Towards the Evolution of Multi-Layered Neural Networks:
A Dynamic Structured Grammatical Evolution Approach

Filipe Assunção, Nuno Lourenço, Penousal Machado, Bernardete Ribeiro
CISUC, Department of Informatics Engineering,
University of Coimbra, Portugal
{fga,naml,machado,bribeiro}@dei.uc.pt

ABSTRACT

Current grammar-based NeuroEvolution approaches have several shortcomings. On the one hand, they do not allow the generation of Artificial Neural Networks (ANNs) composed of more than one hidden-layer. On the other, there is no way to evolve networks with more than one output neuron. To properly evolve ANNs with more than one hidden-layer and multiple output nodes there is the need to know the number of neurons available in previous layers. In this paper we introduce Dynamic Structured Grammatical Evolution (DSGE): a new genotypic representation that overcomes the aforementioned limitations. By enabling the creation of dynamic rules that specify the connection possibilities of each neuron, the methodology enables the evolution of multi-layered ANNs with more than one output neuron. Results in different classification problems show that DSGE evolves effective single and multi-layered ANNs, with a varying number of output neurons.

CCS CONCEPTS

•**Computing methodologies** → **Neural networks; Genetic programming;** *Supervised learning by classification;* •**Theory of computation** → Genetic programming;

KEYWORDS

NeuroEvolution, Artificial Neural Networks, Classification, Grammar-based Genetic Programming

1 INTRODUCTION

Machine Learning (ML) approaches, such as Artificial Neural Networks (ANNs), are often used to learn how to distinguish between multiple classes of a given problem. However, to reach near-optimal classifiers a laborious process of trial-and-error is needed to hand-craft and tune the parameters of ML methodologies. In the specific case of ANNs there are at least two manual steps that need to be

considered: (i) the definition of the topology of the network, i.e., number of hidden-layers, number of neurons of each hidden-layer, and how should the layers be connected between each other; and (ii) the choice and parameterisation of the learning algorithm that is used to tune the weights and bias of the network connections (e.g., initial weights distribution and learning rate).

Evolutionary Artificial Neural Networks (EANNs) or NeuroEvolution refers to methodologies that aim at the automatic search and optimisation of the ANNs' parameters using Evolutionary Computation (EC). With the popularisation of Deep Learning (DL) and the need for ANNs with a larger number of hidden-layers, NeuroEvolution has been vastly used in recent works [19].

The goal of the current work is the proposal of a novel Grammar-based Genetic Programming (GGP) methodology for evolving the topologies, weights and bias of ANNs. We rely on a GGP methodology because in this way we allow the direct encoding of different topologies for solving distinct problems in a plug-and-play fashion, requiring the user to set a grammar capable of describing the parameters of the ANN to be optimised. One of the shortcomings of previous GGP methodologies applied to NeuroEvolution is the fact that they are limited to the evolution of network topologies with one hidden-layer. The proposed approach, called Dynamic Structured Grammatical Evolution (DSGE) solves this constraint.

The remainder of the document is organised as follows: In Section 2 we detail Structured Grammatical Evolution (SGE) and survey the state of the art on NeuroEvolution; Then, in Sections 3 and 4, DSGE and its adaption to the evolution of multi-layered ANNs are presented, respectively; The results and comparison of DSGE with other GGP approaches is conducted in Section 5; Finally, in Section 6, conclusions are drawn and future work is addressed.

2 BACKGROUND AND STATE OF THE ART

In the following sub-sections we present SGE, which serves as base for the development of DSGE. Then, we survey EC approaches for the evolution of ANNs.

2.1 Structured Grammatical Evolution

Structured Grammatical Evolution (SGE) was proposed by Lourenço et al. [11] as a new genotypic representation for Grammatical Evolution (GE) [13]. The new representation aims at solving the redundancy and locality issues in GE, and consists of a list of genes, one for each non-terminal symbol. Furthermore, a gene is a list of integers of the size of the maximum number of possible expansions for the non-terminal it encodes; each integer is a value in the interval $[0, non\_terminal\_possibilities-1]$, where $non\_terminal\_possibilities$ is the number of possibilities for the expansion of the considered non-terminal symbol. Consequently, there is the need to pre-process

the input grammar to find the maximum number of expansions for each non-terminal (further detailed in Section 3.1 of [11]).

To deal with recursion, a set of intermediate symbols are created, which are no more than the same production rule replicated a pre-defined number of times (recursion level). The direct result of the SGE representation is that, on the one hand, the decoding of individuals no longer depends on the modulus operation and thus its redundancy is removed; on the other hand, locality is enhanced, as codons are associated with specific non-terminals.

The genotype to phenotype mapping procedure is similar to GE, with two main differences: (i) the modulus operation is not used; and (ii) integers are not read sequentially from a single list of integers, but are rather read sequentially from lists associated to each non-terminal symbol. An example of the mapping procedure is detailed in Section 3 of [11].

2.2 NeuroEvolution

Works focusing the automatic generation of ANNs are often grouped according to the aspects of the ANN they optimise: (i) evolution of the network parameters; (ii) evolution of the topology; and (iii) evolution of both the topology and parameters.

The gradient descent nature of learning algorithms, such as back-propagation, makes them likely to get trapped in local optima. One way to overcome this limitation is by using EC to tune the ANN weights and bias values. Two types of representation are commonly used: binary [21] or real (e.g., CoSyNE [6], Gravitational Swarm and Particle Swarm Optimization applied to OCR [4], training of deep neural networks [3]).

When training ANNs using EC the topology of the network is provided and is kept fixed during evolution. Nevertheless, approaches tackling the automatic evolution of the topology of ANNs have also been proposed. In this type of approaches for finding the adequate weights some authors use a off-the-shelf learning methodology (e.g., backpropagation) or evolve both the weights and the topology simultaneously.

Regarding the used representation for the evolution of the topology of ANNs it is possible to divide the methodologies into two main types: those that use direct encodings (e.g., Topology-optimization Evolutionary Neural Network [12], NeuroEvolution of Augmenting Topologies [17]) and those that use indirect encodings (e.g., Cellular Encoding [8]). In the first two works the genotype is a direct representation of the network, and in the latter a mapping procedure has to be applied to transform the genotype into an interpretable network. Focusing on indirect representations, in the next section we further detail grammar-based approaches.

2.3 Grammar-based NeuroEvolution

Over the last years several approaches applying GE to NeuroEvolution have been proposed. However, the works that focus the evolution of the networks topology are limited to generating one-hidden-layered ANNs, because of the difficulties in tracking the number of neurons available in previous layers.

Tsoulos et al. [20] and Ahmadizar et al. [1] describe two different approaches based on GE for the evolution of both the topology and parameters of one-hidden-layered ANNs. While the first evolves the topology and weights using GE, the latter combines GE with a Genetic Algorithm (GA): GE is applied to the evolution of the

```
<start>  ::= <float> (0)

<float>  ::= <first>.<second> (0)

<first>  ::= 0 (0)
         |   1 (1)
         |   2 (2)

<second> ::= <digit> <second> (0)
         |   <digit>            (1)

<digit>  ::= 0  (0)
         |   ...  (...)
         |   9    (9)
```

<start>	<float>	<first>	<second>	<digit>
[0]	[0]	[1]	[0, 0, 1]	[2, 5, 9]

Figure 1: Example of a grammar (top) and of a genotype of a candidate solution (bottom).

topology and the GA is used for searching the real values (i.e., weights and bias). The use of a GA to optimise the real values is motivated by the fact that GE is not suited for the evolution and tuning of real values. For that reason, Soltanian et al. [16] just use GE to optimise the topology of the network, and rely on the backpropagation algorithm to train the evolved topologies.

Although GE is the most common approach for the evolution of ANNs by means of grammars it is not the only one. Si et al. in [14] present Grammatical Swarm Neural Networks (GSNN): an approach that uses Grammatical Swarm for the evolution of the weights and bias of a fixed ANN topology, and thus, EC is used just for the generation of real numbers. In [9], Jung and Reggia detail a method for searching adequate topologies of ANNs based on descriptive encoding languages: a formal way of defining the environmental space and how should individuals be formed; the Resilient Backpropagation (RPROP) algorithm is used for training the ANNs. SGE has also been used for the evolution of the topology and weights of ANNs [2].

3 DYNAMIC STRUCTURED GRAMMATICAL EVOLUTION

Dynamic Structured Grammatical Evolution (DSGE) is our novel GGP approach. With the proposed methodology the gain is twofold: (i) all the genotype is used, i.e., while in GE and SGE the genotype encodes the largest allowed sequence, in DSGE the genotype grows as needed; and (ii) there is no need to pre-process the grammar in order to compute the maximum tree-sizes of each non-terminal symbol, so that intermediate grammar derivation rules are created. In the next sections we describe in detail the components that make these gains possible.

3.1 Representation

Each candidate solution encodes an ordered sequence of the derivation steps of the used grammar that are needed to generate a specific solution for the problem at hand. The representation is similar to the one used in SGE, with one main difference: instead of computing and generating the maximum number of derivations for each of the grammar's non-terminal symbols, a variable length representation is used, where just the number of needed derivations are encoded.

Algorithm 1 Random candidate solution creation.

1: **procedure** CREATE_INDIVIDUAL(grammar, max_depth, genotype, symbol, depth)
2:     expansion = randint(0, len(grammar[symbol])-1)
3:     **if** is_recursive(symbol) **then**
4:         **if** expansion in grammar.recursive(symbol) **then**
5:             **if** depth ≥ max_depth[symbol] **then**:
6:                 non_rec = grammar.non_recursive(symbol)
7:                 expansion = choice(non_rec)
8:     **if** symbol in genotype **then**
9:         genotype[symbol].append(expansion)
10:    **else**
11:        genotype[symbol] = [expansion]
12:    expansion_symbols = grammar[symbol][expansion]
13:    **for** sym in expansion_symbols **do**
14:        **if** not is_terminal(sym) **then**
15:            **if** symbol == sym **then**
16:                create_individual(grammar, max_depth, genotype, symbol, depth+1)
17:            **else**
18:                create_individual(grammar, max_depth, genotype, symbol, 0)

Consequently, there is no need to create intermediate symbols to deal with recursive rules.

To limit the genotype size, a maximum depth value is defined for each non-terminal symbol. Allowing different limits for each non-terminal symbol provides an intuitive and flexible way of constraining the search space by limiting, for instance, the maximum number of hidden-layers, neurons or connections.

Figure 1 represents an example grammar for the generation of real numbers in the $[0, 3[$ interval, along with the representation of a candidate solution encoding the number 1.259. The genotype is encoded as a list of genes, where each gene encodes an ordered sequence of choices for expanding a given non-terminal symbol, as a list of integers. The genotype to phenotype mapping is detailed in Section 3.3.

3.2 Initialisation

Algorithm 1 details the recursive function that is used to generate each candidate solution. The input parameters are: the grammar that describes the domain of the problem; the maximum depth of each non-terminal symbol; the genotype (which is initially empty); the non-terminal symbol that we want to expand (initially the start symbol is used); and the current sub-tree depth (initialised to 0). Then, for the non-terminal symbol given as input, one of the possible derivation rules is selected (lines 2-11) and the non-terminal symbols of the chosen derivation rule are recursively expanded (lines 12-18). However, when selecting the expansion rule there is the need to check whether or not the maximum sub-tree depth has already been reached (lines 3-5). If that happens, only non-recursive derivation rules can be selected for expanding the current non-terminal symbol (lines 6-7). This procedure is repeated until an initial population with the desired size is created.

3.3 Mapping Function

To map the candidate solutions genotype into the phenotype that will later be interpreted as an ANN we use Algorithm 2. The algorithm is similar to the one used to generate the initial population but, instead of randomly selecting the derivation rule to use in the expansion of the non-terminal symbol, we use the choice that is encoded in the individual's genotype (lines 12-22). During evolution the genetic operators may change the genotype in a way

---

Algorithm 2 Genotype to phenotype mapping procedure.

1: **procedure** MAPPING(genotype, grammar, max_depth, read_integers, symbol, depth)
2:     phenotype = ""
3:     **if** symbol not in read_integers **then**
4:         read_integers[symbol] = 0
5:     **if** symbol not in genotype **then**
6:         genotype[symbol] = []
7:     **if** read_integers[symbol] ≥ len(genotype[symbol]) **then**
8:         **if** depth ≥ max_depth[symbol] **then**
9:             generate_terminal_expansion(genotype, symbol)
10:        **else**
11:            generate_expansion(genotype, symbol)
12:    gen_int = genotype[symbol][read_integer[symbol]]
13:    expansion = grammar[symbol][gen_int]
14:    read_integers[symbol] += 1
15:    **for** sym in expansion **do**
16:        **if** is_terminal(sym) **then**
17:            phenotype += sym
18:        **else**
19:            **if** symbol == sym **then**
20:                phenotype += mapping(genotype, grammar, max_depth, read_integers, sym, depth+1)
21:            **else**
22:                phenotype += mapping(genotype, grammar, max_depth, read_integers, sym, 0)
23:    **return** phenotype

---

that requires a larger number of integers than the ones available. When this happens, the following genotype's repair procedure is applied: new derivation rules are selected at random and added to the genotype of the individual (lines 3-11). In addition to returning the genotype the algorithm also computes which genotype integers are being used, which will later help in the application of the genetic operators. Table 1 shows an example of the mapping procedure applied to the genotype of Figure 1.

3.4 Genetic Operators

To explore the problem's domain and therefore promote evolution we rely on mutation and crossover.

Mutation. is restricted to integers that are used in the genotype to phenotype mapping and changes a randomly selected expansion option (encoded as an integer) to another valid one, constraint to the restrictions on the maximum sub-tree depth. To do so, we first select one gene; the probability of selecting the i-th gene (p_i) is proportional to the number of integers of that non-terminal symbol that are used in the genotype to phenotype mapping (*read_integers*):

$$p_i = \frac{read\_integers_i}{\sum_{j=1}^{n} read\_integers_j},$$

where n is the total number of genes. Additionally, genes where there is just one possibility for expansion (e.g, <start> or <float> of the grammar of Figure 1) are not considered for mutation purposes. After selecting the gene to be mutated, we randomly select one of its integers and replace it with another valid possibility.

Considering the genotype of Figure 1, a possible result from the application of the mutation operator is [[0], [0], [2], [0, 0, 1], [2, 5, 9]], that represents the number 2.259.

Crossover. is used to recombine two parents (selected using tournament selection) to generate two offspring. We use one-point crossover. As such, after selecting the cutting point (at random) the genetic material is exchanged between the two parents. The choice

Table 1: Example of the mapping procedure. Each row represents a derivation step. The list of genes represents the integers needed for expanding start, float, first, second and digit, respectively.

Derivation step	Integers left
<start>	[[0], [0], [1], [0, 0, 1], [2, 5, 9]]
<float>	[[], [0], [1], [0, 0, 1], [2, 5, 9]]
<first>.<second>	[[], [], [1], [0, 0, 1], [2, 5, 9]]
1.<second>	[[], [], [], [0, 0, 1], [2, 5, 9]]
1.<digit><second>	[[], [], [], [0, 1], [2, 5, 9]]
1.2<second>	[[], [], [], [0, 1], [5, 9]]
1.2<digit><second>	[[], [], [], [1], [5, 9]]
1.25<second>	[[], [], [], [1], [9]]
1.25<digit>	[[], [], [], [], [9]]
1.259	[[], [], [], [], []]

of the cutting point is done at the gene level and not at the integers level, i.e., what is exchanged between parents are genes and not the integers.

Given two parents [[0], [0], [2], | [0,0,1], [2,5,9]] and [[0], [0], [1], | [0,0,0,1], [1,0,2,4]], where | denotes the cutting point, the generated offspring would be [[0], [0], [2], [0,0,0,1], [1,0,2,4]] and [[0], [0], [1], [0,0,1], [2,5,9]].

3.5 Fitness Evaluation

To enable the comparison of the evolved ANNs with the results from a previous work [2] the performance is measured as the Root Mean Squared Error (RMSE) obtained while solving a classification task. In addition, for better dealing with unbalanced datasets, this metric considers the RMSE per class, and the resulting fitness function is the multiplication of the exponential values of the multiple RMSEs per class, as follows:

$$\text{fitness} = \prod_{c=1}^{m} exp\left(\sqrt{\frac{\sum_{i=1}^{n_c}(o_i - t_i)^2}{n_c}}\right),$$

where m is the number of classes of the problem, n_c is the number of instances of the problem that belong to class c, o_i is the confidence value predicted by the evolved network, and t_i is the target value. This way, higher errors are more penalised than lower ones, helping the evolved networks to better generalise to unseen data.

4 EVOLUTION OF MULTI-LAYERED ANNs

Figure 2 shows the grammar that was used in [2], which is similar to those used in other works focusing the grammar-based evolution of ANNs [1, 16, 20]. The rationale behind the design of this grammar is the evolution of networks composed of one hidden-layer, where only the hidden-neurons as well as the weights of the connections from the input and to the output neurons are evolved.

Three major drawbacks can be pointed to the previous grammatical formulation: (i) it only allows the generation of networks with one hidden-layer; (ii) the output neuron is always connected to all neurons in the hidden-layer; and (iii) there is no way to define multiple output nodes, at least one that reuses the hidden-nodes, instead of creating new ones.

These drawbacks are related to limitations of the evolutionary engines that are overcome by the approach presented in this paper. Figure 3 represents a grammar capable of representing multi-layered ANNs, with a variable number of neurons in each layer.

```
<sigexpr> ::= <node>
          |   <node> + <sigexpr>
<node> ::= <weight> * sig(<sum> + <bias>)
<sum> ::= <weight> * <features>
       |   <sum> + <sum>
<features> ::= x₁
            |   . . .
            |   xₙ
<weight> ::= <number>
<bias> ::= <number>
<number> ::= <digit>.<digit><digit>
          |   –<digit>.<digit><digit>
<digit> ::= 0 | 1 | 2 | 3 | 4
         |   5 | 6 | 7 | 8 | 9
```

Figure 2: Grammar used in [2]. n represents the number of features of the problem.

However, there is no way of knowing how many neurons are in the previous layer, so that the established connections are valid. In the next sections we detail the adaptions we introduce to allow the evolution of multi-layered networks.

4.1 Dynamic Production Rules

To know the neurons available in each hidden-layer we create new production rules, in run-time. More precisely, considering the grammar of Figure 3, for each i-th <layer> non-terminal symbol we create a <features-i> production rule, encoding the features that layer can use as input.

For the first hidden-layer, <features-1> has the available features as expansion possibilities, i.e., the ones that are initially defined in the grammar ($x_1, \ldots, x_n$, where n represents the number of features). Then, for the next hidden-layers there are two possibilities: (i) let the connections be established to all the neurons in previous layers (including input neurons); or (ii) limit the connections to the neurons in the previous layer.

In the current work we have decided for the first option, with the restriction that the neurons in the output layer can only be connected to hidden-nodes. When establishing the connections between neurons, the probability of choosing a neuron in the previous layer is proportional to the number of previous hidden-layers. More specifically:

$$P(neuron_{i-1}) = \sum_{j=1}^{j=i-2} P(neuron_j),$$

i.e., when establishing the connections of the i-th layer, the probability of linking to a neuron in the previous layer ($i - 1$) is equal to the probability of linking to a neuron in the remaining layers ($1, \ldots, i - 2$). The rationale is to minimise the emergence of deep networks with useless neurons, in the sense that they are not connected (directly or indirectly) to output nodes.

For example, using the grammar of Figure 3, when generating an ANN with two hidden-layers, with 4 and 3 nodes, respectively, the following production rules are created and added to the grammar:

        <features-1> ::= x_1 | . . . | x_n
        <features-2> ::= $h_{1,1}$ | $h_{1,2}$ | $h_{1,3}$ | $h_{1,4}$
        <features-3> ::= $h_{2,1}$ | $h_{2,2}$ | $h_{2,3}$,

```
<start> ::= <hidden-layers> -- <output-layer>

<hidden-layers> ::= <hidden-layers> -- <hidden-layers>
              |    <layer> -- <hidden-layers>
              |    <layer>

<output-layer> ::= sig( <sum> + <float>)

<layer> ::= <nodes>

<nodes> ::= <nodes> - <nodes>
        |   <node> - <nodes>
        |   <node> - <node>

<node> ::= sig(<sum> + <float>)

<sum> ::= <float> * <features>
      |   <sum> + <sum>
      |   <sum> + <sum>

<float> ::= <digit>.<digit><digit>
        |  -<digit>.<digit><digit>

<digit> ::= 0 | 1 | 2 | 3 | 4
        |   5 | 6 | 7 | 8 | 9

<features> ::= x₁
          |    . . .
          |    xₙ
```

Figure 3: Grammar used for evolving multi-layered ANNs. n represents the number of features of the problem; - and -- are used as neuron and layer separators, respectively.

where x represents input features, n the total number of input features, and $h_{i,j}$ the output of the j-th neuron of the i-th layer.

4.2 Evolutionary Engine Components

To cope with dynamic derivation rules some of the evolutionary components detailed in Section 3 are adapted. The representation of the candidate solutions is kept the same, with the additional genes for each of the dynamic rules (e.g., <features-i>) that are created. The number of dynamic rules can be different from individual to individual, making the grammar a property of the individual.

When mapping the genotype to the phenotype, if during the expansion of a non-terminal symbol a dynamic rule is called, it is necessary to know which one of the possibilities should be used. For example, in the example provided in the end of Section 4.1, when expanding the <sum> non-terminal symbol we need to know to which dynamic rule <features> corresponds to: <features-1>, <features-2> or <features-3>. Thus, we add a *num_layer* parameter to Algorithm 2, which is initialised at zero and incremented by one each time a new <layer> non-terminal symbol is expanded. Then, when the <features> non-terminal symbol is read, we expand it based on the derivation rule <features-num_layer>. However, before applying the expansion possibility it is necessary to check if it is valid. The genetic operators may have changed the number of neurons in one of the layers, or even the number of existing layers. Consequently, when the mapping procedure is applied, the genotype may be encoding a connection to a neuron in a previous layer that has been erased. In such a scenario, the genotype is fixed by replacing the faulty connection with a valid one. The dynamic rule is also corrected.

Finally, it is important to mention that now the number of genes can vary from individual to individual, which is generated by a potentially different number of hidden-layers. Therefore, in the application of the crossover operator, we only allow the exchange of the genes that are common to both parents.

Table 2: Properties of the used datasets.

	Flame	WDBC	Ionosphere	Sonar
Num. Features	2	30	34	60
Num. Instances	240	569	351	208
Class 0	36.25%	62.74%	35.90%	53.37%
Class 1	63.75%	37.26%	64.10%	46.63%

5 EXPERIMENTAL RESULTS

The conducted experiments are divided into two steps. First, we use the same grammar as in [2] to check whether or not DSGE is capable of generating one-hidden-layered ANNs that perform better than those found using GE and SGE. Then, using a new grammatical formulation, we check whether or not DSGE is suitable for the evolution of multi-layered ANNs.

5.1 Datasets

We selected four binary classification problems. The problems have an increasing complexity in terms of the available number of features of the classification task that is to be performed (see Table 2). In the next paragraphs we present a brief description of the datasets. We focus on binary problems to enable comparison with other grammar-based approaches.

Flame [5] – This dataset contains artificially generated data for clustering purposes.

Wisconsin Breast Cancer Detection (WDBC) [10, 18] – The WDBC dataset is comprised of features extracted from digitalised images of breast masses that are to be classified into malign and benign.

Ionosphere [10, 15] – This benchmark is used for the classification of ionosphere radar returns, where the returns are classified into two different classes: good if it returns evidences of structure; and bad otherwise.

Sonar [7, 10] – The sonar dataset contains features extracted from sonar signals that allow a classification model to separate between signals that are bounced off a metal cylinder or a rock cylinder.

5.2 Grammar

Two grammars are used in the conducted experiments. For the first set of experiments, targeting the evolution of one-hidden-layered ANNs the same grammar of [2] is used (see Figure 2). This grammar allows the evolution of the topology and parameters of ANNs, where all the hidden-nodes are connected to the output-node; the evolved topologies cannot have more than one hidden-layer and are restricted to just one output neuron.

Then we introduce the grammar detailed in Figure 3, which enables the generation of non fully-connected layers with more than one hidden-layer and a varying number of neurons in each layer. Note that it is not required that all neurons are directly or indirectly connected to the output nodes. Also, recall from Section 4.1 that the connections can be established to any input or hidden-node in previous layers, except for the connections of the output nodes that can only be established to hidden-nodes. In the production rules <hidden-layers>, <nodes> and <sum> a higher probability is given to the recursive expansion of the non-terminal

Table 3: Experimental parameters.

Parameter	Value
Num. runs	30
Population size	100
Num. generations (1 hidden-layer)	500
Num. generations ($\geq$ 1 hidden layers)	1500 / 3500
Crossover rate	95%
Mutation rate	1 mutation in 95% of the individuals
Tournament size	3
Elite size	1%
GE Parameter	**Value**
Individual size	200
Wrapping	0
SGE Parameter	**Value**
Recursion level	6
DSGE Parameter	**Value**
Max. depth (1 hidden-layer)	{sigexpr: 6, sum: 3}
Max. depth ($\geq$ 1 hidden-layers)	{hidden-layers: 3, nodes: 5, sum: 4}
Dataset Parameter	**Value**
Training percentage	70%
Testing percentage	30%

Figure 4: Fitness evolution of the best individuals across generations for the sonar dataset.

symbol, due to the higher difficulty of tuning parameters in deeper and more complex topologies.

5.3 Experimental Setup

Table 3 details the experimental parameters used for the tests conducted with GE, SGE and DSGE. The parameters where 1 hidden-layer and $\geq$ 1 hidden-layers are mentioned refer to the experiments conducted with the grammars of Figures 2 and 3, respectively.

To make the exploration of the search space similar we only allow one mutation in 95% of the population individuals. In the search for multi-layered networks the domain is bigger and, consequently we perform longer runs. For the experiments focusing on the generation of one-hidden-layered networks we restrict the domains to allowing similar networks in terms of the neurons and connections they can have: GE, SGE and one-hidden-layered DSGE are constrained to generated networks with up to 7 neurons and 8 connections in each neuron. In the experiments targeting networks that can have more than one hidden-layer these limits are increased: the maximum number of neurons in each layer is 32, the maximum number of connections of each neuron is 16; it is possible to generate networks with up to 8 hidden-layers.

The used datasets are all randomly partitioned in the same way: 70% of each class instances are used for training and the remaining 30% for testing. We only measure the individuals performance using the train data, and thus the test data is kept aside the evolutionary process and exclusively used for validation purposes, i.e., to evaluate the behaviour of the networks in the classification of instances that have not been seen during the creation of the ANN. The datasets are used as they are obtained, i.e., no pre-processing or data augmentation methodologies are applied.

5.4 Evolution of One-Hidden-Layered ANNs

To investigate whether or not DSGE performs better than previous grammar-based approaches we focus on experiments for the evolution of one-hidden-layered ANNs. For that, we use the grammar of Figure 2.

Figure 4 shows the evolution of the fitness across generations for GE, SGE and DSGE on the sonar dataset. Because of space constraints we only p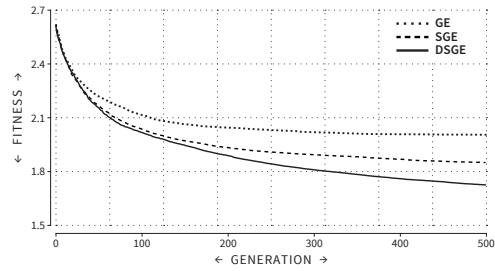resent the fitness evolution results on the most challenging dataset. Results are averages of 30 independent runs. The figure clearly shows DSGE outperforms the other methods, indicating that the new representation, where no mutation is silent promotes locality and the efficient exploration of the search space.

Table 4 reports the results obtained with DSGE, which are compared with other grammar-based approaches: GE and SGE. Results are averages of the best network (in terms of fitness) of each of the evolutionary runs, and are formatted as follows: mean $\pm$ standard deviation. For all experiments we record fitness, RMSE, Area Under the ROC Curve (AUROC) and f-measure. Except for the fitness metric, which is only computed over the train set, all metrics are calculated for the train and test sets. In addition, we also report the average number of neurons and used features.

An analysis of the results shows that DSGE performs better than the other approaches for all the datasets, and for all the metrics considered. Regarding the structure of the generated networks, DSGE is capable of finding solutions which are more complex in terms of the number of used neurons and input features. As the complexity of the networks grows more real values have to be tuned; nonetheless, DSGE is capable of performing such tuning, reaching solutions that, despite being more complex, perform better in the tested classification problems.

To verify if the differences between the tested approaches are significant we perform a statistical analysis. In [2] we have already demonstrated that SGE is consistently statistically superior to GE. As such, we will now focus in analysing if DSGE is statistically superior to SGE. To check if the samples follow a Normal Distribution we use the Kolmogorov-Smirnov and Shapiro-Wilk tests, with a significance level of $\alpha = 0.05$. The tests revealed that data does not follow any distribution and, as such, a non-parametric test (Mann-Whitney U, $\alpha = 0.05$) will be used to perform the pairwise comparison for each recorded metric. Table 4 uses a graphical overview to present the results of the statistical analysis: $\sim$ indicates no statistical difference between DSGE and SGE and + signals that DSGE is statistically superior to SGE. The effect size is denoted by the number of + signals, where +, ++ and + + + correspond respectively to low ($0.1 \leq r < 0.3$), medium ($0.3 \leq r < 0.5$) and large ($r \geq 0.5$) effect sizes. The statistical analysis reveals that DSGE is consistently statistically superior to SGE: DSGE is never worse than SGE and is only equivalent in 5 situations. In all the remaining 31 comparisons, DSGE is statistically superior, with a medium effect size in 11 occasions and a large effect size in 20 occasions.

Focusing on the comparison between train and test set results in DSGE the differences between train and test performance are,

Table 4: Evolution of one-hidden-layered ANNs. Comparison between DSGE and other grammar-based approaches. Results are averages of 30 independent runs. + and ~ represent the result of statistical tests (see text).

			Flame	WDBC	Ionosphere	Sonar
	Fitness	GE	1.58 ± 0.36	1.55 ± 0.18	1.82 ± 0.28	2.01 ± 0.23
		SGE	1.32 ± 0.25	1.46 ± 0.08	1.48 ± 0.18	1.85 ± 0.18
		DSGE	1.16 ± 0.13+++	1.36 ± 0.06+++	1.38 ± 0.13+++	1.73 ± 0.12+++
Train	RMSE	GE	0.28 ± 0.15	0.24 ± 0.09	0.33 ± 0.10	0.38 ± 0.08
		SGE	0.16 ± 0.13	0.19 ± 0.03	0.21 ± 0.07	0.34 ± 0.06
		DSGE	0.08 ± 0.06+++	0.15 ± 0.03+++	0.17 ± 0.05++	0.30 ± 0.05++
	Accuracy	GE	0.90 ± 0.10	0.92 ± 0.12	0.79 ± 0.20	0.76 ± 0.16
		SGE	0.96 ± 0.08	0.95 ± 0.02	0.93 ± 0.11	0.84 ± 0.12
		DSGE	0.99 ± 0.03+++	0.97 ± 0.01+++	0.97 ± 0.03+++	0.90 ± 0.04++
	AUROC	GE	0.96 ± 0.05	0.98 ± 0.02	0.86 ± 0.18	0.88 ± 0.06
		SGE	0.98 ± 0.04	0.99 ± 0.01	0.94 ± 0.04	0.91 ± 0.04
		DSGE	0.99 ± 0.01+++	0.99 ± 0.00+++	0.96 ± 0.03+++	0.93 ± 0.04~
	F-measure	GE	0.91 ± 0.09	0.88 ± 0.18	0.78 ± 0.32	0.70 ± 0.29
		SGE	0.96 ± 0.08	0.93 ± 0.03	0.93 ± 0.18	0.78 ± 0.23
		DSGE	0.99 ± 0.02+++	0.96 ± 0.02+++	0.98 ± 0.02+++	0.88 ± 0.05++
Test	RMSE	GE	0.31 ± 0.15	0.27 ± 0.09	0.38 ± 0.07	0.45 ± 0.06
		SGE	0.22 ± 0.13	0.23 ± 0.04	0.32 ± 0.05	0.44 ± 0.04
		DSGE	0.14 ± 0.08++	0.20 ± 0.03++	0.28 ± 0.04+++	0.43 ± 0.04~
	Accuracy	GE	0.88 ± 0.11	0.90 ± 0.12	0.76 ± 0.18	0.68 ± 0.12
		SGE	0.93 ± 0.09	0.93 ± 0.02	0.87 ± 0.10	0.73 ± 0.09
		DSGE	0.97 ± 0.05+++	0.95 ± 0.02+++	0.90 ± 0.03++	0.76 ± 0.05~
	AUROC	GE	0.93 ± 0.08	0.97 ± 0.03	0.83 ± 0.09	0.81 ± 0.05
		SGE	0.96 ± 0.08	0.98 ± 0.02	0.90 ± 0.05	0.82 ± 0.05
		DSGE	0.99 ± 0.03++	0.98 ± 0.01++	0.93 ± 0.04++	0.83 ± 0.04~
	F-measure	GE	0.89 ± 0.09	0.86 ± 0.18	0.76 ± 0.31	0.61 ± 0.25
		SGE	0.94 ± 0.09	0.91 ± 0.03	0.89 ± 0.17	0.64 ± 0.20
		DSGE	0.98 ± 0.04+++	0.93 ± 0.03+++	0.93 ± 0.02++	0.72 ± 0.06~
	Num. Neurons	GE	3.33 ± 1.40	3.13 ± 1.53	2.50 ± 1.41	2.53 ± 1.20
		SGE	4.87 ± 1.83	3.73 ± 1.53	3.53 ± 1.36	3.07 ± 1.39
		DSGE	6.47 ± 1.20	6.23 ± 1.58	5.97 ± 1.78	6.13 ± 1.69
	Num. Features	GE	1.97 ± 0.18	8.40 ± 3.81	7.33 ± 5.33	9.40 ± 5.73
		SGE	2.00 ± 0.00	12.0 ± 6.51	12.1 ± 5.79	13.3 ± 6.42
		DSGE	2.00 ± 0.00	14.5 ± 3.52	13.3 ± 4.74	17.6 ± 4.66

Table 5: DSGE evolution of multi-layered ANNs. + and ~ symbols represent the result of statsitical tests, and have the same meaning as in Table 4.

		Flame	WDBC	Ionosphere	Sonar
Fitness	1 H-L	1.16 ± 0.13	1.36 ± 0.06	1.38 ± 0.13	1.73 ± 0.12
	≥ 1 H-Ls	1.15 ± 0.18~	1.35 ± 0.13~	1.36 ± 0.19~	1.57 ± 0.18~
	> 1 H-Ls	1.08 ± 0.13+++	1.32 ± 0.11~	1.30 ± 0.11+++	1.53 ± 0.15+++
Train — RMSE	1 H-L	0.08 ± 0.06	0.15 ± 0.03	0.17 ± 0.05	0.30 ± 0.05
	≥ 1 H-Ls	0.07 ± 0.08~	0.15 ± 0.05~	0.17 ± 0.08~	0.26 ± 0.07~
	> 1 H-Ls	0.07 ± 0.05+++	0.07 ± 0.04~	0.14 ± 0.05+++	0.25 ± 0.06+++
Accuracy	1 H-L	0.99 ± 0.03	0.97 ± 0.01	0.97 ± 0.03	0.90 ± 0.04
	≥ 1 H-Ls	0.99 ± 0.02~	0.97 ± 0.02~	0.97 ± 0.03~	0.92 ± 0.05~
	> 1 H-Ls	0.99 ± 0.02~	0.97 ± 0.02~	0.98 ± 0.02~	0.93 ± 0.04+++
AUROC	1 H-L	0.99 ± 0.01	0.99 ± 0.00	0.96 ± 0.03	0.93 ± 0.04
	≥ 1 H-Ls	0.99 ± 0.01~	0.99 ± 0.01~	0.96 ± 0.04~	0.93 ± 0.05~
	> 1 H-Ls	1.00 ± 0.01~	0.99 ± 0.00~	0.97 ± 0.02~	0.93 ± 0.04~
F-measure	1 H-L	0.99 ± 0.02	0.96 ± 0.02	0.98 ± 0.02	0.88 ± 0.05
	≥ 1 H-Ls	0.99 ± 0.02~	0.96 ± 0.03~	0.97 ± 0.02~	0.91 ± 0.06~
	> 1 H-Ls	0.99 ± 0.01~	0.97 ± 0.03~	0.98 ± 0.01+++	0.92 ± 0.05+++
Test — RMSE	1 H-L	0.14 ± 0.08	0.20 ± 0.03	0.28 ± 0.04	0.43 ± 0.04
	≥ 1 H-Ls	0.17 ± 0.08~	0.20 ± 0.04~	0.31 ± 0.05~	0.44 ± 0.05~
	> 1 H-Ls	0.16 ± 0.08~	0.20 ± 0.04~	0.29 ± 0.05~	0.43 ± 0.06~
Accuracy	1 H-L	0.97 ± 0.05	0.95 ± 0.02	0.90 ± 0.03	0.76 ± 0.05
	≥ 1 H-Ls	0.96 ± 0.04~	0.95 ± 0.02~	0.89 ± 0.03~	0.77 ± 0.05~
	> 1 H-Ls	0.97 ± 0.04~	0.95 ± 0.02~	0.90 ± 0.03~	0.78 ± 0.06~
AUROC	1 H-L	0.99 ± 0.03	0.98 ± 0.01	0.93 ± 0.04	0.83 ± 0.04
	≥ 1 H-Ls	0.98 ± 0.03~	0.99 ± 0.01~	0.91 ± 0.06~	0.82 ± 0.07~
	> 1 H-Ls	0.99 ± 0.02~	0.99 ± 0.01~	0.93 ± 0.03~	0.82 ± 0.07~
F-measure	1 H-L	0.98 ± 0.04	0.93 ± 0.03	0.93 ± 0.02	0.72 ± 0.06
	≥ 1 H-Ls	0.97 ± 0.03~	0.93 ± 0.03~	0.92 ± 0.02~	0.73 ± 0.07~
	> 1 H-Ls	0.97 ± 0.03~	0.93 ± 0.03~	0.92 ± 0.02~	0.74 ± 0.08~
Num. H-Ls	1 H-L	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	≥ 1 H-Ls	2.27 ± 0.98	2.23 ± 1.04	1.90 ± 0.84	2.37 ± 0.96
	> 1 H-Ls	2.81 ± 0.60	2.95 ± 0.52	2.50 ± 0.51	2.95 ± 1.12
Num. Neurons	1 H-L	3.33 ± 1.40	3.13 ± 1.53	2.50 ± 1.41	2.53 ± 0.38
	≥ 1 H-Ls	10.6 ± 6.49	9.73 ± 7.03	7.17 ± 4.98	9.27 ± 4.55
	> 1 H-Ls	12.5 ± 6.64	12.8 ± 6.70	9.33 ± 5.24	10.9 ± 4.12
Num. Features	1 H-L	2.00 ± 0.00	14.5 ± 3.52	13.1 ± 6.87	19.8 ± 8.39
	≥ 1 H-Ls	2.00 ± 0.00	16.3 ± 7.03	13.8 ± 6.66	21.8 ± 8.48
	> 1 H-Ls	2.00 ± 0.00	18.2 ± 6.91	15.8 ± 5.48	24.0 ± 7.54

on average, 0.09, 0.06, 0.04 and 0.06, for RMSE, accuracy, AUROC and f-measure, respectively. For SGE the differences are 0.08, 0.06, 0.04 and 0.06 and for GE 0.05, 0.04, 0.04 and 0.04, for the RMSE, accuracy, AUROC and f-measure, respectively. Despite the fact that the differences in DSGE are superior to the ones in SGE and GE, it is our perception that this is a result of the better DSGE results and not an indicator of overfitting.

In [2] we showed that the results obtained with SGE are superior to those of other grammar-based approaches (namely, the ones described in [1, 16, 20]). Additionally, we showed that it is beneficial to evolve both the topology and weights of a network, since it has better results than ANNs obtained by hand-crafting the topology of the networks and training them using backpropagation. As DSGE is statistically superior to SGE it is then clear that it is also better than previous methods.

5.5 Evolution of Multi-Layered ANNs

To search for ANNs that may have more than one hidden-layer we use the grammar of Figure 3.

The experimental results are presented in Table 5. For each metric, the first two rows present the averages of the tests conducted with the grammars of Figures 2 and 3, respectively, i.e., the grammars for encoding networks with just one hidden-layer (1 H-L, from Table 4) or that allow the generation of networks with multiple hidden-layers (≥ 1 H-Ls). Additionally, we analyse the ANNs that have more than one hidden-layer (last row, > 1 H-Ls), discarding the evolutionary runs that resulted in networks with just one hidden-layer.

Results of 1 H-L and ≥ 1 H-Ls are averages of 30 independent runs; in the > 1 H-Ls row results are averages of 21, 19, 18 and 21 independent runs for the flame, WDBC, ionosphere and sonar datasets, respectively. That is, for the flame, WDBC, ionosphere and sonar datasets in 9, 11, 12 and 9 runs the best network is composed by only one hidden-layer. For the flame and WDBC we perform runs with 1500 generations, and for the ionosphere and sonar 3500 generations are used. The rationale behind the different number of generations is related to the number of features of each problem and the possible progression margin: in flame and WDBC the metrics in train and test already have values close to optimum, thus suggesting that one hidden-layer is likely enough to solve the classification task.

The experimental results show that by using DSGE it is possible to evolve effective multi-layered ANNs. By comparing the first two rows (1 H-L and ≥ 1 H-Ls) it is also clear that the results are equivalent, which is confirmed by a statistical analysis (Mann-Whitney, $\alpha = 0.05$) that shows no statistical differences. Focusing on the comparison between one-hidden-layered ANNs and those that have more than one hidden-layer (> 1 H-Ls) a small, but perceptive difference exists. That difference is statistically significant in some of the training metrics, and in none of the testing metrics. When there is a statistical difference the effect size is always large. Moreover,

the difference is larger in the datasets that have more input features and a greater margin for improvement. On the contrary, almost no improvement is observable in the flame dataset, which is expected as the problem only has two features, and thus an ANN with one hidden-layer already performs close to optimal. In a nutshell, although there are no statistical differences between the performance of 1 H-L and ≥ 1 H-Ls runs, when the evolutionary process results in multi-layered ANNs the differences begin to emerge. This result shows that DSGE is able to cope with the higher dimensionality of the search space associated with the evolution of multi-layered topologies and indicates that in complex datasets, where there is a clear advantage in using deeper topologies, DSGE is likely to outperform other grammar-based approaches in train and test.

The complexity of the generated ANNs (in terms of the number of used neurons and hidden-layers) is greater when allowing the generation of multi-layered ANNs. The difference is even larger if we consider only those networks that have more than one hidden-layer. More neurons means more weights and bias values that have to be evolved, making the evolutionary task more difficult. ANNs with fewer layers tend to have less neurons, and consequently are benefited from an evolutionary point of view, as less real values need to be tuned. Thus, in future experiments it is our intention to use the evolutionary approach to evolve the initial set of weights and then apply a fine tuning stage (e.g, backpropagation or resilient backpropagation) during a maximum number of epochs proportional to the number of neurons and/or hidden-layers. As we allow hidden-nodes to connect to nodes in previous layers (including input features) the number of used features is also higher.

Preliminary experiments concerning the evolution of ANNs with more than one output neuron have also been conducted, i.e., instead of using just one output neuron and the sigmoid function as activation function we used two output nodes (one per class) with the softmax activation function. Obtained performances are similar. However, results are not presented due to the lack of space.

6 CONCLUSION AND FUTURE WORK

In this paper we propose DSGE: a new genotypic representation for SGE. The gain is two fold: (i) while in previous grammmar-based representations, the genotype encodes the largest allowed sequence, in DSGE the genotype grows as needed; and (ii) there is no need to pre-process the grammar in order to expand recursive production rules, as the maximum depth is defined for each sub-tree. Most importantly, DSGE solves a limitation of other GGP methodologies by allowing the evolution of solutions to dynamic domains, such as ANNs, where there is the need to know the number of neurons available in previous layers so that valid individuals are generated.

Results show that DSGE is able to evolve the topology and weights of one-hidden-layered ANNs that perform statistically better than those evolved using GE or SGE. Moreover, the results are also better than those provided by hand-crafted ANNs finetuned using backpropagation, and than the ones generated using other GE-based approaches [1, 16, 20]. Results concerning the evolution of multi-layered ANNs despite not statistical show that the methodology is suitable for the evolution of accurate ANNs. Experiments on more, and more difficult datasets are needed to better analyse the evolution of ANNs with more than one hidden-layer. However,

the ones reported in the current paper had to be used in order to compare DSGE with previous approaches.

Future work will focus on the performance of experiments using more complex datasets (e.g., MNIST and CIFAR) and on the generalisation of the allowed layer types, so that it is possible to evolve networks that use, for example, convolution and/or pooling layers.

ACKNOWLEDGMENTS

REFERENCES

[1] Fardin Ahmadizar, Khabat Soltanian, Fardin AkhlaghianTab, and Ioannis Tsoulos. 2015. Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm. *Engineering Applications of Artificial Intelligence* 39 (2015), 1–13.

[2] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. 2017. Automatic Generation of Neural Networks with Structured Grammatical Evolution. In *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE.

[3] Omid E David and Iddo Greental. 2014. Genetic algorithms for evolving deep neural networks. In *Proceedings of the 2014 Conf. companion on Genetic and evolutionary computation companion*. ACM, 1451–1452.

[4] Lucian-Ovidiu Fedorovici, Radu-Emil Precup, Florin Dragan, and Constantin Purcaru. 2013. Evolutionary optimization-based training of convolutional neural networks for OCR applications. In *System Theory, Control and Computing (ICSTCC), 2013 17th International Conf.* IEEE, 207–212.

[5] Limin Fu and Enzo Medico. 2007. FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data. *BMC bioinformatics* 8, 1 (2007), 1.

[6] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. 2008. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research* 9, May (2008), 937–965.

[7] R. Paul Gorman and Terrence J. Sejnowski. 1988. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural networks* 1, 1 (1988), 75–89.

[8] Frederic Gruau. 1992. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In *International Workshop on Combinations of Genetic Algorithms and Neural Networks, COGANN-92*. IEEE, 55–74.

[9] Jae-Yoon Jung and James A Reggia. 2006. Evolutionary design of neural network architectures using a descriptive encoding language. *IEEE transactions on evolutionary computation* 10, 6 (2006), 676–688.

[10] M. Lichman. 2013. UCI ML Repository. (2013). archive.ics.uci.edu/ml

[11] Nuno Lourenço, Francisco B. Pereira, and Ernesto Costa. 2016. Unveiling the properties of structured grammatical evolution. *Genetic Programming and Evolvable Machines* (2016), 1–39.

[12] Miguel Rocha, Paulo Cortez, and José Neves. 2007. Evolution of neural networks for classification and regression. *Neurocomputing* 70, 16 (2007), 2809–2816.

[13] Conor Ryan, JJ Collins, and Michael O'Neill. 1998. *Grammatical evolution: Evolving programs for an arbitrary language.* Springer Berlin Heidelberg, Berlin, Heidelberg, 83–96.

[14] Tapas Si, Arunava De, and Anup Kumar Bhattacharjee. 2014. Grammatical Swarm for Artificial Neural Network Training. In *International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. IEEE, 1657–1661.

[15] Vincent G. Sigillito, Simon P. Wing, Larrie V. Hutton, and Kile B. Baker. 1989. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest* 10, 3 (1989), 262–266.

[16] Khabat Soltanian, Fardin Akhlaghian Tab, Fardin Ahmadi Zar, and Ioannis Tsoulos. 2013. Artificial neural networks generation using grammatical evolution. In *21st Iranian Conference on Electrical Engineering (ICEE)*. IEEE, 1–5.

[17] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[18] W. Nick Street, William H. Wolberg, and Olvi L. Mangasarian. 1993. Nuclear feature extraction for breast tumor diagnosis. In *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*. International Society for Optics and Photonics, 861–870.

[19] Sreenivas Sremath Tirumala, S Ali, and C Phani Ramesh. 2016. Evolving deep neural networks: A new prospect. In *Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conf. on*. IEEE, 69–74.

[20] Ioannis Tsoulos, Dimitris Gavrilis, and Euripidis Glavas. 2008. Neural network construction and training using grammatical evolution. *Neurocomputing* 72, 1 (2008), 269–277.

[21] Darrell Whitley, Timothy Starkweather, and Christopher Bogart. 1990. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel computing* 14, 3 (1990), 347–361.