Genetic Programming of Augmenting Topologies for Hypercube-Based Indirect Encoding of Artificial Neural Networks

Jan Drchal and Miroslav Šnorek

Department of Computer Science and Engineering,
FEE CTU, Karlovo náměstí 13, 121 35, Praha 2, Czech Republic
{drchajan,snorek}@fel.cvut.cz

Abstract. In this paper we present a novel algorithm called GPAT (Genetic Programming of Augmenting Topologies) which evolves Genetic Programming (GP) trees in a similar way as a well-established neuro-evolutionary algorithm NEAT (NeuroEvolution of Augmenting Topologies) does. The evolution starts from a minimal form and gradually adds structure as needed. A niching evolutionary algorithm is used to protect individuals of a variable complexity in a single population. Although GPAT is a general approach we employ it mainly to evolve artificial neural networks by means of Hypercube-based indirect encoding which is an approach allowing for evolution of large-scale neural networks having theoretically unlimited size. We perform also experiments for directly encoded problems. The results show that GPAT outperforms both GP and NEAT taking the best of both.

1 Introduction

Recently, there has been a growing interest in the techniques which evolve large-scale artificial neural networks (ANNs). Classical training algorithms (such as back-propagation, second order optimization and evolutionary computation) suffer from poor convergence caused by the large dimension of the optimized synaptic weights (or other parameters). This problem was already successfully addressed by neuro-evolutionary approaches [1,2,3] which employ indirect encoding of ANNs. The idea of the indirect encoding is inspired by the Nature, where relatively short genomes encode highly complex structures. This immense compression of information is, among others, facilitated by regularities found at all scales of magnification.

This paper deals with a related state-of-the-art approach called HyperNEAT [4] which employes the so-called Hypercube-based indirect encoding and a well-established neuro-evolutionary algorithm NEAT. HyperNEAT works in a following way: at first, one have to decide for the structure of a *final network* (number and types of neurons and possible connections), also each neuron is assigned coordinates. Such template is called the *substrate*. Second, NEAT algorithm is used to evolve the CPPNs (Compositional and Pattern Producing Networks). The CPPN in HyperNEAT has a form of a common neural network, with an exception of using nodes with special transfer functions which are either symmetric, periodic or of a different type to reflect the above mentioned regularities found in living creatures. Third, the *final network* is constructed according to the *substrate*. The CPPN is then used to determine synaptic weights of all possible connections found in the *substrate* (coordinates of all pairs of neurons are fed to inputs). Fourth, the *final network* is evaluated on a problem domain to get the fitness value and provide a feedback to NEAT. HyperNEAT was successfully used to evolve networks having almost 8 millions of connections [4].

In [5] Buk et al. presented a modification of HyperNEAT, where NEAT was replaced by Genetic Programming (GP), as the *base algorithm* evolving CPPNs. It was shown to be superior in a speed of convergence to NEAT on a simulated robotic task. Here, we combine advantages of both GP and NEAT and propose a novel algorithm called Genetic Programming of Augmenting Topologies (GPAT). We compare all three *base algorithms* on problems having both directly and indirectly encoded genomes. GPAT in combination with Hypercube-based encoding will be denoted as HyperGPAT.

The paper[1] is organized as follows. Section 2 describes the theoretical background. Section 3 introduces GPAT algorithm. Section 4 describes the test problems and the experimental setup. Section 5 discusses the results. Final section concludes the paper.

2 Background

In this section, we briefly describe NEAT and GP base algorithms.

NEAT. NeuroEvolution of Augmenting Topologies (NEAT) [6] is an algorithm originally developed for evolution of both parameters (weights) and topology of artificial neural networks. It was later enhanced to produce the CPPNs with heterogenous nodes for the HyperNEAT algorithm instead of producing the neural networks directly. It works with genomes of variable size. NEAT introduced a concept of *innovation numbers*, which are gene labels allowing effective genome alignment in order to facilitate crossover-like operations. Moreover, innovation numbers are used for computation of a genotypical distance between two individuals. The distance measure is needed by niching evolutionary algorithm, which is a core of NEAT. Because NEAT evolves networks of different complexity (sizes) niching was found to be necessary for protection of new topology innovations. An important NEAT property is the *complexification* – it starts with simple networks and gradually adds new neurons and connections.

The niching algorithm used in NEAT is *Explicit Fitness Sharing* [7] (EFS) as it uses a genotypical distance measure. Fitness sharing reduces the fitness of an individual given the number of similar individuals (similar individuals form a niche) in the population. The version used in NEAT divides the population into mutually exclusive niche (species). When a new individual is to be assigned to a species it goes through a list of the already existing species and is always compared with the individual designated as the *species' representative*. The comparison is done via evaluation of a distance measure. If the two individuals are similar enough (their distance is below a predefined *speciation threshold* δ) the new individual is assigned to the species, otherwise, it tries the next one in the list. If no species is compatible enough, a new one is created and the individual becomes its *representative*.

[1] Note, the the source codes and detailed experimental settings can be found at: `http://neuron.felk.cvut.cz/~drchaj1/`.

GP. Genetic Programming [8] is a well-known evolutionary approach which evolves syntactic trees (or forests of trees). In this paper, we use a slightly simplified version of GP as a *base algorithm*, omitting a commonly used crossover operator as it was not found beneficial by Buk [5] for Hypercube-based domain. More specifically, to create a new generation, we employ a tournament selection (tournament of size 2) to select N (the population size) parents. Each selected parent then produces a single offspring. Algorithm continues by sorting all $2N$ individuals by their fitness and successively reduces their number back to N, keeping only the fittest.

The initial population is created using the *grow method* [8]. The depth of trees is limited to avoid bloating. We use two types of mutations: a *structural mutation* selects a random node in a tree and replaces it by a random subtree again using the *grow method* or replaces the node by a random node of the same arity. A *parametric mutation* selects a random constant node (if exists) and applies a Cauchy mutation [9] to it. A newly created random constant node is initialized by a random value from a selected interval.

3 Our Approach

In this section we describe GPAT algorithm. The algorithm works exactly as NEAT with an exception of genome representation, genetic operators and distance measure. Unlike in NEAT, where genomes encode neural networks, GPAT genomes represent trees (more specifically forests of trees, to facilitate multiple outputs). GPAT uses nodes of variable arity starting with zero children. Moreover, each children of a node is assigned a constant which has a similar function as a synaptic weight in ANN. The constants are therefore associated with links contrary to constants represented only by terminal nodes in GP.

For a detailed description of NEAT, especially speciation and selection, see [6]. In our implementation, we use an adaptive speciation threshold δ: it is doubled when a number of species exceeds the target number of species n_S, otherwise it is divided by 2.

GPAT Nodes. We use the following non-terminals for GPAT: + $\left(\sum_{i=1}^{C} c_i x_i\right)$, $\star$ $\left(\prod_{i=1}^{C} x_i\right)$, A $\left(\arctan\left(\sum_{i=1}^{C} x_i\right)\right)$, S $\left(\sin\left(\sum_{i=1}^{C} x_i\right)\right)$ and G $\left(e^{\sum_{i=1}^{C} x_i}\right)$ for all experiments in this paper. The output of each node is given in parentheses, C stands for the node arity, c_i is the i-th child's constant and x_i is the i-th child's output. For $C = 0$ the output is defined as 0 for all node types. Note, that constants are used only by the + node, they are stored but not used for other types of nodes. The given set is optimized for Hypercube-based indirect encoding, it contains symmetric, periodic and other functions. The set of terminals is composed of a problem-specific number of inputs and a fixed constant 1.

Similarly to NEAT, GPAT starts with a population of simplest forests possible. In our case this means a forests with all trees having a + terminal as the root with zero children (such trees always output zero).

Genetic Operators. A new individual is created by the following structural and parametric mutations which can be applied with a given probability for each tree of the forest:

Add Link Mutation. With a probability p_{AL} generate a random terminal t and connect it to a random tree non-terminal node. This mutation closely resembles *add link mutation* as found in NEAT, except, links always connect terminals to non-terminals.

Add Node Mutation. With a probability p_{AN} choose a random link l and replace it by the structure $l_1 \rightarrow n \rightarrow l_2$, where l_1 and l_2 are new links and n a new non-terminal random node. This mutation was again inspired by its NEAT counterpart.

Insert Root Mutation. Create a new random root node and assign the original root as its only child with a probability p_{IR}.

Switch Node Mutation. Chose a random non-terminal or terminal node and change its type randomly with a probability p_{SN}.

Cauchy Parametric Mutation. Apply a Cauchy mutation as described in GP section to each constant of the tree with a probability p_{CM}.

Replace Constant Parametric Mutation. With a probability p_{RC} for each constant of the tree reset its value to a random number from a given interval $[-a_R, a_R]$.

Currently we do not employ any crossover-like operator, which would require to implement *innovation number* mechanism (see [6]). To limit bloating no structural mutation was allowed for genotypes large enough (here the limit was: depth > 5, number of constants > 10 and the number of nodes > 12).

Distance Measure. Unlike in NEAT we have decided to use a simpler distance measure not based on innovation numbers. This is facilitated by the fact that we employ trees instead of networks[2].

Our approach called the *generalized* distance measure, covers many already published approaches for GP trees, it is mostly inspired by [10], although we do not take constant values into account. It allows to change behavior according to parameters. At first, all node types are partitioned to sets A_0, A_1, $A_2, \ldots$, where $A_0 = \{NIL\}$ contains only a special NIL node and A_1 is reserved for constants. Then we define an auxiliary function to compute the distance between two node types $x \in A_i$ and $y \in A_j$:

$$d'(x,y) = \begin{cases} 0 \text{ if } x = y \text{ (note, this implies } i = j\text{)}, \\ 1 \text{ otherwise.} \end{cases}$$

The *generalized* measure is defined as $d(p,q) = d'(p,q)$, if neither p, nor q have descendants, $d(p(s_1,\ldots,s_n),q(t_1,\ldots,t_m)) = d(q(t_1,\ldots,t_m),p(s_1,\ldots,s_n)))$, for $m < n$ and $d(p(s_1,\ldots,s_n),q(t_1,\ldots,t_m)) = d'(p,q) + \max(\alpha,\delta_{pq})\frac{1}{K}\left(\sum_{i=1}^{n} d(s_i,t_i) + \right.$ $\left. +\beta\sum_{j=n+1}^{m} d(NIL,t_j)\right)$ for $n \leq m$. The meaning of constants is as follows:

- $\alpha \in \{0,1\}$ decides whether to descend into subtrees, when nodes have different types (descends when $\alpha = 1$),

[2] We have also experimented with NEAT-like distance measure, the preliminary results show, that both approaches are comparable.

- $\beta \in \{0,1\}$ decides whether to include information concerning the missing subtree in one of the trees (includes when $\beta = 1$),
- δ_{pq} is the Kronecker delta,
- $K > 0$ controls the influence of a node depth in the tree (for $K = 1$ the node depth does not matter).

Along with the *generalized* distance measure we have also used the *random* distance as a control treatment, where the distance between two trees is defined as a random number from interval $[0,1)$ using uniform distribution. GPAT with the *random* measure will be denoted as GPAT-R. Note, that although this section dealt with tree distance measures, in fact we evolve forests with GPAT to allow multiple outputs. The distance between two forests is computed as an average of distances between corresponding trees.

4 Experimental Setup

In this section, we present both directly (Symbolic Regression and Maze Navigation) and indirectly encoded problems used to compare the *base algorithms* (NEAT, GP, GPAT and GPAT-R). Such experiments will be helpful to show, whether there is a difference in a choice of the right distance measure for direct and indirect encoding domains.

Symbolic Regression. The first, most straightforward problem is Symbolic Regression, the test functions are in Tab. 1. The equations are divided into four groups according to their dimensionality (the number of inputs). They were selected from a larger set, of which they presented the most diverse behaviors. Note, that the equation 4D-V was obtained by solving a Visual Discrimination problem (see below). The 1D, 2D and 3D functions were sampled in a hypercube having minimum and maximum coordinates -10 and 10, 4D used boundaries of -1 and 1. We have used 20 equidistant samples in each dimension for 1D, 7 for 2D and 3D and 5 for 4D. The fitness is computed using *Mean Squared Error* as $1/(1+MSE)$ which lies in $(0,1]$. The problem was considered to be solved, when target fitness 0.95 was reached. Note, neither testing nor validation sets were used.

Table 1. Symbolic Regression functions

ID	Function	ID	Function
1D-F	$1.5x_1^3 + 2.3x_1^2 - 1.1x_1 + 3.7$	3D-K	$x_1x_2^2x_3 - x_1x_2 + x_2x_3$
1D-H	$0.1x_1^2 + 0.2\sin(x)$	4D-C	$1.5x_1x_2x_3x_4$
2D-I	$1.5x_1x_2^2 + 2.3x_1x_2 - 1.1x_2^2$	4D-F	$1.5x_1x_2x_3 + 2.3x_1 - 1.1x_2 - 1.1x_4$
2D-K	$x_1x_2^2 + x_1x_2 - x_2^2$	4D-G	$1.5x_1x_3x_4x_2^2 - 1.1x_2 + 2.3x_1$
3D-E	$1.5x_1x_2 + 2.3x_1 + x_2x_3 - 1.1x_3$	4D-V	$e^{-ax_2^2}x_1\left(be^{-x_3^2} - c\sin(1-x_1)\right)$
3D-H	$1.5x_1x_2^2x_3 + 2.3x_1x_2x_3 - 1.1x_2 + 5.3$		$(a,b,c) = (1.36709, 2.43454, 0.393788)$

Maze Navigation. Maze navigation (see Fig. 1) presents a *deceptive task* to show abilities to overcome local extremes. The problem is a simple target approach, where a simulated robot tries to navigate through a maze from a starting position to a target one. Fitness is derived from the Manhattan distance between the robot and the target after 150 steps. We have used not only two different maps, but for each map also different types of sensors. For MAZE-1 the sensors were: distance to target, binary wall sensors (1 step ahead, ahead-left, ahead-right) and binary target detection (ahead, left, right and behind). For MAZE-2 we have used four wall range finders instead of three wall detection sensors. The controller has a single output: for output values less than -0.5 the robot turns left and advances, for values greater than 0.5 it turns right and advances, otherwise it makes only a step forward. The problem was considered to be solved if and only if the robot reaches the target. Both tasks are deceptive as there are local extremes in a proximity of the global one.

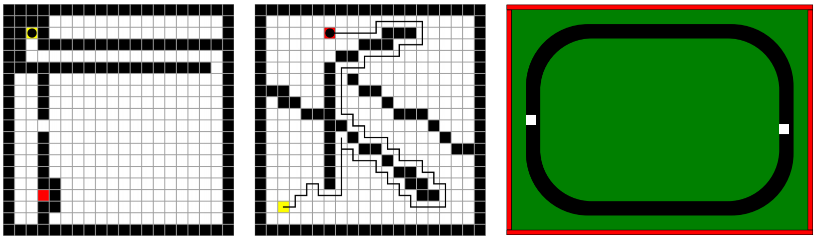

Fig. 1. MAZE-1 (left), MAZE-2 (middle) maps, Mobile Robot Navigation arena (right). MAZE-2 shows a possible solution. Mobile Robot Navigation map shows roads (fast) and grass (slow) surfaces. Two white squares determine starting positions.

Bit Permutation. Bit permutation is a set of three problems which are similar to the Bit Mirroring in [11]. All possible combinations of inputs (0s and 1s) were tested. In this paper we did experiments with $n = 6$ inputs/outputs (64 possibilities). The fitness is computed as a number of all correct output bits over all test patterns (in contrary to [11].) The problem was considered to be solved when all input patterns generated correct output patterns. The substrate is composed of two layers of neurons (input and output). Nodes in each layer are placed equidistant (with a distance 1). A bias neuron (having coordinate 0) is connected to all output neurons. The CPPN has therefore 2 outputs. The three problems are:

Bit Reverse. Bits $1, 2, \ldots n$ labeled from the LSB to the MSB: $o_j = i_{n-j+1}$, for $j \in \{1, 2, \ldots, n\}$.

Bit Shift. Logical shift to the right: $o_n = 0$, $o_j = i_{j+1}$, for $j \in \{1, \ldots, n-1\}$.

Bit Rotate. Circular shift to the right: $o_j = i_{j+1} \mod n$, for $j \in \{1, \ldots, n\}$.

Parity Problem. In this experiment we evolve a network computing odd parity. The function has *n* inputs (in our case 4) and a single output. The substrate is similar to the previous one, although we added a biased hidden layer (CPPN with 4 outputs). A *final network* must give correct answers for all 16 test cases.

Visual Discrimination. As presented in [4], the task is to detect the larger object of two objects, projected on a two-dimensional array of input neurons. Unlike in original experiment, we employed a smaller input resolution of 5×5 and the problem was considered to be solved only for success in all test cases.

Mobile Robot Navigation. In the last experiment, we have employed ViVAE (Visual Vector Agent Environment) [5,12]. The task was to evolve robotic controller to drive two robots on a map (See Figure 4) at a maximum possible speed. The best strategy is to drive on a side of a road to avoid collisions. The problem was considered to be solved when a fitness reached 0.797 (value found experimentally).

Parameter Settings. The size of population was set to 100, the maximum number of generations was 1000 (for Mobile Robot Navigation they were set to 100 and 50) for all algorithms (GP, GPEFS and NEAT). All experiments were repeated for 200 times with an exception of the Mobile Robot Navigation, which was repeated for 20 times, only.

We have used the same node types for all experiments. In the case of NEAT they were: Bipolar Sigmoid, Sin, Linear and Gaussian. For GP: Add, Multiply, ArcTan, Sin and Gaussian, see [5]. GPAT nodes were described above. Along with inputs, GP uses a fixed constant of -1. Note, that other parameter settings, e.g., mutation rates are available at the paper support page (see above).

To test the properties of the *generalized* distance measure, we did experiments for all 8 combinations of $K \in \{1,2\}$, $\alpha \in \{0,1\}$ and $\beta \in \{0,1\}$, we will write these configurations as tuples, e.g., $(2,0,1)$, in the following text.

5 Results

The results of all experiments are summarized in Fig. 2 showing success rates for both direct and indirect problems. One can see that NEAT performs the worst of all algorithms with an exception of Visual Discrimination where it is comparable with GPAT-R and MAZE-2 where it significantly[3] outperforms GP. Note, that NEAT never outperforms GP on indirect encoding problems: this supports the results in [5].

GP is a significant winner on three symbolic regression problems (2D-K, 3D-K and 4D-V) and a nonsignificant winner on Parity. Interestingly 2D-K and 3D-K are equations with a minimum number of constants (zero and one), while 4D-V is a solution of the Visual Discrimination evolved by GP (GP is most probably biased to evolve such solutions).

[3] We use Fisher's Exact test (two-tailed, significance level 5%) to compare the numbers of successful experiments.

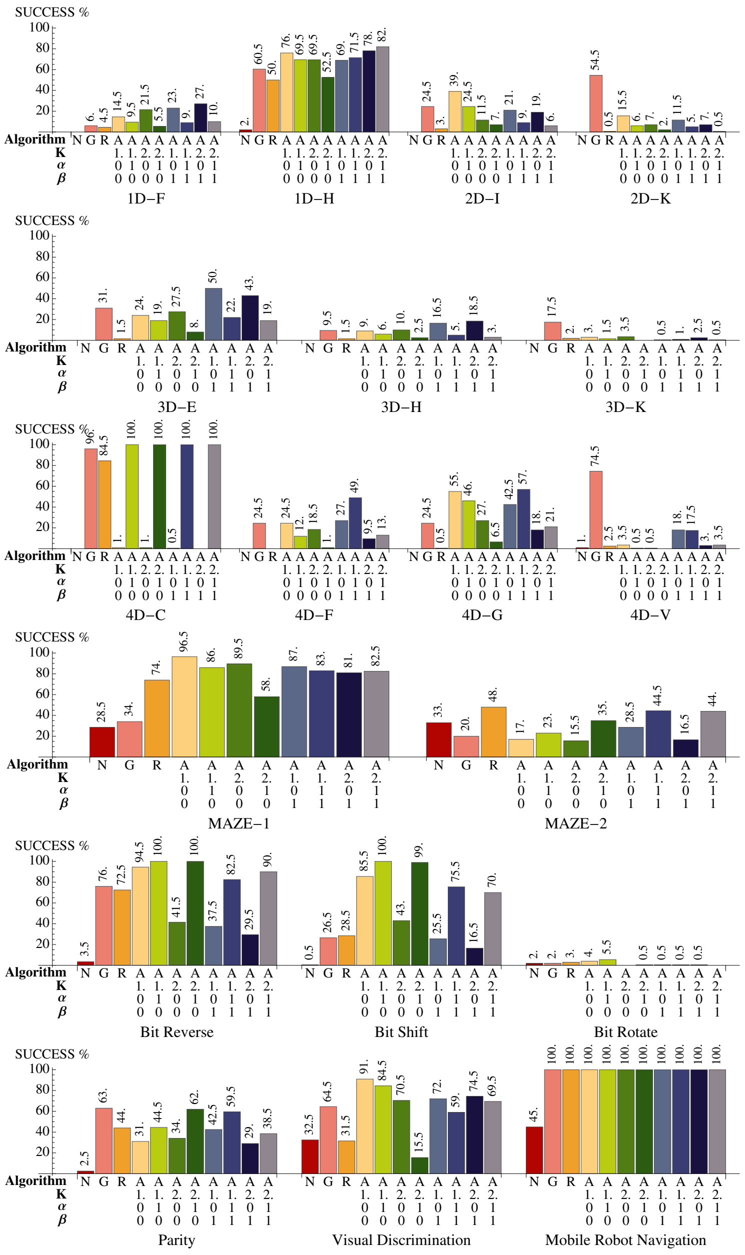

Fig. 2. Experiment results for all test problems. The success rate over 200 experiments (20 for Mobile Robot Navigation) is shown above bars. NEAT is labeled as N, GP as G, GPAT as A and GPAT-R as R. The values of distance measure constants K, α and β are shown.

GPAT-R, the control treatment, performed significantly better than NEAT in most cases. On the other hand it performed worse than GP on all with an exception of both maze tasks (significantly better) and Bit Shift (non-significantly better).

Finally, with a proper choice of the parameters K, α and β, GPAT is a significant winner on all problems with an exception of the before mentioned 2D-K, 3D-K, and 4D-V dominated by GP and MAZE-2 and Parity where GPAT is non-significantly worse than GPAT-R and GP. Even without selecting the best configuration for each problem and having them fixed either to $(1,0,0)$ or $(1,1,1)$, GPAT is a winner (at least non significantly worse than the best of NEAT, GP or GPAT-R) for 8 out of 13 direct problems and 5 out of 6 indirect problems.

As one can see, GPAT success rate shows a noticeable dependence on distance measure parameters. The cases where GPAT-R outperforms GPAT clearly indicates where inappropriate parameter settings were chosen. The choice of $K = 2$ (for $K > 1$ the deeper the node is in a tree the smaller influence it has on a distance) seems to rather harm the performance when compared to $K = 1$ (with few exceptions, e.g., Parity) which contradicts results in [13]. The choice of α (enables/disables a comparison of subtrees with different root node types) and β (take or not the size and shape of subtrees missing in one of the trees into account) parameters is highly problem-dependent. However, for all indirect encoding problems with an exception of Visual Discrimination, setting $\alpha = 1$ improves the performance. For 4D-C (direct encoding) the difference even makes up to 100%. Setting $\beta = 1$ most often leads to a reduced performance. On the other hand, when $\alpha = 1$, setting $\beta = 1$ led to improvement (see Visual Discrimination, Parity, 4D-F, 4D-G, 4D-V and maze tasks). This influence of α and β should be further examined.

6 Conclusions

In this paper, we have proposed a novel algorithm called GPAT which is inspired by the well-known NEAT and GP algorithms. It takes the complexification property and the niching evolutionary algorithm from NEAT while operating on tree-structures known from GP. The use of trees allowed for a simpler distance measure and one such the *generalized* measure was proposed and tested. Although the choice of node types for GPAT was optimized for Hypercube-based indirect encoding tasks, the algorithm is general and can be simply adapted to any task where GP is applicable.

GPAT outperformed both NEAT and GP on most benchmarks. For all indirect encoding problems GP was superior to NEAT which supports results in [5]. Our estimation is, that this is caused by a larger search-space for NEAT as it has to optimize more random constants than a comparable tree. Also, some of the tested problems demanded high accuracy (e.g., Symbolic Regression) which might be disadvantageous for NEAT. This has to be, however, further examined.

In the future we plan to experiment with *innovation number* based distance measure resembling the one used in NEAT and compare it to the proposed *generalized* measure. The use of innovation numbers will also allow to implement crossover (mating) operator efficiently. The *generalized* measure can be further extended to take values of constants into account similalrly as in [10]. Moreover, there is a possibility to employ other measures, i.e., edit-distance based measures [14].

Acknowledgement. This research has been supported by the Ministry of Education,Youth and Sports of the Czech Republic as a part of the institutional development support at the CTU in Prague.

References

1. Eggenberger-Hotz, P.: Creation of Neural Networks Based on Developmental and Evolutionary Principles. In: Gerstner, W., Hasler, M., Germond, A., Nicoud, J.-D. (eds.) ICANN 1997. LNCS, vol. 1327, pp. 337–342. Springer, Heidelberg (1997)
2. Gruau, F.: Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm. PhD thesis, Ecole Normale Supirieure de Lyon, France (1994)
3. Koutnik, J., Gomez, F., Schmidhuber, J.: Evolving Neural Networks in Compressed Weight Space. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation - GECCO 2010, p. 619. ACM Press, New York (2010)
4. Gauci, J., Stanley, K.O.: Generating Large-Scale Neural Networks Through Discovering Geometric Regularities. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation - GECCO 2007, pp. 997–1004. ACM Press, New York (2007)
5. 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)
6. Stanley, K.O.: Efficient Evolution of Neural Networks through Complexification. PhD thesis, The University of Texas at Austin (2004)
7. Mahfoud, S.W.: A Comparison of Parallel and Sequential Niching Methods. In: Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 136–143. Morgan Kaufmann (1995)
8. Poli, R., Langdon, W.B., Mcphee, N.F.: A Field Guide to Genetic Programming (March 2008), Published via `http://lulu.com`
9. Yao, X., Yong, L., Guangming, L.: Evolutionary Programming Made Faster. IEEE Transactions on Evolutionary Computation 3, 82–102 (1999)
10. Ekárt, A., Németh, S.Z.: A Metric for Genetic Programs and Fitness Sharing. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 259–270. Springer, Heidelberg (2000)
11. Clune, J., Stanley, K.O., Pennock, R.T., Ofria, C.: On the Performance of Indirect Encoding Across the Continuum of Regularity. IEEE Transaction on Evolutionary Computation 15(3), 346–367 (2011)
12. Drchal, J., Koutnik, J., Snorek, M.: HyperNEAT Controlled Robots Learn How to Drive on Roads in Simulated Environment. In: CEC 2009 Proceedings of the Eleventh Conference on Congress on Evolutionary Computation, Trondheim, pp. 1087–1092. IEEE Press (2009)
13. Igel, C., Chellapilla, K.: Investigating the Influence of Depth and Degree of Genotypic Change on Fitness in Genetic Programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, FL, USA, pp. 1061–1068. Morgan Kaufmann (1999)
14. Nguyen, T.H., Nguyen, X.H.: A Brief Overview of Population Diversity Measures in Genetic Programming. In: Proceedings of the Third Asian Pacific Workshop on Genetic Programming, pp. 128–139 (2006)