BIO-COMPUTATION: A REPORT OF THE ASSIGNMENT

Mohamed Ziaan Ahmed – S1900125

## INTRODUCTION

This paper is an analysis of three classification problems that were assigned as part of Bio-Computation Module’s coursework – with the goal of solving them with some form of evolutionary algorithm.

The first and the second dataset are a series of binary inputs and a binary output. A genetic algorithm (GA) was selected for these two datasets, the goal being to produce a function/classifier that can convert the binary inputs into their actual outputs with as much accuracy as possible.

The third dataset is a series of floating-point values and binary outputs.

## BACKGROUND RESEARCH

Data mining is concerned with developing researching and applying computerized methods and techniques to detect patterns in large collections of data (Peña-Ayala, 2014). They are used to classify information into groups based on the similarity of the information, that is to say, to find rules that the data adheres to. It could be said that GA implemented for the assignment would be part of this category, as the result of this GA is a sequence of steps, which can deterministically get expected output with a high degree of accuracy.

#### GENETIC ALGORITHM

The GA is part class of evolutionary algorithms –developed by John Holland – based on Darwinian theory of evolution of natural selection and survival of the fittest which were first. These techniques are ones that rather than accurately simulating actual evolution, are inspired by the strategies involved in the process (*The Nature of Code*., n.d.).

Though suitable for a wide variety of use cases optimization problems, especially when an exact or brute-force approach is too computationally expensive. A limitation of GA is that they can only be used for a problem that has been concretely defined. That is to say, a problem that can be encoded in a way that a computer can understand. The fitness function is another important aspect of the GA, as this is the part that determines how a good a chromosome (candidate solution) is.

Most GAs including the one employed for this assignment follow the same model. First a random population of chromosome genotype is created, which can be encoded in binary, real, or etc. As in nature, only the fittest of the population survive to procreate – the selection process is done based on a fitness function. The genetic material of the fittest solutions is used to create new solutions – referred to as crossover. A mutation factor may be introduced at this point as per a pre-determined percentage to introduce some randomness into the solution space – this can be very important to escape local optimums. In this way a new population of candidate solutions is produced. This process is referred to as “evolution” and is repeated until a solution that meets a certain set of requirements is found, or for a fixed set of time, or for a fixed set of generations (Hussain and Muhammad, 2020; *The Nature of Code*., n.d.).

A common issue with GA is premature convergence – that is to say, the solutions getting stuck in a local optimum. It has been observed that low population sizes will converge very quickly, which on the end, will take a long time to converge resulting in wastage of computing resources. Hence, to effectively utilize a GA, a balance between exploration (searching new areas to discover better fitness values) and exploitation (searching for the fittest of the currently discovered solutions) is essential (Hussain and Muhammad, 2020).

#### NEURAL NETWORK

Genotypes and Phenotypes

## IMPLEMENTATION

This section is cornered with the implementation methodology of the solutions selected for their respective datasets.

#### GENETIC ALGORITHM

The GA implemented for dataset 1 and dataset 2 evolves over generations a classifier that is able to classify the datasets inputs to get an output.

The nature of the datasets being binary, the logical conclusion was to generate a circuit which would pass the datasets values through logic gates to get an output. That is to say, the chromosomes for this GA are a sequence of logical operators, that work against datasets bit values in a reductive fashion, leaving only a single bit value at the end of the operation. That is to say, we are more or less using a fitness function to produce a fitness function.

The chromosomes have a dna attribute which holds the classifier. This attribute is an array of logical operators. When the population is randomly initialized at the beginning, the choice has to happen among the available logical operators.

Only positive fitness values are implemented. It works on the basis of the fitness being the number of correct answers a chromosome is able to produce. The success condition of the GA being a chromosome being able to produce a classifier that is able to classify the entire dataset correctly.

The generation of the new population is based on both accept-reject sampling and a tournament selection. In the original implementation, a roulette wheel was used, but this became unwieldy as the used version required a container of chromosomes, clones of the chromosomes existing in in this container as per their fitness ratios, and selecting a chromosome from the container randomly. As the number of chromosomes in the container increases exponentially as the population size increases, this made the program take a long time to process it. It also introduced a lot of noise into the population, as in some iterations, a lot of children chromosome had very bad fitness values.

The accept-rejection sampling makes the program work a lot faster due to how simple it is. It simply takes simply takes a random chromosome from the current population, generates a random value between 0 and the current best chromosomes fitness value, and if the selected chromosome has a higher fitness, this chromosome gets selected – this idea is inspired by the JavaScript implementation of the same concept by Daniel Schiffman (The Coding Train, n.d.). Two such chromosomes are selected, one point cross over from the mid-point is done, a mutation factor is then introduced at this point. This results in an output of two children chromosome and a clone of the main parent.

The resulting three chromosomes are then put in a tournament selection process, and the chromosome with the highest fitness enters the new population.

Algorithm:

1. A population of random chromones are generated.
2. Fitness is calculated.
3. Natural selection
4. New population generation
5. Evaluation

Repeat from step 2, if evaluation condition is not met, until either the perfect chromosome is generated, or the maximum number of generations have been met.

#### NEURAL NETWORK

The

## EXPERIMENTATION

#### 4.1 DATASET 1

Dataset one has 32 rows. Each consisting of a bitstring of length 5 and the output that it produces, which is a binary value. Form example, the first row of the dataset 1 is “00000 0”. This means that the input “00000” produces the output of “0”.

One of the main differences in dataset 1 and 2 is the length of the input bitstring. The implemented GA works on both these datasets, but the fitness function applied is slightly different for both. The ‘gates’ through which the datasets input bits are passed through have been arbitrarily decided. Each chromosome has dna (an array) attribute. This is the attribute which holds the series of logical operators that are to be used for it.

The fitness function works by looping through the genes (elements) of the dna, and mapping the input values to a series of hardcoded mappings. Since dataset 1 is being utilized in this section, this means only bits in the range of 1 and 5 can be used. For example, this can mean the 1st gene will map to the 2nd dataset input bit or something similar to get an output. This output is chained as an input to the next gene with another input bit. The gene is exhausted with all or some of the input bits, and we finally have a single binary output. Please note that the produced output is initialized at 1, and should be considered a bias. The produced output is compared to the actual output, and if it matches, this chromosome gets an increase of fitness of 1 point.

Each chromosome loops through the entire dataset, and we get a final fitness value for that chromosome. This is then done for all the chromosomes in the current population pool.

Current configuration works with a bias of 1. And the OR, AND, XOR gates available for the initial population creation.

Let us now see how it performs.

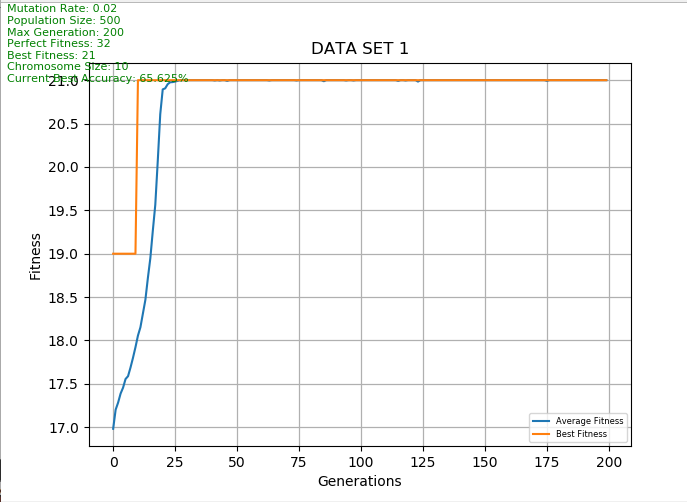


Figure 1: Dataset 1, initial run

As can be seen in Figure 1, the orange line plots the best fitness over generations and the blue plots the average fitness over generations. The GA gets stuck in a local optimum almost immediately, and the population converges very quickly. It can be surmised that there is not enough diversity in the population.

Next, we can see how a change in the mutation rate will affect the GA.

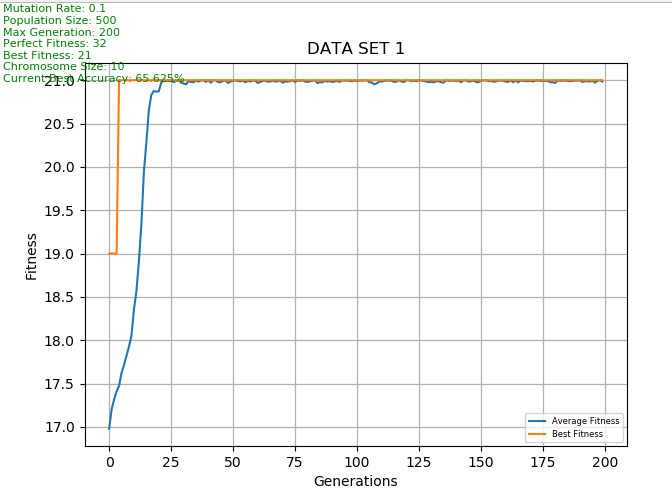


Figure 2: Dataset 1, mutation changed from 2% to 10%

Despite the change in mutation rate, nothing has changed.

With a change in population size to 200, it was attempted, yet the change was minimal as well. It can be surmised that GA is getting stuck in the local optimum because there is not enough diversity.

Even adding an additional logical operator to genes available for the production of chromosomes did not bring about an increase in max fitness.

At this point, the configuration was changed so that a total of 6 genes are now available for the creation of chromosomes. This means OR, AND, XOR, NAND, NOR, and XNOR are now available.

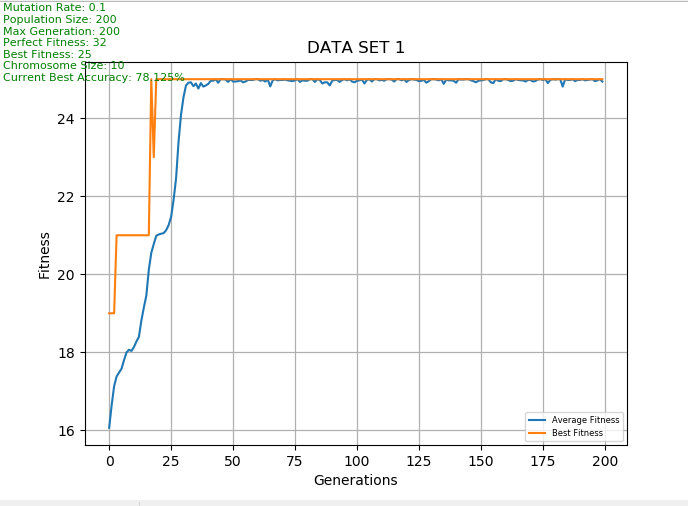


Figure 3: Dataset 1, additional logical operators introduced

Due to that fact that this worked, the best fitness does in fact increase when there is more diversity in the gene pool.

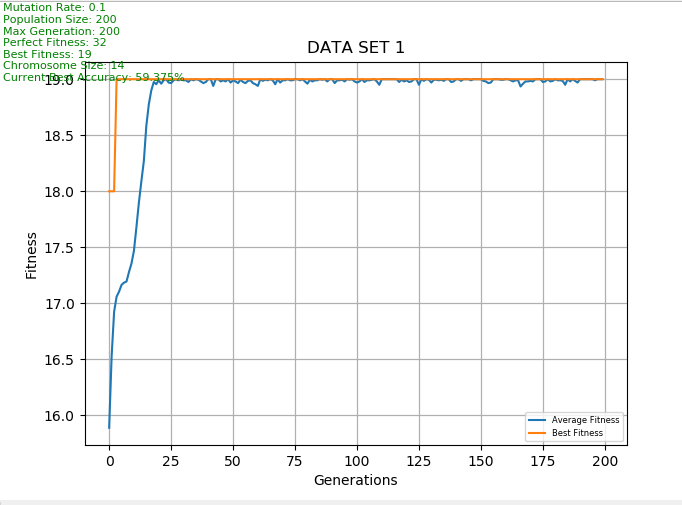


Figure 4: Dataset 1, the size of the chromosome has been increased

The size of the chromosome refers to the size of the circuit chain. A large chromosome means more genes can be stored, and expressed. But it seems that this decreased overall fitness. This change is reverted back.

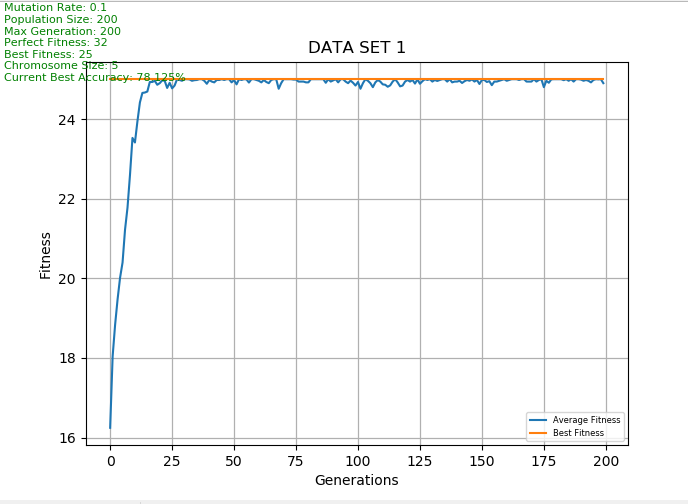


Figure 5: Dataset1, chromosome size decreased

Despite the change in chromosome size, the GA is able to maintain fitness to the levels of the run of Figure 3. From this it can be conclusion that increases chromosome size too much is detrimental to the fitness. But also, that after a certain point, there is a point of diminishing returns.

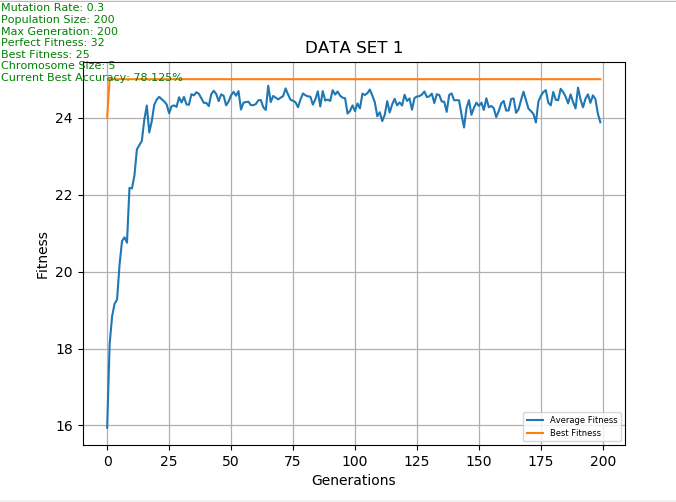


Figure 6: Dataset 1, mutation rate changed to 30%

The mutations rate is shown as per the average fitness fluctuating over the generations, but best fitness is maintained. This is not ideal as this will prevent convergence.

#### 4.2 DATASET 2

Dataset two has 64 rows. Each consisting of a bitstring of length 6 and the output that it produces, which is a binary value. The GA used for the dataset 1 and 2 are mostly the same. There are some minor changes in the fitness function to account for the extra bit. All 6 genes are available for chromosome production.

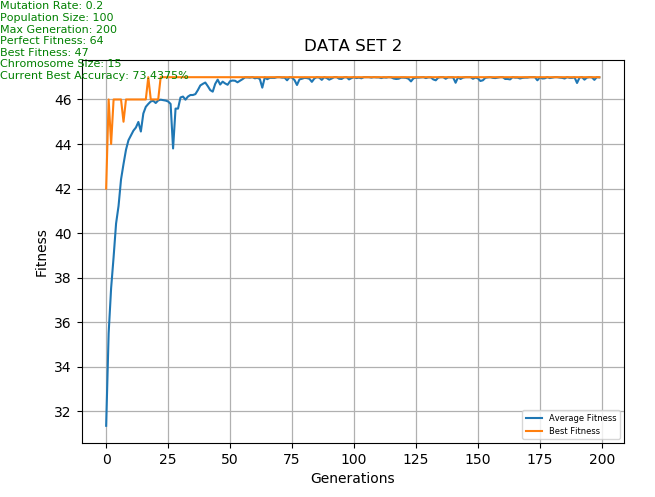


Figure : Dataset 2, initial run

The output accuracy is within the acceptable range.

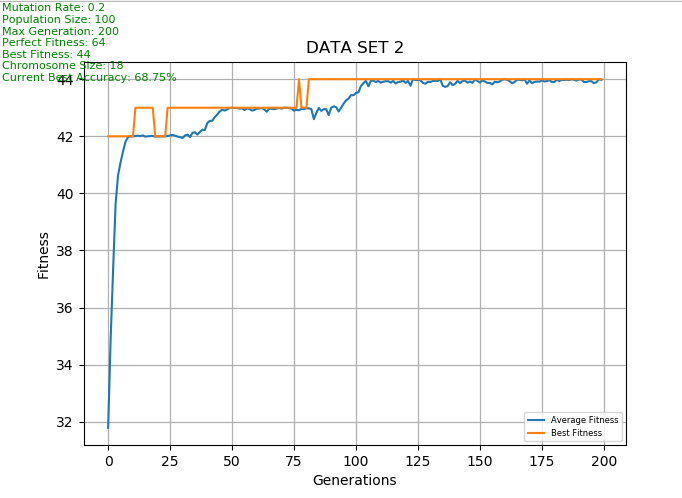


Figure : Dataset 2, chromosome size increased to 18

The increase of chromosome size by 3 seems to decrease overall accuracy.

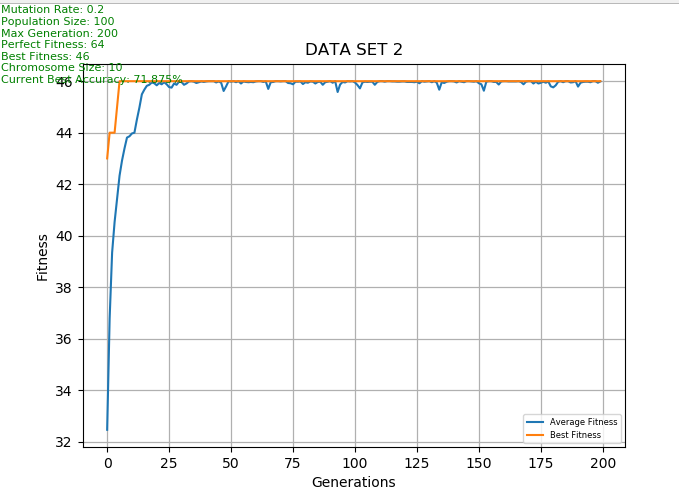


Figure : Dataset 2, chromosome size decreased

Decreasing the chromosome size past a certain level seems to decrease accuracy.

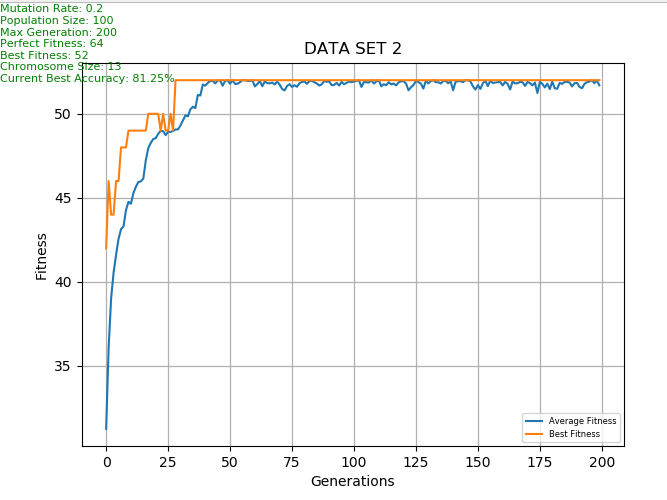
****

Figure : Dataset 2, chromosome size changed.

After the chromosome size was experimented with, it has been found that 13 gives exceptionally good results.

Decreasing the population size to 100 seems to still give the same results.

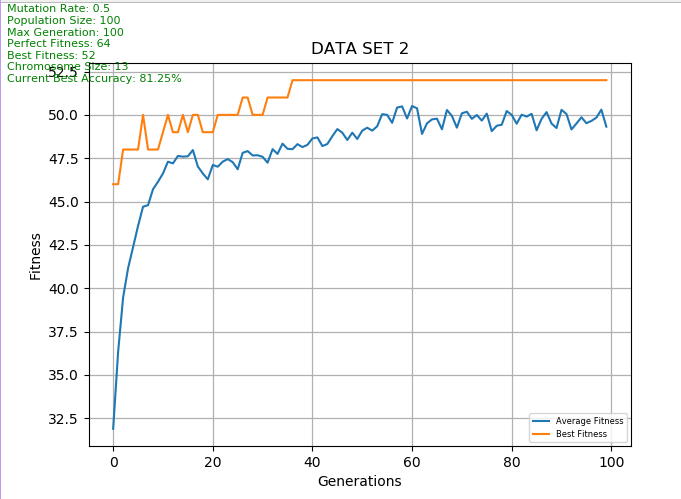


Figure : Dataset 2, increase mutation rate to 50%

The graphs clearly show that a diverse combination of genes the population consists of, yet the best fitness is maintained.

#### 4.3 DATASET 3

## CONCLUSION

## REFERENCES

Hussain, A. and Muhammad, Y.S. (2020) Trade-off between exploration and exploitation with genetic algorithm using a novel selection operator. *Complex & Intelligent Systems*. [online]. 6 (1), pp.1–14.

Peña-Ayala, A. (2014) Educational data mining: A survey and a data mining-based analysis of recent works. *Expert Systems with Applications*. [online]. 41 (4, Part 1), pp.1432–1462.

The Coding Train (no date) *9.7: Genetic Algorithm: Pool Selection - The Nature of Code* [online]. Available from: https://www.youtube.com/watch?v=816ayuhDo0E&list=PLRqwX-V7Uu6bJM3VgzjNV5YxVxUwzALHV&index=10 [Accessed 7 August 2021].

*The Nature of Code*. (no date) [online]. Available from: https://natureofcode.com/book/chapter-9-the-evolution-of-code/ [Accessed 6 August 2021].