L-system rule only depends on at most k symbols of left context and l symbols of right context). Furthermore, a speedup to this algorithm is described by using letter occurrence arithmetic (similar to [9] for context-free systems). In particular, if a matrix defined using a generalization of the Parikh map from letters to subwords calculated on the input words is invertible, then a context-sensitive L-system can be inferred in polynomial time without fixing any parameters.

2 Preliminaries

This section provides definitions of the terms and symbols used throughout the paper.

The set of integers (positive integers, non-negative integers) is denoted by $\mathbb{Z}$ (respectively $\mathbb{N}$, $\mathbb{N}_0$). Given a vector $\boldsymbol{v}$, let $\boldsymbol{v}(i)$ be the ith component of $\boldsymbol{v}$. If M is a matrix, let $M_{*,j}$ be the column vector for the jth column of M. If X is a finite set, then $|X|$ is the number of elements in X.

An *alphabet* is a finite set of symbols. If V is an alphabet, then V^* is the set of all strings (or words) using letters from V. A *language* L is any subset of V^*. The *length* of a word w is denoted by $|w|$, and for any letter $a \in V$, $|w|_a$ is the number of occurrences of a's in w. Also, $V^i = \{w \mid w \in V^*, |w| = i\}$ and $V^{\leq i} = \{w \mid w \in V^*, |w| \leq i\}$. For a language L, $\text{alph}(L) = \{a \in V \mid w \in L, |w|_a > 0\}$. Given a fixed ordering of the letters of V, $V = \{a_1, \ldots, a_k\}$, the *Parikh map* of w, $\psi(w)$, is the vector $(|w|_{a_1}, \ldots, |w|_{a_k})$. If $w \in V^*$, a *subword* of w is any y such that $w = xyz$, $x, z \in V^*$. If $w = yz$, then y is a *prefix* of w and z is a *suffix* of w. If $1 \leq i \leq j \leq |w|$, then $w[i, j]$ is the substring between positions i and j of w.

A *context-free L-system* ($\mathsf{0L}$-system) is one in which productions are applied to symbols regardless of their context within the string. The $\mathsf{0L}$-system is denoted by $G = (V, \omega, P)$, where V is an alphabet, $\omega \in V^*$ is the *axiom*, and $P \subseteq V \times V^*$ is a finite set of *productions*. A production $(a, x) \in P$ is denoted by $a \rightarrow x$. The letter a is referred to as the production *predecessor*, and the word x as its *successor*. We assume that for each predecessor $a \in V$ there is at least one production $a \rightarrow x$ in P. The system $\mathsf{0L}$-system G is said to be deterministic ($\mathsf{D0L}$-system) if for each $a \in V$ there is exactly one such production. Given a word $\mu = a_1 \ldots a_m \in V^*$, we write $\mu \Rightarrow \nu$ and say that μ directly derives ν if $\nu = x_1 \cdots x_m$, where $a_i \rightarrow x_i \in P$ for all $1 \leq i \leq m$. The (not necessarily direct) derivation $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$ (the result of applying $\Rightarrow$ zero, one, or more times). The *language generated* by a $\mathsf{0L}$-system G is $L(G) = \{w \mid \omega \Rightarrow^* w\}$. The *developmental sequence* of length $n \in \mathbb{N}_0$ is the sequence of words $(w_1, w_2, \ldots, w_n)$, such that $\omega = w_1 \Rightarrow w_2 \cdots \Rightarrow w_n$. We call w_n the nth word generated by G.

In contrast to $\mathsf{0L}$-systems, the production chosen for each symbol in a context-sensitive L-system may depend on the surrounding symbols. Given $k, l \in \mathbb{N}_0$ such that $k + l > 0$, a deterministic (context-sensitive) (k, l)-system is a tuple $G = (V, \omega, P)$, where V and ω are the alphabet and axiom, respectively, and P is a finite set of productions of the form $u < a > v \rightarrow x$. We assume that $a \in V$, $u, v, x \in V^*, |u| \leq k$, and $|v| \leq l$. The letter a is called the *strict predecessor*, and the words u and v are the *left* and *right context*, respectively. If several context-sensitive productions could potentially be applied to the same symbol due to differently sized contexts, we assume that the production with the longest applicable left context, and then the longest applicable right context, will be chosen. In a deterministic (k, l)-system there is thus exactly one production that can be applied to any letter in any given context. If $\mu = a_1 \ldots a_m \in V^*$, we write $\mu \Rightarrow \nu$ if $\nu = x_1 \cdots x_m$ and for any $i = 1, \ldots, m$, the production $u_i < a_i > v_i \rightarrow x_i$ belongs to P, the left context u_i is the longest suffix of $a_1 \cdots a_{i-1}$, and the right context v_i is the longest prefix of $a_{i+1}a_{i+2} \cdots a_m$ among all the productions in P with the strict predecessor a_i.

3 Inferring L-Systems

In this section we study the problem of inferring an L-system from an initial sequence of words assumed to have been generated by it. We begin with the simplest case, the inference of $\mathsf{D0L}$-systems, and use the resulting algorithms in the more complex case of context-sensitive L-systems.

3.1 Deterministic Context-free L-Systems

We define the following problem:

> $\mathsf{D0L}$ **Inductive inference problem**: Given alphabet $V = \{a_1, \dots, a_m\}$ and a sequence of n words over V, $\varrho = (w_1, \dots, w_n)$, $n > 1$, determine a $\mathsf{D0L}$-system G that generates ϱ as the developmental sequence of length n.

We say that G is *compatible* with ϱ if the developmental sequence of length n in G is ϱ. We also denote $S(\varrho)$ as the sum of the lengths of words in ϱ: $S(\varrho) = |w_1| + \cdots + |w_n|$. This is used when defining the time complexity of the algorithms in the paper. Before presenting the full algorithm for inductive inference, an intermediate algorithm is needed which is used within the full algorithm. The intermediate algorithm is provided with the lengths of the production successors for each letter of the alphabet as input, and it is able to determine whether the input words can produce a $\mathsf{D0L}$-system with these successor lengths.

Proposition 1. *Given an alphabet $V = \{a_1, \dots, a_m\}$, a sequence $\varrho = (w_1, \dots, w_n)$ of n words over V such that $V = \mathrm{alph}(\{w_1, \dots, w_{n-1}\})$ and a set of integers $j_1, \dots, j_m \in \mathbb{N}_0$, there exists at most one $\mathsf{D0L}$-system G over V that is compatible with ϱ and satisfies condition $j_i = |x_i|$ for each production $a_i \to x_i \in P$. Furthermore, G can be determined in $\mathsf{O}(S(\varrho))$ time.*

A constructive proof of this proposition is given by Algorithm 1. It scans the first letter of w_1, say a_i, and creates a production such that $a_i \to x_i$, where x_i consists of the first j_i letters of w_2. The algorithm continues with the second letter of w_1 and the subsequent letters of w_2. As each new letter is encountered, a production is added to the production set P. After processing all letters of w_1, the algorithm proceeds in the same way for all pairs of consecutive words $w_p, w_{p+1} \in \varrho$, $p \leq n-1$. Since $V = \mathrm{alph}(\{w_1, \dots, w_{n-1}\})$, every letter as well as its successor will be encountered, ensuring that every production has been determined. The result is a $\mathsf{D0L}$-system G compatible with the developmental sequence ϱ. The algorithm will fail if a letter is encountered for which a production has already been found and the subsequent letters of the next word do not match the production successor. It can be seen that this algorithm runs in $\mathsf{O}(S(\varrho))$ time.

Proposition 2. *Consider alphabet $V = \{a_1, \dots, a_m\}$ and a sequence $\varrho = (w_1, \dots, w_n)$ of $n > 1$ words over V. Let k_i, $1 \leq i \leq m$ be one plus the length of the first word in ϱ following the word in which a_i occurs for the first time. A $\mathsf{D0L}$-system G compatible with ϱ can then be found or reported as non-existent in the worst-case time $\mathsf{O}((k_1 k_2 \cdots k_m) \cdot S(\varrho))$.*

The constructive proof of this proposition consists of Algorithm 2 that uses a brute force approach to find a $\mathsf{D0L}$-system compatible with the developmental

Algorithm 1 Determines the unique D0L-system G compatible with ϱ, if it exists.

Input: Alphabet $V = \{a_1, \ldots, a_m\}$, sequence of words $\varrho = (w_1, \ldots, w_n)$ such that $V = \text{alph}(\{w_1, \ldots, w_{n-1}\})$, and the set $\beta = \{j_1 \ldots, j_m\}$ of the successor length for each letter a_i.

Output: The D0L-system G over V compatible with ϱ, if one exists, or $\emptyset$ otherwise,

```
 1: Let $x_1, \ldots, x_m$ be string variables set to null
 2: for $p$ from 1 to $n-1$ do
 3:     Let $r \leftarrow 1$
 4:     for $q$ from 1 to $|w_p|$ do
 5:         Let $i$ be such that $a_i$ is equal to $w_p[q]$
 6:         if $x_i$ is null then
 7:             $x_i \leftarrow w_{p+1}[r, r+j_i-1]$
 8:         else if $x_i \neq w_{p+1}[r, r+j_i-1]$ then
 9:             return $\emptyset$
10:         end if
11:         $r = r + j_i$
12:     end for
13: end for
14: return D0L-system with axiom $w_1$ and production set $P = \{a_i \to x_i \mid 1 \le i \le m\}$.
```

sequence ϱ by trying all possible combinations of successor lengths. The algorithm begins by determining, for each letter $a_i \in V$, the first word $w_p \in \varrho$ in which a_i occurs. This requires time $\mathsf{O}(S(\varrho))$. Any D0L-system potentially compatible with ϱ has successor x_i of symbol a_i of length $|w_{p+1}|$ at most. There are $k_i = |w_{p+1}| + 1$ (including zero) possible values for the length $j_i = |x_i|$ of this successor, and $k_1 \cdot k_2 \cdots k_m$ possible combinations of the lengths $j_1, \ldots, j_m$, $0 \le j_i \le k_i - 1$, overall. For each such combination, Algorithm 2 calls Algorithm 1 to find a D0L-system compatible with ϱ. If, in some iteration, such a D0L-system is found, it is reported as the output of Algorithm 2. In the opposite case, the algorithm reports that no D0L-system compatible with the given sequence ϱ exists. As each iteration of Algorithm 1 requires $\mathsf{O}(S(\varrho))$ time, the overall complexity of Algorithm 2 is $\mathsf{O}(k_1 \cdot k_2 \cdots k_m \cdot S(\varrho))$ (or $\mathsf{O}(S(\varrho)^{m+1})$ as an upper bound). This time grows exponentially as m increases, but, if m is taken to be a fixed variable, the algorithm has polynomial time complexity with respect to $S(\varrho)$.

The construction presented in [15] with the goal of determining decidability of the D0L-system existence is very similar to the algorithm given here. We use Algorithm 2 as a subroutine within the context-sensitive algorithms later in the paper.

Different D0L-systems may generate the same sequence ϱ. For instance, the sequences generated by the systems $G_1 = (\{A, B, C\}, AB, \{A \to C, B \to AB, C \to C\})$ and $G_2 = (\{A, B, C\}, AB, \{A \to CA, B \to B, C \to C\})$ are the same. Algorithm 2 can be easily modified to determine every compatible D0L-system. Instead of returning a D0L-system G as soon as one is found, output G and continue the procedure.

Algorithm 2 Solves the D0L inference problem

```
Input: alphabet $V = \{a_1, \ldots, a_m\}$, sequence of words $\varrho = (w_1, \ldots, w_n)$,
Output: a D0L-system $G$ over $V$ that gives $\varrho$ as the first $n$ words generated, or $\emptyset$ if
    none exists
 1: Let $y_1, \ldots, y_m$ be integer variables set to $-1$
 2: for each letter position $q$ of each word $w_p$ where $p$ is from 1 to $n-1$ do
 3:     Let $i$ be such that $a_i$ is equal to $w_p[q]$
 4:     if $y_i$ is equal to $-1$ then
 5:         $y_i \leftarrow p$
 6:     end if
 7: end for
 8: for each vector $\beta = (j_1, \ldots, j_m)$, where $0 \leq j_i \leq |w_{y_i+1}|$ do
 9:     set $G$ to the output of Algorithm 1 with $\varrho$ and $\beta$
10:     if $G \neq \emptyset$ then
11:         return $G$
12:     end if
13: end for
14: return $\emptyset$
```

Corollary 1. *For a fixed alphabet V, and sequence of words $\varrho = (w_1, \ldots, w_n)$, $n > 1$ such that $V = \mathrm{alph}(\{w_1, \ldots, w_{n-1}\})$, there is an algorithm to find every* D0L*-system over V that is compatible with ϱ in polynomial time $S(\varrho)$.*

3.2 Deterministic Context-Sensitive L-Systems

We now address the inductive inference problem for context-sensitive L-systems.

> **Deterministic context-sensitive inductive inference problem**: Given an alphabet $V = \{a_1, \ldots, a_m\}$, context lengths $k, l \in \mathbb{N}$, and a sequence $\varrho = (w_1, \ldots, w_n)$ of $n > 1$ words over V, find a deterministic (k,l)-system G that generates ϱ as the developmental sequence of length n.

Similarly to the case of D0L systems, we say that G is compatible with an input sequence $\varrho = (w_1, \ldots, w_n)$ if ϱ is the developmental sequence of length n in G.

Algorithm 1 can be extended to deterministic (k,l)-systems by replacing productions of the form $a \to x$ with productions of the form $u < a > v \to x$, for each substring uav of length $k + l + 1$ appearing in ϱ. The first k symbols then are the left context u, the next symbol is the strict predecessor a, and the last l symbols are the right context v. The inference process thus involves "sliding a window" of length $k + l + 1$ over each word of $w_1, \ldots, w_{n-1}$. In addition, separate productions with shorter contexts are considered near the beginning and end of each word w_i. As discussed in Section 2, when applying rules for a context-sensitive L-system, the production with the longest applicable left context and then the longest applicable right context will be chosen. The left and right contexts can, however, be shorter than k and l when there are less than that many symbols of context in the letter a in w_i being rewritten, at which point, they are still the longest contexts possible. A deterministic (k,l)-system in which the context are always the longest possible up to the limit k,l is said to be of *maximal context.*

Formalizing these concepts, given a sequence of words $\varrho = (w_1,\dots,w_n)$ over V we define the *set of (k,l)-predecessors in ϱ*, denoted by $\Delta_{k,l}(\varrho)$, as the set of all triplets $u < a > v$ such that:

- $a \in V$,
- uav is a subword of some $w \in \{w_1,\dots,w_{n-1}\}$,
- either $|u| = k$ or $0 \le |u| < k$ and uav is a prefix of w, and
- either $|v| = l$ or $0 \le |v| < l$ and uav is a suffix of w.

We further define a $\Delta_{k,l}(\varrho)$-subset (k,l)-system to be a tuple $G = (V,\omega,P)$, where V is an alphabet, $\omega \in V^*$ is the axiom, and P is a finite set of productions $u < a > v \to x$ such that $u < a > v \in \Delta_{k,l}(\varrho)$. A $\Delta_{k,l}(\varrho)$-subset (k,l)-system is said to be compatible with ϱ if it can generate ϱ as the first sequence of words.

A $\Delta_{k,l}(\varrho)$-subset (k,l)-system may be incomplete, in the sense there may be words generated past w_{n-1} that have a subword uav such that $u < a > v \notin \Delta_{k,l}(\varrho)$. Nevertheless, by taking all remaining words $u < a > v \notin \Delta_{k,l}(\varrho)$ such that $|u| \le k$ and $|v| \le l$, and creating productions $u < a > v \to \chi$ where $\chi \in V^*$ is an arbitrary successor, results in a completely specified deterministic (k,l)-system. It is thus easy to extend any $\Delta_{k,l}(\varrho)$-subset (k,l)-system into a fully specified deterministic (k,l)-system.

Proposition 3. *Given an alphabet $V = \{a_1,\dots,a_m\}$, context sizes k and l, a sequence of n words $\varrho = (w_1,\dots,w_n)$ over V, and a function $f(u,a,v)$ into $\mathbb{N}_0$ defining the length of the successors for predecessors $u < a > v \in \Delta_{k,l}(\varrho)$, there exists at most one $\Delta_{k,l}(\varrho)$-subset (k,l)-system G such that:*

- *G is compatible with ϱ,*
- *$u < a > v \to x_{u,a,v}$ is a production with $f(u,a,v) = |x_{u,a,v}|$.*

Furthermore, G can be determined in $\mathsf{O}((k+l)\cdot S(\varrho))$ time when data is stored in the form of a trie.

Proof. We construct Algorithm 3 that extends Algorithm 1 to the context-sensitive case. The goal is to take the sequence ϱ and a length associated with each predecessor string in $\Delta_{k,l}(\varrho)$ as input (these input length correspond to parameters $j \in \beta$ in Algorithm 1), and determine a $\Delta_{k,l}(\varrho)$-subset (k,l)-system compatible with these lengths. To this end, each predecessor string $u < a > v \in \Delta_{k,l}(\varrho)$ as well as the length of its successor, $f(u,a,v)$, is stored in a trie data structure. The trie enables the lookup of $u < a > v$ information in time linearly proportional to $|uav|$. The algorithm "slides a window" over each word, $w_1,\dots,w_{n-1} \in \varrho$. As each new predecessor $u < a > v$ is encountered, the prescribed length of its successor, stored in the trie, is used to determine the letters from the next word that make up the successor (as done in Algorithm 1). The result is also stored in the trie. If a successor has previously been found, the algorithm compares it with the current candidate. If they do not match, the

$\Delta_{k,l}(\varrho)$-subset (k, l)-system G sought does not exist. Alternatively, if no conflict occurs, the trie contains the inferred $\Delta_{k,l}(\varrho)$-subset (k, l)-system G, The time taken to slide such a window over ϱ is $\mathsf{O}((k + l) \cdot S(\varrho))$. □

Since each $\Delta_{k,l}(\varrho)$-subset (k, l)-system can be extended into a deterministic (k, l)-system by associating all elements not stored in the trie with identity productions, the following is true:

Corollary 2. *Given an alphabet $V = \{a_1, \ldots, a_m\}$, context sizes k and l, a sequence of n words $\varrho = (w_1, \ldots, w_n)$ over V, and a function $f(u, a, v)$ to $\mathbb{N}_0$ defining the length of the successors for predecessors $u < a > v \in \Delta_{k,l}(\varrho)$, a deterministic (k, l)-system G can be found, if it exists, such that:*

– *G is compatible with ϱ,*
– *$u < a > v \to x_{u,a,v}$ is a production with $f(u, a, v) = |x_{u,a,v}|$.*

Furthermore, one such system G, if it exists, can be determined in $\mathsf{O}((k+l)\cdot S(\varrho))$ time.

It is also possible to use Algorithm 3 as as a subprogram, similar to Algorithm 1, to consider all possible successor lengths.

Proposition 4. *Given an m-letter alphabet V, context sizes k and l, and a sequence of words $\varrho = (w_1, \ldots, w_n)$ where q is the longest word in ϱ, a $\Delta_{k,l}(\varrho)$-subset (k, l)-system compatible with ϱ can be found, if one exists, in worst case time $\mathsf{O}(q^{(m+1)^{k+l+1}} \cdot (k + l) \cdot S(\varrho))$.*

Proof. We proceed by constructing Algorithm 4 that extends Algorithm 2 to the context-sensitive case. The algorithm starts by creating an empty trie. It then scans ϱ while sliding a window, such that when reading $u < a > v$, it stores this predecessor as well as the following information associated with it in the trie:

– the first word where $u < a > v$ occurs, denoted by $g(u, a, v)$ with $1 \leq g(u, a, v) \leq n - 1$,
– a unique natural number denoted by $h(u, a, v)$ such that the jth triple added to the trie is assigned j.

Both $g(u, a, v)$ and $h(u, a, v)$ can be determined as it is sliding the window. At the end, it can determine r such that $r = |\Delta_{k,l}(\varrho)|$ and each $h(u, a, v)$ is a unique number in $\{1, \ldots, r\}$. Next, it introduces two vectors α and β with r components. With a depth-first traversal of the trie, when scanning $u < a > v$, the algorithm stores one plus the length of the word following the occurrence of $u < a > v$, $1+|w_{g(u,a,v)+1}|$, at position $h(u, a, v)$ of α; the length of the successor with predecessor $u < a > v$ is strictly less than this amount. The algorithm continues as with Algorithm 2 through all combinations of $\beta = (j_1, \ldots, j_r)$, where $0 \leq j_i < \alpha(i)$, for $1 \leq i \leq r$. There are $\prod_{1\leq i\leq r} \alpha(i)$ such combinations for β, and it calls Algorithm 3 for each. This is $\mathsf{O}((k + l) \cdot S(\varrho) \cdot \prod_{1\leq i\leq r} \alpha(i))$ time. Simplifying by letting q be the longest word in ϱ, we get $\mathsf{O}(q^r \cdot (k + l) \cdot S(\varrho))$. Here, an upper bound for r is $(m+1)^{k+l+1}$ since there are m possibilities for a_i, $(m + 1)^k$ for u (due to prefixes shorter than k), and similarly for v. Hence, the time is $\mathsf{O}(q^{(m+1)^{k+l+1}} \cdot (k + l) \cdot S(\varrho))$. □

This can be done in polynomial time if m, k, and l are fixed. For context-sensitive L-systems in the literature, k and l are often quite small (often only one). Therefore, for these systems, the time mainly depends on the alphabet size.

By associating any undefined elements in the trie with identity productions, the following is true:

Corollary 3. *Given an m-letter alphabet V, context sizes k and l, and a given sequence of words $\varrho = (w_1, \ldots, w_n)$ where q is the longest word in ϱ, a deterministic (k,l)-system compatible with ϱ can be found, if one exists, in worst case $\mathsf{O}(q^{(m+1)^{k+l+1}} \cdot (k+l) \cdot S(\varrho))$ time.*

If the alphabet size and context sizes are fixed, the algorithm runs in polynomial time.

Corollary 4. *For a fixed size alphabet V, fixed k and l context sizes, and a given sequence of words ϱ, there is an algorithm to find a deterministic (k,l)-system compatible with ϱ in polynomial time, $S(\varrho)$.*

4 Speedups Using Letter Occurrence Arithmetic

4.1 Context-Free Case

This section will first present a mathematical approach to speeding up inductive inference of D0L-systems. The idea as applied to D0L systems is in fact already known [9], but we review it here as it helps understand the context-sensitive case. We then extend it to context-sensitive L-systems, for which it was not described before.

Let $G = (V, \omega, P)$ be a D0L system over alphabet $V = \{a_1, \ldots, a_m\}$, x_i the successor of production $a_i \to x_i \in P$, and $x_i^{(j)} = |x_i|_{a_j}$ the number of occurrences of letter a_j in this successor for $1 \leq i, j \leq m$. The *growth matrix* of G, denoted by $M(G)$, is then the $m \times m$ matrix such that position i, j contains $x_i^{(j)}$.

Given a sequence of words $\varrho = (w_1, \ldots, w_n)$ over $V = \{a_1, \ldots, a_m\}$, and $s, r \in \mathbb{N}$ such that $1 \leq s \leq s + r - 1 \leq n$, let $Y_{s,r}(\varrho)$ be the $r \times m$ matrix such that element i, j is $|w_{s+i-1}|_{a_j}$: the number of occurrences of letter a_j in word w_{s+i-1}. In other words, row i of $Y_{s,r}(\varrho)$ is the Parikh map of w_{s+i-1}. We then have:

$$\underbrace{\begin{bmatrix} y_1^{(1)} & y_1^{(2)} & \ldots & y_1^{(m)} \\ y_2^{(1)} & y_2^{(2)} & \ldots & y_2^{(m)} \\ \vdots & & \ddots & \vdots \\ y_m^{(1)} & y_m^{(2)} & \ldots & y_m^{(m)} \end{bmatrix}}_{Y_{1,m}(\varrho)} \underbrace{\begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \ldots & y_2^{(m)} \\ \vdots & & \ddots & \vdots \\ x_m^{(1)} & x_m^{(2)} & \ldots & x_m^{(m)} \end{bmatrix}}_{M(G)} = \underbrace{\begin{bmatrix} y_2^{(1)} & y_2^{(2)} & \ldots & y_2^{(m)} \\ y_3^{(1)} & y_3^{(2)} & \ldots & y_3^{(m)} \\ \vdots & & \ddots & \vdots \\ y_{m+1}^{(1)} & y_{m+1}^{(2)} & \ldots & y_{m+1}^{(m)} \end{bmatrix}}_{Y_{2,m}(\varrho)} \qquad (1)$$

Now, suppose that we are given an initial sequence of words, $\varrho = (w_1, \ldots, w_{m+1})$, generated by an unknown D0L-system G over $V = \{a_1, \ldots, a_m\}$. The growth matrix M of G is a (not necessarily unique) solution to the equation

$$Y_{1,m}(\varrho)M = Y_{2,m}(\varrho). \tag{2}$$

The sum of the entries in row i of M is the length of the presumed successor x_i of a_i. Given this length for each $i = 1, \ldots, m$, we can use Algorithm 1 to infer the $\mathsf{D0L}$-system compatible with ϱ, if it exists. As the word lengths are non-negative integers, only integer solutions to Equation 2 are of interest, i.e., we consider Equation 2 as a system of linear diophantine equations. The general solution to such a system can be calculated in polynomial time (Corollary 5.3c of [16]). The solution space of possible entries in M is further reduced to be finite, because each element $x_i^{(j)}$ must satisfy the inequality $0 \le x_i^{(j)} \le |w_{p+1}|_{a_j}$: the number of occurrences of letter a_j in the successor x_i of a_i cannot exceed the number of occurrences of a_j in the word w_{p+1} derived from a word w_p in which a_i occurs. Although we do not have a quantitative evaluation of the resulting speedup, the use of diophantine equations appears to significantly reduce the number of calls to Algorithm 1, compared to the brute-force Algorithm 2.

An important special case occurs when matrix $Y_{1,m}(\varrho)$ is invertible. The growth matrix $M = Y_{1,m}(\varrho)^{-1}Y_{2,m}(\varrho)$ is then unique. If it contains anything other than non-negative integers, a $\mathsf{D0L}$-system compatible with the given sequence ϱ does not exist. If, in contrast, all elements of M are non-negative integers, Algorithm 1 can find the $\mathsf{D0L}$-system compatible with ϱ (which is then unique) or determine that such a system does not exist, in $\mathsf{O}(S(\varrho))$ time (Proposition 1). Since the inverse of an $m \times m$ matrix can be calculated in $\mathsf{O}(m^{2.376})$ time ([17] combined with later result on faster matrix multiplication [18]), the following is immediate:

Proposition 5. *Given an m-letter alphabet V and a sequence of words $\varrho = (w_1, \ldots, w_{m+1})$ such that $Y_{1,m}(\varrho)$ is invertible, there is an algorithm that determines the unique* $\mathsf{D0L}$ *system compatible with ϱ, or reports that none exists, in time $\mathsf{O}(S(\varrho) + m^{2.376})$. Furthermore, if m is fixed, then the algorithm runs in time $\mathsf{O}(S(\varrho))$.*

Lastly, if more than $m + 1$ words are given as input, as long as there are m consecutive words starting at word i such that $Y_{i,m}$ is invertible, then this is enough to uniquely determine a $\mathsf{D0L}$ system if it exists.

4.2 Context-Sensitive Case

This procedure is extended next to work with deterministic (k, l)-systems.

Let $k, l \in \mathbb{N}$, and consider a deterministic (k, l)-system $G = (V, \omega, P)$ that is maximal context. Next, consider some fixed ordering (such as lexicographic) of all elements in $V^{\le k} < V > V^{\le l}$ (all possible windows including maximal contexts). Let r be the number of these words, and let z_i be the ith such word, for $1 \le i \le r$. Here, if $m = |V|$, then $r \le (m + 1)^{k+l}m$.

Given a word $w \in V^*$, define the (k,l)-windowed Parikh vector of w as the r-coordinate vector $\psi_{k,l}(w)$, where for $1 \leq i \leq r$,

$$\begin{aligned}\psi_{k,l}(w)(i) = |\{q \mid\ & z_i = u < a > v, w[q,s] = uav,\\ & \text{and either } k = |u| \text{ or } |u| < k \text{ and } q = 1,\\ & \text{and either } l = |v| \text{ or } |v| < l \text{ and } s = |w|\}|.\end{aligned}$$

This can be explained as follows: Consider position i of the vector where $z_i = u < a > v$. Intuitively, position i would give the number of times a production with predecessor $u < a > v$ would get applied when rewriting w with maximal contexts. When $|u| = k$ and $|v| = l$, this is the number of times z_i occurs as a subword of w; when $|u| < k$ and $|v| = l$, this is 1 if uav is a prefix of w and 0 otherwise; when $|u| = k$ and $|v| < l$, this is 1 if uav is a suffix of w and 0 otherwise; when $|u| < k$ and $|v| < l$, this is 1 if $uav = w$ and 0 otherwise.

Let i satisfy $1 \leq i \leq r$, let x_i be the string such that $(z_i = u < a > v) \to x_i \in P$, and let $x_i^{(j)} = \psi(x_i)(j)$, for $1 \leq j \leq m$ ($\psi(x_i)$ is the normal Parikh vector). The growth matrix of G, denoted by $M(G)$, is the $r \times m$ matrix:

$$M(G) = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \dots & y_2^{(m)} \\ \vdots & & \ddots & \vdots \\ x_r^{(1)} & x_r^{(2)} & \dots & x_r^{(m)} \end{bmatrix} \tag{3}$$

Let w_i be the ith word derived by G, for $1 \leq i \leq n$ (thus $\omega = w_1$). Furthermore, let $t_i^{(j)} = \psi_{k,l}(w_i)(j)$ for $1 \leq i < n, 1 \leq j \leq r$. Let $s, r \in \mathbb{N}$ be such that $1 \leq s \leq s+r-1 \leq n-1$, and let $T_{s,r}(\varrho)$ be the $r \times r$ matrix such that the element at position i,j is $t_{i+s-1}(j)$ (that is, the rows are $\psi_{k,l}(w_s), \dots, \psi_{k,l}(w_{s+r-1})$). Similar to Equation 1, we have:

$$T_{1,r}(\varrho)M(G) = Y_{2,r}(\varrho). \tag{4}$$

Note $Y_{2,r}(\varrho)$ is an $r \times m$ matrix calculated using the normal Parikh map.

Now, suppose that we are given an initial sequence of words $\varrho = (w_1, \dots, w_n)$ over $V = \{a_1, \dots, a_m\}$ and context sizes k, l, and the goal is to determine if this sequence can be generated by an unknown deterministic (k,l)-system. Then, on input ϱ, an algorithm can scan one word at a time while sliding a window, and make a trie to hold $\Delta_{k,l}(\varrho)$ as with the proof of Proposition 4 (recall $\Delta_{k,l}(\varrho)$ is the set of all triplets $u < a > v$ that occur with maximal contexts in all but the last word of ϱ). Let $r = |\Delta_{k,l}(\varrho)|$. A vector α with r components is calculated. In the trie, when scanning $u < a > v$, the following are stored: the index of the first word where $u < a > v$ occurs, $g(u,a,v)$; and some unique number $h(u,a,v)$ from 1 to r giving a position of α. The length $1 + |w_{g(u,a,v)+1}|$ is stored in position $h(u,a,v)$ of α. Indeed, after all words $u < a > v$ are added to the trie based on ϱ, it is possible to calculate $r = |\Delta_{k,l}(\varrho)|$, and the $h(u,a,v)$ values then provides the fixed ordering of the elements in $\Delta_{k,l}(\varrho)$. This ordering can then be used for the calculation of $\psi_{k,l}(w_i)$.

Assume henceforth that $n \geq r+1$. Then, as with $\mathsf{D0L}$ systems, an intermediate goal is instead to determine all integer matrices M such that

$$T_{1,r}(\varrho)M = Y_{2,r}(\varrho). \tag{5}$$

By Equation 4, if M is the growth matrix of a (k,l)-system that is compatible with ϱ, then M is a solution to this equation. In this case, the sum of the entries of row i gives the length of the successor of the production with predecessor $u < a > v$, where $h(u,a,v) = i$.

Instead of using brute force to try all possibilities of length combinations, the procedure instead calculates the inverse of $T_{1,r}(\varrho)$ if it exists. If the inverse does exist, it solves for M as $T_{1,r}(\varrho)^{-1}Y_{2,r}(\varrho)$. Indeed, if there is a (k,l)-system compatible with ϱ, then M must be the growth matrix of the maximal context (k,l)-system compatible with ϱ. From M, the length of each production is implied, and Proposition 3 then provides an algorithm to assess compatibility. In terms of complexity, the trie can be built in $\mathsf{O}((k+l)\cdot S(\varrho))$ time. The inverse can be computed in $\mathsf{O}(r^{2.376})$ if it exists and so $M = T_{1,r}(\varrho)^{-1} \cdot Y_{2,r}(\varrho)$ can again be computed in $\mathsf{O}(r^{2.376})$ time. Then the row sums can be stored back in the trie, and by using Proposition 3, the unique maximal context $\Delta_{k,l}(\varrho)$-subset (k,l)-system, if it exists, can be computed in time $\mathsf{O}((k+l)\cdot S(\varrho))$ time.

Proposition 6. *Given an m-letter alphabet V, context sizes k,l, and a sequence of words $\varrho = (w_1,\ldots,w_n)$ over V, with $r = |\Delta_{k,l}(\varrho)|$, $n \geq r+1$, and such that $T_{1,r}(\varrho)$ is invertible, there is an algorithm that determines the unique maximal context $\Delta_{k,l}(\varrho)$-subset (k,l)-system compatible with ϱ, or reports that none exists, in time $\mathsf{O}((k+l)\cdot S(\varrho) + r^{2.376})$.*

By setting all unused productions not in the trie to be identity productions, the following is implied:

Corollary 5. *Given an m-letter alphabet V, context sizes k,l, and a sequence of words $\varrho = (w_1,\ldots,w_n)$ over V, with $r = |\Delta_{k,l}(\varrho)|$, $n \geq r+1$, and such that $T_{1,r}(\varrho)$ is invertible, there is an algorithm that determines a deterministic (k,l)-system compatible with ϱ, or reports that none exist, in time $\mathsf{O}((k+l)\cdot S(\varrho) + r^{2.376})$.*

As with $\mathsf{D0L}$-systems, more generally there can be more than one matrix that is a solution to Equation 5. It is again possible to consider Equation 5 as a system of linear diophantine equations. Then for each solution of M, Proposition 3 can be used to assess compatibility. Lastly, to use matrix inversion, if $n > r+1$, then it is only necessary to have $T_{i,r}$ be an invertible matrix for some i in order to apply this approach.

5 Conclusions and Future Directions

In this paper, polynomial time algorithms are provided that solve the inductive inference problem, when the size of the alphabet and the context sizes are fixed.

Then a speedup is provided using letter occurrence arithmetic. For context-sensitive L-systems, if a matrix defined by using a generalization of the Parikh map on the input words gives an invertible matrix, then the context-sensitive system can be inferred in polynomial time in the context lengths and the sum of the input word lengths. This technique can also be used when the matrix is not invertible by using solutions to linear diophantine equations.

Some immediate questions arise from this work. First, is there a complexity class such that inferring different types of L systems where the alphabet size is not fixed is hard for that class? Also, can any approaches presented here work for nondeterministic (or stochastic) L-systems? In addition, from a practical perspective, can these approaches be combined with a computer vision approach to automatically infer L-systems from sequences of images?

Acknowledgements

The research of all authors was supported in part by a grant from the Plant Phenotyping and Imaging Research Centre (P2IRC), and in part by grants from Natural Sciences and Engineering Research Council of Canada (I. McQuillan grant 2016-06172, J. Bernard scholarship, P. Prusinkiewicz grant 2014-05325).

References

1. G. Rozenberg, A. Salomaa, The Mathematical Theory of L Systems, Academic Press, Inc., New York, 1980.
2. P. Prusinkiewicz, A. Lindenmayer, The Algorithimic Beauty of Plants, Springer Verlag, New York, 1990.
3. P. Prusinkiewicz, Graphical applications of L-systems, in: Proceedings of Graphics Interface '86 / Vision Interface '86, 1986, pp. 247–253.
4. P. Prusinkiewicz, Designing and growing virtual plants with L-systems, in: Proceedings of the XXVI International Horticultural Congress. Acta Horticulturae, Vol. 630, 2004, pp. 15–28.
5. P. Prusinkiewicz, L. Mündermann, R. Karwowski, 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, SIGGRAPH '01, ACM, 2001, pp. 289–300.
6. F. Ben-Naoum, A survey on L-system inference, INFOCOMP Journal of Computer Science 8 (3) (2009) 29–39.
7. B. Runqiang, P. Chen, K. Burrage, J. Hanan, P. Room, 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, pp. 514–524.
8. X. Sirault, J. Fripp, A. Paproki, P. Kuffner, C. Nguyen, R. Li, H. Daily, J. Guo, R. Furbank, Plantscan^{TM}: a three-dimensional phenotyping platform for capturing the structural dynamic of plant development and growth, in: 7th International Conference on Functional-Structural Plant Models, 2013, pp. 45–48.

9. P. Doucet, The syntactic inference problem for D0L-sequences, L Systems (1974) 146–161.
10. R. Nakano, N. Yamada, Number theory-based induction of deterministic context-free L-system grammar, in: International Conference on Knowledge Discovery and Information Retrieval, SCITEPRESS, 2010, pp. 194–199.
11. H. Jürgensen, A. Lindenmayer, Inference algorithms for developmental systems with cell lineages, Bulletin of Mathematical Biology 49 (1) (1987) 93–123.
12. H. Feliciangeli, G. T. Herman, Algorithms for producing grammars from sample derivations: a common problem of formal language theory and developmental biology, Journal of Computer and System Sciences 7 (1) (1973) 97 – 118.
13. C. G. Nevill-Manning, I. H. Witten, Identifying hierarchical structure in sequences: A linear-time algorithm, Journal of Artificial Intelligence Research 7 (1) (1997) 67–82.
14. J. Flum, M. Grohe, Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series), Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
15. G. T. Herman, G. Rozenberg, Developmental Systems and Languages, North-Holland Publishing Company, Oxford, 1975.
16. A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons, Inc., New York, NY, USA, 1986.
17. J. R. Bunch, J. E. Hopcroft, Triangular factorization and inversion by fast matrix multiplication, Mathematics of Computation 28 (125) (1974) 231–236.
18. D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, Journal of Symbolic Computation 9 (3) (1990) 251 – 280, computational algebraic complexity editorial.