SET10107 Computational Intelligence

40484293

ABSTRACT

This study focuses on the analysis and implementation of an Evolutionary Algorithm (EA) to improve the weights of a neural network to help navigate the landing of a spacecraft. The provided code serves as a skeleton of the EA that requires implementation of any missing evolutionary operators or the potential development of an alternative search algorithm.

This report presents the detailing of the implementation process and the relevant research that has been undertaken. The evaluation is conducted by completing multiple runs to find both training and testing fitness sets, showing the effectiveness of the proposed approach at that time.

1. INTRODUCTION

Provided is a Java-based framework that encompasses a spacecraft simulation that has a fitness function for evaluation landing performance, a neural network controller and a partially implemented EA. It requires the implementation of the missing evolutionary operators such as Selection, Reproduction, and Replacement, and to evaluate the performance of the newly changed algorithm on both training and test sets by conducting a thorough analysis that shows the effectiveness of the approach.

Furthermore, the exploration of the various parameters such as activation functions, the number of hidden nodes, and the mutation rate with the goal to develop the algorithm with the ultimate objective to try and land the spacecrafts.

In the following report the implementation and experimental testing process is discussed with the addition to discussing the results and shedding light on the efficiency and potential areas for future improvement.

2. APPROACH

Before designing a solution to help experiment and analyse the current skeleton EA, I researched various types of selection and crossover methods commonly used for optimisation before implementing. The book Introduction to Evolutionary Computing by A.E. Eiben and J.E. Smith (Eiben & Smith, 2015) was most of the background reading to gain more knowledge and understanding into the possible selection and crossover methods that could be implemented.

2.1 Reproduction

The first operator that was modified was that of Reproduction. This operator is responsible for taking the parents individuals that will be chosen from the selection methods and alter them by manipulating their chromosomes and creating new offspring for the population.

Four different crossover operators where chosen, One-Point Crossover, Two-Point Crossover, Uniform Crossover and Arithmetic Crossover.

One-Point Crossover works by choosing a number randomly in the range and then splitting the two parents at that random point selected, exchanging the tails and creating the two children. The benefits of One-Point Crossover are its simplicity to not only implement but to understand its workings. It ensures that genetic material from both parents is passed to the children and therefore helps maintain diversity within the population. It has the balance between exploitation and exploration by exploiting the genetic material of the parent chromosomes but also exploring the new combinations of segments from the different parts of the chromosomes. Overall, it has a good balance between simplicity, effectiveness and efficiency making it a reliable and popular choice.

Two-Point Crossover is the adaption of One-Point Crossover. It means that the chromosome is broken into two or more segments instead of the one in One-Point Crossover. This operator allows for a greater exploration of the search space in comparison to One-Point. It introduces more variability in the exchange between the parent chromosomes allowing for more potential in a wider range of offspring. Like One-Point this operator has that same balance between exploitation and exploration. Overall, it enhances diversity in the population and provides a reliable method for diversifying the offspring.

Uniform Crossover in contrast to the previous two operators works by treating each gene independently. With this it then makes a random choice as to what genes should be inherited from each parent. In each position, if the value is below a certain parameter, usually 0.5 that gene is inherited from the first parent, and above is inherited from the second parent (Eiben & Smith, 2015). This operator helps enhance the exploration by allowing genetic material from both parents to be equally represented in the offspring which in turn helps in searching through the search space, leading to potentially discovering more promising solutions within the algorithm. Which in turn reduces bias towards specific parent chromosomes. With this method it maintains diversity within the population. It is also seen as easy to implement in comparison to other complex crossover methods. Overall, it can promote exploration in the search space for potential optimal solutions and, reduce bias in specific parents makes it a good option for implementation.

Arithmetic Crossover blends material from both parents using a weighted average, this ensures that the information from both parents is kept in the offspring produced, maintaining the population diversity. Unlike the previous crossover methods this method results in a smooth transition between the parents instead of discrete swapping of genes. This allows for a more exploration across the search space creating a more gradual and efficient search for the optimal solutions while exploiting the information present in the parent chromosomes. Overall, its ability to create smooth transitions and preserve information can make it an essential tool for optimising the potential results.

2.2 Selection

The second operator that was modified was that of the Selection Methods. The purpose of these methods is to begin the reproduction of the population by selecting the parent individuals that later reproduce the offspring of the next generation. These methods use a variety of different approaches to try and select the best possible individuals to try and start the process of achieving optimal results over time.

For the selection process four different operators were selected for implementation, Random Selection, Tournament Selection, Roulette Wheel Selection, and Rank-Based Selection.

Random Selection chooses parents randomly from the population without taking into consideration the fitness. This allows for equal chances that every individual has in the population to be chosen. Once selected the parents are then paired randomly to then create children based on what crossover operator is being used by the algorithm. Although random selection is good at promoting exploration by pairing random individuals ensures that the offspring can be diverse ensuring new and potentially promising solutions is does not consider the exploitation of the individuals in the population as good. As it does not consider the fitness of the individuals as a result it may fail to exploit the promising solutions possible within the population. Overall, its valuable at exploration, is considered simplistic and efficient to implement and avoids bias making it a good baseline for creating reasonably suitable offspring.

Tournament Selection selects its individuals by randomly selecting a subset of individuals within the population and then choosing the individual with the best fitness. Its then repeated for every pair in the population ensuring only the strongest is chosen to create offspring. Tournament selection promotes exploitation by picking the individual with the best fitness concentrating its search around promising results. However, although its focuses more on exploitation it still considers exploration by ensuring that individuals from different areas in the search space have the potential to participate in the reproduction of offspring, preventing premature convergence. Overall, tournament selection is a popular and effective operator for selecting individuals within a population. With being able to adapt its tournament size it further helps the exploitation and exploitation of the selection method ensuring that the population remains diverse and aids to the finding of optimal results.

Roulette Wheel Selection bases its selection on the individual's fitness. It works by calculating the fitness proportion of every individual in the population and creating a roulette wheel where all individuals in the population are represented by a segment in the wheel whose length is proportional to its fitness proportion. The wheel is then spun and a random number between 0 and 1 is selected, the individual whose segment contains the number chosen is then the mating partner. In a more simplified way, the higher the fitness of the individual the higher the probability of it getting chosen while it still being randomly selected. This operator is seen as relatively simple to implement as it involves basic arithmetic operations while also remaining efficient in its selection process. Roulette Wheel focuses more on the exploitation of the population more than exploration as individuals with the best fitness are at a higher probability of being chosen meaning there is more of a probability of finding optimal results through this exploitation, although it does allow for some degree of randomness in its selection method so does allow for exploration with the search space. Overall, this method is efficient, has the advantage of fitness-based selection but also has its potential disadvantages of population diversity with its low exploration of the search space.

Rank-Based Selection operates by ranking the individuals in the population based on their fitness. Selection probabilities are then assigned to each individual in the rank, typically them with higher ranks are assigned higher probabilities. This helps ensure that the individuals with the better fitness have a greater chance of being selected to reproduce. This exploits the search space by selecting the best fitness individual but also promotes exploration by allowing a chance for worse fitness individuals to be selected as there is still a chance a lower probability could be selected. Overall, this operator is effective in selecting appropriate individuals from the population with its advantages of fitness-based selection and efficiency.

2.3 Replacement

Replacement operators where the last to be changed in the algorithm. Once the children of the parents are created using the Selection and Reproduction methods the Replacement operator puts the newly created offspring back into the population so the process can continue with constantly improving the fitness of the individuals overall.

For the replacement operators three different operators chosen for implementation are, Elitist Replacement, Steady State Replacement, and Generational Replacement.

Elitist Replacement is known as this as it replaces the worst individual in the population with the newly generated offspring. It identifies the worst individual in the population by their fitness, once identified there are then replaced by the offspring that was been generated through crossover and mutation as the offspring are expected to inherit the beneficial traits from the selected parents. Overall, this operator helps ensure that the best individuals are kept within the population preventing the possible loss of potential solutions and continues the progress towards better outcomes.

Steady State Replacement works by only replacing a subset of the population per generation. This is instead of generating a whole new population, maintaining diversity and creates more exploration. Steady State Replacement involves evaluating a subset of the population per generation so overall reduces computational overhead. Overall, it offers a balanced approach to the replacement of the population while combining continuous evolution and efficiency.

Generational Replacement ensures each generation starts fresh as it replaces each generation with an entirely new generation. With replacing the entire generation, it encourages exploration by

being able to explore new areas of the search space while also exploiting promising solutions. Overall, this is a widely used operator that has simplicity and ability to maintain population diversity.

2.4 Evolutionary Algorithm and Neural Network

The Evolutionary Algorithm implemented to train the Neural Network is a basic algorithm that includes the initialisation of a population, selection methods for the parents, crossover for creating the children, mutation, evaluation, and replacement of the individuals in the population.

The selection method is one of the four previously discussed and the crossover will also be one of the four previously discussed. The mutation is then applied to the children of the population after they are created in the crossover operators, this introduces diversity into the population as a higher mutation rate determines the likelihood of mutation happening leading to more exploration and diversity within the population whereas a lower rate creates a more gradual change. The mutation change however determines the level of of mutation that will be made, higher the change the more significant the change made to the solution, again adding diversity. However a lower mutation change would encourage more subtle changes adding to the exploitation of current solutions. The mutation randomly modifies the weights of individuals' chromosomes. The fitness of each individual is evaluated using a fitness function. This function measures the performance of the Neural Network. Finally, the algorithm will use replacement operators previously discussed to replace current individuals in the population with the offspring that has been created. This EA is providing a flexible framework for the exploration into optimising the weights of the Neural Network.

Within the Evolutionary Algorithm the Neural Network parameters mainly used are that of the hidden nodes and the activation function. Hidden nodes perform a weighted sum of the input values then followed by the activation function allowing the Neural Network to understand important relationships and patterns within the data. Overall, hidden nodes in the Neural Network allow for the network to learn, make predictions, and generalise based on the data input by extracting relevant features. Adjusting the number of hidden nodes is incredibly important aspect of the design to help tailor the network to specific tasks.

Activation Functions are important as they introduce nonlinearity that allows the Neural Network to model complex relationships in the data. Common activation functions include that of Hyperbolic Tangent (tanh), Rectified Linear Unit (ReLU), Sigmoid, and also Linear. They determined the output of a neuron in the network based on its input, impacting how good the network can be at recognising patterns in the data and how the network behaves.

Overall, the purpose of the algorithm is to land a fleet of spacecrafts. In order to try and find the optimal results and try and land the spacecrafts, varying options of the operators and parameters were tested and explored.

3. EXPERIMENTS & ANALYSIS

The aim of this Evolutionary Algorithm is to learn a Neural Network to safely eight spacecrafts, with the optimal possible result being as close to the fitness of zero. To being the experiments of the Evolutionary Algorithm the approach of assessing a set of operators and carrying the lowest fitness forward was the experimental testing that was conducted.

Each of the previously discussed operators and parameters where all tested individually totaling to nine tests overall with the addition of the final run of the algorithm. Each test was running a total of ten times per operator or parameter to gain the average fitness to determine the lowest fitness of which was the operator or parameter, which was then changed and carried forward to then conduct the next set of tests. On each run of the algorithm, it ran for 20,000 evaluations presenting the best fitness of both the test and training set at the end. This approach allowed the narrowing of optimal results as the fitness continued to decrease as the tests were conducted.

To illustrate the experiment findings and discuss the analysis the mean fitness of each test both training and test fitness will be illustrated on a chart.

The first set of tests that were conducted was that of the **Activation Function**. Linear, Tanh, Rectified Linear Unit (ReLU), and Sigmoid were all evaluated by running a total of ten times each to give the mean fitness it produced. Activation Function was chosen first as it can affect the speed of an EA, meaning that the best option for not only optimal results but at the best possible speed could be identified. It showed the results that linear was the best option having a mean training fitness of 0.109 and mean test fitness of 0.211.

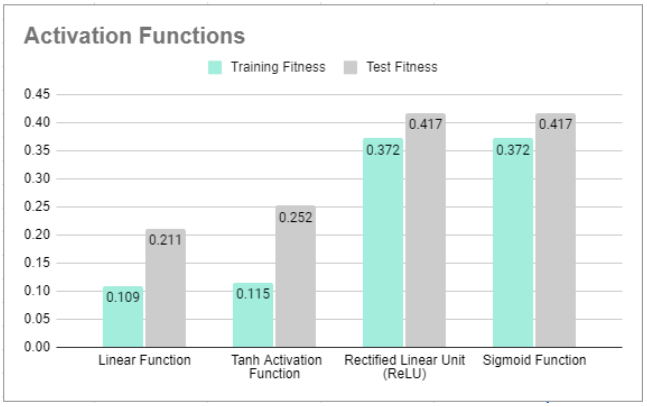

Figure 1 - Activation Function Results

The second test conducted was on the **Reproduction Methods**. The given EA skeleton had already implemented the selection method of Random Selection so justified my choice to assess the Reproduction Methods to find the best mean fitness. The reproduction methods chosen were that of One-Point Crossover, Two-Point Crossover, Uniform Crossover, and Arithmetic Crossover. As shown in the results Uniform Crossover had the lowest mean fitness with 0.083 training and 0.194 in test.

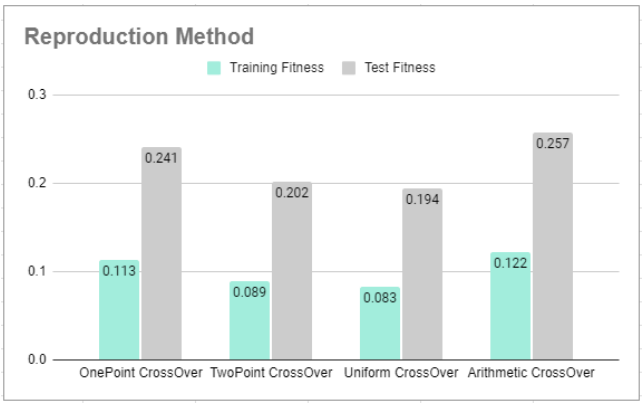

Figure 2 - Reproduction Method Results

The third test conducted was then **Selection Method** as the Reproduction Method was decided an investigation into a better selection method other than random must be completed. The selection methods assessed where that of Random, Tournament, Roulette Wheel, and Rank-Based. The results showed tournament to have the lowest mean fitness with 0.072 as its training and 0.186 as its test fitness.

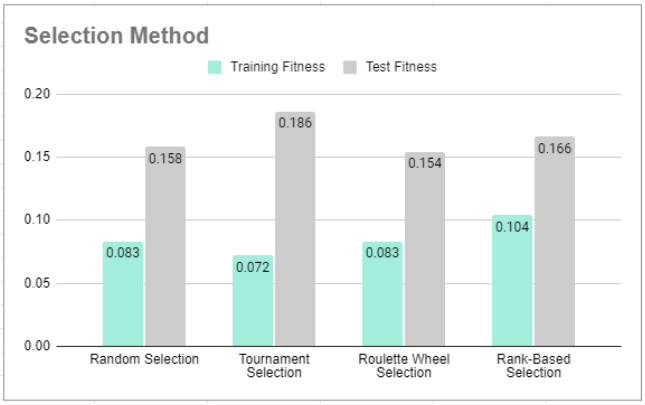

Figure 3 - Selection Method Results

As the selection method chosen was that of tournament the fourth test conduced was then the **Tournament Size** of that method. Tournament size can influence the diversity of the population and help aid further with exploitation and exploration. The tournament sizes of 3, 5, 10, and 15 were decided on as the lowest you can go is two, so I felt one up from the lowest while five is the more reasonable choice of which I then just continued to increase by five. The results showed that tournament size three was the best feasible option coming out with a mean fitness of 0.055 in the training set and 0.138 in its test set.

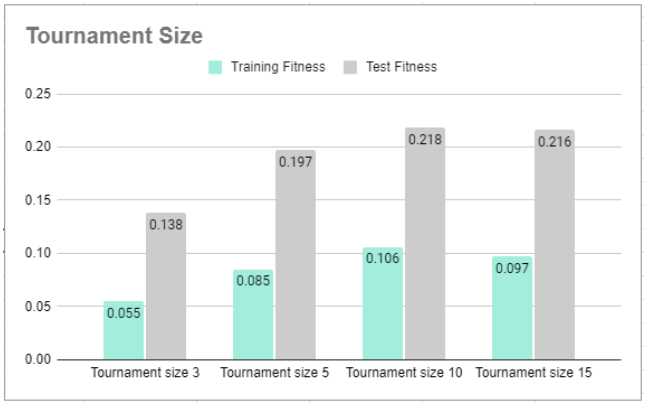

Figure 4 - Tournament Size Results

Now that the two main methods have been decided on and the skeleton EA has already the Elitist Replacement operator changing parameters to fine tune the algorithm was the next set of tests. Test five was the changing of the **Mutation Rate**. The original mutation rate was 0.01 and I felt increasing the rang more towards 0.1 could possibility be interesting. With that said the rates decided on where that of 0.01, 0.45, 0.75 and 0.1. The outcomes of these results were that 0.45 showed to be the best option with the lowest mean fitness at 0.047 in its training set and 0.143 in its test set.

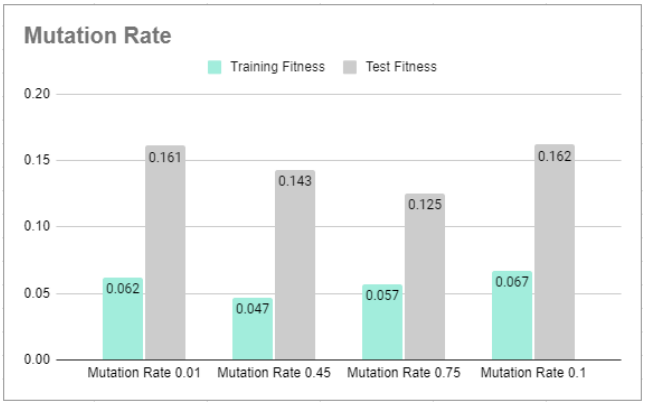

Figure 5 - Mutation Rate Results

Once the Mutation Rate was changed the next test, test six, was to explore the **Mutation Change.** The original parameter was that of 0.1 and I felt trying to increase in somewhat of increments of five could prove promising, meaning the Mutation Change tests consisted of 0.1, 0.5, 1.0, and 1.5. The parameter that gave the best results based on the mean fitness was that of 0.5 with a mean fitness of 0.03 in its training set and 0.054 in its testing set.

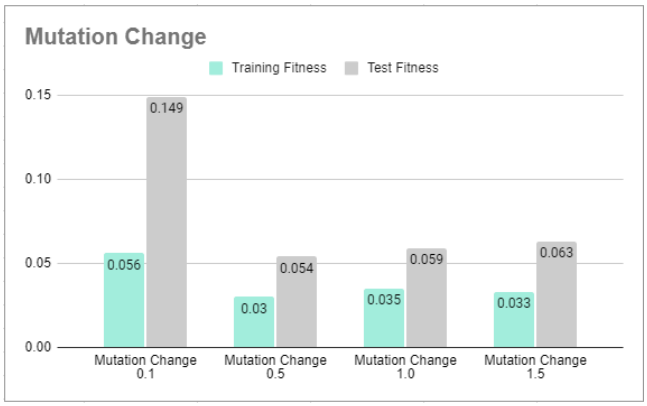

Figure 6 - Mutation Change Results

Test seven was to investigate the **Population Size**. The original population size was set to 40, I felt increasing this to 50 and incrementing it by 50 each time could show a range of results from small to large population sizes, meaning 50, 100, 150, and 200 were the population sizes that were tested in the algorithm. The results showed that the population size of 50 was the best with a mean fitness score of 0.016 in training and 0.033 in test, although this showed an increase in the previous training fitness score from 0.03 it wasn't a drastic difference so still pointed towards optimal results for the algorithm.

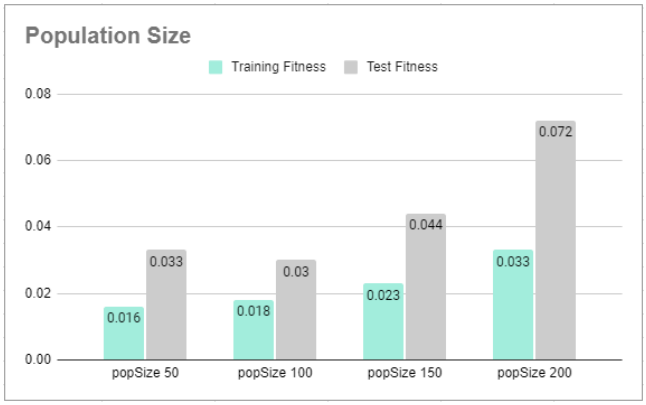

Figure 7 - Population Size Results

With the EA parameters accessed I felt trying to explore the Neural Network parameters of **Hidden Nodes** could further enhance the algorithm to potentially obtain optimal results. The number of hidden nodes that were tested was that of 3, 5, 10 and 20. I felt keeping the nodes small as they were set to 5 the whole duration of testing could benefit the algorithm however exploring the potential to a more larger number felt necessary to determine the best possible outcome. The results did show that five was the best option but with ten very closely behind with results showing 0.015 for the hidden nodes of 5 and 0.016 for the hidden nodes of 10.

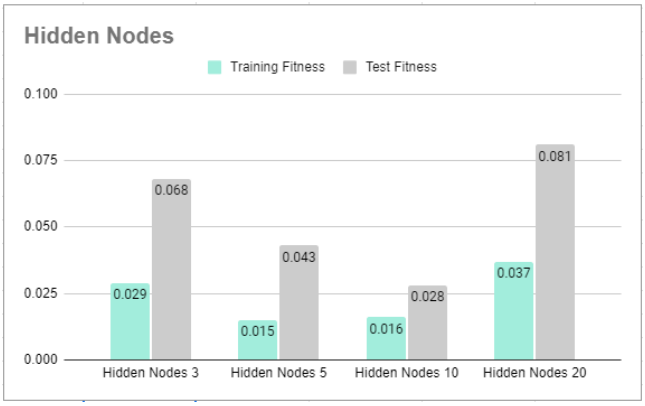

Figure 8 - Hidden Nodes Results

The final test was that of the **Replacement Method**. The methods assessed were, Elitist, Steady State, and Generational. This was tested at the end as the Elitist method was already implemented and can work as a very good option for replacement within an EA. The results showed that Elitist did in fact have the lowest mean fitness at 0.02 in the training set and 0.053 in its test set, however Steady State was close behind with 0.021 as its mean training fitness.

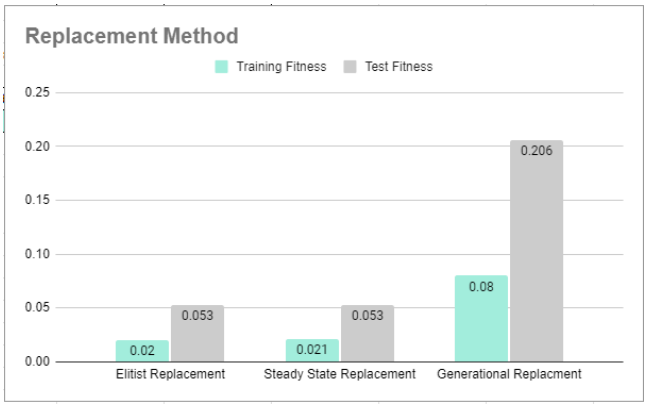

Figure 9 - Replacement Method Results

With all tests completed the experimental method of carrying the lowest fitness forward proved to be successful. This approach allowed the neural network to continue to learn, gaining more optimal results as the testing went on as only one parameter or operator was being changed at a time allowing for the best chance of fine tuning the algorithm for optimum fitness results. Although the second last test was producing the fitness result of 0.015 and the last test was producing a slightly higher fitness of 0.02 it was still a successful result as the optimal fitness is zero. With this fluctuation its justified as it works on average, which is always changing regardless to the number of times the algorithm is run the results could still be different each time in a small fluctuation, which is expected with this method of testing.

4. CONCLUSIONS

In conclusion, the final parameters and operators that produced the best results are displayed in Figure 10. These final parameters and operators produced the best results with a training fitness of 0.015 and a test fitness of 0.043 as shown in Figure 11. These results are gathered from running the algorithm a total of ten times to gain the overall mean fitness score of the Evolutionary Algorithm.

Activation Function	Linear Function
Reproduction Method	Uniform Crossover
Selection Method	Tournament Selection
Tournament Size	3
Mutation Rate	0.45
Mutation Change	0.5
Population Size	50
HIdden Nodes	5
Replacement Function	Elitist Replacement

Figure 10 - Final Parameters and Operators

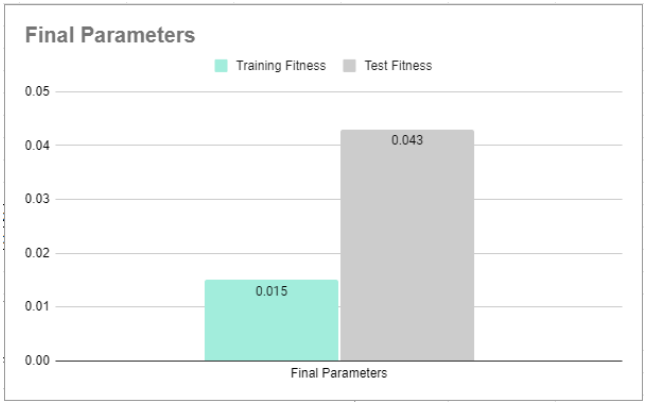

Figure 11 - Final Parameters Fitness Results

With this testing complete the aim of the algorithm to efficiently land a fleet of spacecrafts safety no matter the starting velocity or starting position can be considered a success as the optimal fitness is zero, the closer the algorithm is to zero the better it is at achieving its goal overall as it becomes a better functioning algorithm.

Although the fitness can technically be lower as its not at zero, getting a fitness of zero is a rare occurrence as the goal is more towards finding the best quality solution rather than trying to reach the absolute global optimum.

5. FUTURE WORK

With all testing complete and gaining the results presented the idea of future work could be starting from the beginning and trying a different approach in the sense of the order the parameters and operators have been evaluated in to try and possible see if solutions could be found in less tests. The possibility of also trying other operators in the place of the ones evaluated to see if they also produce more optimal results in a shorter time would be interesting to explore.

6. REFERENCES

Eiben, A. E., & Smith, J. E. (2015). Introduction to Evolutionary Computing (2nd ed.). Natural Computing Series. Springer.