Performance of diverse and elitist evolutionary algorithms against EvoMan

Fernando Gomez-Acebo Ruiz
Vrije Universiteit
Amsterdam, the Netherlands
f.gomez-acebo.ruiz@student.vu.nl

Eline van de Lagemaat
Vrije Universiteit
Amsterdam, the Netherlands
2628256@student.vu.nl

Sofia Tavitian
Vrije Universiteit
Amsterdam, the Netherlands
s.h.n.tavitian@student.vu.nl

Ziya Alim Sungkar
Vrije Universiteit
Leiden, the Netherlands
z.a.sungkar@student.vu.nl

1 INTRODUCTION

The concept of evolutionary computing has existed since the 1940s when Turing proposed “evolutionary search”. This involves using trial and error to solve complex problems, imitating Darwinian evolution[1]. From there, a promising scientific field emerged, as it has the capacity to explore solutions to problems that lie in a vast, non-linear solution space, where a straightforward mathematical approach to finding the solution might not exist. Algorithms are used to test the fitness of solution populations and "genetically" combining the fittest individuals to arrive at an optimum. Varying the crossover, mutation and selection methods gives rise to algorithms that traverse the solution space in a manner catered to the problem at hand. This variation allows these algorithms to search for solutions close to the current fittest solutions while simultaneously exploring the more distant solution space, preventing convergence to a local optimum.

This paper examines two algorithms designed to enhance player performance in the EvoMan game. It explores whether maintaining population diversity in an evolutionary algorithm is beneficial to finding an optimal solution as opposed to a more elitist algorithm which only evolves the best performing individuals. If elitism wins out, this could be beneficial to computational cost. We expect to find that a diverse algorithm takes longer to converge but finds better solutions than an elitist method.

To test this hypothesis, we build two algorithms, one diverse and one elitist in selection, recombination and mutation methods. The methodology describes the algorithms in detail along with parameter settings, after which we explore the results.

2 METHOD

A diverse and an elitist evolutionary algorithm are implemented as class objects, inheriting from a general class that defines the evolutionary structure while collecting fitness and diversity data.

The population consists of individuals, their genomes is a collection of weights describing probabilities for performing certain actions in game. As the generations go by, these weight are updated and selected based on success.

,
,
2024.

Table 1: Overview of algorithmic components for elitist and diverse evolutionary algorithms.

	Elitist	**Diverse**
Parent selection	Tournament	Fitness sharing
Crossover	Multipoint	Gene pool
Mutation	Uniform mutation	Cauchy mutation
Survivor selection	Elitist	Roulette Wheel

2.1 Evolution

2.1.1 `create_population`. A population of 100 individuals is instantiated by building random genomes comprised of floating points between -1 and 1.

2.1.2 `simulate`. Each individual of the population plays a game against a specified enemy and their fitness is determined.

2.1.3 `evolve`. The basic steps of an evolutionary algorithm are performed for a preset number of generations. An initial population plays the game against a given enemy and based on calculated fitness ten parents are selected. Using multi-parent crossover, the ten parents produce 100 offspring, each undergoing a mutation.

2.1.4 `calculate_population_diversity`. The genome diversity in a population is calculated. This is quantified as the averaged pairwise Euclidean distance between all individuals.

2.2 Elitist algorithm

2.2.1 `tournament_selection`. Tournament selection repeatedly samples subsets of the population of size k. It performs elitist selection by only keeping the best performing individual in this subset and discarding the rest. This process is repeated until the desired selection size is reached.

2.2.2 `multipoint_crossover`. For every child, 19 crossover points in the genome are chosen. This divides the genome in 20 sections which are each filled with the genes of a parent. As there are twice as many sections than parents, the parents provide genomes sections twice. The order in which parents provide genome sections is randomized for each child and so are the crossover points. This method retains large sections of a genome, promoting more elitist selection.

2.2.3 `uniform_mutation`. Genetic diversity is introduced by randomly altering individual genes (weights) in the neural network, with probability p_{mutate}. The gene's value is then replaced by a new random value sampled from the uniform distribution in the desired weight range [`low_weight, upp_weight`].

2.2.4 `elitist_selection`. All individuals are ranked in order of fitness, the best n of them are selected and the rest discarded. This is the elitist selection is ever going to get.

2.3 Diverse algorithm

2.3.1 `fitness_sharing`. This algorithm punishes the effective fitness of individuals that are too genetically similar to each other. The similarity between genomes is calculated as their Euclidean distance and a sharing function is applied based on this value. If two individuals are closer than a predefined sharing radius σ_{share}, they share part of their fitness with each other, reducing their effective fitness.

$$sh(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{\text{share}}}\right)^{\alpha} & \text{if } d < \sigma_{\text{share}} \\ 0 & \text{if } d \geq \sigma_{\text{share}} \end{cases}$$

The effective fitness of an individual f' is calculated by dividing its original fitness by the sum of sharing values for all individuals within the sharing radius. This way, individuals in "crowded" areas have their fitness values reduced more significantly than those in less crowded areas, promoting genetic diversity.

$$f'(i) = \frac{f(i)}{\sum_{j=1}^{N} sh(d_{ij})}$$

where d_{ij} is the calculated distance between individuals i and j.

2.3.2 `gene_pool_crossover`. A parent pool is created for each gene in the genome. Crossover is then performed by randomly selecting from the pool for each respective gene. This allows greater recombination by selecting genes independently from different parents, leading to more unique combinations of genes, and reducing the likelihood of offspring inheriting entire large segments from a single parent.

2.3.3 `cauchy_mutation`.

$$f(x) = \frac{1}{\pi\gamma\left[1 + \left(\frac{x - x_0}{\gamma}\right)^2\right]}$$

Genetic mutation is performed by adding Cauchy-distributed random values to individual genes (weights), scaled by a factor γ, with probability p_{mutate}. The mutated values are then clipped into the desired weight range [`low_weight, upp_weight`]. This method enhances diversity due to the Cauchy distribution having a heavier tail than normal distributions. This allows for both small and occasionally large mutation, increasing the probability of escaping local optima.

2.4 Hyperparameters

Each algorithm trains against enemies [1,2,3] for 30 generations using 10 hidden neurons. For each enemy it performs 10 individual runs, resulting in 60 total simulations. Genomes consist of 265 weights, calculated from (20 input + 1 bias) * (10 neurons) + (10 neurons + 1 bias) * (5 output).

Table 2: Parameter settings for evolutionary simulations.

Parameter	Value
Hidden neurons	10
Weight bounds	$[-1, 1]$
Population size	100
Generations	30
Parents for crossover	10
Enemies	1,2,3
Mutation rate	0.2
Cauchy scale	0.4
Sigma share	0.3

The population size stays constant at 100 individuals and every evolution algorithm goes through 30 generations. Each genome is a weight within the range [-1, 1]. These parameter values were chosen based on those used in the original code [2].

Mutations occur with a probability of 20%, to balance exploration and exploitation in the solution space. The Cauchy mutation is scaled by a factor of 0.4 as this improved diversity above the elitist algorithm. Sigma share of 0.3 is implemented to promote diversity in the algorithm.

Both crossover methods generate offspring from a selected group of ten parents, a number chosen to promote the retention of the fittest individuals.

3 RESULTS AND DISCUSSION

Figure 1 shows the best and average performance (with standard deviation) of our diverse and elitist algorithms on the selected enemies, together with the averaged genetic diversity per generation across all enemies. It can be observed that in all cases both the best elitist and diverse individuals converge early, and achieve similar fitness.

On average, the elitist algorithm performs slightly better when trained on enemies two and three, while the diverse one outperforms it when trained on enemy 1. However, the standard deviation indicates that this difference is not statistically significant.

The genetic diversity plot indicates that the diverse algorithm is generally more diverse. However, it does not exhibit increased diversity in the first three generations, a critical period during which solution improvements are most significant. The three enemy plots of Figure 1 show that the diverse algorithm generally converges slightly faster than the elitist one. At first glance this observation seems to contradicts our initial hypothesis. However the fast convergence occurs within the first few generations, when the diverse algorithm shows less diversity. Therefore it could be correct to say that an elitist algorithm converges faster, but for future work we should ask ourselves why our diverse algorithm was not showing as much diversity as the elitist one in the first few generations. It is also important to note that the y-axis for diversity ranges from 12 to 14, thus what may appear as a large difference is in fact small.

While we originally posited that by maintaining diversity local minima could be escaped from, it seems that the diversity level in the diverse algorithm is not sufficient to cross the critical diversity threshold necessary for successful exploration beyond the local minimum. Diverse evolution does not appear to increase diversity

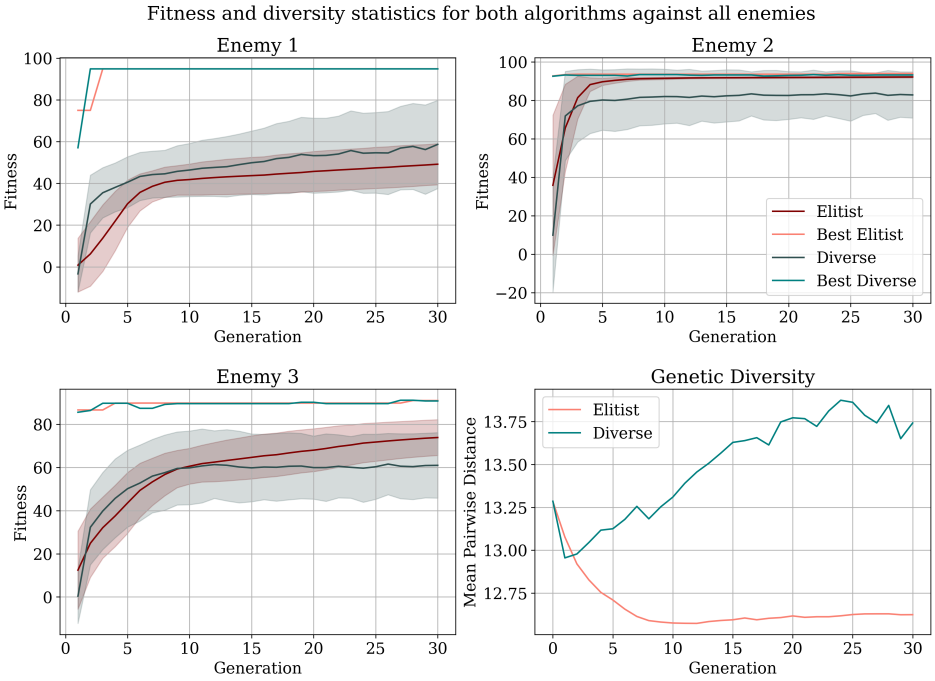

Figure 1: Comparison of fitness for elitist and diverse algorithms against enemies [1,2,3]. All algorithms are averaged over their 10 respective runs and 100 individuals per population with standard deviation shown as a shaded area. The diversity subplot combines all enemies per EA and averages the genomic diversity over 3000 individuals per algorithm.

significantly higher than the average difference from randomly created genomes, which lies around 13.25. Future work could therefore explore mutation methods that introduce larger changes in the individual, to increase genetic diversity further.

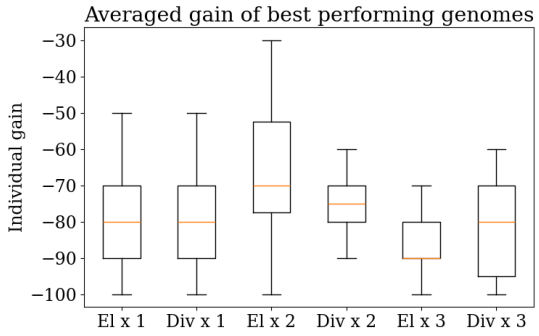

Figure 2: Individual gain of best performing specialist genomes against other enemies.

The current configuration of 10 parents and 100 offspring also limits the initial diversity of the diverse algorithm. Increasing the number of parents may prevent the risk or premature convergence to sub-optimal results, and enhance genetic diversity in the initial population.

For Figure 2 the best performing specialists were run against the enemies they had not been trained on and their gains ($e_{\text{player}} - e_{\text{enemy}}$) were calculated. We can see that all gains were negative ranging, between -30 to -100, with a median of -70 for each group. The genomes may have specialized too much on the enemies they were trained against and failed to generalize well for new scenarios, which is expected, given that our goal was to create specialist individuals. Interestingly, both algorithms show similar performances. An individual originating from a diverse population does not necessarily make it a better generalist, it only means it has explored more space before arriving to an optimal solution for that enemy.

4 CONCLUSION

In conclusion, while both algorithms achieved similar fitness levels, the diverse algorithm did not consistently show a significant advantage in escaping local minima or finding superior solutions, for example due to its limited parent pool size. It also did not exhibit slower convergence than the elitist algorithm, meaning it was not more costly to run. Future work could focus on enhancing genetic diversity and avoiding premature convergence, for instance by increasing the parent pool size from 10 to 50 individuals. Experimenting with adaptive diversity mechanisms, such as dynamic mutation rates or adaptive fitness sharing, could help maintain a more balanced exploration-exploitation trade-off throughout the evolution process.

REFERENCES

[1] Agoston E Eiben and James E Smith. 2015. *Introduction to evolutionary computing*. Springer.
[2] Karine Miras. 2019. *EvoMan - Framework 1.0.*