Empirical study on the performance of Neuro Evolution of Augmenting Topologies (NEAT)

Domen Vake
domen.vake@innroenew.eu
Innorenew CoE
Izola, Slovenia

Aleksandar Tošić
aleksandar.tosic@upr.si
University of Primorska, UP FAMNIT
Koper, Slovenia
Innorenew CoE
Izola, Slovenia

Jernej Vičič
jernej.vicic@upr.si
University of Primorska, UP FAMNIT
Koper, Slovenia
ZRC-SAZU
Ljubljana, Slovenia

ABSTRACT

In this paper we provide empirical results on training a neural network with a genetic algorithm. We test various features of the generalized genetic algorithms, namely spieciation and fitness sharing and present the statistical analysis of all three variations. An obstacle avoidance problem was created in which the objective is for vehicles to traverse the course. We present interesting observations about the differences between evolutionary techniques and argue that there is a significant benefit in approaches that aim to diversify the gene pool as a mechanism for avoiding local minima.

I.2.1 ARTIFICIAL INTELLIGENCE Applications and Expert Systems

1 INTRODUCTION

Genetic algorithms (GA) have been used extensively for various optimization problems. Arguably, their wide usage spectrum can be accredited to their simplicity and the fact no assumptions are made about the problem. Consequently, most variations of genetic algorithms have strived to maintain these properties. Many techniques were proposed in an effort to diversify the gene pool and at the same time avoid getting stuck in local minima.

In some cases, finding multiple sub-optimal solutions is beneficial [4]. With the recent development of neural networks, genetic algorithms have regained a lot of attention as an viable learning technique. There are many variations to how typical GA functions such as gene encoding, crossover, and mutation are implemented when applied to neural networks. A very promising family of algorithms are descendents of the general NEAT algorithm.

Authors proposed some extensions of the original NEAT algorithm [10] such as rtNEAT [8] that allows evolution to occur in real time rather than through the iteration of generations as used by most genetic algorithms. The basic idea is to put the population under constant evaluation with a "lifetime" timer on each individual in the population. Phased pruning implemented in SharpNEAT framework [2] adds periodic pruning of the network topologies of candidate solutions during the evolution process. HyperNEAT [9] is specialized to evolve large scale structures. HyperNEAT has recently been extended to also evolve plastic Artificial Neural Networks and to evolve the location of every neuron in the network separately.

The first video game to implement Content-Generating NEAT (cgNEAT) [3] that evolves custom video game content based on user preferences is Galactic Arms Race, a space-shooter game in which unique particle system weapons are evolved based on player usage statistics. Neuro-Evolving Robotic Operatives (NERO) [5] is a video game that applies NEAT to train robots that compete among themselves. odNEAT [7] is an online and decentralized version of NEAT designed for multi-robot systems, it is executed onboard robots themselves during task execution to continuously optimize the parameters and the topology of the artificial neural network-based controllers.

2 IMPLEMENTATION

In this section we show specifics to our implementation. The optimization problem is agent based, two-dimensional driving simulation where the objective is to have agents traverse the obstacle course. The agent is considered evolved if it has made a full loop around the track without hitting the wall. It's fitness is based on the distance traveled within a fixed amount of time and is weighted by the amount of checkpoints reached. This ensures the agents move through course and avoid driving in circles.

Agents

Every tick of simulation the agent's task is to make a decision on the move it wants to perform within the environment. The decision pool consists of five possible moves, which are *do nothing*, *drive forward*, *drive backwards/brake*, *turn left* and *turn right*. The turning is dependant on the speed of the agent, so if the agent is standing still, turning has no effect on it. The decision is chosen with the use of the artificial neural network that is represented as agent's genome and it is based on the agents relative location within the track.

The agents are aware of their surroundings with the help of sight lines, which are represented as lines that fan out from the agent's location as shown in the figure 1. Each line calculates a possible intersection with a wall and tells the agent the distance to the closest wall in the line's direction, if one exists. With agent's speed values are then passed to the genome for a prediction.

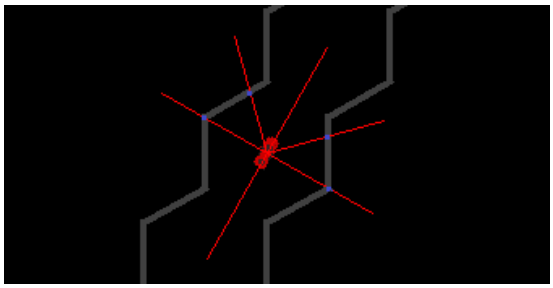


Figure 1: Figure shows the agent and his sight lines (red lines) and the detection of the wall on the track (blue points).

Genome encoding

Artificial neural network in the genome is represented as the list of nodes and a list of connections of the network as described in [10]. Upon their creation, all new genes are given an innovation number to ensure the differentiation between the genes. If the gene already exists somewhere in the population (i.e. connection that connects nodes x an y), it is given the same number as the original gene, otherwise a new incrementally higher innovation number is given to the gene. To ensure, that the same gene does not get more than one innovation number, genes must never be deleted, so the number doesn't get lost. For that purpose every connection has a value that represents whether the gene is active and should be represented in the phenotype of the agent. The starting artificial neural network of the agents is a fully connected network with the input layer of size 7 (speed and all sight lines) and output layer of size 5 (all the possible actions of the agents). Activation function of the nodes in the output layer is the *rectified linear unit* and all the other nodes use the *sigmoid* function.

Crossover

Innovation numbers provide a way to crossover two genomes by matching the genes of the two parents. The genes are split into groups of matching genes (genes that are contained in genomes of both parents), disjoint genes (genes that are contained in only one parent in the middle) and excess genes (genes that are contained in only one parent at the end). When creating an offspring genome, the matching genes are inherited from a random parent, whereas the disjoint and excess genes are inherited from the more fit parent[10].

Mutation

With crossover the population is likely to discard genes that don't provide good agent behaviour, but that can lead to gene deprivation. We introduce genetic innovation to the system by mutating the genome, where some mutations affect the topology and some the optimization of the network [11].

Edge mutation. When optimizing the current topology of the network the existing connections are being mutated. The weight of the connection is either being multiplied by a number between 0 and 2, to provide weight optimization or it is multiplied by -1, to change the polarity of the connection. In case of topological mutation we add a new connection between two unconnected nodes in a way, that doesn't create a cycle in a digraph representation of the network and we give it a random weight or we deactivate a connection. The mutation rates we used were 0.25 for *new connection/deactivate connection*, 0.8 for *adjust weight* mutation and 0.2 for *flip weight mutation.*

Vertex mutation. We can also mutate the genome of the network by adding new vertices to increase its complexity and add new options to the pool of solutions. A new vertex is added by choosing a random edge e between vertices A and B and deactivating it. A new vertex C is inserted, and two new edges created connecting vertex A and B with C. The weight of the edge leading to the new vertex C, is set to 1 and the edge leading to the vertex B is set to same wight of the disabled edge e to minimize the performance impact of the new genes on the genome. The mutation rates for the *new node* mutation that we used were 0.01.

Species

When adding innovation to the genome, it is likely that the mutation will first reduce the fitness of the agent and will be removed before it has the chance to evolve and optimize. To counter that and protect innovation, we introduce the notion of speciation, where the agents are split into groups that represent species, based on the genotypes.

Genetic Distance. To find the genetic difference between two genomes the idea of compatibility distance (δ) is introduced. The less genetic history two genomes have, the more disjoint (D) and excess genes (E) and the less matching genes they have[10]. There numbers can be normalized with the total number of genes in the larger genome (N) and use them to calculate the compatibility distance. We also take into account the average weight differences of matching genes ($\overline{W}$). The coefficients w_1, w_2 and w_3 represent the importance of each of the factors and can be adjusted.

$$\delta = \frac{w_1 E}{N} + \frac{w_2 D}{N} + w_3 \overline{W}$$

Every generation each agent is placed into a species, if he's compatibility distance to the species representative is smaller than the prefixed threshold (δ_t). If the agent does not fit into any of the species, he now represents a new species. The weights that were used in our case are $w_1 = 1.3$, $w_2 = 1.3$, $w_3 = 1.0$ and $\delta_t = 2.0$.

Selection. In the selection step, we remove the worse half of the agents from the population based on their fitness score. That would in general remove most of the innovation within the population, since the mutated agents tend to perform worse. So instead of removing bottom half in general, we do it per species. That means that every agent only competes with agents that are a part of the same species. This provides an extra layer of protection of the new genes that have not yet had the time to adjust and optimize.

Explicit Fitness Sharing. We used the *explicit fitness sharing*[1] niching technique, which normalizes the fitness (f) of the agent according to the size of the species that he's in. With niching it is unlikely that one species would take over the whole population therefore it widens the search in the solution space.

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))}$$

If the distance between the agents i and j ($\delta(i,j)$) is smaller than the threshold, the value of the sharing function *sh* is set to 1 otherwise its set to 0[6].

3 RESULTS AND CONCLUSIONS

Four tracks were prepared as shown on figure 4. We tested how the model performs if we remove the different features of the algorithm that provide the protection of the innovation within the population. Explicit fitness sharing and speciation were chosen. From that, we formed four tests: *normal*, that includes all sections of the algorithm, *no efs*, that have explicit fitness sharing disabled, *no speciation* with disabled speciation (only one species during simulation) and *no efs and speciation* that does neither include the explicit fitness sharing nor speciation. The size of the population was set to 1000 and all simulations ran for 250 generations wheres every generation was 750 ticks long. Each test was ran ten times on each of the tracks and for every generation max fitness, mean fitness, standard deviation and whether the model has found the sufficient solution were collected. Also every fifth generation we collected the data of all species, their size, max fitness, mean fitness and standard deviation.

Table 1 shows the aggregated results for individual tracks across multiple runs. Track 1 has seen the best results and least complexity in the neural network as agents only need to turn one direction to successfully traverse the track. The second best results were obtained by track 3, in which 75% of the simulations evolved and completed the track. Tracks 2 and 4 were arguably the hardest with 60% and 40% respectively. When considering only the evolution rate, all the test showed similar results, since all tests had 67.5-70.0% evolution rate. The test, where explicit fitness sharing was disabled found the best solution in two of the four tracks. This shows, that the weights for the explicit fitness sharing might not have been optimal, for the problem at hand. In the column $\mu(x)$ shows the mean fitness through all the generations and simulations of the test. The data shows, that the test where speciation and explicit fitness sharing were both disabled in general performed better. However the optimal solution was not found in any of the tracks. This could be a consequence of the model finding and optimizing to a local minimum fast, but due to the lack of innovation, being unable to escape.

Table 1: Table shows the data gathered from tests. Every row represents summarized data from 10 iterations of the test on specific track Fitness is represented as x and is normalized by the highest fitness achieved on that track. Tests: N-*normal*, E-*no efs*, S-*no speciation*, SE-*no efs and speciation*

	Track	Test	$max(x)$	$\mu(x)$	$\sigma(x)$	Evolved[%]
1	Track 1	N	0,99	0,12	0,58	100
2	Track 1	E	0,98	0,18	0,45	100
3	Track 1	S	1	0,13	0,57	100
4	Track 1	SE	0,98	0,21	0,46	100
5	Track 2	N	0,96	0,03	0,15	50
6	Track 2	E	1	0,05	0,20	50
7	Track 2	S	0,99	0,04	0,24	80
8	Track 2	SE	0,88	0,05	0,19	60
9	Track 3	N	0,98	0,05	0,24	80
10	Track 3	E	1	0,12	0,37	80
11	Track 3	S	0,93	0,06	0,26	80
12	Track 3	SE	0,83	0,09	0,28	60
13	Track 4	N	1	0,01	0,16	40
14	Track 4	E	0,61	0,03	0,14	50
15	Track 4	S	0,64	0,01	0,05	20
16	Track 4	SE	0,84	0,06	0,24	50

Figures 2, and 3 illustrate the impact of fitness sharing on the evolution of existing species and the emergence of new ones. We observe that the number of different species that emerged is significantly higher when fitness sharing is enabled. This is expected as the mechanism allows new species to be preserved across generations in order to diversify the gene pool. However, we also observe that a significant number of species that emerged survived across all generations and achieved significant improvements to their fitness (representative). Additionally, we can observe that in both cases,

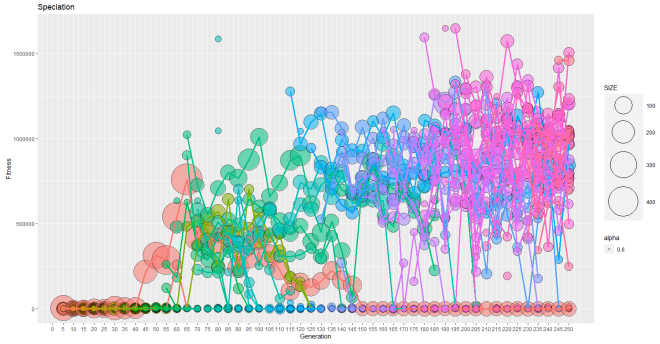


Figure 2: Species with explicit fitness sharing enabled

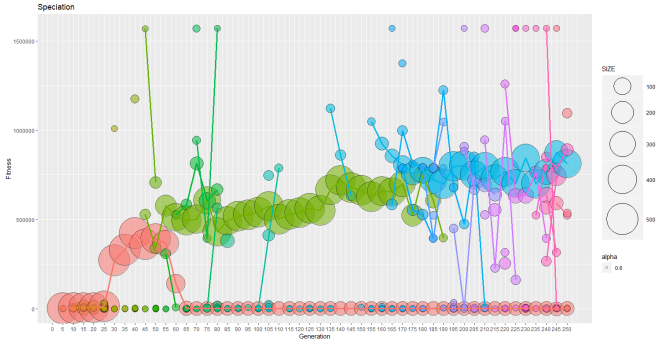
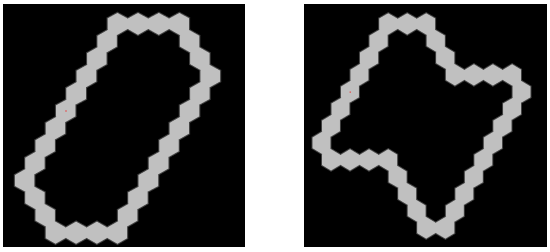


Figure 3: Species with explicit fitness sharing disabled



(a) Track 1 **(b) Track 2**

(c) Track 3 **(d) Track 4**

Figure 4: Tracks used for testing the model's performance

some newly emerged species never evolve and improve their fitness which eventually causes them to be removed. This indicates that even with explicit fitness sharing, species with bad genes do not impact the overall results even though their genes are initially protected.

With no fitness sharing, there is a trend of one big species and a few smaller ones, that explore the solution space and when a new innovation is found with better results, it takes over the population. Contrary to that, evolution of the population, that shares fitness splits into more species and is searches the space for a wider set of solutions.

All the presented software is available under opensource licence at Github[1].

[1]NEAT-driving: https://github.com/VakeDomen/NEAT-driving

4 ACKNOWLEDGMENTS

The authors gratefully acknowledge the European Commission for funding the InnoRenew CoE (Grant Agreement #739574) under the H2020 Widespread Teaming programme and investment funding from the Republic of Slovenia and the European Regional Development Fund.

REFERENCES

[1] David E Goldberg, Jon Richardson, et al. 1987. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*. Hillsdale, NJ: Lawrence Erlbaum, 41–49.

[2] Colin Green. 2004. Phased searching with NEAT: alternating between complexification and simplification. *Unpublished manuscript* (2004).

[3] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley. 2009. Evolving content in the galactic arms race video game. In *IEEE CIG*. 241–248.

[4] Jian-Ping Li, Marton E Balazs, Geoffrey T Parks, and P John Clarkson. 2002. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary computation* 10, 3 (2002), 207–234.

[5] Risto Miikkulainen, Bobby D Bryant, Ryan Cornelius, Igor V Karpov, Kenneth O Stanley, and Chern Han Yong. 2006. Computational intelligence in games. *Computational Intelligence: Principles and Practice* (2006), 155–191.

[6] Bruno Sareni and Laurent Krahenbuhl. 1998. Fitness sharing and niching methods revisited. *IEEE transactions on Evolutionary Computation* 2, 3 (1998), 97–106.

[7] Fernando Silva, Paulo Urbano, Luís Correia, and Anders Lyhne Christensen. 2015. odNEAT: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation* 23, 3 (2015), 421–449.

[8] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. 2003. Evolving adaptive neural networks with and without adaptive synapses. In *CEC'03.*, Vol. 4. IEEE, 2557–2564.

[9] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212.

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

[11] Pushpendra Kumar Yadav and NL Prajapati. 2012. An overview of genetic algorithm and modeling. *IJSRP* 2, 9 (2012), 1–4.