Task 1: Specialist Agent

Evolutionary Computing - Group 84 - 01/10/2021

Jaimie Rutgers
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2598649

Martijn Wesselius
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2599666

Ekaterina Geraskina
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2636212

Dominic Istha
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
2600140

1 INTRODUCTION

Reinforcement Learning algorithms have had several successes in (video) games in the past decades [1, 15, 17, 18, 21]. However, video games have large state and action spaces and require diverse behaviors, consistent individual behaviors, fast adaptation, and memory of past states [19]. These are difficult issues for standard Reinforcement Learning algorithms, but can be solved within a Neuroevolutionary approach [13]. Neuroevolution is an approach to optimize the weights and/or topology of an Artificial Neural Network (ANN), using an Evolutionary Algorithm, whenever it is not possible to apply standard optimization techniques [16].

Evolutionary Algorithms can broadly be categorized into fixed-topology methods and topology and weight evolving methods. Evolutionary algorithms with a fixed topology can be effectively used to optimize the connection weights of neural networks. They do however require human design to determine the optimal structure and number of hidden nodes in a neural network. In recent years, researchers have obtained promising results with evolutionary algorithms that are simultaneously able to evolve the structure, nodes and weights of neural networks, so-called Topology and Weight Evolving Artificial Neural Networks (TWEANNs) [20].

The goal of our research is to determine and analyse the difference in performance between a fixed-topology method and a TWEANN. The fixed-topology method we will implement is a Particle Swarm Optimization (PSO) algorithm, which has been shown to be able to effectively optimize weights in neural networks [2, 8–10]. One of the first TWEANNs shown to be able to outperform fixed-topology methods is the NeuroEvolution of Augmenting Topologies (NEAT) method, proposed by Stanley and Miikkulainen [20]. The performance will be tested on the EvoMan framework which is a framework for testing optimization algorithms with artificial agent controllers in electronic games. This framework was introduced by de Araújo and de França [5] and inspired by the action-platformer game Mega Man II.

We begin in Section 2 by describing the PSO and NEAT methods and their underlying principles. Next, we will present the experimental setup and motivate the parameter choices. In Section 3, we will analyse and compare the results of PSO and NEAT and compare them to the results of de Araujo and de Franca [6] on the EvoMan framework. In Section 4, we will discuss the experiment and results and present our concluding remarks. Section 5 briefly states the contributions of the authors to this report.

2 METHODS

2.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a gradient-free optimization algorithm proposed by Eberhart and Kennedy [7] in 1995. The algorithm is inspired by the behaviour of bird flocks searching for food in order to survive. Gradient-free optimization algorithms, like PSO, do not require derivatives which makes them very flexible in their use-cases.

The PSO algorithm starts by creating an initial population or swarm of s particles each with a position x_i in an n-dimensional solution space and a combined direction and velocity v_i. For this problem setting, each particle is thus a set of n weights corresponding to the connections in the ANN. The performance of this ANN in controlling the Evoman player is then evaluated by the fitness function discussed in section 2.3. At every iteration, the movement of each particle is determined by its current velocity v_i, the velocity of its best prior position p_i and the velocity of the best position of any particle in the past p_g. This movement is added to the current position of the particle to update its location in the solution space. Consequently, more and more particles will eventually move to areas where better solutions are found until the local or global optimal solution is discovered. The iteration steps for a particle's velocity and position are as follows:

$$\mathbf{v_i}(t+1) = \omega \mathbf{v_i}(t) + c_1 r_1 (\mathbf{p}_i(t) - \mathbf{x}_i(t)) + c_2 r_2 (\mathbf{p}_g(t) - \mathbf{x}_i(t)) \quad (1)$$

$$\mathbf{x_i}(t+1) = \mathbf{x_i}(t) + \mathbf{v}_i(t+1) \quad (2)$$

In Formula 1, ω is the inertia weight, which determines how strongly the current velocity of the particle is taken into account and controls the degree of exploration of the search [2]. The variables c_1 and c_2 are acceleration coefficients, which control the influence of the personal best and global best particle positions, respectively. This regulates how far a particle can move in a single iteration [2]. Variables r_1 and r_2 are random numbers between (0,1) which bring some stochasticity into the equation. This helps the algorithm in escaping possible local optima.

2.2 NeuroEvolution of Augementing Topologies (NEAT)

PSO is a fixed-topology method, as the number of hidden neurons in the network are predefined and only the weights evolve. In contrast, researchers have proposed neuroevolution methods that simultaneously evolve the topology and the connection weights of a network: Topology and Weight Evolving Artificial Neural Networks (TWEANNs). One of the first TWEANNs shown to outperform fixed-topology methods is the NeuroEvolution of Augmenting Topologies (NEAT) method [20]. NEAT simultaneously optimizes and complexifies solutions, resulting in a robust and fast optimization method. Below, we will briefly describe the NEAT method and highlight its strengths based on its four key components.

Mutation. NEAT encodes neural networks as genomes consisting of connection genes and node genes. A connection gene connects two node genes and denotes the weight of the connection and whether the gene is expressed or not. Genetic mutation can perturb the connection weights and add or remove nodes, changing the structure of the network. Through genetic mutation, solutions with both varying weights and different topologies are explored.

Crossover. Performing meaningful crossovers between neural networks of different sizes and with different topologies is challenging, but NEAT efficiently overcomes this problem. Each connection gene also denotes a global innovation number, which represents the chronology of every gene in the system. Two genes with the same innovation number represent the same structure, since they have the same historical origin. During crossover, the parent genes with the same innovation number are lined up and the genes that do not match are inherited from the more fit parent. This proves to be a simple and efficient method to perform meaningful crossover for topology evolving neural networks.

Speciation. NEAT speciates the population, so topological innovations get the time to optimize before having to compete with the population at large. Each topological innovation leads to a promising part of the search space, but may initially suffer from a reduced fitness. To prevent such solutions from being removed from the population prematurely, individuals only compete with topologically similar individuals first. Speciation protects innovation and allows the population to explore the search space in many directions simultaneously.

Minimizing Dimensionality. NEAT starts with a uniform population, instead of a random initial population like most TWEANNs. Since innovation is protected through speciation, NEAT can start with a uniform population and only introduce structural changes when necessary. This allows NEAT to limit the number of searches through weight dimensions, significantly increasing the efficiency.

2.3 Experimental setup

The algorithms are executed for three different enemies, which will be referred to as an experiment. The research was done in limited time (a couple of weeks) and with limited resources. Therefore, we use the enemies that are evaluated quickest by the framework, which turned out to be enemies 1, 4 and 6. Both algorithms are executed for 100 generations, against a single enemy. This process is repeated 10 times for both PSO and NEAT, resulting in 10 runs. For each generation and each of the 10 runs, we store the mean and maximum fitness of the population to evaluate the algorithms. The individual fitness is expressed by the following equation:

$$\text{fitness} = \gamma * (100 - e_e) + \alpha * e_p - \log t \tag{3}$$

where e_e and e_p are the energy measures of the enemy and player respectively, with $e_e, e_p \in [0, 100]$, t the number of time steps the game took and γ and α are constants with values 0.9 and 0.1, respectively [6]. Finally, we run the best solution per run another 5 times to account for stochasticity. These results are evaluated using the individual gain, which is expressed by $g_i = e_p - e_e$. A t-test is then applied to test for a significant difference in mean individual gains between the two algorithms.

The PSO algorithm was implemented in Python with the PySwarms library [14]. The particle number s was set to 20, because according to Dai et al. [4] the convergence rate and search performance of PSO does not increase significantly if one goes beyond that number. Choosing a lower number of particles will increase iteration speed, but greatly diminishes the search capability of the algorithm. The number of dimensions n equals the number of weights in the given ANN, which on itself is determined by the number of hidden nodes. In this instance, n equals 265, since we use 10 hidden nodes following [6]. For the inertia parameter ω we opted for a starting value of 0.73 as proposed by Dai et al. [4], which decreases according to an exponential decay function [11]. The value of c_1 was set to 2.5 and non-linearly decreases during a run [3]. The second acceleration coefficient c_2 was set to 0.5 which increases linearly by 20% during a single run [22]. This to ensure a greater exploration in the first generations and a greater exploitation in later generations. Lastly, a boundary strategy to normalize inputs was implemented on the weights discovered by the PSO algorithm . Weights are cut off to the boundaries when they fall outside the interval (-1,1).

The NEAT algorithm for the experiments conducted here, is implemented using the NEAT-Python library [12]. The individuals in the population are all feed-forward neural networks in our experiments. The configurations of NEAT are the same for the three experiments against different enemies. Below is a brief explanation the parameter choices for the configuration of NEAT.

The number of hidden nodes of the initial population of NEAT is set to be 10, because de Araujo and de Franca [6] found a neural network with 10 hidden nodes to perform reasonably well on the task at hand. The activation function utilized is a linear activation function, which is clamped at -1 and 1. This allows for an efficient translation to an action either being taken or not, based on whether the output value of the output node is closer to -1 or 1. To continue, any specie that hasn't improved for five generations is removed from the population, since early experiments showed that with a population of size 10, most species contained bad solutions which barely improved. Removing these unpromising species quickly from the population improves efficiency. To prevent good solutions at the boundary of what is achievable to be removed, the best two individuals in the best two species are always preserved. During initial experiments the mutation rates were altered to observe its effect. Eventually, all mutation rates and mutation powers were set to 0.2 as the initial experiments showed this to be a good balance between sufficient mutations without introducing to much randomness. Only the mutation rate of the weights was set higher, at a value of 0.8, as it seems worthwhile to explore a greater variety of weights. All other parameter choices were taken from the recommended configurations of the NEAT-Python example for the task of single pole balancing [12].

3 RESULTS

Figures 1, 2 and 3 show the average and standard deviation over the 10 runs of the mean and maximum fitness, per generation. In all three figures, we observe that the NEAT algorithm outperforms the PSO, both in terms of the mean and maximum fitness. Furthermore, in Figures 1 and 2 we observe after around 50 iterations zero to no variance in the maximum for the NEAT algorithm, implying the algorithm has converged to the same solution in all 10 runs. Regarding PSO, we observe a greater variance in maximum values compared to NEAT, especially in Figures 1 and 2. This would imply that the algorithm is able to find good weights for the ANN, resulting in a higher fitness, but that the algorithm is not very stable as the maximum fitness can vary from around 50 to around 95 (Figure 2). This can be potentially validated by the fact that the mean fitness for PSO is both low and stable, meaning that overall PSO performs poorly.

Comparisons to de Araujo and de Franca [6] are difficult to make, as they only report the maximum fitness values (90, 94 and 73 for enemies 1, 4 and 6, respectively) and do not report mean fitness values nor any results across generations. When comparing to the maximum fitness values of de Araujo and de Franca [6], our version of NEAT achieves a comparable fitness on enemies 1 and 4 (around 90) and even seems to outperform de Araujo and de Franca [6] on enemy 6, with an average maximum fitness well above 75.

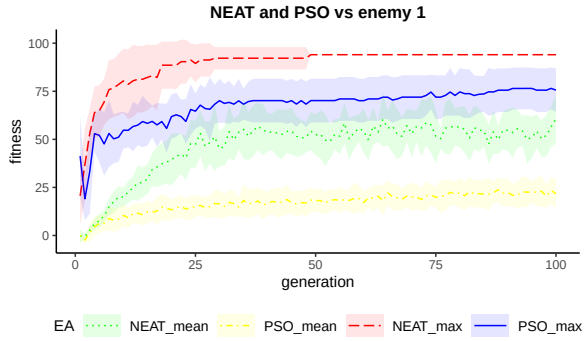

Figure 1: Performance of NEAT and PSO vs. enemy 1

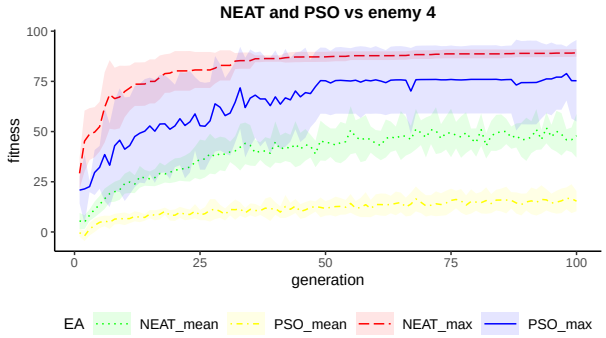

Figure 2: Performance of NEAT and PSO vs. enemy 4

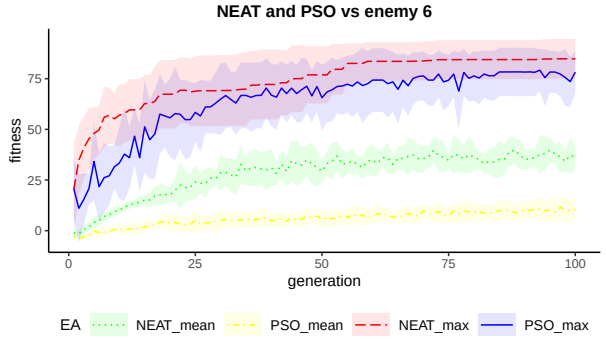

Figure 3: Performance of NEAT and PSO vs. enemy 6

Finally, Figure 4 shows boxplots of the individual gains for the best solutions of each run. As mentioned in Section 2.3, t-tests will be carried out test the difference between the mean individual Ggain values for NEAT and PSO.

For enemy 1, the individual gain of the NEAT player is always the same. This is also reflected in the zero variance of the maximum fitness depicted in Figure 1. For PSO, the player loses from enemy 1 for all runs except one, but in that case obtains the same individual gain as NEAT. Since there is no variance in gain for NEAT, a Wilcoxon test was carried out instead of a t-test. Results show,

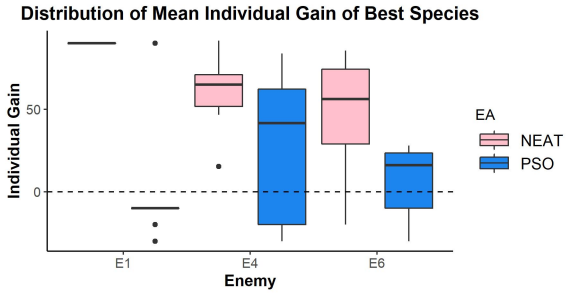

Figure 4: Individual gains for best runs

that NEAT (μ = 90.00, σ = 0.00) is significantly better than PSO (μ = 6.08, σ = 21.46), (W = 90, p-value < 0.001).

For enemy 4, the NEAT player (μ = 60.34, σ = 20.61) performs worse than for enemy 1, though it still always wins. The PSO player, on the other hand, performs better for enemy 4 (μ = 26.34 , σ = 43.86) than for enemies 1 and 6. Nevertheless, the difference in the mean performance between NEAT player and PSO player is still significant (t = 2.22, p-value = 0.04).

For enemy 6, the NEAT player loses a few times (μ = 44.38, σ = 39.35), but the mean individual gain is still comparable to that for enemy 4. The PSO player (μ = 6.08, σ = 21.46) is still significantly worse than NEAT player (t = 2.70, p-value = 0.02).

4 DISCUSSION

The goal for this research was to determine and analyse the difference in performance between a fixed-topology method (PSO) and a TWEANN (NEAT). To conclude, NEAT outperformed PSO, both in terms of acquired fitness and individual gain, as well as the stability of the method. In rare cases, PSO is able to find good solutions, resulting in a high fitness and individual gain, and achieving comparable results to NEAT. The mean performance, however, is low, which leads to the believe that PSO behaves more like random search, instead of providing a stable method to converge to an optimum. In addition, given that PSO is a fixed-topology method, the assumed network structure of PSO might not be optimal. NEAT on the other hand is able to evolve the topology as well, resulting in a more stable and higher performance.

Our results clearly demonstrate the advantage of weight and topology evolving methods over fixed-topology methods. PSO could potentially be improved by investigating different network structures. However, by eliminating the need for human design, topology evolving methods are able to efficiently solve increasingly complex tasks. TWEANNs, like NEAT, may yield promising results in a variety of applications, far beyond (video) games, in the future.

5 CONTRIBUTIONS

Jaimie Rutgers and Martijn Wesselius implemented the PSO method, Ekaterina Geraskina and Dominic Istha implemented the NEAT method. Ekaterina visualized and tested the results. The report itself is the result of the combined efforts of all authors.

REFERENCES

[1] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[2] Marcio Carvalho and Teresa B Ludermir. 2007. Particle swarm optimization of neural network architectures andweights. In *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*. IEEE, 336–339.

[3] Amitava Chatterjee and Patrick Siarry. 2006. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Computers & operations research* 33, 3 (2006), 859–871.

[4] Yuntao Dai, Liqiang Liu, and Ying Li. 2011. An intelligent parameter selection method for particle swarm optimization algorithm. In *2011 Fourth international joint conference on computational sciences and optimization*. IEEE, 960–964.

[5] Karine da Silva Miras de Araújo and Fabrício Olivetti de França. 2016. An electronic-game framework for evaluating coevolutionary algorithms. *arXiv preprint arXiv:1604.00644* (2016).

[6] Karine da Silva Miras de Araujo and Fabrício Olivetti de Franca. 2016. Evolving a generalized strategy for an action-platformer video game framework. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1303–1310.

[7] Russell Eberhart and James Kennedy. 1995. Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, Vol. 4. Citeseer, 1942–1948.

[8] Beatriz A Garro and Roberto A Vázquez. 2015. Designing artificial neural networks using particle swarm optimization algorithms. *Computational intelligence and neuroscience* 2015 (2015).

[9] Francisco Erivaldo Fernandes Junior and Gary G Yen. 2019. Particle swarm optimization of deep neural networks architectures for image classification. *Swarm and Evolutionary Computation* 49 (2019), 62–74.

[10] Kuok King Kuok, S Harun, and SM Shamsuddin. 2010. Particle swarm optimization feedforward neural network for modeling runoff. *International Journal of Environmental Science & Technology* 7, 1 (2010), 67–78.

[11] Hui-Rong Li and Yue-Lin Gao. 2009. Particle swarm optimization algorithm with exponent decreasing inertia weight and stochastic mutation. In *2009 second international conference on information and computing science*, Vol. 1. IEEE, 66–69.

[12] Alan McIntyre, Matt Kallada, Cesar G. Miguel, and Carolina Feher da Silva. [n. d.]. neat-python. https://github.com/CodeReclaimers/neat-python. ([n. d.]).

[13] 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.

[14] Lester James V. Miranda. [n. d.]. PySwarms-Python. https://pyswarms.readthedocs.io/en/latest/index.html. ([n. d.]).

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[16] Sebastian Risi and Julian Togelius. 2015. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 1 (2015), 25–41.

[17] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[19] Kenneth O Stanley, Bobby D Bryant, and Risto Miikkulainen. 2005. Real-time neuroevolution in the NERO video game. *IEEE transactions on evolutionary computation* 9, 6 (2005), 653–668.

[20] Kenneth O Stanley and Risto Miikkulainen. 2002. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, Vol. 2. IEEE, 1757–1762.

[21] Gerald Tesauro. 1995. Temporal Difference Learning and TD-Gammon. *Commun. ACM* 38, 3 (March 1995), 58–68. https://doi.org/10.1145/203330.203343

[22] Jianbin Xin, Guimin Chen, and Yubao Hai. 2009. A particle swarm optimizer with multi-stage linearly-decreasing inertia weight. In *2009 International Joint Conference on Computational Sciences and Optimization*, Vol. 1. IEEE, 505–508.