Computational Complexity of Inferring Deterministic Lindenmayer Systems

CMPT400 Final Report
Student: Sonja Linghui Shan
Supervisors: Ian McQuillan and Christopher Duffy

Department of Computer Science, University of Saskatchewan
Saskatoon, SK, Canada

Abstract. Lindenmayer systems (L-systems) represent a type of formal grammar and are parallel string rewriting systems. L-systems have been used to model plant structure and development. Research efforts in automatically deriving models of plant growth from images have motivated the L-system inference problem. This report analyzes the computational complexity of inferring deterministic L-systems assuming a variable number of alphabet symbols. Reducing from One-in-three 3SAT, we prove that the deterministic context-free L-system inference problem is NP-complete. Furthermore, we show that the deterministic context-sensitive L-system inference problem is in NP and we strongly believe that it is NP-complete assuming fixed context size, for this we show a reduction with an example One-in-three 3SAT problem instance.

1 Introduction

Lindenmayer systems (L-systems) represent a type of formal grammar and are parallel string rewriting systems. The alphabet of an L-system can be interpreted geometrically, this enables L-systems to simulate topological structure and temporal development. Applications of L-systems include modelling plant architecture, synthesizing realistic images of plants, and differentiating plant species. [PH89]

Research efforts also include inferring L-systems. Previously, to obtain plant growth models, researchers measure and record plant features daily, and use which to derive models of growth manually. With developments in cameras and image recognition techniques, we can automate the acquisition of information. However, existing approaches have limitations in automatically inferring models from sequences of images. [MBP18] This inference problem can be abstracted into an L-system inference problem. And the computation complexity of which, assuming a variable number of alphabet symbols, is the focus of this report. We will expand on the definition of L-system inference problems in the following section.

2 Preliminaries

This section provides definitions of the terms and notations used throughout this report following how they are defined by McQuillan, Bernard, and Prusinkiewicz in *Algorithms for Inferring Context-Sensitive L-systems* [MBP18], and provides an introduction to the L-system inference problem.

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. The length of a word w is denoted by $|w|$.

A *context-free L-system* (0L-system) is one in which productions are applied to symbols regardless of their context within the string. The 0L-system is denoted by $G = (V, \omega, P)$, where V is an alphabet, $\omega \in V^*$ is the axiom, and $P \subset V \times V^*$ is a finite set of productions. A production $(a, z) \in P$ is denoted by $a \to z$. The letter a is referred to as the production *predecessor*, and the word z as its *successor*. We assume that for each predecessor $a \in V$ there is at least one production $a \to z$ in P. The 0L-system G is said to be deterministic (D0L-system) if for each $a \in V$ there is exactly one such production. The D0L-system G is said to be propogating (PD0L-system) if $|z| \neq 0$ for all $a \to z$ in P.

In contrast to 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 $u < a > v \to z$. We assume that $a \in V$, $u, v, z \in V^*$, $|u| \leq k$, and $|v| \leq l$, and that there is exactly one production from $u < a > v$. 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.

Following the definitions above, an L-system G can generate a sequence of strings, S, by repeatedly rewriting the axiom ω with production rules in P. The L-system inference problem asks how G can be inferred from S. Our report looks into the computation complexity of such inference for both deterministic context-free and context-sensitive L-systems.

3 Related Work

McQuillan et al. [MBP18] studied the L-system inference problem and presented exact algorithms for inferring both deterministic context-free and context-sensitive L-systems from observed strings. They concluded that these algorithms are of polynomial complexity assuming a fixed number of alphabet symbols and fixed context size. Our report builds on their work, we modify the assumption and let the number of alphabet symbols vary to examine the resulted computational complexity.

Prusinkiewicz and Hanan [PH89] provided background on Lindenmayer systems and its applications in their book. The complexity proofs and discussions recorded by Garey and Johnson [GJ79] alongwith their catalogue of known NP-complete problems helped to guide the proofs in this report.

4 Results

4.1 Complexity of Inferring Deterministic Context-Free L-systems

Proposition 1 *The deterministic context-free L-system inference problem is in NP.*

We separate the problem solving into two steps. The first step is to generate an integer guess for each letter of the alphabet via nondeterminism. Each integer guess represents the length of the successor of a letter. The second step is to verify whether the L-system constructed with the guess is compatible with the observed sequence of strings. We already know the time complexity of the second step is polynomial to the length of the input sequence. [MBP18] Therefore the time complexity of this inference problem depends on the first guess-generating step. We intend to show that a nondeterministic Turning machine can complete this step in polynomial time. We construct a 4-tape nondeterministic Turning machine as follows:

- Let T_0 be the tape containing the input sequence of strings generated by an unknown L-system.
- Let T_1 be the tape containing the alphabet of the input sequence of strings. They are arranged in their order of appearance on T_0.
- Let T_2 be the tape containing the length of the next word for each letter on T_1. The values are recorded in binary.
- Let T_3 be the tape containing the guesses for the letters on T_1. The values are recorded in binary.

The following is such a Turning machine constructed for an example problem. We use $ as delimiter.

```
T0: A$AB$ABA$ABAAB
T1: AB
T2: 10$11
T3: 01$00
```

Consider the operations of this Turning machine:

- For each letter l on T_0, the TM scans T_1 to check whether l is recorded on T_1. If l is already recorded on T_1, then the TM moves on to process the next letter on T_0.

- If l is not recorded on T_1: The TM first adds l to T_1 and 0 to T_2. Then it traverses T_0 and T_2 to count the length of the string immediately after the string l is in. The counting is done via adding one to what is already on T_2. Next, the TM non-deterministically generates a guess on T_3 for l. The letter l corresponds to a value, n, on T_2. n has $k = \log_2 n$ bits. For each of these k bits, the TM non-deterministically chooses to record 0 or 1 on T_3. Therefore producing a guess of k bits for l.

Consider the space complexity of this Turning machine. Let the size of the alphabet be n and the length of the longest word in the input sequence be m.

- The length of T_0 is bounded by the length of the input sequence.
- The length of T_1 is bounded by n.
- The length of T_2 and T_3 are bounded by $n \times \log m$.

Consider the time complexity of this Turning machine. Let the size of the alphabet be n and the length of the longest word in the input sequence be m. For each input letter l on T_0:

- traversing T_1 to find out whether l has been recorded is a $\mathcal{O}(n)$ operation.
- If l is recorded on T_1, the TM moves on to the next input on T_0.
- If l is not recorded on T_1,
  - going through T_0 and performing additions on T_2 to count m is a $\mathcal{O}(m)$ operation, and
  - making a guess involves iterating through the bits of m on T_2 which is a $\mathcal{O}(\log m)$ operation.

Therefore, the time complexity of processing each input letter l is $\mathcal{O}(n)$ or $\mathcal{O}(n)$ + $\mathcal{O}(m)$ + $\mathcal{O}(\log m)$. And the time complexity of processing the whole input sequence and generating a guess is $\mathcal{O}(\text{length of input} \times max(m, n))$ which is polynomial to the length of input. Therefore, the deterministic context-free L-system inference problem is in NP.

Proposition 2 *The deterministic context-free L-system inference problem is NP-complete. Furthermore, the propagating deterministic context-free L-system inference problem is NP-complete.*

Proof. We have shown that the D0L-system inference problem $\in$ NP. To prove NP-completeness, we construct a reduction with the known NP-complete decision problem One-in-three 3SAT with no negated literal [GJ79].

Let U be the set of variables and $C = \{c_1, c_2, ..., c_n\}$ be the set of clauses in an arbitrary instance of One-in-three 3SAT. We must construct a sequence of strings, S, such that a valid D0L-system can be inferred from S if and only if C is satisfiable.

Let the alphabet of S be $U \cup \{\$, *\}$. Let S be a sequence of four strings: $S = s_1, s_2, s_3, s_4$. Let v_1^i, v_2^i, and v_3^i be the three variables in an arbitrary clause c_i. We construct S as follows:

- Let $s_1 = \$v_1^1v_2^1v_3^1\$v_1^2v_2^2v_3^2\$v_1^3...v_3^{n-1}\$v_1^nv_2^nv_3^n\$$. The variables in each clause c_i are concatenated to form strings $v_1^iv_2^iv_3^i$, and these n strings are concatenated and delimited with \$ to form s_1. s_1 contains $3n$ variables (letters from U) and $n+1$ \$'s.
- Let $s_2 = \$*\$*\$...\$*\$$ by concatenating n $*$'s delimited with \$. s_2 contains n $*$'s and $n+1$ \$'s.
- Let $s_3 = s_4 = \$\$\$...\$$ by concatenating $n+1$ \$'s.

Given the above construction, assume that a valid D0L-system can be inferred from S, we are to show that C is satisfiable. Consider s_3 and s_4, the only valid rewriting rule for \$ is $\$ \rightarrow \$$. Given $\$ \rightarrow \$$, the $n+1$ \$'s in s_3 are produced by the $n+1$ \$'s in s_2, therefore the only valid rewriting rule for $*$ is $* \rightarrow \lambda$. Similarly, the $n+1$ \$'s in s_2 are produced by the $n+1$ \$'s in s_1, therefore the ith $*$ in s_2 has to be produced by exactly one of v_1^i, v_2^i, or v_3^i in s_1.

Thus, if a valid D0L-system can be inferred from S, then, for each triple v_1^i, v_2^i, and v_3^i in s_1, exactly one of the three has a $*$ as successor and the other two have successors of length zero. If we set the $*$-producing variable to $True$ and the other two to $False$, c_i is satisfied.

Conversely, assume that C is satisfiable, then, for each c_i, exactly one v_j^i of v_1^i, v_2^i, and v_3^i is set to $True$. If we let v_j^i be the $*$-producing variable and the other two variables have successors of length zero, then the ith $v_1^iv_2^iv_3^i$ substring in s_1 can produce the ith $*$ in s_2. Combining the above with $\$ \rightarrow \$$, we have a valid D0L-system inferred from S.

For the PD0L-systems which do not allow successors of length zero, we already know that the inference problem is in NP, we modify the construction above to prove NP-hardness. Instead of S, we construct S' as follows:

- $s_1' = s_1$ in S.
- $s_2' = \$****\$****\$...\$****\$$. s_2' includes $4n$ $*$'s and $n+1$ \$'s.
- $s_3' = s_2'$.

If a valid PD0L-system can be inferred from S', then, considering s_2' and s_3', $\$ \rightarrow \$$ and $* \rightarrow *$ are the only valid rewriting rules. Given $\$ \rightarrow \$$, the ith 4-$*$-substring in s_2 is produced by the ith triple of v_1^i, v_2^i, or v_3^i in s_1. Since no variable is allowed to have successor of length zero, in order to produce the ith 4-$*$-substring, exactly one v_j^i of v_1^i, v_2^i, or v_3^i has to produce $**$ and exactly two of v_1^i, v_2^i, or v_3^i have to produce $*$. If we set v_j^i to $True$ and the other two variables to $False$, then c_i is satisfied.

Conversely, if C is satisfiable, then, for each c_i, exactly one v_j^i of v_1^i, v_2^i, and v_3^i is set to $True$. If we let v_j^i produce $**$ and the other two variables produce $*$, then ith $v_1^iv_2^iv_3^i$ substring in s_1 produces the ith 4$*$-substring in s_2. Combining these rules with $\$ \rightarrow \$$ and $* \rightarrow *$, we have a valid PD0L-system inferred from S'.

4.2 Complexity of Inferring Deterministic Context-Sensitive L-systems

Proposition 3 *The deterministic context-sensitive L-system inference problem is in NP.*

We intend to show that a nondeterministic Turning machine can solve the guess-generating step for this problem in polynomial time. We construct a 5-tape nondeterministic Turning machine as follows:

- Let T_0 be the tape containing the input sequence of strings generated by an unknown L-system.
- Let T_1 be the tape containing letters from the alphabet as they appearance on T_0. Note that T_1 may have repeating letters if they are found with different contexts.
- Let T_2 be the tape containing the context correspond to each letter on T_1.
- Let T_3 be the tape containing the length of the next word for each letter on T_1 The values are recorded in binary.
- Let T_4 be the tape containing the guesses for the letters on T_1 The values are recorded in binary.

The following is such a Turning machine, partially constructed, for an example problem with left-context length = right-context length = 1. We use \$ as delimiter and e to represent blank context.

```
T0: XUAX$XADAX$XUAAX
T1: X$U$A$X
T2: eXU$XUA$UAX$AXe
T3: 101$101$101$101
T4: 001$100$101$000
```

Consider the operations of this Turning machine:

- For each letter l on T_0, the TM scans T_1 for a match. If a match of l, l', is found on T_1, the TM traverses T_2 to examine whether the context of l' is a match for $u < l > v$. u and v are the left and right contexts of l, respectively. If l is recorded on T_1 and $u < l > v$ is recorded on T_2, then the TM moves on the the next letter on T_0.
- If l is not recorded on T_1 or its corresponding context on T_2 is not $u < l > v$: The TM first add l to T_1, $u < l > v$ to T_2, and 0 to T_3.
  Then it traverses T_0 and T_3 to count the length of the word immediately after the word $u < l > v$ is in. The counting is done via adding one to what is already on T_3.
  Next, the TM non-deterministically generates a guess on T_4. The guess is the length of the successor of $u < l > v$. $u < l > v$ on T_2 corresponds to a value n on T_3. n has $k = \log_2 n$ bits. For each of these k bits, the TM non-deterministically chooses to record 0 or 1 on T_4. Therefore producing a guess of k bits.

Consider the space complexity of this Turning machine. Let the length of input sequence be n. Let the lengths of the left context and the right context be i and j. Let the length of the longest word in the input sequence be m.

- The length of T_0 is bounded by n.
- The length of T_1 is bounded by n. The upper bound handles the case where every letter in the input has an unique context.
- The length of T_2 is bounded by $n \times (i + j)$. T_2 records the corresponding contexts for letters on T_1.
- The length of T_3 and T_4 are bounded by $n \times \log m$.

Consider the time complexity of this Turning machine. Let the length of input sequence be n. Let the lengths of the left context and the right context be i and j. Let the length of the longest word in the input sequence be m. For each input letter l on T_0:

- traversing T_1 to find out whether l has been recorded is a $\mathcal{O}(n)$ operation.
- If l is recorded on T_1,
  - traversing T_2 to find out whether the context for the recorded letter match the context of the input letter l is a $\mathcal{O}(n \times (i + j))$ operation.
  - If $u < l > v$ is recorded on T_2, the TM moves on to the next input on T_0.
- If l is not recorded on T_1 or if the corresponding context on T_2 is not $u < l > v$,
  - going through T_0 and performing additions on T_3 to count m is a $\mathcal{O}(m)$ operation, and
  - making a guess involves iterating through the bits of m on T_3 which is a $\mathcal{O}(\log m)$ operation.

Therefore, the time complexity of processing each input letter l is

- $\mathcal{O}(n) + \mathcal{O}(n \times (i + j)) = \mathcal{O}(n \times (i + j))$, or
- $\mathcal{O}(n) + \mathcal{O}(n \times (i + j)) + \mathcal{O}(m) + \mathcal{O}(\log m) = \mathcal{O}(n \times (i + j))$

And the time complexity of processing the whole input sequence is $\mathcal{O}(n^2 \times (i+j))$. Since i and j are the lengths of the left context and the right context, they are bounded by n. The time complexity can therefore be simplified to $\mathcal{O}(n^3)$ and the problem is therefore in NP.

Proposition 4 *The deterministic context-sensitive L-system inference problem is NP-complete.*

We have shown that the deterministic context-sensitive L-system inference problem is in NP. To prove NP-completeness, we propose a transformation from the version of One-in-three 3SAT decision problem with no negated literal.

Consider an One-in-three 3SAT example problem:

$$C = (A \cup B \cup D) \cap (D \cup F \cup E)$$

Given C, we construct a sequence of strings, $S = s_1, s_2, s_3, s_4, s_5$, such that a valid deterministic context-sensitive $(1, 1)$-system can be inferred from S if and only if C is satisfiable. Let the alphabet of S be $\{\$, \#, *\} \cup$ the alphabet of C. We construct S as follows:

- $s_1 = \$\#\#A\#\#B\#\#D\#\#\$\#\#D\#\#F\#\#E\#\#\$$
- $s_2 = \$\# * \#\$\# * \#\$$
- $s_3 = \$\$\$$
- $s_4 = \$\#\#\$\#\#\$$
- $s_5 = \$\#\#\$\#\#\$$

Let P be the set of production rules that contain a variable in their predecessors. Let V be an arbitrary variable, the rules in P take the following forms:

- $\# < \# > V \rightarrow$
- $\# < V > \# \rightarrow$
- $V < \# > \# \rightarrow$

Let Q be a set containing the following production rules:

- $< \$ > \# \rightarrow$
- $\$ < \# > \# \rightarrow$
- $\# < \# > \$ \rightarrow$
- $\# < \$ > \# \rightarrow$
- $\# < \$ > \rightarrow$

Notice that
† all predecessors in Q are contained both s_1 and s_4, and
‡ $P \cup Q$ contains exactly all available production rules for rewriting s_1.

With these two conditions, we can show that
♣ every $*$ in s_2 is produced with a rule in P, and
♠ every \$ and # are produced with a rule in Q.

To show ♣, we proceed by contradiction.

Proof. Let $q \in Q$ be a rule that produces at least one $*$. Since the predecessor in q is a substring in s_4, s_5 would contain at least one $*$ as a result of applying the rule q. This contradicts with our construction of s_5. Therefore, no $q \in Q$ can produce $*$, this considered with ‡ shows that every $*$ in s_2 is produced with a rule in P.

To show ♠, consider the following cases.

Proof. Let p be an arbitrary rule in P. Consider the #'s in s_2.

- If p produces the first # in s_2, the successors of $< \$ > \#$ and $\$ < \# > \#$ combined is \$. Given ♣ and s_2, the successors of $\# < \# > \$$, $\# < \$ > \#$, and $\$ < \# > \#$ combined is at most #\$#. This considered with † means that s_5 would start with \$#\$#, which contradicts our construction of s_5. Therefore p cannot produce the first # in s_2.

- If p produces the second # in s_2, the successors of $\# < \# > \$$, $\# < \$ > \#$, and $\$ < \# > \#$ combined is at most $\$\#$. This considered with † means that s_5 would contain $\$\#\$\#$, which contradicts our construction of s_5. Therefore p cannot produce the second # in s_2.
- If p produces the third # in s_2, the successors of $\# < \# > \$$, $\# < \$ > \#$, and $\$ < \# > \#$ combined is at most $\#\$$. This would force the successors of $\# < \# > \$$ and $\# < \$ >$ to be $\#\#\$$ so that s_4 can be rewritten into s_5. However, if the successors of $\# < \# > \$$ and $\# < \$ >$ is $\#\#\$$, given †, s_2 would end with $\#\#\$$ which contradicts our construction of s_2. Therefore p cannot produce the third # in s_2.
- If p produces the last # in s_2, the successors of $\# < \# > \$$ and $\# < \$ >$ combined is $\$$. And given ♣, the successors of $< \$ > \#$ and $\$ < \# > \#$ combined is $\$\#$. Then the successors of $\# < \# > \$$, $\# < \$ > \#$, and $\$ < \# > \#$ combined has to be $\#\$\#\#$ so that s_4 can be rewritten into s_5. This considered with † means that s_2 contains $\#\$\#\#$, which contradicts our construction of s_2. Therefore p cannot produce the last # in s_2.

Given the construction of s_1 and s_2, if p cannot include # in its successor, it cannot include $\$$. Since ‡ and no $p \in P$ can produce $\$$ or #, every $\$$ and # are produced with a rule in Q.

With ♣ and ♠, consider the $\#\#A\#\#B\#\#D\#\#$ substring in s_2, it concerns six rules in P: $\#\#A$, $\#A\#$, $A\#\#$, etc., but only one of these six rules can be the one producing the first $*$ in s_2. Similarly, for the next substring delimited with $\$$ in s_2, only one of six rules concerning which can be $*$-producing. Notice that the variables in such a substring correspond to the variables in a clause in C. Therefore, if a valid deterministic $(1, 1)$-system can be inferred from S, then one of A, B, D and one of D, F, E are included in the predecessor of a $*$-producing rule. If we set exactly those variables to true and the others to false, then C is satisfied.

5 Discussion

We showed that the deterministic context-free L-system inference problem is NP-complete assuming a variable number of alphabet symbols. Therefore it is unlikely that a simple and elegant algorithm can be found for it. This can help directing research efforts in automatically inferring L-systems more effectively.

We analyzed the NP-completeness of deterministic context-sensitive L-system inference problems assuming fixed context size. If one assumes variable context size, the computational complexity remains as a question for future research endeavours.

6 Conclusion

In this report, we investigated the computation complexity of deterministic L-system inference problems assuming a variable number of alphabet symbols. We

added to the class of known NP-complete problems by proving that the deterministic context-free L-system inference problem is NP-complete. Additionally, we proved that the deterministic context-sensitive L-system inference problem is in NP. We constructed a reduction with an example One-in-three 3SAT to show that it is highly probable that the deterministic context-sensitive L-system inference problem, assuming fixed context size, is NP-complete.

References

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman amp; Co., 1979. ISBN: 0716710447.

[PH89] P. Prusinkiewicz and James Hanan. *Lindenmayer Systems, Fractals and Plants*. Berlin, Heidelberg: Springer-Verlag, 1989. ISBN: 3540970924.

[MBP18] Ian McQuillan, Jason Bernard, and Przemyslaw Prusinkiewicz. "Algorithms for Inferring Context-Sensitive L-Systems". In: Jan. 2018, pp. 117–130. ISBN: 978-3-319-92434-2. DOI: 10.1007/978-3-319-92435-9_9.