UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS710 Assignment 1

Genetic Programming and Symbolic Regression

Student Number: u15024522

1 Introduction

This assignment involves implementing genetic programming to evolve a model for forecasting COVID-19 infections, deaths and recoveries for a particular day.

More specifically, the toolset provided by ECJ [**?**], A java-based evolutionary computation research system, was used to construct a genetic programming approach to solving the above-mentioned problem as a time-series analysis problem.

2 Representation

2.1 Nodes

The possible nodes for each individual comprises nodes from a function set and a terminal set. Nodes in the function set take a certain number of parameters and perform a calculation whereas terminals simply represent some constant value.

Initially, the function set consisted only of elementary numerical operators and the terminal set was made up of the number COVID-19 cases, deaths and recoveries on each of the 30 days preceding the date for which a prediction was being made. Unfortunately, this led to the same problem identified by Hui [1] in that the genetic program always converged to a local minimum which predominantly uses yesterday's stock price for its prediction.

As a result, a similar approach to Hoi's [1] was used to overcome the problem. Specifically - instead of using the number COVID-19 cases, deaths and recoveries - the daily changes in COVID-19 cases, deaths and recoveries was used. In addition, these were not directly used as terminal nodes; rather, the only terminal node used was the ephemeral floating random constant atom R.

The terminal node R produced constants in the range [0, 1) which were taken as arguments to 3 nodes in the function set (C, D and H) that map the value to the daily change in cases, deaths or recoveries on a specific day within the last 30 days. The rest of the function set was made up of the elementary numerical operators. The full function and terminal set are detailed in table 1.

Symbol	Arity	Description
+	2	The addition operator
-	2	The subtraction operator
*	2	The multiplication operator
/	2	The protected division operator (division by 0 results in 1)
C	1	Maps a value in the range [0,1) to the change in confirmed cases from the previous day for a particular day within the last 30 days
D	1	Maps a value in the range [0,1) to the change in deaths from the previous day for a particular day within the last 30 days
H	1	Maps a value in the range [0,1) to the change in recoveries from the previous day for a particular day within the last 30 days
R	0	The ephemeral random floating constant atom R in the range [0,1)

Table 1: Function and Terminal Set

2.2 Individuals

Each chromosome within a population consists of 3 syntax trees. To clarify, syntax trees include nodes and links the nodes. Nodes indicate the instructions to execute while the links indicate the arguments for each instruction. As per Koza [2], the internal nodes are called functions while the tree's leaves are called terminals. Given the function and terminal set that was identified, an example of one such syntax tree is depicted in figure 1.

The reason for having each chromosome comprise 3 different syntax trees is because the first tree will be optimized for predicting the number of confirmed cases, the second for predicting the number of deaths and the third for predicting the number of recoveries. The trees will be optimized in this way by means of the chromosome's fitness evaluation, described in section 4.

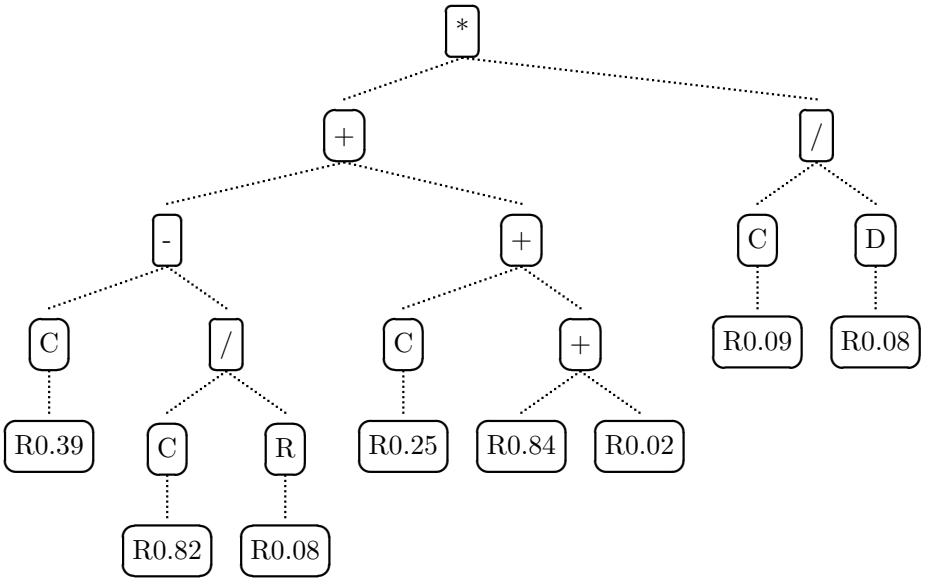

Figure 1: Syntax Tree

3 Initial Population Generation

Of course, a population consists of a collection of the individuals that have been described in section 2.2. Importantly, the genetic algorithm applied only made use of a single population consisting of 2048 individuals. This means that only a single population was evolved and cross-population breeding was not applied.

The initial population was generated using the ramped-half-and-half method proposed by Koza [2] to enhance population diversity of structure. The maximum tree depth allowed during the initial population generation was 8. So, in other words, the population is divided equally among individuals to be initialized with trees having depths 2, 3, 4, 5, 6, 7 and 8. For each depth group, half of the trees are initialized with the full technique and half with the grow technique.

In addition, to cover as much of the search space as possible an effort was made to reduce duplicates within the initial population. The strategy used to reduce the number of duplicates was as follows: if a duplicate individual is created as part of the initial population, it will be regenerated up to 100 times to try replace it with a new, original individual. After 100 attempts, if no unique individual was generated, the duplicate individual will be used.

4 Fitness Function

As a precursor to understanding how the fitness of a chromosome is calculated, it is important to note that the training data consists of a collection of records

from which the cumulative number of cases, deaths and recoveries associated with the COVID-19 virus on a particular day can be extracted.

To determine the fitness of individuals, we use the accuracy of each of the individual's 3 trees when predicting the number of cases, deaths or recoveries for each day in the training period. To be more specific, for a particular day the accuracy of the first tree is determined by calculating the difference between its prediction and the actual number of cases. Then, the difference is divided by the actual number of cases to get a ratio for which a lower ratio indicates a better fitness.

The same calculation is applied to the second tree to find its accuracy in predicting the number of deaths and to the third tree to find its accuracy in predicting the number of recoveries. The process is repeated for each day in the training data set until, eventually, the fitness of the chromosome is calculated as the average accuracy in calculating cases, deaths and recoveries over the training period. Algorithm 1 depicts the fitness function in its entirety.

Algorithm 1: Fitness Function

```
foreach date in trainingData do
    cases = trainingData.cases(date);
    predictedCases = individual.trees[0].predict(date);
    casesAccuracy = abs(cases - predictedCases) / cases;

    deaths = trainingData.deaths(date);
    predictedDeaths = individual.trees[1].predict(date);
    deathsAccuracy = abs(deaths - predictedDeaths) / deaths;

    recovered = trainingData.recovered(date);
    predictedRecovered = individual.trees[2].predict(date);
    recoveredAccuracy = abs(recovered - predictedRecovered) / recovered;

    fitness += (casesAccuracy + deathsAccuracy + recoveredAccuracy) / 3;
end

return fitness / trainingData.numDays;
```

5 Selection Method

To generate a new population, specific genetic operators are applied to members of an existing population according to predefined probabilities. These operators are explained in more detail in section 6. However, before these operators can be applied, the members of the existing population to which they need to be applied must be determined - hence, a selection method is necessary.

For this assignment, tournament selection [3] was used as the selection method for all genetic operators. This entails selecting T individuals entirely at random from the existing population. Then, from this set of individuals (ie. the *tournament*) the fittest individual is selected. After some initial test runs the size, T, of the tournament used was 256. On average, this tournament size produced the best balance between genetic diversity and convergence.

In the cases where genetic operators required more than one individual from the

existing population, the tournament selection method was simply reapplied to select each individual.

6 Genetic Operators

As has been mentioned, genetic operators are applied to members of an existing population to generate a new population. For this assignment the operators used included reproduction, crossover and mutation. As detailed by Koza and Poli [4], these operators can be briefly summarized as follows:

- Reproduction: Copy the selected individual program to the new population as is.
- Crossover: Create offspring for the new population by combining randomly chosen parts from two selected programs.
- Mutation: Create one new offspring for the new population by randomly mutating a randomly chosen part of a selected individual.

Of course, these operators will not be used uniformly to generate new populations. Rather, every operator is associated with a probability which determines the likelihood that it will be used to select the next individual for the new population. Specifically, the probability for crossover was 0.9, the probability for mutation was 0.05 and the probability for mutation was also 0.05.

In addition, when applying operators decisions must be made regarding whether to select terminal (leaf) or non-terminal (internal) nodes. The probability for selecting a terminal node was chosen as 0.1 whereas the probability for selecting a terminal node was chosen as 0.9.

Finally, a maximum tree depth of 16 was applied as a constraint to the individuals of a new population. In the case of crossover and mutation, the operators would not try to produce a valid individual more than once before failing.

With the selection methods and genetic operators having been identified, figure 2 depicts the entire breeding pipeline for generating a new population from an existing population.

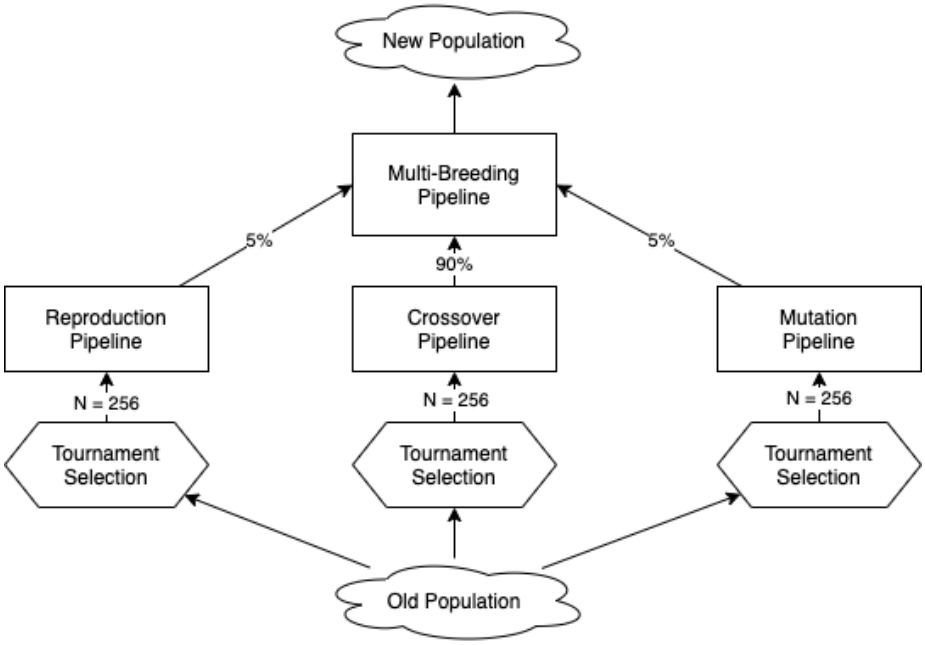

Figure 2: Breeding Pipeling

7 Termination Criteria

As is customary for most genetic programming approaches, certain termination criteria are necessary to ensure a finite amount of time before a solution is produced. For this assignment we include a maximum number of generations to be run as well as a problem-specific success predicate as the termination criteria.

The success predicate entails an individual predicting the cases, deaths and recoveries for all dates in the training set within a 5% range. The maximum number of generations allowed was chosen as 256. Notably, in the tests that were run the success predicate was never triggered.

8 Experimental Setup

The parameter values used for the genetic programming algorithm have been mentioned throughout the preceding sections. Nevertheless, the most prominent parameters are as follows:

- Population Size: 2048
- Crossover Rate: 0.9
- Mutation Rate: 0.05
- Reproduction Rate: 0.05

- Maximum Generations: 256

The data set used for training and testing can be found at `https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset`. Specifically, the data set comprising reported cases, deaths and recoveries between 22-01-2020 and 29-08-2020 was extracted from the aforementioned link.

Furthermore, the data set was then split into a training set, used for the execution of the genetic algorithm, and an evaluation set, kept for cross-validation. Also, it is important to note that, due to the structure of the function and terminal sets, the first 30 days included in the data set were not directly used to evaluate fitness. The size and precise dates used for the split data set is shown in figure 3. This approach was heavily inspired by Hui [1].

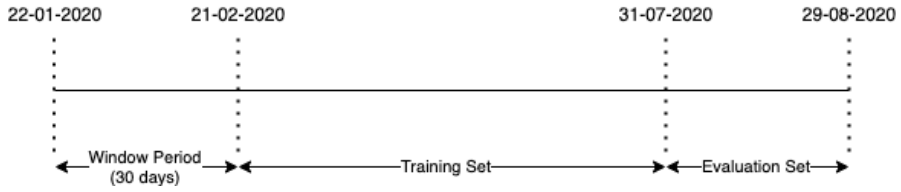

Figure 3: Data Set Division

To test the algorithm a script was written which would run the algorithm on the data set with 10 different sets of seeds, writing the output of the program, it's execution time and the results of the algorithm to files which could be referenced later. The script is available in the github repository for this project (available at `https://github.com/marcus-bornman/cos_710_assignment_1`).

The machine used for development and testing purposes was a 2015 15-inch Macbook Pro with a 2.5 GHz Quad-Core Intel Core i7 processor, 16GB of memory and 500GB of storage. At the time of experimentation the machine had a battery cycle count of 684.

9 Results and Discussion

As has been mentioned, 10 test runs were executed. Each run entailed running the genetic algorithm with a different set of seeds. The best individual fitness, number of hits, mean squared error and runtime for each of the 10 runs are detailed in table 2. To clarify, a single hit is a day for which the prediction of the best individual was within an average of 5% of the actual cases, deaths or recoveries.

Run	Best Individual Fitness	Number of Hits	Mean Squared Error	Runtime
1	0.16981653012815615	11	0.02884	0m56.559s
2	0.11708801796762536	18	0.01371	5m46.942s
3	0.07924233842929927	39	0.00628	7m59.488s
4	0.10260196769743599	36	0.01053	7m38.561s
5	0.09179369543784227	33	0.00843	5m31.616s
6	0.14551612276194176	10	0.02117	2m17.149s
7	0.08699608922920267	49	0.00757	3m9.732s
8	0.10736871065116832	22	0.01153	2m56.155s
9	0.11435758695560931	21	0.01308	2m5.639s
10	0.1395859048040605	14	0.01948	3m45.321s

Table 2: Test Results

As can be derived from the results, the best mean squared error was 0.00628; the average mean squared error was 0.01354; the best fitness was 0.07924 and, lastly the average fitness was 0.10971.

When compared to the results of genetic programming approaches that were used to solve similar time-series problems [1, **?**] the results of these tests also indicate the effectiveness of the approach to perform time series predictions. Given the short run times, limited generations and relatively small population size the algorithm produced a function that could predict test data within a 7% accuracy on average.

An area of concern, however, is that the functions produced have not yet been used for cross-validation. A part of the data set that could be used for cross validation has been separated as part of this assignment but performing the cross-validation has been left as an area for future work.

References

[1] A. Hui, "Using genetic programming to perform time-series forecasting of stock prices," *Genetic Algorithms and Genetic Programming at Stanford*, pp. 83–90, 2003.

[2] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[3] B. L. Miller, D. E. Goldberg, *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.

[4] J. R. Koza and R. Poli, "Genetic programming," in *Search methodologies*, pp. 127–164, Springer, 2005.