Error Correction of Enumerative Induction of Deterministic Context-free L-system Grammar

Ryohei Nakano

***Abstract*—This paper addresses error correction capability of grammatical induction method called LGIC for deterministic context-free L-systems. LGIC induces L-system grammars from a transmuted string mY. In the method, sets of parameter values are enumerated and those located within the tolerable distance from mY are used to form candidates of production rules. Each candidate of production rules is used to generate a candidate string Z, and the similarity between Z and mY is calculated and stored. Finally, several candidates having the strongest similarities are selected as the final solutions. Our experiments using strings having replacement- or insertion-type transmutation showed that LGIC error correction strongly depends on some system parameter and works better for replacement-type than for insertion-type.**

***Index Terms*—grammatical induction, L-system, error correction, plant model, transmutation**

I. Introduction

L-systems were originally developed by Lindenmayer as a mathematical theory of plant development [1]. The central concept is rewriting, which is a powerful mechanism for generating complex objects from a simple initial object using production rules.

As for rewriting systems, Chomsky's work on formal grammars is well known. L-systems and formal grammars are both string rewriting systems, but there is the essential difference; that is, productions are applied in parallel in L-systems, while applied sequentially in formal grammars.

The reverse process of rewriting is called grammatical induction, which discovers a set of production rules given a set of strings. Although grammatical induction of formal grammars has been studied for decades, the induction of context-free grammars is still an open problem. The induction of L-system grammars is also an open problem little explored so far.

L-systems can be broadly divided using two aspects: (1) deterministic or stochastic, and (2) context-free or context-sensitive. McCormack [2] addressed computer graphics modeling through evolution of deterministic/stochastic context-free L-systems. Nevill-Manning [3] proposed a simple algorithm called Sequitur, which uncovers structures like context-free grammars from various sequences; however, it worked with limited success for grammatical induction of L-system grammar. Schlecht, et al. [4] proposed statistical structural inference for microscopic 3D images through learning stochastic L-system model. Damasevicius [5] addressed structural analysis of DNA sequences through evolution of stochastic context-free L-system grammars.

This work was supported in part by Grants-in-Aid for Scientific Research (C) 22500212 and Chubu University Grant 24IS27A.

R. Nakano is with the Department of Computer Science, Graduate School of Engineering, Chubu University, 1200 Matsumotocho, Kasugai 487-8501, Japan. email: nakano@cs.chubu.ac.jp

Nakano and Yamada [6] proposed an efficient induction method for deterministic context-free L-system, employing a number theory-based approach. The method assumes a given string includes no errors, making it possible to employ the number theory. In the real world, however, any object may have some noise, errors, or transmutation. As for transmutation, we can consider several types such as replacement-type, insertion-type, deletion-type, or mixed-type.

Nakano and Suzumura [7] proposed an enumerative induction method called LGIC for deterministic context-free L-systems. The method has error correction capability, which means it discovers L-system grammars from a transmuted string mY. LGIC works as follows. First, sets of parameter values are enumerated and those within the tolerable distance from mY are used to form candidates of production rules. Each candidate of production rules is used to generate a candidate string Z, and the similarity between Z and mY is calculated, and finally several best candidates are selected as the final solutions.

This paper evaluates the error correction capability of LGIC using strings having different amount of replacement- or insertion-type transmutation. Our experiments showed LGIC error correction strongly depends on some system parameter and works better for replacement-type than for insertion-type.

II. Background

D0L-system. The simplest class of L-systems are deterministic context-free called D0L-systems. D0L-system is defined as $G = (V, C, \omega, P)$, where V and C denote sets of *variables* and *constants*, ω is an initial string called *axiom*, and P is a set of *production rules*. A variable is a symbol that is replaced in rewriting, and a constant is a symbol that remains unchanged in rewriting and is used to control turtle graphics.

Notation. Shown below is the notation employed in this paper. Here we consider the following two production rules.

rule A : $A \rightarrow$????????

rule B : $B \rightarrow$??????

mY: given transmuted string.
n: the number of rewritings.
$Z^{(n)}$: string obtained after n times rewritings.
$\alpha_A,\ \alpha_B,\ \alpha_K$: the numbers of variables A, B and constant K occurring in the right side of rule A.
$\beta_A,\ \beta_B,\ \beta_K$: the numbers of variables A, B and constant K occurring in the right side of rule B.
$y_A,\ y_B,\ y_K$: the numbers of variables A, B and constant K occurring in mY.
$z_A^{(n)},\ z_B^{(n)},\ z_K^{(n)}$: the numbers of variables A, B and constant K occurring in $Z^{(n)}$.

Transmutation. As for string transmutation, there can be several types: replacement-type (r-type), insertion-type (i-type), deletion-type (d-type), or mixed-type (m-type). In this paper, r-type and i-type transmutation are considered. In r-type transmutation designated symbols of a string is replaced with symbols selected randomly from the set of symbols, while in i-type transmutation symbols randomly selected are inserted at designated positions of a string.

As for how transmutation occurs, we consider two rates: *coverage rate* P_c and *occurrence rate* P_o. We assume transmutation occurs only locally around the center of an original string Y. The coverage rate P_c represents the proportion of transmutation area to the whole Y. For example, $P_c = 0.25$ means 25 % area around the center of Y is to be transmuted. The occurrence rate P_o represents the probability of transmutation in the transmutation area. Thus, overall *transmutation rate* P_t can be computed as follows:

$$P_t \quad = \quad P_c \times P_o \tag{1}$$

Valid Transmutation. Simple transmutation will generate an invalid string, which means the string cannot be drawn through turtle graphics. To keep the transmutation valid, the numbers of left and right square brackets are to be monitored throughout transmutation and controlled if necessary. Specifically, in the transmutation area the number $count_\ell$ of left square brackets should be larger than or equal to the number $count_r$ of right ones. Moreover, when the transmutation ends, we should assure $count_\ell$ = $count_r$ by adding the right square brackets if necessary. By applying such controlled transmutation, we get a valid transmuted string mY from the original string Y.

III. L-system Grammar Induction with Error Correction

The enumerative induction method called LGIC (L-system Grammar Induction with error Correction) [7] is explained. Here we consider the following D0L-system. Given a transmuted string mY, LGIC generates candidates of a set of rewritings n and rules A and B.

$$\begin{aligned} & n = ?, \quad \text{axiom} : A \\ & \text{rule A} : A \rightarrow ???????? \\ & \text{rule B} : B \rightarrow ????? \end{aligned}$$

Growth Equation. The above D0L-system has the following growth of the numbers of occurrences of A and B.

$$(1 \quad 0)\ \mathbf{T}^n = (z_A^{(n)} \quad z_B^{(n)}), \quad \mathbf{T} = \begin{pmatrix} \alpha_A & \alpha_B \\ \beta_A & \beta_B \end{pmatrix} \tag{2}$$

Enumeration of Variable Parameters and the Number of Rewritings. Since mY is transmuted, we cannot rely on a number theory-based approach [6] any more. Here we employ an enumerative approach. To find a reasonable set of $(n, \alpha_A, \alpha_B, \beta_A, \beta_B)$, we examine all the combinations of value ranges $[0,\ max\_var]$ of five integer parameters, where $z_A^{(n)}$ and $z_B^{(n)}$ are easily calculated using eq.(2). Then, the following difference from mY is calculated to evaluate how good the set is.

$$diff \quad = \quad |z_A^{(n)} - y_A| + |z_B^{(n)} - y_B| \tag{3}$$

If $diff$ is smaller than or equal to the tolerable distance, the set $(n, \alpha_A, \alpha_B, \beta_A, \beta_B)$ is selected for the later processing.

Selection of Constant Parameters. For each constant K we repeat the following independently. Using the following equations we calculate $r_A^{(n)}$ and $r_B^{(n)}$, the numbers of A and B rewritings occurred until n rewritings.

$$r_A^{(n)} \quad = \quad 1 + z_A^{(1)} + z_A^{(2)} + \cdots + z_A^{(n-1)} \tag{4}$$

$$r_B^{(n)} \quad = \quad z_B^{(1)} + z_B^{(2)} + \cdots + z_B^{(n-1)} \tag{5}$$

Then we have the following equation whose coefficients and solution are non-negative integers.

$$z_K^{(n)} \quad = \quad r_A^{(n)}\ \alpha_K + r_B^{(n)}\ \beta_K \tag{6}$$

This can be used to narrow down the upper bounds of constant parameters α_K and β_K.

$$\alpha_K \quad \leq \quad \lceil z_K^{(n)} / r_A^{(n)} \rceil + 1 \equiv upper_A \tag{7}$$

$$\beta_K \quad \leq \quad \lceil z_K^{(n)} / r_B^{(n)} \rceil + 1 \equiv upper_B \tag{8}$$

For each set $(n, \alpha_A, \alpha_B, \beta_A, \beta_B)$, we find the best pair of (α_K, β_K) which minimizes $|z_K^{(n)} - y_K|$ from all the integer combinations of $[0,\ upper_A] \times [0,\ upper_B]$.

Generate-and-test of Rule Candidates. Now we have candidates of $(n, \alpha_A, \alpha_B, \beta_A, \beta_B, \alpha_K, \beta_K)$. Since in given string mY the right sides of rules A and B repeatedly appear as substrings, we exhaustively extract from mY the following two substrings:

(a) a substring having α_A A's, α_B B's, and α_K K's to form a rule A candidate,
(b) a substring having β_A A's, β_B B's, and β_K K's to form a rule B candidate.

For each combination of rules A and B candidates, we rewrite n times beginning with the axiom to generate a string $Z^{(n)}$. Then, we calculate the similarity between $Z^{(n)}$ and mY. Finally, several pairs of candidates having the strongest similarities are selected as solutions.

Similarity between Two Strings. As stated above, we need the measure to evaluate the similarity between two strings. LGIC employs the longest common subsequence (LCS) [8]. For example, an LCS of ABCDABC and BDCAB is BDAB or BCAB. Given two strings we may have more than one LCSs, but, of course, the length of each LCS is the same. As the similarity between two strings S_1 and S_2, we use the length of LCS of S_1 and S_2. Note that LCS can cope with any type of transmutation.

LCSs and its length can be found using dynamic programming [8]. The code size is very small, but the processing time will be long if two string lengths get large. Here we only need the length of an LCS, which makes the memory size much smaller accelerating the processing speed.

Another reasonable measure is Levenshtein distance [9], which is defined as the minimum number of modifications required to transform one string into the other. We consider these two measures will result in much the same result.

Procedure of LGIC Method. LGIC goes as below. The tolerable distance tol_diff and the number of final solutions $tops$ are system parameters.

(step 1) Select parameter candidates.
(step 1.1) Count occurrences in mY to get y_A, y_B, and y_K.
(step 1.2)] Select a set of parameters $(n, \alpha_A, \alpha_B, \beta_A, \beta_B)$ whose $diff$ is smaller than or equal to tol_diff.
(step 1.3) For each $(n, \alpha_A, \alpha_B, \beta_A, \beta_B)$ selected above, select the best pair (α_K, β_K).

(step 2) For each $(n, \alpha_A, \alpha_B, \beta_A, \beta_B, \alpha_K, \beta_K)$ generate rule candidates.
(step 2.1) From mY get a substring having α_A A's, α_B B's, and α_K K's to form a rule A candidates.
(step 2.2) From mY get a substring having β_A A's, β_B B's, and β_K K's to form a rule B candidates.
(step 2.3) For each combination of rule A candidates and rule B candidates, generate a string $Z^{(n)}$, and calculate the similarity between $Z^{(n)}$ and mY.

(step 3) Among the candidates, select $tops$ candidates having the strongest similarities as the final solutions.

IV. Experiments

Error correction capability of LGIC was evaluated using a plant model. A plant model ex05n is a slight variation of ex05p shown in [1]. Figure 1 shows ex05n, which was used in our experiments. The string length of ex05n is 4,243. PC with Xeon(R), 2.66GHz, dual was used.

(ex05p) $n = 7$, axiom : X
rule : $X \rightarrow F[+X][-X]FX$
rule : $F \rightarrow FF$
(ex05n) $n = 6$, axiom : X
rule : $X \rightarrow F[+X][-X]FX$
rule : $F \rightarrow FF$

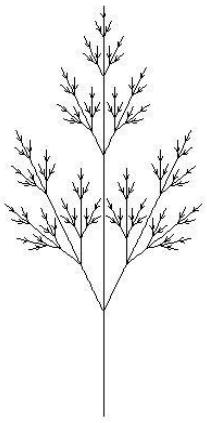

Fig. 1. Normal plant model ex05n

As for transmutation, we considered replacement-type (r-type) and insertion-type (i-type). For each transmutation, we considered combinations of three coverage rates P_c = 0.25, 0.5, 0.75 and four occurrence rates P_o = 0.25, 0.5, 0.75, 1.0. For each combination we transmuted ex05n five times changing a seed for random number generator.

As for LGIC system parameters, the tolerable distance tol_diff is set to be 100 at first and then changed to 150 for r-type, and to 200 for i-type transmutation; this is because LGIC with 150 improved its performance only slightly for i-type transmutation. The number of final solutions $tops$ is set to be 10. Moreover, the maximum of variable or constant occurrences in a rule max_var is set to be 10.

A. Error Correction for r-type Transmutation

First, we show the results of LGIC error correction for r-type transmutation. Table I shows the success rates of LGIC error correction with tol_diff = 100 for r-type transmutation. When $P_t = P_c \times P_o \leq$ 1/8 (= 0.125), LGIC with tol_diff = 100 discovered the true grammar ex05n with probability 1 (= 5 out of 5 runs). However, when P_t exceeds 3/16 (= 0.188), the success rate rapidly dropped to around zero. The table also shows once P_t exceeds the boundary, LGIC cannot discover the true grammar for any higher P_t. Moreover, when the true grammar was found, it was always rated No.1 showing the strongest similarity. Thus, the number of final solutions $tops$ = 10 is reasonable.

TABLE I
Success rates of LGIC error correction for r-type transmutation (tol_diff=100)

	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
$P_c = 0.25$	5/5	5/5	1/5	0/5
$P_c = 0.50$	5/5	0/5	0/5	0/5
$P_c = 0.75$	0/5	0/5	0/5	0/5

Figure 2 shows a plant model transmuted in r-type with P_c = 0.25 and P_o = 0.50. LGIC with tol_diff = 100 successfully discovered the true grammar from this transmuted plant. Figure 3 shows a plant model transmuted in r-type with P_c = 0.50 and P_o = 0.25. Even from this transmuted plant, LGIC with tol_diff = 100 discovered the true grammar.

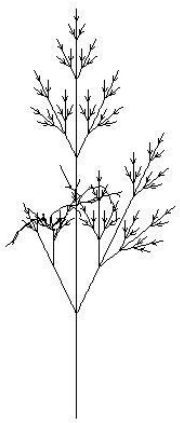

Fig. 2. Plant model transmuted in r-type (P_c = 0.25, P_o = 0.50)

Table II shows CPU time spent and the number of similarity calculations needed by LGIC with tol_diff = 100 for r-type transmutation. In the experiments we observed most CPU time was spent for similarity calculation. There is a tendency that CPU time is proportional to the number of similarity calculations.

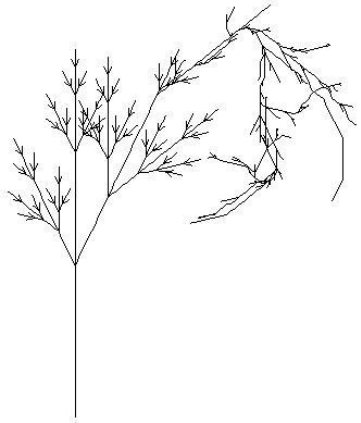

Fig. 3. Plant model transmuted in r-type (P_c = 0.50, P_o = 0.25)

Table III shows the average statistics of Table II. For each P_o, average CPU time spent by LGIC gets longer as P_c gets larger; however, for each P_c, average CPU time does not necessarily increase even if P_o gets larger. This may indicate that the number of rule candidates gets larger when P_c gets larger for a fixed P_o, but doesn't always get larger even if P_o gets larger for a fixed P_c.

TABLE II
CPU TIME (SEC) OF LGIC FOR R-TYPE TRANSMUTATION TOGETHER WITH THE NUMBER OF SIMILARITY CALCULATIONS IN PARENTHESES (tol_diff=100)

P_c	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
0.25	93.8 (23)	329.4 (163)	98.0 (26)	127.2 (45)
	158.8 (59)	133.7 (50)	95.7 (22)	161.1 (61)
	187.4 (71)	247.8 (113)	199.5 (83)	107.3 (33)
	158.3 (62)	426.6 (215)	107.7 (31)	217.5 (100)
	205.7 (92)	350.1 (173)	439.7 (223)	286.8 (140)
0.50	619.4 (301)	501.1 (244)	684.4 (338)	186.8 (33)
	307.9 (130)	545.8 (273)	734.2 (357)	205.4 (46)
	594.9 (297)	370.0 (167)	390.7 (157)	315.1 (103)
	1898.3 (1044)	239.9 (84)	375.3 (142)	251.0 (64)
	711.0 (357)	562.3 (287)	845.1 (436)	388.7 (157)
0.75	1556.8 (855)	2049.7 (1108)	856.7 (394)	700.4 (278)
	1232.2 (664)	1606.7 (866)	694.4 (282)	921.1 (434)
	422.3 (168)	2567.6 (1401)	1102.6 (530)	553.4 (194)
	423.6 (177)	2236.8 (1194)	773.3 (326)	874.7 (354)
	1878.1 (1029)	1206.9 (639)	560.9 (206)	874.9 (370)

TABLE III
AVERAGE CPU TIME (SEC) OF LGIC FOR R-TYPE TRANSMUTATION TOGETHER WITH THE AVERAGE NUMBER OF SIMILARITY CALCULATIONS IN PARENTHESES (tol_diff=100)

P_c	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
0.25	160.8 (61)	297.5 (143)	188.1 (77)	180.0 (76)
0.50	826.3 (426)	443.8 (211)	605.9 (286)	269.4 (81)
0.75	1102.6 (579)	1933.5 (1042)	797.6 (348)	784.9 (326)

Now we changed tol_diff from 100 to 150, which will increase LGIC error correction capability since the search range is enlarged. Table IV shows the success rates of LGIC with tol_diff = 150 for r-type transmutation. When P_t = $P_c \times P_o$ = 3/16 (= 0.1875), LGIC with tol_diff = 150 discovered the true grammar with probability 0.8 or 1. However, when P_t = 0.25, the success rate dropped to 20 % for $P_c = P_o = 0.5$. Three combinations shown in parentheses in Table IV were skipped since LGIC will fail for such P_t.

TABLE IV
SUCCESS RATES OF LGIC ERROR CORRECTION FOR R-TYPE TRANSMUTATION (tol_diff=150)

	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
$P_c = 0.25$	5/5	5/5	5/5	4/5
$P_c = 0.50$	5/5	1/5	0/5	(0/5)
$P_c = 0.75$	4/5	0/5	(0/5)	(0/5)

Figure 4 shows a plant model transmuted in r-type with P_c = P_o = 0.50. LGIC with tol_diff = 150 successfully discovered the true grammar from this transmuted plant. Figure 5 shows a plant model transmuted in r-type with P_c = 0.75 and P_o = 0.25. Even from this transmuted plant, LGIC with tol_diff = 150 discovered the true grammar.

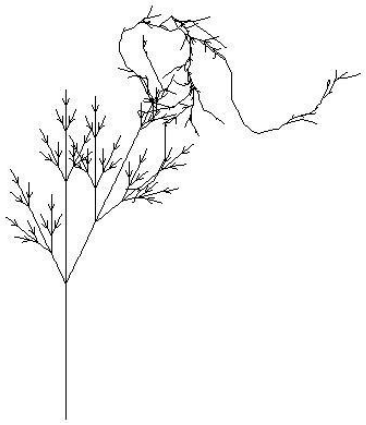

Fig. 4. Plant model transmuted in r-type (P_c = 0.50, P_o = 0.50)

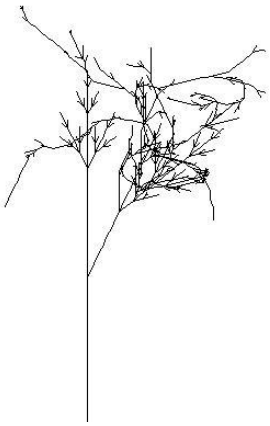

Fig. 5. Plant model transmuted in r-type (P_c = 0.75, P_o = 0.25)

Table V shows CPU time spent and the number of similarity calculations needed by LGIC with tol_diff = 150 for r-type transmutation. When $P_t \leq$ 1/8, it is obvious LGIC with tol_diff = 150 will succeed; thus, skipped. On the other hand, when $P_t \geq$ 1/2, it is also obvious LGIC with tol_diff = 150 will fail; thus, three combinations were skipped. Again, there is a tendency that CPU time is proportional to the number of similarity calculations.

Table VI shows the average of Table V. Again, for each P_o, average CPU time gets longer as P_c gets larger; however, for each P_c, it does not necessarily increase even if P_o gets

larger. Moreover, when tol_diff gets larger, the number of rule candidates gets larger, and therefore, CPU time gets longer. When tol_diff was changed from 100 to 150, CPU time increased more than double in most cases.

TABLE V
CPU TIME (SEC) OF LGIC FOR R-TYPE TRANSMUTATION TOGETHER WITH THE NUMBER OF SIMILARITY CALCULATIONS IN PARENTHESES (tol_diff=150)

P_c	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
0.25	n/a	n/a	633.9 (315) 329.1 (125) 692.9 (338) 250.3 (80) n/a	397.7 (166) 672.4 (328) 299.5 (104) 580.7 (273) 488.0 (222)
0.50	n/a	1922.8 (1008) 1932.4 (1026) 1638.3 (841) 1621.9 (811) 1716.6 (888)	1449.9 (710) 992.2 (438) 2249.3 (1157) 997.3 (423) 1720.8 (889)	n/a
0.75	3786.1 (2095) 3599.8 (1959) 2646.8 (1418) 3191.4 (1724) 3904.4 (2144)	2767.7 (1424) 2834.8 (1495) 3255.8 (1686) 4325.3 (2350) 2067.1 (1025)	n/a	n/a

TABLE VI
AVERAGE CPU TIME (SEC) OF LGIC FOR R-TYPE TRANSMUTATION TOGETHER WITH THE AVERAGE NUMBER OF SIMILARITY CALCULATIONS IN PARENTHESES (tol_diff=150)

P_c	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
0.25	n/a	n/a	476.6 (215)	487.7 (219)
0.50	n/a	1766.4 (915)	1481.9 (723)	n/a
0.75	3425.7 (1868)	3050.1 (1596)	n/a	n/a

B. Error Correction for i-type Transmutation

Next, we show the results for i-type transmutation. Table VII shows the success rates of LGIC with tol_diff = 100 for i-type transmutation. The success range is quite limited; Only when $P_t = P_c \times P_o$ = 1/16 (= 0.0625), LGIC with tol_diff = 100 discovered the true grammar with probability 0.8. For the other combinations, however, LGIC with tol_diff = 100 could not discover the true grammar at all. Compared with the results for r-type transmutation, LGIC with tol_diff = 100 could not work well for i-type transmutation.

TABLE VII
SUCCESS RATES OF LGIC ERROR CORRECTION FOR I-TYPE TRANSMUTATION (tol_diff=100)

	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
$P_c = 0.25$	4/5	0/5	(0/5)	(0/5)
$P_c = 0.50$	0/5	(0/5)	(0/5)	(0/5)
$P_c = 0.75$	(0/5)	(0/5)	(0/5)	(0/5)

Figure 6 shows a plant model transmuted in i-type with P_c = P_o = 0.25. LGIC with tol_diff = 100 successfully discovered the true grammar from this transmuted plant.

Table VIII shows CPU time and the number of similarity calculations needed by LGIC with tol_diff = 100 for i-type transmutation. Again, there is a tendency that CPU time is proportional to the number of similarity calculations.

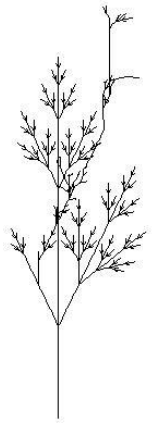

Fig. 6. Plant model transmuted in i-type (P_c = 0.25, P_o = 0.25)

Table IX shows the average of Table VIII. For P_o= 0.25, average CPU time gets longer as P_c gets larger; however, for P_c= 0.25, it does not increase even if P_o gets larger.

TABLE VIII
CPU TIME (SEC) OF LGIC FOR I-TYPE TRANSMUTATION TOGETHER WITH THE NUMBER OF SIMILARITY CALCULATIONS IN PARENTHESES (tol_diff=100)

P_c	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
0.25	552.3 (271) 474.4 (226) 499.8 (240) 257.7 (112) 171.7 (63)	139.8 (43) 813.6 (367) 130.5 (34) 285.2 (102) 144.1 (45)	n/a	n/a
0.50	268.3 (92) 600.5 (253) 254.0 (84) 307.9 (114) 989.9 (438)	n/a	n/a	n/a
0.75	n/a	n/a	n/a	n/a

TABLE IX
AVERAGE CPU TIME (SEC) OF LGIC FOR I-TYPE TRANSMUTATION TOGETHER WITH THE AVERAGE NUMBER OF SIMILARITY CALCULATIONS IN PARENTHESES (tol_diff=100)

P_c	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
0.25	391.2 (182)	302.6 (118)	n/a	n/a
0.50	484.1 (196)	n/a	n/a	n/a
0.75	n/a	n/a	n/a	n/a

Now we changed tol_diff from 100 to 200, which will increase error correction capability of LGIC. Table X shows the success rates of LGIC with tol_diff = 200 for i-type transmutation. When $P_t = P_c \times P_o$= 1/16 (= 0.0625), LGIC with tol_diff = 200 discovered the true grammar with probability 1. When P_t = 1/8 (= 0.125), however, the success rate dropped to 80 %. Further, in most combinations where $P_t \geq$ 3/16 (= 0.1875), LGIC with tol_diff = 200 could not discover the true grammar. Compared with the results with tol_diff = 150 for r-type transmutation, we see LGIC could not work well for i-type even with tol_diff = 200. This is because insertion-type transmutation drastically change occurrence frequencies of symbols, which prevents

LGIC from selecting the right set of parameters. Deletion-type transmutation will work similarly.

TABLE X
SUCCESS RATES OF LGIC ERROR CORRECTION FOR I-TYPE TRANSMUTATION (tol_diff=200)

	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
$P_c = 0.25$	5/5	4/5	0/5	(0/5)
$P_c = 0.50$	4/5	0/5	(0/5)	(0/5)
$P_c = 0.75$	0/5	(0/5)	(0/5)	(0/5)

Figure 7 shows a plant model transmuted in i-type with P_c = 0.25 and P_o = 0.50. LGIC with tol_diff = 200 successfully discovered the true grammar from this transmuted plant. Figure 8 shows a plant model transmuted in i-type with P_c = 0.50 and P_o = 0.25. Even from this transmuted plant, LGIC with tol_diff = 200 discovered the true grammar.

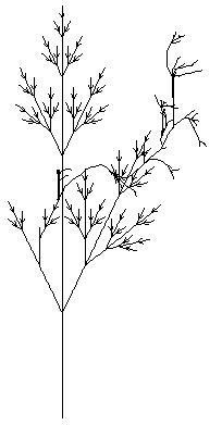

Fig. 7. Plant model transmuted in i-type (P_c = 0.25, P_o = 0.50)

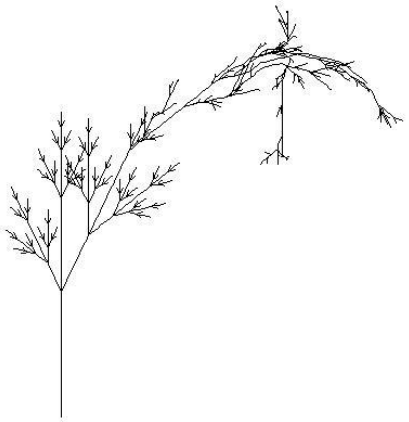

Fig. 8. Plant model transmuted in i-type (P_c = 0.50, P_o = 0.25)

Table XI shows CPU time and the number of similarity calculations needed by LGIC with tol_diff = 200 for i-type transmutation. Again, there is a tendency that CPU time is proportional to the number of similarity calculations.

Table XII shows the average of Table XI. Again, for each P_o, average CPU time gets longer as P_c gets larger; however, for each P_c, average CPU time does not necessarily increase even if P_o gets larger. Moreover, when tol_diff gets larger, the number of rule candidates gets larger, and therefore, CPU time gets longer. When tol_diff was changed from 100 to 200, CPU time increased around three times.

TABLE XI
CPU TIME (SEC) OF LGIC FOR I-TYPE TRANSMUTATION TOGETHER WITH THE NUMBER OF SIMILARITY CALCULATIONS IN PARENTHESES (tol_diff=200)

P_c	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
0.25	n/a n/a n/a n/a 1050.9 (466)	643.3 (208) 1337.3 (534) 803.8 (286) 970.6 (355) 594.6 (187)	647.9 (180) 675.7 (189) 759.4 (225) 1125.7 (377) 780.8 (243)	n/a
0.50	1125.4 (390) 2092.8 (846) 978.3 (312) 1329.2 (502) 3506.3 (1544)	1911.5 (602) 3087.8 (1048) 1661.1 (499) 1978.5 (606) 1804.2 (577)	n/a	n/a
0.75	3751.6 (1436) 4478.9 (1791) 2302.6 (818) 3441.2 (1308) 3137.7 (1193)	n/a	n/a	n/a

TABLE XII
AVERAGE CPU TIME (SEC) OF LGIC FOR I-TYPE TRANSMUTATION TOGETHER WITH THE AVERAGE NUMBER OF SIMILARITY CALCULATIONS IN PARENTHESES (tol_diff=200)

P_c	$P_o = 0.25$	$P_o = 0.50$	$P_o = 0.75$	$P_o = 1.0$
0.25	1050.9 (466)	869.9 (314)	797.9 (243)	n/a
0.50	1806.4 (719)	2088.6 (666)	n.a	n/a
0.75	3422.4 (1309)	n/a	n/a	n/a

V. CONCLUSION

This paper evaluated error correction capability of grammatical induction called LGIC for deterministic context-free L-systems. Given a transmuted string, LGIC induces L-system grammars, employing an enumerative approach to find suitable parameters. Our experiments using strings with replacement- or insertion-type transmutation showed that LGIC error correction strongly depends on some system parameter and works much better for replacement-type than for insertion-type. In the future we plan to invent another induction method to overcome these drawbacks.

REFERENCES

[1] P. Prusinkiewicz and A. Lindenmayer, *The algorithmic beauty of plants*. New York: Springer-Verlag, 1990.

[2] J. McCormack, "Interactive evolution of L-system grammars for computer graphics modelling," in *Complex Systems: From Biology to Computation*. ISO Press, Amsterdam, 1993, pp. 118–130.

[3] C. Nevill-Manning, "Inferring sequential structure," Univ of Waikato, Tech. Rep. Doctoral Thesis, 1996.

[4] J. Schlecht, K. Barnard, E. Springgs, and B. Pryor, "Inferring grammar-based structure models from 3d microscopy data," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[5] R. Damasevicius, "Structural analysis of regulatory DNA sequences using grammar inference and support vector machine," *Neurocomputing*, vol. 73, pp. 633–638, 2010.

[6] R. Nakano and N. Yamada, "Number theory-based induction of deterministic context-free L-system grammar," in *Proc. Int. Joint Conf. on Knowledge Discovery, Knowledge Engineering and Knowledge Management 2010*, pp. 194–199.

[7] R. Nakano and S. Suzumura, "Grammatical induction with error correction for deterministic context-free L-systems," in *Proc. of the World Congress on Engineering and Computer Science 2012, WCECS 2012, 24-26 Oct, 2012, San Francisco, USA*, pp. 534–538.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*. MIT Press, 1990.

[9] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.