A Descriptive Encoding Language for Evolving Modular Neural Networks

Jae-Yoon Jung and James A. Reggia

Department of Computer Science, University of Maryland,
College Park, MD 20742, USA
{jung, reggia}@cs.umd.edu

Abstract. Evolutionary algorithms are a promising approach for the automated design of artificial neural networks, but they require a compact and efficient genetic encoding scheme to represent repetitive and recurrent modules in networks. Here we introduce a problem-independent approach based on a human-readable descriptive encoding using a high-level language. We show that this approach is useful in designing hierarchical structures and modular neural networks, and can be used to describe the search space as well as the final resultant networks.

1 Introduction and Background

Neuroevolution refers to the design of artificial neural networks using evolutionary computation methods. It involves searching through a space of weights and/or architectures without substantial human intervention, trying to obtain an optimal network for a given task. The evaluation (fitness) and improvement of each network is based on its overall behavior (not just on its performance, but also on other properties of the network), and this makes neuroevolution a promising method for solving complex problems involving reinforcement learning [5,20] or designing recurrent networks [14].

An important research issue in neuroevolution is determining how to best encode possible solution networks as developmental "programs" forming the genotype in a compact and efficient manner. In early work in this field, Kitano encoded a set of developmental rules as a *graph generation grammar* having the form of a matrix [9,10]. Such an approach is developmental in nature because the information (e.g., a set of ordered rules and parameters) stored in the genotype describes a way of "growing" the phenotype. Constructing a network connectivity matrix begins with the initial start symbol in a chromosome, and rules are then applied to replace each non-terminal symbol in the chromosome with a 2x2 matrix of symbols, until there are all terminal symbols. Other well known but somewhat different approaches include Gruau's *cellular encoding*, where each rule defines a transformation or modification of a cell and the rules constitute a tree structure such that the order of execution for each rule is specified [3,4]; edge encoding [15]; and geometry-based cellular encoding [11,12].

Also of relevance to the work described here is past research that has proposed layer-based, parametric encoding schemes in which the basic unit of network

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 519–530, 2004.

architecture is a *set* of nodes (i.e., a layer) and network properties such as layer size and learning rates are encoded in the chromosomes as parameters that are altered during the evolutionary process [6,7,16,19].

Many of the encoding schemes above are very difficult to apply for designing large networks due to the scalability problem and their procedural nature, and are based on specific problem-dependent assumptions (e.g., the number of hidden nodes is manually selected [14]). Here we present an encoding scheme which appears to address these limitations. A layer-based, hierarchical architecture representation in our approach enables a high-level specification of multi-modular networks, and we let users incorporate their domain knowledge and restrictions on the types of networks evolved by explicitly choosing an appropriate set of network properties and their legal values. Thus our system does not a priori restrict whether architecture, layer sizes, learning method, etc. form the focus of evolution as many previous approaches have done, but instead allows the user to select which aspects of networks are to evolve. Furthermore, our approach is analogous to the abstraction process used in contemporary programming, in the sense that users write a text file specifying the problem to solve using a given high-level language. Our system then parses this file and searches for a solution within the designated search space, and finally it produces the results as another human readable text file. Thus we believe that our approach facilitates automated design of large scale neural networks covering a wide range of problem domains, not only because of its encoding efficiency, but also because it increases human readability and understandability of the initial environment specification and the final resultant networks.

2 Descriptive Encoding Methodology

Our encoding scheme is an extension of both the grammatical and the parametric encoding methods described above. Modular, hierarchical structure is essential when the size of the resultant neural network is expected to be large, since monolithic networks can behave irregularly as the network size becomes larger. Moreover, there is substantial evidence that a basic underlying neurobiological unit of cognitive function is a region (layer), e.g., in cerebral cortex [8], which strengthens the argument that hierarchical structure of layers should be the base architecture of any functional unit. Parametric encoding can also reduce the complexity of a genotype when there is a regular pattern in the network features, and opens the possibility for users to specify a set of appropriate network features according to given problem requirements.

We refer to our approach as a *descriptive encoding* since it enables users to "describe" the target space of neural networks to be considered in a natural, non-procedural human readable format. A user writes a text file like the ones later in this paper to specify sets of layers with appropriate properties, their legal evolvable values, and inter-layer connectivity. This input description does *not* specify individual connections, nor their weights. The specification of legal values affects the range of valid genetic operations. In other words, a description file

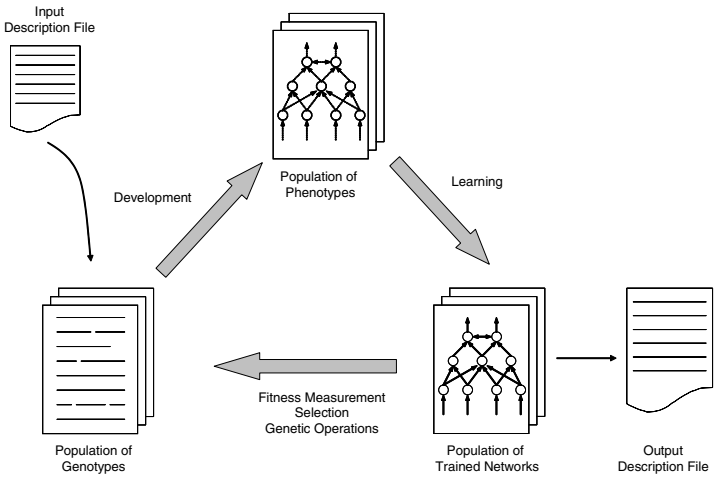


Fig. 1. The development, learning, and evolution procedure used in our system. The input description file (upper left) is a human-written specification of the class of neural networks to be evolved (the space to be searched) by the evolutionary process; the output description file (lower right) is a human-readable specification of the best specific networks obtained.

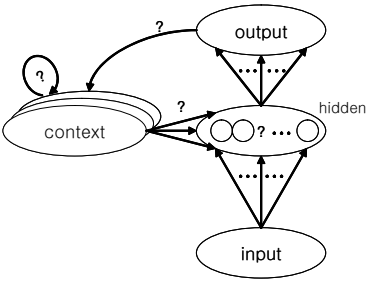
specifies the initial population and environment variables, and confines the search space of genetic operators throughout the evolutionary process. The evolutionary process in our system involves an initialization step plus a repeated cycle of three stages, as in Figure 1. First, the text description file prepared by the user is parsed and an initial random population of chromosomes (genotypes) is created within the search space represented by the description (left part of Figure 1). In the Development stage, a new population of realized networks (phenotypes) is created or "grown" from the genotype population. Each phenotype network keeps actual and specific nodes, connection weights, and biases. The Learning stage involves training each phenotype network if the user specifies one or more learning rules in the description file, making use of an input/output pattern file. Evolutionary computation is often less effective for local, fine tuning tasks [21], so we adopt neural network training methods. After the training stage, each individual network is evaluated according to user-defined fitness criteria and genetic operators are applied to the genotypes. Currently, fitness criteria may reflect both network performance (e.g., mean squared error) and a penalty for a large network (e.g., total number of nodes), or other measures.

2.1 Explicitly Incorporating Domain Knowledge

When searching for an optimal neural network using evolutionary computation methods, a network designer usually wants to restrict the architecture, learning rules, etc. to some proper subset of all possible models. Thus, many problem

```
[SEQUENCE
  [LAYER input  [SIZE 45] [CONNECT hidden]
                [LEARN_RULE [EVOLVE bp rprop]]]
  [LAYER hidden [SIZE [EVOLVE 1 45]] [CONNECT output]
                [LEARN_RULE [EVOLVE bp rprop]]]
  [LAYER output [SIZE 45] [CONNECT [EVOLVE context]]
                [CONNECT_RADIUS 1]]
  [SEQUENCE context
    [LAYER con [NUMBER  [EVOLVE 1 5]] [SIZE 45]
               [CONNECT [EVOLVE con hidden]]]]
]
```

(a)



(b)

Fig. 2. (a) Part of a description file; other information, such as details of the evolutionary process, is not shown. (b) An illustration of the corresponding class of recurrent networks that are described in (a). In effect, the description file specifies the search space to be used by the evolutionary process, while simultaneously providing a tree structure to be used by genetic operators such as crossover and mutation.

specific constraints need to be applied in creating the initial population and maintaining it within a restricted subspace of the space of all neural networks. For example, the range of architectures and valid property values for each individual network in the initial population will depend upon the specific problem being addressed. While such constraints and initialization procedures have been treated implicitly in previous approaches, our encoding scheme permits them to be described in a compact and explicit manner.

Figure 2 illustrates an example description written in our language for evolving a recurrent network, motivated by [17]. Each semantic block, enclosed in brackets [$\cdots$], starts with a type identifier followed by an optional name and a list of properties (a much simplified grammar for part of our encoding language is given in the Appendix to make this more precise). A network contains other (sub)networks and/or layers recursively, and a network type identifier (SEQUENCE, PARALLEL, or COLLECTION) indicates the arrangement of the subnetworks contained in this network. We define a layer as a set (sometimes one or two dimensional, depending on the problem) of nodes that share the same properties, and it is the basic module of our network representation scheme. For example, the description in Figure 2a indicates that a sequence of four types of layers are to be used: input, hidden, output, and con layers. Properties fill in the details of the network architecture (e.g., layer size and connectivity) and specify other network features including learning rules and activation dynamics.

Most previous neuroevolution research has focused a priori on some limited number of network features (e.g., network weights, number of nodes in the hidden layer) assuming that the other features are fixed, and this situation has prevented neuroevolution models developed by researchers from being used more widely in different environments. To overcome this limitation, we let users decide which properties are necessary to solve their problems, and what factors should

be evolved, from a set of supported properties that include architectures, activation dynamics, and learning rules. Unspecified properties may be replaced with default values and are treated as being fixed after initialization. For example, in Figure 2a, the input layer has a fixed number of 45 nodes and is connected to the hidden layer, while the single hidden layer has a SIZE within the range 1 to 45. The EVOLVE attribute indicates that the hidden layer's size will be randomly selected initially and is to be modified within the specified range during the evolution process. Note that the learning rules to be used for connections originating from both input and hidden layers are also declared as an evolvable property. After processing input patterns from the hidden layer, the output layer propagates its output to the layers in the context network, and the CONNECT_RADIUS property defines one-to-one connectivity in this case. Since the number of layers in the context network may vary from 1 to 5 (i.e., LAYER con has an evolvable NUMBER property), this output connectivity can be linked to any of these layers that were selected in a random manner during the evolution process. Finally, the layers in the context network are arranged as a sequence, and are connected to the hidden layer or themselves. Figure 2b depicts the corresponding search space schematically for the description of Figure 2a, and the details of each individual genotype (shown as question marks in the picture) will be assigned within this space at the initialization step and forced to remain within this space during the evolution process. Since the genotype structure is a tree, genetic operators used in GP [13] can be easily applied to them.

3 Results with the Parallel Multiple XOR Problem

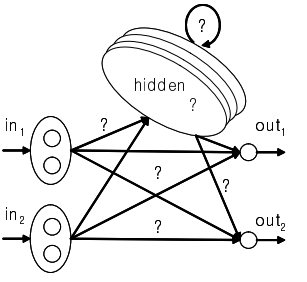
We use a parallel multiple exclusive-or (XOR) problem that currently runs on our system to illustrate the transformation of a description file into the results of an evolutionary process. The parallel multiple XOR problem is defined by a training data set where the correct result of each output node is the XOR of a corresponding separate pair of input nodes, independent of the values of the other output nodes. Of course, the evolutionary process has no a priori knowledge that this is the desired behavior; it is only evident from the training data. Further, this problem is difficult in the sense that there are apparent partial relationships between unrelated input and output nodes that must be identified without a priori domain knowledge. Also, since individual XOR problems are not linearly separable, the evolutionary process must discover that at least one hidden layer is necessary. A minimal network with no hidden layer cannot be the solution, although it is legal.

3.1 Encoding Details

The description file in Figure 3a defines a parallel dual-XOR problem with four input nodes, where layers in_1 and out_1 form one XOR gate, while layer in_2 is paired with out_2 to form another one. This description specifies both the initial networks to be created and the search space. It indicates that the desired

```
[SEQUENCE dual_xor
  [PARALLEL input
    [LAYER in  [NUMBER 2][SIZE 2]
               [CONNECT [EVOLVE hidden output]]]]
  [COLLECTION hidden
    [LAYER hid [NUMBER  [EVOLVE 0 10]]
               [SIZE    [EVOLVE 1  5]]
               [CONNECT [EVOLVE hidden output]]]]
  [PARALLEL output
    [LAYER out [NUMBER 2][SIZE 1]]]]
```

(a)



(b)

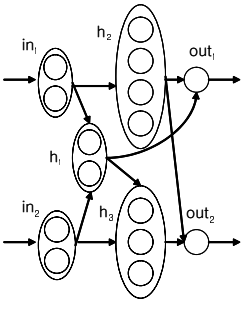
Fig. 3. Initial description file (a) and sketch of the space of networks to be searched (b) for a simple parallel dual-XOR problem with four input nodes and two output nodes.

overall structure is a sequential network named dual_XOR consisting of two input layers in parallel, a set (or "COLLECTION") of zero to 10 hidden layers, and two single-node output layers. The NUMBER statements assign the range of how many layers of each type may be created with the same properties in the network. So the description for the input layer is equivalent (except for optional layer names) to specifying this:

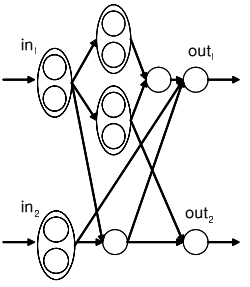
```
[LAYER in1 [SIZE 2] [CONNECT [EVOLVE hidden output]] ]
[LAYER in2 [SIZE 2] [CONNECT [EVOLVE hidden output]] ]
```

The CONNECT property in the input layers descriptor indicates that nodes in each input layer may evolve to connect to hidden layers, output layers, neither or both. The COLLECTION description indicates that networks can evolve to have 0 to 10 hidden layers, each with 1 to 5 nodes, and that they can be connected arbitrarily to themselves and to the output layers. The EVOLVE attributes listed here indicate that the connections from input layers to hidden and/or output layers, the number and size of hidden layers, and the connections from hidden layers to other hidden layers and output layers, are all evolvable. These combinations are randomly and independently decided at the initialization step and enforced by genetic operators throughout the evolution process.

Each chromosome created from this description stores a representation of an architecture in the form of a tree, as well as other network features as embedded parameters (properties) in the tree. This hierarchical description of the network architecture has some benefits over a linear list of layers in previous layer-based encoding schemes, since it directly maps the topology of a network into the representation. These benefits are: 1) it enables crossover operation on a set of topologically neighboring layers, which was not possible with point crossover operators; 2) functionally separated blocks can be easily specified and identified in a large scale, multi modular network; and 3) reusable subnetworks can be defined to address the scalability problem (e.g., like ADFs in GP [13]). Figure 4a and b illustrate two example networks automatically generated from the description



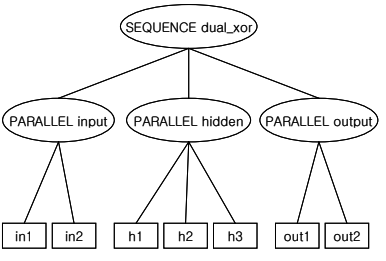
(a)



(b)

```
[SEQUENCE dual_xor
  [PARALLEL input
    [LAYER in1 [SIZE 2][CONNECT h1 h2]]
    [LAYER in2 [SIZE 2][CONNECT h1 h3]]]
  [PARALLEL hidden
    [LAYER h1  [SIZE 2][CONNECT h3 out1]]
    [LAYER h2  [SIZE 4][CONNECT out1 out2]]
    [LAYER h3  [SIZE 3][CONNECT out2]]]
  [PARALLEL output
    [LAYER out1 [SIZE 1]]
    [LAYER out2 [SIZE 1]]]
]
```

(c)



(d)

Fig. 4. (a),(b) Examples of neural network architectures randomly created from the description during initialization. Input and output layers are the same, but the number of hidden layers and their connections are quite different and specific now. Arrows indicate sets of connections between layers (i.e., not individual node-to-node connections). (c) The chromosome description of the network illustrated in (a), as it would be written in our descriptive language. This is *not* a description file written by the user, but is automatically generated from that description file. Note that no EVOLVE attributes are present, for example. (d) Top part of the tree-like structure of the genotype in (c), making it directly usable by GP operators. Each rectangle designates a layer.

file of Figure 3a; they show different numbers of layers and topologies. Figure 4c shows the corresponding chromosome or genotype structure of one of these, the network in Figure 4a. Note that the overall structure is the same with the initial description in Figure 3a, but the COLLECTION hidden network has been replaced with a PARALLEL network with three layers and each property has a fixed value (i.e., the EVOLVE attributes are gone). Figure 4d shows the tree like structure of this genotype, making it amenable to standard GP operators.

3.2 The Evolutionary Procedure

The description file created by the user not only specifies the search space, but also sets a variety of parameter values that influence the evolutionary process. Figure 5 shows the description for the learning and evolutionary parameters used to solve the parallel dual XOR problem. This is also a part of the description

```
[TRAINING
  [TRAIN_DATA "./inout_pattern.txt"] [MAX_TRAIN 100]]
[EVOLUTION
  [FITNESS weighted_sum] [ALPHA 0.5] [BETA 0.2] [GAMMA 0.2]
  [SELECTION tournament] [TOURNAMENT_POOL 3] [ELITISM 0]
  [MUTATION_PROB 0.7]    [CROSSOVER_PROB 0.0]
  [MAX_GENERATION 50]    [MAX_POPULATION 50]
]
```

Fig. 5. Training and evolutionary parameters for a parallel XOR problem.

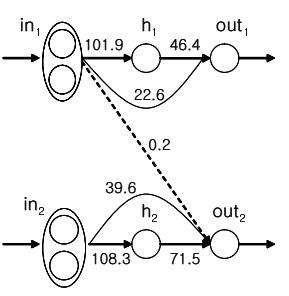
file following the network description, providing users with a systematic way to control the training and evolutionary procedure. In the Learning stage, each phenotype network is trained for 100 epochs with a default (as it is not specified in the description) backpropagation algorithm (RPROP [18]) plus the input/output pattern file specified in the TRAIN_DATA property (recurrent connections are deleted after the mutation operation). Currently, the fitness value of each phenotype network is calculated as a weighted sum of three reciprocally normalized criteria: mean squared error (MSE, e), total number of network nodes (n), and total number of layer-to-layer connections (c) (ultimately, we wish to allow the user to specify more general fitness functions). These three criteria are weighted with coefficients α, β, and γ which the user assigns (Figure 5). MSE reflects the output performance of the network, and the other two measures are adopted as a penalty for larger networks. Currently connections are counted on a layer-to-layer basis, not on a node-to-node basis, since the latter may be correlated or proportional to the total number of nodes. More specifically, the fitness value of a network is:

$$fit = \alpha \cdot \left(\frac{e_{max} - e}{e_{max} - e_{min}} \right) + \beta \cdot \left(\frac{n_{max} - n}{n_{max} - n_{min}} \right) + \gamma \cdot \left(\frac{c_{max} - c}{c_{max} - c_{min}} \right) \quad (1)$$

where $x_{min}(x_{max})$ denotes the minimum (maximum) value of criterion x among the population. Although this simple weighted sum was sufficient for the XOR problem, other multi-objective optimization methods [2] can be easily applied. Tournament selection with tournament size 3 is specified by the user for selecting candidate individuals for the next generation. Selection is based on the trained network fitness values, while genetic operations are done with genotype trees. The fitness value of each network is compared with that of three other randomly selected networks. Then one fittest network is selected and the corresponding chromosome (genotype) is copied to the next generation. No individuals other than the tournament winners are inserted into the new population and no elitism is used. The mutation rate is 0.7 and no crossover is used. Operators can mutate layer size, direction of an existing inter-layer connection, and can add or delete a new layer or connection. The initial description file is implicitly used here to constrain the resultant network population to remain in the search space

```
[SEQUENCE dual_xor
  [PARALLEL input
    [LAYER in1 [SIZE 2][CONNECT [h1 out1 out2]]
    [LAYER in2 [SIZE 2][CONNECT [h2 out2]]]
  [PARALLEL hidden
    [LAYER h1  [SIZE 1][CONNECT out1]]
    [LAYER h2  [SIZE 1][CONNECT out2]]]
  [PARALLEL output
    [LAYER out1 [SIZE 1]]
    [LAYER out2 [SIZE 1]]]]
```

(a)



(b)

Fig. 6. (a) A typical separate channel network evolved for the dual XOR problem. Each input layer uses its own hidden node (separate layer), and a direct connection to the correct matching output node. (b) The average value of absolute weights on connections between each layer is specified. A dotted line between in_1 and out_2 shows persistent, non-optimal connectivity with very small weights.

specified by the user; i.e., mutation only occurs within the legal range specified in the description file. After these genetic operations, a new genotype population is generated and starts another cycle. The total number of generations in a simulation and population size are both 50.

3.3 Results of the Evolutionary Process

Given the above information, our system generated near-optimal networks that both solved the dual XOR problem and had a small number of nodes. One of the best, but commonly found, resultant networks that correctly solve the problem is depicted in Figure 6a, when the ratio of $\alpha = 0.5$, $\beta = 0.2$, and $\gamma = 0.2$ was used. Note that this text description is a part of the output file which is generated by the system, and still a legal description in our language. Since it is a specific network, no EVOLVE attributes are present. The final resultant output also contains various statistical data and network details including trained connection weights, as summarized in Figure 6b. This result clearly demonstrates that our system can identify the partial, internal relationships between input and output patterns present in the dual XOR problem and represent them within a modular architecture, without a priori information about this in the initial network description. Ignoring the dotted line connections which have a near zero weight values shown in Figure 6b, we can easily see that this is a near-optimal network for the dual XOR problem in terms of the number of network nodes and connections.

4 Encoding Properties

Our approach has some important encoding properties. It can represent recurrent network architectures, and is scalable with respect to node/connectivity changes. More specifically, the encoding approach we are using has:

- *Closure* : A representation scheme can be said to be closed if all genotypes produced are mapped into a valid set of phenotype networks [1]. First, every genotype at the initial step is decoded into a valid phenotype since the initial population of genotypes is based on the user-defined description. Next, our approach is closed with respect to mutation operators that change property values in a layer, since we only allow property values to be mutated within the legal ranges defined by users or the system. This is checked at runtime and any illegal mutation result is discarded with replacement by another mutation to keep the population size fixed. Although our encoding scheme is not closed with crossover operators on a grammar level, it can be constrained to be closed on a system level by adjusting invalid property values, according to the description file. For example, if the number of layers in a network becomes too large after a crossover operation, such a network may be deleted (or the whole network structure could be adjusted to maintain legal genotypes).
- *Completeness* : Our encoding scheme can be used to represent any recurrent neural network architecture. This can be easily seen from the fact that if we confine the size of each and every layer to be one node, our encoding scheme is equivalent to a direct encoding which specifies full connectivity on a node-to-node basis. Combined with the closure property, this property ensures that our encoding can safely replace the direct encoding or the equivalent encodings.
- *Scalability* : This property can be defined by how decoding time and genotype space complexity are affected by a single change in a phenotype [1]. Our encoding scheme takes $O(1)$ time and space in a node addition/deletion, since changing the number of nodes means just changing a parameter value in a property in the corresponding genotype, and node addition/deletion does not make substantial changes in time and space requirements during the genotype-phenotype mapping. In a similar way, a node-to-node connection addition/deletion in a phenotype will cost $O(1)$ space in genotype and $O(N+C)$ decoding time, as N denotes the total number of nodes in a network, and C denotes the total number of layer-to-layer connections. If a connection is deleted in a phenotype, it will split the corresponding source and target layers since nodes in these layers do not share connectivity anymore, but this split is equivalent to deleting a node in both layers plus creating two single-node layers, which will cost $O(1)$ space (assuming a constant layer size) and $O(N+C)$ additional decoding time. In general, our scheme is $O(1)$ scalable with respect to nodes and $O(N+C)$ scalable with respect to connectivity.

5 Discussion

In this paper we have introduced a problem-independent evolving neural network system based on an encoding scheme and a high-level description language. Our approach can efficiently represent the hierarchical structure of multi-layer neural networks, which is a desired property for designing large scale networks. The tree structured encoding scheme used in our approach is amenable to GP operators, and we have shown that it is scalable and can be mapped into a valid set of recurrent phenotype networks. We have demonstrated with the parallel XOR problem that our system can identify relationships between input and output patterns and incorporate them into an optimal architecture. The use of a description file provides users with a systematic, non-procedural methodology for specifying the search space and evolution parameters, and the same language used for the network description can be used to produce a human readable final network description.

References

1. K. Balakrishnan and V. Honavar, Properties of genetic representations of neural architectures, *Proc. of the World Congress on Neural Networks*, pp. 807-813, 1995.
2. C. Coello, Evolutionary multi-objective optimization: a critical review, *Evolutionary Optimization*, R. Sarkar et al. (eds.), Kluwer, pp. 117-146, 2002.
3. F. Gruau, Neural network synthesis using cellular encoding and the genetic algorithm, *Ph.D. Thesis*, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France, 1994.
4. F. Gruau, Automatic definition of modular neural networks, *Adaptive Behavior*, 3:151-183, 1995.
5. F. Gruau, D. Whitley, and L. Pyeatt, A comparison between cellular encoding and direct encoding for genetic neural networks, *Genetic Programming 1996: Proc. of the First Annual Conference*, MIT Press, pp. 81-89, 1996.
6. S. A. Harp, T. Samad, and A. Guha, Toward the genetic synthesis of neural networks, *Proc. of 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 379-384, 1989.
7. T. S. Hussain and R. A. Browse, Network generating attribute grammar encoding, *Proc. of International Joint Conference on Neural Networks (IJCNN '98)*, 1:431-436, 1998.
8. E. R. Kandel and J. H. Schwartz, *Principles of Neural Science*, Elsevier, 1983.
9. H. Kitano, Designing neural networks using genetic algorithms with graph generation system, *Complex Systems*, 4:461-476, 1990.
10. H. Kitano, Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physica D*, 75:225-238, 1994.
11. J. Kodjabachian and J. A. Meyer, Evolution and development of control architectures in animats, *Robotics and Autonomous Systems*, 16:161-182, 1995.
12. J. Kodjabachian and J. A. Meyer, Evolution and development of modular control architectures for 1-D locomotion in six-legged animats, *Connection Science*, 10:211-254, 1998.
13. J. R. Koza, *Genetic Programming II*, MIT Press, 1994.

14. K. W. C. Ku, M. W. Mak, and W. C. Siu, Adding learning to cellular genetic algorithms for training recurrent neural networks, *IEEE Trans. on Neural Networks*, 10(2):239-252, 1999.
15. S. Luke and L. Spector, Evolving graphs and networks with edge encoding: preliminary report, *Late Breaking Papers at the Genetic Programming 1996 Conference*, pp. 117-124, 1996.
16. M. Mandischer, Representation and evolution of neural networks, *Artificial Neural Nets and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele (eds.), Springer, pp. 643-649, 1993.
17. M. J. Radio, J. A. Reggia, and R. S. Berndt, Learning word pronunciations using a recurrent neural network, *Proc. of the International Joint Conference on Neural Networks (IJCNN-01)*, 1:11-15, 2001.
18. M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: the RPROP algorithm, *Proc. of the IEEE International Conference on Neural Networks*, 1:586-591, 1993.
19. W. Schiffmann, Encoding feedforward networks for topology optimization by simulated evolution, *Proc. of 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies (KES 2000)*, 1:361-364, 2000.
20. K. O. Stanley and R. Miikkulainen, Efficient reinforcement learning through evolving neural network topologies, *Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, Morgan Kaufmann, pp. 569-577, 2002.
21. X. Yao, Evolving artificial neural networks, *Proc. of the IEEE*, 87(9):1423-1447, 1999.

Appendix – A Simplified and Partial Encoding Grammar

```
<description>  := <network> <training> <evolution>
<network>      := [<net_type> <name> <sub_network>] | <layer>
<training>     := [TRAINING  <tr_prop_list>]
<evolution>    := [EVOLUTION <ev_prop_list>]
<net_type>     := SEQUENCE | PARALLEL | COLLECTION
<sub_network>  := <network> <sub_network> | <network>
<layer>        := [LAYER <name> <ly_prop_list>]
<ly_prop_list> := <ly_property> <ly_prop_list> | <ly_property>
<tr_prop_list> := <tr_property> <tr_prop_list> | <tr_property>
<ev_prop_list> := <ev_property> <ev_prop_list> | <ev_property>
<ly_property>  := [NUMBER     <value>] | [SIZE      <value>] | ...
<tr_property>  := [TRAIN_DATA <path> ] | [MAX_TRAIN <value>] | ...
<ev_property>  := [FITNESS    <name> ] | [SELECTION <name> ] | ...
<value>        := [EVOLVE <range_value>] | [<range_value>] |
                  [EVOLVE <fixed_value>] |  <fixed_value>
<range_value>  := <fixed_value> <range_value> | <fixed_value>
<fixed_value>  := <integer>  | <float> | <literals>
<name>         := <literals>
<path>         := ‘‘<name>’’
```