L-system

An **L-system** or **Lindenmayer system** is a parallel rewriting system and a type of formal grammar. An L-system consists of an alphabet of symbols that can be used to make strings, a collection of production rules that expand each symbol into some larger string of symbols, an initial "axiom" string from which to begin construction, and a mechanism for translating the generated strings into geometric structures. L-systems were introduced and developed in 1968 by Aristid Lindenmayer, a Hungarian theoretical biologist and botanist at the University of Utrecht.[1] Lindenmayer used L-systems to describe the behaviour of plant cells and to model the growth processes of plant development. L-systems have also been used to model the morphology of a variety of organisms[2] and can be used to generate self-similar fractals.

L-system trees form realistic models of natural patterns

Origins

As a biologist, Lindenmayer worked with yeast and filamentous fungi and studied the growth patterns of various types of bacteria, such as the cyanobacteria *Anabaena catenula*. Originally, the L-systems were devised to provide a formal description of the development of such simple multicellular organisms, and to illustrate the neighbourhood relationships between plant cells. Later on, this system was extended to describe higher plants and complex branching structures.

'Weeds', generated using an L-system in 3D.

L-system structure

The recursive nature of the L-system rules leads to self-similarity and thereby, fractal-like forms are easy to describe with an L-system. Plant models and natural-looking organic forms are easy to define, as by increasing the recursion level the form slowly 'grows' and becomes more complex. Lindenmayer systems are also popular in the generation of artificial life.

L-system grammars are very similar to the semi-Thue grammar (see Chomsky hierarchy). L-systems are now commonly known as *parametric* L systems, defined as a tuple

G = (*V*, ω, *P*),

where

- **V** (the *alphabet*) is a set of symbols containing both elements that can be replaced (*variables*) and those which cannot be replaced ("constants" or "terminals")
- **ω** (*start*, *axiom* or *initiator*) is a string of symbols from **V** defining the initial state of the system
- **P** is a set of *production rules* or *productions* defining the way variables can be replaced with combinations of constants and other variables. A production consists of two strings, the *predecessor* and the *successor*. For any symbol A which is a member of the set V which does not appear on the left hand side of a production in P, the identity production A → A is assumed; these symbols are called *constants* or *terminals*. (See Law of identity).

The rules of the L-system grammar are applied iteratively starting from the initial state. As many rules as possible are applied simultaneously, per iteration. The fact that each iteration employs as many rules as possible differentiates an L-system from a formal language generated by a formal grammar, which applies only one rule per iteration. If the production rules were to be applied only one at a time, one would quite simply generate a string in a language, and all such sequences of applications would produce the language specified by the grammar. There are some strings in some languages, however, that cannot be generated if the grammar is treated as an L-system rather than a language specification. For example,[3] suppose there is a rule S→SS in a grammar. If productions are done one at a time, then starting from S, we can get first SS, and then, applying the rule again, SSS. However, if all applicable rules are applied at every step, as in an L-system, then we cannot get this sentential form. Instead, the first step would give us SS, but the second would apply the rule twice, giving us SSSS. Thus, the set of strings produced by an L-systems from a given grammar is a subset of the formal language defined by the grammar, and if we take a language to be defined as a set of strings, this means that a given L-system is effectively a subset of the formal language defined by the L-system's grammar.

An L-system is *context-free* if each production rule refers only to an individual symbol and not to its neighbours. Context-free L-systems are thus specified by a context-free grammar. If a rule depends not only on a single symbol but also on its neighbours, it is termed a *context-sensitive* L-system.

If there is exactly one production for each symbol, then the L-system is said to be *deterministic* (a deterministic context-free L-system is popularly called a *D0L system*). If there are several, and each is chosen with a certain probability during each iteration, then it is a *stochastic* L-system.

Using L-systems for generating graphical images requires that the symbols in the model refer to elements of a drawing on the computer screen. For example, the program *Fractint* uses turtle graphics (similar to those in the Logo programming language) to produce screen images. It interprets each constant in an L-system model as a turtle command.

Examples of L-systems

Example 1: algae

Lindenmayer's original L-system for modelling the growth of algae.

variables : A B
constants : none
axiom : A
rules : (A → AB), (B → A)

which produces:

n = 0 : A
n = 1 : AB
n = 2 : ABA
n = 3 : ABAAB
n = 4 : ABAABABA
n = 5 : ABAABABAABAAB
n = 6 : ABAABABAABAABABAABABA
n = 7 : ABAABABAABAABABAABABAABAABABAABAAB

Example 1: algae, explained

```
n=0:               A             start (axiom/initiator)
                  / \
n=1:             A   B           the initial single A spawned into AB by rule (A → AB), rule (B → A)
couldn't be applied
                /|     \
n=2:           A B      A        former string AB with all rules applied, A spawned into AB again,
former B turned into A
             / | |       | \
n=3:         A B A       A B     note all A's producing a copy of themselves in the first place, then
a B, which turns ...
           / | | | \     | \ \
n=4:       A B A A B     A B A   ... into an A one generation later, starting to spawn/repeat/recurse
then
```

The result is the sequence of Fibonacci words. If one counts the length of each string, the Fibonacci sequence of numbers is obtained (skipping the first 1, due to the choice of axiom):

1 2 3 5 8 13 21 34 55 89 ...

If it is not desired to skip the first 1, axiom *B* can be used. That would place a *B* node before the topmost node (*A*) of the graph above.

For each string, if one counts the *k*-th position from the left end of the string, the value is determined by whether a multiple of the golden ratio falls within the interval $(k-1, k)$. The ratio of A to B likewise converges to the golden mean.

This example yields the same result (in terms of the length of each string, not the sequence of *A*s and *B*s) if the rule (*A* → *AB*) is replaced with (*A* → *BA*), except that the strings are mirrored.

This sequence is a locally catenative sequence because $G(n) = G(n-1)G(n-2)$, where $G(n)$ is the *n*-th generation.

Example 2: fractal (binary) tree

- **variables** : 0, 1
- **constants**: "[", "]"
- **axiom** : 0
- **rules** : (1 → 11), (0 → 1[0]0)

The shape is built by recursively feeding the axiom through the production rules. Each character of the input string is checked against the rule list to determine which character or string to replace it with in the output string. In this example, a '1' in the input string becomes '11' in the output string, while '[' remains the same. Applying this to the axiom of '0', one gets:

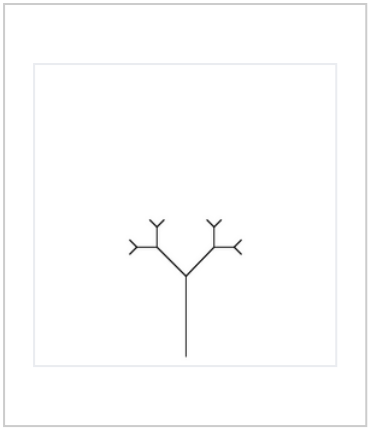
axiom:	0
1st recursion:	1[0]0
2nd recursion:	11[1[0]0]1[0]0
3rd recursion:	1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0
...	

It can be seen that this string quickly grows in size and complexity. This string can be drawn as an image by using turtle graphics, where each symbol is assigned a graphical operation for the turtle to perform. For example, in the sample above, the turtle may be given the following instructions:

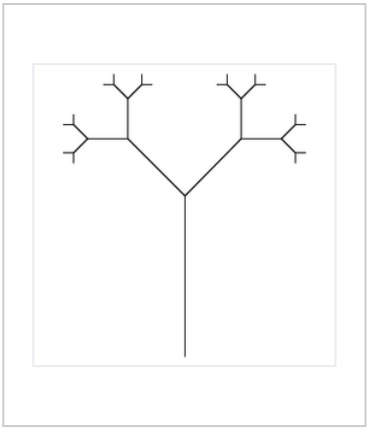
- 0: draw a line segment ending in a leaf
- 1: draw a line segment
- [: push position and angle, turn left 45 degrees
-]: pop position and angle, turn right 45 degrees

The push and pop refer to a LIFO stack (more technical grammar would have separate symbols for "push position" and "turn left"). When the turtle interpretation encounters a '[', the current position and angle are saved, and are then restored when the interpretation encounters a ']'. If multiple values have been "pushed," then a "pop" restores the most recently saved values. Applying the graphical rules listed above to the earlier recursion, one gets:

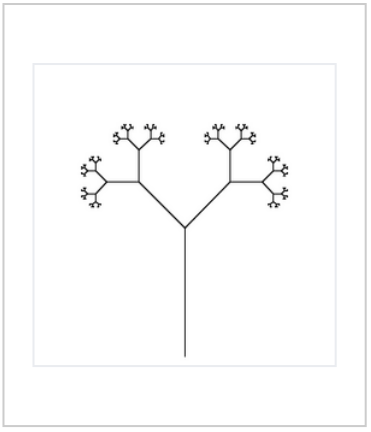
Axiom First recursion Second recursion



Third recursion



Fourth recursion



Seventh recursion, scaled down ten times

Example 3: Cantor set

variables : A B
constants : none
start : A {starting character string}
rules : (A → ABA), (B → BBB)

Let *A* mean "draw forward" and *B* mean "move forward".

This produces the famous Cantor's fractal set on a real straight line **R**.

Example 4: Koch curve

A variant of the Koch curve which uses only right angles.

variables : F
constants : + −
start : F
rules : (F → F+F−F−F+F)

Here, F means "draw forward", + means "turn left 90°", and − means "turn right 90°" (see turtle graphics).

n = 0:

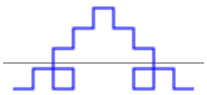
F

n = 1:

F+F−F−F+F

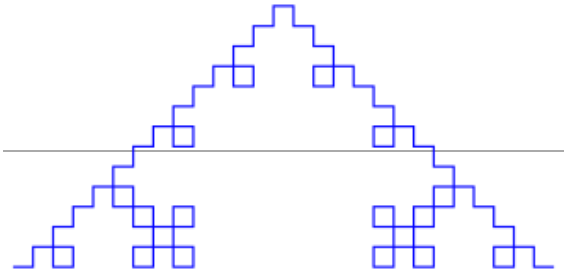
n = 2:

F+F−F−F+F+F+F−F−F+F−F+F−F−F+F−F+F−F−F+F+F+F−F−F+F



n = 3:

F+F−F−F+F+F+F−F−F+F−F+F−F−F+F−F+F−F−F+F+F+F−F−F+F+
F+F−F−F+F+F+F−F−F+F−F+F−F−F+F−F+F−F−F+F+F+F−F−F+F−
F+F−F−F+F+F+F−F−F+F−F+F−F−F+F−F+F−F−F+F+F+F−F−F+F−
F+F−F−F+F+F+F−F−F+F−F+F−F−F+F−F+F−F−F+F+F+F−F−F+F+
F+F−F−F+F+F+F−F−F+F−F+F−F−F+F−F+F−F−F+F+F+F−F−F+F

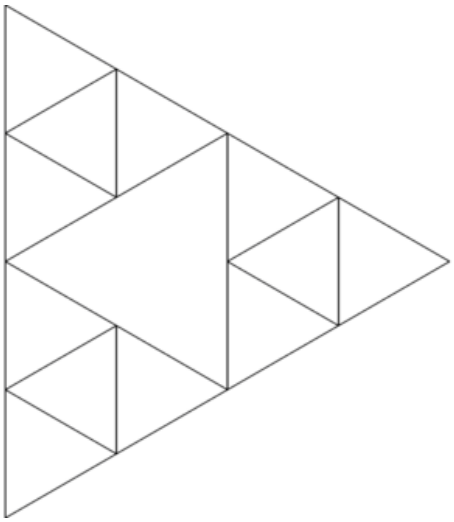


Example 5: Sierpinski triangle

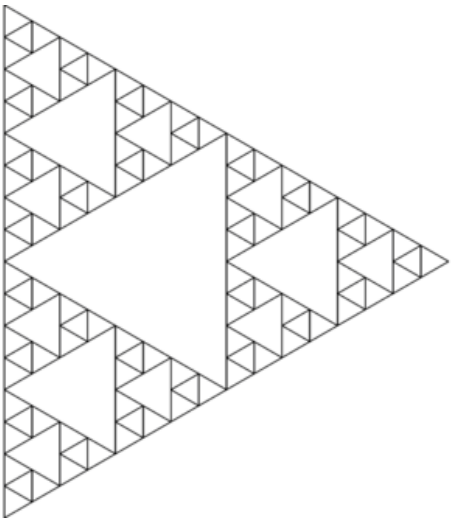
The Sierpinski triangle drawn using an L-system.

variables : F G
constants : + −
start : F−G−G
rules : (F → F−G+F+G−F), (G → GG)
angle : 120°

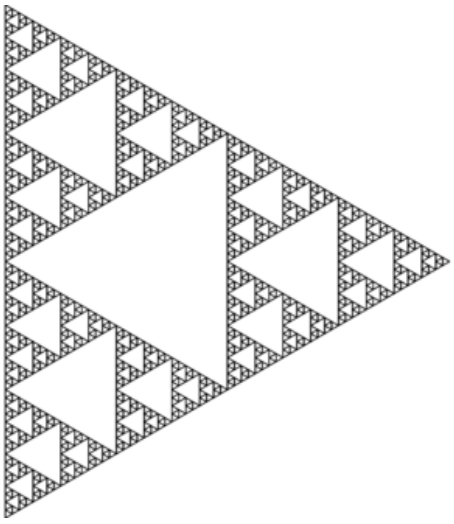
Here, F and G both mean "draw forward", + means "turn left by angle", and − means "turn right by angle".



n = 2



n = 4

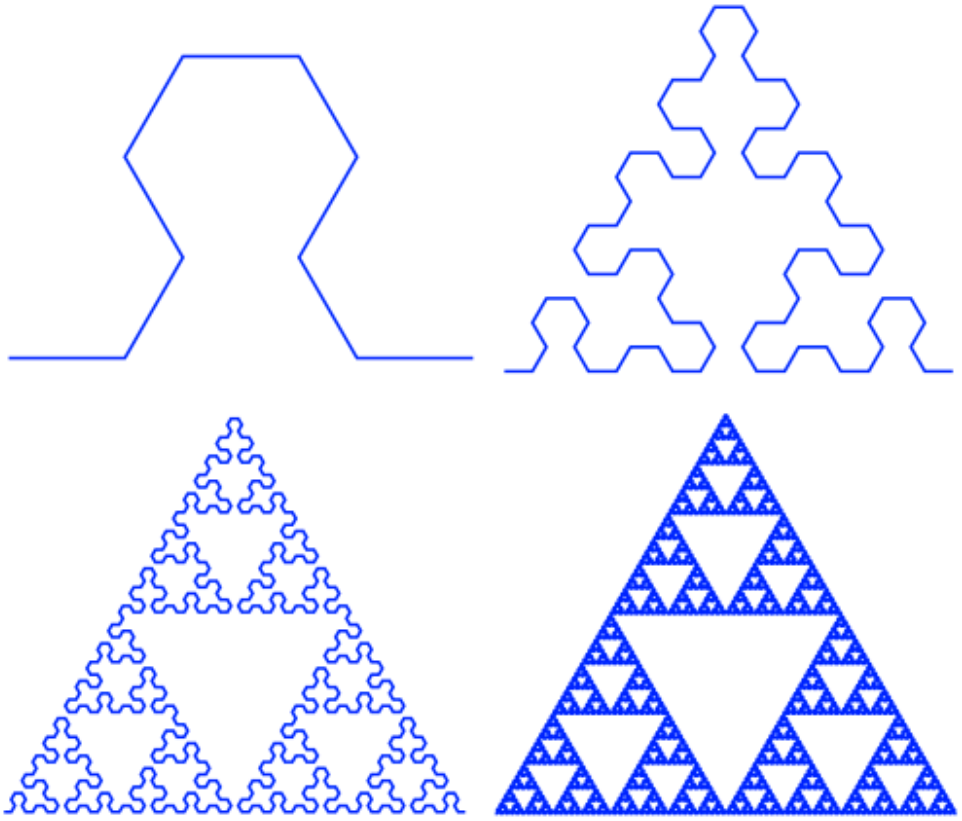


n = 6

It is also possible to approximate the Sierpinski triangle using a Sierpiński arrowhead curve L-system.

variables : A B
constants : + −
start : A
rules : (A → B−A−B), (B → A+B+A)
angle : 60°

Here, A and B both mean "draw forward", + means "turn left by angle", and − means "turn right by angle" (see turtle graphics).



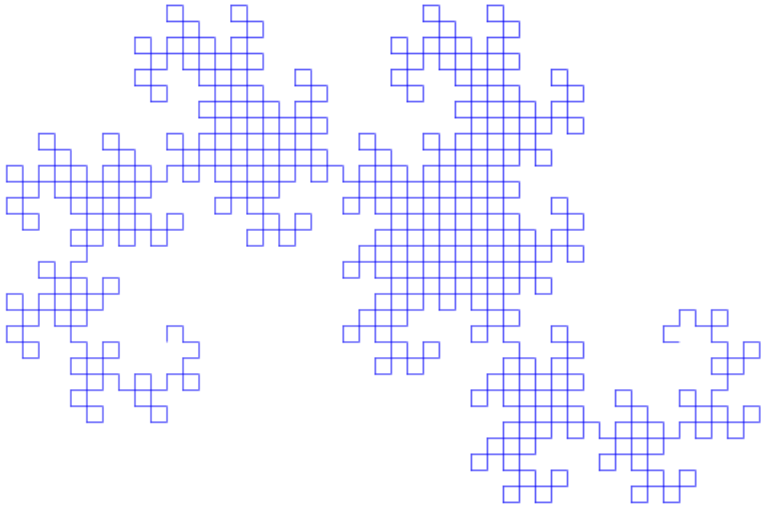
Evolution for $n = 2$, $n = 4$, $n = 6$, $n = 8$

Example 6: dragon curve

The dragon curve drawn using an L-system.

variables : F G
constants : + −
start : F
rules : (F → F+G), (G → F-G)
angle : 90°

Here, F and G both mean "draw forward", + means "turn left by angle", and − means "turn right by angle".



Dragon curve for $n = 10$

Example 7: fractal plant

variables : X F
constants : + − [ ]
start : -X
rules : (X → F+[[X]-X]-F[-FX]+X), (F → FF)
angle : 25°

First one needs to initialize an empty stack. This follows the LIFO (Last in, First Out) method to add and remove elements. Here, F means "draw forward", − means "turn right 25°", and + means "turn left 25°". X does not correspond to any drawing action and is used to control the evolution of the curve. The square bracket "[" corresponds to saving the current values for position and angle, so the position and angle are pushed to the top of the stack, when the "]" token is encountered, the stack is popped and the position and angle are reset. Every "[" comes before every "]" token.

Fractal plant for $n = 6$

Variations

A number of elaborations on this basic L-system technique have been developed which can be used in conjunction with each other. Among these are stochastic grammars, context sensitive grammars, and parametric grammars.

Stochastic grammars

The grammar model we have discussed thus far has been deterministic—that is, given any symbol in the grammar's alphabet, there has been exactly one production rule, which is always chosen, and always performs the same conversion. One alternative is to specify more than one production rule for a symbol, giving each a probability of occurring. For example, in the grammar of Example 2, we could change the rule for rewriting "0" from:

0 → 1[0]0

to a probabilistic rule:

0 (0.5) → 1[0]0
0 (0.5) → 0

Under this production, whenever a "0" is encountered during string rewriting, there would be a 50% chance it would behave as previously described, and a 50% chance it would not change during production. When a stochastic grammar is used in an evolutionary context, it is advisable to incorporate a

random seed into the genotype, so that the stochastic properties of the image remain constant between generations.

Context sensitive grammars

A context sensitive production rule looks not only at the symbol it is modifying, but the symbols on the string appearing before and after it. For instance, the production rule:

b < a > c → aa

transforms "a" to "aa", but only if the "a" occurs between a "b" and a "c" in the input string:

…bac…

As with stochastic productions, there are multiple productions to handle symbols in different contexts. If no production rule can be found for a given context, the identity production is assumed, and the symbol does not change on transformation. If context-sensitive and context-free productions both exist within the same grammar, the context-sensitive production is assumed to take precedence when it is applicable.

Parametric grammars

In a parametric grammar, each symbol in the alphabet has a parameter list associated with it. A symbol coupled with its parameter list is called a module, and a string in a parametric grammar is a series of modules. An example string might be:

a(0,1)[b(0,0)]a(1,2)

The parameters can be used by the drawing functions, and also by the production rules. The production rules can use the parameters in two ways: first, in a conditional statement determining whether the rule will apply, and second, the production rule can modify the actual parameters. For example, look at:

a(x,y) : x == 0 → a(1, y+1)b(2,3)

The module a(x,y) undergoes transformation under this production rule if the conditional x=0 is met. For example, a(0,2) would undergo transformation, and a(1,2) would not.

In the transformation portion of the production rule, the parameters as well as entire modules can be affected. In the above example, the module b(x,y) is added to the string, with initial parameters (2,3). Also, the parameters of the already existing module are transformed. Under the above production rule,

a(0,2)

Becomes

a(1,3)b(2,3)

as the "x" parameter of a(x,y) is explicitly transformed to a "1" and the "y" parameter of a is incremented by one.

Parametric grammars allow line lengths and branching angles to be determined by the grammar, rather than the turtle interpretation methods. Also, if age is given as a parameter for a module, rules can change depending on the age of a plant segment, allowing animations of the entire life-cycle of the tree to be created.

Bi-directional grammars

The bi-directional model explicitly separates the symbolic rewriting system from the shape assignment. For example, the string rewriting process in the Example 2 (Fractal tree) is independent on how graphical operations are assigned to the symbols. In other words, an infinite number of draw methods are applicable to a given rewriting system.

The bi-directional model consists of 1) a forward process constructs the derivation tree with production rules, and 2) a backward process realizes the tree with shapes in a stepwise manner (from leaves to the root). Each inverse-derivation step involves essential geometric-topological reasoning. With this bi-directional framework, design constraints and objectives are encoded in the grammar-shape translation. In architectural design applications, the bi-directional grammar features consistent interior connectivity and a rich spatial hierarchy.[4]

L-system Construction and Inference

Manual L-System Construction

Historically, the construction of L-systems relied heavily on manual efforts by experts,[5][6][7] requiring detailed measurements, domain knowledge, and significant time investment. The process often involved analyzing biological structures and encoding their developmental rules into L-systems, symbol by symbol. This labor-intensive method made creating accurate models for complex processes both tedious and error-prone.

A notable example is Nishida's [7] work on Japanese Cypress trees, where he manually segmented branches from a series of images and identified 42 distinct growth mechanisms to construct a stochastic L-system. Despite the significant effort involved, the resulting system provided only an approximation of the tree's growth, illustrating the challenges of manually encoding such detailed biological processes. This arduous task was described as "tedious and intricate," underscoring the limitations of manual approaches.

The challenges of manual L-system construction are also well-documented in The Algorithmic Beauty of Plants [6] by Przemyslaw Prusinkiewicz and Aristid Lindenmayerd. The book demonstrates how L-systems can elegantly model plant growth and fractal patterns, but the examples often required expert intervention to define the necessary rules.

Manual construction was further constrained by the need for domain-specific expertise, as seen in other applications of L-systems beyond biology, such as architectural design and urban modeling.[8] In these fields, creating an accurate L-system required not only an understanding of the L-system formalism but also extensive knowledge of the domain being modeled.

L-system Inference

The idea of automating L-system inference emerged to address the inefficiencies of manual methods, which often required extensive expertise, measurements, and trial-and-error processes. This automation aimed to enable the inference of L-systems directly from observational data, eliminating the need for manual encoding of rules.

Initial algorithms primarily targeted deterministic context-free L-systems (D0L-systems), which are among the simplest types of L-systems. These early efforts demonstrated the feasibility of automatic inference but were severely limited in scope, typically handling only systems with small alphabets and simple rewriting rules.[9][10][11][12] For instance, Nakano's [10] work highlighted the challenges of inferring L-systems with larger alphabets and more complex structures, describing the task as "immensely complicated".

Manual and Semi-Automated Tools

Early tools for L-system inference were often designed to assist experts rather than replace them. For example, systems that presented a population of potential L-systems to the user, allowing them to select aesthetically pleasing or plausible options, reduced some of the manual burden.[12][13] However, these tools relied heavily on human judgment and did not fully automate the inference process.

Domain-Specific Inference Approaches

Some early algorithms were tightly integrated into specific research domains mainly plant modeling.[13] These approaches utilized domain knowledge to constrain the search space and achieve better results. However, their reliance on predefined domain-specific rules limited their generalizability and applicability to other areas.

Generalized Inference Algorithms

Attempts to create generalized algorithms for L-system inference began with deterministic context-free systems. Researchers aimed to infer L-systems from data alone, such as sequences of strings or temporal data from images, without relying on domain-specific knowledge. These algorithms encountered significant challenges,[14][15] including:

- The exponential growth of the search space with increasing alphabet size and rule complexity.
- Dealing with imperfect or noisy data, which introduced errors in the inferred systems.
- Limitations in computational efficiency, as exhaustive search methods became intractable for all but the simplest cases.

Bernard's PhD dissertation,[16] supervised by Dr. Ian McQuillan at the University of Saskatchewan, represents a significant advancement in L-system inference, introducing the Plant Model Inference Tools (PMIT) suite. Despite the name, this tool is problem agnostic, and is so-named due to the source of the original funding from the P2IRC project. These tools address the challenges of inferring deterministic, stochastic, and parametric L-systems:

Deterministic Context-Free L-Systems (D0L):

The PMIT-D0L tool improved the state-of-the-art by enabling the inference of L-systems with up to 31 symbols, compared to previous algorithms that managed only two. This was achieved through novel encoding techniques and search-space reduction methods.

Deterministic Context-Sensitive L-Systems (D(j,k)L):

The PMIT-DCSL tool further improved the inference of deterministic L-systems by demonstrating that the techniques worked in the context-sensitive case with little modification. This tool also presented further improvements allowing for the inference of deterministic L-systems with up to hundreds of symbols. Furthermore, this work and McQuillan's [17] theoretical paper proves the complexity of context-sensitive L-systems inference. In an unpublished work, Bernard claims to show that context-sensitivity never changes the fundamental nature of the inference problem regardless of the selection rule. That is to say, inferring context-sensitive stochastic L-systems is possible if inferring context-free L-system is possible.

Stochastic L-Systems (S0L):

For stochastic L-systems, PMIT-S0L was developed, which uses a hybrid greedy and genetic algorithm approach to infer systems from multiple string sequences. The tool demonstrated the ability to infer rewriting rules and probabilities with high accuracy, a first in the field.

Temporal Parametric L-Systems:

McQuillan first realized that parametric L-systems could be thought of as stochastic L-systems; however, this did not solve the problem of inferring the parametric selection rules. Using Cartesian Genetic Programming, parametric L-systems could be inferred along with the parametric selection rules so long as the parameter set included time (in order to, provide a sequence to the parameters, but time is a reasonable parameter for any real process). This tool, PMIT-PARAM, successfully inferred complex systems with up to 27 rewriting rules, setting a new benchmark in L-system inference.

Open problems

There are many open problems involving studies of L-systems. For example:

- Characterisation of all the deterministic context-free L-systems which are locally catenative. (A complete solution is known only in the case where there are only two variables).[18]

Types of L-systems

L-systems on the real line **R**:

- Prouhet-Thue-Morse system

Well-known L-systems on a plane $\mathbf{R}^2$ are:

- space-filling curves (Hilbert curve, Peano's curves, Dekking's church, kolams),
- median space-filling curves (Lévy C curve, Harter-Heighway dragon curve, Davis-Knuth terdragon),

- tilings (sphinx tiling, Penrose tiling)

See also

- Digital morphogenesis
- Iterated function system
- Reaction–diffusion system – Type of mathematical model that provides diffusing-chemical-reagent simulations (including Life-like)
- Stochastic context-free grammar
- *The Algorithmic Beauty of Plants*

Notes

1. Lindenmayer, Aristid (March 1968). "Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs". *Journal of Theoretical Biology*. **18** (3): 300–315. Bibcode:1968JThBi..18..300L (https://ui.adsabs.harvard.edu/abs/1968JThBi..18..300L). doi:10.1016/0022-5193(68)90080-5 (https://doi.org/10.1016%2F0022-5193%2868%2990080-5). ISSN 0022-5193 (https://search.worldcat.org/issn/0022-5193). PMID 5659072 (https://pubmed.ncbi.nlm.nih.gov/5659072).
2. Grzegorz Rozenberg and Arto Salomaa. The mathematical theory of L systems (Academic Press, New York, 1980). ISBN 0-12-597140-0
3. "L-systems" (https://encyclopediaofmath.org/index.php?title=L-systems&oldid=47547). *Encyclopedia of Mathematics*. Springer. Retrieved 26 July 2022.
4. Hua, H., 2017, December. A Bi-Directional Procedural Model for Architectural Design (https://www.researchgate.net/profile/Hao_Hua/publication/311357805_A_Bi-Directional_Procedural_Model_for_Architectural_Design/links/5a29fb8baca2728e05dafa4b/A-Bi-Directional-Procedural-Model-for-Architectural-Design.pdf). In Computer Graphics Forum (Vol. 36, No. 8, pp. 219-231).
5. Dinnus Frijters and Aristid Lindenmayer. A model for the growth and owering of Aster novae-angliae on the basis of table < 10 > L-systems. In L systems, pages 2452. Springer, 1974.
6. Prusinkiewicz, P., & Lindenmayer, A. (2012). *The algorithmic beauty of plants*. Springer Science & Business Media.
7. T. Nishida, K0L-system simulating almost but not exactly the same development-case of Japanese Cypress, Memoirs of the Faculty of Science, Kyoto University, Series B 8 (1) (1980) 97122.
8. Pascal Muller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. ACM Transactions On Graphics, 25(3):614623, 2006.
9. Bian Runqiang, Phoebe Chen, Kevin Burrage, Jim Hanan, Peter Room, and John Belward. Derivation of L-system models from measurements of biological branching structures using genetic algorithms. In Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pages 514524. Springer, 2002.
10. Ryohei Nakano. Emergent induction of deterministic context-free L-system grammar. In Innovations in Bio-inspired Computing and Applications, pages 7584. Springer International Publishing, 2014.
11. P. G. Doucet. The syntactic inference problem for D0L-sequences. L Systems, pages 146161, 1974
12. Roger Curry. On the evolution of parametric L-systems. Technical report, University of Calgary, 2000.

13. Fabricio Anastacio, Przemyslaw Prusinkiewicz, and Mario Costa Sousa. Sketch-based parameterization of L-systems using illustration-inspired construction lines and depth modulation. Computers & Graphics, 33(4):440451, 2009.
14. Colin De La Higuera. A bibliographical study of grammatical inference. Pattern Recognition, 38(9):1332 1348, 2005.
15. Kari, L., Rozenberg, G., & Salomaa, A. (1997). *L systems* (pp. 253-328). Springer Berlin Heidelberg.
16. Bernard, J. (2020). *Inferring Different Types of Lindenmayer Systems Using Artificial Intelligence* (Doctoral dissertation, University of Saskatchewan).
17. McQuillan, I., Bernard, J., & Prusinkiewicz, P. (2018). Algorithms for inferring context-sensitive L-systems. In *Unconventional Computation and Natural Computation: 17th International Conference, UCNC 2018, Fontainebleau, France, June 25-29, 2018, Proceedings 17* (pp. 117-130). Springer International Publishing.
18. Kari, Lila; Rozenberg, Grzegorz; Salomaa, Arto (1997). "L Systems". *Handbook of Formal Languages*. pp. 253–328. doi:10.1007/978-3-642-59136-5_5 (https://doi.org/10.1007%2F978-3-642-59136-5_5). ISBN 978-3-642-63863-3.

Books

- Przemysław Prusinkiewicz, Aristid Lindenmayer – The Algorithmic Beauty of Plants PDF version available here for free (https://algorithmicbotany.org/papers/#abop) Archived (https://web.archive.org/web/20210410150043/https://algorithmicbotany.org/papers/#abop) 2021-04-10 at the Wayback Machine
- Grzegorz Rozenberg, Arto Salomaa – *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology* ISBN 978-3-540-55320-5
- D.S. Ebert, F.K. Musgrave, et al. – *Texturing and Modeling: A Procedural Approach*, ISBN 0-12-228730-4
- Burry, Jane, Burry Mark, (2010). The New Mathematics of Architecture, New York: Thames and Hudson.
- Aristid Lindenmayer, "Mathematical models for cellular interaction in development (https://www0.cs.ucl.ac.uk/staff/p.bentley/teaching/L6_reading/lsystems.pdf)." J. Theoret. Biology, 18:280—315, 1968.

External links

- Algorithmic Botany at the University of Calgary (https://algorithmicbotany.org/)
- L-Systems (https://rue-a.github.io/L-Systems/): A user friendly page to generate fractals and plants from L-Systems.
- Branching: L-system Tree (https://www.mizuno.org/applet/branching/) A Java applet and its source code (open source) of the botanical tree growth simulation using the L-system.
- Fractint L-System True Fractals (https://web.archive.org/web/20020503212834/https://spanky.triumf.ca/WWW/FRACTINT/lsys/truefractal.html)
- OpenAlea (https://openalea.gforge.inria.fr/) Archived (https://web.archive.org/web/20051017111619/https://openalea.gforge.inria.fr/) 2005-10-17 at the Wayback Machine: an open-source software environment for plant modeling,[1] which contains L-Py (https://github.com/openalea/lpy), an open-source python implementation of the Lindenmayer systems[2]
- "powerPlant" an open-source landscape modelling software (https://sourceforge.net/projects/pplant/)

- An evolutionary L-systems generator (anyos*) (https://www.cs.ucl.ac.uk/staff/W.Langdon/pfeiffer.html)
- An implementation of L-systems in Racket (https://github.com/rfindler/lindenmayer/)
- Griffiths, Dave (2004). "LsystemComposition" (https://www.pawfal.org/index.php?page=LsystemComposition). *Pawfal*. Archived (https://web.archive.org/web/20041106154657/https://www.pawfal.org/index.php?page=LsystemComposition) from the original on 2004-11-06. Retrieved 2012-04-19. Page about using L-systems and genetic algorithms to generate music.
- eXtended L-Systems (XL), Relational Growth Grammars, and open-source software platform GroIMP. (https://www.grogra.de/)
- A JAVA applet with many fractal figures generated by L-systems. (https://to-campos.planetaclix.pt/fractal/plantae.htm) Archived (https://web.archive.org/web/20160806081808/https://to-campos.planetaclix.pt/fractal/plantae.htm) 2016-08-06 at the Wayback Machine
- Manousakis, Stelios (June 2006). *Musical L-Systems* (https://www.modularbrains.net/support/SteliosManousakis-Musical_L-systems.pdf) (PDF) (Master's thesis). Royal Conservatory of The Hague. Archived (https://web.archive.org/web/20110723211305/http://www.modularbrains.net/support/SteliosManousakis-Musical_L-systems.pdf) (PDF) from the original on 2011-07-23. Retrieved 2022-07-19.
- Online experiments with L-Systems using JSXGraph (JavaScript) (https://jsxgraph.uni-bayreuth.de/wiki/index.php/L-systems)
- Flea (https://flea.sourceforge.net/) A Ruby implementation of LSYSTEM, using a Domain Specific Language instead of terse generator commands
- Lindenmayer power (https://madflame991.blogspot.com/p/lindenmayer-power.html) A plant and fractal generator using L-systems (JavaScript)
- Rozenberg, G.; Salomaa, A. (2001) [1994], "L-systems" (https://www.encyclopediaofmath.org/index.php?title=L-systems), *Encyclopedia of Mathematics*, EMS Press
- Laurens Lapré's L-Parser (https://laurenslapre.nl/lapre_004.htm) Archived (https://web.archive.org/web/20130913220627/https://laurenslapre.nl/lapre_004.htm) 2013-09-13 at the Wayback Machine
- HTML5 L-Systems – try out experiments online (https://www.kevs3d.co.uk/dev/lsystems/)
- The vector-graphics program (https://inkscape.org/) Inkscape features an L-System Parser
- Liou, Cheng-Yuan; Wu, Tai-Hei; Lee, Chia-Ying (2009). "Modeling complexity in musical rhythm". *Complexity*. **15** (4): 19–30. doi:10.1002/cplx.20291 (https://doi.org/10.1002%2Fcplx.20291). S2CID 18737938 (https://api.semanticscholar.org/CorpusID:18737938).
- An implementation of a L-system parser and simple turtle graphics in the Icon programming language (https://xojoc.pw/dailyprogrammer/fractals.html)
- A Lindenmeyer System Generator by Nolan Carroll (https://nolandc.com/sandbox/fractals/)
- Bloogen: L-Systems with a genetic twist (https://bloogen.com/)
- Inferring Different Types of Lindenmayer Systems Using Artificial Intelligence (https://harvest.usask.ca/bitstream/10388/13620/8/BERNARD-DISSERTATION-2020.pdf)

1. Pradal, Christophe; Fournier, Christian; Valduriez, Patrick; Cohen-Boulakia, Sarah (2015). "OpenAlea". *Proceedings of the 27th International Conference on Scientific and Statistical Database Management* (https://hal.archives-ouvertes.fr/hal-01166298/file/openalea-PradalCohen-Boulakia.pdf) (PDF). pp. 1–6. doi:10.1145/2791347.2791365 (https://doi.org/10.1145%2F2791347.2791365). ISBN 9781450337090. S2CID 14246115 (https://api.semanticscholar.org/CorpusID:14246115). Archived (https://web.archive.org/web/20191017173603/https://hal.archives-ouvertes.fr/hal-01166298/file/openalea-PradalCohen-Boulakia.pdf) (PDF) from the original on 2019-10-17.
2. Boudon, Frédéric; Pradal, Christophe; Cokelaer, Thomas; Prusinkiewicz, Przemyslaw; Godin, Christophe (2012). "L-Py: An L-System Simulation Framework for Modeling Plant

Architecture Development Based on a Dynamic Language" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3362793). *Frontiers in Plant Science*. **3**: 76. Bibcode:2012FrPS....3...76B (https://ui.adsabs.harvard.edu/abs/2012FrPS....3...76B). doi:10.3389/fpls.2012.00076 (https://doi.org/10.3389%2Ffpls.2012.00076). PMC 3362793 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3362793). PMID 22670147 (https://pubmed.ncbi.nlm.nih.gov/22670147).
