Evolutionary Computing

Task 1: Testing the efficacy of different Evolutionary Algorithms in an electronic-game framework

ADRIAN FUERTES BLANCO (11320141), CATERINA BURANELLI (13344897), KAMIEL GULPEN (11308699), and KATJA VAN DER PERK (10650075), Vrije Universiteit, The Netherlands

*Team 59

Authors' address: Adrian Fuertes Blanco (11320141); Caterina Buranelli (13344897); Kamiel Gulpen (11308699); Katja van der Perk (10650075), Vrije Universiteit, Faculty of Science, Amsterdam, The Netherlands.

2020. Manuscript submitted to ACM

1 INTRODUCTION

In order to test and compare two Evolutionary Algorithms (EAs) EvoMan, an electronic-game framework, was used. In this game, an algorithm-controlled player faces 8 enemies. Each enemy follows a different set of fixed rules and, through generations, the EA can learn how to fight and beat all the given enemies.

The problem in this assignment can be described as an optimization problem. An optimization problem lays its focus on finding the inputs while the model and a description of the desired outputs are known [3]. In this project, the desired output is producing an agent with the best fitness, meaning an agent who can beat three enemies in the lowest amount of time and with the most health left.

In order to achieve that, a Genetic Algorithm was implemented. The features which define this algorithm concern the selection strategy, the recombination and the mutation strategy. These features will be further explained in the Methods section. The two different algorithms tested differ in the mating strategies chosen. In the first algorithm, the parents within each generation are not formed by the whole population, but by individuals with the best fitness value. Therefore in the mating pool duplicates are allowed. This algorithm will be referred to as *Tournament*. In contrast, the second algorithm takes the whole population as parents and not only the best part of it. This algorithm will be referred to as *All*. The comparison resulted in the following research question:

What is the difference in performance, compared over 3 different enemies, of a genetic algorithm with a mating pool based on the whole population and a genetic algorithm with a mating pool selected by tournament?

The three hypotheses, made to answer this research question are:

H1: The genetic algorithm with a mating pool selected by tournament has a better performance against enemy 1

H2: The genetic algorithm with a mating pool selected by tournament has a better performance against enemy 2

H3: The genetic algorithm with a mating pool selected by tournament has a better performance against enemy 3

The results of both algorithms are compared to the algorithms which are given in the baseline paper [1]. From this paper the so-called GA and NEAT are used as baseline.

2 METHODS

In this analysis the enemies fought by the agent were number 1 (Flashman), 2 (Airman) and 3 (Woodman). The population consists of 50 individuals whom evaluated over 15 generations, starting from generation 0. The number of hidden neurons was kept at 10 and the initial set of weights was randomly generated from the Uniform distribution, with -1 and 1 as lower and upper bounds. The budget for this analysis was equal to the number of generations used in the algorithm, as more generations would give more time to search better solutions. However this would take more time for each algorithm and enemy to search through the search space.

The DEAP framework (Distributed Evolutionary Algorithms in Python) [4, 5] toolbox was used for the implementation of the EAs. This framework includes the structure of the algorithm and it allows the use of different features and functions, enabling variation and customization for the task at hand. The algorithm starts with setting the initial parameters and randomly generating the phenotypes of generation 0.

The fitness function used to evaluate the success of the algorithm, was kept the same as in the original algorithm. This is given in the initial EvoMan framework [1, 2], where fitness is a function of the enemies' life (e_e), the player's life (e_p) and the time spent in a fight (t). The exact function is described as follows:

$$fitness = \gamma * (100 - e_e) + \alpha * e_p - \log t \quad \text{where } \gamma = 0.9 \text{ and } \alpha = 0.1$$

The individual gain is used as an alternative metric to compare the two algorithms and is given as follows:

individual gain = player life - enemy life

For the recombination strategy, the n-Point Crossover method [3] is used, with $n = 2$; it has been implemented using a function from the DEAP toolbox. The genotypes are split into three segments where two random numbers r_1 and r_2 in the range $[1, L - 1]$ determine the location of the splits. L refers to the length of the genotype. The values are swapped between both individuals and two children are generated. This does not happen for each couple: the probability that two parents will actually mate, after being selected, is set at 80%.

The mutation operator used in the framework is the Flip bit method [3], where each attribute is flipped with a probability of 50%. This operator will change the Lp_m value, where L is the length of the genotype and p_m is the mutation rate. In this algorithm, simulated annealing is added to the mutation part: mutation only happens with a probability given by MUTPB, which is calculated as $1 - \frac{g}{g_{end}}$, where g is the current generation and g_{end} is the total number of generations. This way, local optima should be avoided in the beginning, while the algorithm will explore a smaller area of the genotype space during the last generations. [3]

The selection algorithm for the individuals uses a method to ensure the best individuals are kept in the next generation. The method selects the k best individuals, where k is chosen to be 10%, and swaps them with the k worst individuals. The fittest individuals survive and the next generation is composed of them and their offspring. This selection strategy is implemented with the `deap.tools.selBest` method.

The difference in the two algorithms compared can be found in the selection method of the parents for breeding the next generation. One algorithm uses the whole population as parents, while the other one randomly divides the population into groups of k individuals - $k = 5$ in our case - and selects the best one out of the tournament. The larger is k, the higher the probability of selecting high fitness individuals. This implies that the programmer has more control on the selection pressure. Using this method, the main issue given by a selection based only on the absolute fitness value of the individuals is avoided: the algorithm will not end up prematurely, as the random element guarantees a more deep research into the solution space. Moreover, the tournament selection is an operator built on the ranking order of the individuals, therefore it does not involve a more general knowledge of how to measure the quality of the performance, than the relative fitness between the members. This is useful not only in this specific case, but also in the evolutionary game field overall. Tournament selection was implemented using `deap.tools.selTournament` method from the DEAP toolbox.

Finally, after the ten simulations, for each enemy the best individual gain was taken and tested five times. The mean values of these solutions were saved and visualised to check whether the algorithms have significantly different result in the analysis.

3 RESULTS AND DISCUSSION

When tested against enemy 1 (Figure 1a and 1d), the individual gain of the Tournament algorithm significantly outperformed the All algorithm (Table 1). Figure 1a shows the maximum fitness level is reached after generation 12 for both algorithms, showing a higher rate of improvement for the Tournament algorithm. The same pattern can be found when comparing the relative change in mean fitness. These findings supports $H1$.

When tested against enemy 2 (Figure 1b and 1d), both the algorithms reach the maximum fitness value in generation zero; as this is due to random dynamics, it is more informative to follow the trend of the means after more generations. From that, indeed, it shows that the individual gain of the Tournament algorithm is significantly better than the All

Table 1. For each enemy, the following values are given: the mean of the best personal gain for the two tested EAs and the pvalue of the t-test, the mean of the best fitness for the two tested EAs and the two baselines.

		Individual gain			Individual fitness			
	Enemy	Tournament	All	pvalue	Tournament	All	NEAT	GA
1	Flashman	65	-15	< 0.01	90	71	90	90
2	Airman	85.2	77.4	< 0.01	93	93	94	90
3	Woodman	47.4	40.4	< 0.5	88	88	80	58

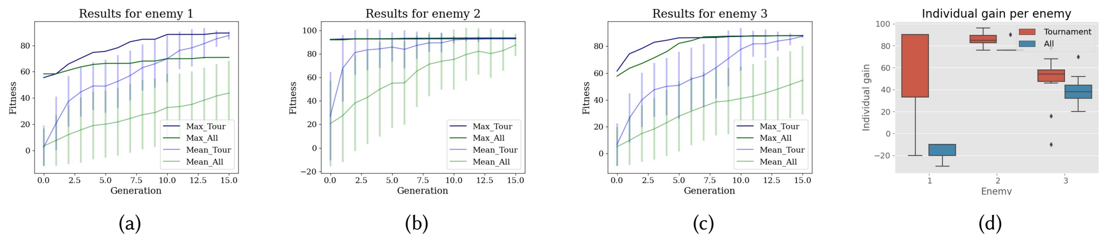

Fig. 1. Mean (95% confidence interval) and maximum fitness per generation of the two compared EAs for (a) enemy 1, (b) enemy 2 and (c) enemy 3. (d) Boxplot with the invidual gain for enemy in the two EAs.

algorithm(Table 1). Figure 1b shows the curve increases fast in the beginning and from generation 11 the mean reaches the maximum value. These findings supports *H*2.

When tested against enemy 3 (Figure 1c and 1d), Figure 1c shows that the maximum fitness value is reached at generation 5 and 7 for the Tournament and All algorithm respectively. The mean fitness for the Tournament has a faster increase than the mean fitness for the All algorithm. In the final generation of the Tournament algorithm the mean fitness value reaches the max fitness value. So visually there seems to be a difference in the mean fitness value. The individual gain of the Tournament algorithm has a higher score than the All algorithm, but this is not significant (Table 1). This means that the null-hypothesis can not be rejected and *H*3 can not be accepted.

The answer to the research question, which algorithm performs better compared over 3 different enemies, is that the Tournament algorithm performs significantly better for enemy 1 and 2 and performs better but not significantly for enemy 3. As observed in Table 1, the Tournament algorithm only outperformed NEAT and GA, our baseline, when tested against enemy 3 and performed equally against enemy 1. The All algorithm only performed better than the baseline against enemy 3. A point to mention here is that in the NEAT algorithm the number of generations is set to 100, while here this number is set to 15: therefore it is possible that with more generations, thus more time to perform the simulation, the Tournament algorithm would perform even better. Another cause for the lower score could be that they were stuck in local optima.

4 CONCLUSIONS

In general, the Tournament algorithm produced better results than the All one; moreover, it was also good relatively to the baseline. A possibility for improvement could be found in increasing the budget: the number of generations used to search through the solution space.

REFERENCES

[1] Miras, Karine, F. O. (2016a). An electronic-game framework for evaluating coevolutionary algorithms. *arXiv preprint arXiv:1604.00644.*

[2] Miras, Karine, F. O. (2016b). Evolving a generalized strategy for an action-platformer video game framework. In *2016 IEEE Congress on Evolu- tionary Computation (CEC)*, pages 1303–1310. IEEE.

[3] Eiben, A.E. and Smith, J.E. (2004), *Introduction to Evolutionary Computing*, Assembly Automation, Vol. 24 No. 3, pp. 324-324.

[4] https://github.com/DEAP/deap

[5] Félix-Antoine Fortin, François-Michel De Rainville *et al.*. 2012. *DEAP: evolutionary algorithms made easy.* J. Mach. Learn. Res. 13, 1 (January 2012), 2171–2175.