Obtaining L-systems Rules from Strings

Edmar Santos[1], Regina C. Coelho[2]

ABSTRACT
This paper presents a proposal to solve the Inverse Problem of Lindenmayer in the deterministic and free-context L-system grammar class. The proposal of this paper is to show a methodology that can obtain an L-system rule from a string representing the development stage of any object. The strings used in the tests were obtained from known grammars. However, they are dealt with as of having an unknown origin to assure the impartiality of the methodology. The idea presented here consists in the regression of growth of the string analyzed by an algorithm built based on relations of growth obtained from string generated by known deterministic grammars. In the tests carried out, all the strings submitted to the proposed algorithm could be reverted to an L-system rule identical to the original rule used in the synthesis of the string. It is also interesting to observe that the obtaining of these rules occurred practically in real time with tested grammars.

Key Words: L-systems. Inverse Problem. Inverse Problem of Lindenmayer.

1. INTRODUCTION

Grammatical systems of parallel rewrite, known as Lindenmayer Systems or L-systems [1], have become the target of scientific focus of many researchers. In the original conception of the L-systems only biological proposals were presented, destined for the representation of development processes of living creatures with very simple graphic representations [2]. However, considering the potentialities visualized in those systems, other more improved approaches have been presented, including non-biological proposals and more sophisticated mechanisms for graphic representations of those processes. L-systems have been used, for instance, for the generation of fractals [3], [6]; such as Peano curves [4]; for the realistic image synthesis of trees and some plants [5], [6]; for the analysis of cellular layers development process [7]; for the generation of digital straight lines [8]; for the recognition of shapes in Computer Vision [9], [10]; for the generation of musical notes

[1] UNASP – Adventist Universitary Center of São Paulo. Postal Code 05.859-001 – São Paulo – SP - Brazil edmar.santos@unasp.edu.br

[2] UNIFESP – Federal University of São Paulo. Talim Street, 330 – Postal Code 12.231-280 – São José dos Campos – SP – Brazil - rccoelho@unifesp.br

[11]; for searching of neural network structures [12]; for the virtual creatures synthesis for simulation environments [13]; for artificial neural synthesis [14]-[17], and many other approaches.

Nevertheless, the great problem of the studies that use the L-systems is determining what appropriate system can conveniently represents the development process of the desired object. This problem can be understood as a case of Inverse Problem, a widely studied theory in areas such as geophysics [18], [19], physics [20], research with medical images [21], [22], and others. This theory, in general, deals with a problem where one intends to infer the representation of a model based on the observation and analysis of data belonging to the same system in that model [23]-[26]. In the L-system theory this problem is known as the Inverse Problem of Lindenmayer.

This work intends to collaborate with the solution the Inverse Problem of Lindenmayer. For that, a methodology is proposed to explore the self-similarity characteristics and parallel rewrite of those systems. The idea consists in reverting one string, derived of a determined state of development of one self-similar structure, in one L-system rule. This rule must allow the reproduction of that object. It is assumed that those strings have already been obtained by some process, that is, this work does not intend to discuss the methods of obtaining those strings.

Thus, this work is structured in the following way: in section 2 the correlated studies are presented, indicating the state-of-the-art the Inverse Problem of Lindenmayer. Next, in section 3, the formalism of L-systems, including some concepts of formal languages is presented. Section 4 brings the proposed methodology in this paper. In section 5, the results obtained and the discussions about these results are presented, followed by the conclusions in section 6.

2. CORRELATED STUDIES

In the field of Formal Languages Theory, the question of the Inverse Problem is known as Grammatical Inference, many times referred only as Grammatical Induction. The processes of grammatical inferences have been studied, for instance, in areas such as machine learning [27], computational biology [28]-[30], pattern recognition in structured documents [31], speech recognition [32], etc. Great efforts have been made in the attempt to achieve methods capable of assisting us in the

search of formalisms for the studies of these processes in these diverse areas. It is cited, for instance, the work of SCHWEHM e OST [33] that have proposed a method for obtaining of stochastic regular grammars using a genetic algorithm in a high-power multiprocessed system. Their work presents interesting results, since the obtained grammar in their test differs in just centesimal values in the probabilistic weights of the original rules, used as a model.

The Inverse Problem appears as a particularity of Grammatical Inference, now acting in the context of Lindenmayer Systems formalism [34]. This it means that, in the search processes for L-system grammars, the inherent properties in these systems must be taken into account.

We can cite the work of KOZA [35] as one of the first works related to Inverse Problem which presents a methodology to discover rules for the Lindenmayer system based on a genetic algorithm. Koza was able to obtain a similar rule proposing some manual adjustments to make it identical to the original rule. In his tests, Koza used an L-system grammar to construct the fractal of the Quadratic Koch Island used in the tests.

RUDOLPH and ALBER work's [36] deals with a proposal to solve the Inverse Problem of Lindenmayer in the towers transmission project. The proposal is based on the execution of evolutionary algorithms that intends to converge the grammar generation in the genotype of intended object. The grammar rule obtained in each case was not revealed.

Another work that can be referred is the COSTA and LANDRY [37]. In this work, a method to describe tree-shape fractals and to reconstruct these models is formally proposed. That proposal consists in reconstructing an object by means of a grammar generated by a genetic algorithm from the sequences of images obtained around the original object. Their results point to similar shapes to the samples in the majority of tests. The rules of the grammar found, used in the reconstruction of the object, were not presented. In spite of the contribution of those studies, it is evident that the question about the Inverse Problem of Lindenmayer still challenges the researchers.

3. LINDENMAYER SYSTEMS

Formally introduced by the biologist Aristid Lindenmayer in 1968 [1], based on the Chomsky grammars [2], the Lindenmayer System, or just L-systems, was initially proposed for the representation of development processes of living creatures. It deals with a grammatical system of rewrite where the productions are substituted in parallel in each development phase. This confers to these grammar classes the recursive potentiality to easily represent complex objects.

The formal definition of an L-system grammar, very similar to the definition of the Chomsky grammar, can be make by a sorted quadruple {V, T, P, S} where V represents the set of variable symbols ($V \neq \{\}$).

Normally, the {S, F} symbols are used in V. T represents the set of terminal symbols; usually the symbols {+,-, [,]} are used, where T can be an empty set. P represents the set of production rules, where $P \subset V \times V^*$. S represents the axiom, were $S \in V+$. The big difference between Chomsky grammars and the L-systems is in the application order of the production rules. Whilst in the conventional grammars the production rules are applied in a sequential way, in the L-systems the application of the rules occurs in a simultaneous and parallel way along whole the string that will be rewritten.

The L-systems are used almost always associated with the graphic interpretation of their productions. The mechanism of graphic interpretation, proposed by Prusinkiewicz [2], is based on the turtle geometry of the Logo language [38] and became the classic mechanism for graphic interpretations at the L-systems. An example of L-system grammar and its graphic interpretation of production in the forth level, indicated by n, is presented in Fig. 1. The angle used in this example is indicated by α.

$$
\begin{aligned}
n &= 4 \\
\alpha &= 25^{\circ} \\
V &= \{S, F\} \\
T &= \{+, -, [,]\} \\
P &= \{\ F \rightarrow F[+F]F[-F] \\
S &= F
\end{aligned}
$$

Fig. 1 One example of the L-system grammar

4. OBTAINMENT OF L-SYSTEMS RULES

Firstly, the proposal presented in this paper determines the limits of exploration in the space of the classes in L-systems grammars. Next, the methodology on these limits is explained. These considerations are presented in the next sections.

A. Delimitation of the Problem

The Inverse Problem of Lindenmayer is seen as a problem of high complexity and it is not a simple task to solve it. This paper explore the class of Deterministic and Free-Context L-system grammars, and grammars that present the restriction of having the axiom formed by one symbol, and the grammar being constituted by one production rule. The grammar presented in Fig. 1 is an example in this category. It is also important to consider that strings used in this paper were obtained from known grammars, with the previous restrictions. However they were dealt as being of unknown origin, in order to allow the validation of the proposal. The intention is to present a methodology that allows finding an L-system rule using, for this, just one of these strings, regardless the production level at which such string has been originated. The rule found will have to be able to reproduce the string again.

The process of parallel and simultaneous rewrite that the string has gone through the productions is the only known factor. All the other information for solution of the problem is obtained from the analyzed string.

B. Proposed Methodology

Conceptually, the idea presented here is very intuitive. Observe in Fig. 2 the production scheme of an L-system grammar. This is the same grammar that was presented before in section 3. In this figure is possible to notice that, from the first production level, all of the others levels are composed exactly by the same parts. These parts are the results of the format of the grammar production rule, determined by the presence of terminal symbols acting as separators of the evolution points. The balloons indicate these exact points along the production string. It is noticeable that, after the exhaustive rewrite of the productions, these points will always be separated by the same terminal symbols. Then, one understands that the process of rewrite of

the L-systems confers to the production of these grammars a characteristic behavior.

Therefore, the idea of this paper is to revert at once the growth of the string promoted by the rewrite process, in all these evolution point indicated by balloons, as it is shown by the scheme of Fig. 3.

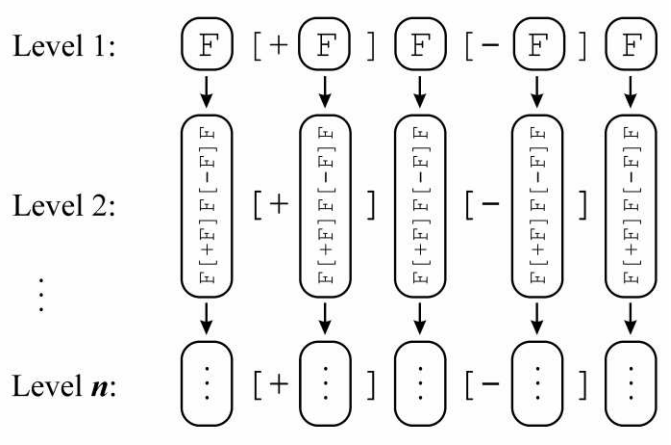

Fig. 2 L-system production scheme

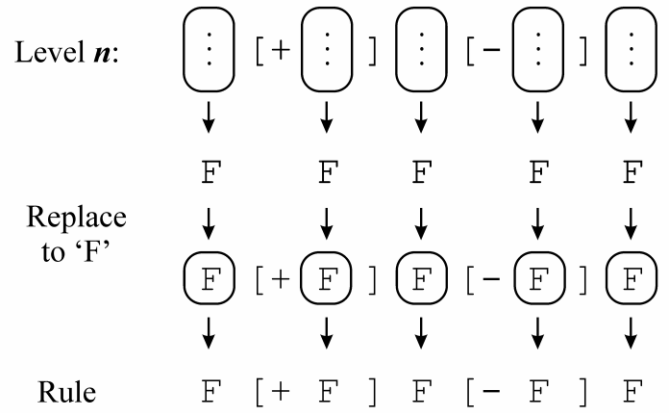

Fig. 3 Scheme of string reversion

To assure that the reversion process results in a rule potentially capable of reproducing the analyzed string, it is necessary to consider some important factors:

i. The quantity of `F' s in an L-system rule that could result in the analyzed string will have to be found;

ii. It will be necessary to verify if the reversion promoted in the analyzed string produces the same amount of `F' s found in (i);

iii. It will be necessary to verify if the string obtained in the reversion is capable of producing the same amount of terminal symbols presented in the analyzed string.

Thus, this proposal of reversion can be organized and expressed in the way of the presented algorithm in Fig. 4. Obviously, the formalization of these concepts requires the consideration of other factors. In the process of searching for the generating rule, the great problem is identifying the points of evolution in the string so that one can effect the regression process.

```
Function InverseProblem(w:string):string
begin
1   Ft←CountSimbol('F', w)
2   St←length(w)-Ft
3   if Ft>1 then
4    for Fq←2 to int(sqrt(Ft)) do
5      n←ln(Ft)/ln(Fq)
6      if isInteger(n) then
7        str←w
8        oldpatt←Slice(w,Ft,Fq)
9        if Replace(str,oldpatt,'F')=Fq then
10         rule←TerminalClear(str,n)
11         if isPossible(rule,Fq,n,St) then
12            return rule
13         end if
14       end if
15     end if
16   end for
17  end if
18  rule←w
End
```

Fig. 4 Algorithm 1

It is possible to establish the relation between the quantity of 'F's present in the string and its growth order during the productions through the analysis of known grammar string. This relation can be expressed by "(1)". In the equation, F_t represents the quantity of 'F' characters counted in the string, F_q represents the quantity of 'F's present in the production rules, and ***n*** represents the level of the production of string.

$$F_t = F_q^{\,n} \qquad (1)$$

According to this paper proposal, one can notice that only F_t can be known counting `F' characters present in the string that will be analyzed. This count is indicated by ***CountSimbol()*** in line 1 of the algorithm. If the return is 0 or 1, the string cannot represent a state of development of an object and the algorithm stops running and the result is ***w***. The conditional deviation in line 3 is responsible for this verification. The return of ***w*** also occurs in case the algorithm does not find any rule.

Thus, we proposed an iterative search in F_q to find values for the ***n*** exponent that can result in the already known value F_t, that is, one can find what quantity of 'F's in an L-system rule would be responsible for the production of the quantity of 'F's present in the analyzed string, and at which this would be possible in the ***n*** level.

Knowing the possible values of F_q and n is also fundamental to identify the evolution points in the analyzed string. The iteration in F_q ends in the highest possible value of an exponentiation capable of producing F_t, which is equivalent to the integer part of its own square root, according to line 4.

Therefore, considering the relation of Equation 1, the inverse relation can be established by means of a logarithmic function, shown in Equation 2, to determine the values for n that satisfy the equality. This is the relation expressed in line 5 of the algorithm.

$$\log_{F_q} F_t = \frac{\ln F_t}{\ln F_q} = n \qquad (2)$$

Once the iteration in F_q obtains an integer value for n, since F_t cannot be the resultant of a fractionary exponent, it is possible to know the probable format of the production rule of the analyzed string, that is, it is already possible to know how many `F' s will be present in the probable production rule and at which level the analyzed string may have been produced. This assures the first consideration presented in (i). The presence of the conditional deviation, in line 6, makes it possible for the algorithm to skip to the next iteration in case the obtained value of n is not an integer number.

In line 8, the ***Slice()*** function is required, whose parameters are: the values of F_t and F_q and the initial string. As the value of F_q indicates how many 'F's will be present in the sought rule, and each 'F' will be responsible for the production of a part of the string, so there must be equal F_q parts in this string and each one of these parts must have k 'F' characters, where $k = F_t / F_q$. Thus, this function must identify and return to the sub-string corresponding to one of these parts. This function was conveniently implemented to return to the first part, locating the contained sub-string since the first occurrence of 'F' until k.

In line 9, the function ***Replace()*** is required, whose parameters are: the ***str*** string, the ***oldpatt*** sub-string returned by the previous function, and the 'F' symbol. This function locates and substitutes all the occurrences of ***oldpatt*** in the ***str*** string for 'F'. This function also returns how many substitutions are made. A conditional deviation verifies if the quantity of substitutions corresponds to the number of parts that the probable rule should have, that is, if it is equal to F_q. If this condition is not verified, it means that the analyzed string cannot have been produced by a rule

formed by the of current F_q and ***n*** values and, therefore, the next iteration in F_q is executed; else, ***str*** already represents a rule capable of producing ***w*** and it is returned as a solution to the problem and the algorithm is finished. The conditional deviation existing in line 9 guarantees the consideration made in (ii).

However, some production rules can cause the accumulation of terminal symbols in the string. It is the case, for instance, of a rule such as $F + F$- . In Fig. 5, the production of this rule is presented in 3 levels, indicating at each level the parts produced by the `F' s of the previous level. This figure also indicates the sub-string returned by the function ***slice()*** applied to the production of the third level, followed by the substitution of this sub-string by the function ***replace()***. At the end, the accumulation of terminal symbols indicated by the balloons can be noticed.

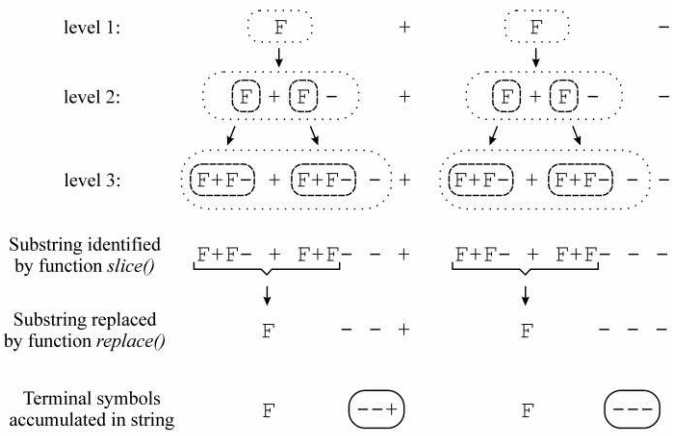

Fig. 5 Accumulation of terminal characters

The ***TerminalClear()*** function in the line 10 is required to eliminate these terminal symbols accumulated during the production levels. This function receives as parameters the ***str*** string resulting from the line before, and the value of ***n***. This function removes these symbols in the same proportion they were accumulated during the rewrite process. That is made by identifying how many characters were accumulated by the production level before the first occurrence of 'F'. In such a way, the quantity of terminal symbols existing before the first occurrence of 'F' is divided by ***n***. The quotient obtained indicates how many terminal symbols must be removed before 'F' from each production level. One can notice that the first level must be excluded from the calculation, since it represents the production rule per se, then, the quotient obtained is multiplied by ***n-1***. The product represents the total amount of terminal symbols that must be removed before all the occurrences of 'F' in ***str***. The function must execute the same operation for the existing symbols after each

occurrence of ‘F’ in its due proportion. In case it is verified that the string does not present accumulation of terminal symbols, no alteration is made in ***str***.

Finally, in line 11 of algorithm in Fig. 4, the function ***isPossible()*** receives the ***rule*** string resulting from the previous function, the values of F_q and ***n***, besides S_t, that indicates how many terminal symbols there are in ***w***. This function is required to verify if the string obtained by the reversion process is capable of producing the same quantity of terminal symbols existing in ***w***, without the need to effect the productions until the level indicated by ***n***. As in the rewrite process the terminal symbols are accumulated level by level, from the rule itself, it would be enough to know how many terminal symbols would be present in each level, and add them up.

It must also be noticed that for each existing `F' in string to be derived, the same quantity of terminal symbols present in the production rule will be inserted, plus the symbols already existing resulting from the last derivation. In such a way, the quantity of terminal symbols existing in one particular production level could be determined by Equation 3, in which F_q and ***T*** represent, respectively, the quantity of `F' s and the quantity of terminal symbols present in the rule.

$$S_r = \sum_{i=0}^{n-1} F_q^i \cdot T \qquad (3)$$

The returned value in S_r by Equation 3, representing the quantity of terminal symbols that would be present in the production ***rule*** until the level ***n***, must be compared with S_t. In case these values are equal, the result of the function will be ***true*** and the algorithm will return ***rule*** as a solution to the problem; otherwise, the algorithm will execute the next iteration, if there is any, or it will be finished returning ***w***. Thus, this function assures the final consideration made in (iii).

It is also important to observe that, for convenience’s sake, the implemented algorithm returns the first rule found capable of reproducing the string supplied for analysis, being ended in the sequence. However, if the algorithm were not interrupted, in some cases, it would be possible to find other rules potentially capable of reproducing the same entered string. This is the case, for instance, of the string “FFFFFFFFFFFFFFFF” that may have been produced by the application of one rule formed only by “FF”, in four levels of rewrite, or it may have been generated by the application of the rule “FFFF”, after the second level of production. The complexity of formation of the rules would just imply a slower or faster development process.

5. DISCUSSION AND RESULTS

Based on the above considerations, we performed the analysis of several strings produced by grammars known in different productions levels. The proposed methodology was applied on two groups of strings:

- Group (a) - Strings produced by 20 grammars, and some of them are variations of grammars widely known, the most obtained from [34]: in this group, for each grammar, strings were generated from the level one until the level six of production. One hundred twenty different strings were tested.
- Group (b) - Strings produced by randomly generated grammars: a grammar generator of random sizes and variable complexity was developed. One million different grammars were generated. Each of these grammars was used to produce a string at the level four of iteration. Next, all the strings were submitted to the process reversal.

The grammar of the random generator was built based on a non-deterministic grammar. In this grammar, primitive strings can lead to a Context-free, Deterministic L-system rule. The strings used as primitive rules were: [F], + F, F +, F-, F- and FF. From an F axiom, the rewriting process is performed a random number of times (no more than 10 to avoid generating a very large rule). This process also ensures that no rule containing only one 'F' is returned. The code of this grammar generator is reproduced in the algorithm in Fig. 6. This code is executed in accordance with the number of grammars to be generated. In this algorithm, has a function called ***randomN()*** on line 3, with the parameter 10, which returns an randomly generated integer between 1 and 10. This number indicates how many times the process of rewriting in the formation of a new rule will be performed. In line 7, a similar function is called randomly to obtain a primitive string contained in G, which is the ***randomG()*** function. The G set contains primitive strings which will be used in the rewriting process to form new rules.

```
Procedure GrammarGenerator():string
begin
1   repeat
2     rule←'F';
3     for j←1 to randomN(10) do
4        str←Ø
5        for k←1 to length(rule) do
6           if rule[k]='F' do
7             str←str+randomG(G)
8           else
9             str←rule[k]
10          end if
11       end for
12    rule←str
13    end for
14  until count('F',rule)>1
15  result rule
End
```

Fig. 6 Algorithm 2

Rules obtained in the implementation of the algorithm in Fig. 6, were stored on a list to be excluded from all duplicated rules, leaving only different rules.

The observations of the implementation of the proposed methodology for the strings of the (a) and (b) groups are:

- The proposed algorithm in Fig. 4 was able to obtain the L-system rule in all the strings submitted to the test, both for strings of the (a) group and for the (b) group, returning the identical rule to the on used in the synthesis of the tested string;
- The search time of the L-system rules of these strings presented variations according to the length of the submitted strings. However, the execution of the algorithm showed that the solution is found inside of an extremely fast time interval. For instance, in the analysis of one of the largest tested string, having 7.872.138 characters (about 1836 pages in a common text editor), the rule was obtained, on average, in 688 milliseconds;
- Other relevant information, which can be obtained by the algorithm in Fig. 4, is the level of production where the rule found can reproduce the analysed string. This production level is given by the value ***n*** at the moment in which that a rule is found.

All the results presented were obtained in a system with a 1.800Mhz, 512MB RAM memory CPU configuration, running Windows XP SP2. The algorithms were developed using IDE / RAD Delphi7 Object Pascal language through.

6. CONCLUSION

This paper presented a proposal to solve the Inverse Problem of Lindenmayer in a subclass of L-system grammars. The tests carried out until now have produced results 100% exact. It was possible to obtain a rule that allowed the reproduction of the analyzed string in all the tests. We can notice that it is possible to obtain identical rules to the original rules used in the synthesis of the samples. Moreover, the execution of the algorithm to obtain these rules has evidenced an extremely fast execution time.

Future studies will involve the study of methods to automatic obtain strings, from known objects, or still, to obtain to methods that support other groups of L-system grammars, besides the class supported by this proposal, as for instance, grammar with more than one production rule.

References

[1] Lindenmayer, A. Mathematical models for cellular interaction in development. *Journal of Theoretical Biology.* 1968; **18** : 300-315.

[2] PRUSINKIEWICZ, P. Graphical applications of L-systems. In: *Proceedings of Graphics Interface '86 - Vision Interface '86.* 1986; 247-253.

[3] SZILARD, A. L.; QUINTON, R. E. An interpretation for DOL Systems by computer graphics, *The Science Terrapin.* 1979; **4** : 8-13.

[4] SIROMONEY, R.; SUBRAMANIAN, K. G. Space-filling curves and infinite graphs. In: H. EHRIG, H.; NAGL, M.; ROZENBERG, G. Graph grammars and their application to computer science, Second International Workshop, *Lecture Notes in Computer Science,* Springer-Verlag, Berlin. 1983; **153** : 380-391.

[5] AONO, M.; KUNII, T. L. Botanical tree image generation. *IEEE Computer Graphics and Applications*, Tokyo. 1984; **4**(5) : 10-34.

[6] SMITH, A. R. Plants, fractals, and formal languages. *Computer Graphics.* 1984; **18**(3) : 1-10.

[7] BOER, M. J. M.; FRACCHIA, F. D.; PRUSINKIEWICZ, P. Analysis and simulation of the development of cellular layers. In: LANGTON, C. G. et al. *Artificial Life II, SDI Studies in the Sciences of Complexity*, Addison-Wesley. 1991; **X** : 465-483.

[8] BRONS, R. Theoretical and linguistic method for describing straight lines. In: EARN-SHAW, *R. A. Algorithms for Computer Graphics*, Spinger-Verlag, Berlin Heidelberg. 1995; 19-57.

[9] HOLLIDAY, D. J.; SAMAL, A. Object recognition using L-system fractals. *Pattern Recognition Letters.* 1995; **16** : 33-42.

[10] SAMAL, A.; PETERSON, B.; HOLLIDAY, D. J. Recognition of plants using a stochastic L-system model. In: *Journal of Electronic Imaging*, SPIE. 2002; **11**(1) : 50-58.

[11] PRUSINKIEWICZ, P. Score generation with L-systems. *Proceedings of the 1986 International Computer Music Conference.* 1986; 455-457.

[12] AHO, I.; KEMPPAINEN, H.; KOSKIMIES, K.; MÄKINEN, E.; NIEMI, T. *Searching neural network structures with L Systems and genetic algorithms*. Tampere: Department of Computer Science - University of Tampere. 1997.

[13] HORNBY, G. S.; POLLACK, J. B. Evolving L-systems to generate virtual creatures. *Computers and Graphics.* 2001; **6(**25) : 1041-1048.

[14] HAMILTON, P. A language to describe the growth of neuritis. *Biological Cybernetic.* 1993; **68** : 559-565.

[15] MCCORMICK, B. H.; MULCHANDANI, K. L-system modeling of neurons. Visualization in *Biomedical Computing Conference Proceedings*. 1994.

[16] KALAY, A.; PARNAS, H.; SHAMIR, E. Neuronal growth via hybrid system of self-growing and diffusion based grammar rules. *I Bulletin of Mathematical Biology.* 1995; **57**(2) : 205-227.

[17] COELHO, R. C.; JAQUES, O. Generating three-dimensional neural cells based on bayes rules and interpolation with thin plate splines. *Lecture Notes in Computer Science* - Progress in Pattern Recognition, Speech and Image Analysis. 2003; **2905** : 675-682.

[18] MENKE, W. Geophysical data analysis: discrete inverse theory. Academic Press. 1989.

[19] PARKER, R. L. *Geophysical inverse theory*. Princeton University Press. 1994.

[20] BERTERO, M.; BOCCACCI, P. *Introduction to inverse problem in imaging*. Institute of Physics Publishing. 1998.

[21] NATTERER, F.; WÛBBELING, F. *Mathematical methods in image reconstruction*. Society for Industrial and Applied Mathematics. 2001.

[22] UHLMANN, G. *Inside out: inverse problems and applications*. Mathematical Sciences Research Institute Publications. 2003.

[23] CHALMOND, B. Modeling and inverse problems in image analysis. Springer. 2003.

[24] RAMM, A. G. Inverse Problems: mathematical and analytical techniques with applications to engineering. Springer. 2005.

[25] TARANTOLA, A. *Inverse problem theory and methods for model parameter.* Society for Industrial and Applied Mathematics. 2005.

[26] TARANTOLA, A. Popper, Bayes and the inverse problem. *Nature Physics.* 2006; **2**(8) : 492-494.

[27] HIGUERA, C. de La. Current trends in grammatical inference. In: F.J.F. et al. Advances in Pattern Recognition, Joint IAPR International Workshops SS-PR+SPR2000, *Lecture Notes in Computer Science*, Springer-Verlag. 2000; **1876** : 28-31.

[28] SEARLS, D. The computational linguistics of biological sequences. In: HUNTER, L. *Artificial Intelligence and Molecular Biology*, AAAI Press. 1993; 47-120.

[29] GRATE, L.; HERBSTER, M.; HUGHEY, R.; HAUSSLER, D.; MIAN, I. S.; NOLLER, H. RNA modeling using gibbs sampling and stochastic context free grammars. In: *Proc. of Second Int. Conf. on Intelligent Systems for Molecular Biology*, Menlo Park, CA: AAAI/MIT Press. 1994.

[30] SAKAKIBARA, Y.; BROWN, M.; HUGHEY, R.; MIAN, I. S.; SJOLANDER, K.; UNDERWOOD, R.; HAUSSLER, D. *Stochastic context-free grammars for tRNA modeling*. Nuclear Acids Research. 1994; **22** : 5112–5120.

[31] SAIDI, A. S.; TAYEB-BEY, S. Grammatical inference in document recognition: In: Grammatical Inference, *Lecture Notes in Computer Science*, London, Springer-Verlag. 1998; **1433** : 175-186.

[32] KASHYAP, R.L. Syntactic decision rules for recognition of spoken words and phrases using stochastic automaton. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 1979; **1**(2) : 154-163.

[33] SCHWEHM, M.; OST, A. Inference of stochastic regular grammars by massively parallel genetic algorithms. In: ESHELMAN, L. J. Genetic Algorithms: *Proceedings of the sixth Int. Conf. (ICGA95)*, Morgan Kaufmann Publishers. 1995; 520-527.

[34] PRUSINKIEWICZ, P.; LINDENMAYER, A. *The algorithmic beauty of plants*. Springer-Verlag, New York. 1990.

[35] KOZA, J. R. Discovery of rewrite rules in Lindenmayer systems and state transition rules in cellular automata via genetic programming. *Symposium on Pattern Formation (SPF-93)*, Claremont, California. 1993; 1-19.

[36] RUDOLPH, S.; ALBER, R. An evolutionary approach to the inverse problem in rule based design representations. *Proceedings 7th International Conference on Artificial Intelligence in Design*, Cambridge University, Cambridge, UK, July 15-17th. 2002.

[37] COSTA, E. L.; LANDRY, J. A. Generating grammatical plant models with genetic algorithms. *Proceeding of the International Conference on Adaptive and Natural Computing Algorithms*. Coimbra, Portugal: March 21-23. Springer Wien, New York. 2005; 230-234.

[38] ABELSON, H.; diSESSA A. A. *Turtle geometry*. M.I.T. Press, Cambridge. 1982.