I Will Survive: Comparing Replacement Strategies of Evolutionary Algorithms within a Static Environment

Evolutionary Computing | Standard Assignment 1: Specialist Agent | Team: 112 | September 29, 2024

Mikel Blom
Vrije Universiteit Amsterdam
2867792

Alexandra Genis
Vrije Universiteit Amsterdam
2834714

Daimy van Loo
Vrije Universiteit Amsterdam
2852535

Martino Zaratin
Vrije Universiteit Amsterdam
2860138

ABSTRACT

The development of autonomous agents is relevant to many fields of industry and is commonly achieved through Evolutionary Algorithms (EAs). An integral part of EAs is their survivor selection mechanism, to which there are two common approaches - generational and steady state selection. While steady state selection has been found more effective in dynamic environments, research is still lacking on differences between these strategies in a static environment. Hence, this study compared the performance of two EAs, employing steady state and generational survival strategies respectively, in a static game context. The findings suggest that, contrary to expectations, both approaches have the same impact on algorithm performance in a static environment. Implications of these findings are discussed.

1 INTRODUCTION

The field of Artificial Intelligence (AI) has historically concerned itself with the development of autonomous agents for various purposes such as manufacturer industries or unmanned vehicles [1]. Control structures for such agents are often developed by using Evolutionary Algorithms (EAs) [7]. EA is a broad term for a family of biomimetic computational methods that aim to find the solution for a given problem by recreating an evolutionary cycle with a population of solutions [3]. Eiben and Smith offer a comprehensive overview of the field [3]. One important component of each EA, which has considerable impact on the outcome is survivor selection, a mechanism defining which individuals of a given population are transferred to the next generation [8]. To this date there are only two different survivor selection mechanisms known - the generational (GEN) and the steady state (STS) selection approach. GEN implies that the parent generation is fully replaced by the offspring, while STS replaces only a fraction of the parent population with offspring.

Vavak and Fogarty compared STS and GEN in genetic algorithms (GAs) within a dynamic environment and found a "deleting the oldest" steady state replacement strategy to be superior to the generational GA in terms of performance [9]. Similarly, Zavoianu et al. found the STS model to outperform its GEN opponent in the context of asynchronous, multi-objective EAs [10]. Further, Jones and Soule found significant differences in genetic robustness between the outcomes of STS and GEN selection strategies, which they attribute to selection pressure [4]. Yet, the concrete involvement of selection pressure in genetic robustness as well as any further potential differences between these two approaches still require research [4].

While prior research investigated differences between STS and GEN selection in the context of dynamic or otherwise complex environments, those environments also presented an intricate interaction between different factors in which confounding influences on the algorithms cannot be ruled out easily. Thus, it is still unclear how these two approaches would behave in a simple, predominantly static environment. Accordingly, the present study investigates the differences between steady state survivor selection and generational survivor selection in the evolution of autonomous agents in a static environment.

Based on prior research we expect to see a higher performance of the algorithm using STS selection than the algorithm applying a GEN selection method. To test this hypothesis, two EAs were created using an STS and GEN selection approach respectively. These algorithms were subsequently compared on their performance, for which a game context was selected as the most straightforward implementation option. EvoMan, the game framework used for this purpose, will be further introduced in the Methods section.

2 METHODS

The present study compares two approaches for survivor selection - STS and GEN - for the evolution of a controller network for an autonomous agent in a game context.

2.1 Implementation Details

Code written for the present study used Python Version 3.14 [6] and adapted the EvoMan framework [5].

2.2 EvoMan Framework

EvoMan is a multipurpose Computational Intelligence framework created for video games and serves as a benchmarking platform for the development of autonomous agents [2]. Inspired by Capcom's 1987 game Mega Man II, the framework features eight 'boss fight' stages where the player faces unique enemies, each with distinct attack patterns and weaponry. Both player and enemy start with 100 energy points, decreasing after each attack. The game ends when either of them reaches an energy score of 0, determining the winner.

2.2.1 Autonomous Agents. The EvoMan framework allows for both player and enemy to be controlled by either the user or an autonomous agent [1]. Possible actions of each player are moving left or right, jumping, shooting, and interrupting a jump. In the human player version, these actions can be controlled through keypresses. In the case of autonomous play, control over each action is facilitated through input from 20 different sensors, containing information about the environment. This information is fed to an autonomous controller, which makes decisions about the next action based on that input. Controllers can be specialist, meaning optimisation of a player against only one enemy, or generalist, implying the optimisation for multiple different enemies. The present study utilized the neural network (NN) controller provided within the EvoMan framework to construct a specialist agent.

2.3 Evolutionary Algorithms

For the present study, two Evolutionary Algorithms (EA1 and EA2) were constructed in line with instructions from Eiben and Smith [3]. Considering the large search-space size, all tuneable hyperparameters were selected on the basis of feasibility, namely through heuristics and line/grid searches. Hyperparameters of each method are elaborated on in the respective sections.

EA1 and EA2 both evolve populations over 50 generations/epochs with the goal of optimizing performance against one enemy. Early experimentation clarified that convergence could be achieved within the first 20 epochs. However, marginal improvements are possible if the algorithm is allowed to run for an extended period of time.

Accordingly, 50 epochs balance the potential for improvement with computational cost. Further, both algorithms have a largely shared architecture. The only differences comprise their survivor selection mechanism and a related part of parent selection. While EA1 is equipped with a STS survivor selection mechanism, EA2 employs GEN selection. First, the shared setup of both algorithms is described while their difference will be elaborated on further below.

2.3.1 Population. The population was initialised by random draws from a uniform distribution of continuous values ($U[-1, 1]$). As the player controller neural network uses a Sigmoid activation function, weights valued around 0 can be assumed to be most useful. A population size of n=100 was chosen to balance the need for diversity, the quality of the found result, and a sufficiently fast computation time.

An individual consisted of the weights and biases for the NN in the player controller, which contained 10 hidden neurons. This resulted in a vector length of 265 for each individual.

2.3.2 Evaluation. The standard fitness function of the EvoMan framework $fitness = \gamma(100 - e_e) + \alpha e_p - \log(t)$ with $\gamma = 0.9$ and $\alpha = 0.1$ was adapted for evaluation [2] . Fitness comprises a composite fitness score based on player energy points, enemy energy points, and the time it took to finish the game.

2.3.3 Tournament Selection. For selecting the parents, standard tournament selection was employed, with $k = 5$ candidates. During hyperparameter tuning, we initially assumed that the optimal value for k would be near 5. This was confirmed by a line search for $k \in \{2, 3, ..., 8\}$ yielding that the optimal balance between convergence and diversity is indeed achieved for $k = 5$ in combination with random draws without replacement. Since we use exponentially ranked populations, the tournament winner is the individual with the highest rank [3, p. 85]; thus fitness values are not recalculated when performing the tournament selection.

Further, EA1 required the choice of a generation gap G to balance the risk of too similar behaviour of both algorithms (too large G) and the possibility of underperformance (too small G). Initially, experimentation led to a baseline generation gap at 0.5. A further line search for $G \in \{0.2, \ldots, 0.8\}$ by increments of 0.1 revealed a $G = 0.8$ to be most suitable. This implies that parents are selected until the number of children is sufficient to replace 80% of the population.

In EA2, parent selection is repeated until the number of children reaches the size of the population during each epoch.

2.3.4 Whole Arithmetic Crossover. For recombination, whole arithmetic crossover was adapted with an $\alpha = 0.5$ selected to reduce the hyperparameter search space [3]. This implies that a single child is produced as the weighted average of its parents.

2.3.5 Non-Uniform Mutation. Each child is mutated using a non-uniform mutation with probability P [3, p. 57]. Mutations are drawn from a Gaussian distribution with scale parameter/mutation step size σ. Non-Uniform Mutation was chosen based on implementation simplicity compared to some other methods proposed in Eiben and Smith [3, Chapter 4]. For the tuning of the mutation hyperparameters, first experiments were performed on the ranges $P \in [0.2, 0.8]$ and $\sigma \in [0.01, 0.2]$. We ultimately narrowed down our grid search to $P \in \{0.7, 0.75, 0.8\}$ and $\sigma \in \{0.06, 0.08, 0.1\}$. An optimum was found for $P = 0.75$ and $\sigma = 0.1$.

2.3.6 Survivor Selection EA1: Steady State Model. The steady state model replaces the "bottom" 80% of the population, which scored lowest on fitness, with all newly generated offspring.

2.3.7 Survivor Selection EA2: Generational Model. In the generational model, the whole parent population is replaced by the offspring, with one exception: if the population of offspring does not contain an individual that performs at least as well as the previous best individual, this previous best individual replaces the worst individual from the offspring. This ensures that the best individual found always remains in the population.

2.4 Experiments

The performance of EA1 and EA2 in training a specialist controller NN was compared in an experimental setup. For training and testing, three enemies were selected, namely e1, e2, and e3. This selection was based on the similarity of enemies, as higher similarity allowed for straight-forward comparison between the two models. Enemies used for training and testing were static, implying deterministic behaviour.

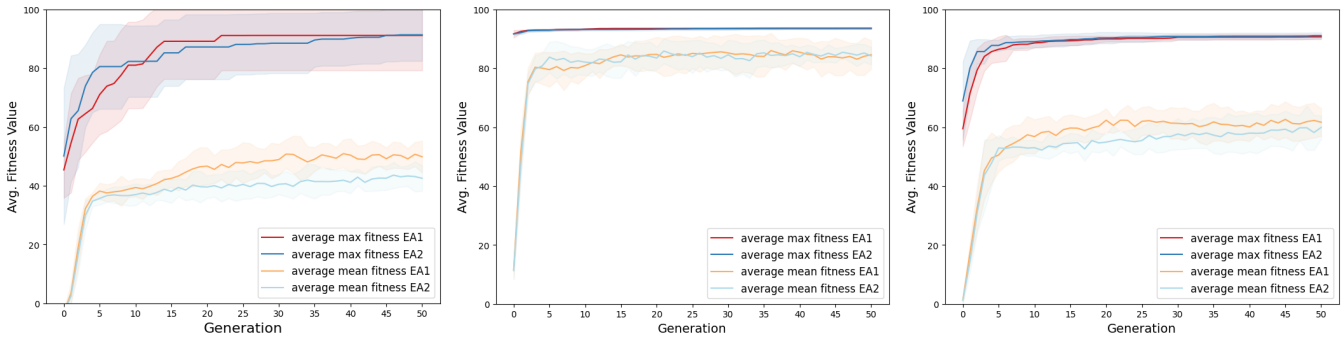
2.4.1 Algorithm Training. Parameters of EA1 and EA2 were initially optimised through parameter tuning on e3. Subsequently, 10 training runs of each EA were performed on e1, e2, and e3, respectively. From each of the 60 training runs, the solution with highest fitness for the individual enemies was selected per run, resulting in a set of 60 unique solutions.

2.4.2 Algorithm Testing. Each solution was tested five times on the enemy it had been trained on and evaluated based on the individual gain metric (g), which is the difference between player energy points (e_p) and enemy energy points (e_e): $g = e_p - e_e$. The average individual gain of these five test runs was then used for statistical testing.

2.4.3 Statistical Tests. A two-sided t-test evaluated the difference in testing performance of each EA on each of the three enemies.

3 RESULTS

3.0.1 Training. Fig. 1 presents the performance of EA1 and EA2 on each enemy during training. While EA1 has a slightly higher mean than EA2 in most experiments, their standard deviations appear to overlap, which indicates similar performance. Likewise, the overall behavior of both algorithms is similar. A shared characteristic is the gap between the mean fitness and the max fitness, which is located much higher in all plots. This gap implies that the population doesn't lose diversity. In the case of e3, the enemy used for parameter tuning, both algorithms show early convergence during the first few epochs. A more distinct but still small difference in performance between EA1 and EA2 can be observed when training on e1. No difference is visible in performance on e2. Further, for e2 both algorithms show strong early convergence. The results imply that the tuning of parameters obtained for e3 works well for e2 and e1.



(a) Enemy 1 **(b) Enemy 2** **(c) Enemy 3**

Figure 1: Mean and Maximum Fitness Values of EA1 and EA2 per enemy

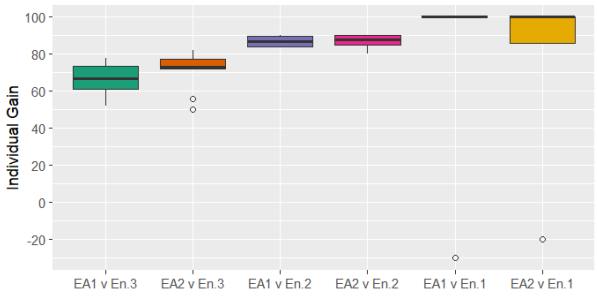


Figure 2: Individual Gain of EA1 and EA2 tests

3.0.2 Testing. Fig. 2 displays the distribution of individual gain of each EA per enemy. Both EA1 and EA2 are able to beat each of the three enemies with some consistency, since all box-plots are situated well above an individual gain of 0. The performance of EA1 and EA2 appears very close, as all box-plots for the same enemy are positioned around the same values of individual gain. An interesting result is that both EA1 and EA2 have exactly one outlier individual for enemy 1, which is not able to beat the enemy.

3.0.3 Statistical Tests. The three two sided t-tests are presented in Table 1 and showed no significant difference between the mean individual gain of each EA for any enemy (all $p > 0.05$).

Enemy	t-value	df	p-value
e3	-0.905	17.54	0.378
e2	-0.29	17.07	0.775
e1	0.183	17.81	0.857

Table 1: Differences in individual gain between EA1 and EA2 on all three enemies.

4 DISCUSSION

The present study investigated the differences between steady state survivor selection and generational survivor selection in the evolution of autonomous agents in a static environment. Two algorithms, EA1 (steady state) and EA2 (generational), were used to evolve weights for the controller NN of an autonomous player agent in the benchmarking game EvoMan. Based on prior research, EA1 was expected to yield higher performance.

Results revealed that both algorithms were capable of producing an agent which consistently beat the enemy. However, there was no difference in performance between the algorithms.

One possible explanation for this unexpected result is that the generational gap of EA1 was set too high, increasing its similarity to EA2. Setting the generational gap to 0.8 worked best in our model but was also obtained as a result of its interaction with the other hyperparameters. This demonstrates that the component-wise approach to parameter tuning can be quite problematic because of the latent interaction effects within the algorithm. Further research could thus conduct this experiment with a different combination of hyperparameters or use a predefined algorithm while merely adjusting the replacement strategy.

Another interesting result is that both EAs seem to have a small chance of getting stuck on a sub-optimal solution with enemy 1, where one out of ten times the best individual after training is not able to beat the enemy. Further research could investigate whether a mechanism restarting the algorithm in case of it getting stuck could resolve this problem.

One other limitation is the use of whole arithmetic crossover with one child ($\alpha = 0.5$). This is not optimal because an $\alpha \neq 0.5$ could have fostered more diversity.

Finally, the testing yielded too little data for meaningful statistical results. Future studies are advised to replicate the experiment with a higher quantity of runs.

Evolutionary Algorithms find applications in the training of autonomous agents in many different industries. The full impact of the two main replacement strategies within these algorithms - steady state and generational survivor selection - remains widely unknown. Contrary to prior research, this study reveals that these two selection strategies perform equally well in a static game context.

REFERENCES

[1] Karine Da Silva Miras De Araujo and Fabricio Olivetti De Franca. 2016. Evolving a generalized strategy for an action-platformer video game framework. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, Vancouver, BC, Canada, 1303–1310. https://doi.org/10.1109/CEC.2016.7743938

[2] Karine da Silva Miras de Araújo and Fabrício Olivetti de França. 2016. An electronic-game framework for evaluating coevolutionary algorithms. (April 2016). http://arxiv.org/abs/1604.00644 arXiv:1604.00644 [cs].

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

[4] Josh Jones and Terry Soule. 2006. Comparing genetic robustness in generational vs. steady state evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, Seattle Washington USA, 143–150. https://doi.org/10.1145/1143997.1144024

[5] Karine Miras. 2016. Evoman Framework. https://github.com/karinemiras/evoman_framework. (2016). Accessed: 2024-09-28.

[6] Python Software Foundation. 2023. Python Language Reference, version 3.11. https://www.python.org/. (2023). Accessed: 2024-09-28.

[7] Ralf Salomon. 1997. The evolution of different neuronal control structures for autonomous agents. *Robotics and Autonomous Systems* 22, 3-4 (Dec. 1997), 199–213. https://doi.org/10.1016/S0921-8890(97)00039-0

[8] Jim Smith and Frank Vavak. [n. d.]. Replacement Strategies in Steady State Genetic Algorithms: Static Environments. ([n. d.]).

[9] F. Vavak and T.C. Fogarty. 1996. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, Nagoya, Japan, 192–195. https://doi.org/10.1109/ICEC.1996.542359

[10] Alexandru-Ciprian Zăvoianu, Edwin Lughofer, Werner Koppelstätter, Günther Weidenholzer, Wolfgang Amrhein, and Erich Peter Klement. 2015. Performance comparison of generational and steady-state asynchronous multi-objective evolutionary algorithms for computationally-intensive problems. *Knowledge-Based Systems* 87 (Oct. 2015), 47–60. https://doi.org/10.1016/j.knosys.2015.05.029