CS547 Assignment 2
Project Cost Estimation

Wojciech Cichoradzki
201506554

01/12/2019

1 Abstract

A Python 3 script was implemented which goal is to find appropriate function that takes a set to inputs and returns predicted development cost of the project (effort) using genetic programming approach. Three different datasets were evaluated and compared against non-genetic programming approach that used linear regression. This reports consists of the analysis of the solutions of three datasets:

1. Albrecht dataset consisting of 7 inputs and 24 records
2. Kemerer dataset consistsing of 6 inputs and 16 records
3. China dataset consisting of 16 inputs and 499 records

The results show better performance of genetic programming approach compared to linear regression for given dataset.

2 Methodology

2.1 Genetic Programming configuration

Each dataset was evaluated using similar configuration of the script. Configurations for each datasets are stored inside the `configs` directory. The shared configuration for all genetic programming tests included:

- Mating chance - 50%
- Mutation chance - 20%
- Number of generations - 1000
- Population size - 200
- Depth limit of the syntax tree - 20

Moreover each dataset was split so part of the dataset was used to train the model while the rest was used for validation testing. The train/test split for each dataset was as follows:

1. Albrecht: 70/30 %
2. Kemerer 66/34 %
3. China 66/34 %

Test data was processed to include only meaningful inputs (e.g. exluding ID numbers). When it comes to function syntax generation the following primitives and constants were allowed to be generated:

1. Addition of 2 elements
2. Subtraction of 2 elements
3. Multiplication of 2 elements
4. Negation of 1 element
5. Random ephemeral integer constant from the set of $\{-1, 0, 1\}$
6. Random ephemeral float constant between -1 and 1 rounded to 3 decimal points.

2.1.1 Prediction metric

To evaluate validity of the candidate solution the Mean Magnitude of Relative Error (MMRE) metric was used calculated as follows:

$$MMRE = \frac{\sum_{i=1}^{N} \frac{|f_a(i) - f_p(i)|}{f_a(i)}}{N} \tag{1}$$

where N is the size of test data, i is test number, $f_a(i)$ is the actual result for i^{th} test, $f_p(i)$ is the predicted results for the i^{th} test. This value was then fed as the fitness function result to the genetic programming solver as a minimizer function.

2.2 Linear regression comparison metric

Linear regression comparison was evaluated using weka software using same test/train split setting as for respectful tests performed in the genetic programming part. The predicted values for test data were then evaluated later in MS Excel to calculate the $MMRE$ since weka is not providing this metric out of the box. All the weka-related results are stored in the `/results/<test-name>/weka` directories.

3 Evaluation of results

3.1 Albrecht dataset

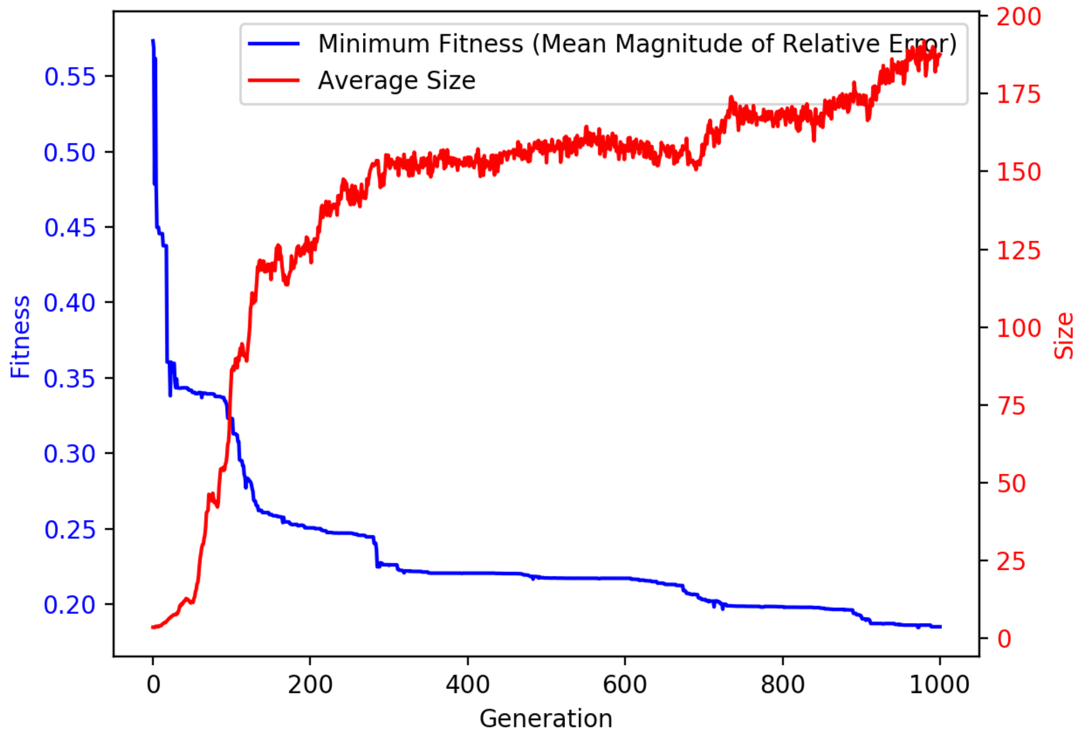

Figure 1: Fitness vs syntax tree size over generations in genetic programming approach

As seen in fig. 1, genetic programming approach on the train data achieved quick decrease in the $MMRE$ value over first 100 generations alongside rapid growth of the syntax tree size to over 100 nodes. Over the next generations, until the ending point, it reached the minimum of under 20% mean magnitude of relative error and around 185 nodes in the return function.

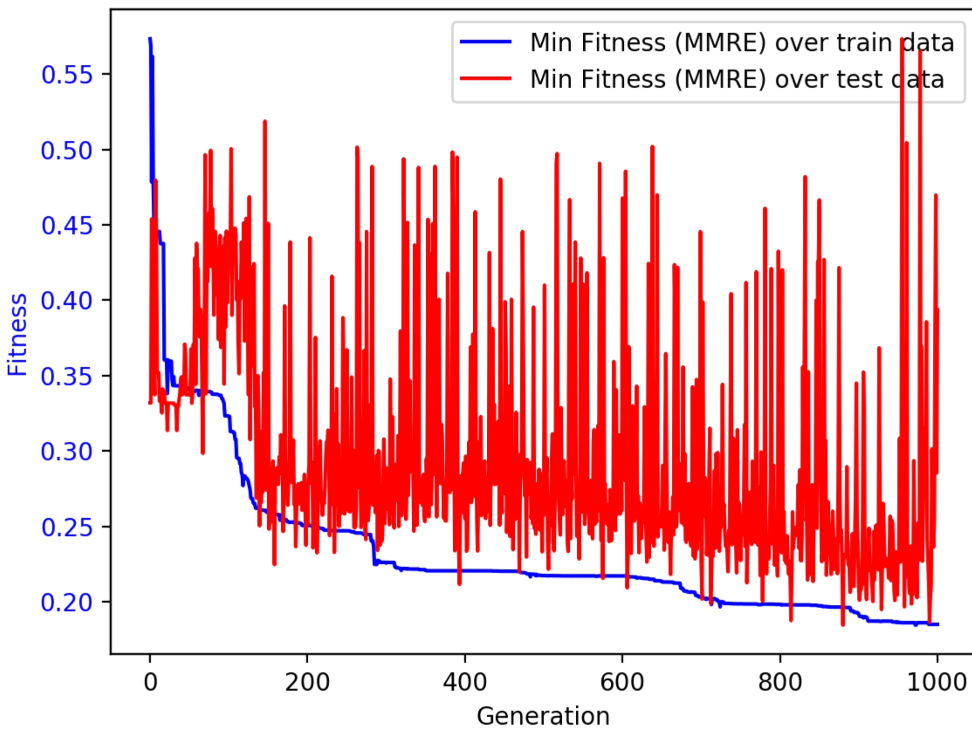

Figure 2: Comparison of MMRE on test and train data over generations

Next, the result function was compared against train and test data in fig.2. MMRE over test data was varying greatly as the model was trained hence generating a lot of noise on the graph. Nevertheless, it can be noticed that the average curve over test data was also decreasing over generations as the model was trained resulting at best around 40% - 50% MMRE. The spike at the end of the graph at roughly 950^{th} generation may be a result of overfitting the function on train data.

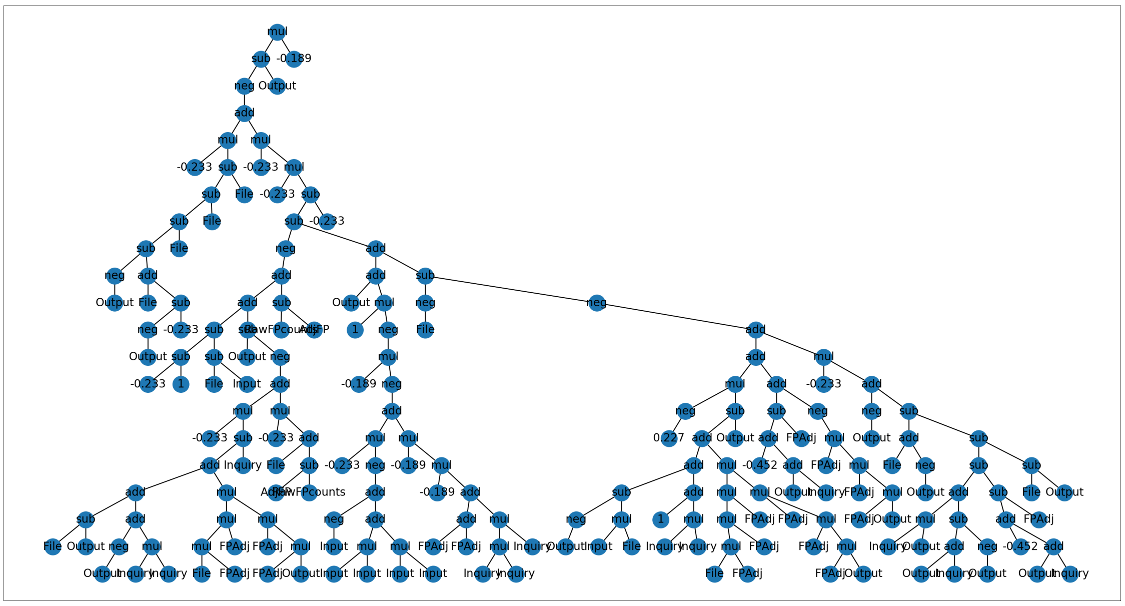

Figure 3: Syntax tree graph of the best resulting function for the albrecht dataset

Nevertheless, the albrecht dataset was too small to find the well-performing approximation model, the best function that the genetic programming solver came up with can be seen above in fig. 3. It consists of over 150 nodes and reached the limit of 20 levels depth. Due to this limit, it could be a blocking factor fo the function to evolve further and yield better results.

Lastly, the results from running weka linear's regression classifier yielded much simpler function:

$$Effort = 0.405 * Output - 2.2232 \tag{2}$$

The resulting MMRE over test data of the linear regression solution equaled to 49.78% which is within the upper limit of the best performing genetic programming solutions.

As a result, both genetic programming and linear regression classified provided somewhat comparable solutions with the mean magnitude of relative error at around 50%. This magnitude of error is very large and as such is not suitable for any further practical usage for project cost estimation. Genetic programming solutions were slightly outperforming linear regression ones, however it took them significantly more time. In order to improve the results with the genetic algorithm, one could try expanding the limit of 20 depth, as my machine was performing very slowly when this limit was incrased. With even larger search space, and ideally larger train dataset, genetic programming might have outperfomed linear regression classifier.

3.2 Kemerer dataset

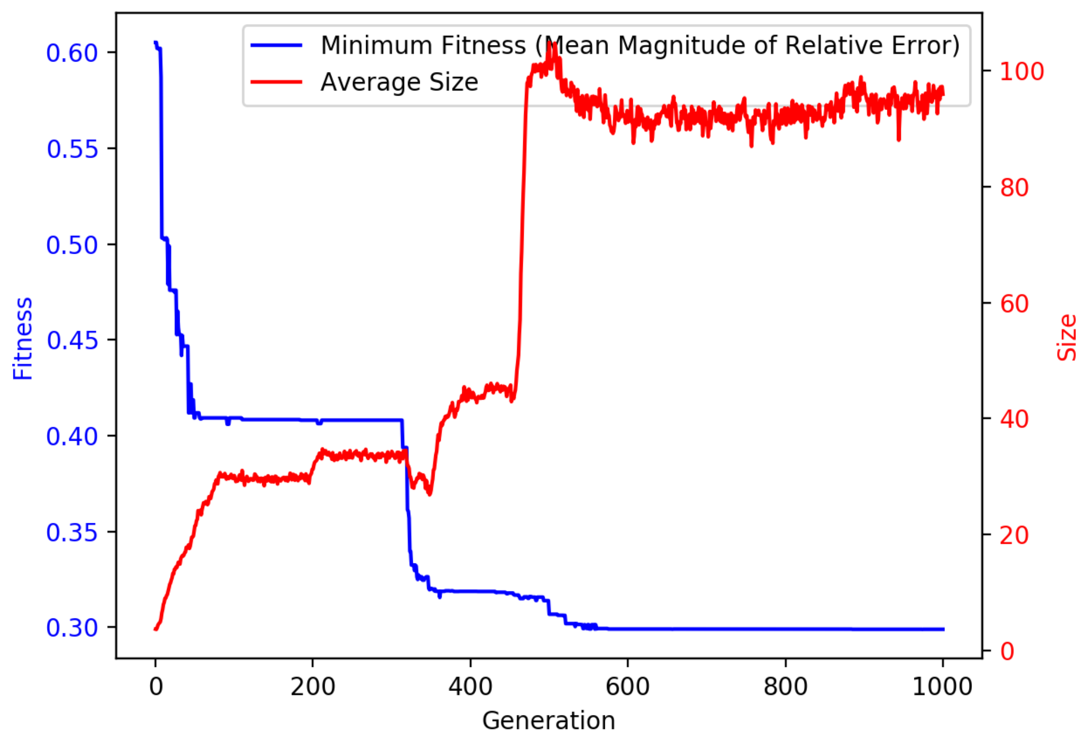

Figure 4: Fitness vs syntax tree size over generations in genetic programming approach

As seen in fig. 4, genetic programming approach on the kemerer train data again achieved quick decrease in the $MMRE$ value over first 50 generations, then it stagnated for about 200 generation only to jump with a decrease by about 0.1 MMRE before generation 400. After generation 600, it maintained same fitness value until the end of experiment, resulting in the final minimum mean magnitude of relative error metric at around 30% on train data and the size of the syntax tree of roughly 100 nodes.

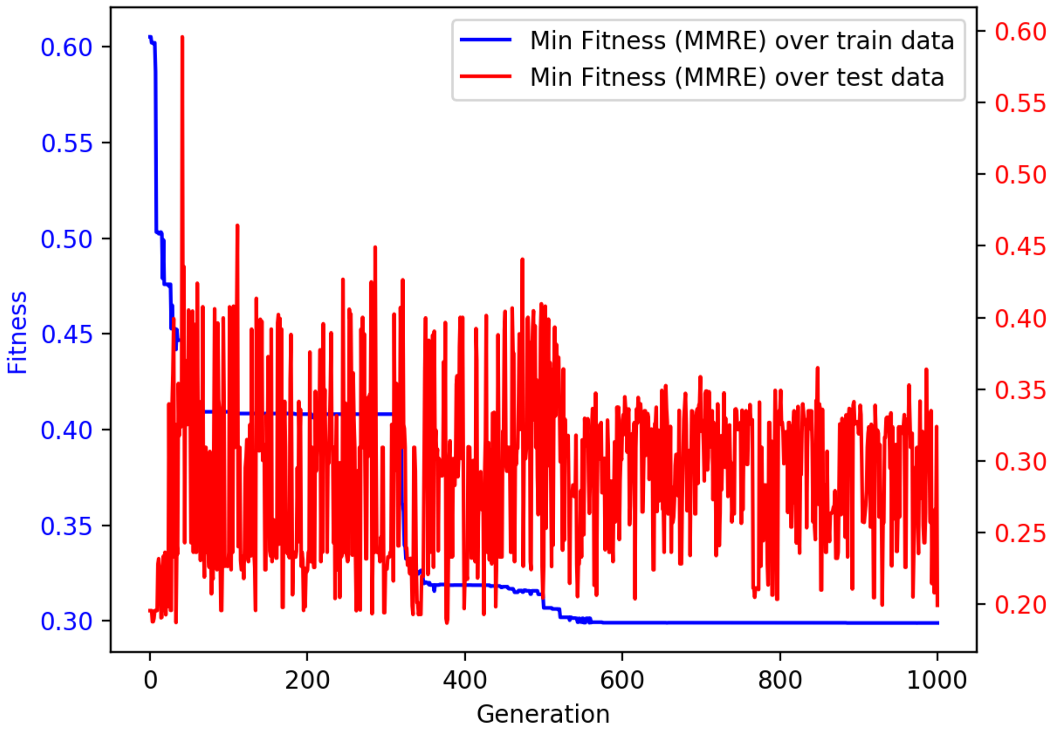

Figure 5: Comparison of MMRE on test and train data over generations

As seen in fig. 5 MMRE over test data was varying between 40-20% until the stagnation after 600 generations when it reached the range between 35-20%.

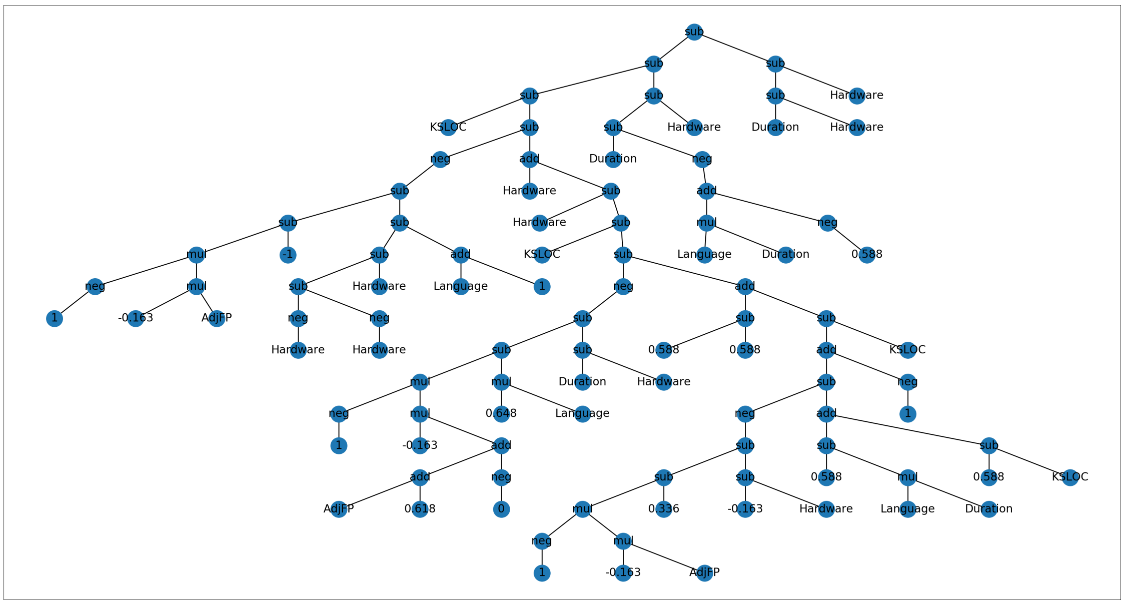

Figure 6: Syntax tree graph of the best resulting function for the kemerer dataset

The generated effort function is much cleaner compared to albrecht dataset result, while still being much more complex comparing against the generated by weka's linear's regression function:

$$Effort = 80.9791 * Hardware + 0.4557 * AdjFP - 445.5517 \tag{3}$$

In this case, the resulting MMRE of the linear regression solution equaled to the whopping 111.83% which is 4 to 6 times larger value than the genetic programming solution.

As a result, it the case of kemerer dataset, genetic programming yielded a much more optimal solution than the weka's linear regression classifier. It could be due to the larger search space and the non-polynomial result of the optimal effort function.

3.3 China dataset

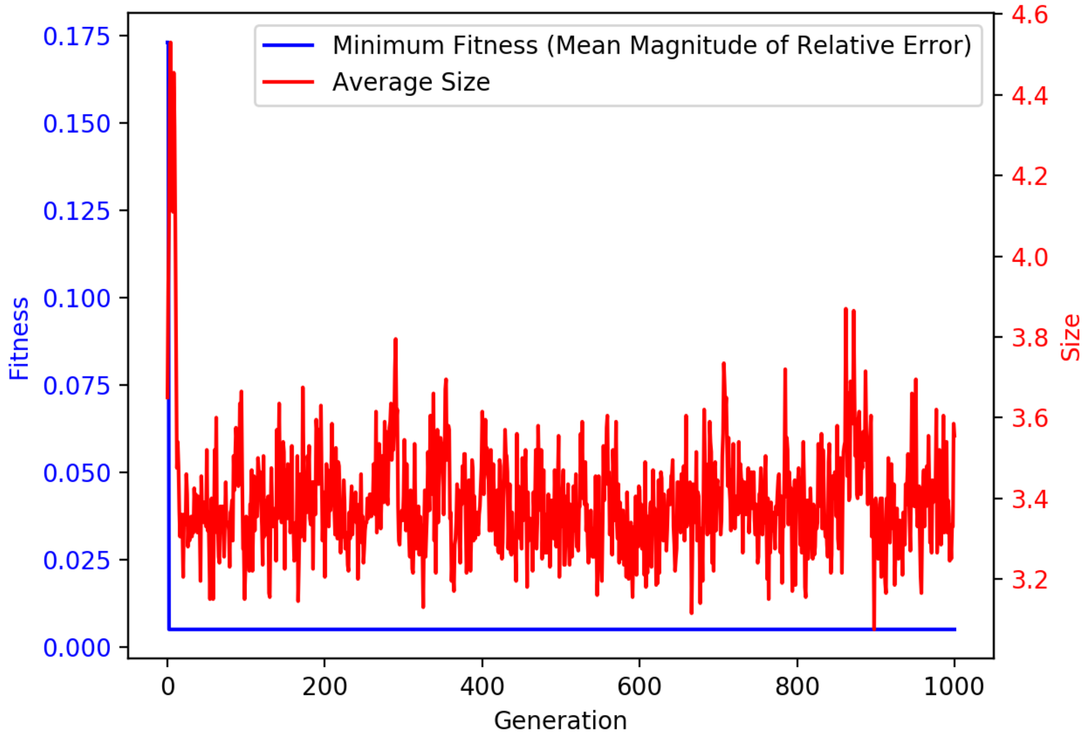

Figure 7: Fitness vs syntax tree size over generations in genetic programming approach

As seen in fig. 7, a very interesting situation occured when running genetic programming solver against the china dataset. In the second generation, the almost-error-prone solution was found that resulted in the MMRE of just over 0.5% and the size of just 3 nodes!. Over the next 998 generations, the program could not come up with anything better.

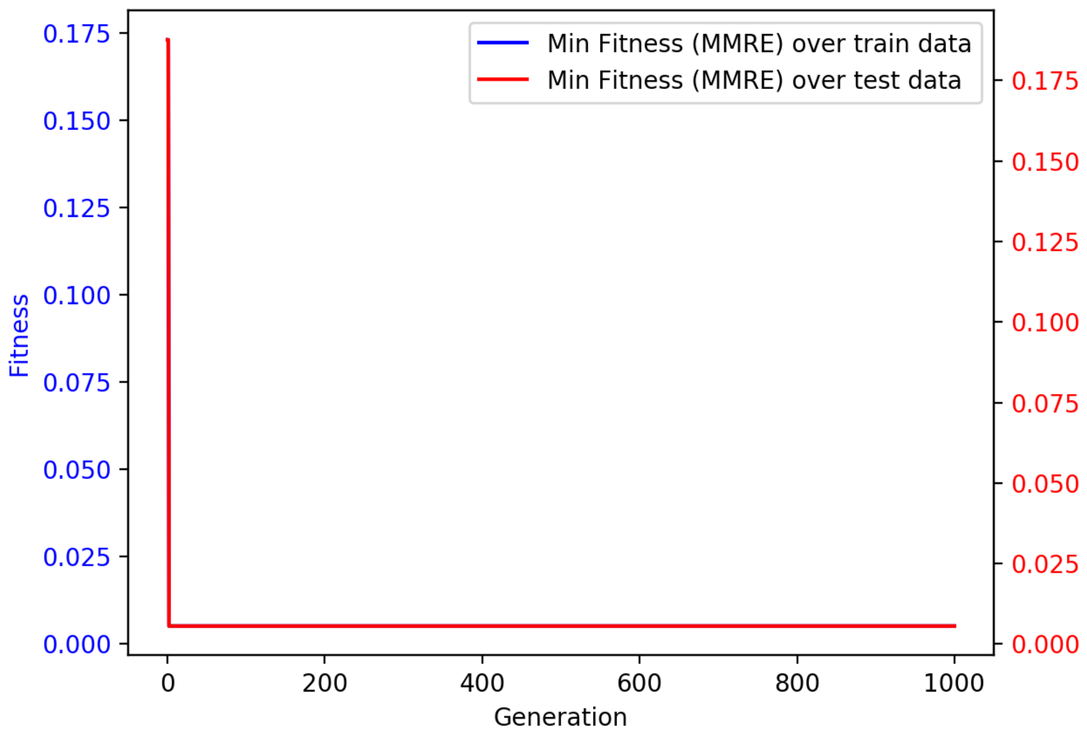

Figure 8: Comparison of MMRE on test and train data over generations

Even more interestingly, the resulting function yielded identical result against test data. That means that the function is in clear correlation to the effort result.

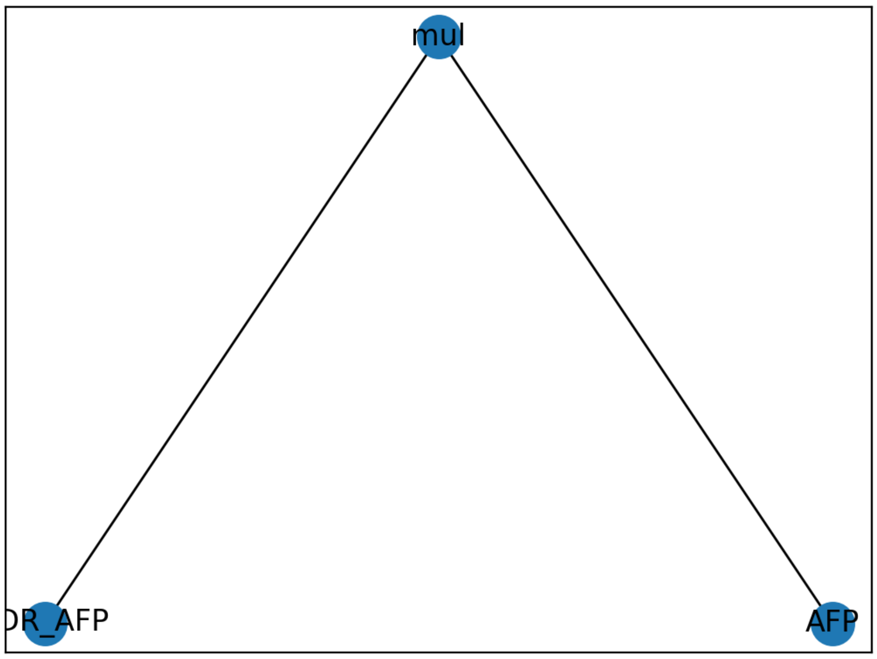

Figure 9: Syntax tree graph of the best resulting function for the china dataset

As seen in fig. 9, the effort function is simply two (out of 16 in total) inputs multipled together.

The function that weka classifier came up with is much more complex:

$$\begin{aligned}
&Effort = \\
&-0.0027 * Input + \\
&0.022 * Enquiry + \\
&-0.013 * File + \\
&0.0191 * Interface + \\
&0.0059 * Added + \\
&0.5822 * PDR\_AFP + \\
&-0.2798 * PDR\_UFP + \\
&-0.2209 * NPDR\_AFP + \\
&1.1732 * Resource + \\
&3.9372
\end{aligned}$$

(4)

In this case, the resulting MMRE of the linear regression solution equaled to 64.38% which is vastly larger than <1% from the genetic programming solution.

As a result, it the case of the china dataset, genetic programming created an almost-optimal solution very quickly. However, it has to be noted that there is a possibility than the effort in the china dataset could be actually a result of the `DR_AFP * AFP` and the relative error comes from the floating-point arithmetic error. If that's the case, these two inputs should have been excluded from the search.