

Evolution Programs Evolved

Christian Jacob

University of Erlangen-Nürnberg, Lehrstuhl für Programmiersprachen,
Martensstr. 3, D-91058 ERLANGEN, Germany
email: jacob@informatik.uni-erlangen.de
<http://www2.informatik.uni-erlangen.de/IMMD-II/Persons/jacob>

Abstract. Growth grammars in the form of parallel rewrite systems (L-systems) are used to model morphogenetic processes of plant structures. With the help of evolutionary programming techniques developmental programs are bred which encode plants that exhibit characteristic growth patterns advantageous in competitive environments.

Program evolution is demonstrated on the basis of extended genetic programming on symbolic expressions with genetic operators and expression generation strongly relying on templates and pattern matching.

1 Introduction: Why model morphogenesis?

One reason for attempting to model nature is to gain more and more detailed insight into natural processes. Computer simulations have turned out to provide excellent means to explore phenomena observed in nature. One crucial characteristic of natural organisms is their ability to grow and form new structures, these processes being subsumed as *structure formation* or *morphogenesis*. Such structure formation can be interpreted as the execution of »developmental programs«, there is no blueprint for an organism, instead, complex »rule systems« encode how to build organs and how to combine diversified parts to form a complete and functioning organism. However, these »programs« are highly complex and parametrized with diverse influence from organism-internal signals (genes mutually switching on and off by activator and repressor mechanisms) as well as from the environment in which development takes place.

There is a special area of morphogenesis which is being studied extensively: the morphological modeling of plant growth in 3-dimensional space, an area that turns out to me of more and more importance for realistic simulations of botanical ecosystems, such as forests, thus gaining new insight into natural interaction processes and understanding how environmental factors influence those ecosystems. The creation of plant models in 3D space is required for being able to model interfaces between a plant and its environment, like light interception, its mechanic and hydraulic architecture, or interaction with neighboring plants.

In the sequel, we will focus on modeling growth processes of plant structures in

*H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.)
PPSN-IV, Parallel Problem Solving from Nature IV (1996),
Berlin, Germany.
Springer-Verlag, Berlin, Germany.
pp. 42-51.*

three dimensions. Chapter 2 will briefly mention different approaches used for modeling plant morphology. We will discuss parallel rewrite systems, commonly termed L-systems, in more detail, as these will serve as our preferred formalization of developmental programs. In chapter 3, combining those L-systems with evolutionary programming techniques will lead to a first step in being able to model evolution in ecosystems, or at least, to help design L-systems encoding plants that exhibit characteristic growth patterns, as will be shown in detail by several example evolution experiments in chapter 4. Finally, in chapter 5, we will conclude with an outlook towards a simulation system for coevolutionary plant development.

2 Morphological modeling of plant growth

Quite a lot of diverse approaches for modeling plant morphology have been developed within the last decade (with some being extensions of techniques dating back more than 20 years): iterated function systems [Peitgen et al., 1993], cellular automata or voxel space growth [Green, 1989], Lindenmayer systems [Lindenmayer, 1975; Prusinkiewicz and Lindenmayer, 1990], or stochastic growth grammars [Kurth, 1994].

In the scope of this article we focus on Lindenmayer systems (L-systems), a special type of string based rewrite systems, named after the biologist Aristid Lindenmayer (1925-1989). L-systems are successfully being used in theoretical biology for describing and simulating all different kinds of natural growth processes (for a great number of examples see e.g. Prusinkiewicz and Lindenmayer, 1990). With L-systems all letters in a given word are replaced in parallel and simultaneously. This feature makes Lindenmayer systems especially suitable for describing fractal structures, cell divisions in multicellular organisms [Jacob, 1995b], or flowering stages of herbaceous plants [Prusinkiewicz and Lindenmayer, 1990, Jacob, 1995a and 1996], as we will demonstrate in the sequel.

D0L-systems (D0 meaning: deterministic with no context) are the simplest type of L-systems. Formally a D0L-system $L = (\Sigma, \alpha, P, T)$, capable of encoding geometrical structures, consists of the following ingredients:

- an *alphabet* $\Sigma = \{\sigma_1, \dots, \sigma_n\}$, each symbol of which stands for a morphological unit, like a cell, an internode, a sprout, or a leaf,
- a start string α , referred to as the *axiom*, which is an element of Σ^* , the set of all finite words over the alphabet Σ ,
- $P = \{p_1, \dots, p_k\}$, a set of productions or *rewrite rules* $p_i: \Sigma \rightarrow \Sigma^*$ with $\sigma \rightarrow p_i(\sigma)$ for each $\sigma \in \Sigma$, which replaces a symbol by a (possibly empty) string of symbols, and which are to be applied in parallel to all symbols of a string,
- a *geometrical interpretation* T , a 3D semantics, for some of the symbols from Σ , translating a string into a spatial structure, i.e. special symbols represent commands to draw graphic objects like points, lines, polygons etc; this translation is commonly known as *turtle geometry* interpretation.

H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.)
PPSN-IV, Parallel Problem Solving from Nature IV (1996),
Berlin, Germany.
Springer-Verlag, Berlin, Germany.
pp. 42-51.

Whenever there is no explicit rewriting rule for a symbol σ , the identity mapping is assumed. Iterated application of an L-system $L = (\Sigma, \alpha, P, T)$ generates a (potentially infinite) sequence of words $\alpha^{(0)}\alpha^{(1)}\alpha^{(2)}\dots$ as exemplified in fig 1, where P^k denotes k -fold parallel application of the productions in P , and each string $\alpha^{(i+1)}$ is generated from the preceding string $\alpha^{(i)} = \alpha_1^{(i)}\alpha_2^{(i)}\dots\alpha_{n_i}^{(i)}$ with $\alpha_m^{(i)} \in \Sigma$ the following way ($p_j \in P$):

$$\alpha^{(i+1)} = P(\alpha^{(i)}) = p_{i_1}(\alpha_1^{(i)})p_{i_2}(\alpha_2^{(i)})\dots p_{i_{n_i}}(\alpha_{n_i}^{(i)}) .$$

Fig. 2 shows an example L-system describing growth sequences of sprouts, leaves and blooms of an artificial flower, together with its graphical interpretation. The DOL-system encodes turtle geometry macros for generating graphical representations of the leaves, blooms and stalks. All the non-italic, bold terms (**f**, **pu**, **pd**, **rl**, **rr**, **yl**, **yr**) represent commands to move the turtle (**f**: forward, **b**: backward) and change the drawing tool's orientation by rotation around its longitudinal, lateral, and vertical axes (**rl/rr**: roll left/right, **pu/pd**: pitch up/down, **yl/yr**: yaw left/right), thus translating a one-dimensional string into a 3D geometrical object resembling a plant (some of these commands do not occur in the example L-system). In order to be able to generate branching structures a kind of stacking mechanism for the turtle's position and orientation is necessary. For each string of the form $s_1[s_2]s_3$ the strings s_1 , s_2 , and s_3 are interpreted in sequence, however, before starting the interpretation of s_2 the current turtle position and orientation are pushed on a stack, so that, having finished interpreting s_2 , the turtle is reset to its prior coordinates and orientation.

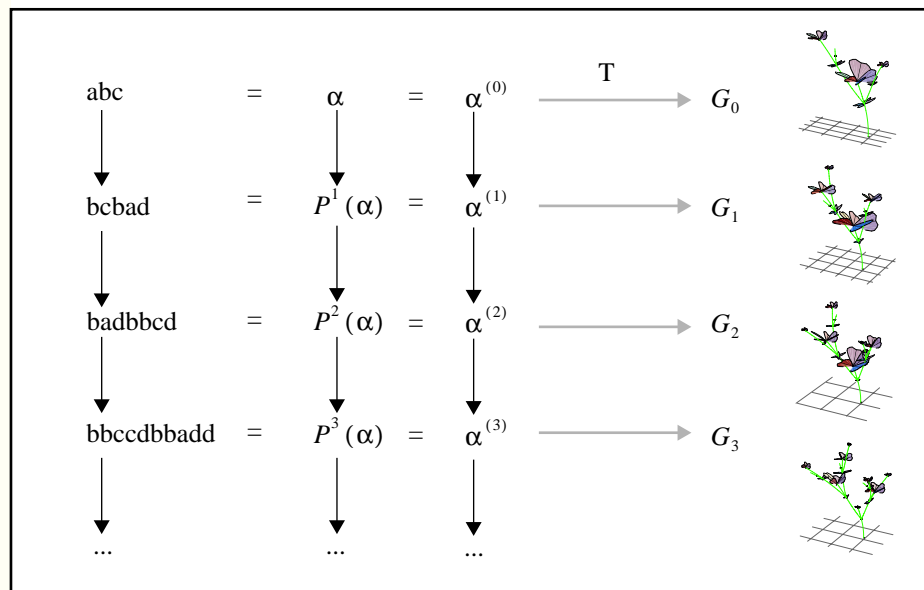


Fig. 1. Rewriting with DOL-systems and geometrical interpretation for an axiom abc and productions $a \rightarrow bc$ and $c \rightarrow ad$.

*H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.)
PPSN-IV, Parallel Problem Solving from Nature IV (1996),
Berlin, Germany.
Springer-Verlag, Berlin, Germany.
pp. 42-51.*

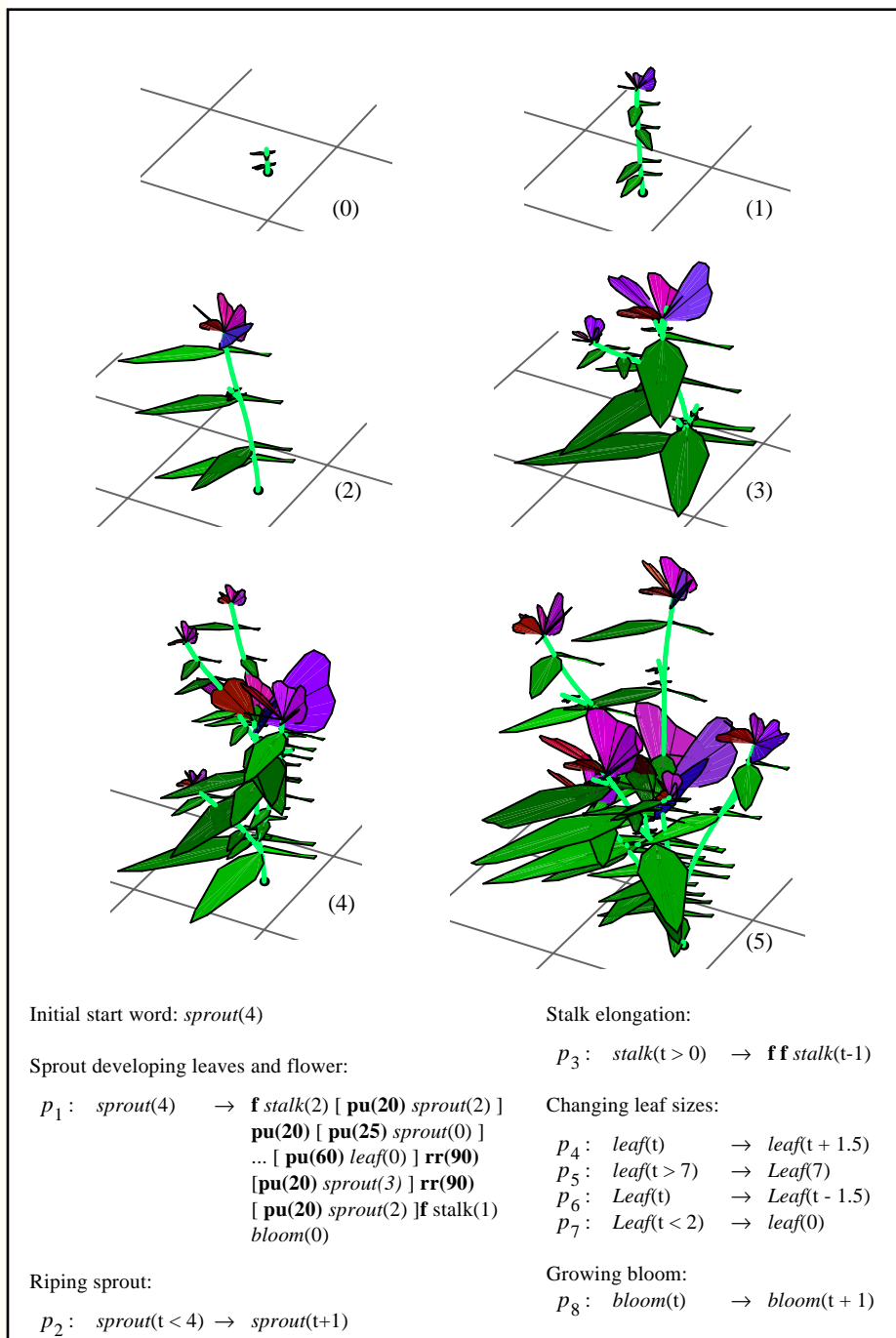


Fig. 2. Example of a growth grammar (L-system) modeling plantlike geometrical structures

H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.)
 PPSN-IV, Parallel Problem Solving from Nature IV (1996),
 Berlin, Germany.
 Springer-Verlag, Berlin, Germany.
 pp. 42-51.

3 Growth grammars and evolution

As we have seen that L-systems can be used to model »developmental programs« we will now try to simulate evolution within populations of plant species. Evolution within ecosystems means that diverse »genetic programs« struggle for survival, or for their ability to best cope with environmental conditions. In this section, we describe the basic ideas how to use evolutionary programming techniques to evolve L-systems encoding plants with characteristic growth structures influencing their light-gathering capability or reproductive potential. The described ideas are only the prerequisites for a much more complex coevolutionary simulation system, which is currently being implemented but could not yet be included in this article.

3.1 Encoding context-sensitive L-systems by symbolic expressions

Genetic Programming (GP) has been introduced as a method to automatically develop populations of computer programs, encoded by symbolic expressions, through simulated evolution [Koza, 1992 and 1994]. In order to use expression evolution for L-systems a proper encoding scheme has to be defined. We will consider the more general case of context-sensitive IL-systems (with $I = 0, 1, 2, \dots$ referring to the number of context symbols). In context-sensitive IL-systems – more precisely: (m,n) L-systems – the rewriting of a letter depends on m of its left and n of its right neighbors, where m and n are fixed integers. These systems are denoted as (m,n) L-systems which resemble context-sensitive Chomsky grammars, but – as L-system rewriting is parallel in nature – every symbol is rewritten in each derivation step; this is especially important whenever there is an overlap of context strings. Each IL-system rule has the general form

$$\text{left context} < \text{predecessor} > \text{right context} \rightarrow \text{successor}.$$

This means that the *predecessor* symbol, whenever occurring within the *left/right context* symbols, is replaced by the *successor* symbol. Thus each rule can be represented by a symbolic expression of the form

```
LRule[ LEFT[ leftContext ], PRED[ predecessor ]
      RIGHT[ rightContext ], SUCC[ successor ] ].
```

Accordingly, an L-system with its axiom and rule set is encoded by an expression of the form

```
LSystem[ □ AXIOM, LRULES[ □□ LRule ] ],
```

where we use a pattern notation with $\square F$ denoting a term of the form $F[\dots]$, and $\square\square F$ representing a non-empty sequence of $\square F$ expressions. So our example L-system of the previous section would be represented as follows:

```
LSystem[ AXIOM[ sprout[ 4 ] ],
  LRULES[
    LRule[ LEFT[ ], PRED[ sprout[ 4 ] ], RIGHT[ ],
          SUCC[ f, stalk[ 2 ], STACK[ pu[ 20 ], ... ], bloom[ 0 ] ] ],
    LRule[ LEFT[ ], PRED[ sprout[ t < 4 ] ], RIGHT[ ], SUCC[ sprout[ t + 1 ] ] ],
    LRule[ LEFT[ ], PRED[ stalk[ t > 0 ] ], RIGHT[ ], SUCC[ f, f, stalk[ t - 1 ] ] ],
```

where L-system bracketing of the form $[s]$ is now represented as `STACK[s]`.

H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.)
PPSN-IV, Parallel Problem Solving from Nature IV (1996),
Berlin, Germany.
Springer-Verlag, Berlin, Germany.
pp. 42-51.

3.2 Stochastic generation of L-system encodings by templates

One of the main differences of our GP approach to the GP paradigm introduced by Koza (1992) is the use of high-level building blocks for generating expression as well as modifying expressions. Instead of just defining a set of function symbols together with their arities, each expression from a template pool, as depicted in figure 2, serves as a possibly partial description of a genotype, encoding an L-system in our case. Each of these templates (marked with 1., 2., 3., ...) is associated with a set of attributes like, e.g., predicates constraining the set of subexpressions that can be plugged in. Thus the encoded L-system productions are restricted to contextfree forms (4., LEFT[], RIGHT[]) with their left-hand sides (4., □ PRED) constrained to *sprout*[*i*], with *i* replaced by an integer number from the interval, say, [0,4] (5.), and their right-hand sides (4., □ SUCC) defined either as a sequence (SEQ) of expressions (6., 7. and 8.) or a bracketed expression sequence (6., □ STACK) for which the productions are omitted here. Each expression is constructed from a start pattern (here: LSystem[□ AXIOM,□ LRULES]) by recursively inserting matching expressions from the expression pool until all pattern blanks – marked by □ – have been replaced by according subexpressions. Of course, one has to take care that this construction loop eventually comes to an end.

```

1.  LSystem[□ AXIOM,□ LRULES],                                     (1)
2.  AXIOM[sprout[4]],                                             (1)
3.  LRULES[
      LRule[LEFT[], PRED[ sprout[4] ], RIGHT[],
            SUCC[SEQ[SEQ[f],SEQ[stalk[2]], STACK[PD[60],leaf[0]], ...],
            □□ SEQ,
            SEQ[f],SEQ[stalk[1]], bloom[0]]],
      LRule[LEFT[], PRED[ sprout[t<4] ], RIGHT[], SUCC[sprout[t+1]],
      ...
      LRule[LEFT[], PRED[ bloom[6] ], RIGHT[], SUCC[ bloom[1]],
      □□□ LRule
    ],
    ],
4.  LRule[LEFT[],□ PRED,RIGHT[],□ SUCC],                         (1)
5.  PRED[sprout[i]],                                             (1)
6.  SUCC[□ SEQ | □ STACK],                                       (1)
7.  SEQ[□□ {□ sprout | □ stalk | □ leaf | □ bloom | f |
        □ YL | □ YR | □ PU | □ PD | □ RL | □ RR | □ SEQ}],      (1)
8.  SEQ[□□ {□ sprout | □ stalk | □ leaf | □ bloom | f |
        □ YL | □ YR | □ PU | □ PD | □ RL | □ RR }],            (4)
    ...
    sprout[sproutIndex], ..., bloom[bloomIndex]                 (1,...,3)

```

Fig. 3. A typical set of templates used for generating L-system encodings

Some further remarks should be made about the encoding scheme of figure 3. The templates basically describe a DOL-system as exemplified in figure 2. However, the $\square\square$ SEQ pattern within the first LRule expression (3.) enables the generation system to create rule variants by inserting new expression sequences, i.e. one or more expressions of the form SEQ[...]. Accordingly, the L-system encoding might be enhanced by additional rules by replacing the $\square\square\square$ LRule pattern with a (possibly empty) sequence of LRule[...] expressions. Each sequence of expressions is constructable via alternative templates (7. and 8.), where SEQ[$\square\square$ [a|b|c]] denotes an arbitrary sequence of expressions taken from the set {a,b,c} as, e.g., SEQ[a,a,c,b] or SEQ[c,c,c,a,b,a] with the sequence length restricted to, say, between 4 and 6.

Whenever there are several templates matching one pattern the expressions are selected with probabilities proportional to their weights (see the bracketed numbers right from the expressions in figure 2, serving as fitness values for the templates) leading to a kind of fitness proportionate selection among templates. The pattern matching is not unique for the \square SEQ pattern (6.) for which there are two matching templates (7. and 8.), a recursive (SEQ[... \square SEQ]) and non-recursive version which will be selected with a probability of 0.8, thus implicitly restricting the nesting of expressions. For further details on expression templates see [Jacob, 1994 and 1996].

3.3 Changing evolution program genes

Principally, we want to be able to change any subexpression within the developmental programs which should not be considered as parametrized modules encoding some fixed L-system in a black box fashion [Niklas, 1986]. As with the expression generating operators the basic idea is, again, to use templates for controlling which subexpressions should be in the scope of each operator:

- *Mutation* replaces a randomly selected subexpression by an expression – with the same head expression – generated from the template pool.
- *Crossover* is a recombination operator between two or more expressions. Subexpressions with the same head are selected within the expressions and interchanged.
- *Deletion* erases expression arguments whenever this is possible according to restrictions of the number of arguments for the selected expression.
- *Duplication* inserts a copy of a randomly selected argument as an additional subexpression.
- *Permutation* interchanges the sequence of arguments of an expression.

These operators are defined for general expressions and are not especially tailored to L-system encodings. Subexpressions are chosen according to operator specific selection schemes based on pattern matching mechanisms. Thus it is, e.g., possible to restrict recombinations to subexpressions with head SUCC, i.e. expressions of the form SUCC[...], which allows to interchange right-hand sides of L-system productions, or permute only LRule expressions thus changing the ordering of rules. With a set of templates defined for each operator the effects of genetic operators can be constrained in a problem specific way. A more detailed description of a general scheme for the application of genetic operators on expressions can be found in [Jacob, 1996].

*H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.)
PPSN-IV, Parallel Problem Solving from Nature IV (1996),
Berlin, Germany.
Springer-Verlag, Berlin, Germany.
pp. 42-51.*

4 Breeding evolution programs

In order to demonstrate by a simple example how plantlike structures with specific characteristics could be developed by L-system evolution, we will consider the following influential factors important in ecosystem competition: a plant's insemination and light gathering capability, and its ability to shed shadow on neighboring plants. We try to incorporate these factors with the following fitness function.

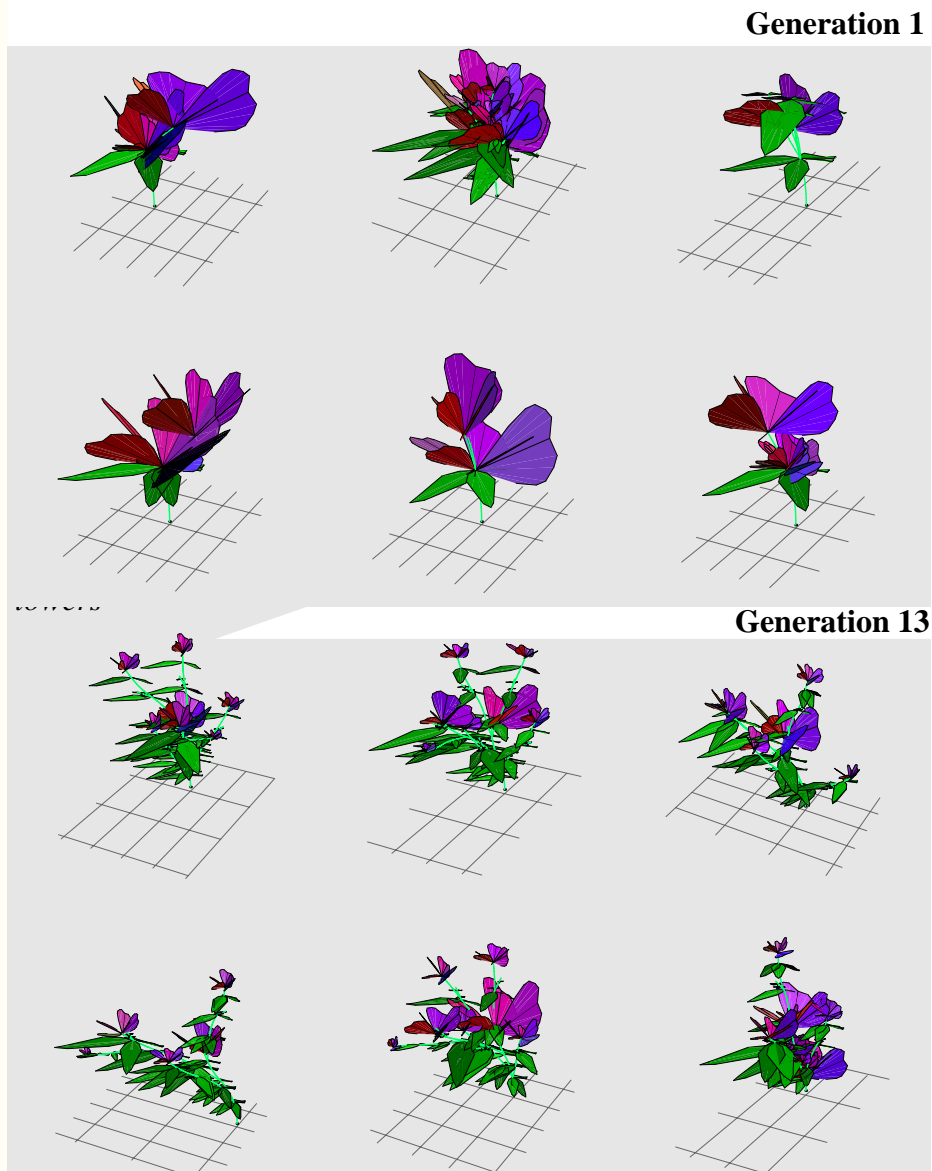


Fig. 4. Snapshots of the phenotypes from an evolution of plant encoding development programs. The plants are depicted with different scales in order to show most of their details

PPSN-IV, *Parallel Problem Solving from Nature IV* (1996),
Berlin, Germany.
Springer-Verlag, Berlin, Germany.
pp. 42-51.

$$fitness(plant) = \sum_{i=0}^m \Delta x_i \cdot \Delta y_i \cdot \Delta z_i + 2 \cdot B_i + L_i$$

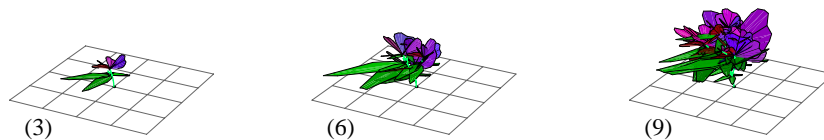
Δx , Δy , Δz : extension of the geometric plant structure in x-, y-, and z-dimension

B , : the number of blooms blossoming; L : the number of leaves the plant carries

m : the maximum number of L-system iterations

The total fitness is computed as the sum of the plant's extension, blooms, and leaves for each L-system iteration. This means that fitness does not only depend on how the plant flourishes at a certain growth stage but on the whole structure generation process from an initial sprout to a full-grown plant. In the following example we start with a small population of six L-system genomes each generated with the templates of figure 3. A maximum number of $m = 6$ L-system iterations is used. Figure 4 shows two stages of a typical evolution sequence. The individuals are selected proportional to their fitness.

Generation 1



Generation 13

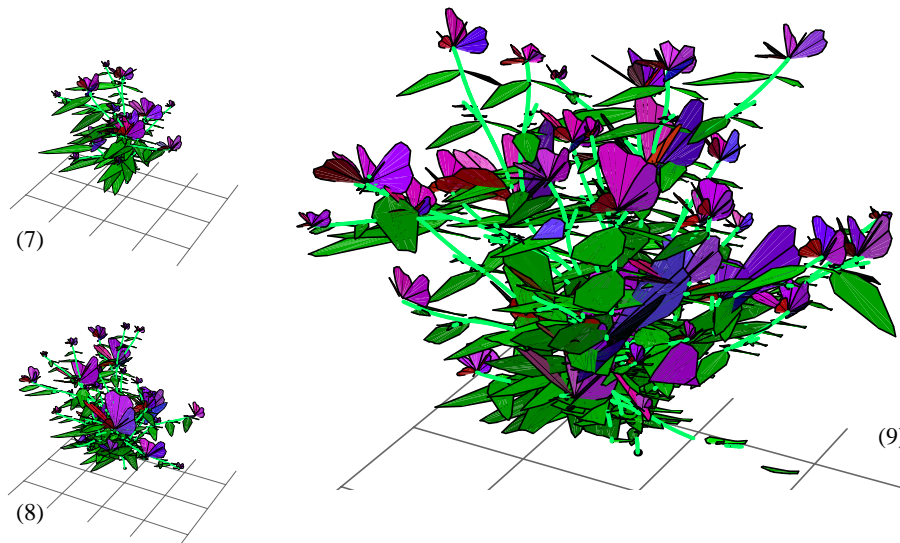


Fig. 5. Growth stages of the best individuals from generation 1 and 13, the numbers in brackets denoting L-system iterations. The previous growth patterns for the generation 13 individual can be found in figure 2.

*H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.)
PPSN-IV, Parallel Problem Solving from Nature IV (1996),
Berlin, Germany.
Springer-Verlag, Berlin, Germany.
pp. 42-51.*

All individuals of generation 13 encode for complex growth patterns which is on the one side due to the property of non-causality of L-systems which means that small changes within the rewrite rules can have dramatic effects on the encoded phenotypic appearance. On the other side, evolutionary selection and variation applied to sets of L-systems drive the evolution process to ever more competitive encodings of development programs. Figure 5 shows growth stages of the best individual of the initial population (indiv. 2) compared to the best individual after 13 generations (indiv. 1). This best evolved L-system encodes a plantlike structure that grows rapidly, widespread, and with a bunch of blooms and leaves.

5 Steps towards realistic development and coevolution

Of course, there are many more factors that influence an organism's development in a natural competitive environment. With our simple example we only intended to show the feasibility of inferring L-systems encoding morphogenetic processes exhibiting specific characteristics. The described evolutionary programming techniques are currently being incorporated into a coevolutionary system for simulating plants growing in competition for resources like nutrients, sunlight, or space. Finally, on a more microscopic level, our L-system based developmental programs will be extended by functional genome structures like operons, promoters, and regulators in order to include intra-gene interaction into evolution processes.

References

1. Green, N., *Voxel space automata: Modeling with stochastic growth processes in voxel space*, Computer Graphics, ACM SIGGRAPH, 23(3), 1989, pp. 175-184.
2. Jacob, C., *Genetic L-System Programming*, Parallel Problem Solving from Nature - PPSN III, LNCS 866, Springer, Berlin, 1994, pp. 334-343.
3. Jacob, C., *MathEvolvica - Simulierte Evolution von Entwicklungsprogrammen der Natur*, Dissertation, Arbeitsberichte des Instituts für mathematische Maschinen und Datenverarbeitung, Band 28, Nummer 10, Erlangen, 1995.
4. Jacob, C., *Modeling Growth with L-Systems & Mathematica*, Mathematica in Education and Research, Vol. 4, No. 3, TELOS Springer, 1995.
5. Jacob, C., *Evolving Evolution Programs: Genetic Programming and L-Systems*, in: Koza, J.R., Goldberg, D.E., Fogel, D.B., and Riolo, R.L. (eds.), Genetic Programming 1996, MIT Press, Cambridge, MA, 1996.
6. Koza, J., *Genetic Programming*, MIT-Press, Cambridge, MA, 1992.
7. Koza, J., *Discovery of rewrite rules in Lindenmayer systems and state transition rules in cellular automata via genetic programming*, Symposium on Pattern Formation - SPF-93, Claremont, CA, 1993.
8. Koza, J., *Genetic Programming II*, MIT-Press, Cambridge, MA, 1994.
9. Kurth, W., *Growth Grammar Interpreter GROGRA 2.4*, Berichte des Forschungszentrums Waldökosysteme, Universität Göttingen, Reihe B, Band 38, 1994.
10. Lindenmayer, A., and Rozenberg, G. (eds.), *Automata, Languages, Development*, North-Holland, 1975.
11. Niklas, K.J., *Computer-simulated Plant Evolution*, in: Scientific American, 254 (March), 1986, pp. 68-75.
12. Peitgen, H.-O., Jürgens, H., Saupe, D., *Chaos and Fractals*, New Frontiers of Science, Springer-Verlag, 1993.
13. Prusinkiewicz, P., and Lindenmayer, A., *The Algorithmic Beauty of Plants*, Springer, New York, 1990.

H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.)
PPSN-IV, Parallel Problem Solving from Nature IV (1996),
Berlin, Germany.
Springer-Verlag, Berlin, Germany.
pp. 42-51.