Effect of constant vs adaptive mutation

Vrije Universiteit Amsterdam - Evolutionary Computation: Task 2 - Group 1

18-10-2021

Alicja Dobrzeniecka

2737091

a.m.dobrzeniecka@student.vu.nl

Melis Nur Verir

2726079

m.n.verir@student.vu.nl

Rumyana Krumova

2718449

r.krumova@student.vu.nl

Dominique Woof

2634493

d.woof@student.vu.nl

KEYWORDS

Artificial intelligence (AI), Evolutionary Algorithms (EA), mutation operator, crossover operator, adaptive mutation operator, hill climbers, tournament selection, multi-objective optimization

ACM Reference Format:

Alicja Dobrzeniecka, Rumyana Krumova, Melis Nur Verir, Dominique Woof

2021. Effect of constant mutation vs adaptive one: Vrije Universiteit Amsterdam - Evolutionary Computation: Task 2 - Group 1 18-10-2021. In Proceedings of ACM Conference (Conference'17). ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.

https://doi.org/10.1145/nnnnnnn.nnn

1. INTRODUCTION

Recently, artificial intelligence (AI) and evolutionary algorithms (EA) are being used together to tackle optimization challenges [1]. One good example of such challenge is training an AI to play the EvoMan video game. This is the subject of this research and will be done by means of EAs. The two main forces underlying evolutionary systems are **selection** and **variation operators**. The creation of new individuals is achieved through the variation operators, which can be divided into two types: **recombination** and **mutation**. This research will focus on the latter. Mutation is an exploitive force which creates small random variations and can thereby introduce novelties within the populations. The values of the parameters within the operators can be chosen by means of either **parameter tuning** or **parameter control** [2]. When chosen for parameter control, there are three common types: **deterministic, adaptive,** and **self-adaptive** [3]. In this research the difference between a mutation operator with adaptive parameter control and a mutation operator without any parameter control will be explored.

1.1 Research question

As mutation plays an important role within evolutionary algorithms, it seems likely that the **choice of mutation operator** can have a large impact on the new population and consequently to the whole algorithm. This assumption serves as the motivation for choosing to investigate the difference in performance of two evolutionary algorithms that differ only in a choice of mutation operator.

Parameter control places the responsibility for the parameterisation task (partly) with the evolutionary process and thereby offers the possibility to use appropriate values in various stages of the evolution. However, improvement in the performance of the EA does not have to be trivial, the EA can for example get stuck in a local optimum. This is addressed, inter alia, in the 2007 article *Parameter Setting in Evolutionary Algorithms* [3]. Furthermore, our previous research has shown that an EA with only non-adaptive operators is already capable of creating very fit individuals. Hence, it might be possible that the implementation of adaptive operators makes little to no difference in terms of the resulting fitness of the individuals.

To gain more insight into this topic, the first EA in this research uses a constant mutation operator, namely the Gaussian mutation operator from the DEAP framework [4], while the second algorithm will make use of a custom adaptive mutation operator. The research question of this study is as follows: **Can we implement an adaptive mutation operator that outperforms the constant Gaussian mutation operator from the DEAP framework?**

1.2 Hypothesis

Based on the knowledge obtained from Eiben and Smith [2] and our previous research, our hypothesis is that the adaptive mutation operator will outperform the constant Gaussian mutation operator from DEAP. To verify this, the two EAs will be trained and tested by means of the EvoMan framework [8]. Previous research showed that finding optimal parameter values for various operators is a difficult task. Apart from it being a time-consuming task, **the optimal value may vary** during different stages of evolution and thus finding the one perfect fixed parameter value might neither be feasible nor realistic. Therefore, determining the value of the parameter by means of a feedback rule using real time statistics, is believed to improve the quality of the mutation operator and thereby the overall performance of the algorithm.

2. METHODS

2.1 The Evolutionary Algorithms and their Operators

2.1.1 Parent selection. In this research we implemented a **tournament selection operator** to choose parents for the creation of new offspring. This operator does not require global knowledge of population fitness. One randomly selects *k* individuals and compares their fitness. The individual with the highest fitness is chosen as the first parent. Then one should repeat this process with new random individuals from the population for the second parent. The advantages of this approach are that from one hand the implementation is simple and computational costs are low and from the other hand, one can easily control the selection pressure by adjusting the *k* value. In our implementation we used *k=4* as the tests showed that this value works well for the respective problem.

2.1.2 Crossover. In this research the **two-point crossover operator** is chosen in accordance with the results from our previous research, in which it was concluded that it outperforms the uniform crossover operator when used in the EvoMan setting. Two-point crossover is a technique where two new individuals are created by interchanging the genetic information between two points in the genotypes of the parents [5].

Table 1: Two-points crossover settings

Parameter	Value
First and second point	Random value from a uniform distribution

This operator's parameters, shown in *table 1,* are a pair of crossover points, randomly chosen from a uniform distribution. This operator interchanges the genetic information by performing one-point crossover twice. After crossover has taken place, mutation is performed over each new individual.

2.1.3 Evolutionary Algorithm 1: Gaussian mutation. The type of mutation operator used in the first algorithm is a Gaussian operator provided by the DEAP framework [6]. This operator's mutation is a technique, where a value from a Gaussian distribution is drawn from a distribution with a set mean and sigma. Hence the parameters are constant. The mutation value is added to certain genes, on the condition that the random number generated for the specific gene is smaller than the mutation rate, which is again constant.

Regarding the parameters for the Gaussian distribution, in this research the mean is set to 0 and sigma to 0.1 in order to make sure that the changes resulting from the mutation are not too large. With too large mutation values the EA is prone to having difficulties obtaining the best solution as it can fall from local optimum more easily. The mutation rate is set to 0.2.

2.1.4 Evolutionary Algorithm 2: Adaptive mutation. The type of mutation operator used in the second algorithm is a custom implementation of adaptive mutation. Here during a mutation, a value generated from the normal distribution is added to certain genes, on the condition that the current randomly chosen number is smaller than the mutation rate. The **mutation rate** is given an initial value at the beginning but **evolves** over the generations according to the comparison of the **average population fitness** and the current **offspring fitness**. If the offspring's fitness is smaller than the average population fitness, it means that the solution has low quality, and the mutation rate is increased to increase the chances of a successful mutation, hence better offspring. If the offspring's fitness is bigger than the average population fitness, it means that the solution is of high quality, and the mutation rate is lowered.

Regarding the parameters for the algorithm, the initial value of the mutation rate is set to 0.4. Due to computational exhaustion the adjustment of a mutation rate is applied not in every generation but in every three generations.

2.1.5 Probability-based selection. It was decided to implement a survivor selection that takes advantages from both elitism and probability-based results. The **survivor selection operator** determines which individuals are to be eliminated and which are to be kept in the next generation, while at the same time maintaining the population diversity and size.

Within this specific selection, **randomness** is introduced with respect to how many of the individuals with the highest fitness are guaranteed to survive. The random variable used to determine this takes on a value between 0.02 and 0.05. In the case of 100 individuals, this entails that between 2% and 5% of the fittest individuals will always survive. Next, the rest of the individuals are selected form the population (now consisting of parents and new offspring) with the use of **probability**, giving a higher chance to those individuals that obtained higher fitness but not guaranteeing them to be chosen.

As a result, **diversity** will be preserved and the assumption that the best individuals do not always result in the best offspring is still met. Only selecting and mating individuals with high fitness causes the offspring to all have similar characteristics which causes the fall in the diversity.

2.1.6 Elitism. The elitism strategy means passing the genes from the fittest individuals to the next population without making any changes to their gene structure. It entails that these individuals will **not undergo the process of crossover and mutation**. This method was selected so that some of the best solutions are retained and not lost when variation operators are applied. In our experiments we use the benefits of elitism to sustain all non-redundant individuals of the previous population.

The amount of the individuals chosen as elitists depends on a chosen percentage range. After some literature research and testing, we decided that the elitism ratio should be randomized between (0.1, 0.2) [7]. We tested the upper boundary equal to 0.1, 0.2 and 0.3. The best performance was obtained with the upper boundary equal to 0.2.

After the elitists are chosen, they are preserved and at the end of the evolution of the current generation they are compared to the final new offspring. If the elitists are better than the worst offspring, then they are taken to the next generation, otherwise they are discarded. After some testing, it became clear that combining these elitist individuals with the offspring resulting from the mutation and crossover methods, improves the performance.

2.1.6 Local search: Hill climbers. In our EAs we implemented a simple version of local search. This approach is supposed to examine candidate solutions that outperform the current individual. This process often leads to the discovery of local optima. A **local optimum** is a solution that is superior to the solutions of the **neighboring individuals**. Our **Hill climbers** method is applied in each generation to only four best individuals. With this design we wanted to avoid the risk of getting stuck with local optima in all solutions. In this operator we try to improve the individuals by applying mutation to their genes. After the mutation phase we evaluate if the fitness improved, if it is not the case then we continue the search for a better solution. There are two break conditions, either we find a better individual, or we arrive at the limit of trials. The limit of trials is not constant but deterministic. We start with 20 trials and then with each generation we subtract 1 from the amount. This is due to computational costs but also the fact that often as the number of generations increases gets more difficult to find a better solution by applying mutation only. In general, we noticed that the implementation of local search allows us to find the best solution faster.

2.2 Experimental setup

2.2.1 Initial setup and additional functions. Each simulation is initialized by using the default EvoMan parameters, and by selecting the group of enemies necessary for the current experiment. Experiments are run with **population size 100** and **10 generations**. Enemies' groups that are used for the current research are **[2,6]** and **[7,8]**.

The neural network contains **10 hidden layers**. The initial genes for the **100 individuals** that were created at the beginning of each experiment, are given a random value between **-1 and 1**. During evolution, there is also a limitation method, which keeps the new genes in the range of -1 and 1.

The fitness function used for the general strategies experiments is given by the EvoMan framework and it is the average of the obtained individual fitness for each one of the training enemies. The individual fitness is obtained by the following formula:

$$f = 0.9 \cdot (100 - e) + 0.1 \cdot p - \log t \quad (1)$$

Where f is the fitness, p is the player energy, e is the enemy energy, and t is the time for ending the fight.
Finally, a normalization function is used, which is necessary for the process of converting the fitness values to probabilities in survival selection.

2.2.2 Budget. The predetermined budget assumed roughly 60 minutes per experiment. Eventually, it took us around 18 hours to run 20 experiments on a Ryzen 7-4800H/32GB/960 GTX1660Ti 144Hz.

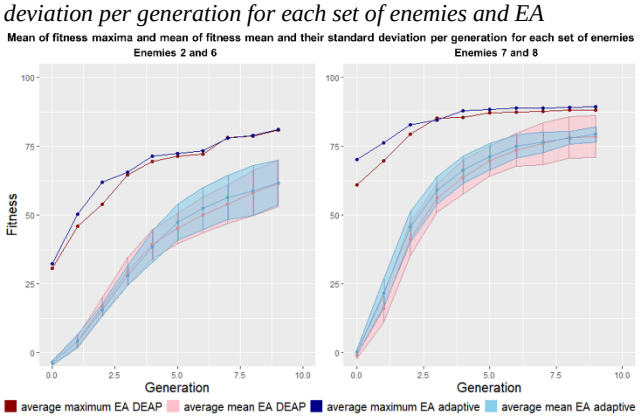
3. RESULTS AND DISCUSSION

As explained in *Section 2.2.1*, two EAs are trained and tested for **10** generations, **10** runs and two groups of enemies. The results of these runs are shown in figures below.

3.1 Training results

Figure 1 shows that the average fitness maximum, mean and standard deviation per generation are somewhat similar for both EAs during the training for each set of enemies. It can also be seen that the training accelerates between generation **0 and 4** but starts to decrease in pace after. Some differences in the standard deviation **depending on the set of enemies** that we trained on, can be observed. The standard deviation of the training for enemies' group **[2, 6]** has higher maximum values for the adaptive mutation, whereas for the enemy group **[7, 8]** they are higher for the Gaussian mutation. The mean for both algorithms in both groups does not appear to be significantly different. However, the mean is lower in the set of enemies **[2, 6]**. The maxima for the group **[2, 6]** come together after generation 6, whereas for the **[7, 8]** group the adaptive mutation stays a bit higher. Overall, the algorithms show similar behavior in both sets of enemies.

Figure 1. Plots for mean of fitness maxima, mean and standard deviation per generation for each set of enemies and EA



3.2 Test results

In order to test the two EAs, the best solution of each of the 10 training runs was tested 5 times against all enemies for both EAs. The average gains of each set of 5 tests were calculated and the results are visualized in the boxplots in *figure 2*.

Figure 2. Box plots for the gain per enemy group and algorithm

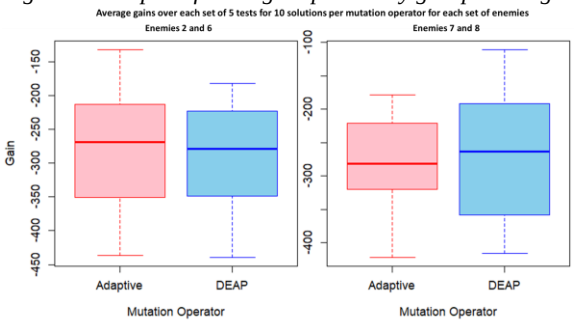


Figure 2 indicates that for both sets of enemies neither of the two EAs outperforms the other. This is suggested by the observation that in both cases the boxes overlap, the median lines lie within the overlap between the two boxes, and the differences between the median lines seem very small. Furthermore, the ranges of all plots do not seem to be very condensed, hence both methods show a considerably large variance.

Table 2: The average gain of our best solution

Enemy	Player Points	Enemy Points	Gain
1	0	90	-90
2	77.2	0	77.2
3	0	52	-52
4	0	70	-70
5	15.32	6	9.32
6	0	54	-54
7	46.64	12	34.64
8	34.44	0	34.44

Table 2 shows the gain for our best solution. Up until now we were able to win against 4 enemies.

3.4 Comparison with EvoMan framework initial experiments

In the paper *Evolving a generalized strategy for an action platformer video game framework* [8], which was included in the EvoMan framework, experiments with the same game are performed, while using the NEAT algorithm. Comparing the results for enemy **[7, 8]** for both solutions we can conclude that they are similar, because both of them win 4 times with a similar average gain. The same applies for the pair of **[2, 6]** enemies.

3.5 Comparison with self-adaptive mutation

Besides researching constant and adaptive mutation, we decided to also verify the effectiveness of a custom self-adaptive mutation. Our self-adaptive algorithm is built around the idea of taking the last weight of each individual as its sigma during mutation. It turned out that we were not able to get better results by using our custom self-adaptive mutation for our enemy groups, see *figure 3*. This is why this research switched to focus on the adaptive approach.

Figure 3. Box plot for the obtained gain for the three implemented EAs for the enemy set 2 and 6

3.6 T-test analysis

To determine if the observed differences in fitness mean during the testing of both algorithms is of statistical significance, a t-test was performed over the test data of both sets of enemies. For all tests the hypotheses were the following:

$$\boldsymbol{H0}: \textit{the true difference in means is equal to } 0$$
$$\boldsymbol{H1}: \textit{the true difference in means is not equal to } 0$$

The significance level used for each test is **0.05**. The results obtained from the t-tests are denoted in *table 3*.

Table 3: *P-values of t-tests*

Enemies	P-Value
[7,8]	0.6619
[2,6]	0.9527

4. CONCLUSIONS

As this research showed, it was not possible to implement an adaptive mutation operator that significantly outperforms the Gaussian mutation operator from DEAP. The training and testing of both algorithms appeared similar which may indicate that the choice of enemy sets to train on were not diverse enough for both algorithms to show their full capability and hence perform differently. Also, making the mutation operator adaptive may not have had as much impact as making another operator adaptive would have had. Considering the significance level of 0.05, the null hypothesis can indeed not be rejected for enemy either set of enemies, thus suggesting that it is not trivial that the means are significantly different. Hence, it cannot be concluded that one of the two algorithms performs significantly better than the other.

We have a few additional hypotheses regarding why no significant difference between the EAs was observed. First, we decided to run only 10 generations per experiment. According to the training plots there is a high chance that with the increase of generations we could achieve better results in terms of fitness as the lines still seemed to be increasing. Perhaps, a greater difference between the two EAs could then be observed. Furthermore, due to computational power and time limits, we applied the adaptive operation only in every 3 generations, which could result in less effectiveness of this method than if we performed it in every generation.

Overall, we conclude that with our choices of operators and in our setting, it has not been possible to significantly outperform the Gaussian mutation operator from DEAP. However, to verify if this conclusion could be generalized for adaptive parameter control or even parameter control operators in general, we advise further research regarding varying ways of parameter control. We also recommend training and testing the EAs on a wider range of different parameter settings, think about sets of enemies for example, and population or generation size. Finally, it cannot be excluded that the results obtained are due to either chance, inaccuracies, amount of data, or other factors.

5. WORK DIVISION

All team members participated in coding and later training the model and testing it. Rumyana Krumova implemented the crossover and the mutation functions. Dominique Woof performed the coding for visualizing the data, worked on elitism and self-adapted mutation. Melis Nur Verir implemented the survivor selection and elitism improvements. Alicja Dobrzeniecka implemented the parent selection and the local search. All members also participated in preparing the report by dividing it into sections.

6. REFERENCES

[1] Fathi, M., & Bevrani, H. (2019). *Optimization in Electrical Engineering* (1st ed. 2019 ed.). Springer Publishing.

[2] Aguston Eiben, Zbigniew Michalewicz, Marc Schoenauer, Jim Smith. Parameter Control in Evolutionary Algorithms. Fernando G. Lobo and Cláudio F. Lima and Zbigniew Michalewicz. *Parameter Setting in Evolutionary Algorithms*, 54 (54), Springer Verlag, pp.19-46, 2007, Studies in Computational Intelligence, 978-3-540-69431-1.10.1007/978-3-540-69432-8.inria-00140549

[3] Eiben and Smith(2015), Introduction to Evolutionary Computing

[4] DEAP documentation — DEAP 1.3.1 documentation. (2009). DEAP. https://deap.readthedocs.io/en/master/index.html

[5] Bera, S. (2021, 18 september). Crossover Operator — The Heart of Genetic Algorithm. Medium. https://medium.com/@samiranbera_66038/crossover-operator-the-heart-of-genetic-algorithm-6c0fdcb405c0

[6] Evolutionary Tools — DEAP 1.3.1 documentation. (2009b). DEAP. Evolutionary Tools — DEAP 1.3.1 documentation

[7] B.Suri and R.Goyal, On the Effectiveness of Using Elitist Genetic Algorithm in Mutation Testing Shweta Rani

[8] Evolving a generalized strategy for an action-platformer video game framework, https://github.com/karinemiras/EvoMan_framework/blob/master/multi_evolution.pdf