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

COS710 Assignment 3

Genetic Programming and Modularization

Student Number: u15024522

1 Introduction

This assignment involves employing genetic programming and incorporating modularization using automatically defined functions (ADFs) to produce a classifier for postoperative patient diagnosis. In addition, the results of this approach are compared to those produced by a previous genetic programming approach that did not incorporate modularization - see `https://github.com/marcus-bornman/cos_710_assignment_2`.

The toolset provided by ECJ [1] - a java-based evolutionary computation research system - was used to construct the genetic programming approach to evolve a population of arrays of trees. The first tree in each array represents the main decision-tree for classifying postoperative patient diagnosis for the patients in the Post-Operative Patient Data Set (available from the UCI Machine Learning repository [2]). The rest of the trees in each array represent ADFs.

The aforementioned data set consist of 90 records. The data set was randomly divided into a training set of 70 records and an evaluation set of 20 records. Each record consists of 9 attributes, which can be described as follows:

- L-CORE (patient's internal temperature in C): high (> 37), mid (≥ 36 and ≤ 37), low (< 36)
- L-SURF (patient's surface temperature in C): high (> 36.5), mid (≥ 36.5 and ≤ 35), low (< 35)
- L-O2 (oxygen saturation in excellent (≥ 98), good (≥ 90 and < 98), fair (≥ 80 and < 90), poor (< 80)
- L-BP (last measurement of blood pressure): high ($> 130/90$), mid ($\leq 130/90$ and $\geq 90/70$), low ($< 90/70$)
- SURF-STBL (stability of patient's surface temperature): stable, moderate, unstable
- CORE-STBL (stability of patient's core temperature) stable, moderate, unstable
- BP-STBL (stability of patient's blood pressure) stable, moderate, unstable
- COMFORT (patient's perceived comfort at discharge, measured as an integer between 0 and 20)
- ADM-DECS (discharge decision): I (patient sent to Intensive Care Unit), S (patient prepared to go home), A (patient sent to general hospital floor)

The classification task of this data set is to determine where patients in a postoperative recovery area should be sent to next (the discharge decision). For this assignment, the *COMFORT* attribute is not considered for the classification problem as it contains missing values. So, the classifier will use 7 patient attributes to classify patient's into one of three discharge decisions.

All of the code that was used to employ the genetic algorithm and the information necessary to reproduce the results can be found on github at `https://github.com/marcus-bornman/cos_710_assignment_3`.

2 Representation

2.1 Nodes

Nodes for the main decision tree, or ADFs, comprise decision and end nodes. Decision nodes take a number of children and select one of them based on the value of a patient's attribute. There are 7 decision nodes - one for each attribute, excluding *COMFORT* as it contains missing values. The number of children for decision nodes depends on the possible values for their associated attributes. The Blood Pressure (*BP*) node, for instance, has 3 children - one for *low*, one for *mid* and one for *high*. End nodes represent one of the three discharge decisions (*I*, *S* or *A*); or, in the main decision-tree, end nodes may also reference one of 3 ADFs. The representation of ADFs is further explained in section 2.2. The end nodes and decision nodes are described in table 1.

Symbol	**Arity**	**Description**
IT	3	The internal temperature decision node - If the patient's internal temperature is low then select the first child - If the patient's internal temperature is mid then select the second child - If the patient's internal temperature is high then select the third child
ITS	3	The internal temperature stability decision node - If the patient's internal temperature is unstable then select the first child - If the patient's internal temperature is moderately stable then select the second child If the patient's internal temperature is stable then select the third child
ST	3	The surface temperature decision node - If the patient's surface temperature is low then select the first child - If the patient's surface temperature is mid then select the second child - If the patient's surface temperature is high then select the third child
STS	3	The surface temperature stability decision node - If the patient's surface temperature is unstable then select the first child - If the patient's surface temperature is moderately stable then select the second child - If the patient's surface temperature is stable then select the third child
BP	3	The blood pressure decision node - If the patient's blood pressure is low then select the first child - If the patient's blood pressure is mid then select the second child - If the patient's blood pressure is high then select the third child
BPS	3	The blood pressure stability decision node - If the patient's blood pressure is unstable then select the first child - If the patient's blood pressure is moderately stable then select the second child - If the patient's blood pressure is stable then select the third child
OS	4	The oxygen saturation decision node - If the patient's oxygen saturation is poor then select the first child - If the patient's oxygen saturation is fair then select the second child - If the patient's oxygen saturation is good then select the third child - If the patient's oxygen saturation is excellent then select the fourth child
I	0	The end node for the patient to be sent to an intensive care unit
S	0	The end node for the patient to be sent home
A	0	The end node for the patient to be sent to the general hospital floor
ADF0	0	Returns the result of executing the first ADF of the individual.
ADF1	0	Returns the result of executing the second ADF of the individual.
ADF2	0	Returns the result of executing the third ADF of the individual.

Table 1: Decision Tree Nodes

2.2 Individuals

Each chromosome within a population consists of a 4 decision trees for determining the discharge decision for a patient. As per the architecture implemented by Bruce [3], the first decision tree represents the main program while the other three trees represent ADFs.

The function set and terminal set for the main program are made up of all the available nodes in table 1. For the ADFs the terminal sets all consist of only *I*, *S* and *A* - hence, there are no references between the function-defining trees. In addition - the function set for the first ADF consists of *BP* and *BPS*, the function set for the second ADF consists of *IT* and *ITS*, and the function set for the third ADF consists of *ST* and *STS*.

Given the function and terminal sets described above for the main program and each ADF, an example of an individual is depicted in figure 1.

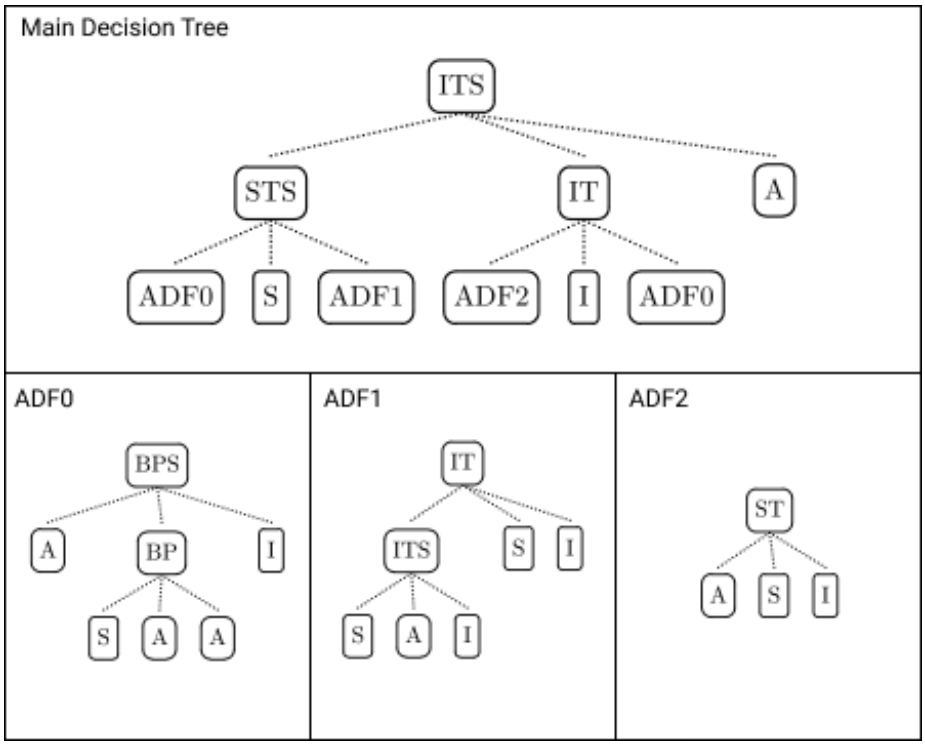

Figure 1: Example of an Individual

2.3 Populations

Population representation and generation is the same as that applied by the previous genetic programming approach that did not incorporate modularization. To reiterate, the genetic algorithm 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 [4] to enhance population diversity of structure. The maximum tree depth allowed during the initial population generation was 4. So, in other words, the population is divided equally among individuals to be initialized with trees having depths 2, 3, and 4. 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.

3 Fitness Function

Again, fitness evaluation was applied using the same means applied by the previous genetic programming approach that did not incorporate modularization.

For clarity, to determine the fitness of individuals, we use the number of patients for which the individual classified the discharge decision incorrectly. To be more specific, for a particular patient the predicted discharge decision is determined using the main decision tree of the individual in question; then, if the predicted discharge decision is not the same as the recorded discharge decision the fitness is increased by 1. This process is repeated for all patients in the training set.

Of course, using this calculation for the fitness of an individual means an individual with a lower fitness value is deemed better than one with a higher fitness value. Algorithm 1 depicts the fitness function in its entirety.

Algorithm 1: Fitness Function

```
foreach patient in trainingData do
    recordedDecision = trainingData.dischargeDecision(patient);
    predictedDecision = individual.mainDecisionTree.predict(patient);
    if predictedDecision != recordedDecision then
        numErrors++;
end
return numErrors;
```

4 Selection Method

As per the approach that did not incorporate modularization, tournament selection [5] was used as the selection method for all genetic operators. To expand T individuals are selected entirely at random from the existing population. Then, from this set of individuals (ie. the *tournament*) the fittest individual is selected. The size, T, of the tournament was 48. If genetic operators required more than one individual from the existing population, the method was reapplied to select each individual.

5 Genetic Operators

Similar to the genetic programming approach that did not incorporate modularization, the genetic operators used for this assignment included reproduction, crossover and mutation. Reproduction followed the textbook definition [6] in that selected individuals were copied to the new population as is. Mutation consisted of selecting a tree from the selected individual; then, selecting a random node in that tree; and, finally, replacing the subtree rooted at that node in its entirety by a randomly-generated tree. Crossover selects the same tree from each of 2 selected individuals; then, a node is selected in each tree; and, finally, the two subtrees rooted at those nodes are swapped.

In terms of probabilities associated with genetic operators, the same probabilities as the approach that did not incorporate modularization were used. Specifically, the probability for crossover was 0.8, the probability for mutation was 0.1 and the probability for mutation was also 0.1. In addition, when applying operators decisions must be made regarding whether to select internal or leaf nodes. The probability for selecting a leaf node was chosen as 0.1 whereas the probability for selecting an internal 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 try to produce a valid individual up to 10 times before proceeding to the application of the next operator.

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.

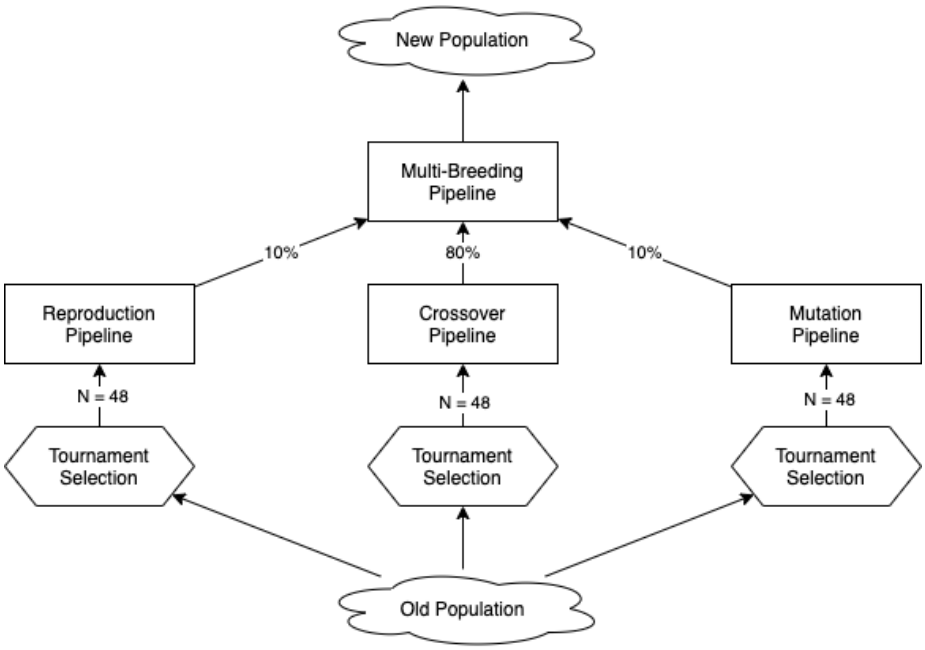

Figure 2: Breeding Pipeline

6 Termination Criteria

Termination criteria are necessary to ensure a finite amount of time before a solution is produced. For this assignment - as was done for the approach that did not incorporate modularization - 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 discharge decisions for all patients in the training set correctly. The maximum number of generations allowed was chosen as 256. Notably, in the tests that were run the success predicate was never attained.

7 Results

Firstly, the machine used for development and producing results was a 2015 15-inch Macbook Pro with a 2.5 GHz Quad-Core Intel Core i7 processor, 16GB of memory, 500GB of storage and a battery cycle count of 704.

As has been mentioned, the data set used for training and testing was randomly split into a training set of 70 patients and an evaluation set of 20 patients. To run tests a script was written which would run the algorithm on the training set with 10 different sets of seeds whilst also testing the best individuals against the evaluation set. The script is available in the github repository for this project (available at `https://github.com/marcus-bornman/cos_710_assignment_3`). The results for the 10 test runs can be seen in table 2 while the best, average and standard deviation of the accuracy for training and accuracy for testing are summarized in table 3.

Run	**Correct Training Classifications**	**Correct Evaluation Classifications**	**Correct Overall Classifications**
1	55	13	68
2	50	16	66
3	59	11	70
4	57	15	72
5	55	16	71
6	53	16	69
7	60	12	72
8	56	11	67
9	55	14	69
10	58	15	73

Table 2: Test Results

	Average Accuracy	**Best Accuracy**	**Standard Deviation**
Training Set	79.71%	85.71%	3.99%
Evaluation Set	69.50%	80.00%	9.60%
Overall	77.44%	81.11%	2.43%

Table 3: Classification Accuracy After Modularization

8 Discussion

For the previous genetic programming approach that did not incorporate modularization, results were compared to those in the existing literature [7, 8, 9] and it was noted that - while additional research is necessary to explore the application of a GP approach using a larger data set - the classifiers produced by the genetic programming approach performed relatively well. For this assignment, modularization - in the form of automatically defined functions - has been added to the genetic programming approach to investigate its effect in terms of accuracy, computational effort and structural complexity.

When compared to the results of the approach that did not incorporate modularization, the classifiers produced after adding modularization showed similar results in terms of accuracy. For instance, the best accuracy achieved on the training set before modularization was 87.14% and, after modularization, this decreased slightly to 85.71%. On the evaluation set, however, the best accuracy improved from 75% to 80% - indicating that introducing modularization may lead to better generalization.

In terms of computational effort, since no solutions were able to classify the data with a 100% accuracy, it is unfortunately not possible to calculate how many programs have to be evaluated before a solution can be found. As an alternative for determining affects on computational effort, the runtimes for 10 test runs using the genetic programming approaches with and without modularization were recorded and are shown in figure 3. As can be seen, runtimes for the approach that incorporates modularization are consistently shorter than the previous approach. Considering the approaches used the same genetic algorithm parameters, this is a strong indicator that introducing modularization has decreased computational effort.

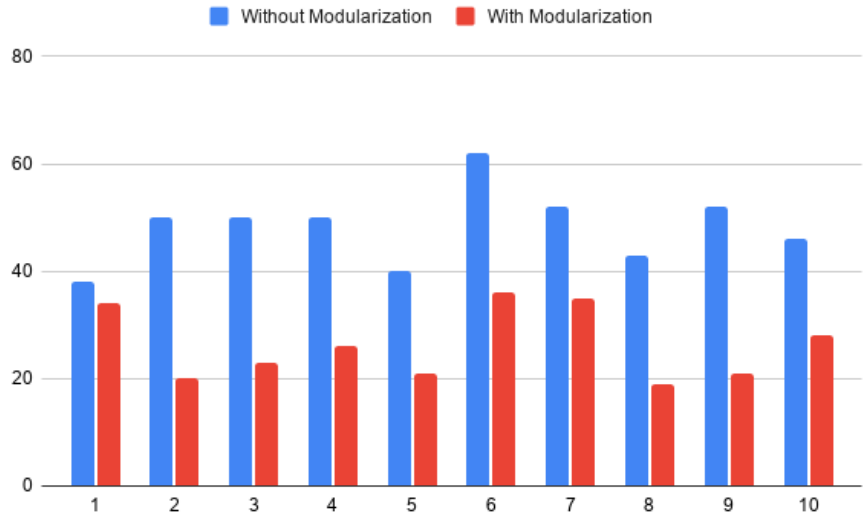

Figure 3: Runtimes for 10 Test Runs With and Without Modularization

Finally, in considering the structural complexity of individuals produced by either approach, we consider the number terminal and function nodes that individuals are composed of. For the approach that did not incorporate modularization, the fittest individuals for each of 10 test runs had an average of 48.11 terminal and function nodes. For the approach that did incorporate modularization, this average was 36.03. Hence, it would seem that the approach using modularization evolves towards less complex individuals. However, since only the best individual per test run was considered in this calculation, no conclusions are drawn regarding the complexity of individuals during the evolutionary process.

In conclusion, expanding the previous genetic programming approach to produce a classifier for postoperative patient diagnosis by adding modularization led to similar results in terms of accuracy and improvements in both computational effort and structural complexity.

References

[1] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, J. Bassett, R. Hubley, and A. Chircop, "Ecj: A java-based evolutionary computation research system," *Downloadable versions and documentation can be found at the following url: http://cs. gmu. edu/eclab/projects/ecj*, vol. 880, 2006.

[2] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[3] W. S. Bruce, "The application of genetic programming to the automatic generation of object-oriented programs," 1995.

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

[5] 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.

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

[7] A. B. Owen, "Tubular neighbors for regression and classification," 1999.

[8] P. Luukka, "Pca for fuzzy data and similarity classifier in building recognition system for post-operative patient data," *Expert systems with applications*, vol. 36, no. 2, pp. 1222–1228, 2009.

[9] S. R. Dash and S. Dehuri, "Comparative study of different classification techniques for post operative patient dataset," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, no. 5, pp. 1101–1108, 2013.