Technical Report 95-30 October 1995

Rijksuniversiteit te Leiden
Vakgroep Informatica

Using L-Systems as Graph Grammar: G2L-Systems

Egbert J.W. Boers

SEIS

Department of Computer Science
Leiden University
P.O. Box 9512
2300 RA Leiden
The Netherlands

GAiN

Using L-systems as Graph Grammars: G2L-Systems[†]

Egbert J.W. Boers
Leiden University
Department of Computer Science
Email: boers@wi.leidenuniv.nl

Abstract—*This paper will show how the interpretation of strings resulting from L-systems can be adapted in such a way that L-systems can be used as graph grammars. One of the advantages of this adaptation is the possibility to make use of the context dependency available in L-systems. This context is calculated by using the graph interpretation of the strings at each successive step of the rewriting mechanism. The modifications to L-systems necessary in order to use them as graph grammars will be explained. A simple proof will be given that every possible directed graph can be generated using this representation. At the end of the paper an application of this G2L-system based graph grammar will be presented. In this application a genetic algorithm optimizes the rewriting rules of an G2L-system, trying to come to a more scalable method for finding good artificial neural network architectures in comparison with methods that do not use graph grammars. The presented G2L-system was specifically designed for this application to favour modular networks, but might be useful in general as well.*

Keywords—L-systems, Graph Grammars, Genetic Algorithms, Artificial Neural Networks, Modularity.

1 Introduction

The development of plants and animals is governed by the genetic information contained in each cell of the organism. Each cell contains the same genetic information (the *genotype*), which determines the way in which each cell behaves and, as a result of that, the final form and functioning of the organism (the *phenotype*). This genetic information is not a blueprint of that final form but can be seen as a *recipe* [4] that is followed not by the organism as a whole but by each cell individually. The shape and behaviour of a cell depend on those genes that are expressed in its interior. Which genes actually are expressed depends on the context of the cell. Already at the very beginning of an organism's life, subtle intercellular interaction takes place, changing the set of genes that are expressed in each cell. This process of *cell differentiation* is responsible for the formation of all different organs.

In order to model this kind of development in plants the biologist Aristid Lindenmayer developed a mathematical construct called 'L-systems' [12]. With an L-system, a sequence of symbols can be rewritten into another sequence, by rewriting all symbols in the string in parallel into other symbols, using so-called 'rewriting rules'. Whether a specific rewriting rule can be applied or not depends on which rules have been applied in the past and on the neighbouring symbols of the symbol to be rewritten.

In our research [1, 2, 3, 8] we tried to find a way to implement a *growth model* that could generate artificial neural networks. To optimize the design of such a network for a specific task, we used a genetic algorithm to optimize the rules to be used by the growth model. What we needed was a context sensitive graph rewriting system that could generate *modular* graphs. After trying several alternatives [1] we focused on L-systems because they make use of context and were already used to generate biological structures. The changes to the L-system mechanism which were needed to construct a graph grammar will be presented in this paper.

The rest of this paper has the following structure: the next section will describe in short the workings of simple L-systems. Sections 3 and 4 will explore some of the possibilities of L-systems. Sections 5 and 6 will give a description of how we can adapt the operation and interpretation of a 2L-system (see section 3) to get a Graph-2L-System (G2L-System). Section 7 will give a short description of the application of the developed grammar: genetically optimizing artificial neural networks. Also a short introduction to the ideas behind genetic algorithms and artificial neural networks will be presented. A comparison with other methods that use graph grammars to generate neural networks is given in section 8. Our conclusions of using the G2L-system and some further research possibilities will be in section 9.

2 L-systems

L-systems are parallel string rewriting mechanisms. A grammar, the definition of what is called a *language* in formal language theory, consists of an alphabet, a starting string and a set of rewriting rules. The starting string, also known as the *axiom*, is rewritten by applying the rewriting rules: each rule describes how a certain symbol or string should be rewritten into another string of symbols. Whereas in most other grammars rewriting rules are applied sequentially, in an L-system all rewriting rules are applied in parallel to generate the next string in the rewriting process.

[†]This work has been presented at the Fifth International Workshop on Graph Grammars and their Application to Computer Science, Williamsburg, Virginia, November 13–18, 1994. This paper is available as Technical Report 95-30, Computer Science Department, Leiden University. It is available by ftp:
ftp://ftp.wi.leidenuniv.nl/pub/CS/TechnicalReports/1995/tr95-30.ps.gz

We define the L-system grammar G of a language L as:

$$G = \{\Sigma, \Pi, \alpha\},$$

where Σ is the finite set of symbols or alphabet of the language, Π is the finite set of rewriting rules (also: production rules), and $\alpha \in \Sigma^*$ is the starting string (axiom) of L.

$$\Pi = \{\pi | \pi : \Sigma \rightarrow \Sigma^*\}$$

is the set of rewriting rules. Each rewriting rule defines a unique rewriting of a symbol of Σ, the left side of the rewriting rule, into a string $s \in \Sigma^*$, the right side of the rewriting rule. All symbols in Σ that do not appear as the left side of a rewriting rule are rewritten into themselves. For clarity, these default rewriting rules are usually not included in Π.

Example 1—If we take the L-system

$$G = \{\Sigma, \Pi, \alpha\}$$

with

$$\Sigma = \{A, B, C\},$$
$$\Pi = \{A \rightarrow BA, B \rightarrow CB, C \rightarrow AC\} \text{ and}$$
$$\alpha = ABC$$

we get the following language:

$$L = \{ABC, BACBAC, CBBAACCBBAAC, \ldots\}. \ \square$$

The example shows that each rewriting rule is applied in parallel in consecutive *rewriting steps*. After each rewriting step, all symbols of the original string which *matched* with the left side of a rewriting rule have been rewritten into their successor, as defined by the right side of this rewriting rule. The language L is the set of all strings that are generated by applying the rewriting rules; each rewriting step generates the next string of the language until no more rules apply.

Alternatively, one could view the set of rewriting rules Π as an *operator*. Applying the operator Π to a string executes all rewriting rules as explained above. Using this operator, we can write the language L produced by a grammar G in a shorter form as:

$$L = \{\alpha, \Pi\alpha, \Pi(\Pi\alpha), \Pi(\Pi(\Pi\alpha)), \ldots\}$$
$$= \Pi^*\alpha.$$

3 Extensions of L-systems

The simple L-systems described in the previous paragraph can be extended with *context sensitivity*, which is used to generate conditional rewriting rules.

Example 2—The rewriting rule

$$A < B \rightarrow C$$

expresses that B may only be rewritten if it is preceded by an A. $\square$

In general the context sensitive rewriting rules will have this form:

$$L < P > R \rightarrow S,$$

with

$$P \in \Sigma \text{ and } L, R, S \in \Sigma^*.$$

P, the predecessor, and S, the successor, are what we earlier called the left and right side of a rewriting rule. L and R, the left and right context respectively, determine whether or not a rewriting rule will be applied to a symbol P in the string. Only those symbols P with L on its left and R on its right side will be rewritten. One or both contexts may be empty, in which case only the context that is present in the rewriting rule will be checked.

Example 3—Suppose our current string in the rewriting process of an L-system is:

$$ABBACAADBAABBAC,$$

and we have the following rewriting rules:

$$CA < A \rightarrow EG$$
$$A < A > B \rightarrow BE$$
$$B \rightarrow RT,$$

the string that results after one rewriting step is:

$$ARTRTACAEGDRTABERTRTAC. \ \square$$

L-systems without context are called 0L-systems. L-systems with one-sided context or two-sided context are called 1L-systems or 2L-systems respectively. It is allowed to have empty contexts, which implies[†] that:

$$\text{0L-systems} \subset \text{1L-systems} \subset \text{2L-systems}.$$

If two rewriting rules apply for a certain symbol, one with and one without context, the one with context is used. In general, the rewriting rule that is more specific will prevail. It is, however, possible to get conflicts between several rules that can be applied to the same symbol.

Two other extensions are *probabilistic* rewriting rules and rewriting rule *ranges*. With probabilistic rewriting rules, more than one successor can be given for the same $L < P > R$, each with a fixed probability. When rewriting a string, one of the rules is selected with a chance proportional to its probability. Rewriting rule ranges introduce a temporal aspect to the L-system by telling which rules are active at a certain rewriting step.

4 Interpretation of strings

When we attach a specific meaning (based on a LOGO-style turtle, [17]) to the symbols in a string, we are able to visualize the strings resulting from the rewriting steps.

The usual—see e.g. [14]—interpretation of the symbols used is as follows:

- 'F': draw a line in the current direction,
- '+': rotate φ° to the left and
- '–': rotate φ° to the right.

Example 4—The well-known Koch graph from figure 1, can be described by the following L-system:

$$G = \{\{F, +, -\}, \{F \rightarrow F + F - -F + F\}, F\}$$
$$\varphi = 60°.$$

[†]The proof that this is a strict hierarchy is trivial.

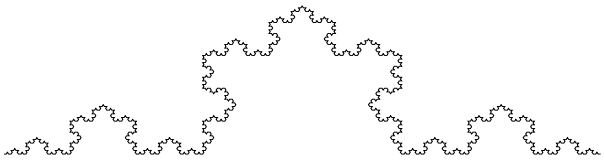


Figure 1: The Koch graph (after 5 rewriting steps)

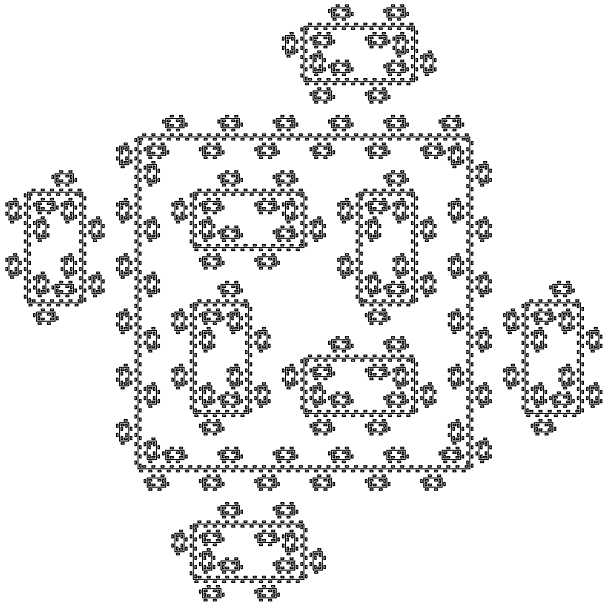
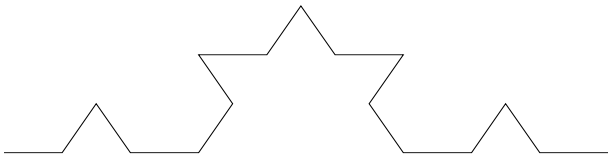


Figure 2: The recursive islands of example 5

The axiom and the rewriting rule, also called the *initiator* and *generator*, look like:

After one rewriting step, the initiator will look like the generator; each successive rewriting step replaces every line with the (scaled) generator, so the next rewriting step will result in:



and so on. ❑

Other symbols frequently mentioned in the literature are:

- *'f'*: move in the current direction but do *not* draw a line,
- '[': push the current position and direction on a stack, and continue,
- ']': pop a position and direction from the stack and continue from there.

Example 5—The effect of the move instruction can be seen in figure 2. This figure has been drawn after two rewriting steps of the following L-system:

$$G = \{\Sigma, \Pi, \alpha\}$$

with

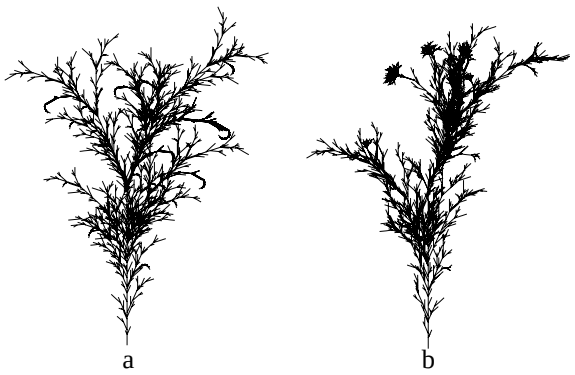


Figure 3: 2L-systems (see text)

$\Sigma = \{F, f, +, -\}$,

$\Pi = \{F \rightarrow F\text{–}f\text{+}FF\text{–}F\text{–}FF\text{–}Ff\text{–}FF\text{+}f\text{–}FF\text{+}F\text{+}FF\text{+}Ff\text{+}FFF,$
$f \rightarrow ffffff\}$,

$\alpha = F - F - F - F$,

and $\varphi = 90°$. ❑

The usage of brackets for push and pop operations while interpreting the string was introduced by Lindenmayer [12] to achieve more realistic representations of plants and trees. A nice example is a drawing, see figure 3a, of a 2L-system taken from [10]. A stochastic adaptation of this is shown in figure 3b, where in just one rule, $0 < 0 > 1 \rightarrow 1\,[+F1F1]$, the '+' symbol was changed in a '–' symbol in 50% of the cases.

Introducing brackets in 2L-systems has some effect on the determination of the left and right contexts of the rewriting rules. The left context consists of the path before the predecessor, and the right context consists of a subtree after the predecessor. To detect this, the tree representation of the string must be examined at each rewriting step.

Example 6—(from [14]) The rewriting rule with the following left-side

$$BC < S > G\,[H]\,M,$$

can be applied to the symbol S in the string

$$ABC\,[DE]\;[SG\,[HI\,[JK]\,L]\,MNO]\;.$$

Figure 4 gives a graphical interpretation of this match. ❑

For an extensive look at the possibilities of and some more extensions to this kind of interpretation of the strings resulting from an L-system see e.g. [10, 13, 14].

5 A graph interpretation: GL-systems

L-systems originally were constructed to model biological growth, which makes it a logical choice to use them when trying to describe the growth of the brain (see section 7). For an L-system to generate the graph topologies that are needed, a suitable interpretation has been constructed in [1, 2 and 3]. Here, a generalized interpretation is given, allowing the construction of cyclic graphs, which allows for recurrent networks.

Instead of the graphical interpretation of a string resulting from the rewriting steps, a direct transformation into a graph is possible using

$$\Sigma_g = \mathcal{Z} \cup \{ [\,,]\,\} \cup \Sigma,$$

with $\mathcal{Z}$ the set of integer numbers where each $j \in \mathcal{Z}$ is read as one symbol, called a *connect*. The symbols '[' and ']' are used to denote subgraphs (see below). Each $n \in \Sigma$ in the string represents a node in the corresponding graph. The numbers *j* directly behind a symbol from Σ or ']' connect the node or subgraph with a directed edge to the j^{th} node or subgraph to the left or right in the string, depending on the sign of *j*.

Example 7—The string *A*1 2 *B*2 *C*0 1 *D*–3 represents the graphs drawn in figure 5. The symbols in the nodes are not labels of the graph, but are used as matching symbols for the GL-system. ❑

Theorem—All possible finite unlabelled directed graphs can be represented and generated with a finite GL-system.

Proof—Label all nodes of an arbitrary graph *F* with A_i, $i = 1 \ldots n$, with *n* the number of nodes in *F*. For each edge (A_i, A_j) insert the number $j - i$ directly after A_i in the string $A_1 \ldots A_n$. The resulting string *S* can now be used directly to construct the GL-system *G* that generates the graph, simply by taking the string *S* as axiom. ❑

The special symbols '[' and ']' are used to group nodes into subgraphs (or *modules*), which can be recursively repeated. Each subgraph is assigned a *level*, which is

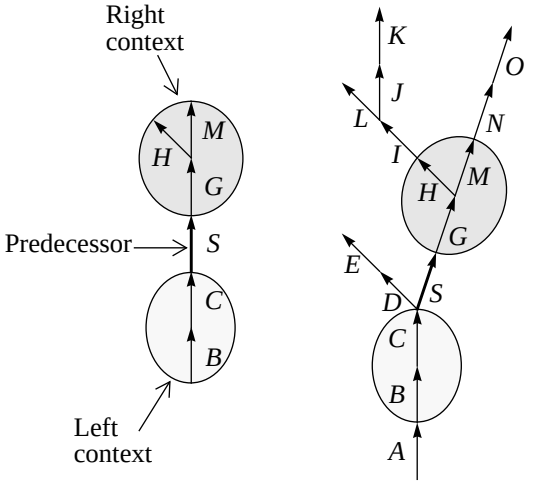


Figure 4: Matching in bracketed 2L-systems.

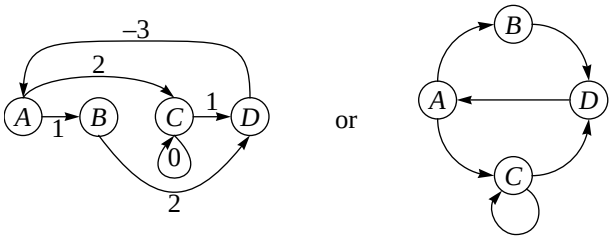


Figure 5: the graphs of example 7.

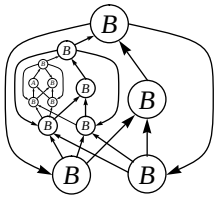


Figure 6: The recursive graph of example 8.

calculated by counting the number of surrounding bracket pairs. In calculating the j^{th} node or subgraph when making the connections, each subgraph is seen as a unity, and can be regarded as one node on the same level as the subgraph. The numbers directly to the right of a closing bracket will connect all *output nodes* of the subgraph. All connections made *to* a subgraph are connected with all its *input nodes*. There are two possible definitions of in- and output nodes:

- The first possibility defines the output nodes of a subgraph as those nodes that have no outgoing edges to other nodes of the same subgraph. The input nodes of a subgraph are those nodes that do not have incoming edges from within the same subgraph. GL-systems with this interpretation will be referred to as *strictly modular*. This definition is particularly suited for generating layered networks, and is therefore used in the application described in section 7.
- The second possibility takes into account the order of the symbols in the string representing the graph. Besides having the input and output nodes as defined in the previous definition, also those nodes that have no edges to nodes in the same subgraph that are denoted to the right in the string are output nodes and those nodes that do not have edges from nodes of the same subgraph that are denoted to the left in the string are input nodes. GL-systems with this interpretation will be referred to as *not strictly modular*.

As a consequence, specific edges to nodes within a subgraph can not be made from outside of the subgraph, while edges from a node in a subgraph to nodes or other subgraphs outside its own subgraph are possible, after counting across the subgraph boundary, the level at which the nodes are counted is decreased by one. This idea of limiting the possible connections from and to a subgraph corresponds to a sense of information hiding and improves the capability of the GL-system to generate modular graphs, which is important for our purposes (see section 7).

Example 8—A nice example of defining a recursive graph is the following not strictly modular GL-system:

$$\begin{aligned} G = \{ &\{A, B, [,], 1, -2\}, \\ &\{A \rightarrow [BB]\,1\,[AB]\,1B - 2\}, \\ &A\}. \end{aligned}$$

After three rewriting steps the graph will look as shown in figure 6. The string will be:

[*BB*]1[[*BB*]1[[*BB*]1[*AB*]1*B*–2*B*]1*B*–2*B*]1*B*–2. ❑

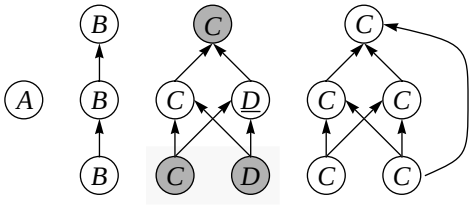


Figure 7: The successive rewriting steps

6 Context in G2L-systems

The definition of *context* in G2L-systems is not the same as in normal L-systems. Usually the context of a symbol that is being rewritten is directly on the left and right side of that symbol. In bracketed 2L-systems, context can be seen by looking at the path towards, and a subtree directly after the predecessor. This limits the part of the string that has to be interpreted in order to match context.

In G2L-systems, however, context can only be determined after the complete string has been interpreted. Also, not only single symbols may be rewritten, but complete substrings can be replaced by others. Here, a left to right order for matching the rewriting rules is followed, and a preference for more specific rewriting rules when more rules are possible. The specifity of a rewriting rule is determined simply by counting the number of symbols in the left side of the rule. G2L-systems that have no conflicts, which means that no ambiguity exists about which rule has to be applied, will be called *conflict-free*.

The left context of a node (or subgraph) is defined as the set of all symbols and subgraphs in the string that have edges to the node (or subgraph) in the graph interpretation of the string. The right context is defined in the same way, looking at the edges going away from the node (or subgraph). When, if present in the rewriting rule, both context parts of the rule are a subset of the left and right context in the graph, the rule is said to match and can be applied.

Example 9—Examine the following strictly modular G2L-system:

$$G = \{\Sigma_g, \Pi, \alpha\}$$

with

$$\begin{aligned}\Sigma &= \{A, B, C, D\}\\ \Pi &= \{A \rightarrow B1B1B,\\ &\quad B > B \rightarrow [CD],\\ &\quad B \rightarrow C,\\ &\quad C < D \rightarrow C,\\ &\quad D > D \rightarrow C2\}\\ \alpha &= A.\end{aligned}$$

The successive steps in the rewriting process are shown in figure 7. After the last step no more rules apply:

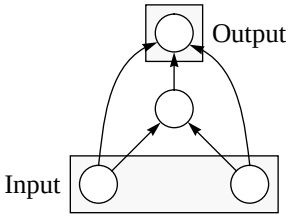


Figure 8: The simplest architecture that can implement the logical XOR function.

$$\begin{gathered}A\\ \Downarrow\\ B1B1B\\ \Downarrow\\ [CD]\,1\,[C\underline{D}]\,1C\\ \Downarrow\\ [CC2]\,1\,[CC]\,1C.\end{gathered}$$

To show the workings of left and right context, these are drawn in figure 7 for the underlined symbol $\underline{D}$ after the second rewriting step. The left context of $\underline{D}$ is the set $\{C, D, [C, D]\}$ and the right context is $\{C\}$. Note that the other D does not have a C in its left context. ❑

7 Application

The application for which the G2L-system was developed is an attempt to imitate the course of evolution of animal brains in nature with *genetic algorithms*, with the purpose of creating a means of finding optimal *artificial neural network* architectures. It can be seen as a reverse engineering attempt of the process that has 'invented' the human brain [1]. The following will give a short introduction to the terminology and functioning of artificial neural networks and genetic algorithms, and will explain why it is useful to use G2L-systems. A more complete introduction of neural networks and genetic algorithms can be found in [1 or 5 and 9].

7.1 Artificial neural networks

Artificial neural networks try, in principle, to imitate the computations as performed by the neural system of animals. A neuron and its axon, dendrites and synapses, is usually modelled as a simple computational unit that implements a simple threshold function on a weighted sum of its input values. The *architecture* of an artificial neural network is defined as the directed graph representing the *connectivity* of the network. Each node represents a unit and each directed edge represents a weighted connection from one unit to another.

Example 10—Figure 8 shows an example of the architecture of the simplest artificial neural network that can implement the logical XOR function [15, 16]. The nodes of this network can be topologically sorted, so the network is a *feedforward network*. A very simple way to restrict grammars to generating feedforward networks is allowing only positive connects. ❑

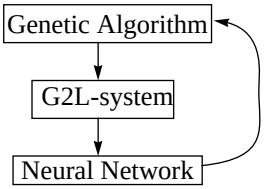


Figure 9: Where the G2L-system is used.

7.2 Genetic algorithms

One of the major problems in neural network research is the difficulty of finding the right architecture for a neural network that has to perform a certain function [8]. Genetic algorithms have been tried as search method, when looking for the right architecture. Genetic algorithms [5] are search algorithms that work on a population of possible solutions. Generated offspring has to compete for survival in a Darwinian way, based on the relative quality of the generated solutions.

7.3 Using G2L-systems

Early attempts of searching in an evolutionary way for optimal network architectures failed on larger problems because of the exponentially growing search-space of possible network architectures, that were usually coded as *blueprints* in the *genes* of the population members. Our current research concentrates on using the described G2L-system to speed up the search by decreasing the number of bits needed to code for one network, and by increasing the inherent modularity that is caused by the presented interpretation [2, 3]. The G2L-system is used as a *recipe*, not coding each connection and node separately, but compactly describing how the network should *grow* from the axiom. The complete system is schematically drawn in figure [9].

A major problem that was encountered in reporting on the work described above is the lack of a drawing tool able to visualize the generated network architectures. Currently a tool is under development that uses the subgraph information available in the string to ease the task of optimizing the graph lay-out. The successive levels are step-size optimized using a genetic algorithm. Further extensions, namely also using the rewriting rules, are being examined.

8 Other graph grammar approaches

Recently several other graph grammar mediated genetic algorithms for optimizing artificial neural network architectures have been developed [6, 7, 11]. The complete paper will give a thorough comparison with the grammar described here.

9 Conclusion

The L-system based graph grammar described in this paper allows for a very compact coding for a certain kind of graph. Although it is shown that all possible graphs can be generated with a G2L-system, clearly, iterating and recursive application of rewriting rules leads to networks that are very modular. It is expected that the architecture optimalization using genetic algorithms is greatly helped with the use of G2L-systems.

10 References

[1] E.J.W. Boers and H. Kuiper; *Biological Metaphors and the Design of Modular Artificial Neural Networks*. Unpublished Master's thesis, Leiden University, 1992.

[2] E.J.W. Boers, H. Kuiper, B.L.M. Happel and I.G. Sprinkhuizen-Kuyper; 'Designing modular artificial neural networks'. In: H.A.G. Wijshoff (ed.); *Proceedings of the Conference Computing Science in The Netherlands (CSN'93)*, 89–96, 1993.

[3] E.J.W. Boers, H. Kuiper, B.L.M. Happel and I.G. Sprinkhuizen-Kuyper; 'Biological metaphors in designing modular artificial neural networks'. Abstract in: S. Gielen and B. Kappen (eds.); *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, Springer-Verlag, 1993.

[4] R. Dawkins; *The Blind Watchmaker.* Longman, 1986. Reprinted with appendix by Penguin, London, 1991.

[5] D.E. Goldberg; *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, 1989.

[6] F.C. Gruau; 'Cellular encoding of genetic neural networks'. Technical Report 92-21, Institute IMAG, Grenoble, France, 1992.

[7] F.C Gruau; *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, l'Ecole Normale Supérieure de Lyon, 1994.

[8] B.L.M. Happel and J.M.J. Murre; 'The design and evolution of modular neural network architectures'. To appear in: *Neural Networks*, Pergamon, 1994.

[9] J. Hertz, A. Krogh and R.G. Palmer; *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, 1991.

[10] P. Hogeweg and B. Hesper; 'A model study on biomorphological description'. In: *Pattern Recognition*, 6, 165–179, 1974.

[11] H. Kitano; 'Designing neural networks using genetic algorithms with graph generation system'. In: *Complex Systems*, 4, 461–476, Champaign, IL, 1990.

[12] A. Lindenmayer; 'Mathematical models for cellular interaction in development, parts I and II'. In: *Journal of Theoretical Biology*, 18, 280–315, 1968.

[13] P. Prusinkiewicz and J. Hanan; *Lindenmayer Systems, Fractals and Plants*. Springer-Verlag, New York, 1989.

[14] P. Prusinkiewicz and A. Lindenmayer; *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

[15] I.G. Sprinkhuizen-Kuyper and E.J.W. Boers; 'Classification of all stationary points on a neural network error surface'. In: J.C. Bioch and S.H. Nienhuys-Cheng (eds.); *BENELEARN-94, The 4th Belgian-Dutch Conference on Machine Learning*, 1994.

[16] I.G. Sprinkhuizen-Kuyper and E.J.W. Boers; 'The error surface of the simplest XOR network has no local minima'. Technical Report 94-21, Department of Computer Science, Leiden University, 1994.

[17] A.L. Szilard and R.E. Quinton; 'An interpretation for D0L-systems by computer graphics'. In: *The Science Terrapin, 4*, 8–13, 1979.