SET10107 Computational Intelligence Coursework

40284966 Gabriele Balysevaite

ABSTRACT

The coursework requires to evolve weights for a Multi-Layer Perceptron artificial neural network to control the landing of fleet of identical spacecrafts. To achieve this an Evolutionary algorithm (EA) was implemented along with multiple operators applied to find the best suited method for this problem. Further parameter tuning was performed to find the most successful combination of operators and parameters to safely land the spacecraft.

ACM Reference format:
40284966 Gabriele Balysevaite. 2020. SET10107 Computational Intelligence Coursework. In *Proceedings of Coursework, Edinburgh Napier University, April 2020 (SET10107),* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Evolutionary algorithms (EA) are employed to search for optimal solutions by evolving a population set. The premise of evolutionary algorithm is similar to the process of natural selection, the main steps are: initialization, selection, genetic operators (mutation, crossover and replacement selection), and termination. This report documents experiments on chosen selection, mutation, crossover and replacement operators with intention to find the combination of operators that lead to the best fitness value of the trained neural network. Further parameter tuning is attempted, but due to time constrains only tournament size and population size were investigated.

2 APPROACH

2.1 The Steady-State Genetic Algorithm

The Steady-State (or incremental) Genetic Algorithm was chosen to solve this problem. In Steady-State GA population is updated couple members at the time rather than all at once. One of the most important advantage is that Steady-State GA uses half the memory of a traditional genetic algorithm because there is only one population at a time [8]. Moreover, An important feature of not creating more offspring than the current population size at each generation is that the generational computational time is reduced, most dramatically in the case of the steady-state GA [4]. The algorithm works as in Figure 1 [6]:

2.2 Operators

There are multiple ways each for initialisation, selection, recombination (or reproduction), mutation and replacement to be performed.

```
BEGIN
  $INITIALISE$ $population$ with random candidate solutions;
  $EVALUATE$ each candidate;
  REPEAT UNTIL ( $TERMINATION$ $CONDITION$ is satisfied ) DO
    1 $SELECT$ parents;
    2 $RECOMBINE$ pairs of parents;
    3 $MUTATE$ the resulting offspring;
    4 $EVALUATE$ new candidates;
    5 $SELECT$ individuals for the next generation;
  OD
END
```

Figure 1: Pseudo code for general EA

It is important to note that the recombination and mutation operators working on candidates must match the given representation [6], whereas selection requires only fitness of the individual, therefore selection operator is independent from representation. It is important to mention, that there are alternative ways to initialise population, such as opposition-based initialisation [12], but it was decided that random initialisation will suffice for this coursework.

2.2.1 *Selection.*

The first operation in reproductive phase of the algorithm is selection. Its purpose is to select parents from the population which will be used to reproduce children for further solution exploration. Following selection methods were implemented and used:

Tournament selection In tournament selection t number of random individuals are selected from population. Then the fittest individual from the tournament is selected to be the parent. The advantages of this selection method are that it is extremely simple and easy to implement, and that it is tunable: by setting the tournament size t, you can change how selective the technique is. [8] As seen later, tournament size is one of the parameters explored in testing part.

Roulette wheel selection Or otherwise known as Fitness proportionate selection, where individuals are selected in proportion to their fitness - if an individual has a higher fitness, it's selected more often.[8] The fitness of individual is used to compute a probability of selection for said individual. The formula in Figure 2 is followed, where *fi* is fitness of the individual and divisor is the sum of all individual fitness values in the population. To select an individual a random number is generated in the interval between 0 and sum of all fitness values and individual whose value CONTINUE

$$p_i = \frac{f_i}{\Sigma_{j=1}^{N} f_j}$$

Figure 2: Individual probability formula for roulette selection

Elitist selection This simple selection method chooses the best individuals in the population as parents. This allows to ensure the most fitting individuals are allowed to copy their traits to the next generation. [5] Additionally, such a strategy can add selective pressure and improve convergence speed.[1]

2.2.2 Reproduction. Or else known as recombination. The basic operator for producing new chromosomes in the GA is that of crossover. Crossover is an operation which, in this case, takes in two parents and produces two children, which both have some parts of both parent's genetic material.
One-point crossover For one-point crossover one cut point is chosen at random. Until the cut point the child is assigned chromosome gene values from one parent, after cut point gene values are taken from the second parent.
Two-point crossover In two-point crossover two cut points are randomly selected. Genes at index in between those cut points are assigned values from one parent, while genes before first and after the second cut point are assigned values from another parent. Figure 3 below illustrates the two-point crossover.

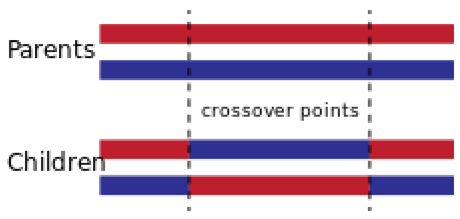

Figure 3: Two-point crossover

Uniform crossover In uniform crossover each gene is selected randomly from one of the corresponding genes of the parent chromosomes.

Simple arithmetic crossover For this type of crossover a single cut point is chosen randomly. The first child gene values from the cut point are calculated by the formula in Fig. 5, where α is a variable multiplier in range from 0-1, k is the cut point, x and y values of parents gene. The second child gene values are calculated by reversing place of α.

$$\left\langle x_1, \ldots, x_k, \alpha \cdot y_{k+1} + (1-\alpha) \cdot x_{k+1}, \ldots, \alpha \cdot y_n + (1-\alpha) \cdot x_n \right\rangle$$

Figure 4: Formula to calculate first child gene values in arithmetic crossover

2.2.3 Mutation. Mutation is a process, where values of one or more genes are altered. Normally it takes place after crossover to produce new children, so it creates new adaptive solutions good avoid local optima.[8]. The role of mutation is often seen as providing a guarantee that the probability of searching any given string will never be zero and acting as a safety net to recover good genetic material that may be lost through the action of selection and crossover. [3]

Inversion mutation In order to perform inversion, a string of genes is selected and their positions are inversed.
Swap mutation Swap mutation randomly swaps positions of two genes.
Creep mutation In creep mutation, a random gene is selected and its value is changed with a random value between lower and upper bound.[11]
Standard mutation In the coursework code provided mutation iterates through the chromosome, then adds or subtracts to gene value by the provided delta change value depending on random Boolean parameter.

2.2.4 Replacement. Replacement strategy helps to find out the individuals that would replace the current generation to form next generation of population. **Replace Worst** Was provided with the coursework code and used in testing. Replace worst algorithm the current least fit member of the population is replaced.
Restricted tournament selection Also known as kill tournament, is similar to parent selection tournament. For replacement tournament you choose a number of parents at random and replace the worst [10]. In this implementation, both selection and replacement tournaments are the same size.
Deterministic crowding In deterministic crowding replacement each parent and each child competes, then each child replaces the nearest parent if it has a higher fitness.[7] In this implementation, the distance is calculated with helper function *getDistance*, which adds the absolute distances between all of individuals genes. Figure 4 shows pseudo-code for the deterministic crowding implementation.

```
If $(d(P_i, O_i) + d(P_j, O_j)) \leq (d(P_i, O_j) + d(P_j, O_i))$ then
    If $f(O_i)$ is better than $f(P_i)$ then replace $P_i$ with $O_i$.
    If $f(O_j)$ is better than $f(P_j)$ then replace $P_j$ with $O_j$.
Else
    If $f(O_i)$ is better than $f(P_j)$ then replace $P_j$ with $O_i$.
    If $f(O_j)$ is better than $f(P_i)$ then replace $P_i$ with $O_j$.
```

Figure 5: Pseudo-code for deterministic crowding

Elitist replacement With this method the best two of the four individuals stay or are added to the population.

2.2.5 Neural network parameters. At the moment, parameter tuning and parameter control are the two major forms of setting neural network parameter values. In tuning, the values are set before the run and remain fixed during it. Alternatively, parameter control, starts the run with initial values which are changed during the run.

Tournament size population size were tested and the mean results are explained in the Results section.
Min and max gene values Minimum and maximum gene values were normalised, because according to V. Alto it can make convergence rate slower.[2]

3 RESULTS

Parameter tuning was done where selection, crossover, mutation and replacement were selected by running a combination of each methods for 10000 evaluations on training data set. Top 10 were selected and run again with 20000 evaluation. Table 1 shows the best performing combinations in training and test runs. This method of selecting operations for selection, reproduction, mutation and replacement is extremely limited, but was chosen due to time constrains.

The starting parameters were chosen as follows:

- Selection - Tournament
- Reproduction - Two point
- Replacement - Restricted Tournament selection
- Mutation - Swap
- tournamentSize - 20
- numHidden - 5
- minGene - -1.0
- maxGene - 1.0
- popSize - 40
- mutateRate - 0.04
- mutateChange - 0.1

Numerous runs were done to find out what operators lead to the best fitness. All operators were tested as well as tournament size and population size parameters were tuned.

3.1 Selection operators

Tournament selection (size 20), Roulette wheel selection, random selection and elitist selection were tested. The training and test data set fitness of each selection type averaged over 10 runs each with 20000 evaluations in Table 2.

Operator	Training set fitness	Test set fitness
Tournament	0.0896	0.1857
Roulette	0.1011	0.1986
Random	0.0853	0.1642
Elitist	0.0988	0.2072

Table 2: Training and test set fitness for different selection operators

Random selection achieved the best results, followed by tournament selection with size 20. Both training and test results for random and tournament selection are not too distant from each other (training set difference - 0.0043, test set - 0.0215), however tournament selection heavily depends on tournament size. Therefore, more tests were run to see how tournament size effects training and test set fitness (see Table 3).

Tournament size	Training set fitness	Test set fitness
5	0.1141	0.2411
10	0.0916	0.1920
15	0.0841	0.1818
20	0.0896	0.1857
25	0.1064	0.2406
30	0.1224	0.2360

Table 3: Tournament size influence on training and test set fitness

From the table it is observed, that tournament selection with tournament size 15 performed best, nevertheless Random selection achieved best results of all selection operators and will be used for further tests.

3.2 Reproduction operators

One-point crossover, two-point crossover, uniform and arithmetic crossover were tested for reproduction operator. Due to time pressure, results are averaged of 5 runs of 20000 evaluations each. The results are shown in Table 4.

Crossover operator	Training set fitness	Test set fitness
One-Point	0.1065	0.1854
Two-Point	0.0751	0.1534
Arithmetic	0.1147	0.2158
Uniform	0.0690	0.1469

Table 4: Training and test set fitness for different reproduction (crossover) operators

Uniform crossover preformed best, followed closely by two-point crossover. The difference is 0.0061 on training data set and 0.0065 on test data set. Since the results are so close, each of the crossover methods could be used for further tests, but it was decided to continue other tests with uniform crossover as it did achieve slightly better performance.

3.3 Mutation operators

Inversion, swap, creep and standard (provided with the task) mutations were tested. Mutation rate was 0.04 and mutation change 0.1. Same as in Reproduction operators subsection, results are averaged of 5 runs of 20000 evaluations each. Table 5 demonstrates the results.

Mutation operator	Training set fitness	Test set fitness
Inversion	0.0630	0.1063
Swap	0.0690	0.1469
Creep	0.0864	0.1678
Standard	0.0400	0.0847

Table 5: Training and test set fitness for different reproduction (crossover) operators

Combination of operators (In order: Selection, Reproduction, Mutation, Replacement)	Training set 10k	Training data fitness	Test data fitness
Roulette, Uniform, Swap, RestrictedTS	0.0218	0.0984	0.2405
Roulette, OnePoint, Inversion, Keep-best	0.0271	0.1216	0.1736
Tournament,TwoPoint, Swap, Worst	0.0414	0.0236	0.1014
Random, Uniform, Swap, RestrictedTS	0.0513	0.0318	0.0331
Random, Uniform, Standard, RestrictedTS	0.0585	0.0710	0.1190
Tournament, TwoPoint, Creep, Keep-best	0.0588	0.1068	0.2462
Tournament, TwoPoint, Standart, RestrictedTS	0.0618	0.2784	0.3339
Random, Uniform, Creep, Worst	0.0636	0.1102	0.2355
Random, Arithmetic, Inversion, RestrictedTS	0.0640	0.1338	0.2060
Roulette, TwoPoint, Creep, Deterministic	0.0658	0.0891	0.1631

Table 1: Selection of initial combination of operators

For the training data set inversion and swap performed really closely, but mutation provided with the coursework code accomplished the best results, therefore future this mutation type will be used in further experiments.

3.4 Replacement operators

Random replacement, deterministic crowding replacement, restricted tournament selection (tournament size 20) and keep-best replacements were tested. Results are averaged of 5 runs of 20000 evaluations each. Table 6 demonstrates the results.

Mutation operator	Training set fitness	Test set fitness
Deterministic	0.1314	0.2331
RestrictedTS	0.0757	0.1676
Random	0.0940	0.2185
Keep-best	0.1157	0.2152

Table 6: Training and test set fitness for different replacement operators

From replacement operators restricted tournament selection (size 15) performed best, however same as with tournament selection, tournament replacement depends on tournament size. To confirm the best size, same tournament sizes as for selection were run and results are shown in Table 7. Results are averaged of 5 runs of 20000 evaluations each.

Tournament size	Training set fitness	Test set fitness
5	0.0809	0.1895
10	0.0423	0.0898
15	0.0757	0.1676
20	0.0635	0.1859

Table 7: Tournament size affect on fitness when tournament replacement is used

3.5 Population size

In "Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling" the influence of population size on GA performance was studied.[9] Populations from 5 to 200 member were tested and 100 member population size was found the most effective. Despite, that the problem solved is binary encoded, some of the same same population sizes - 20, 40, 60, 100, 200 - will be used in order to find population size, which allows to achieve best results. Again, each population size was averaged over 5 run with 20000 evaluations each. Results shown in Fig. 6, clearly demonstrates that for this algorithm population size 40 allows to achieve the best fitness, however this might be because the rest of the parameters and operators were tested and tuned using population of this size.

4 CONCLUSION

Parameters that achieved the best fitness, from explored options:

- Selection - Random
- Reproduction - Uniform
- Replacement - Restricted Tournament selection
- Mutation - Standard (provided)
- tournamentSize - 10
- numHidden - 5
- minGene - -1.0
- maxGene - 1.0
- popSize - 40
- mutateRate - 0.04
- mutateChange - 0.1

The table below shows the mean results of 10 runs which was 20000 evaluations each:

I believe a better result could have been achieved the research was done more thoroughly and more consistent experiments performed. Also, I think my algorithm was over fitting, because training results were significantly better compare to test data set result.However, the overall result is satisfactory for presented spacecraft landing problem.

5 FUTURE WORK

In future attempts to improve the algorithm, more consistent experiments should be done to get a better understanding how each operator type and each parameters affects the fitness of the algorithm. All tests should be run the same amount of time. Automated parameter tuning could be implemented to explorer larger parameter

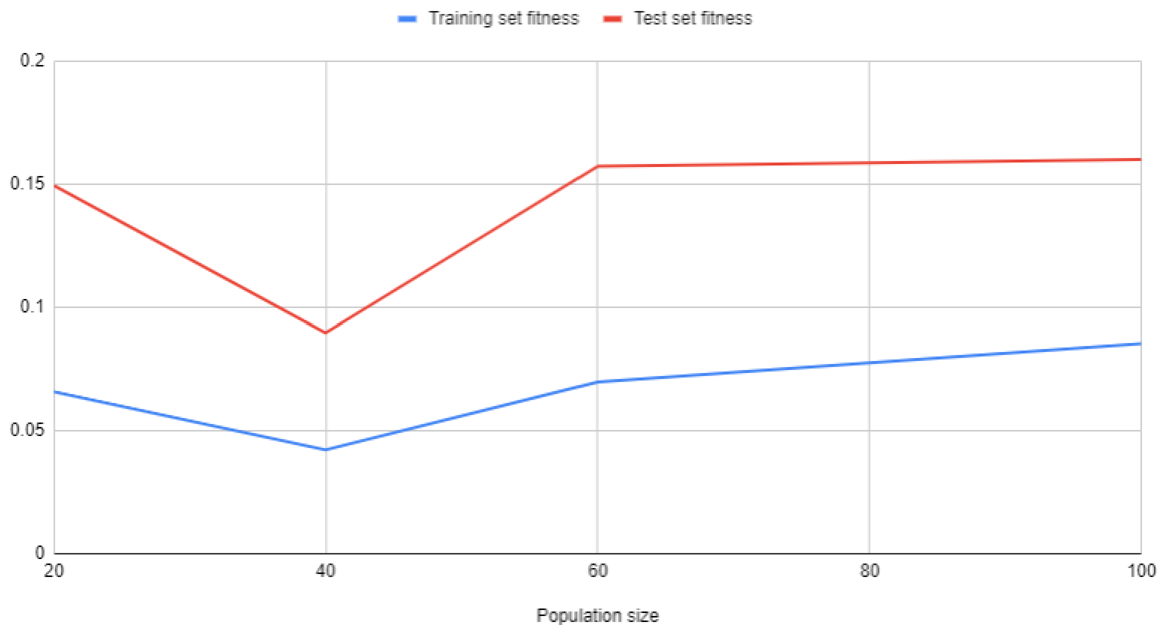

Figure 6: Training and test fitness changes from changing population size

Run	Training set fitness	Test set fitness
1	0.09872021611337546	0.20879373974558874
2	0.10466330994220809	0.24252457537890845
3	0.06331413291931608	0.10813606670454023
4	0.08230694332814455	0.152903439255349
5	0.026449505782863307	0.05224162239936054
6	0.025546024234989227	0.11146746765298819
7	0.10402726246136666	0.21792423762611438
8	0.09780407488095134	0.20342553441723077
9	0.0814104810247752	0.16233871797844662
10	0.10131398868313349	0.2331361371762515
Mean value:	0.07855559394	0.1692891538

space. Additionally, different neural network activation functions could be investigated to detect their influence on fitness values.

REFERENCES

[1] Chang Wook Ahn and Rudrapatna S Ramakrishna. 2003. Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation* 7, 4 (2003), 367–385.
[2] Valentina Alto. 2019. Neural Networks: parameters, hyperparameters and optimization strategies. https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5
[3] L. Booker. [n.d.]. Improving search in genetic algorithms. ([n. d.]).
[4] Andrew Chipperfield, Peter Fleming, Hartmut Pohlheim, and Carlos Fonseca. 1994. *Genetic Algorithm TOOLBOX For Use with MATLAB.* Technical Report.
[5] Haiming Du, Zaichao Wang, WEI Zhan, and Jinyi Guo. 2018. Elitism and distance strategy for selection of evolutionary algorithms. *IEEE Access* 6 (2018), 44531–44541.
[6] Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing.* Vol. 53. Springer.
[7] Manuel Lozano, Francisco Herrera, and José Ramón Cano. 2008. Replacement strategies to preserve useful diversity in steady-state genetic algorithms. *Information Sciences* 178, 23 (2008), 4421–4433.
[8] Sean Luke. 2013. *Essentials of metaheuristics.* Vol. 2. Lulu Raleigh.
[9] Olympia Roeva, Stefka Fidanova, and Marcin Paprzycki. 2013. Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. *2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013*, 371–376.
[10] Jim Smith. 2007. On replacement strategies in steady state evolutionary algorithms. *Evolutionary Computation* 15, 1 (2007), 29–59.
[11] Nitasha Soni and Tapas Kumar. 2014. Study of Various Mutation Operators in Genetic Algorithms.
[12] Hamid R Tizhoosh. 2005. Opposition-based learning: a new scheme for machine intelligence. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Vol. 1. IEEE, 695–701.