

Production-rule complexity of recursive structures

Konstantin L Kouptsov
New York University
klk206@panix.com

Complex recursive structures, such as fractals, are often described by sets of production rules, also known as L-systems. According to this description, a structure is characterized by a sequence of elements, or letters, from a finite alphabet; a production rule prescribes replacement of each letter by a sequence of other letters.

Practical applications of rewriting systems however are quite restricted since the real structures are not exactly recursive: they involve casual deviations due to external factors. There exist fractals corresponding to the strings, invariant under the rewriting rules. Some strings which are not invariant may also characterize a fractal structure. Sometimes the sequence of symbolic descriptions of a structure is known while the rewriting rules yet to be discovered. Following the Kolmogorov's idea to separate regular and random factors while considering the system's complexity it may be possible to filter out the underlying recursive structure by solving an inverse L-systems problem: given a sequence of symbolic strings determine the simplest (if any) rule set performing the transformation.

The current paper is a work-in-progress report, presented at the Fourth International Conference on Complex Systems (Nashua, 2002), in which I give a preliminary account for issues involved in solving the inverse problem, and discuss an algorithm for finding the shortest description of the symbolic strings (analogous to the Chaitin's algorithmic complexity).

1 Introduction

Biological structures are well known for their hierarchical organization which is dictated by the efficiency of functioning of the organism as a whole. Their description sometimes admits a recursive approach: the structure of a particular component at one level of hierarchy is a composition of lower-level parts; the structure of a single component largely resembles the structure of a whole.

For phenomenological description of plants structure, a biologist Aristid Lindenmayer proposed in 1968 a mathematical formalism [7] called *L-systems*. The L-systems language well suits for definition of fractal structures such as Koch and Hilbert curves, Sierpinski gasket, fern leaf. The language provides a simplified version of the object structure by abstracting from the random variations. In spite of that, it catches the very essence of the geometry: the computer model of the object based on the L-description looks quite like a live photograph (recursive generation of landscapes and plants is fast enough to be used in computer animations, movies, and games).

The role of L-description of a recursive structure can be understood in the context of the Kolmogorov-Chaitin complexity [2, 5]. An individual finite object (which may be encoded by a finite binary string) contains the amount of information equal to the length $l(p)$ of the shortest (binary) program p that computes the object. An object encoded by a trivial binary string such as $00000\dots 0$ contains minimal amount of information, whereas the completely random string (obtained by flipping the coin) represents the minimal program by itself, so the complexity of a random string is proportional to its length. For the intermediate objects the description contains both regular (meaningful) information and the random (accidental) one. At a Tallinn conference in 1973 A. N. Kolmogorov has proposed to consider the separation of these two parts in the case of finite set representation. This idea was later generalized [4, 11]. The approach is the following: given the data object D , identify the most probable (optimal) finite set A of which the object is a typical example. The shortest description of the set A is a “program” p , containing no redundant information. The choice of D within A is the “data” d , the random part. The minimization of description length is performed over all sets $A \supset D$. The complete description of the object is given by the pair $\langle p, d \rangle$ which can be represented as a single binary string.

Despite the seemingly machine-oriented definition, the algorithmic complexity is, on the contrary, a universal and absolute characteristics of an object, up to an additive constant, and is machine independent in the asymptotic sense. Expression of the shortest program computing the object using different languages may involve an overhead which can be attributed to the redundancy of the language and contain information for translation of one language into another. Practical computer languages are highly redundant, as they contain long words and combinations of symbols that themselves are not chosen randomly, but follow some predefined pattern. In order to compare “lengths” of the programs written in these languages one has to introduce quite

sophisticated probability measures, putting less weight on pattern words. On the other hand, these measures are completely irrelevant to the problem and complicate things.

Kolmogorov and independently Chaitin [2] proposed to consider binary programs in which every bit is 0 or 1 with equal probability. Definition of algorithmic (Kolmogorov-Chaitin) complexity may, for example, be done by means of specialized (prefix) Turing machines. Let T_0, T_1, \dots be a set of Turing machines that compute x . The *conditional complexity* of x given y is

$$K(x|y) = \min_{i,p} \{l(p) : T_i(\langle p, y \rangle) = x\}. \quad (1.1)$$

The unconditional *Kolmogorov complexity* of x is then defined as

$$K(x) = K(x|\epsilon), \quad (1.2)$$

where ϵ denoted the empty string. The variables i and p are usually combined into $\lambda = \langle i, p \rangle$, and the above definition is given in terms of the universal Turing machine $U(\langle \lambda, y \rangle) = T_i(\langle p, y \rangle)$, where the minimum is calculated over all binary strings λ . Every (self-delimiting) binary string is a valid program that, if halts, computes something. In [3], Chaitin demonstrates quite practical LISP simulation of a universal Turing machine and discusses the properties of algorithmic complexity.

More intuitive approach to complexity suggests that a complexity of two completely irrelevant objects must be the sum of complexities of each of them. In order for the algorithmic complexity to correspond to this approach, the binary program must be *self-delimiting*, such that two programs appended into a single string can be easily separated.

2 Binary versus extended alphabet

The reasoning behind the choice of binary description is quite clear. We need a description (or a programming language, or a computational model) that is:

- (a) *Non-redundant, or syntactically incompressible.* Every symbol must bear maximum amount of information. There must be no correlation between symbols in the program, as any correlation would allow reduction of the length of the program. However, if a shorter program yields exactly the same object that a larger program, the shorter one may be considered as *semantically-compressed* version of the larger one.
- (b) *Syntax-free.* Any combination of symbols of an alphabet must be a valid program. Then a simple probability measure can be introduced. Existence of a syntax also means that we assume some structure of an object beforehand, which gives incorrect complexity measure.

There is a contradiction between self-delimiting and syntax-free properties of a program. Two of the possible delimiting schemes either use a special

closing symbol (LISP [2]), or a length-denoting structure in the beginning of the program [3]. Both introduce a bias in symbol probabilities.

- (c) *Problem-oriented.* Binary numbers seem natural choice for information encoding, especially considering the current state of computer technology. However, they are not applicable to description of tri-fold or other odd-fold symmetry objects, since there is no finite binary representation of $1/(2n+1)$ numbers. Thus, the binary description of a Koch curve (Fig. 1.2), for example, is quite awkward. However, the curve is very elegantly encoded using 3-letter alphabet.

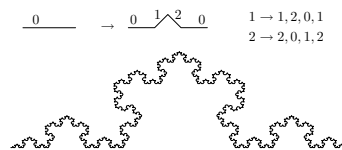


Figure 1.1: Ternary encoding of the Koch curve

Thus the extended alphabet is sometimes preferable to $\{0, 1\}$ alphabet.

Recursive structures, such as fractals, are conveniently described by a set of production rules, the L-systems, operating on a finite alphabet with N letters. It was shown (see overview in [10], Section 3.2) that the *parallel derivation grammar* may have the same generating power as the formal grammar of the Chomsky type. According to the Church's Thesis, the recursively enumerable languages describe the largest set obtainable by any effective computational model. Remarkable illustration of this point is the computation performed by a cellular automaton (an example of an L-system) [12].

Thus, the L-system complexity measure, satisfying all the requirements above, is equivalent to the Chaitin's one and convenient for our purposes.

3 Parallel derivation systems

The parallel derivation systems (string rewriting systems, L-systems) have received due attention over the past years. Among many publications in this subject I would like to mention the comprehensive works by Book and Otto [1] and P. Vitányi [10].

An L-system is a string rewriting system. Let *alphabet* $A = \{a, b, \dots\}$ be a finite set of elements, or *letters*. The *free monoid* A^* generated by A is a set of all finite strings over A , i.e.

$$A^* = \{\lambda, a, b, aa, ab, bb, ba, aaa, \dots\},$$

where λ is the empty string. Elements of A^* are called *words*. Let $A^+ = A^* \setminus \{\lambda\}$. A *rewriting system* is a set of *rules* $(u, v) \in A^+ \times A^*$, written as $u \rightarrow v$.

We start from the initial string $S_0 \in A^+$. At every consecutive state $n + 1$ the string S_{n+1} is obtained by simultaneous replacement of each letter $u \in S_n$ by a string v which may be empty. The rewriting may depend on l left and r right neighbors of the letter — hence $(l, r)L$ notation. The resulting string S_{n+1} is a concatenation of all v 's. The process is repeated thereby generating an infinite number of strings.

It is also possible to divide letters of the alphabet into *terminals* and *non-terminals* requiring the final strings to consist only of terminals. This action makes great attribution to the derivation power of an L-system.

4 Invariant strings and limit process

The Cantor set description can be generated by an L-system over a binary alphabet using starting string “1” and rules: $0 \rightarrow 000$, $1 \rightarrow 101$. Further, a geometrical meaning to the 0,1 symbols – a line of length $1/3^n$ colored white or black may be assigned. The infinite string 10100010100000000101... appears to be invariant under these generating rules.

Other examples of invariant strings are

- a) the rabbit sequence 1011010110110... with the rules $0 \rightarrow 1$, $1 \rightarrow 10$,
- b) Thue-Morse sequence 011010011001... with $0 \rightarrow 01$, $1 \rightarrow 10$,
- c) Thue sequence 1101101111101101111... with $1 \rightarrow 110$, $0 \rightarrow 111$.

The invariance property of an infinite string S_∞ means its self-similarity in the fractal-geometry sense: one can regenerate the whole string from a specially chosen piece by applying the rules (instead of just amplification). The subsequent finite strings S_n in this context are the *averaged* description of the object. Application of the rules to every letter of S_n may then be seen as a *refinement* of the structure of the object. Thus, for the Cantor set, $1 \rightarrow 101$ reveals the central gap, and $101 \rightarrow 101000101$ further gaps in the left and right segments.

We have seen that S_n converges to S_∞ in the refinement process. There has been observed [8, 9] another example of convergence in which the symmetry of the approximating (finite) strings in no way resembles the symmetry of the limiting structure (Fig. 1.4 (a)).

In this example, the generating process starts from a very irregularly shaped curve which suggests nothing about the final curve. After few steps of “refinement” the curve starts acquiring the right shape, however the convergence is only in some average sense. The final curve is a quite symmetric fractal-looking line having its own generating rules (and a fractal dimension of $D = 1.246477\dots$) (Fig. 1.4 (b)).

By combination of the subtle analysis of the underlying model [6] and a brute force, it is possible to find the generating rules for approximating curves. There are 32 “letters” (numbers) in the *primary* alphabet. Each number is associated with a vector representing one of possible 14 directed line elements composing

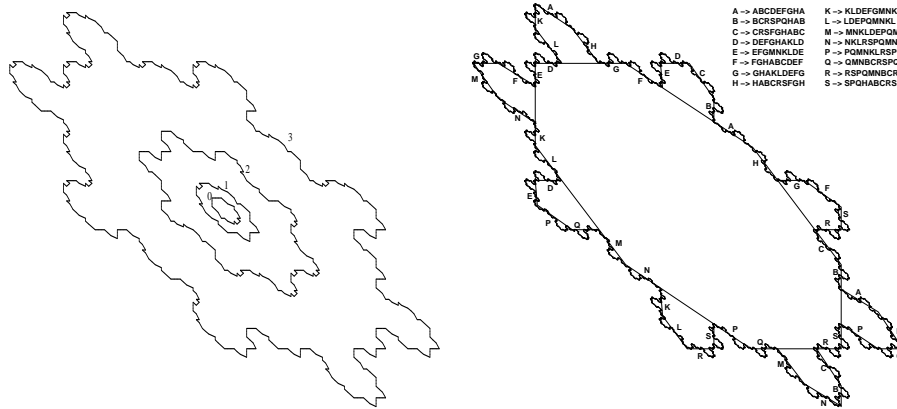


Figure 1.2: a) Successive approximations S_n to a fractal curve S_∞ . b) S_∞ with its own generating rules.

the curves,

$$\left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} 3,7,10,15,19,23,27, \begin{pmatrix} 1 \\ 0 \end{pmatrix} 14,26, \begin{pmatrix} 0 \\ 1 \end{pmatrix} 18,25,31, \begin{pmatrix} 1 \\ 1 \end{pmatrix} 16,17,21, \right. \\ \left. \begin{pmatrix} 2 \\ 1 \end{pmatrix} 13,28, \begin{pmatrix} 1 \\ 2 \end{pmatrix} 30, \begin{pmatrix} 2 \\ 2 \end{pmatrix} 29, \begin{pmatrix} 1 \\ -1 \end{pmatrix} 1, \begin{pmatrix} 0 \\ -1 \end{pmatrix} 2,12, \right. \\ \left. \begin{pmatrix} -1 \\ 0 \end{pmatrix} 11,20,24, \begin{pmatrix} -1 \\ -1 \end{pmatrix} 6,8,9, \begin{pmatrix} -1 \\ -2 \end{pmatrix} 0,4, \begin{pmatrix} -2 \\ -1 \end{pmatrix} 22, \begin{pmatrix} -2 \\ -2 \end{pmatrix} 5 \right\},$$

where the subscripts denote associations. The vectors can be grouped into sequences (*words*) thus forming the *secondary* alphabet.

$$\begin{aligned} A &= (27, 16, 21) & B &= (26, 12, 20, 28) & C &= (29, 21, 9) & D &= (19, 22) \\ a &= (5, 2, 15) & b &= (7, 14, 1, 8) & c &= (22, 5, 11) & d &= (9, 3) \\ E &= (30) & F &= (24) & U &= (26, 12, 20, 14, 1, 18) & o &= () \\ e &= (25) & f &= (6) & u &= (22, 5, 10, 21, 9) \end{aligned} \tag{1.3}$$

Capital and small letter notation represent the symmetry of the curves with respect to the long diagonal line, as one can see by plotting sequences of vectors for each letter. It appears that these sequences are the forming blocks of the curves. Note that there is a *null* symbol in the alphabet.

Initial string for the 0-th level becomes

$$\begin{aligned} &14, 1, 18, 5, 2, 15, 9, 3, 6, 22, 5, 10, 21, 9, 24, 19, 22, 27, 16, 21, 26, 12, 20 \\ &= adfuFDAU, \end{aligned}$$

and the rewriting rules become

$A \rightarrow dfca$	$B \rightarrow dfue$	$C \rightarrow Ua$	$D \rightarrow dba$	$E \rightarrow oAUa$	$F \rightarrow o$
$a \rightarrow ACFD$	$b \rightarrow EUFD$	$c \rightarrow Au$	$d \rightarrow ABD$	$e \rightarrow Auao$	$f \rightarrow o$
$U \rightarrow Eue$	$u \rightarrow U$	$o \rightarrow dfUFD$			

(1.4)

Emergence of the short-diagonal symmetry of the curves is quite remarkable, since it is not at all reflected in the rewriting rules.

One can think that the shape of the limit curve S_∞ is determined only by the rewriting rules (1.4). However, using the same rewriting rules and a different S_0 , we obtained different fractal curves with other fractal dimensions. Thus the encoded information about the shape of S_∞ , and its properties in general, is shared by the initial string and the rewriting rules. The author finds it challenging to make a connection between the properties of S_0 or (1.4) and the final object.

The generating rules given above are not unique. They are given in the above form for illustration purposes, and have not been optimized or “spruced up” in any way. Other, neater and more concise rule systems are possible. How can one possibly find another set of rules, or enumerate all of them, or find the shortest one? (Of course, approach with “isomorphisms” would do part of the job). Generally, how to systematize the brute force approach?

5 Guess-Match brute-force algorithm

Given a set of strings $\{S_n\}$, we would like to answer a few questions. Is there a finite *dictionary*, i.e. a set of letter sequences acting as building blocks for all the strings (in a way that english clauses are made up from words)? Are these strings related by substitution rules? If yes, what is the substitution system having a particular property (e.g. the shortest)?

The answer to the first question is a different matter and is not addressed in the current paper. Here we assume that the dictionary and the rewriting rules exist.

From a sequence $\{s_n\}_{n=1}^\infty$ of strings we form a set of pairs $\mathcal{S} = \{\dots, (S_n, S_{n+1}), (S_{n+1}, S_{n+2}), \dots\}$. Generally, the algorithm should work for any set of pairs, and this property is used in the recursion.

Let $\mathcal{V} = \{u_n \rightarrow v_n\}$ be a set of rules. In the proper rule $r = u \rightarrow v \in \mathcal{V}$ the u part is non-empty. The set $\{u_n\}$ must be *prefix-free*, so there is no confusion whether $0 \rightarrow \dots$ or $00 \rightarrow \dots$ is to be used in $\dots 00 \dots$

Suppose the rule $r = u \rightarrow v$ is identified. Then there are pairs in \mathcal{V} such that

$$(S_n, S_{n+1}) = (S'_n u S''_n, S'_{n+1} v S''_{n+1}) \quad (1.5)$$

The rule r is added to \mathcal{V} and all pairs of the form 1.5 are eliminated from \mathcal{S} by replacing (S_n, S_{n+1}) by (S'_n, S'_{n+1}) and (S''_n, S''_{n+1}) in \mathcal{V} . The process of elimination is continued until either the set \mathcal{S} is empty, or the only possible rule to be chosen is improper. In the first case the problem is solved and the rule set \mathcal{V} is found. In the last case the algorithm simply backtracks.

The algorithm consists of two parts that call each other recursively. One part tries to determine a next rule to try, either enumerating all possible rules or using heuristics. The chosen rule is then applied to a pair of strings, and splits them into two pairs. The rule is applied at all possible places, and is eliminated

if any of the resulting pairs is improper. The process is repeated until no more rules can be found. The total number of pairs may vary with every found rule, but the length of pairs in \mathcal{S} is reduced with each step. The total number of rules to be tried is also bounded. Thus the algorithm will finally terminate whether yielding result or not.

All above said works when initial strings do not contain a random component. In the contrary case, the search algorithm needs to be expanded to allow for nonexact matches of substrings. This is done by studying the probability distributions of substrings of approximately the same length, or by introducing the notion of *editing distance*. The probabilistic formulation of the problem goes beyond the goals of the current article and will be published elsewhere.

6 Conclusion

When measuring Chaitin's algorithmic complexity of a recursive structure, one has to choose the language properly reflecting the symmetry of the object, for example by using the extended alphabet rather than the binary one. L-system approach seems very convenient for description of the fractal structures due to its inherent recursive property and adequate expressive power, so the idea of defining algorithmic complexity in terms of length of rewriting rules is being conveyed by this article. When dealing with rewriting systems we may encounter interesting phenomena like convergence of the refinement process.

This paper demonstrates the first steps in studying applications of parallel derivation systems to measure the complexity of recursive objects.

Bibliography

- [1] BOOK, R., and F. OTTO, *String Rewriting Systems*, Springer-Verlag Berlin (1993).
- [2] CHAITIN, Gregory J., *Algorithmic Information Theory*, Cambridge University Press (1987).
- [3] CHAITIN, Gregory J., *Exploring Randomness*, Springer-Verlag (2001).
- [4] GÁCS, Péter, John T. TROMP, and Paul M. B. VITÁNYI, "Algorithmic statistic", *IEEE Transactions on Information Theory* (2001).
- [5] KOLMOGOROV, A. N., "Three approaches to the quantitative definition of information", *Problems Inform. Transmission* **1**, 1 (1965), 1–7.
- [6] KOUPTSOV, K. L., J. H. LOWENSTEIN, and F. VIVALDI, "Quadratic rational rotations of the torus and dual lattice maps", *MPEJ*, 02-148, <http://mpej.unige.ch/cgi-bin/mps?key=02-148>, Genève mirror.
- [7] LINDENMAYER, A., "Mathematical models for cellular interactions in development", *J. Theor. Biol.* **18** (1968), 280–315.

- [8] LOWENSTEIN, J. H., and F. VIVALDI, “Anomalous transport in a model of hamiltonian round-off errors”, *Nonlinearity* **11** (1998), 1321–1350.
- [9] LOWENSTEIN, J. H., and F. VIVALDI, “Embedding dynamics for round-off errors near a periodic orbit”, *Chaos* **10** (2000), 747–755.
- [10] VITÁNYI, Paul M. B., *Lindenmayer Systems: Structure, Language, and Growth Functions*, Mathematisch Centrum Amsterdam (1980).
- [11] VITÁNYI, Paul M. B., “Meaningful information”, *arXiv: cs/0111053* (2001).
- [12] WOLFRAM, S., *A new kind of science*, Wolfram Media, Inc. (2002).