Emergent Induction of L-system Grammar from a String with Deletion-type Transmutation

Ryohei Nakano

Chubu University, 1200 Matsumoto-cho, Kasugai, 487-8501, Japan

Keywords: Grammatical Induction, L-system, Error Correction, Transmutation.

Abstract: L-system is a computational model to capture the growth process of plants. Once a noise-tolerant grammatical induction called LGIC2 was proposed for deterministic context-free L-systems. LGIC2 induces L-system grammars from a transmuted string mY, employing an emergent approach. That is, frequently appearing substrings are extracted from mY to form grammar candidates. A grammar candidate can be used to generate a string Z; however, the number of grammar candidates gets huge. Thus, LGIC2 introduced three pruning techniques to narrow down candidates to get only promising ones. Candidates having the strongest similarities between mY and Z are selected as the final solutions. So far, LGIC2 has been evaluated for replacement- and insertion-type transmutations. This paper evaluates the performance of LGIC2 for deletion-type transmutation, after slightly modifying the method.

1 INTRODUCTION

L-system was introduced by Lindenmayer as a computational model to capture the growth process of plants (Prusinkiewicz and Lindenmayer, 1990). The central concept of L-system is rewriting, and the relationship with fractals led L-systems to various practical applications (Prusinkiewicz and Hanan, 1989).

The reverse process of rewriting is grammatical induction (GI), which discovers grammars from strings. Induction of L-system grammars has been little explored so far. A survey paper (Higuera, 2005) suggested that for many applications it is necessary to be able to cope with noise, but most grammatical inference algorithms are not robust to noise.

L-systems can be classified using two aspects: (1) deterministic or stochastic, and (2) context-free or context-sensitive. Deterministic context-free L-system is the simplest, but very useful class.

Recent research on induction of deterministic context-free L-system is summarized below. A fast induction method called LGIN1 (L-system GI based on Number theory, ver.1) was once proposed (Nakano and Yamada, 2010), employing the number theory. However, it could not cope with a transmuted string. Here, as for transmutation, we consider r(replacement)-type, i(insertion)-type, d(letion)-type, or mixed-type. To cope with a transmuted string, an enumerative method called LGIC1 (L-system GI with error Correction, ver.1) was proposed (Nakano and Suzumura, 2012), where combinations of parameter values are enumerated to form grammar candidates. It worked well for r-type with low transmutation rate, but did not work for i-type or high rate r-type (Nakano, 2013a).

Then, an emergent method called LGIC2 (LGIC, ver.2) was proposed (Nakano, 2013b), where frequently appearing substrings are extracted from a transmuted string to form grammar candidates. In the method, the number of grammar candidates gets huge; thus, three pruning techniques were introduced. LGIC2 worked very nicely for r-type and i-type, overcoming the drawback of LGIC1.

This paper evaluates the performance of LGIC2 for deletion-type transmutation, after slightly modifying the method to fit this type.

2 BACKGROUND

D0L-system

The deterministic context-free L-system is called D0L-system, defined as $G = (V, C, \omega, P)$, where $V$ and $C$ are sets of variables and constants, $\omega$ is an initial string called axiom, and $P$ is a set of production rules. A variable is replaced in rewriting, while a constant remains unchanged and controls turtle graphics.
In this paper we consider the following DOL-system having two rules. Here \( n \) denotes the number of rewritings, while \( A \) and \( B \) denote variables.

\[
\text{axiom} : A, \quad n = ? \\
\text{rule A} : A \rightarrow ??? ????? \\
\text{rule B} : B \rightarrow ??? ??????
\]

We have three strings: \( Y \), \( mY \), and \( Z \). \( Y \) is a normal string generated using the original grammar, while \( mY \) is a string generated by applying transmutation to \( Y \). \( Z \) is a string generated using a grammar candidate, where a grammar candidate means a candidate set of \( n \), rules \( A \) and \( B \).

**Transmutation**

Among four types of transmutation, only d-type is considered here since we want to know how LGIC2 works for d-type. We assume transmutation occurs locally around the center of \( Y \), considering two rates: coverage rate \( P_c \) and occurrence rate \( P_o \). \( P_c \) represents the proportion of transmutation area to the whole \( Y \), while \( P_o \) represents the probability of transmutation in the area. Thus, overall transmutation rate \( P_t \) is calculated as follows:

\[
P_t = P_c \times P_o \tag{1}
\]

**Valid Transmutation**

Simple transmutation will generate an invalid string, which means such a string cannot be drawn through turtle graphics. To keep the transmutation valid, the numbers of left and right square brackets are monitored and controlled if necessary. That is, in the transmutation area the number \( count_l \) of left square brackets should be larger than or equal to the number \( count_r \) of right ones. Moreover, when the transmutation ends, we assure \( count_l \geq count_r \), by adding the right brackets if necessary. Using such control, we get a valid transmuted string \( mY \) from the normal string \( Y \).

3 **LGIC2: EMERGENT INDUCTION OF L-system GRAMMAR**

An emergent induction method LGIC2 (L-system GI with error Correction, ver.2) (Nakano, 2013b) is explained and slightly modified. Given a transmuted string \( mY \), LGIC2 generates grammar candidates, aiming at discovering the original grammar.

**Basic Framework**

The basic framework of LGIC2 is simple. Since a right side of each production rule appears repeatedly in \( mY \), we extract frequently appearing substrings from \( mY \) to form rule candidates. Then such a rule candidate is combined with its reasonable \( n \), the number of rewritings, to form a grammar candidate.

The main drawback of this approach is the combinatorial growth in the number of grammar candidates. Without any pruning it took ten days to finish two thirds of the processing for \( mY \) whose length is about 4,000. Thus, we need to narrow down candidates to get only promising ones. LGIC2 introduces the following three pruning techniques.

**Pruning by Frequency**

Since the right side of each original rule appears many times in \( mY \), we can discard less frequent substrings whose frequency is less than \( \text{min}_\text{freq} \). The threshold value may depend on the length of \( mY \). In our experiments we use \( \text{min}_\text{freq} = 50 \)

**The Number of Rewritings**

Now that we have a pair of rule candidates, we consider how to determine \( n \), the number of rewritings. The suitable \( n \) will depend on a transmutation type. For r-type or i-type we tried to find \( n \) generating the longest \( Z \) which satisfies \( \text{len}(Z) \leq \text{len}(mY) \) since \( \text{len}(Z) \leq \text{len}(mY) \). For d-type, however, the situation is different since \( \text{len}(mY) < \text{len}(Z) \). Thus, we should select \( n \) generating the shortest \( Z \) which satisfies \( \text{len}(mY) < \text{len}(Z) \).

**Pruning by Goodness of Fit**

The goodness of fit is a statistical measure which evaluates how well a model fits to observed data \( mY \). The goodness of fit can be evaluated by \( \chi^2 \) values. Let the numbers of symbol occurrences in \( mY \) and \( Z \) be \( \{ y_i \} \) and \( \{ z_i \} \) respectively. Then calculate \( \{ p_i = z_i / \text{len}(Z) \} \), and we have the following \( \chi^2 \) value. Here \( I \) is the number of all kinds of variables and constants.

\[
\chi^2 = \sum_i^I (y_i - \text{len}(mY) \times p_i)^2 / (\text{len}(mY) \times p_i) \tag{2}
\]

We discard a grammar candidate if \( \chi^2 \) is greater than \( \text{max}_{\text{chi}2} \). For r-type or i-type transmutation we used \( \text{max}_{\text{chi}2} = 150 \) since the value gets very large, more than 200 for high transmutation rate \( P_t \), even for the original grammar. For d-type, however, the fitting turned out to be extremely good. Thus, we can use a very small value \( \text{max}_{\text{chi}2} = 10 \) for d-type.

**Similarity between Two Strings**

As the similarity between two strings, LGIC2 employs the longest common subsequence (LCS) (Cormen, Leiserson and Rivest, 1990). Let \( \text{LCS}(S_1, S_2) \)
The first two strings are $S_1$ and $S_2$. Given two strings we may have more than one LCS, but the length of each LCS is the same. Note that LCS can cope with any type of transmutation. The length of LCS can be found using dynamic programming. Another measure is Levenshtein distance (Levenshtein, 1966) will result in much the same result.

**Pruning by Contractive Embedding**

In complex data, the cost of evaluating the distance between two objects is usually very high. Here we calculate the similarity measure LCS between two strings. Our experiments showed it takes about two seconds to calculate one LCS if each string length is around 4,000. Thus, the number of LCS calculations to generate a string $mY$ rest as a right side candidate of rule B. Here the length of $rsl_2$ should satisfy $2 \leq \text{len}(rsl_2) \leq \text{max}_z rsl$, and the number of $rsl_2$ occurrences in $mYrest$ should be more than or equal to $\min_f rsl$.

(step 3) For each pair of $rsl_1$ and $rsl_2$ selected above, find the number of rewritings $n$ to generate a string $Z$. Here $n$ is selected to be the smallest integer which satisfies $\text{len}(mY) \leq \text{len}(Z)$.

** Procedure of LGIC2 Method**

LGIC2 has the following four system parameters:

- $\text{max}_z rsl$: maximum length of rule right side
- $\min_f rsl$: minimum frequency of rule right side
- $\text{max}_2 h2$: maximum $\chi^2$ value
- tops: the number of final best solutions

LGIC2 goes as below:

(step 1) Extract a substring $rsl_1$ from $mY$ as a right side candidate of rule A. Here the length of $rsl_1$ should satisfy $2 \leq \text{len}(rsl_1) \leq \text{max}_z rsl$, and the number of $rsl_1$ occurrences in $mY$ should be more than or equal to $\min_f rsl$.

(step 2) Eliminate any occurrences of $rsl_1$ from $mY$ to get $mYrest$, and then extract a substring $rsl_2$ from $mYrest$ as a right side candidate of rule B. Here the length of $rsl_2$ should satisfy $2 \leq \text{len}(rsl_2) \leq \text{max}_z rsl$, and the number of $rsl_2$ occurrences in $mYrest$ should be more than or equal to $\min_f rsl$.

(step 3) For each pair of $rsl_1$ and $rsl_2$ selected above, find the number of rewritings $n$ to generate a string $Z$. Here $n$ is selected to be the smallest integer which satisfies $\text{len}(mY) \leq \text{len}(Z)$.

(step 4) Calculate $\chi^2$ value using $mY$ and $Z$, and then discard the grammar candidate as inappropriate if $\chi^2 > \text{max}_2 h2$.

(step 5) Calculate $\text{ubLCS}(mY, Z)$, the upper bound of LCS($mY, Z$), and then discard the grammar candidate if $\text{ubLCS}(mY, Z) < \text{LCS}(mY, Z)$. Here $Z_t$ is the string generated by the $t$-th best grammar candidate found so far, and note that $t = \text{tops}$.

(step 6) Calculate LCS($mY, Z$), and then keep the grammar candidate if the LCS is within the best $\text{tops}$ at that time. Go to step 1 if there is another candidate.

(step 7) Output $\text{tops}$ candidates having the largest LCSs as the final solutions.

**4 EXPERIMENTS**

LGIC2 was evaluated using a transmuted plant model. A plant model ex05n, a slight variation of bracketed OL-system example (Prusinkiewicz and Lindenmayer, 1990), was used as a normal model in our experiments. Figure 1 shows ex05n whose string length is 4,243. PC with Xeon(R), 2.66GHz, dual was used in our experiments.

\[
\begin{align*}
\text{ex05n} & \quad n = 6, \text{ axiom : } X \\
& \quad \text{rule : } X \rightarrow F[+X][-X]FX \\
& \quad \text{rule : } F \rightarrow FF
\end{align*}
\]

Figure 1: Normal plant model ex05n.

Here, we considered only d(eletion)-type transmutation with combinations of three coverage rates $P_e = 0.25, 0.5, 0.75$ and four occurrence rates $P_o = 0.25, 0.5, 0.75, 1.00$. For each combination we transmuted the normal model of ex05n five times changing a seed for random number generator.

Following the description given in the previous section, LGIC2 system parameters were set as follows: $\text{max}_z rsl = 15$, $\min_f rsl = 5$, $\text{max}_2 h2 = 10$, and $\text{tops} = 30$.

Table 1 shows the success rates of LGIC2 for d-
type transmutation. LGIC2 almost perfectly discovered the original grammar; however, for the case of $P_c = 0.75$ and $P_o = 1.00$, although the correct rules were found successfully, the number of rewritings $n$ was found to be 5 instead of 6 because the length of $mY$ was 1,064, too short to find the correct $n = 6$.

Table 2 shows the length of $mY$ for d-type transmutation. Note again the length of $Y$ is 4,243.

### Table 1: Success rates of LGIC2 for d-type transmutation.

<table>
<thead>
<tr>
<th>$P_c$</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
</tr>
<tr>
<td>0.50</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
</tr>
<tr>
<td>0.75</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
</tr>
</tbody>
</table>

### Table 2: Length of $mY$ transmuted in d-type.

<table>
<thead>
<tr>
<th>$P_c$</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>3,974</td>
<td>3,711</td>
<td>3,433</td>
<td>3,188</td>
</tr>
<tr>
<td>0.50</td>
<td>3,982</td>
<td>3,696</td>
<td>3,423</td>
<td>3,188</td>
</tr>
<tr>
<td>0.75</td>
<td>4,000</td>
<td>3,709</td>
<td>3,467</td>
<td>3,188</td>
</tr>
<tr>
<td>1.00</td>
<td>3,956</td>
<td>3,724</td>
<td>3,455</td>
<td>3,188</td>
</tr>
</tbody>
</table>

### Table 3: Length of LCS between $mY$ transmuted in d-type and normal $Y$.

<table>
<thead>
<tr>
<th>$P_c$</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>3,965</td>
<td>3,702</td>
<td>3,433</td>
<td>3,188</td>
</tr>
<tr>
<td>0.50</td>
<td>3,975</td>
<td>3,692</td>
<td>3,423</td>
<td>3,188</td>
</tr>
<tr>
<td>0.75</td>
<td>3,981</td>
<td>3,730</td>
<td>3,465</td>
<td>3,188</td>
</tr>
<tr>
<td>1.00</td>
<td>3,989</td>
<td>3,702</td>
<td>3,467</td>
<td>3,188</td>
</tr>
<tr>
<td>0.25</td>
<td>3,709</td>
<td>3,209</td>
<td>2,624</td>
<td>2,128</td>
</tr>
<tr>
<td>0.50</td>
<td>3,681</td>
<td>3,115</td>
<td>2,604</td>
<td>2,128</td>
</tr>
<tr>
<td>0.75</td>
<td>3,681</td>
<td>3,180</td>
<td>2,667</td>
<td>2,128</td>
</tr>
<tr>
<td>1.00</td>
<td>3,714</td>
<td>3,148</td>
<td>2,655</td>
<td>2,128</td>
</tr>
</tbody>
</table>

### Table 4: Ranking of the original grammar for d-type transmutation.

<table>
<thead>
<tr>
<th>$P_c$</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>No.3</td>
<td>No.1</td>
<td>No.1</td>
<td>No.1</td>
</tr>
<tr>
<td>0.50</td>
<td>No.3</td>
<td>No.1</td>
<td>No.1</td>
<td>No.1</td>
</tr>
<tr>
<td>0.75</td>
<td>No.3</td>
<td>No.1</td>
<td>No.1</td>
<td>No.1</td>
</tr>
<tr>
<td>1.00</td>
<td>No.1</td>
<td>No.1</td>
<td>No.1</td>
<td>No.1</td>
</tr>
</tbody>
</table>

Even from these transmuted plants, LGIC2 successfully discovered the original grammar.

Table 5 shows $\chi^2$ value calculated from normal $Y$ and transmuted $mY$. Note that $\chi^2$ values for r-type or i-type transmutation are very large for most cases, and exceed 100 for high transmutation rate $P_o$. However, $\chi^2$ value for d-type is quite small probably because occurrence rates of strings will not change drastically even if large deletion may happen. This nature can be used for pruning; thus, we adopted $\max_{x} \chi^2 > 0.10$.

Table 6 shows the average CPU time of LGIC2 for d-type transmutation. For each $P_c$, average CPU time gets shorter as $P_o$ gets larger, and for each $P_o$, average CPU time gets shorter as $P_c$ gets larger. These ten-
Figure 2: Plant model transmuted in d-type ($P_c = 0.50, P_o = 0.50$).

Figure 3: Plant model transmuted in d-type ($P_c = 0.25, P_o = 1.00$).

dencies can be understood if we consider larger transmutation rate $P_t ( = P_c \times P_o)$ makes transmuted string $mY$ shorter, which will reduce the number of different substrings extracted from $mY$.

Table 7 shows the average number of LCS calculations for d-type transmutation. For each $P_o$, average number of LCS calculations gets smaller as $P_c$ gets larger. We consider the above discussion may explain this tendency. Overall, the average numbers of LCS calculations are very small due to our pruning.

From our experiments described above, we can say that an emergent approach of LGIC2 together with pruning techniques works very well for d-type transmutation.

5 CONCLUSION

This paper examined how a noise-tolerant emergent induction LGIC2 works for d-type transmutation.

<table>
<thead>
<tr>
<th>$P_c$</th>
<th>$P_o$</th>
<th>$P_t$</th>
<th>$P_o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.50</td>
<td>0.75</td>
<td>1.00</td>
</tr>
<tr>
<td>0.887</td>
<td>0.974</td>
<td>1.059</td>
<td>0.652</td>
</tr>
<tr>
<td>0.210</td>
<td>0.468</td>
<td>0.219</td>
<td>0.652</td>
</tr>
<tr>
<td>0.130</td>
<td>0.248</td>
<td>0.126</td>
<td>0.652</td>
</tr>
<tr>
<td>0.170</td>
<td>0.252</td>
<td>0.664</td>
<td>0.652</td>
</tr>
</tbody>
</table>

Table 6: Average CPU time (sec) of LGIC2 for d-type transmutation.

<table>
<thead>
<tr>
<th>$P_c$</th>
<th>$P_o$</th>
<th>$P_t$</th>
<th>$P_o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.50</td>
<td>0.75</td>
<td>1.00</td>
</tr>
<tr>
<td>923.8</td>
<td>791.1</td>
<td>546.0</td>
<td>283.2</td>
</tr>
<tr>
<td>551.5</td>
<td>431.0</td>
<td>246.2</td>
<td>61.3</td>
</tr>
<tr>
<td>433.2</td>
<td>287.4</td>
<td>143.6</td>
<td>18.0</td>
</tr>
</tbody>
</table>

LGIC2 was slightly modified to fit d-type transmutation. Our experiments using a simple plant model showed LGIC2 discovered the original grammar as a top candidate for almost all cases in minutes for high transmutation and in about 15 minutes at the lowest transmutation. In the future we plan to apply the method to other plant models.

ACKNOWLEDGEMENTS

This work was supported by Grants-in-Aid for Scientific Research (C) 25330294 and Chubu University Grant 26IS19A.
REFERENCES


