Non-incremental inference of 0L-systems with positive sample

Farah Ben-Naoum and Mustapha Mechab
EEDIS, UDL, Computer Science Department. Sidi Bel-Abbes Algeria bp 89 22000

Keywords: grammatical inference, non incremental Inference, positive sample, 0L-system, rules generalization, self-similarity, heuristic training algorithm, method SAR.

Introduction

Many algorithms of grammatical inference were developed for several types of grammars (Adriaans, Fernau and Van Zaanen, 2002). The inference problem consists of finding, from a set of strings, a grammar that produces all the strings of this set (Miclet and Cornuéjols, 2002). We are interested here by the inference of a particular class of L-systems, noted the 0L-systems (Yokomori, 1992). We present a *heuristic algorithm*, which only uses positive sample corresponding to either independent or developmental biological structures. The *positive sample* noted I_+ is the set of all strings that the inferred 0L-system must produce. It is a *non-incremental algorithm* in the sense that the positive sample becomes unchanged (new strings cannot be added progressively) (Miclet and Cornuéjols, 2002).

Biological motivation

Developmental biology and inference: The problem of inductive inference has in recent years been extensively investigated. It has particular significance for developmental languages in which the description of the developmental stages of an organism is formally defined as a series of strings of symbols. The problem is to devise developmental rules which transform the strings of symbols in a way consistent with observations for a particular species. In this sense, there is an overlap between model building in developmental biology and the *grammatical inference* problem. This problem enters the realm of developmental biology in the following way. The biologist interested in a particular plant is confronted with a large number of experimental observations. His task is to explain on the basis of such experimental results the way in which the particular plant develops (Feliciangeli, Gabor and Herman, 1973).

Studying self-similarities in plant structures: In the growth processes of many living organisms, especially plants, regularly repeated appearances of closely related biological structures are readily noticeable. This phenomenon of *self-similarity* has been tentatively captured by several botanical notions (Prusinkiewicz 2004), (Ferraro, Godin, and Prusinkiewicz, 2005). Our main objective is to use the *grammatical inference* in the detection of this *self-similarity* from a symbolic representation of a tree structure in development, or from several tree structures corresponding to different varieties in the aim to compare them.

Complementary arguments for such a biological motivation, as well as some similar problems, are discussed in (Herman and Walker, 1972).

Specificities of the proposed method

Inference of L-systems was recently studied using genetic programming (Jacob, 1998). By proposing this new heuristic method we avoid some of the drawbacks of the genetic algorithms, i.e.: uncertainty on the algorithm convergence, unknown time of convergence, the great spatial complexity dependent on the size of the population ... (Koza, 1993). On the other hand, the advantages of this algorithm, regarding to the other heuristic algorithms already proposed (Doucet, 1974; Feliciangeli, Gabor and Herman, 1973; Herman and Walker, 1972; Nevill-Manning and Witten, 1997), are its ability to:

- work on tree structures while previous works were limited to simple sequences like red algae,
- perform on any kind of sequence (containing ordered or unordered strings of I_+, independent or developmental, structures), the others performs only on ordered sequences in development,

- generalize the rules by studying the possible existing recursivities. The generalization consists on the creation of a grammar that generates the language containing the positive sample. By comparison, let us consider the method of Nevill-Manning and Witten (1997) in which they proposed an algorithm that forms a grammar from a sequence based on repeated phrases in that sequence. Each repetition gives rise to a rule in the grammar, and the repeated subsequence is replaced by a non-terminal symbol. In this case the grammar is inferred from a positive sample containing only one sequence, and it can only generate this sequence without generalization (only non-recursive grammar are inferred by this method). This process cannot represent the biological development of a plant structure in which repeated modules as well as modules generated by a regular developmental model, called self-similar modules, can be found. The detection of self-similar modules and their representation by recursive rules avoids doing the generalization of the resulting L-system. This represents the specificity of our new method.

Description of the SAR method

The particularity of L-systems is their ability to models the development of higher plants and complex branching structures, described as configurations of modules in space; the term module denotes any discrete constructional unit that is repeated as the plant develops (Prusinkiewicz, Hammel, Mech, and Hanan, 1995).

The main idea in the inference of 0L-system generating a sequence of strings associated with the sequence of structures of plants is to explore all sub-strings of the sequence strings and to select those representing modules that correspond to two cases: In the first one, the selected module can be decomposed into other nested modules; the nesting must be done in a regular manner which leads to the creation of a recursive rule. In this case, the modules are called self-similar in the sense that they are composed of a succession of possibly nested identical sub-modules. In the second case, the selected module is only repeated in the sequence of the developmental structure, and then the rule inferred simply associates one symbol with this module. These two cases are considered respectively in the first and the second loop *repeat* of the SAR algorithm mentioned in the annex.

The selection of modules must take into account the vocabulary used to represent the branching structure in the 0L-systems using the turtle interpretation (Prusinkiewicz, Hammel, Mech, and Hanan, 1995). Each module corresponds to a valid plant structure if and only if the associated string (containing brackets to represent branches) is well-parenthesed.

The proposed ***Method SAR***: (search of self-similarities and redundancies) builds a 0L-system starting from a positive sample $I_+=\{x_i, i=1..|I_+|\}$ by making a partition of each string x_i into repeated or recursive sub-strings being able to be produced in the same step of derivation (to deal with the parallelism in the derivation in the L-systems). This partition includes well-parenthesed sub-strings lengths varying from n to 2 with n=len(x_i). The recursive application of this principle to unit I_+ enables us to reconstitute all the successive steps for the generation of all strings of I_+.

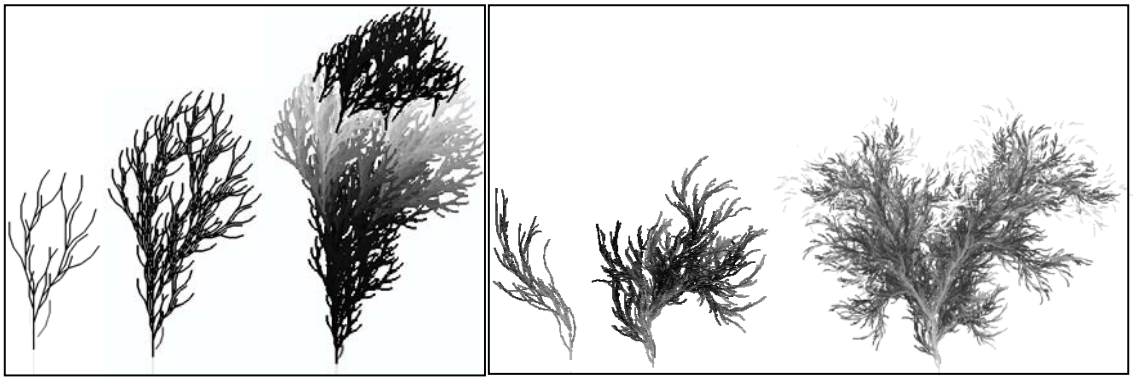
The method SAR identifies in the limit the 0L-system generating I_+ , since it makes it possible to reduce successively all the strings of I_+ after N steps (Gold, 1967), the algorithm then converges towards a solution, starting from a finite N, i.e.$H^{N-1}(I_+)= H^N(I_+)$. Where N is at most equal to the number of productions created by the algorithm plus 1.

The method SAR was developed in python which requires about 550 lines for the algorithm. The temporal complexity of SAR is $O(r.m.n)$, where $n=|x_i|$ and x_i is the longest string in the positive sample, m is the size of the positive sample I_+. In the worst case the algorithm creates 1 production at each step of the two loops *Repeat* doing the detection of nested modules and the detection of repeated modules in the SAR algorithm given in the annex, then the algorithm performs this creation of productions r times, with r=max(number of recursive versus non-recursive symbols).

Examples

1) **Positive sample I_+ containing strings associated to independent plants**. The first one correspond to the step 3 of development of the plant of figure2(a) generated by the L-system

L_1(Axiom: F productions: F→FF-[Fc-F+F+F]d+[Fc+F-F-F]d c→c, d→d; homomorphism c→, d→;), the second correspond to the step 2 of development of the plant of figure2(b) generated by the L-system L_2(axiom: F productions: F→FF+\[,+F-&Fc[-F+F+F+F]d-F]-\[,-&Fc[-F+F+F+F]d+F] c→c, d→d; homomorphism c→, d→;). The resulting inferred L-system must be the union of L_1 and L_2.



(a) (b)

Figure 2: (a) Steps 2, 3 and 4 of the plant generated by L_1,
(b) Steps 2, 3 and 4 of the plant generated by L_2, using the L-studio v.4.0.5 beta

Result of the SAR algorithm: axiom: h

productions:	homomorphism:
h -> b h -> a	b -> F
b->bb+\[,+bgbc[-b+b+b+b]d-b]-\[,gbc[-b+b+b+b]d+b]	a -> F
a -> aa-[ac-a+a+a]d+[ac+a-a-a]d	g -> -&
c -> c,	c -> ,
d -> d;	d -> ;

The resulting L-system is the union of L_1 and L_2. The development of trees associated to L_1 and L_2 correspond respectively to the recursive symbols a and b in the inferred L-system; representing the self-similarity in the development of the tow structures of figures 2(a) and 2(b). They also have common sequences of successive ‘,’ and ‘;’ represented respectively by the recursive symbols c and d. Then we find in the inferred L-system common rules corresponding to c and d.

2) **Positive sample I_+ containing a sequence of strings associated with different stages of development of the plant** (stages 2 and 3 of figure1).

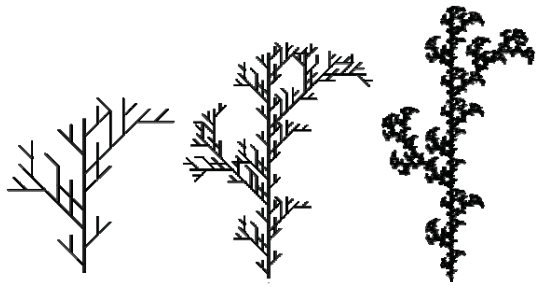


Figure1: Steps 2,3,5 of the plant generated by the L-system L_1(Axiom: A, A→A[+BA]A[-CA]A, B→BF[-FD], C→CF[+FD], homomorphism A→F, B→F[-FD], C→F[+FD], D→[F+F]) using L-studio v.4.0.5 beta

The resulting L-system L_2 must be equivalent to the one used to create the positive sample. Result of the SAR algorithm:

axiom: c

productions:	homomorphism:
c -> c[+bc]c[-ac]c	c -> F
b -> bF[-e]	b -> F[-e]
a -> aF[+e]	a -> F[+e]
	e -> F[F+F]

The symbols c, a, b are recursive and then represent self-similar parts in the tree of figure1.

To study the equivalence between this resulting L-system L_2 and the one used to generate I_+ (noted L_1), we verify whether they are isomorphic by an adequate change of the names of the generated symbols. The SAR algorithm detects the repetition of F[F+F] represented by the production

e→F[F+F] in L_2, we also remark the repetition of FD in L_1 with D→[F+F]. Then the symbol e is equivalent to FD. By replacing e by FD, and e→F[F+F] by D→[F+F] in L_2, and after replacing the axiom c (in L_2) by A (used in L_1), and the symbols b and a (in L_2) by respectively B and C (used in L_1), we obtain: (axiom: A, productions: D->[F+F], A->A[+BA]A[-CA]A, A->F, B->BF[-FD], B->F[-FD], C->CF[+FD], C->F[+FD]). We thus obtain a transformed L-system identical to that used to generate I_+.

3) **Positive sample I_+ containing seven strings associated with an abstract language**:

I_+={[aaaa[ccccccc]aaaadt]b[aaaa[cc]aaaadt]b[aaaa[ccccccc]aaaadt]b, dtcfcfcfdt[aaaaaa[ccccccc]aaaaaadt]b [aaaaaa[ccccccc]aaaaaadt]bdt, dgrtmmmAmAmmmAmAmmmmmAmAmmmAmAmmm, dthgjdt, [dt][dt][dt], [aaaaaa[ccccccccccc]aaaaaadt]b[aaaaaa[ccccccccccc]aaaaaadt]b, dtdthgdthgdtdtdthgdthgdtdt }

Result of the SAR algorithm: axiom: u

production	u -> dgrtA	p ->	A -> mAmAm	homomorp	o -> s
s:	u -> qkqpq	p[i[e]iq]b	k -> kcf	hism:	n -> [q]
u -> o	p ->	o -> qoqoq	i -> ia	u -> qsjq	k -> cf
u -> p	[i[e]iq]b	n -> n[q]	e -> ec	s -> hg	i -> a
u -> n				q -> dt	e -> c

This abstract example finally shows how the SAR approach may be used to infer rules from more complex structures.

Conclusion

This paper addressed the problem of inferring a 0L-system from a sequence of tree structures. To this end, we introduced the notion of self-similarity detected from symbolic representation of trees. In practice, the proposed SAR method proved its efficiency for the inference of abstract languages as well as for the languages which, associated with the turtle interpretation, expresses structures of trees and their development.

The class of 0L-systems inferred by this method concern parenthesized languages generating tree structures, to infer more realistic structures we must first study the learn-ability of each class of L-systems (Fernau and De la Higuera, 2004; Yokomori, 1992) and next try to find the appropriate extension of our method. For example, in the future, we attempt to combine our inference method to an adaptation of some methods of learning stochastic grammars in the aim to generate stochastic L-systems. We also aim to perform the algorithm to work incrementally, and include negative samples (strings that the inferred 0L-system must refuse). This method can be considered as a theoretical step towards more practical results in the future. In this aim, we intend to compare the SAR method with the method described in (Godin and Ferraro, 2007) for quantifying self-similarity in plants.

Acknowledgments: I would like to thank Christophe Godin and Pascal Ferraro for the interactive feedback that helped creating the SAR-method.

References

Adriaans, P. , Fernau, H., Van Zaanen, M., (2002). *Grammatical Inference: Algorithms and Applications; 6th International Colloquium, ICGI 2002*, volume 2484 of *LNCS/LNAI*. Springer.

Cornuéjols, A., Miclet, L., (2002). *Apprentisage artificiel : concepts et algorithmes*. Eyrolles.

Doucet, P. G., (1974). *The syntactic inference problem for D0L-sequences*, lecture notes in Computer Science 15.

Feliciangeli, H., Gabor, Herman, T., (1973). *Algorithms for producing grammars from sample derivations: a common problem of formal language theory and developmental biology*, journal of computer and system sciences 7, 97-118.

Fernau, H., De la Higuera, C., (2004). *Grammar Induction:An Invitation for Formal Language Theorists,* Grammars **,** volume 7, 45–55.

Ferraro, P., Godin, C., Prusinkiewicz, P., (2005). *Toward a Quantification of Self-similarity in Plants*, in Fractals, Vol. 13, No. 2, 91-109.

Godin, C. and Ferraro, P. (2007). *A general method for quantifying the structural self-similarity of trees*. Technical report, INRIA.

Gold, E.M., (1967). *Language Identification in the limit*, Information and control, Vol.10, No.5, pp, 447-474.
Herman, G.T., Walker, A. D., (1972). *The syntactic inference problem applied to biological systems*, in "Machine Intelligence 7" (Michie, Ed.), Edinburgh University Press, Edinburgh.
Jacob, C., (1998). *Genetic L-system programming*, Chair of programming Languages, Department of computer Science, University of Erlangen-Nurnberg, Germany.
Koza, J. R., (1993). *Genetic programming, On the programming of computers by means of natural selection*, MIT Press, London.
Nevill-Manning, Craig G., Witten, Ian H., (1997). *Identifying Hierarchical Structure in Sequences: A linear-time algorithm*. Journal of Artificial Intelligence Research 7, 67–82
Prusinkiewicz, P., (2004). *Self-similarity in plants: integrating mathematical and biological perspectives*. In: M.M. Novak (Ed.) Thinking in Patterns. Singapore: World Scientific.
Prusinkiewicz, P., Hammel, M., Mech, R., Hanan, J., (1995). *The Artificial Life of Plants*. From artificial life for graphics, animation, and virtual reality, volume 7 of SIGGRAPH'95. ACM Press.
Yokomori, T.,(1992). *Inductive inference of 0L languages*, in: G. Rozenberg and A. Salomaa (eds.), Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology, Springer, 115-- 132.

Annex: Pseudo code of the SAR algorithm:

Repeat for all strings of $I_+=\{x_i, i=1..|I_+|\}$ *#detection of nested modules*
  For all sub-strings y_j of x_i with $2\leq len(y_j)\leq len(x_i)$ and y_j is well-parenthesed:
    If **is_recursive1**(y_j)=true then
      Create a recursive rule of the form $A\rightarrow c_1Ac_2\ Ac_3\ \ldots c_{d-1}\ Ac_d$ and $A\rightarrow\beta$
      Replace the module y_j in x_i by the symbol A.
      Add sub-modules $c_1,c_2,c_3\ \ldots c_{d-1},c_d,\beta$ to I_+ in the aim to be also considered later.
    else if **is_recursive2**(y_j)= true then
      Create the productions $A\rightarrow Aa$, $A\rightarrow a$
      Replace occurrences of y_j in x_i by the symbol A.
      Add the sub-module a to I_+ in the aim to be also considered later.
Until stabilization of I_+
Repeat for all strings of $I_+=\{x_i, i=1..|I_+|\}$ *#detection of repeated modules like in (Nevill-*
  For all sub-strings y_j of x_i with $2\leq len(y_j)\leq len(x_i)$ and y_j is well-parenthesed: *# Manning and Witten, 1997)*
    If y_j is quite simply a repetition in I_+ (i.e. if the number of occurrences of y_j in strings of I_+ is >1) then
      Create the production $A\rightarrow y_j$
      Replace all occurrences of y_j by the symbol A
      Add the sub-module y_j to I_+ in the aim to be also considered later.
Until stabilization of I_+.

Function **is_recursive1**(y_j) *# this function verifies whether y_j correspond to a module that can be regularly*
  begin *decomposed into other nested modules (if y_j is self-similar)*
  $A_0=y_j$ *#max-len-red try to find the longest repeated sub-string of y_j that are also*
  A_1=max-len-red(A_0) *well-parenthesed*
  While max-len-red(A_k)≠'' and are isomorphic to all the precedent created $A_1..A_{k-1}$
    A_{k+1}= max-len-red(A_k)
    Replace occurrences of max-len-red(A_k) in A_k by A_{k+1}
    k=k+1
  if k≥2 then **is_recursive1**(y_j)=true
    return the list $(c_1,c_2,\ldots,c_d,\beta)$ with $y_j= c_1A_0c_2\ A_0c_3\ \ldots c_{d-1}\ A_0c_d$ and $A_k=\beta$
    else **is_recursive1**(y_j)=false
  end

Function **is_recursive2**(y_j) detects the existence of the smallest sub-module "a" in y_j so that y_j is composed of a succession of this module (y_j="aa...a"), if this module exists this function takes the value of true and returns "a".

Example explaining the steps of function **is_recursive1(y_j):**

Let y_j= "aaaβbβcbaβbβccbaaβbβcbaβbβccc", Thus we have:

A_1=max-len-red(y_j)= "aaβbβcbaβbβcc"	⇒	y_j="aA_1bA_1c"
A_2=max-len-red(A_1)="aβbβc"	⇒	A_1="aA_2bA_2c"
A_3=max-len-red(A_2)="β"	⇒	A_2="aA_3bA_3c"
max-len-red(A_3)=""	⇒	end of the loop while

Remark that all modules y_j, A_1, A_2 are isomorphic (they have the same structure up to the indexing of of A_k). We find here a suite in the development of the module y_j that can be generalized by the rules $A\rightarrow aAbAc$, $A\rightarrow\beta$. A is called recursive symbol.