Evolutionary Computing - Task 1

Group 105 | September 29, 2024

Bartek Golik
ID: 2834060

Francijn Keur
ID: 2669946

Jonas Schäfer
ID: 2753619

Pablo Alves
ID: 2833404

1 INTRODUCTION

1.1 Background

Evolutionary computing builds, applies, and studies algorithms based on the Darwinian principles of natural selection [6]. The resulting evolutionary algorithms act as flexible population-based search algorithms, making them a well-suited and popular tool for multi-objective optimization problems [1]. Since its inception in the turn of the 21st century, evolutionary computing has constituted itself as an active area of research with many applications, such as data mining, routing, engineering, or biochemistry [12].

1.2 Motivation

This research report is part of our submission for Task 1 of the Evolutionary Computing 2024/2025 course at VU. In particular, the objective of this first project was to develop, apply, and compare Evolutionary Algorithms for the task of optimizing a video game controller playing Evoman, a video game framework designed to test optimization algorithms [5]. The multi-objective optimization nature of video games makes them a suitable use-case for deploying evolutionary algorithms[1] [2].

1.3 Research question

The main goal of this project was to implement two distinct evolutionary algorithms and utilize these to optimize weights for specialist neural network game controllers. We are interested in the algorithms performance to generate effective video game controllers (algorithmic players) and assess their performance by playing against 3 selected Evoman enemies. Furthermore, we are attempting to assess similarities and differences in the emergence of "strategies" the generated controllers evolve by analyzing their output vectors throughout evolution.

2 METHODS

The Evoman framework, its manual, and seminal paper [4] were provided[2] as a starting point. This environment, which provides 8 different enemies that the player controller can play against, was used for all our experiments.

2.1 Algorithms

For this study, we considered the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) and the Neuro-Evolution of Augmenting Topologies (NEAT) algorithm. This choice was made as both algorithms are widely adopted but differ vastly in their optimization approach. CMA-ES has few hyperparameters, self-adapts, uses a sampling approach and receives a fixed controller network topology. On the other hand, NEAT requires tuning of a large number of hyperparamters but is capable to adapt the topology of the network. As targets, both algorithms start with a randomly initialized[3] fully connected neural network that receives all 20 inputs from the game, has a single hidden layer of size 10 and returns a decision vector of size 5. This minimalist topology choice allowed us to set a baseline for the comparison of both approaches.

2.1.1 *Covariance Matrix Adaptation Evolution Strategy.*

CMA-ES is a widely used evolutionary algorithm utilized for solving non-linear, non-convex optimization problems, especially in continuous domains. It is considered a very robust numerical optimization method. [8] Fundamentally, it samples around a centroid in the search space using a multivariate normal distribution. By evaluating the fitness of sampled individuals, it determines a new centroid and more promising part of the search space. It then adjusts the covariance matrix in this direction, using a step size σ parameter that it self-tunes at runtime. Using this mechanism, CMA-ES efficiently searches in complex search spaces [8].

2.1.2 *NeuroEvolution of Augmenting Topologies.*

The NEAT algorithm, introduced in 2002, is one of the most influential algorithms in Evolutionary Computing [11]. With extensive use in unsupervised learning tasks, it is designed to simultaneously evolve both network weights and topology while addressing common problems found in Topology and Weight Evolving Artificial Neural Networks (TWEANNs) [3]. In particular, it is summarized by historical marking, speciation and incremental complexification. With historical marking, NEAT associates a unique marking number for each new gene created, allowing the tracking of new genes and correct crossover operation at the right spot [3]. With speciation, NEAT has new network topologies compete against their own niche at first, as to compensate topology changes severely impacting fitness in the short term [3]. Lastly, NEAT commences with a minimal network structure, first optimizing weights before adding nodes and edges to the structure, leading to incremental complexification at a minimized structure [3].

2.2 Enemy selection

To evolve specialist agents, three enemies were selected based on the strategies required to defeat them, noted by de Araujo et. al. [4]. Enemies 1 and 3 were arbitrarily chosen from the set of enemies that could be defeated by a simple strategy of standing and shooting to see if the algorithms would adopt similar tactics. Enemy 4, requiring precise jumps to dodge projectiles, was arbitrarily chosen from the remaining options to test if the algorithms would develop distinct strategies when faced with a more complex challenge.

2.3 Fitness function

To evaluate agent performance, EvoMan provides a fitness function that measures the performance after each game [5] and which we used to guide the evolution process. This function, defined in Equation 1 quantifies the fitness of the agent as a function of the final energy of the enemy (e_e), the final energy of the player (e_p), game time (t) and two adjustable parameters (γ and α)[4].

$$\text{fitness} = \gamma * (100 - e_e) + \alpha * e_p - \log(t) \tag{1}$$

2.4 Experimental setup

To facilitate prototyping, implementation, and testing, the DEAP framework [7] and NEAT-Python package [10] were used. To train

[1]In the context of shooting videogames, such as Evoman, this includes, among others: reducing the enemy's health, keeping one's health, beating the enemy as fast as possible and navigating the environment effectively.
[2]This paper also includes a baseline performance of selected evolutionary algorithms
[3]$\mu = 0$ for CMA-ES, $\mu = 0.5$ for NEAT at initialization

[4]By default, the EvoMan framework uses parameter values $\gamma = 0.9, \alpha = 0.1$

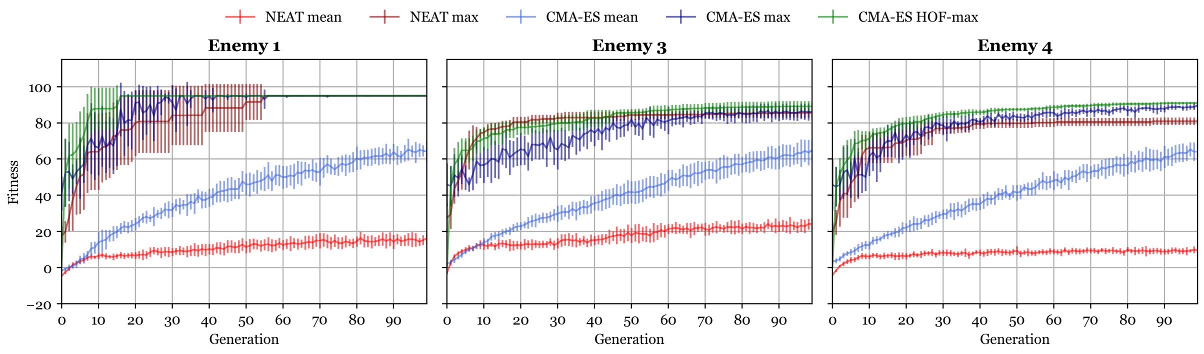

Figure 1: Inspection of convergence and fitness. Visualized are mean and standard deviation of mean and max fitness values obtained from 10 runs per enemy and algorithm, plotted across 100 generations. HOF denotes the hall of fame.

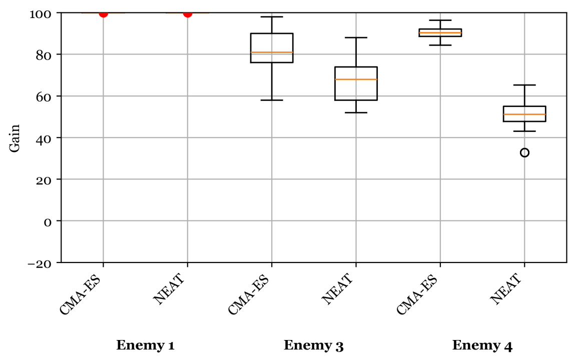

Figure 2: Boxplots of the mean gain of the 10 best specialists of each algorithm and enemy playing against the denoted enemy 5 times. All enemy 1 runs achieved maximum gain.

the specialist agents for each game and algorithm, the simulation mode "Individual evolution" and enemies of type "static" of the Evoman framework were used [5]. No other game environment-altering settings have been adjusted. For compute, common laptops were used and experiments were limited to be on the order of hours.[5].

Gain function. To measure post training performance of the fittest individuals of each population, we used the *individual gain* function defined in Equation 2, where e_p and e_e are again the final player and enemy energy levels respectively[6].

$$Gain = e_p - e_e \tag{2}$$

Fitness function calibration. Qualitative investigation suggested that retaining the players' energy was not weighted enough for our purposes. We further anticipated it would lose utility whenever fast-winning strategies were not achievable. After further analysis, the final fitness function was set by fixing $\gamma = 0.75$, $\alpha = 0.25$ in Equation 1.

Finetuning of hyperparameter values algorithms. Hyperparameter values were not adjusted between experimental runs. CMA-ES hyperparameter values were fine-tuned on enemy 3 experimentally by inspecting convergence speed and final performance of 5 different configurations. Similarly, 15 configurations were assessed for NEAT, starting off with hyperparameter values informed by previous work [4] [9] and the best ones chosen. We opted for fine-tuning on enemy 3 as it shared strategies with enemy 1 (which was easier to defeat) making success likely transferable[7]. Population size and number of generations were set to 100 in both algorithms to improve comparability, which enabled sufficient convergence.

2.4.1 Performance comparisons. In this first experiment, we compare both algorithms by enemy as a function of agent fitness throughout 100 generations[8]. Trained specialist agents were evolved 10 times independently to receive the average of the mean and maximum value over the population of each generation.[9] All statistics were then based on these 10 runs. Additionally, to further compare these two algorithms, we tested the performance of their best solution of each of the 10 runs for each game using the gain metric. Here, each specialist plays the game 5 times and their performance is the mean of the 5 runs[10]. A T-test was then performed to evaluate differences in mean gain between the two algorithms for each enemy.

2.4.2 Average ratio of actions performed throughout evolution. To gain additional insight into the solution space that our algorithms converged to and their qualitative differences, we decided to analyze how often the resulting agents activated their different outputs - aggregated as jumping, shooting, releasing, and moving (left/right) - throughout their games as they were trained. For this,

[5]For further details on environment, algorithm implementations, full parameter specification and the code files generating the results, please consult the *README* file of the GitHub repository and associated code files.
[6]Since the maximum values of e_p and e_e are (by definition) 100, the range of this function spans the $[-100, 100]$ interval

[7]Fine-tuning on enemy 3 would also showcase more easily how the hyperparameter values enable the more complex strategy required to defeat enemy 4.
[8]This value was set after exploratory runs to ensure fitness convergence
[9]For CMA-ES, we also visualize HOF-max, the max value in the hall of fame, so the individual with the highest fitness found thus far. We do this since the algorithm does not retain strong individuals in its population but resamples at each generation.
[10]These 5 game repetitions are done to account for potential pygame timing and EvoMan game stochasticity impacting replicability. The specialist controllers are determinsistic at this stage.

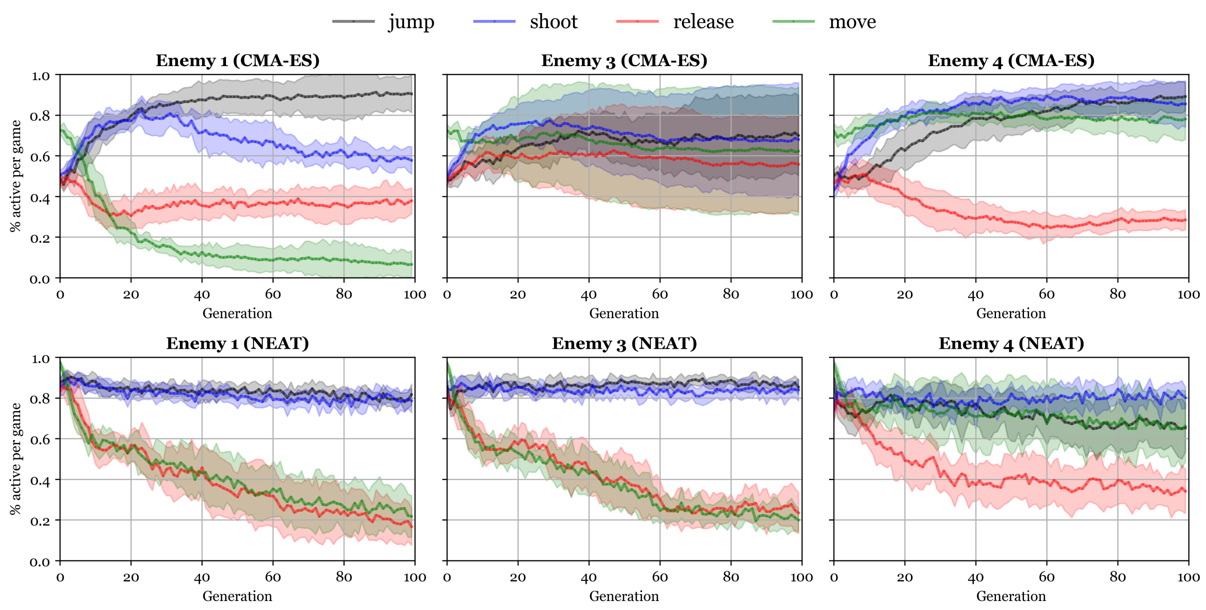

Figure 3: Fraction of time agents spent on their available agent outputs during each evolution generation as a function of enemy (column-wise, Enemy 1 on the left) and algorithm (row-wise, CMAES on top).

we considered the mean fraction of frames that the agent performed these activities[11].

3 RESULTS AND DISCUSSION

As seen in Figure 1, the performance of NEAT and CMA-ES converged to the same solution for the first enemy. For the other two enemies, NEAT was slightly outperformed by CMA-ES. Furthermore, our results showed that the mean fitness of NEAT was much lower than that of CMA-ES and also increases much more slowly. We believe this to be attributed to the algorithms themselves, since NEAT introduces changes that can compromise short-term fitness (thus motivating speciation) but this may also indicate the need for better hyperparameter tuning.

Our comparative analysis of the gain function, as shown in Figure 2, shows that both algorithms achieved a perfect score for the first enemy, defeating the enemy consistently without being hit. For enemies 3 and 4, CMA-ES outperformed NEAT, noting significantly higher gain values (t-test p-values $\simeq 2.25 \times 10^{-7}$ and 0.00, respectively) and a surprisingly consistent performance against enemy 4. For enemy 3, we observe large variances for both algorithms, likely due to the complexity of dodging this enemy's attack. When comparing to the baseline paper [5], we find that NEAT outperforms its baseline for enemy 1 but scores lower for the other two enemies [5]. We are uncertain as to why this is the case. Unfortunately, no CMA-ES baseline comparison values are available.

When comparing the strategies evolved by CMA-ES and NEAT in figure 3, clear differences emerge in their adaptation to different enemies over time. While CMA-ES initially performs each action roughly 50% of the time, NEAT starts off by performing all actions each frame, a consequence of their difference in network initialization. For enemy 1, CMA-ES evolves a minimal-movement strategy, relying heavily on jumping and shooting. Against enemy 3, the strategy involved a balanced mix of actions and large variations without any single dominant action or strategy emerging. For enemy 4, movement, jumping, and shooting were prioritized over releasing from jumps. For NEAT, the strategies evolved for enemies 1 and 3 are very similar and focus mainly on jumping and shooting. Against enemy 4, NEAT placed more emphasis on movement and slightly more frequent releases from jumps compared to the other enemies. Overall, it is interesting to note how NEAT evolved strategies that were more similar to each other than the strategies evolved by CMA-ES. However, since all movement was aggregated, no additional conclusions can be drawn from this experiment.

4 CONCLUSION

In this study, we implemented two distinct evolutionary algorithms which evolve specialist controllers to play against enemies in the EvoMan framework. Emerging controller performance varies by enemy and algorithm but they consistently converge to beat their rule-based enemies, showcasing the ability of evolutionary algorithms at generating "smart" video game players. Our results also showed that CMA-ES consistently converged faster and to a better fitness than NEAT despite fixed topology. We recommend future work to further study these algorithms using algorithmic fine-tuning; to assess the qualitative aspects of the learned agent strategies as to gain additional insight into strategy convergence across algorithms and lastly computational and fitness trade-offs of different evolutionary algorithms.

[11]Unfortunately, the EvoMan framework did not provided us with a straightforward way to access detailed game state information to generate metrics such as shooting accuracy or dodging success rate. Thus, we resorted to this simpler analysis

REFERENCES

[1] Thomas Bartz-Beielstein, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. 2014. Evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4, 3 (2014), 178–195.

[2] T Bullen and M Katchabaw. 2008. Using genetic algorithms to evolve character behaviours in modern video games. *Proceedings of the GAMEON-NA* (2008).

[3] Lin Chen and Damminda Alahakoon. 2006. NeuroEvolution of Augmenting Topologies with Learning for Data Classification. In *2006 International Conference on Information and Automation*. 367–371. https://doi.org/10.1109/ICINFA.2006.374100

[4] 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.

[5] Fabricio Olivetti de Franca, Denis Fantinato, Karine Miras, AE Eiben, and Patricia A Vargas. 2019. Evoman: Game-playing competition. *arXiv preprint arXiv:1912.10445* (2019).

[6] Agoston E Eiben and Marc Schoenauer. 2002. Evolutionary computing. *Inform. Process. Lett.* 82, 1 (2002), 1–6.

[7] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[8] Nikolaus Hansen. 2016. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

[9] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. 2014. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 4 (2014), 355–366.

[10] Alan McIntyre, Matt Kallada, Cesar G. Miguel, Carolina Feher de Silva, and Marcio Lobo Netto. [n. d.]. neat-python. ([n. d.]).

[11] Evgenia Papavasileiou, Jan Cornelis, and Bart Jansen. 2021. A Systematic Literature Review of the Successors of "NeuroEvolution of Augmenting Topologies". *Evolutionary Computation* 29, 1 (2021), 1–73. https://doi.org/10.1162/evco_a_00282

[12] Pradnya A Vikhar. 2016. Evolutionary algorithms: A critical review and its future prospects. In *2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC)*. IEEE, 261–265.