

# Attribute Grammar Encoding based upon a Generic Neural Markup Language: Facilitating the Design of Theoretical Neural Network Models

Talib S. Hussain

Department of Distributed Systems and Logistics  
BBN Technologies  
Cambridge, MA 02138  
E-mail: hussain@ieee.org

**Abstract—** There is a need for tools that facilitate the systematic exploration of novel theoretical neural network models. Existing neural network simulation environments, neural network specification languages, and genetic encoding of neural networks fall short of providing the tools needed for this task. We suggest that a useful approach to the design of such tools may be the use of a grammar to capture neural design principles as well as structural and behavioral elements, and mechanisms to automatically translate the parse trees of the grammar into complete neural network specifications in a generic format. We present the Attribute Grammar Encoding (AGE) as a specific example of our approach that uses attribute grammars to create descriptions of neural network solutions in an XML-based format termed the Generic Neural Markup Language (GNML). Lessons learned from the development of this system are presented to identify and address the issues of a broader application of this approach to other specification formats and other grammar encoding approaches.

## I. INTRODUCTION

The field of neural networks has produced a large number of theoretical neural network models that vary along many different dimensions, including underlying neural functionality, structural properties, mechanisms for the interactions of neural components, internal structural and behavioral regularity, and the overall number and variety of architectures encompassed by the model. Despite these significant developments, a researcher today interested in developing new theoretical neural network models is faced with a paucity of tools that facilitate the systematic exploration of new neural architectures along multiple dimensions of interest. As such, that researcher must generally follow a manual or ad-hoc process of adapting existing models or generating new models from scratch.

Neural network simulation environments [1-7] and specification languages [8-12] offer a rich capability for specifying a wide range of neural architectures, but variations in those architectures must generally be explored in a highly manual fashion. Genetic encodings of neural

networks [13-20] offer the capability to explicitly specify neural properties of interest and explore along those dimensions in a systematic fashion through evolutionary processes, but are generally limited to a small number of structural or functional dimensions. In particular, grammar-based genetic encodings have been shown to be a natural mechanism for specifying broad families of neural structures using a small number of design rules [19, 21, 22], but have been limited as tools for model development due to the typical need for a different interpreter for each grammar.

The wealth of existing neural network architectures and the functional and structural elements that make up those architectures provide a rich foundation of material and techniques for the creation of new theoretical neural network models. For example, new models may be created by composing new architectures from the elements of existing ones, integrating different structural principles within a common framework, incorporating multiple behavioral approaches within a single model, linking architectures together as modules, and so on. While existing tools provide some basic capability for exploiting this foundation, there is a need for new approaches to the design of neural network models that leverage existing discoveries and facilitate the systematic exploration of novel variations.

We suggest a two-tiered approach for the development of effective neural network modeling tools: Use a grammar to capture neural design principles as well as structural and behavioral elements, and automatically translate the parse trees into a complete neural network specification in a generic format that may then be executed by a generic interpreter. We believe this approach addresses many of the key limitations noted earlier, and will provide a rich basis for the creation of grammars that will allow researchers to systematically explore high-level model design issues, such as alternative approaches for integrating different learning mechanisms, different types of behavioral relationships that may exist between different neural modules, the automatic exploration of task decompositions and the selection of

appropriate structural and functional components to specialize on those subtasks.

We present a specific application of this approach, termed the Attribute Grammar Encoding (AGE) [21-23], in which attribute grammars [24] capture a variety of functional and structural design rules, and automatically generate a complete neural network specification in a generic eXtended Markup Language (XML)-based format [25], termed the Generic Neural Markup Language (GNML). We provide a summary of the lessons learned during the development of AGE and discuss the benefits and drawbacks of the overall approach as a general technique for the design of systematic modeling tools for neural networks.

## II. BACKGROUND

The neural network design tools available today generally fall into three categories, neural network simulation environments, neural network specification languages, and techniques for the genetic encoding of neural networks. Numerous neural network simulation environments, such as MATLAB Neural Network Toolbox [2], PDP++ [4], JavaNNS [5] and NeuroSolutions [6] have been developed with the aim of aiding the problem-oriented practitioner to develop new solutions based upon a “toolkit” or “library” of existing architectures. The focus in these simulation environments has been to provide the tools needed to handle domain data, select an appropriate architecture, specify the parameters for that architecture, train the network and analyze the performance of the network.

A number of neural network specification languages, such as AXON [8] and CONNECT [9] and EpsilonNN [10], have been developed with the aim of providing the capability to program a complex network from simpler components and routines, rather than programming from scratch using a traditional programming language. Many simulation environments also incorporate scripting languages which allow the user to define functions and structures. For instance, PDP++ has a scripting language that permits the user to specify new components using a C++-like language. In addition to satisfying the needs of the problem-oriented practitioner, neural network specification languages address the needs of a broad community of researchers, in a diverse set of fields, interested in creating custom architectures in order to explore and test new underlying principles, such as biological, cognitive or machine learning principles. For instance, GENESIS [7] allows the user to manually specify the activation functions of nodes, learning functions and other neural behaviors for improved biological modeling.

Simulation environments that do not incorporate a neural network specification language are generally very rigid as a tool for exploring variations in the neural network models themselves. Neural network specification languages (and

their associated simulation environments) do provide the basic capability required to create and explore new models, but require the researcher to program those variations directly, and as such follow a slow, manual process. Further, the programs developed within a network specification language may only be used within a single simulation environment, although recent efforts [12] have been made to create an XML-based format for the specification of neural networks that may be applied more widely.

A number of techniques for the genetic encoding of neural networks have been developed with the aim of providing the capability to represent some properties of a neural network model in a manner that enables an evolutionary algorithm to automatically manipulate those properties in interesting ways. Early genetic encodings used highly static and limited representations that enabled the exploration of few neural properties and simple networks. For instance, direct encoding [13, 14] specifies only the weight values of a network, while all other aspects of the network, including topology, are kept fixed. A parametric encoding [15, 16] may specify a wide variety of network properties, but usually these are limited in number and to high-level properties. Recent genetic encodings tend to use more complex representation structures, including many that are based on grammars or grammar-like structures, to enable the exploration of a wider variety of structural and behavioral characteristics. For instance, developmental rule encoding [17, 18] and cellular encoding [19] represent a network as a set of re-write rules that may be applied to an initial network configuration to produce a final topology that may range from sparse to dense and highly irregular to highly regular or modular; functional aspects of the neurons generally remain fixed. Standard tree-based, genetic programming encoding has been used to evolve the form of the learning rule from simpler functional elements, but assumes a fixed network topology [20].

Genetic encodings provide a basic tool for the automated exploration of a number of neural network variations, but current techniques are generally limited to a single, relatively homogeneous neural network model. This is due to the fact that the goal is not completeness of the representation, but rather the utility of the manipulations that are enabled by the encoding. Further, since most practitioners are highly problem-oriented, extending the scope of the evolutionary search to explore too many neural properties reduces the effectiveness of the search in generating useful solutions. As such, most existing genetic encodings are purposefully simple and instead incorporate the majority of the properties of a neural model within an interpreter that performs the decoding step of generating a functional neural network for each genetic description.

### III. ATTRIBUTE GRAMMAR ENCODING

A key goal in developing the Attribute Grammar Encoding (AGE) was to create an encoding for neural networks that enabled the consistent representation of a wide variety of neural network models and that made as many aspects of those models as possible available for manipulation within the grammar. To accomplish this goal, an important step was to avoid a pitfall of current grammar-based genetic encodings, which are limited primarily by the practice of incorporating the majority of the model properties within the interpreter.

We adopted a two-tier approach to the representation of neural networks. A generic neural network specification format was created based on an underlying neuron model and model of signal processing. XML was used as a basis for the representation to allow the specification of arbitrary structural and functional components through nesting of different levels of detail. Using this network specification format, termed the Generic Neural Markup Language (GNML), we created attribute grammars that enabled the manipulation of both functional and structural neural properties within the same grammar. Finally, we created a neural interpreter that accepted GNML specifications and produced functional neural networks capable of learning.

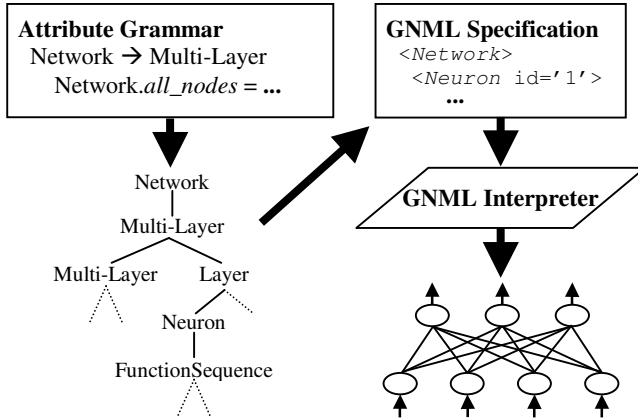


Fig 1. Generating functional neural networks within AGE

An attribute grammar consists of a context-free grammar in which each symbol in a grammar production has a set of attributes associated with it, and each production specifies attribute evaluation rules that determine how the attributes of its right hand symbols and/or left hand symbol are to be computed. Within AGE, the productions of an attribute grammar define a space of possible neural network architectures. To enable a single grammar to specify both functional and structural neural properties, we adopted a grammar design in which the attribute evaluation rules use string and set operations to compose GNML specifications within the attributes directly. Through the process of evaluating the attributes of the parse tree, a complete GNML specification of a single neural network is produced.

Figure 1 illustrates a typical process within the AGE system, in which a parse tree is first generated from the attribute grammar, a GNML specification is then extracted from the attributed parse tree and passed to the GNML interpreter, which finally produces a functional neural network.

#### A. Neuron Model

The neuron model adopted as a basis for the design of GNML and AGE is illustrated in Figure 2. The model exhibits several key properties that are based upon the needs of an effective tool for the exploration of novel neural network models.

To enable the specification of a wide variety of neural network architectures, we developed GNML to be capable of representing networks in which an arbitrary number of signal types may be transmitted and manipulated by the neurons. This enables decisions at the level of the grammar to enforce the need for specific signal types, such as the typical activation and feedback signals. Thus, we define a neuron as a processing element that may receive signals of multiple types and may transmit signals of multiple types. Additionally, a connection within GNML is defined as a directed link that (1) connects a single node or input source of the network to a single node (possibly the same node) or output source of the network, such that (2) the link is capable of carrying signals of a single, fixed, specific type.

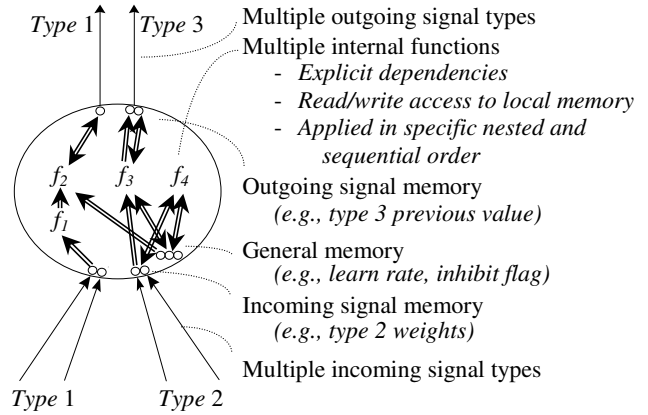


Fig 2. Neuron model for GNML and AGE

To enable the representation and manipulation of complex neural behaviors within the grammar, we require a consistent yet flexible neuron model in which the functionality of a neuron is decomposable into distinct parts that may be manipulated independently. These parts form the basis for some terminal symbols of the grammar. Further, it requires that a neuron must be specified as a finite number of parts since a grammar is not capable of providing infinite parse trees. Ideally, where possible the parts should be common to multiple types of neurons. If we do not have this property, then we will not benefit from the grammar's ability to create

many alternatives from a small set of rules. Thus, we define a neuron to have a set of local memory variables associated with each incoming and outgoing signal type (signal memory), a set of local memory variables that are not associated with the signals (general memory), and a sequence of internal functions. Each function is defined to accept a fixed set of parameters. Those parameters may be signal memory variables, general memory variables, or the results of other functions. Thus, functions may nest. Further, each function may modify the memory parameters that it accepts. Thus, functions have read and write access to memory.

Figure 2 illustrates the neuron model. The double lines with single arrow heads indicate a parameter relationship. A function that is a parameter of another is a nested function. The double lines with two arrow heads indicate a parameter relationship in which the accepting function may actually modify the original memory variables. For example, the function  $f_1$  is nested within the function  $f_2$ . The subscripting indicates visually the order in which the functions are applied, assuming a postfix ordering. Note that because there is a sequence to the application of functions, it is possible that one function may modify a memory variable that a subsequent function may use. For example,  $f_3$  could modify a global variable that is used by  $f_4$ . Also note that not all functions have a direct effect on the output of the node.

### B. Generic Neural Markup Language

```

<Network>
<InSource id='i1' signalType='Activation' />
<InSource id='i2' signalType='Activation' />
<InSource id='i3' signalType='Feedback' />
<OutTarget id='o1' signalType='Activation' />
<Neuron id='1'>
  <NeuronSpec> ... </NeuronSpec>
</Neuron>
<Neuron id='2'>
  <NeuronSpec> ... </NeuronSpec>
</Neuron>
...
<Connection source='i1' target='1'
  signalType='Activation' />
<Connection source='1' target='2'
  signalType='Activation' />
...
<Connection source='i3' target='2'
  signalType='Feedback' />
</Network>

```

Fig 3. High-level GNML format

The basic GNML format is illustrated in Figure 3. At the highest level, the network is represented as a list of the input sources and output targets of the network, each with a unique identifier and the type of signal they provide, followed by a list of the neurons, each with a unique identifier and a nested specification of its functionality (i.e., `<NeuronSpec>` block), and finally a list of all the connections in the network, where each connection indicates its source and target by their unique identifier and indicates the type of signal it transmits.

This format is sufficient to define any arbitrary graph over the neurons, input sources and output targets, and enables each neuron, potentially, to have different internal behavior. The remaining complexity of GNML lies in how the internal functions and memory of the neurons are defined. Some of these will be introduced below in the discussion on the design of grammar productions within AGE.

### C. Attribute Grammar Design

The design of a grammar within AGE is fairly complex [23]. Space limitations preclude an in-depth description of all its features, but several core design principles are introduced.

1. Network $\rightarrow$ Multi-Layer	
i	Network.all_nodes = Multi-Layer.all_nodes
ii	Network.all_conns = Multi-Layer.all_conns
iii	Network.GNML = transform (Multi-Layer.all_nodes, Multi-Layer.all_conns)
a	Multi-Layer.id = "1.1"
2. Multi-Layer <sub>1</sub> $\rightarrow$ Multi-Layer <sub>2</sub> Layer	
i	Multi-Layer <sub>1</sub> .all_nodes = Multi-Layer <sub>2</sub> .all_nodes $\cup$ Layer.nodes
ii	Multi-Layer <sub>1</sub> .all_conns = Multi-Layer <sub>2</sub> .all_conns $\cup$ {id(Multi-Layer <sub>2</sub> .out_nodes) $\times$ id(Layer.nodes), "Activation"}
iii	Multi-Layer <sub>1</sub> .out_nodes = Layer.nodes
a	Multi-Layer <sub>2</sub> .id = Multi-Layer <sub>1</sub> .id + ".1"
b	Layer.id = Multi-Layer <sub>1</sub> .id + ".2"
3. Multi-Layer $\rightarrow$ Layer	
i	Multi-Layer.all_nodes = Layer.nodes
ii	Multi-Layer.all_conns = {}
iii	Multi-Layer.out_nodes = Layer.nodes
a	Layer.id = Multi-Layer.id + ".1"
4. Layer <sub>1</sub> $\rightarrow$ Neuron Layer <sub>2</sub>	
i	Layer <sub>1</sub> .nodes = Layer <sub>2</sub> .nodes 4 [Layer <sub>1</sub> .id + ".1", Neuron.spec]
a	Layer <sub>2</sub> .id = Layer <sub>1</sub> .id + ".2"
5. Layer $\rightarrow$ Neuron	
i	Layer.nodes = [Layer.id + ".1", Neuron.spec]
where	
Each connection is a pair of nodes: (source id, destination id)	
Neuron.spec = detailed specification of neuron's behaviour	
transform() performs final conversion of node and connection sets to complete GNML format	

Fig 4. Attribute grammar productions within AGE

Figure 4 presents an AGE grammar that generates networks with an arbitrary number of layers, where each layers has an arbitrary number of neurons. A standard AGE notation is used in the productions. Each grammar symbol is given a name suggestive of its use (e.g., 'Network'). Each attribute of a symbol is also given a suggestive name and is indicated in lower-case italics and in reference to the symbol (e.g., Network.all\_nodes). Each production consists of a context-free portion (e.g., 1. Network  $\rightarrow$  Multi-Layer), attribute evaluation rules for synthesized attributes, indicated by a lower-case roman numeral (e.g., i. Network.all\_nodes = Multi-Layer.all\_nodes), and attribute evaluation rules for inherited attributes, indicated by an lower case letter (e.g., a. Multi-Layer.id = "1.1"). Within an attribute grammar,

information may flow both up (synthesized) and down (inherited) the parse tree when evaluating attributes. For instance, the *id* attribute passes information down the tree to enable each neuron to be assigned a unique identifier, while the *all\_nodes* and *all\_conns* attributes pass neuron and connection details up the tree to produce a complete specification at the root of both the graph formed by the connections and the functionality of each neuron. This information is readily converted to a complete GNML specification through a simple transformation operation.

The design of grammar productions within AGE involves several levels of detail. At the highest-level, there is the key question of what neural properties, both structural and behavioral will need to be incorporated, and how those will be manipulated. This typically is reflected in the choice of grammar symbols and context-free productions that are generated. For instance, production 2 represents a graphical manipulation in which a layer is added in a fully connected manner on top of a set of layers, as illustrated in Figure 5.

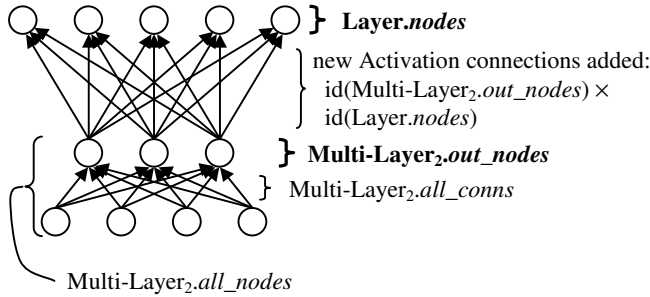


Fig 5. Graph manipulation corresponding to production 2

At an intermediate level of detail, there is the key question of how to translate the desired effect of the grammar production into appropriate manipulations of the attributes. This involves incorporating set and string manipulations into the attribute evaluation rules to compose appropriate GNML descriptions. Finally, at the lowest level of detail, there is the question of what terminals to use. Within AGE, terminal symbols will often represent GNML template strings in which a basic GNML definition of a neuron component is provided, such as a function or memory variable. A key aspect of the design of an AGE grammar is that the terminal may fully or partially define a neuron component. If the definition is partial, then specific keywords are included in the template so that the missing information may be added through the use of string substitution.

For example, Figure 6(a) presents grammar productions that compose a neuron using string substitution and concatenation operations. The basic template for a neuron is presented within an attribute of a terminal symbol (i.e., *neuron.template*). The template is in XML format, as in Figure 6(d), and uses the `<NeuronSpec>` and `</NeuronSpec>` XML tags, as expected by the neural

interpreter. To avoid confusion with the notation used for grammar symbols (i.e., similar names), XML tags are presented italicized and in a different font. To complete the specification of the neuron, string substitution is performed upon the substitution keyword “SUBFUNS”; specifically, the keyword is replaced with a string representing a sequence of function specifications.

6. Neuron $\rightarrow$ neuron ( FunctionSequence )
i Neuron.spec = substitute(neuron.template, “SUBFUNS”, FunctionSequence.spec)
7. FunctionSequence <sub>1</sub> $\rightarrow$ FunctionSequence <sub>2</sub> Function
i FunctionSequence <sub>1</sub> .spec = concat (FunctionSequence <sub>2</sub> .spec, Function.spec)
8. FunctionSequence $\rightarrow$ Function
i FunctionSequence.spec = Function.spec (a)
9. Function $\rightarrow$ multiply (ReadParam <sub>1</sub> , ReadParam <sub>2</sub> )
i Function.spec = substitute( substitute( multiply.template, “SUBREAD1”, ReadParam <sub>1</sub> .spec), “SUBREAD2”, ReadParam <sub>2</sub> .spec)
10. ReadParam $\rightarrow$ Function
i ReadParam.spec = Function.spec (b)
11. ReadParam $\rightarrow$ Memory
i ReadParam.spec = Memory.spec
12. Memory $\rightarrow$ general-memory ( learn-rate )
i Memory.spec = substitute(general-memory.template, “SUBVAR”, learn-rate.spec) (c)
where
neuron.template = <code>&lt;NeuronSpec&gt;</code> SUBFUNS <code>&lt;/NeuronSpec&gt;</code>
multiply.template = <code>&lt;FunctionSpec nativeCode=multiply</code> numReadParams=' 2' > <code>&lt;ReadParam&gt;</code> SUBREAD1 <code>&lt;/ReadParam&gt;</code> <code>&lt;ReadParam&gt;</code> SUBREAD2 <code>&lt;/ReadParam&gt;</code> <code>&lt;/FunctionSpec&gt;</code>
general-memory.template = <code>&lt;GeneralMemorySpec&gt;</code> SUBVAR <code>&lt;/GeneralMemorySpec&gt;</code>
learn-rate.spec = <code>&lt;VariableSpec name=' learnRate' /&gt;</code>
concat (a,b,...n): Given n strings, return the string resulting from concatenating all the strings in order (a + b + ... + n)
substitute(A,b,c): Given three strings A, b and c, returns the string formed by substituting all occurrences of b in A with the string c (d)

Fig 6. Productions that compose neuron specification using string operations

Figure 6(b) presents grammar productions in which a function template is completed through the use of string substitution. The *multiply.template* attribute contains the template XML string, as in Figure 6(d). A single production (i.e., production 9) specifies that two parameters are needed, and completes the neural function specification by substituting the XML specification for each parameter into the template string. The first parameter string is substituted for “SUBREAD1” and the second for “SUBREAD2”. Production 10 enables function nesting.



issues may be inherent in the nature of the complex models that we were developing.

The capability of AGE to represent a variety of neural architectures was very readily exploited through the definition of sub-grammars to define specific neural modules and the use of a variety of productions to integrate those modules in different ways. As such it was a useful tool for exploring novel combinations of neural architectures. Further, variations in grammar productions enabled us to analyze the effects of slight changes upon the structures developed and the network behaviors demonstrated.

Overall, we observed that the process of designing AGE and GNML together provided a very valuable exercise in understanding underlying neural principles. As more neural components and principles were incorporated, fewer changes to the approach were required. The process facilitated a clear focus on higher-level design and research issues, such as examining the interactions between different learning mechanisms, the automatic exploration of task decompositions and the selection of appropriate structural components to specialize on those subtasks and the design of new neural functions from basic elements

#### V. CONCLUSION

There are important synergies between the approaches of neural network specification languages and genetic encoding of neural network models. An important research need for improved tools for modeling neural networks may be addressed by a novel approach in which a grammar is used to define high-level characteristics of a neural network architecture and to generate a neural network specification that is platform and simulation independent. We have presented one specific technique based on this approach that uses attribute grammars and have summarized the lessons learned. The approach has demonstrated merit for the abstract representation, manipulation and design of broad families of neural network architectures.

#### ACKNOWLEDGMENTS

This work was funded in part under DARPA contract F30602-98-C-0168.

#### REFERENCES

- [1] Sarle, W.S. *Frequently Asked Questions on Neural Networks*, 2002. <ftp.sas.com/pub/neural/FAQ.html>
- [2] Demuth, H. and Beale, M. *Neural Network Toolbox: User's Guide, Version 4*. The Mathworks, Inc, 2003. [www.mathworks.com](http://www.mathworks.com)
- [3] Leighton, R.R. *The Aspirin/MIGRANES Neural Network Software Users manual*, MITRE Corp., 1992.
- [4] Dawson, C.K. O'Reilly, R.C. and McClelland, J.L. *PDP++ Software Users Manual, Ver. 3.0*, 2003. [www.cnbc.cmu.edu/Resources/PDP++](http://www.cnbc.cmu.edu/Resources/PDP++)
- [5] Fischer, I., Hennecke, F., Bannes, C. and Zell, A. *JavaNNS Java Neural Network Simulator User Manual, Version 1.1*, University of Tubingen Wilhelm-Schickard-Institute for Computer Science, Department of Computer Architecture, 2001.

- [6] NeuroSolutions, Version 4.2, NeuroDimension, Inc. [www.nd.com](http://www.nd.com)
- [7] *GENESIS Reference Manual*, Version 2.2, 2001. [www.genesis-sim.org/GENESIS](http://www.genesis-sim.org/GENESIS)
- [8] Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley, 1990.
- [9] Kock, G. and Serbedzija, N.B. "Artificial neural networks: From compact descriptions to C++," *International Conference on Artificial Neural Networks*, ICANN'94, p. 1372-1375, 1994.
- [10] Strey, A. "EpsilonNN - A Tool for the Abstract Specification and Parallel Simulation of Neural Networks", *Systems Analysis - Modelling - Simulation (SAMS)*, Gordon and Breach, 34(4), 1999.
- [11] Wiklicky, H. "Graphical design and description of neural networks," *IJCNN International Joint Conference on Neural Networks*, Baltimore, IEEE, p. 459- 464, 1992.
- [12] Rubtsov D. and Butakov S., "Application of XML for neural network exchange," *Computer Standards and Interfaces*, 24(4), p. 311-322, 2002.
- [13] Heistermann, J. "The application of a genetic approach as an algorithm for neural networks," *Parallel Problem Solving from Nature, First Workshop*. Berlin: Springer-Verlag, p. 297-301. 1990.
- [14] Miller, G.F., Todd, P.M. and Hegde, S.U. "Designing neural networks using genetic algorithms," *Third Int.. Conf. on Genetic Algorithms and Their Applications*. Morgan Kaufmann, p. 379-384. 1989.
- [15] Polani, D. and Uthmann, T. "Adaptation of Kohonen feature map topologies by genetic algorithms," *Parallel Problem Solving from Nature 2*. R. Männer and B. Manderick (Eds.), Elsevier, p. 421-429, 1992.
- [16] Schaffer, J.D., Caruana, R.A. and Eshelman, L.J. "Using genetic search to exploit the emergent behavior of neural networks," *Physica D*, 42, p. 244-248, 1990.
- [17] Jacob, C. "Genetic L-system programming," *Parallel Problem Solving from Nature III*, Lecture Notes in Comp. Sci. 866, Berlin: Springer, p. 334-343, 1994.
- [18] Kitano, H. "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, 4, p. 461-476, 1990.
- [19] Gruau, F. "Automatic definition of modular neural networks," *Adaptive Behavior*, 3(2), p. 151-183, 1995.
- [20] Bengio, S., Bengio, Y., and Cloutier, J. "Use of genetic programming for the search of a new learning rule for neural networks," *First Conference on Evolutionary Computation*, p. 324-327. 1994.
- [21] Hussain, T.S. and Browse, R.A. "Network generating attribute grammar encoding," *1998 IEEE International Joint Conference on Neural Network*, p. 431-436, 1998.
- [22] Hussain, T.S. and Browse, R.A. "Evolving neural networks using attribute grammars," *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, p. 37-42, 2000.
- [23] Hussain, T.S. *Attribute Grammar Encoding of the Structure and Behaviour of Artificial Neural Networks*. Ph.D. Thesis, Queen's University, 2003.
- [24] Knuth, D. E. (1968) "The semantics of context-free languages," *Mathematical Systems Theory*, 2, p.127-145.
- [25] Bray, T., Paoli, J., Sperberg-McQueen, C.M. and Maler, E. Extensible Markup Language (XML) 1.0 (Second Edition), World Wide Web Consortium (W3C) Technical Report: Recommendation Oct. 6, 2000.
- [26] Montana, D.J. (1993). "Strongly typed genetic programming," *BBN Tech Report 7866*.