Grammatical Induction with Error Correction for Deterministic Context-free L-systems

Ryohei Nakano and Shinya Suzumura

***Abstract*—This paper addresses grammatical induction with error correction for deterministic context-free L(D0L)-system; that is, proposed is a method of L-system grammar induction from a transmuted string mY. In our method, a set of parameter values is exhaustively searched and if it is located within the tolerable distance from a point determined by a given string mY, then the parameters are used to form candidates of production rules. Candidates of production rules are 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 shown as the final solutions. Our experiments using strings having various r-type transmutation rates showed the proposed method discovered the true grammar when the transmutation rate is less than around 20%.**

***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 of L-systems is rewriting. In general, rewriting 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 the essential difference is that productions are applied in parallel in L-systems, while productions are 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, 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 and S. Suzumura are with the Department of Computer Science, Graduate School of Engineering, Chubu University, 1200 Matsumotocho, Kasugai 487-8501, Japan. email: nakano@cs.chubu.ac.jp and tp11018-4021@sti.chubu.ac.jp

Nakano and Yamada [6] proposed a very efficient method of L-system grammar induction employing a number theory-based approach. The method assumes a given string does not include any errors, which makes it possible to employ the number theory. In the real world, however, any object may have some noise, errors, or transmutations. In string transmutation, we can consider several types such as replacement-type, deletion-type, insertion-type, or mixed-type.

Here we address grammatical induction with error correction for deterministic context-free L(D0L)-system. That is, the paper proposes a method of L-system grammar induction from a transmuted string mY. In our method, a set of parameter values is exhaustively searched and if it is within the tolerable distance from a point determined by a given mY, then the parameters are used to form candidates of production rules. Candidates of production rules are used to generate a candidate string Z, and the similarity between Z and mY is calculated and stored. Finally, several best candidates are shown as the output. Our experiments using replacement-type transmutation revealed how the success rate of our method is influenced by the transmutation rate.

II. Background

D0L-system. The simplest class of L-systems are called D0L-system (deterministic context-free L-system). 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 $Z^{(n)}$.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Transmutation. As for string transmutation, there can be several types: replacement-type (r-type), deletion-type (d-type), insertion-type (i-type), or mixed-type (m-type). Since what each type means is obvious, we skip the explanation. As our first step, only r-type transmutation is considered. In r-type transmutation, a designated symbol is replaced with a symbol selected randomly from the set of symbols.

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 represented 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 parentheses are to be monitored throughout transmutation and controlled if necessary. That is, in the transmutation area the number $count_\ell$ of left parentheses 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 parentheses 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 method proposed below is called *LGIC (L-system Grammar Induction with error Correction)*. Although we consider the following D0L-system having two production rules, the method can be easily extended to a different number of rules.

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

Given a transmuted string mY, we are asked to estimate the number of rewritings n and to induce the rules A and B.

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

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

Exhaustive Search of Variable Parameters and the Number of Rewritings. Since the given string mY is transmuted, we cannot rely on a number theory-based approach as taken in [6]. Here, we employ exhaustive search to find a reasonable set of $(n, \alpha_A, \alpha_B, \beta_A, \beta_B)$ because $z_A^{(n)}$ and $z_B^{(n)}$ are easily calculated using eq.(2). The exhaustive search examines all the combinations of value ranges $[0,\ max\_var]$ of five parameters. Then, the following difference is calculated to evaluate how reasonable the set is.

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

If the above $diff$ 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 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 the axiom n times using the candidates 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. To consider such a measure, we employ the longest common subsequence (LCS) [7]. 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 [7]. 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 [8], 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.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Procedure of LGIC Method. The procedure of LGIC method is shown below. The tolerable distance tol_diff and the number of final solutions $tops$ are given to the procedure as 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

The proposed LGIC method was evaluated using a plant model. Plant model ex05p is shown in [1] and ex05n is its variation with a smaller n. 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 : $F \rightarrow FF$

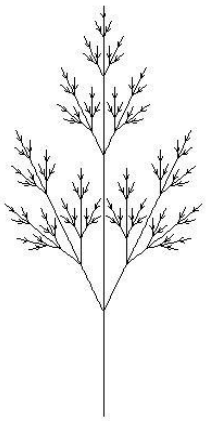

Fig. 1. Normal plant model ex05n

As for transmutation, we examined all the 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; 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.

Table I shows the success rates of LGIC error correction for tol_diff = 100. When $P_t = P_c \times P_o \leq$ 1/8 (= 0.125), LGIC with tol_diff = 100 discovered the true grammar ex05n with 100 % (= 5 out of 5 runs). However, when P_t exceeds 3/16 (= 0.188), the success rate rapidly dropped to around zero. 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 large enough.

TABLE I
Success rates of LGIC error correction for R-type transmutation (original plant model ex05n, 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 one of plant models transmuted with P_c = 0.25 and P_o = 0.50. LGIC with tol_diff = 100 successfully discovered the true grammar for this transmuted plant.

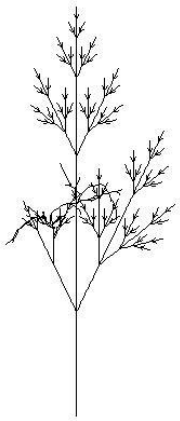

Fig. 2. Plant model transmuted from ex05n (P_c = 0.25, P_o = 0.50)

Figure 3 shows one of plant models transmuted with P_c = 0.50 and P_o = 0.25. Even for this transmuted plant, LGIC with tol_diff = 100 discovered the true grammar.

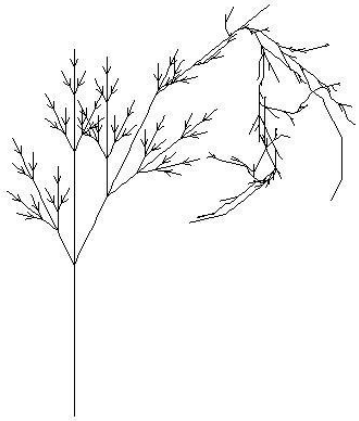

Fig. 3. Plant model transmuted from ex05n (P_c = 0.50, P_o = 0.25)

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Table II shows CPU time spent and the number of similarity calculations needed by LGIC with tol_diff = 100 for each combination of P_c, P_o, and random number seed. In our experiments we observed most CPU time was spent for similarity calculation. Moreover, there is a strong tendency that CPU time is proportional to the number of similarity calculations.

TABLE II
CPU TIME (SEC) OF LGIC 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 shows the average taken for each combination 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 when P_c gets larger, the number of rule candidates will get larger.

TABLE III
AVERAGE CPU TIME (SEC) OF LGIC 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)

Since we came to know tol_diff seriously influences the success rate of LGIC, we changed it from 100 to 150. Then the range to search variable parameters is enlarged. Table IV shows the success rates of LGIC error correction for tol_diff = 150. When $P_t = P_c \times P_o \leq 3/16$ (= 0.188), LGIC with tol_diff = 150 discovered the true grammar with 80 or 100 % (= 4 or 5 out of 5 runs). However, when P_t exceeds about 1/4 (= 0.25), the success rate dropped to 20 % for $P_c = 0.5$ and $P_o = 0.5$.

TABLE IV
SUCCESS RATES OF LGIC ERROR CORRECTION FOR R-TYPE TRANSMUTATION (ORIGINAL PLANT MODEL EX05N, 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 one of plant models transmuted with P_c = 0.50 and P_o = 0.50. LGIC with tol_diff = 150 successfully discovered the true grammar for this transmuted plant.

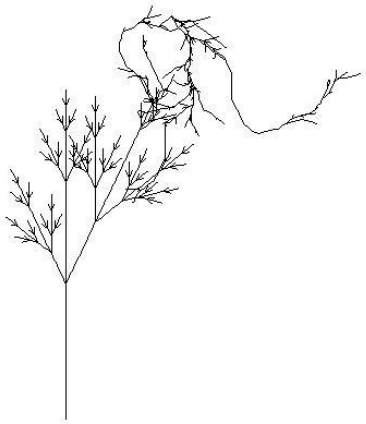

Fig. 4. Plant model transmuted from ex05n (P_c = 0.50, P_o = 0.50)

Figure 5 shows one of plant models transmuted with P_c = 0.75 and P_o = 0.25. Even for this transmuted plant, LGIC with tol_diff = 150 discovered the true grammar.

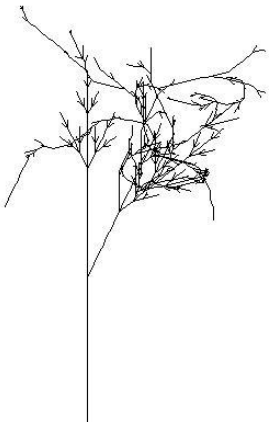

Fig. 5. Plant model transmuted from ex05n (P_c = 0.75, P_o = 0.25)

Table V shows CPU time spent and the number of similarity calculations needed by LGIC with tol_diff = 150. 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 rather obvious LGIC with tol_diff = 150 will fail; thus, skipped. Again, there is a tendency that CPU time is proportional to the number of similarity calculations.

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

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Table VI shows the average taken for each combination of Table V. 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. This may reflect when P_c gets larger, the number of rule candidates will get larger. Moreover, as is rather obvious, 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 VI
AVERAGE CPU TIME (SEC) OF LGIC 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

V. CONCLUSION

This paper proposed a method of grammatical induction with error correction for deterministic context-free L-system. Given a transmuted string, the method induces L-system grammar candidates. As transmutation this paper focuses only on replacement-type. In the method, a set of parameter values is exhaustively searched and if it is located within the tolerable distance from a point determined by the given string, then the parameters are used to form rule candidates. Rule candidates are used to generate a candidate string, and the similarity between a generated string and the given one is calculated, and candidates having the strongest similarities are shown as the output. Our experiments showed the proposed method discovered the true L-system grammar when the transmutation rate is less than around 20%. In the future we plan to extend the method to cope with other types of transmutations such as deletion-type or insertion-type.

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*, 2010, pp. 194–199.
[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*. MIT Press, 1990.
[8] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

ISBN: 978-988-19251-6-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)