Evolutionary Computing Task I

Vrije Universiteit Amsterdam, Group 40

Elena Deiana
e.deiana@student.vu.nl
2749820

Nikita Tiwari
n.tiwari@student.vu.nl
2699057

Jordan Phillips
jordan.phillips@columbia.edu
2754973

Jochem Herrebrugh
j.w.herrebrugh@student.vu.nl
2677519

Cas Hoekstra
c.hoekstra@student.vu.nl
2788848

1 INTRODUCTION

Genetic Algorithms (GA's) are a group of computational models that find potential solutions for a specific problem in a large solution space. By using a chromosome-like data structure of solutions combined with steps as: initialization, crossover, selection, replacement and mutation, it displays great similarities with one of the best problem solvers known to man: evolution [9]. While this way of exploring the solution space does not guarantee to find the optimal solution for a given problem, experience has shown it can provide good solutions for a wide range of problems [1]. The quality of these solutions depend, apart from randomness, from the steps described above. Due to the high number of combinations of parameters within these steps, this article will focus solely on the effect of two different selection procedures within a GA.

1.1 Selection procedures

The selection phase determines which individuals are selected for reproduction and how many offspring each individual produces within a GA. Selection is one of the main steps in GA performance, which largely depends on the convergence rate and total number of generations to reach the global optima.

The first and most traditional selection procedure that is being reviewed, is the Tournament Selection. Tournament Selection holds a tournament of a certain tournament size among the individuals. Every tournament winner is inserted in a mating pool and hereby selected for reproduction. This selection procedure is characterized by a high selection pressure, meaning the degree to which preference is given to better solutions is high when selecting for reproduction. This could result in a quick way of finding the right solution, but could also result in the GA converging too quickly and finding a sub optimal solution [10]. Adjusting the selection pressure is possible by changing the tournament size.

The second selection procedure being reviewed is the Best Selection. This selection simply grabs the top individuals. This creates a rapidly converging system that does not promote diversity as likely many of the best solutions are similar, but it does allow for quickly increasing fitness scores as the quality of the pool in each generation is very high from removing all of the poor performers.

The goal of this assignment is to compare the selection techniques Tournament Selection and Best Selection.

Our research question is the following: *Does the tournament selection method improve performance of player in the evoman framework?* The motivation behind this question was due to fact that the tournament selection proved to give best results in [8] among other selection methods.

1.2 Evoman

'Evoman' is a platform game created by Karina da Silva Miras [4], based on the classic platform game 'MegaMan'. Evoman is aimed to test evolutionary algorithms in their effectivity to create an agent that is able to fight one of eight distinct enemies. Every fight both the agent and the enemy start with 100 lifepoints and during the fight the agent has the ability to shoot projectiles at the enemy and hereby decreasing the enemy's health when hitting. Likewise, the enemy is able to decrease the agent's health by both hitting projectiles and having physical contact with the agent. When the lifepoints of one of the two players reaches zero, the level ends. When the agent has fought all the eight enemies, the game ends. In this research is decided to train against three enemies: 1, 5 and 7. In this way we keep variety in our tests since enemy 1 can take 4 actions, enemy 5 can take 3 actions and enemy 7 can take 6 actions [3].

2 ALGORITHMS AND METHODS

After researching different frameworks such as DEAP, NEAT, Pyvolution and jMetalPy, we chose to work with DEAP (Distributed Evolutionary Algorithms in Python) [6] for its flexibility, extensive documentation, and simplified implementation.

2.1 Parameter Settings

2.1.1 Population size. We used $n = 3$ individuals for both algorithms. We opted for a moderate population size so the populations are able to develop sufficient diversity. This leads to more efficient exploration and exploitation of the solution space [5]. Despite research in [7] recommending a population size between 20 to 100, this amount proved to be far too large for us to tune our algorithms and required additional time budget. Indeed, when running an experiment with a higher population, equal to 50, as shown in 7 resulted in an improvement of the mean of maxes and a constant upward trajectory of the average fitness although at a high variance and computational cost (2 hours for one enemy and one algorithm). Thus we found more success with using a smaller population size.

2.1.2 Initialization. To initialize our population we randomly generated arrays representing the hidden weights in the single neural network with 10 neurons each with values between -5 and 5 and drawn uniformly at random.

2.1.3 Generation Size. The generation size for both algorithms was $n = 7$ and our termination criterion is simply from running the algorithms for 30 generations.

2.1.4 Fitness. To evaluate a solution we use the same fitness function as in [4]. The objective of this fitness function is to find a solution which attempts to minimize the enemies' life while maximizing player life, under the consideration of game duration.

2.2 Algorithm 1 - Tournament Selection

We used the "Mu Plus Lambda" algorithm, in which the next generation population is selected from both the offspring and the population. We think this will allow for a higher quality of solutions in each generation, but does come at a cost of diversity.

2.2.1 Crossover. Uses 'cxEsBlend' crossover method with an alpha = 0.3. This was chosen as it allows for a tunable learning rate, which helps to prevent getting caught in local maxima.

2.2.2 Mutation. Uses 'mutGuassian' with a mu = 0, sigma=1, and indpb=0.5. Since this simply applies a Gaussian to each weight with probability of 0.5, this allowed us to have a far better understanding of what this algorithm was doing with the weights in the neural network. This proved important for training the tournament, as it already promoted higher diversity (from the tournament itself) and thus having less diversity from the mutation method was helpful for more consistent converging.

2.2.3 Selection. Uses the tournament selection method with tournament size = 4'.

2.3 Algorithm 2 - Rank Selection

We used the "Mu Comma Lambda" algorithm, in which the next generation population is selected only from the offspring.

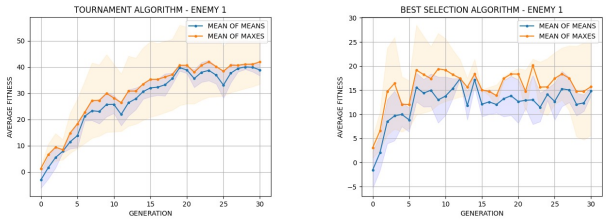
2.3.1 Crossover. Uses 'cxEsBlend' crossover method with an alpha = 0.3. This is a tunable learning rate operator, while this does not directly translate the 'logic' in each of the parents, this ends up getting good results as it helps to preserve diversity.

2.3.2 Mutation. Uses 'mutESLogNormal' with c = 1 and indpb=0.5. This was chosen as it allows for a tunable learning rate, which helps to prevent getting caught in local maxima, and additionally, creates higher diversity in the resulting offspring.

2.3.3 Selection. Rank-selection: Uses the best selection method, that is takes k best individuals according to fitness values.

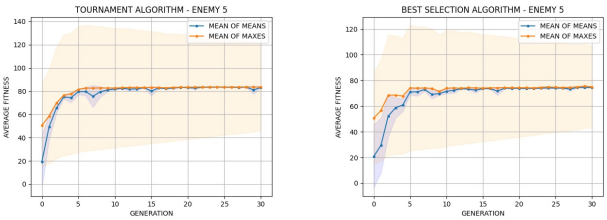
3 RESULTS AND DISCUSSION

Overall, the results were not as satisfactory as hoped for. We believe that there were two main reasons for this. First, we had to cap the generations and the number of individuals at a low number, and second we used random initialization. This created a compounding effect, where the problem is more complex, and we gave the algorithm less time to evolve to find a good solution. However, we believe that the positive results for the easier enemies are a good indication that with more resources this setup would yield good results for all of the enemies.



(a) Algorithm 1 **(b) Algorithm 2**

Figure 1: Enemy 1



(a) Algorithm 1 **(b) Algorithm 2**

Figure 2: Enemy 5

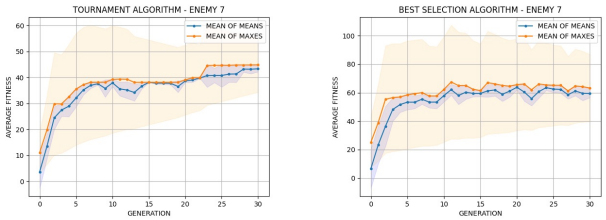
3.1 Algorithm 1 vs. Algorithm 2

We compare the two algorithms in terms of their maximum fitness (orange line) and their mean of means (blue line), which are obtained by averaging the results of 30 generational cycles over 10 experimental runs. The algorithm 1 (with tournament selection as described in 2.2) seems to perform better in general except in the case of enemy 7 as seen in fig. 3, where algorithm 2 (described in 2.3) manages to obtain higher fitness. However mean of mean fitness of algorithm 1 is still increasing at generation '30', while that of algorithm 2 seem to converge at optimal.

In the case of enemy 5, both algorithms perform in a similar pattern, with algorithm 1 having a slightly higher fitness. In general algorithm 2 shows more outliers than algorithm 1 (fig. 4 and 5) This implies that the majority of solutions of algorithm 2 were concentrated around a specific fitness value with a small minority being outside that value, while the solutions of algorithm 1 are within a larger range; we attribute this to the diversity maintenance of the algorithm 1 due to tournament selection. This is also shown in [2].

The hardest enemy to beat for both algorithms seemed to be enemy 1, as seen from fig. 1 and 4. However from fig. 1 it is seen that while algorithm 2 seems to converge to the optimal solution at generation '30', algorithm 1 keeps optimizing (as mean of mean fitness keeps increasing), and this could lead to higher fitness values if allowed to run for more number of generations.

To conclude our comparison of the averaged run results of the two algorithms we can say that tournament and rank selection both managed to win against the three enemies (1, 5, and 7) and obtain different results for the enemy types tested.



(a) Algorithm 1 **(b) Algorithm 2**

Figure 3: Enemy 7

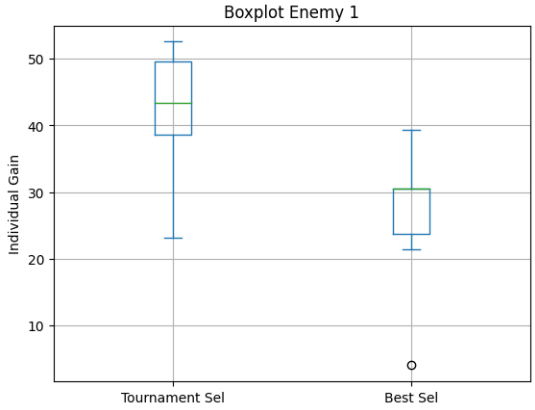


Figure 4: Box plot enemy 1

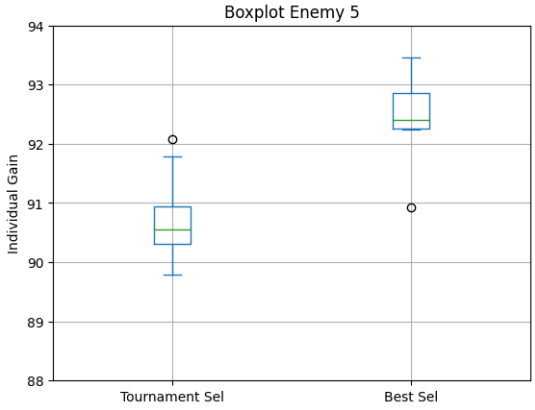


Figure 5: Box plot enemy 5

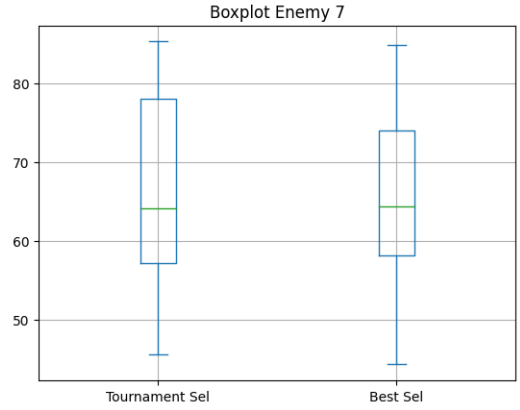


Figure 6: Box plot enemy 7

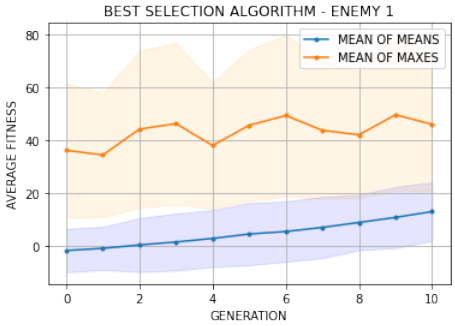


Figure 7: Best selection algorithm with population=50, lamda = 70, 10 generations for enemy 1

3.2 Algorithm vs. Baseline

Compared to the baseline paper [4], our algorithms appears to have lower performance for each enemy (1, 5, and 7) as seen from the results (fig. 1, 2 and 3). In the baseline the parameters used are *crossover probability* = 1, *mutation probability* = 0.20, and *population size* = 100, which is different to the parameter settings used in our experiment. This may be one of the reasons for the obtained results. Some of the similarities between the baseline paper and our findings(i.e., beating the 3 specified enemies) can be explained by the use of the same fitness function and selection mechanism (tournament). We believe that using a higher population size as in the baseline paper and tuning our algorithms could obtain results as obtained in [4].

4 CONCLUSIONS

By using common evolutionary algorithmic methods along with tournament and ranking selection methods, we managed to beat three different enemies without using neuroevolution methods as in [4], though performance of our algorithms was on the lower side. Additionally, increasing the number of population and generations and with parameter tuning could improve the performance of our algorithms significantly in the future.

REFERENCES

[1] Pratibha Bajpai and Manoj Kumar. 2010. Genetic algorithm–an approach to solve global optimization problems. *Indian Journal of computer science and engineering* 1, 3 (2010), 199–206.

[2] Carlos A Coello Coello and Efrén Mezura Montes. 2002. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics* 16, 3 (2002), 193–203.

[3] 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).

[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] A.E. Eiben and J.E. Smith. 2015. *Introduction to Evolutionary Computing*. Springer. https://doi.org/10.1007/978-3-662-44874-8 Gebeurtenis: 2nd edition.

[6] 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.

[7] Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, and VB Surya Prasath. 2019. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information* 10, 12 (2019), 390.

[8] Noraini Mohd Razali, John Geraghty, et al. 2011. Genetic algorithm performance with different selection strategies in solving TSP. In *Proceedings of the world congress on engineering*, Vol. 2. International Association of Engineers Hong Kong, China, 1–6.

[9] Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and computing* 4, 2 (1994), 65–85.

[10] Jinghui Zhong, Xiaomin Hu, Jun Zhang, and Min Gu. 2005. Comparison of performance between different selection strategies on simple genetic algorithms. In *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)*, Vol. 2. IEEE, 1115–1121.