Evaluating variation operators for higher fitness values in a population - Recombination: 2-point Crossover vs 1-point Crossover

Evolutionary Computing Assignment I: Specialist Agent
Team 21

September 25, 2020

Wouter Kok
Vrije Universiteit Amsterdam
The Netherlands
2639768/w.j.kok@student.vu.nl

Israa Ahmed
Vrije Universiteit Amsterdam
The Netherlands
2639584/i2.ahmed@student.vu.nl

SarahLynne Palomo
Vrije Universiteit Amsterdam
The Netherlands
2636769/s.l.palomo@student.vu.nl

Fay Beening
Vrije Universiteit Amsterdam
The Netherlands
2680969/r.f.l.beening@student.vu.nl

1 ABSTRACT

In the process of developing genetic algorithms, defining the optimal parameter values for evolutionary algorithms (EA) presents a challenge for the algorithm creator. This is due to the high number of combinations of the different value permutations, making it impractical to test all of them. Furthermore, the choices for the parameter values are also dependent on the situation that the EA application will be applied to.

It is imperative that the correct choice of values is made as these choices can directly influence the outcome of whether or not the EA's resulting output achieves the desired goals within the problem constraints. Thus, in the scope of evolutionary computing (EC), we use action-platformer game framework *EvoMan* to examine one of these parameters - the variation operator: *n-crossover* [1].

In this paper, we seek to determine whether the use of 2-point crossover results in higher fitness values (max and mean) for a given population than the use of 1-point crossover, when evolving a population of game playing agents over a set number of generations. Our findings showed that our algorithm using 2-point crossover recombination evolved a population's mean fitness values faster than our 1-point crossover algorithm over 10 generations.

Keywords: DEAP, recombination operator, n-crossover, EvoMan

2 INTRODUCTION

Neuroevolution is a form of artificial intelligence that uses evolutionary algorithms (EA's)[2]. It could be applied to digital gaming, which has been around since 1939 [3]. At first users played against machines but nowadays it is more common to play against machines in the form of non-playable characters (NPC's). These non-playable characters can be driven by EAs, in which they are called agents. Agents evolve trough external input and aims to satisfy a fitness function[4, 5]. The challenge is in creating a general game playing agent which is not over-fitted on one particular task[1].

Creating a more diverse pool of genes in the EA population avoids over-fitting. Evolutionary algorithms use the biological method of genetic recombination to create this diversity, it recombines existing strings of digital genes from one entity to the other and vice versa. By crossing over fragments of digital DNA between pairs in the population, new "offspring" are created containing different sets of characteristics. The more DNA fragments that are crossed over, the more unique the offspring will be in terms of the parent population.

The **research goal** of this study is to compare the resulting populations when using a 2-point crossover versus a 1-point crossover for recombining individuals, in order to determine which of the two methods results in a higher mean and max fitness over a population within a set number of generations.

2.1 EvoMan - An Evolutionary Algorithm Test Platform

To investigate the effect of 1-point crossover versus 2-point crossover, the game playing framework *EvoMan* created by Karine da Silva Miras (2016) is used [1]. The *EvoMan* game is a platform level game based on the classic player-platform game MegaMan. The framework is composed of 1 player and 8 distinct enemies, each starting with a health of 100 points. In this 2-dimensional game the player and enemy start on opposite ends of the "field". Each enemy equals 1 level, with 8 levels in total. Both the player and the enemy, decrease in health whenever they are hit by the opposing players projectile. In addition, when the player and enemy come into contact, the player is configured to lose health. The level ends when one of the two players health reaches 0, while the game is over when all 8 enemies have been competed against. For simplicity of our experiment, the most basic three of the eight enemies are chosen to train the different agents, namely, *Flashman*, *Airman* and *Woodman*.

3 METHODS

To set up the experiment we started by designing our two EA's using baseline parameters to keep non-variation operator parameters as simple as possible. For this purpose, we began with the parameters used in the Simple Genetic Algorithm (SGA) which we then adapted to the scope of our experiment as seen in Table 1 [6].

Parameter	Value
Representation:	List of floats
Recombination:	n-point crossover (n={1|2})
Mutation:	Bit-flip
Parent selection:	Fitness proportional (Tournament)
Survivor selection:	Random

Table 1: EA technical tableau

Instead of implementing bit-strings, each individual in the population is represented by a list of 265 float values derived from a random uniform distribution between -1 and 1. These float-lists are used in evolutionary computing terms to symbolise the genotypes that we used to recombine between two parent individuals to create two offspring.

3.1 Implementation

We used python v.3.8.1 together with the DEAP framework v.1.3.1 [7] to utilise the EvoMan game framework to compete each individual in the population against a non-evolving enemy that follows a pre-programmed set of game action rules. The fitness of each individual was evaluated based on the performance against each enemy. If the individual is defeated, it is assigned a low fitness value that is calculated with a dependency on game duration. If the individual wins, it is assigned a high fitness value that is calculated from the remaining amount of health points (max 100), and the game duration. The fitness function [1] was defined as:

$$f = 0.9 * (100 - e) + 0.1 * p - logt \quad (1)$$

where p and e are the current health point level of the individual player and enemy, respectively. The duration of the fight is represented by 't' [1].

The experiment can be replicated by running *deap_specialist_runEAs.py* which runs both EAs and the different runs and prints the results in 2 maps: *deap_specialist_cxOnePoint* and *deap_specialist_cxTwoPoint*. To run the best solutions, run *deap_specialist_controller.py* which prints the results in the map *deap_specialist_run*.

3.2 Parent Selection

Once all of the individuals in the population had been evaluated and had a fitness value assigned, tournament selection of size 2 from the DEAP framework was used to randomly select pairs of individuals

sequentially from the population[8]. The individual with the higher fitness value "wins" and the "loser" becomes a clone of the winner. Subsequently, after all sequential pairs were compared, this modified population formed the mating pool to which recombination operators were applied to. During the experiment, this tournament process is run with each new population against each of the three enemies.

3.3 Recombination

The DEAP crossover function selects sequential pairs of individuals from a population. To minimize the chance of adjacent clones being selected as parents, the position of the individuals in the modified population were shuffled.

A pair of individuals were then selected sequentially from the mating pool as the parents of two children. For the first EA, we applied 1-point crossover where the float list of each parent is split at a single point that is selected at random to remove the positional bias. The 'tails' (float values right from the split) are then are exchanged between the parents while the 'heads' (float values left from the split) remain in place. The new configurations of the float lists become the children that are added to the updated population along with the unmodified configuration of their parents. This process is repeated with the selection of the subsequent pair of individuals in the mating pool until all of the members of the mating pool have been recombined.

An example of the 1-point crossover [9] can be illustrated by:

Parents	Children
030|**5512**	**030**|9378
494|9378	494|**5512**

In the case of the second EA, we apply 2-point crossover where the float list of each parent is split at two randomly selected points. Then the same process of creating children and updating the population as with the first EA is performed. Although in this case the fragment in the middle will be crossed.

An example of the 2-point crossover [9] can be illustrated by:

Parents	Children
03|**05**|**512**	**03**|49|**512**
49|49|378	49|**05**|378

3.4 Mutation

Following the crossover operations we then apply DEAP's bit-flip mutation to the offspring float lists with a random probability of $P_m = 0.1$. Once an individual has been chosen to be mutated the independent probability for each float value in the list to be flipped is $P_v = 0.2$.

3.5 Survival Selection

Lastly, DEAP's random selection function is applied as a survivor selection mechanism to reduce the updated population size of 100 (parents + children) back to the original size of 50. Note that during the parent selection stage we used tournament selection, which cloned the winner of the tournament, that quickly resulted in a homogenous population when we also combined this with survival selection of individuals with the highest fitness value. To solve the problem of early convergence to a local optimum due to elitism, random survival selection was applied to the population instead.

3.6 Iterations and Control Parameters

For each enemy an initialised population is run for 10 generations with the same control parameters on the two EA's. In table2 the chosen parameters are described. The mean and max fitness levels are saved per generation.

The individual with the highest fitness level over the 10 generations is saved and is then played against the 3 enemies 5 times in order to provide mean fitness values for the single enemy challenge. To compare the significance of the difference between the mean values of the two EA's with the corresponding enemy, a Welch Two Sample t-test was performed.

Parameter	Value
Population Size	50
Generation	10
Survival	50%
Crossover Prob	100%
Mutation Prob	10%

Table 2: Parameter values applying to both EA's

4 RESULTS AND DISCUSSION

Playing each population of all 10 generations against each of the 3 enemies (10 times) we get Figure 1, Figure 2 and Figure 3, that shows the different rates of evolution where either the 1- and 2-point crossover has been applied

By inspecting the mean fitness of the population (red lines) some distinct behaviours from both of the algorithms can be observed. Firstly, the mean fitness of the population increases faster over the generations where 2-point crossover has been used compared to the 1-point crossover, in all 3 enemy cases. Secondly, upon closer inspection it can also be observed that the standard deviation on the 2-point crossover is consistently smaller than for the 1-point crossover, indicating a less diverse 'gene' pool. This could be in part due to the multiple crossover points improving the probability that the next round of offspring receives a part of the successful genes from their parents than they would when 1-point crossover is used.

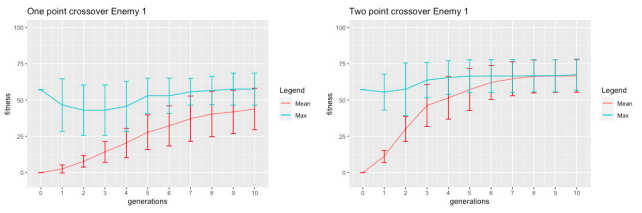

Figure 1: Evolution of the Population against Enemy 1 where 1-point crossover (left) and 2-point crossover (right) have been used

Thirdly, the max fitness level (blue lines) in Figure 1 and Figure 3 shows that although it is possible to lose an individual with higher fitness from the population in earlier generations because of the random survival selection function, we see that the overall max fitness does increase over the generations until it plateaus to the maximum value for that population during the set of 10 runs.

Fourthly, the max fitness (blue lines) in Figure 1, 2 and Figure 3 are at relatively similar levels between the 1-point and 2-point

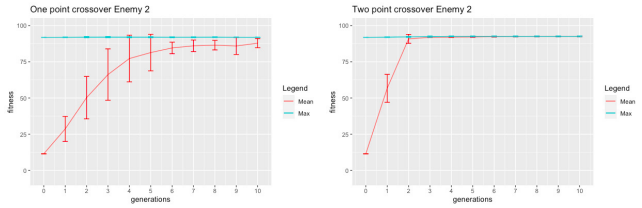

Figure 2: Evolution of the Population against Enemy 2 where 1-point crossover (left) and 2-point crossover (right) have been used

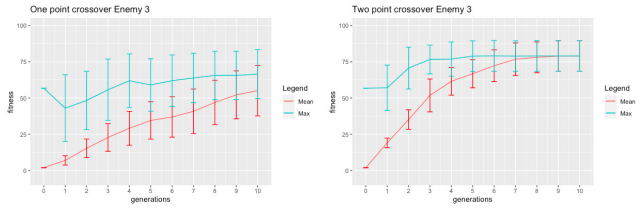

Figure 3: Evolution of the Population against Enemy 3 where 1-point crossover (left) and 2-point crossover (right) have been used

algorithms per enemy. Despite this, in Figure 1 and Figure 3 the max fitness for 2-point algorithm still shows a slightly smaller step change (i.e. 54 to 52) than the 1-point algorithm (i.e. 55 to 45), which again suggests that using 2-point crossover results in less novelty in a population's gene pool.

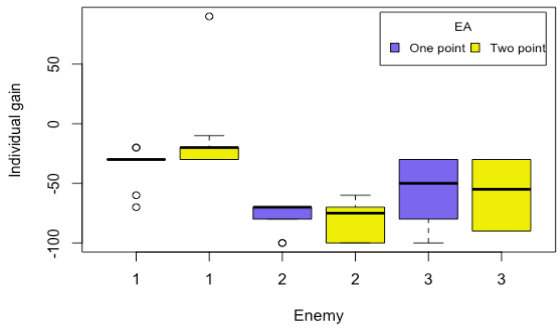

Figure 4: Variability of the Individual Gain Values Over All Experiments

The differences in the variability of the data between the two algorithms can also be observed in the boxplots of the Individual Gain in Figure 4. In this context, Individual Gain represents the performance of the individual (player health - enemy health).

In the boxplot, the 2-point algorithm results in more variability in individual gain in the population while the 1-point algorithm results show more density of the spread with the exception of a small number of outliers. Because we limited the number of runs to 10 against each enemy we see that the results are too few to make conclusive statements regarding the Individual Gain. However, what we can observe is that the means for both algorithms are relatively equal.

Comparing both EA's with the corresponding enemy using a Welch t-test resulted in the p-values: 0.07848, 0.5296 and 1, for enemy 1, 2 and 3, respectively.

4.1 Comparisons to Miras (2016) Paper

Interestingly, when we compared the results of our 1-point crossover final scores with the corresponding scores from the first experiment of Miras'(2016)[1] Genetic Algorithm, our final scores showed remarkably similar corresponding values to Miras' GA in Table 3 with the exception of *FlashMan*, which did not score as well. Note that Miras used real-valued vectors for the member representation, and a population of 100 run over 100 generations (we used float list representation and Pop=50, Gen=10).

Enemy	GA by Miras	1-point EA
FlashMan	90	57
AirMan	90	91
WoodMan	58	50

Table 3: Comparison of the Final Scores for the Single Enemy Challenge in EvoMan between the GA from the Miras(2016) paper and the 1-point EA

Another comparison of note is that Enemy 2 / Airman seems to be an easier opponent to defeat, as is seen by the fast evolution pace of the fitness values in both of the algorithms (n=1/2). This was also observed in Miras' baseline paper.

4.2 Lessons Learned during Testing

Our initial EA design choice was to use elitism in both the parent selection (i.e. tournament) *and* the population survival mechanism (i.e. keep best 50%, discard worst 50%). This resulted in early convergence of the population, therefore we decided to use a random survival of 50% of the population. Changing one of these parameters to random selection resulted in more variance in the fitness values of both EA's.

Additionally, the population size 100 was changed to 50, because the EvoMan framework is based on pygame and due to the nature of its design the graphics could not be disabled, resulting in computationally long processing times. For further study we would use a larger population number to ensure higher accuracy in the results.

5 CONCLUSIONS

This study researched the effect 2-point crossover recombination has, compared to a 1-point crossover, on the overall mean and max fitness of a population over 10 generations. Our results show that using 2-point crossover for parent recombination is more favourable than using 1-point crossover in a problem situation that requires **faster evolution** over a number of generations. However, using 2-point crossover did not influence the highest fitness value that could be found in a population so in this respect it had no apparent advantage over using 1-point crossover. Furthermore, the choice of the number of crossovers used does not appear to affect the overall mean and max fitness of the population since our early results suggest that the evolution of the max fitness value for a given population tends to plateau after the first few generations.

This opens the possibility for further study to investigate the effect of using higher numbers of crossover points during recombination and, more specifically, to determine at what number of n-crossovers is optimal for achieving the fastest possible evolution of a population's fitness before convergence of the mean and max fitness values has been reached.

REFERENCES

[1] Karine Miras and Fabricio De França. 2016. Evolving a generalized strategy for an action-platformer video game framework. In (July 2016), 1303–1310. DOI: 10.1109/CEC.2016.7743938.

[2] Julian Togelius, S. Karakovskiy, Jan Koutnik, and Jurgen Schmidhuber. 2009. Super mario evolution. In (October 2009), 156 –161. DOI: 10.1109/CIG.2009.528648.

[3] Lisa Rougetet. 2016. Machines designed to play nim games. teaching supports for mathematics, algorithmics and computer science (1940 – 1970). *History and Pedagogy of Mathematics*, (July 2016).

[4] Kenneth Stanley, Bobby Bryant, and Risto Miikkulainen. 2006. Real-time neuroevolution in the nero video game. *Evolutionary Computation, IEEE Transactions on*, 9, (January 2006), 653–668. DOI: 10.1109/TEVC.2005.856210.

[5] Ian Dunwell, Sara de Freitas, and Steve Jarvis. 2011. Four-dimensional consideration of feedback in serious games. *Digital Games and Learning*, (January 2011), 42–62.

[6] John H. Holland. 1992. *Adaption in natural and artificial systems*. Volume 1.

[7] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13, 2171–2175.

[8] A.E. Eiben and J.E. Smith. 2015. *Introduction to Evolutionary Computing*. Volume 2, 84–85.

[9] A.E. Eiben and J.E. Smith. 2015. *Introduction to Evolutionary Computing*. Volume 2, 49–54.