Using GP Is NEAT: Evolving Compositional Pattern Production Functions

Filipe Assunção[✉], Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro

CISUC, Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal
{fga,naml,machado,bribeiro}@dei.uc.pt

Abstract. The success of Artificial Neural Networks (ANNs) highly depends on their architecture and on how they are trained. However, making decisions regarding such domain specific issues is not an easy task, and is usually performed by hand, through an exhaustive trial-and-error process. Over the years, researches have developed and proposed methods to automatically train ANNs. One example is the HyperNEAT algorithm, which relies on NeuroEvolution of Augmenting Topologies (NEAT) to create Compositional Pattern Production Networks (CPPNs). CPPNs are networks that encode the mapping between neuron positions and the synaptic weight of the ANN connection between those neurons. Although this approach has obtained some success, it requires meticulous parameterisation to work properly. In this article we present a comparison of different Evolutionary Computation methods to evolve Compositional Pattern Production Functions: structures that have the same goal as CPPNs, but that are encoded as functions instead of networks. In addition to NEAT three methods are used to evolve such functions: Genetic Programming (GP), Grammatical Evolution, and Dynamic Structured Grammatical Evolution. The results show that GP is able to obtain competitive performance, often surpassing the other methods, without requiring the fine tuning of the parameters.

Keywords: Compositional Pattern Production Functions
NeuroEvolution of Augmenting Topologies · Genetic Programming
Grammatical Evolution · Dynamic Structured Grammatical Evolution

1 Introduction

Artificial Neural Networks (ANNs) are hard to train. A lot of algorithmic design choices are involved, and learning is often tuned using an iterative, and cumbersome, trial-and-error process. One of the major decisions is selecting the learning algorithm to use, along with all the needed parameters, which, in turn, have to be optimised. The majority of the learning algorithms are gradient-descent and, as such, have a high probability of becoming trapped in local optima.

One of the goals of NeuroEvolution (NE) is the automatisation of learning in ANNs. To that end, NE applies Evolutionary Computation (EC) applies to search and tune the best solutions for the values of the weights and bias of ANNs. There are many approaches to that end: evolve the learning algorithm parameters, the actual learning rules, or directly tune the weights and bias values of the networks. The approach adopted in this paper is based on the premise that the network's weights follow some pattern that can be learned. Assuming that this pattern exists, it may be easier to evolve a function that outputs the weights of the connections between nodes than evolving the weights directly.

HyperNEAT [19] is based on the use of NeuroEvolution of Augmenting Topologies (NEAT) [20] to evolve Compositional Pattern Production Networks (CPPNs), which are structurally similar to ANNs, and are used as a means to encode the weights of a network. At a high level, CPPNs are a function that, given the position of two neurons, outputs the synaptic weight of the connection between the two given neurons. As such, and since a CPPN can be seen as a function mapping the coordinates of a pair of nodes into a weight, instead of evolving CPPNs one can use conventional EC techniques, such as GP, to evolve functions to the same task. Such functions are known as Compositional Pattern Production Functions (CPPFs) [7]. Since, in essence, CPPNs are CPPFs that use the representation adopted by NEAT, from here on we will refer to both as CPPFs except when it is necessary to make a distinction. Thus, a CPPF evolved by NEAT is actually a CPPN.

In the current work we apply different methods to the optimisation of CPPFs. More precisely, we optimise CPPFs using NEAT, Genetic Programming (GP), Grammatical Evolution (GE), and Dynamic Structured Grammatical Evolution (DSGE). We test these approaches in two different problems: a visual discrimination, and a line following task. Each problem has two setups of different complexity. The results show that GP consistently obtains the best results.

The remainder of the paper is organised as follows. In Sect. 2 we introduce works related with the evolutionary training of ANNs. Then, in Sect. 3, we detail the specifications of CPPFs and of each approach that is going to be used to evolve them. In Sect. 4 experiments are conducted and the results analysed. To end, in Sect. 5, conclusions are drawn, and future work is addressed.

2 Related Work

The use of Evolutionary Algorithms (EAs) to promote learning in ANNs not only avoids the need for manual tuning, but also reduces the risk of getting stuck in local optima: instead of a having a single solution, we have a population of solutions that is evolved. Each candidate solution encodes a trained network and their quality is determined according to the performance on a specific task. There are many approaches to the evolutionary learning of ANNs; they can be divided into three groups: optimisation of the (i) learning algorithm parameters [12,17]; (ii) learning rules [5,18]; or (iii) weights and bias values [3,8,10,21].

Despite the high number of works on the automatic training of ANNs, in the current paper our focus is on approaches that evolve the learning rules.

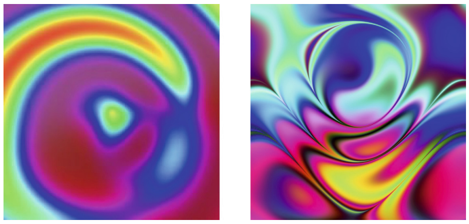

Fig. 1. Example of an image generated by HyperNEAT (left), and NEvAr (right).

More particularly, we are interested in the methods that explore the spatial regularity of the networks. HyperNEAT [19] aims at evolving CPPNs, which can be described as similar to ANNs (in structural terms). CPPNs are able to capture and encode patterns and correlations among network weights. To promote the evolution of the CPPNs it is possible to use NEAT [20]. NEAT was primarily designed as a NE approach for the optimisation of the topology and weights of ANNs; but, since CPPNs are structurally similar to ANNs they can also be evolved by NEAT.

Looking closely at HyperNEAT, what is actually evolved is a function that given spatial data provides an output. For example, one of the first applications of HyperNEAT was Picbreeder [1] – a web platform for evolving images collaboratively. In Picbreeder, the evolved CPPNs take as input two arguments: x, y, which are the coordinates of a pixel in the image; the output is the intensity. Before Picbreeder there were similar systems that aimed at evolving images; one of such methods is NEvAr [15]. The main difference between the two approaches relies on the mechanism used to evolve the function that generates the pixel value: while in Picbreeder the authors rely NEAT, in NEvAr tree-based GP is used to evolve a function, as in common symbolic regression approaches.

The fact that NEAT and GP have been applied to the evolution of functions to generate images is not within the scope of the current work; however it illustrates how both approaches can produce similar results in terms of the images that are generated (check Fig. 1). Additionally, both approaches have already been used in the evolution of functions that map the position of any pair of two nodes into a weight value, i.e., a rule that defines what are the weights of a target network. Stanley et al. trained multiple neuronal controllers using NEAT [20]; Buk et al. followed the same rationale but using GP [7]. The results of [7] show that using GP or NEAT it is possible to find adequate solutions; however, convergence seems to be faster with GP.

3 Generation of CPPFs

The main goal of the current work is to compare the performance of different approaches in the evolution of CPPFs for the training of ANNs. Figure 2 depicts the interaction between evolution, the generated CPPFs and the substrate, which is the network that is trained to solve a specific problem. Four

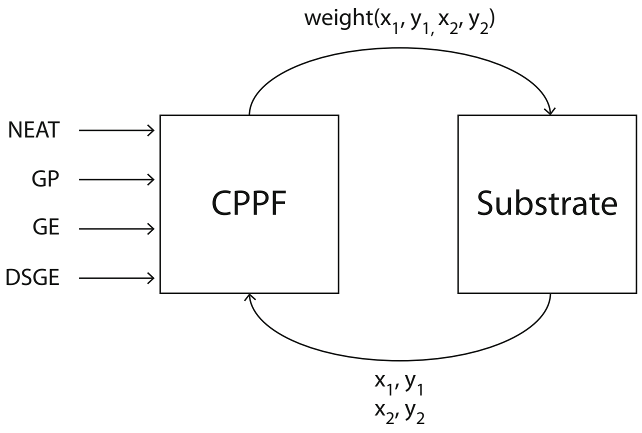

Fig. 2. Overview of the current work, evidencing the interaction between the CPPFs and the substrate.

different approaches are tested: NEAT, GP, GE, and DSGE. These are briefly described in Sects. 3.1, 3.2, 3.3, and 3.4, respectively. CPPFs are functions with 4 inputs: x_1, y_1, x_2, y_2, which similarly to HyperNEAT, are the positions of the neurons in the substrate. The substrate is the ANN that is used to solve the problem at hand. It is a grid of neurons, with a fixed number of inputs and outputs that varies according to the problem to be solved. To know the weights of the connections between the neurons in the substrate we need to query the evolved CPPF; if it returns a value above a defined threshold then the connection between the input neurons ((x_1, y_1) and (x_2, y_2)) exists and the synaptic weight is the one returned by the CPPF; conversely, if the value is bellow the threshold, the connection does not exist.

3.1 NeuroEvolution of Augmenting Topologies

NeuroEvolution of Augmenting Topologies (NEAT) [20] is a NE approach, where each candidate solution encodes an entire network as a list of neurons along with the connections between them. Whilst the majority of other NE approaches initialise the population at random, in NEAT evolution starts from a minimal structure, i.e., initial networks are composed of no hidden-nodes, and thus the input nodes are directly connected to the output nodes. Other novel aspects of NEAT include innovation protection techniques and speciation. In its vanilla form the only genetic operator that is applied to generate offspring is mutation, which aims at changing any node or structural property of a network.

CPPNs are structurally similar to ANNs, and thus it is possible to evolve them using NEAT and its principles. As such, the target of evolution is a network structure, with four inputs and one output, where the activation function of each node is selected from a defined set, and the connections are evolved and have a weight associated to it, as in ANNs. This approach is known as HyperNEAT [19].

3.2 Tree-Based Genetic Programming

In its standard form Genetic Programming (GP) [13] encodes the solutions as trees, where the inner nodes represent functions and leaves represent terminals. The crossover operator exchanges sub-trees between parents; mutation acts similarly to the crossover operator: a random sub-tree of the individual that is to be mutated is replaced by another valid one.

There are several approaches where GP is used to evolve learning rules of ANNs (e.g. [5,18]). GP is well-known for its results in symbolic regression tasks. Typically this assumes evaluating the evolved individuals for different input values. Likewise, the evolution of CPPFs (or CPPNs using HyperNEAT) involves the evaluation of the individuals for all pairs of neurons. The difference is that while in symbolic regression one typically compares the output values with the desired ones, in the evolution of CPPFs one does not know the desired value and, therefore, fitness is assigned according to the performance of the network on a given task. Due to the similarities between symbolic regression and CPPF evolution we consider GP to be a natural choice for this task, and we expect it to be able of discovering effective functions that encode the substrate weights.

3.3 Grammatical Evolution

Grammatical Evolution (GE) [16] uses an indirect encoding to represent solutions as derivations of a user-defined grammar. One of the advantages of this type of approaches is that they are easily generalisable to deal with different domains, just requiring the change of the grammar production rules. Candidate solutions are encoded as a linear sequence of integers, where each integer represents the possibility to further expand a given non-terminal symbol. To that end, the mathematical modulus operation is used, and the expansion possibility is equal to the integer modulus the number of possibilities for expanding the current non-terminal symbol. In GE both mutation and crossover genetic operators are used; mutation randomly changes an integer to another one; one-point crossover is used.

Like in GP, the majority of the works concern the evolution of the weights of the networks (e.g. [2]). Due to its similarities with GP we will focus on the generation of CPPFs.

3.4 Dynamic Structured Grammatical Evolution

Dynamic Structured Grammatical Evolution (DSGE) [4,14] is another grammar-based GP approach. The main difference between GE and DSGE relies on the way the genotype is encoded. While in GE a single list of integers is used, in DSGE there is a list of integers for each non-terminal symbol. This encapsulation of the genetic material promotes locality, in the sense that there is a direct association between the non-terminal symbols and the integers used for their mapping. In addition, the modulus is no longer needed, thus avoiding the redundancy issue commonly pointed out as a disadvantage of GE. In DSGE only the non-terminal symbols that are used are encoded, and consequently the variation operators are applied to integers that are effectively used in the genotype to phenotype mapping.

To promote evolution crossover and mutation are applied; bit-mask crossover is used, where codons (i.e., the set of integers associated to a specific non-terminal symbol) are changed between two parents; the mutation operator is a standard per gene point mutation. For the same reason than in GP and GE we will tackle the evolution of CPPFs using DSGE.

4 Experiments

We conduct experiments with the four evolutionary methodologies described above: NEAT, GP, GE, and DSGE. The objective of the the experiments is the evolution of CPPFs for the training of neuronal controllers for solving two specific tasks, which are described next.

4.1 Problems Description

The experiments are conducted in two different tasks: (i) visual discrimination; and (ii) line following. We selected these problems because they are common benchmarks used by HyperNEAT. Moreover, its performance has been thoroughly studied, and it is possible to compare the approaches without introducing any problem dependency bias. In the upcoming sub-sections we briefly describe the tasks, and the structure of the substrates that are used to solve them.

Visual Discrimination

This problem was one of the first to be used to demonstrate the effectiveness of HyperNEAT [19]. The objective of this visual computation task is to distinguish between two different objects – a target and a distractor – independently of their positions in a field. We test the evolution of CPPFs for two different setups. The two setups have the same input dimensions: a 11×11 image, but the targets and distractors differ. In the big-little setup the target is a 3×3 square, and the distractor is a 1×1 square:

$$\text{target}_{\text{square}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}, \text{distractor}_{\text{square}} = \begin{bmatrix} 1 \end{bmatrix}.$$

To increase the complexity of the task, we then experiment with a triangular target and distractor, which have the same dimensions, but are mirrored (triup-down setup):

$$\text{target}_{\text{triangle}} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & \underline{1} & 0 \\ 1 & 1 & 1 \end{bmatrix}, \text{distractor}_{\text{triangle}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

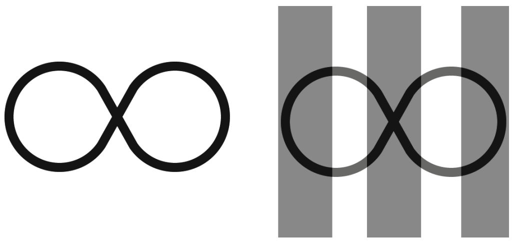

Fig. 3. Line following task. On the left the road that the agent must follow. On the right the same road path with different friction areas.

The identification of the target shape is performed by a neuronal controller, which is trained using a CPPF. To identify the shape, the trained network must generate the highest activation at the center of the target shape. Therefore the goal of evolution is the minimisation of the distance between the response provided by the network and the target center (underlined 1 in the target matrices). To further complexify the problem, as in [7], the positions of the distractor are set at random. This makes the fitness function non-deterministic, which means that two consecutive evaluations can provide two different fitness values.

For this task, the CPPFs receive 6 inputs: additionally to the usual coordinates of the nodes x_1, y_1, x_2, y_2, there are also two delta values $x_1 - x_2$, and $y_1 - y_2$ (as in [6]). The substrate consists of a sandwich network [19], i.e., the input layer is directly connected to the output layer, and both have the same size (in this case 121 neurons, one for each pixel of the image).

Line Following

In the line following task [9] the goal is to evolve the controllers of an agent so that it can effectively navigate a road, i.e., follow a line at maximum speed. The map is made of regions with different friction rates, and thus the agent should strive to steer in those with the lowest resistance. Two setups are tested: in the first one all regions except the road have the same friction rate (Fig. 3, left); on the second there are regions of different friction (marked as the darker stripes on the right side of Fig. 3). The regions outside the road have a friction that is 5 times higher than on the road.

The agent is a robot with two wheels and 5 sensors, each with a range of 3 pixels. The sensors are placed to the front of the agent, and provide a read of the characteristics of the field within range. The substrate is a feedforward network with 15 inputs (which are fed with the sensors data), a hidden-layer, and 2 outputs (that control each of the wheels).

The evolved CPPFs receive the standard 4 inputs, but have 3 outputs, each responsible for encoding a function that represents the weights of a specific part of the substrate: input to output, hidden-connections, and output layer bias. When evolving CPPNs there is no problem in having three different functions, each represented by a different output neuron. However, to accomplish the same with CPPFs authors typically evolve three different functions simultaneously

Table 1. Experimental parameters.

Parameter	**Value**
Number of runs	30
Number of generations	250 / 100
Population size	100
Elite size	1%
Tournament size	3
NEAT Parameter	**Value**
Weight range	(-3, 3)
Minimum weight	0.3
Add node probability	0.03
Add connection probability	0.1
Mutate weight probability	0.8
Reset weight probability	0.1
Reenable connection probability	0.01
Disable connection probability	0.01
Mutate bias probability	0.2
Mutate node type probability	0.2
Std weight mutation	0.2
Std bias mutation	0.5

GP Parameter	**Value**
Crossover probability	0.9
Mutation probability	0.1
Maximum tree-depth	17
GE Parameter	**Value**
Codon size	127
Wrapping	2
Crossover probability	0.9
Mutation probability	0.05
DSGE Parameter	**Value**
Crossover probability	0.9
Mutation probability	$^1/_{\text{codon\_size}}$
Maximum recursivity	17

(as in [7]). We followed a different direction, and evolve a single function that outputs a vector of size 3. This can be easily accomplished by using a function set that operates on vectors and scalars (the multiplication of two vectors performs the dot product; the same happens for division; the trigonometric operations are applied to each one of the components of the vector); and random constants which are vectors of size 3. This approach has also been used in NEvAr [15].

4.2 Experimental Setup

To promote a fair comparison, we adopted and adapted the vanilla implementations of each of the evolutionary engines, which are easily found in public repositories[1]. Table 1 details the parameterisation of the different algorithms. For the two benchmarks and respective setups the parameters are the same, except for the number of generations of each run, which is 250 for the visual discrimination task setups, and 100 for line following.

For the grammar-based approaches (GE and DSGE) we use the grammar of Fig. 4. This grammar is capable of generating CPPFs that provide the weights for the substrate based on multiple inputs; for the visual discrimination task x_1, y_1, x_2, y_2, d_1, and d_2: the first 4 are the positions of the neurons in the substrate

[1] NEAT – https://github.com/noio/peas [6].
GP – https://github.com/DEAP/deap.
GE – https://github.com/jmmcd/ponyge.
DSGE – https://github.com/nunolourenco/dsge.

$$\begin{aligned}
\langle\text{expr}\rangle &::= \langle\text{expr}\rangle\,\langle\text{op}\rangle\,\langle\text{expr}\rangle \\
&\quad\mid \langle\text{var}\rangle \\
&\quad\mid \langle\text{preop}\rangle\,(\langle\text{expr}\rangle) \\
\langle\text{var}\rangle &::= x1 \mid y1 \mid x2 \mid y2 \\
&\quad\mid d1 \mid d2 \\
&\quad\mid \langle\text{float}\rangle \\
\langle\text{preop}\rangle &::= + \mid - \mid * \mid / \\
\langle\text{preop}\rangle &::= sin \mid - \\
\langle\text{float}\rangle &::= -\ \langle\text{first}\rangle.\langle\text{number}\rangle\langle\text{number}\rangle \\
&\quad\mid \langle\text{first}\rangle.\langle\text{number}\rangle\langle\text{number}\rangle \\
\langle\text{first}\rangle &::= 0 \mid 1 \mid 2 \\
\langle\text{number}\rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \\
&\quad\mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}$$

Fig. 4. Grammar used for evolving CPPFs with GE and DSGE. For the line following task in the <var> rule the terminal symbols d1 and d2 are removed from the grammar, and the last expansion possibility of <var> is replaced by [<float>, <float>, <float>].

and the last are the deltas, $x_1 - x_2$ and $y_1 - y_2$, respectively. In the line following task the deltas are not considered.

We decided to use function sets that are commonly used in conjunction with each one of the approaches. As such, when evolving CPPFs – with tree or grammar-based GP – for the visual discrimination tasks we consider a simple function set: sin, addition, subtraction, multiplication, and division. The terminals are the ones used by the grammar of Fig. 4, i.e., the inputs of the CPPF, and random float values that may range between -3 to 3. In NEAT the nodes of the network can use the following activation functions: sin, bound, linear, gaussian, sigmoid, and absolute value. When evolving CPPFs for the line following tasks, the function set is expanded in order to handle vectors and the random constants become random vectors of size 3. No changes are required to the function set of NEAT.

4.3 Experimental Analysis

In each experiment we perform 30 independent evolutionary runs so that we can understand the behaviour of the methods in each of the tested problems and setups. We analyse fitness evolution and convergence speed. In addition, a statistical analysis is performed to assess if any of the approaches is statistically superior to the others in terms of the quality of the evolved solutions.

Visual Discrimination

In the visual discrimination task the goal is to reduce the distance to the center of the target shape, and consequently fitness is to be minimised; recall that the

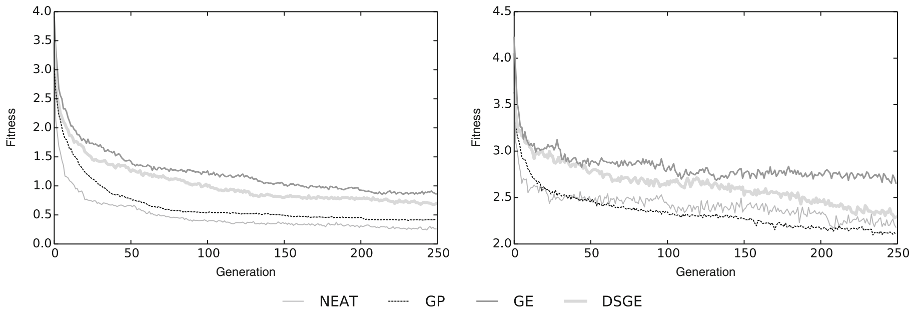

Fig. 5. Evolution of the best individuals across generations in the visual discrimination task for the big-little (left) and triup-down (right) setups. Results are averages of 30 independent runs.

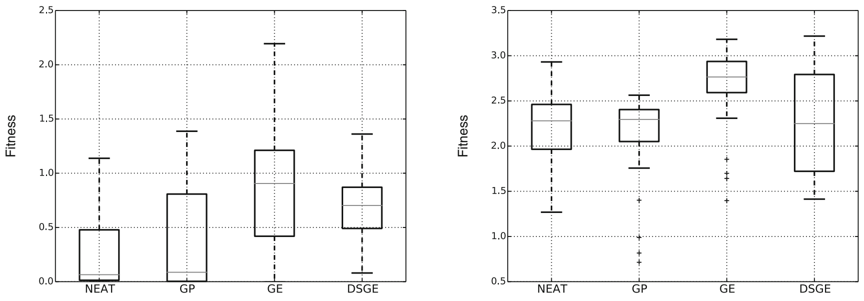

Fig. 6. Analysis of the fitness of the best solutions of the visual discrimination task using box plots. On the left the big-little setup, and on the right the triup-down.

fitness function is not deterministic, and thus the fitness of the same individual evaluated multiple times can vary. Figure 5 depicts, for both setups, the evolution of fitness across generations. The results are averages of the 30 best solutions, one from each of the evolutionary runs. These charts show that with any of the evolutionary engines evolution is promoted and there is convergence. The difference in the setups complexity is noticeable by the analysis of the difference in the fitness scales: in both setups the average fitness of the best solutions starts approximately from the same point, but in the big-little setup it is capable of reaching much lower values than in the triup-down setup. Having the target and distractor shapes with the same size but mirrored makes the problem too challenging for an appropriate function capable of encoding the weights of the substrate to be found in the given number of generations.

Nonetheless, for both setups, in terms of fitness, the results reported by NEAT and GP are superior to those of the grammar-based methods. To better analyse the quality of the generated solutions we use box plots, focusing on the distribution of the quality of the best individuals from each evolutionary run (see Fig. 6). From the box plots it is possible to see that GE has the worse performance. Focusing on each of the setups individually, in the case of the

Table 2. Graphical overview of the statistical results of the visual discrimination experiments with effect sizes for the big-little (left) and triup-down (right) setups.

	NEAT	GP	GE	DSGE
NEAT		$\sim$	+++	+++
GP	$\sim$		+++	++
GE	$\sim$	$\sim$		$\sim$
DSGE	$\sim$	$\sim$	$\sim$	

	NEAT	GP	GE	DSGE
NEAT		$\sim$	+++	$\sim$
GP	$\sim$		+++	$\sim$
GE	$\sim$	$\sim$		$\sim$
DSGE	$\sim$	$\sim$	++	

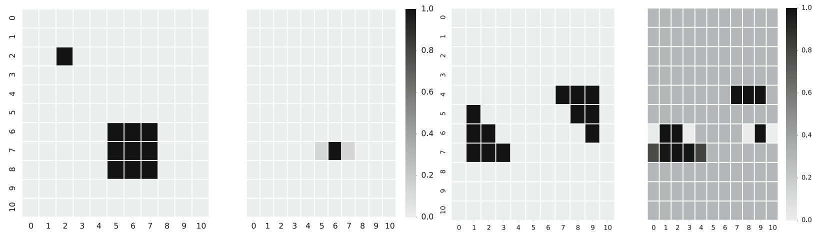

Fig. 7. The left and right figures represent the activations generated by one of the best solutions (discovered using GP), for each of the setups: big-little and triup-down, respectively.

big-little setup NEAT and GP have a close median, with GP having a slightly superior dispersion. For the triup-down setup NEAT and GP also have roughly the same median, but the dispersion is lower in GP. GP has more outliers, but these outliers correspond to runs that achieve better solutions. Looking at the DSGE performance, in the big-little setup it performs worse than NEAT or GP, and in the triup-down setup it has a similar performance, but larger dispersion.

The analysis of the generated solutions reveals that, for the big-little setup, NEAT, GP, GE, and DSGE generate perfect solutions in 5, 8, 3, and 0 out of the 30 runs, respectively. So, despite GP having a slightly higher median and dispersion in the fitness values, it is the approach that finds solutions with a perfect performance most often. In the triup-down setup, no approach is capable of finding a perfect solution.

To better understand if any of the approaches is superior to the others we conduct a statistical study. 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 reveal that the data does not follow a normal distribution and, as such, a non-parametric test (Mann-Whitney U, $\alpha = 0.05$) is used to perform the pairwise comparison of the approaches (with Bonferroni correction). Table 2 presents a graphical overview of the results of the statistical analysis: $\sim$ indicates no statistical difference, and + that the approach in the row is statistically better than the one in the column. The effect size is a measure that quantifies the strength of a phenomenon (larger values mean a stronger effect), and 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 results show that it is not possible to point out a single approach as the best one; there are no differences between NEAT and GP. GE is surpassed by all other approaches, and DSGE is outperformed in the triup-down setup, but has the same performance as NEAT and GP in the big-little.

Figure 7 presents examples of one of the best solutions (for each setup) regarding the activations that are generated for the identification of the target shapes: darker colours mean higher activation values. As perceptible, in the big-little setup the target shape is correctly identified; in the triup-down the trained network fails to identify the target shape, and is activated by parts of the target and distractor shapes.

Line Following

Contrary to the visual discrimination task, the goal of the line following task is the maximisation of the average speed of a robot in a road navigation task, i.e., maximise the distance travelled in a fixed amount of time (3000 time steps). As before, we start by analysing the evolution of fitness across generations (see Fig. 8). GE is the approach that takes the largest number of generations to converge, reaching the lowest results. Conversely, NEAT, GP, and DSGE are the methods that generate the best solutions, with GP outperforming the other two in the easy and hard setups. The box plots allow stronger conclusions than before (see Fig. 9); while in the visual discrimination task NEAT or GP were not superior in the two setups, in the line following task it is perceptible that GP performs better than the remaining approaches in the easy and hard setups, i.e., despite having a few outliers GP has a median that is higher than those of the remaining approaches, and the interquartile range is smaller, meaning that the results are more consistent.

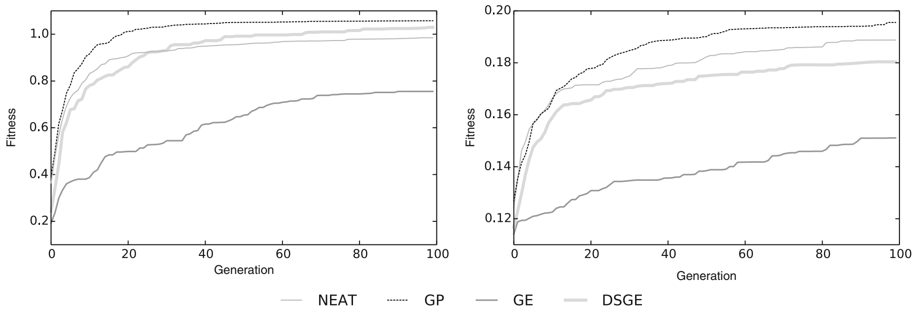

Fig. 8. Evolution of the best individuals across generations in the line following task for the easy (left) and hard (right) setups. Results are averages of 30 independent runs.

To strengthen our analysis, we perform a statistical analysis of the results. The results are reported in Table 3, using the same graphical representation of

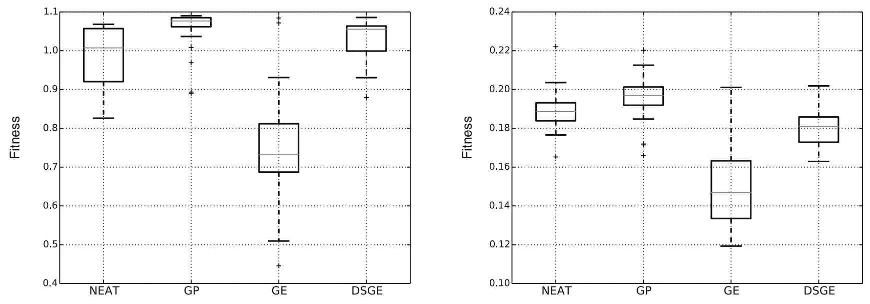

Fig. 9. Analysis of the fitness of the best solutions of the line following task using box plots. On the left the easy setup, and on the right the hard.

Table 3. Graphical overview of the statistical results of the line following experiments with effect sizes for the easy (left) and hard (right) setups.

	NEAT	GP	GE	DSGE
NEAT		∼	+++	∼
GP	+++		+++	++
GE	∼	∼		∼
DSGE	+++	∼	+++	

	NEAT	GP	GE	DSGE
NEAT		∼	+++	+++
GP	+++		+++	+++
GE	∼	∼		∼
DSGE	∼	∼	+++	

the statistical analysis of the visual discrimination task. A brief perusal of the results shows that GP clearly outperforms the other approaches in the easy and hard setups. In fact, the effect size is always large, except in what concerns the comparison between GP and DSGE for the easy setup.

An example of the best models navigating in each of the setups is depicted in Fig. 10. The line marks the path followed by the robot. In the easy setup it is clear that the robot is capable of travelling through the road without any difficulty, never leaving the predefined path. The same cannot be stated for the hard setup, where the regions of different friction make learning more challenging

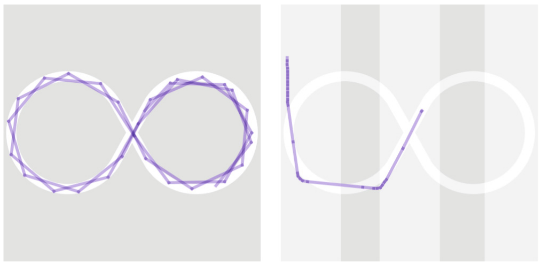

Fig. 10. The left and right figures represent an example of one of the best solutions (found using GP) for the easy and hard setups, respectively.

to the point that none of the evolved models is capable of completing an entire lap on the track in the given execution time; in fact the vast majority of them depict a behaviour similar to the one presented in the example, i.e., they leave the marked path and are unable of getting back to it.

Discussion

Summing up the previous results, we conclude that in the visual discrimination task no approach is superior to the remaining ones. The analysis of the results present in both the evolution and box plots shows that GP and NEAT have a similar performance. The statistical tests also confirm that there are no meaningful differences between the two approaches. Nevertheless, it is possible to say that GP is more effective, since it discovers perfect solutions most often, i.e., solutions with a distance of 0 to the target shape. On the big-little GP discovers perfect solutions 8 times out of 30 runs, and NEAT, GE, and DSGE discover perfect solutions in 5, 2, and 0 runs, respectively.

In the line following task GP outperforms all the other approaches considered in the comparison, which is clearly perceptible by the analysis of the box plots, where the fitness of GP is superior and the quality of the results consistent.

The performance of the grammar-based approaches is fairly disappointing in comparison with the remaining methods, which was an unexpected result. A possible explanation pertains the way float constants are created, which makes the search space larger. While in NEAT and GP the floats are just one terminal, in the grammar-based methods there is the need to associate an expansion possibility to each of the integers of the floats (which have a fixed precision). This might be mitigated by allowing both grammar-based approaches to perform a larger number of evaluations.

Based on the experiments conducted, it is possible to state that GP has better overall performance than NEAT. Additionally, GP requires far less parameterisation, without compromising the end results.

5 Conclusions and Future Work

ANNs are difficult to train, mainly due to the gradient-descent nature of the majority of the learning algorithms, and because of the complexity in setting the hyper-parameters required by the learning algorithms. Therefore, practitioners investigate methods that automatically search for the best weights of the networks. Some of these works focus on the automatic generation of learning rules that can map the network connections into appropriate weights and bias. HyperNEAT is an example of one of such approaches, and it is based on the use of NEAT to promote the evolution of CPPNs.

In this article we compare different evolutionary methods for the generation of CPPFs. Our research hypothesis is that it is possible to replace NEAT in HyperNEAT by a simpler method, that requires far less parameters, without compromising the overall quality of the obtained results. To validate our hypothesis, we apply NEAT, GP, GE and DSGE to the evolution of CPPFs in two benchmarks commonly used in HyperNEAT experiments: visual discrimination and line following, each having two setups that vary in complexity.

The experimental results, supported by statistical analysis, confirm our research hypothesis, i.e., they show that using tree-based GP it is possible to evolve effective CPPFs that, for the considered tasks, outperform the CPPFs discovered by the other methods, including NEAT. This result is somewhat surprising, specially taking into consideration that we used a vanilla implementation of GP and resorted to a basic and generic function set. As such we consider that these results pave the way for the application of GP approaches to the evolution of CPPFs, creating opportunities for the application of more sophisticated GP approaches to this type of tasks.

Future work will focus in four different research directions: (i) applying these approaches to a wider set of problems in order to assess the generality of the conclusions and identify problem specific limitations and strengths of different approaches; (ii) study the impact of the function and terminal sets in evolution; (iii) test how the compared approaches behave in terms of scalability; and (iv) expand the comparison to non-vanilla implementations and graph-based evolutionary methods, such as Cartesian Genetic Programming, which may be suitable for the evolution of CPPFs [11].

Acknowledgments. This work is partially funded by: Fundação para a Ciência e Tecnologia (FCT), Portugal, under the grant SFRH/BD/114865/2016, and is based upon work from COST Action CA15140: ImAppNIO, supported by COST (European Cooperation in Science and Technology): www.cost.eu.

References

1. Secretan, J., et al.: Picbreeder: evolving pictures collaboratively online. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1759–1768. ACM (2008)
2. Ahmadizar, F., Soltanian, K., AkhlaghianTab, F., Tsoulos, I.: Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm. Eng. Appl. Artif. Intell. **39**, 1–13 (2015)
3. Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: Automatic generation of neural networks with structured grammatical evolution. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 1557–1564, June 2017
4. Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: Towards the evolution of multi-layered neural networks: a dynamic structured grammatical evolution approach. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 393–400. ACM, New York (2017). http://doi.acm.org/10.1145/3071178.3071286
5. Bengio, S., Bengio, Y., Cloutier, J.: Use of genetic programming for the search of a new learning rule for neural networks. In: 1994 Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, pp. 324–327. IEEE (1994)
6. van den Berg, T.G., Whiteson, S.: Critical factors in the performance of HyperNEAT. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, pp. 759–766. ACM (2013)

7. Buk, Z., Koutník, J., Šnorek, M.: NEAT in HyperNEAT substituted with genetic programming. In: Kolehmainen, M., Toivanen, P., Beliczynski, B. (eds.) ICANNGA 2009. LNCS, vol. 5495, pp. 243–252. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04921-7_25
8. David, O.E., Greental, I.: Genetic algorithms for evolving deep neural networks. In: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 1451–1452. ACM (2014)
9. Drchal, J., Koutník, J., Snorek, M.: HyperNEAT controlled robots learn how to drive on roads in simulated environment. In: 2009 IEEE Congress on Evolutionary Computation, CEC 2009, pp. 1087–1092. IEEE (2009)
10. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. J. Mach. Learn. Res. **9**(May), 937–965 (2008)
11. Khan, M.M., Khan, G.M., Miller, J.F.: Evolution of neural networks using Cartesian genetic programming. In: 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2010)
12. Kim, H.B., Jung, S.H., Kim, T.G., Park, K.H.: Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates. Neurocomputing **11**(1), 101–106 (1996)
13. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. MIT Press, Cambridge (1992)
14. Lourenço, N., Pereira, F.B., Costa, E.: Unveiling the properties of structured grammatical evolution. Genet. Program Evolvable Mach. **17**(3), 251–289 (2016)
15. Machado, P., Cardoso, A.: All the truth about NEvAr. Appl. Intell. **16**(2), 101–118 (2002)
16. O'Neil, M., Ryan, C.: Grammatical evolution. In: O'Neil, M., Ryan, C. (eds.) Grammatical Evolution, pp. 33–47. Springer, New York (2003). https://doi.org/10.1007/978-1-4615-0447-4
17. Parra, J., Trujillo, L., Melin, P.: Hybrid back-propagation training with evolutionary strategies. Soft. Comput. **18**(8), 1603–1614 (2014)
18. Radi, A., Poli, R.: Discovering efficient learning rules for feedforward neural networks using genetic programming. In: Abraham, A., Jain, L.C., Kacprzyk, J. (eds.) Recent Advances in Intelligent Paradigms and Applications, pp. 133–159. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-7908-1770-6_7
19. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009)
20. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
21. Whitley, D., Starkweather, T., Bogart, C.: Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Comput. **14**(3), 347–361 (1990)