The Importance of Topology Evolution in NeuroEvolution: A Case Study Using Cartesian Genetic Programming of Artificial Neural Networks

Andrew James Turner and Julian Francis Miller

Abstract NeuroEvolution (NE) is the application of evolutionary algorithms to Artificial Neural Networks (ANN). This paper reports on an investigation into the relative importance of weight evolution and topology evolution when training ANN using NE. This investigation used the NE technique Cartesian Genetic Programming of Artificial Neural Networks (CGPANN). The results presented show that the choice of topology has a dramatic impact on the effectiveness of NE when only evolving weights; an issue not faced when manipulating both weights and topology. This paper also presents the surprising result that topology evolution alone is far more effective when training ANN than weight evolution alone. This is a significant result as many methods which train ANN manipulate only weights.

1 Introduction

NeuroEvolution (NE) is the application of evolutionary algorithms to artificial neural networks (ANNs). NE has many advantages over traditional gradient based training methods [19]. NE does not require the neuron transfer functions to be differentiable in order to find a fitness gradient; a process which can be computationally expensive. NE is resilient to becoming trapped in local optima. Gradient based methods are restricted to certain topologies and struggle to train deep ANNs [3, 6]; whereas NE does not. NE also does not require a target behaviour to be known in advance, which is required to produce the error value for gradient methods. This allows NE to be applied to open ended problems.

A. J. Turner (✉) · J. F. Miller
The University of York, York, United Kingdom
e-mail: at568@york.ac.uk

J. F. Miller
e-mail: julian.miller@york.ac.uk

M. Bramer and M. Petridis (eds.), *Research and Development in Intelligent Systems XXX*, 213
DOI: 10.1007/978-3-319-02621-3_15,

NE methods can be split into two groups [19], those which adjust only the connection weights of a user defined network topology and those which adjust both the connection weights and topology. There are many possible advantages to evolving topology as well as weights, including: access to topologies which would otherwise not be considered, searching in a topology space as well as a weight space and not requiring the user to choose a topology before it is know which topologies would be suitable.

Although it is thought that topology and weight evolution offers a significant advantage over weight evolution alone [2, 19], to the authors knowledge there are no publications which truly compare the two approaches. It is true that comparisons can be made between different NE methods which do and do not evolve topologies, but differences in implementation are likely to influence the result, weakening any comparison. It is also not known whether weight evolution is more beneficial than topology evolution, or vice versa, or if they are more effective when used together.

This paper investigates the relative importance of only connection weight evolution, only topology evolution, and both connection weight and topology evolution in NE. This is achieved using the NE method Cartesian Genetic Programming of Artificial Neural Networks (CGPANN) [5, 15] which is the application of the genetic programming technique Cartesian Genetic Programming (CGP) [8, 10] to ANNs. CGPANN is a topology and weight evolving NE method which can easily be adapted to evolve only connection weights or only topology. This makes CGPANN very suitable for investigating the relative importance of topology and weight evolution. CGPANN can also be seeded with random or user-defined topologies meaning that the influence of different topologies for weight only evolution can also be studied.

The remainder of the paper is structured as follows: Sect. 2 discusses NE in terms of weight and topology evolution, Sect. 3 describes CGP and its application to ANN, Sect. 4 describes the experiments presented in this paper with the results given in Sect. 5 and finally Sects. 6 and 7 give an evaluation of the results and closing conclusions.

2 NeuroEvolution - Weights and Topologies

In the NE literature it is thought that topologies, as well as weights, are highly important in the training of ANNs [2, 19]. Theoretically an ANN can be thought of in terms of a topology and weight search space; or as weight spaces associated with any given topology. Using only fixed topologies therefore limits the search to one subset weight space within the topology space; a possible disadvantage of methods which only manipulate weights such as traditional back propagation. Clearly, if a suitable topology is known in advance it could be used to decrease the dimensionality of the search. However, often this is not known in advance. Interestingly, the strategy of just increasing or decreasing the number of neurons during the search, in order to navigate the topology landscape, could be thought of as structural hill climbing [1].

However, this approach is likely to get trapped in topology local optima just as back propagation does with weights.

In the growing number of NE techniques found in the literature, some methods manipulate only weights [4, 11] and others manipulate both weights and topology [1, 5, 12, 13]. It is difficult however to draw empirical conclusions about the benefit of evolving topology as each of these techniques use different encodings to describe the ANN during evolution. When, for example, a weight and topology evolving NE technique outperforms a weight only technique, we may assume that the increase in performance was due to its ability to evolve topologies but it could equally be due to the differences in implementation (or both). For instance, the weight only evolving method Symbiotic Adaptive Neuro-Evolution (SANE) [11] evolves ANNs at a neuron level, with the complete networks assembled using neurons selected from the population. In contrast, the weight and topology evolving method NeuroEvolution of Augmenting Topologies (NEAT) [13] evolves ANNs at a network level and employs strategies to track when ancestral changes took place in order to more effectively make use of the crossover operator. So when it is shown that NEAT out performs SANE on a given benchmark [13] it is not clear if this is due to the ability to evolve topologies or due to other differences between the two methods, or both. Additionally, to the authors knowledge there are no NE techniques which solely rely on the evolution of topology with no alterations to the weights. For these reasons a direct comparison between the importance of weight evolution and topology evolution has not been able to be made.

3 Cartesian Genetic Programming Artificial Neural Networks

CGPANN [5] is a highly competitive [15] NE strategy which uses CGP [8, 10] to evolve both the weights and topology of ANNs. CGPANN chromosomes are usually initialised as random networks, but can be seeded with any starting topology. CGPANN can also be configured to only evolve weights, only evolve topology or evolve weights and topology. For these reasons CGPANN is a suitable tool for empirically investigating the effect topology has when evolving weights and the relative importance of evolving weight and topology for NE strategies.

3.1 CGP

CGP [8, 10] is a form of Genetic Programming which represents computational structures as directed, usually acyclic graphs indexed by their Cartesian coordinates. Each node may take its inputs from any previous node or program input. The program outputs are taken from the output of any internal node or program input. This structure leads to many of the nodes described by the CGP chromosome not contributing to the final operation of the phenotype, these inactive, or “junk”, nodes have been shown

to greatly aid the evolutionary search [7, 17, 20]. The existence of inactive genes in CGP are also useful in that they suppress a phenomenon in GP known as bloat. This is the uncontrolled growth in size of evolved computational structures over evolutionary time [9].

The nodes described by CGP chromosomes are arranged in a rectangular $r \times c$ grid of nodes, where r and c respectively denote the user-defined number of rows and columns. In CGP, nodes in the same column are not allowed to be connected together (as in multi-layer perceptrons). CGP also has a connectivity parameter l called "levels-back" which determines whether a node in a particular column can connect to a node in a previous column. For instance if $l = 1$ all nodes in a column can only connect to nodes in the previous column. Note that levels-back only restricts the connectivity of nodes; it does not restrict whether nodes can be connected to program inputs (terminals). It is important to note that any architecture (limited by the number of nodes) can be constructed by arranging the nodes in a $1 \times n$ format where the n represents the maximum number of nodes (columns) and choosing $l = n$. Using this representation the user does not need to specify the topology, which is then automatically evolved along with the program.

Figure 1 gives the general form of a CGP showing that a CGP chromosome can describe multiple input multiple output (MIMO) programs with a range of node transfer functions and arities. In the chromosome string, also given in Fig. 1, F_i denote the function operation at each node, C_i index where the node gathers its inputs and each O_i denote which nodes provide the outputs of the program. It should be noted that CGP is not limited to only one data type, it may be used for Boolean values, floats, images, audio files, videos etc. CGP generally uses the evolutionary strategy (ES) algorithm $(1 + \lambda) - ES$. In this algorithm in each generation there are $1 + \lambda$ candidates and the fittest is chosen as the parent. The next generation is formed by this parent and λ offspring obtained through mutation of the parent. It is important to note that if no offspring are fitter than the parent, but at least one has the same fitness as the parent, then the offspring is chosen as the new parent. In

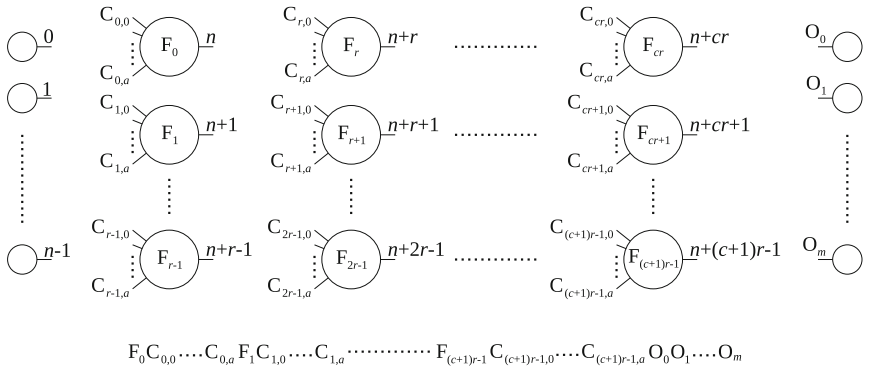

Fig. 1 Depiction of a cartesian genetic programs structure with chromosome encoding below, taken from [10]

CGP, the λ value is commonly set as four. The connection genes in the chromosomes are initialised with random values that obey the constraints imposed by the three CGP structural parameters, r, c, l. The function genes are randomly chosen from the allowed values in the function lookup table. The output genes O_i are randomly initialised to refer to any node or input in the graph. The standard mutation operator used in CGP works by randomly choosing a valid allele at a randomly chosen gene location. The reason why both a simple operator and a simple evolutionary algorithm are so effective is related to the presence of non-coding genes. Simple mutations can connect or disconnect whole sub-programs. For a more detailed description of CGP see [10].

3.2 CGP Encoded Neural Networks

ANNs are a natural application for CGP as both ANNs and CGPs are structured as directed acyclic[1] graphs. CGP is directly compatible with ANNs by using node transfer functions suited to ANNs (radial bias, sigmoidal etc) and encoding extra chromosome values for the weights assigned to each nodes/neurons input. These extra weight values are evolved with the rest of the CGP chromosome and are subject to mutation operations like other genes. The range of values that these weight genes can take is specified by the user; for example $[-10, 10]$.

In order to independently study the effect of weight evolution and topology evolution, the mutation percentage used by the CGPANN was split into two values; weight mutation percentage and topology mutation percentage. The weight mutation percentage is the percentage of weight genes which are changed in the chromosome and the topology mutation percentage is the percentage of connection genes and output genes which are changed. Weight only evolution can then be studied by setting the topology mutation percentage to zero and vice versa. If CGPANN is used with the topology mutation percentage set to zero, the CGPANN method becomes equivalent to a simple genetic algorithm manipulating the weights of a fixed topology. The reason for using CGPANN in this way is so that the same code could be used for all experiments; it also an example of how flexible the CGPANN approach is to different requirements.

In CGPANNs it is possible for there to be multiple connections between the same two nodes [15]. This is because when a connection gene is mutated, no effort is made to ensure that multiple connections between nodes do not occur; although this could easily be implemented if desired. In contrast, the approach taken in this paper is to allow only the first of any connections betweens two nodes into the phenotype. This is done for two reasons. Firstly, it gives CGPANN the ability to indirectly evolve each nodes arity. For instance, if a node had arity six but three inputs were from one node and three from another, then the node effectively has arity two as the majority of the connections would not be decoded into the phenotype. If a mutation operation then

[1] Both CGP and ANNs can also be structured in a recurrent form.

changed one of these inputs to a new different node, then the effective arity would increase to three; it changes during evolution. Secondly, if multiple connections between the same two nodes were encoded into the phenotype, then the maximum weight range set by the user could be exceeded. This results from two connections between the same two nodes being equivalent to one connection with the sum of their weights. It is unknown if this offers an advantage during evolution or not. However as this paper is concerned with a comparison with weight only evolving methods which cannot exceed the user defined maximum weight range, it is required that this ability is removed for a fair comparison.

Since this paper investigates the effect of evolving topology compared to weight only evolution, the implementation of CGPANN used here is restricted so that it only evolves acyclic networks with the node transfer functions fixed. In its unrestricted form CGPANN can evolve weights, topology, node arity and node transfer functions for both recurrent and non-recurrent ANNs. CGPANN also has all of the advantages of regular CGP, including: natural resilience to bloat [9] and redundancy in the chromosomes aiding evolution [7].

4 Experiments

Two experiments are presented in this paper. The first investigates the effect of topology when only evolving connection weights. The second investigates the relative importance of connection weight evolution and topology evolution. Both of these experiments are carried out using CGPANN. The evolutionary algorithm is $(1 + 4) - \text{ES}$. Probabilistic mutation was used where each gene is altered with a given probability. The weights between nodes are limited to $[-10, 10]$. When evolving topologies the CGP levels back parameter is set so that each node can connect to any previous node or the inputs, the outputs can also be taken from any node or input. All of the experiments were repeated fifty times in order to produce averages which are presented in this paper.

4.1 Effect of Topology on Weight Evolution

The first experiment investigates the effect of topology when using weight evolving NE. This is achieved by comparing the results from a range of topologies when only evolving connection weights. This experiment was designed to highlight the impact of the chosen topology on the effectiveness of the search. The results of this experiment are compared with an experiment in which both topology and connection weights evolve (i.e. using CGPANN in its regular form).

When using fixed topologies the initial chromosomes are seeded with an ANN that is fully connected[2]; each with a given number of hidden layers and nodes per layer.

When evolving both topology and weights, the number of hidden layers and nodes per layer are also varied, but here they represent the upper limits of the network topology; the maximum number of rows and columns in the CGPANN chromosome, see Sect. 3. In this case, the CGPANN chromosomes are not seeded with any predefined topology but initialised with randomly created networks. When using fixed topologies the arity of each node is determined by the number of nodes in the previous layer. However, when evolving the topology, the user must specify a node arity. For this experiment the node arity used was ten. Each node can however effectively lower its arity by connecting to the same previous node multiple times; as the implementation of CGPANN used here only decodes the first connection between the same two nodes into the phenotype. All cases were run for five thousand generations (20001 evaluations) and the final fitnesses reported. The connection weight and topology mutation rates were five and zero percent respectively for the fixed topology case and both five percent when evolving weights and topology.

4.2 Weight Evolution Versus Topology Evolution

The second experiment investigates the relative importance of weight evolution and topology evolution when using NE. This is investigated by comparing NE performance in three situations: (a) only evolving weights, (b) only evolving topology and (c) evolving both weights and topology. In all experiments initial populations were seeded with random weights and topologies. We achieved these comparisons by comparing the results obtained using the following three methods:

1. Fixing the topology mutation rate as zero and varying the weight mutation rate.
2. Fixing the weight mutation rate as zero and varying the topology mutation rate.
3. Varying both the weight and topology mutation rate (both with the same value).

For the first method the randomly generated topologies remained fixed and only the connection weights mutated with a number of mutation rates. For the second method the randomly generated weights remained fixed and only the topology mutated with a number of mutation rates. For the third method both the randomly generated weights and topologies were mutated with a number of mutation rates. All of these methods were run for one thousand generations (4001 evaluations). The maximum number of layers/rows was set as thirty with one node-per-layer/column. These CGPANN row and column limits were used as they allow the highest possible number of topologies for the amount of nodes available; for instance all topologies possible when rows $= 15$ and columns $= 2$ are also possible when rows $= 30$ and

[2] Fully connected between layers i.e. a node in hidden layer two has an input from every node in hidden layer one.

columns = 1. This structure can not encode all topologies however, due to the arity limit set for each node.[3] Here the arity of each node was set as ten. Again only the first connection between the same two nodes was decoded into the phenotype.

4.3 Test Problems

In order to investigate the described experiments, benchmark problems are required. Such test problems should be typical of the types of problems ANN are often applied to and should be challenging enough to draw out any differences between weight and topology evolution. The following two test problems were used for the experiments presented in this paper; the first is a control problem and the second is a classification problem (two common applications for ANN).

4.3.1 Double Pole

The Double Pole benchmark [18] is a classic control problem in the artificial intelligence literature and is often used for comparing NE methods [15]. The task is to balance two poles hinged, at their base, on a controllable cart. All movements are limited to one dimension. The cart is placed on a 4.8 m track, with the cart always starting in the centre. The longer of the two poles starts at 1 deg from vertical and the shorter starts at vertical. The cart is controlled by applying a force in the range [−10, 10] N thus moving the cart from side to side. See [18] for diagrams and further details.

The equations which govern the dynamics of the poles and cart are given in Eqs. 1 and 2 with the parameter definitions and limits given in Table 1. The simulations

Table 1 Parameters used for the Double Pole balancing benchmark

Symbol	Description	Values
x	Cart position	$[-2.4, 2.4]$ m
θ_i	ith pole angle	$[-36, 36]$ deg
F	Force applied to cart	$[-10, 10]$ N
$\tilde{F}_i$	Force on cart due to ith pole	-
l_i	Half length of ith pole	$l_1, l_2 = 0.5, 0.05$ m
M	Cart mass	1.0 kg
m_i	Mass of ith pole	$m_1, m_2 = 0.1, 0.01$ kg
μ_c	Friction coefficient between cart and track	0.0005
μ_{pi}	Friction coefficient between ith pole and cart	0.000002

[3] If the arity is set high enough however all topologies are possible as each node can lower its own arity by only utilizing the first of multiple connections between two nodes.

are run using Euler integration with a time step of 0.01 s. The outputs of the ANN are updated every 0.02 s. The simulations are then run for 100,000 time steps using the ANN being tested as the controller. The assigned fitness is the number of time steps (out of a maximum 100,000) over which the ANN keeps both poles within $[-36, 36]$ deg from vertical whilst keeping the cart within the track range.

The inputs to the ANN are cart position, cart velocity, both poles' angles from vertical and both poles' angular velocity. Each of these inputs are scaled to a $[-1, 1]$ range by assuming the following ranges: cart position $[-2.4, 2.4]$ m, cart velocity $[-1.5, 1.5]$ m/s, pole positions $[-36, 36]$ deg and pole velocities $[-115, 115]$ deg/s.

The output from the ANN, also in the range $[-1, 1]$, is also scaled to a force in the range $[-10, 10]$ N. As an extra restraint the magnitude of the applied force is constrained to be always greater than $\frac{1}{256} \times 10$ N. This slightly increases the difficulty of the task. All the node transfer functions use the bipolar sigmoid which produces an output in the range $[-1, 1]$; mapping easily to $[-10, 10]$.

$$\ddot{x} = \frac{F - \mu_c sgn(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i}, \qquad \ddot{\theta}_i = -\frac{3}{4l_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right) \tag{1}$$

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\sin\theta_i\right), \qquad \tilde{m}_i = m_i\left(1 - \frac{3}{4}\cos^2\theta_i\right) \tag{2}$$

4.3.2 Monks Problems

The Monks Problems [14] are a set of classification benchmarks intended for comparing learning algorithms. The classification tasks are based on the appearance of robots which are described by six attributes each with a range of values: head_shape (round, square, octagon), body_shape (round, square, octagon), is_smiling (yes, no), holding (sword, balloon, flag), jacket_color (red, yellow green, blue), has_tie (yes, no). There are three classification tasks described in [14] but only the first is used here. Where the robot belongs to the class if (head_shape = body_shape) or (jacket_color = red) else it does not.

The original document used a randomly selected 124 (of the possible 432) combinations to be used for the training set. As this paper is not concerned with generalisation the ANNs were both trained and tested on the complete training set. All results quoted in this paper are the ANNs final classification error on the training set. The fitness is the classification error (which was to be minimised).

The implementation used here assigns each possible attribute value its own input to the ANN; totalling seventeen inputs. Each of these inputs is set as one if the particular attributes value is present and as zero otherwise. The ANN classifies each sample as belonging to the class if the single ANN output is greater or equal to 0.5. The transfer function used by all the nodes was the unipolar sigmoid.

5 Results

5.1 Effect of Topology on Weight Evolution

The results of the first experiment, investigating the effect of topology when evolving connection weights, are given in Figs. 2 and 3 for the Double Pole and Monks Problem test problems respectively. The left images show the average fitness for a range of fixed topologies when only evolving the weights. The right images, for comparison, show the average fitnesses when evolving both weights and topology for a range of CGPANN topology limits.

Figures 2 and 3 show that when weights are evolved with a fixed topology, the search effectiveness is highly dependent on the topology used. This result has been

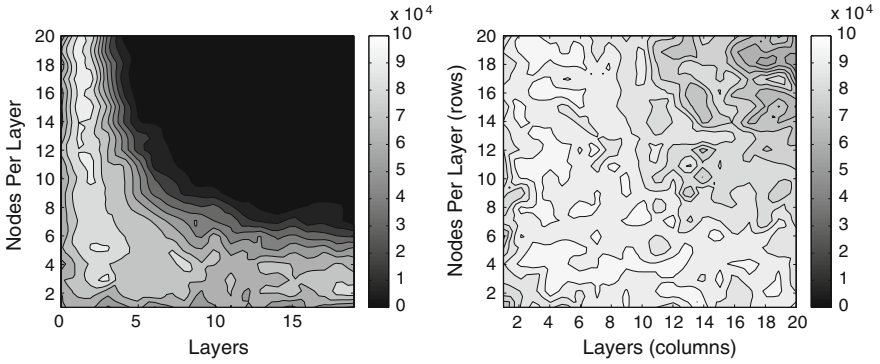

Fig. 2 Double Pole test problem results. *Left*: Fixed topology only evolving weights. *Right*: Random initial topology evolving weights and topology. The higher values (lighter) represent a better fitness. The same fitness scale (time steps balanced) is used for both figures for comparison

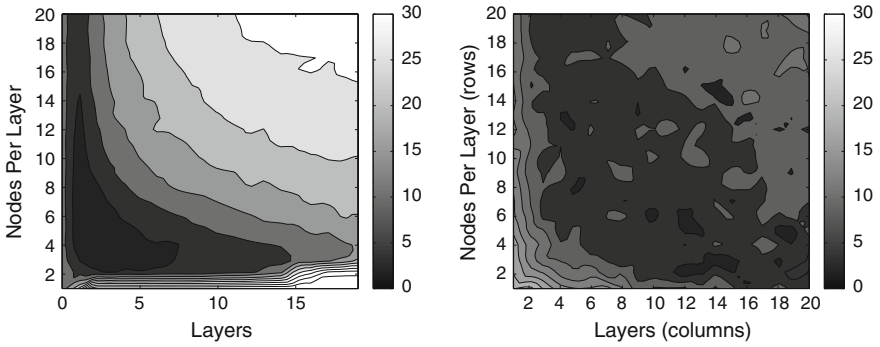

Fig. 3 Monks Problem test problem results. *Left*: Fixed topology only evolving weights. *Right*: Random initial topology evolving weights and topology. The lower values (darker) represent a better fitness. The same fitness scale (percentage error) is used for both figures for comparison

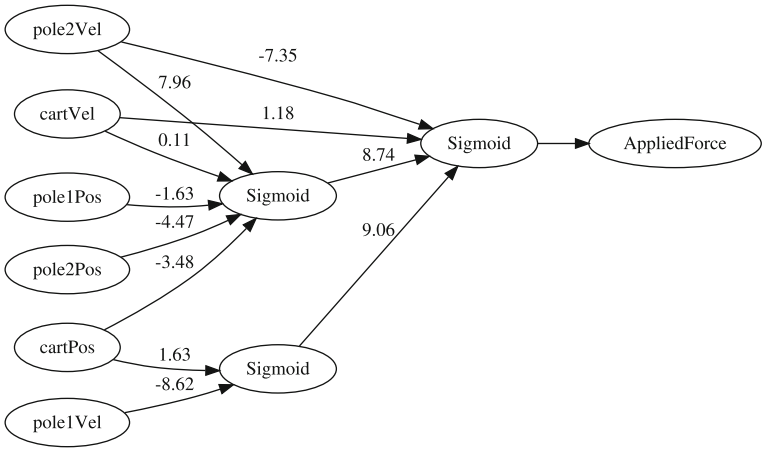

Fig. 4 Example of ANN which scored maximum fitness on the Double Pole test problem. Created using standard CGPANN with both row and column limits set as three

shown previously [4] but, to the authors knowledge, never in so much detail and so explicitly. Similar results are also found when training ANN using back propagation [6]; highlighting the importance of topology across training methods. It can also be seen that when only weights are evolved, the most effective topologies use a small number of layers with many nodes per layer; topologies which are often used when using traditional weight only methods, as other topologies are harder to train [6]. When using CGPANN to evolve both weights and topology, it can be seen that the effectiveness of the search is much more uniform across a range of topology constraints; the issue of selecting a suitable topology prior to training the ANN is vastly reduced. An example of the type of topology created when using CGPANN is given in Fig. 4. Clearly the generated topology is very unconventional, with no clear layers and variable node arity.

5.2 Weight Evolution Versus Topology Evolution

The results of the second experiment, comparing the relative effects of weight evolution and topology evolution, are shown in Fig. 5 for a range of mutation rates.

The results are also analysed using non-parametric statistical tests; as it is likely the distributions are non-normal. The Mann-Whitney U-test and the Kolmogorov-Smirnoff test (KS) are used to identify if the differences between approaches are statistically significant; with $\alpha = 0.05$ in both cases. The effect-size (A), as defined by Vargha et al. [16], is also used to indicate the importance of any statistical difference; with values >0.56 indicating a small effect size, >0.64 a medium and >0.71 a large. The results of these statistical tests are given for both the Double Pole and the

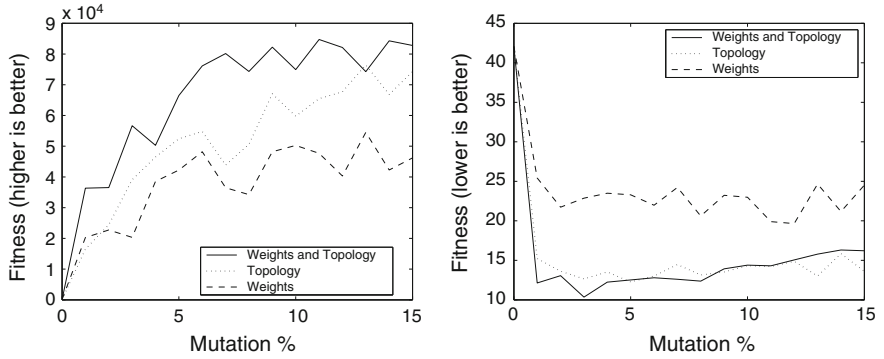

Fig. 5 Comparison of only weight, only topology and weight and topology evolution for Double Pole (*left*) and Monks Problem (*right*) test problems. Higher values represent a better fitness a for the Double Pole test problem and lower for the Monks Problem

Table 2 Statistical analysis of Double Pole and Monks Problem test problem

Problem	Comparison	Mutation %	U-test	KS	A
Double Pole	W & T | T	3	0.176	0.358	0.574
Double Pole	W & T | T	5	0.141	0.508	0.576
Double Pole	W & T | W	3	2.31×10^{-6}	2.76×10^{-6}	0.766
Double Pole	W & T | W	5	2.59×10^{-3}	1.71×10^{-2}	0.661
Double Pole	W | T	3	4.43×10^{-5}	1.78×10^{-4}	0.734
Double Pole	W | T	5	8.11×10^{-2}	3.12×10^{-2}	0.596
Monks Problem	W & T | T	3	0.124	0.241	0.589
Monks Problem	W & T | T	5	0.593	0.358	0.531
Monks Problem	W & T | W	3	9.46×10^{-6}	1.78×10^{-4}	0.757
Monks Problem	W & T | W	5	2.67×10^{-5}	2.76×10^{-5}	0.743
Monks Problem	W | T	3	8.35×10^{-4}	4.43×10^{-3}	0.694
Monks Problem	W | T	5	2.48×10^{-5}	4.23×10^{-4}	0.744

| separates the methods under comparison. *W* weight evolving, *T* topology evolving, *W & T* weight and topology evolving

Monks Problem test problems at three and five percent mutation rates (commonly used mutation values); see Table 2.

It can be seen from Fig. 5 that topology evolution is providing a better results than weight evolution when used independently. This is confirmed to be statistically significant in all cases examined except for the rank-sum test on the Double Pole test problem using a five percent mutation rate. The effect-size values for the weight only and topology only comparison also indicate that the statistical difference is important. This same statistical difference is also present when comparing the use of topology and weight evolution with weight evolution alone. However there is no statistical difference between the use of weight and topology evolution with topology evolution alone. This confirms the unexpected result that topology evolution is significantly more important to the search for effective neural networks than weight evolution.

6 Evaluation

Figures 2 and 3 show that only evolving connection weights leads a to search which is highly dependent upon the topology used. However, by allowing both topology and weight evolution this issue is drastically reduced. It is true that when only evolving connection weights using standard topologies, a few layers and many nodes per layer, good results are produced. However, it can also be seen in Fig. 2 that topologies which have many layers and few nodes per layer also produced good results. It is clear that simply using standard topologies ignores large areas of potentially beneficial search space. In addition, for more challenging problems a suitable network size would not be known in advance.

In Fig. 3, the Monks Problem test problem, it is seen that selecting a suitable topology and then evolving only the weights produces better results than evolving both topology and weights. This result is unsurprising as evaluating the topologies in this way is likely to find suitable locations in the topology space. As with any search, if domain knowledge is known in advance then this can be used to restrict the search space to suitable areas. Interestingly, this was not the case in Fig. 2, where evolving topology produced a better result in nearly all cases.

In the second experiment, we obtained the surprising result that topology evolution alone is significantly superior to weight evolution alone when training ANNs; when using random initial weights and topologies. It has long been thought that topology manipulation offers an advantage when training ANNs. This paper has not only empirically demonstrated that topology manipulation is advantageous, but that it offers benefits in the search more than weight manipulation.

To confirm these results future experiments should include additional test problems and the use of other topology manipulating training methods, such as developmental methods [19]. At the very least however it can be said that CGPANN appears to benefit highly from topology evolution; far more in fact, than weight evolution.

7 Conclusion

This paper has shown the significant benefits of using NE methods, which evolve both weights and topologies. These methods are not dependent upon selecting the correct topology before the search begins and are also strongly aided by the presence of topology manipulation. The surprising result that topology manipulation is more important than weight mutation has also been presented. This result may be one of the reasons some NE methods are so successful at training ANNs. CGPANN has been used throughout this investigation and has been shown to be very versatile; capable of using fixed user chosen topologies, randomly generated topologies, only evolving weights, only evolving topology and evolving both weights and topology. CGPANN is also capable of evolving the activation function used within the neurons; giving CGPANN complete control of the evolved ANN.

References

1. P. Angeline, G. Saunders, and J. Pollack. An Evolutionary Algorithm that Constructs Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, 1994.
2. D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from Architectures to Learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.
3. X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and, Statistics*, 2010.
4. C. Igel. Neuroevolution for Reinforcement Learning using Evolution Strategies. In *Evolutionary Computation*, volume 4, pages 2588–2595. IEEE, 2003.
5. M. M. Khan, G. M. Khan, and J. F. Miller. Evolution of Neural Networks using Cartesian Genetic Programming. In *Proceedings of IEEE World Congress on Computational Intelligence*, 2010.
6. H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring Strategies for Training Deep Neural Networks. *The Journal of Machine Learning Research*, 10:1–40, 2009.
7. J. Miller and S. Smith. Redundancy and Computational Efficiency in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.
8. J. Miller and P. Thomson. Cartesian Genetic Programming. In *Proceedings of the Third European Conference on Genetic Programming (EuroGP2000)*, volume 1802, pages 121–132. Springer-Verlag, 2000.
9. J. F. Miller. What bloat? Cartesian Genetic Programming on Boolean Problems. In *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 295–302, 2001.
10. J. F. Miller, editor. *Cartesian Genetic Programming*. Springer, 2011.
11. D. Moriarty and R. Mikkulainen. Efficient Reinforcement Learning through Symbiotic Evolution. *Machine learning*, 22(1):11–32, 1996.
12. R. Poli. Some Steps Towards a Form of Parallel Distributed Genetic Programming. In *Proceedings of the First On-line Workshop on, Soft Computing*, pages 290–295, 1996.
13. K. Stanley and R. Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary computation*, 10(2):99–127, 2002.
14. S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S. Fahlman, D. Fisher, et al. The Monk's Problems a Performance Comparison of Different Learning Algorithms. Technical report, Carnegie Mellon University, 1991.
15. A. J. Turner and J. F. Miller. Cartesian Genetic Programming encoded Artificial Neural Networks: A Comparison using Three Benchmarks. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-13)*, pages 1005–1012. ACM, 2013.
16. A. Vargha and H. D. Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.
17. V. K. Vassilev and J. F. Miller. The Advantages of Landscape Neutrality in Digital Circuit Evolution. In *Proc. International Conference on Evolvable Systems*, volume 1801 of LNCS, pages 252–263. Springer Verlag, 2000.
18. A. Wieland. Evolving Neural Network Controllers for Unstable Systems. In *Neural Networks, 1991, IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 667–673. IEEE, 1991.
19. X. Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
20. T. Yu and J. F. Miller. Neutrality and the Evolvability of a Boolean Function Landscape. *Genetic programming*, pages 204–217, 2001.