CSC3423 – Biocomputing
Practical1: Genetic Programming

The aim of this practical is to familiarise yourself with the working of a basic genetic programming algorithm. The software you will use is an open-source genetic programming framework called EpochX (http://www.epochx.org/).

Downloading the files for the practical

A compiled version of EpochX plus other relevant files for the tutorial is in Blackboard in Teaching Materials > Practicals > Practical2 > Code and config file of EpochX for practical 2. Download and unzip the file.

Within you will see five files:

- epochx-1.4.1.jar ✉ The jar file contains the whole EpochX framework
- commons-lang-2.5.jar ✉ Library dependency for EpochX
- Mux6.java, Mux6.class, Mux6$1.class ✉ they specify the fitness function of the test dataset (6-bit multiplexer) and the list of possible nodes and leaves for the genetic programming trees

Test problem

The 6-bit multiplexer is a toy classification dataset very popular to evaluate evolutionary machine learning methods. A multiplexer (http://en.wikipedia.org/wiki/Multiplexer) is a digital circuit that has several inputs and one output. The inputs are split in two parts: the address bits and the data bits. The digital circuit will copy the value of one of the data bits to the output. Depending on the values of the address bits a particular data bit will be chosen. The aim of the classification problem is to learn the truth table of the digital circuit. In particular you will be using the 6-bit multiplexer that has two address bits and four data bits. Given that all possible combinations of inputs for the circuit are present, the dataset has 64 instances.

Running the basic example

To run EpochX for this example, from the command line simply run:

```
java -cp epochx-1.4.1.jar:commons-lang-2.5.jar:. Mux6
0       15.0    OR(AND(OR(IF(AND(OR(d1 d0) NOT(d3)) IF(IF(d0 d0 d0) IF(d3 a1 a0)
IF(d2 d0 d1)) IF(OR(d1 d3) IF(a1 d0 a1) OR(d1 d0))) NOT(NOT(OR(d0 d1))))
OR(AND(IF(AND(d3 d1) AND(a0 d2) OR(d0 d1)) AND(AND(d1 a0) IF(a0 d0 a0)))
AND(OR(AND(a0 d0) NOT(a0)) IF(NOT(d3) AND(d1 a1) OR(a1 d2)))))
AND(OR(AND(IF(NOT(a1) OR(a1 d1) AND(d3 d1)) OR(OR(a0 a0) IF(d0 d1 d3)))
```

		NOT(NOT(AND(a1 d1)))) AND(NOT(AND(IF(d3 d3 d1) AND(a1 a0))) OR(IF(IF(d2 d2 d3) NOT(d0) NOT(d0)) NOT(AND(d2 d2))))))
1	14.0	NOT(NOT(IF(IF(OR(IF(d2 d1 a0) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(AND(d2 d3) OR(d3 d2))) AND(AND(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(d0 a1) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 d0))))))
2	12.0	IF(NOT(IF(AND(AND(OR(d3 d1) IF(d2 a0 d1)) AND(NOT(d3) AND(d1 d3))) NOT(NOT(AND(a0 d1))) AND(OR(OR(d3 AND(OR(d0 a1) OR(d1 d0))) IF(d1 a0 a1)) NOT(IF(a1 d2 a0))))) NOT(NOT(OR(OR(OR(d1 d1) NOT(a0)) NOT(OR(d2 a0))))) IF(NOT(IF(IF(OR(d2 a1) IF(d0 d0 a0) OR(d2 d0)) NOT(NOT(d3)) AND(NOT(a0) AND(a1 d0)))) AND(IF(AND(AND(d1 d3) IF(d1 a1 d1)) OR(AND(a1 d2) OR(d3 d0)) NOT(IF(a0 d3 d0))) NOT(IF(NOT(a1) AND(a0 a0) NOT(d0)))) NOT(AND(IF(NOT(d2) IF(a0 d2 d0) NOT(d1)) OR(AND(a0 a1) IF(a0 a1 a0))))))
3	10.0	NOT(NOT(IF(IF(OR(IF(d2 d1 a0) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(AND(d2 d3) OR(IF(AND(d3 d1) d0 d1) d2))) AND(AND(IF(OR(OR(a1 AND(a1 a0)) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(d0 a1) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 d0))))))
4	6.0	NOT(NOT(IF(IF(OR(IF(d2 d1 a0) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(AND(d2 d3) OR(IF(AND(d3 d1) d0 d1) d2))) AND(AND(IF(OR(OR(a1 AND(a1 a0)) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(d0 a0) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 d0))))))
5	6.0	NOT(NOT(IF(IF(OR(IF(d2 d1 a0) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(AND(d2 d3) OR(IF(AND(d3 d1) d0 d1) d2))) AND(AND(IF(OR(OR(a1 AND(a1 a0)) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(d0 a0) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 d0))))))
6	4.0	NOT(NOT(IF(IF(OR(IF(d2 d1 a0) OR(d2 d1)) OR(OR(a0 a0) NOT(a1)) IF(d2 d3 a0)) AND(AND(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) a1) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 AND(d2 IF(a1 NOT(a1) d0))))))))
7	4.0	NOT(NOT(IF(IF(OR(IF(d2 d1 a0) OR(d2 d1)) OR(OR(a0 a0) NOT(a1)) IF(d2 d3 a0)) AND(AND(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) a1) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 AND(d2 IF(a1 NOT(a1) d0))))))))
8	3.0	NOT(NOT(IF(IF(OR(IF(d2 d1 OR(d0 a0)) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(AND(d2 d3) OR(IF(AND(d3 d1) d0 d1) d2))) AND(AND(IF(OR(OR(a1 AND(a1 a0)) NOT(a0)) d0 d1) OR(d3 a0)) AND(OR(d0 a0) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 d0))))))
9	3.0	NOT(NOT(IF(IF(OR(IF(d2 d1 OR(d0 a0)) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(AND(d2 d3) OR(IF(AND(d3 d1) d0 d1) d2))) AND(AND(IF(OR(OR(a1 AND(a1 a0)) NOT(a0)) d0 d1) OR(d3 a0)) AND(OR(d0 a0) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 d0))))))
10	2.0	NOT(NOT(IF(IF(OR(IF(NOT(NOT(OR(IF(d3 a0 d2) IF(d0 d0 d0)))) d1 a0) OR(d2 OR(d1 IF(OR(IF(d2 d1 a0) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(IF(d2 a0 a1) IF(a1 d2 NOT(OR(d3 a0)))))))) OR(OR(a0 a0) NOT(a1)) IF(d2 d3 a0)) AND(AND(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(d0 IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1)) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 AND(d2 IF(d2 d1 a0))))))))
11	2.0	NOT(NOT(IF(IF(OR(IF(NOT(NOT(OR(IF(d3 a0 d2) IF(d0 d0 d0)))) d1 a0) OR(d2 OR(d1 IF(OR(IF(d2 d1 a0) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(IF(d2 a0 a1) IF(a1 d2 NOT(OR(d3 a0)))))))) OR(OR(a0 a0) NOT(a1)) IF(d2 d3 a0)) AND(AND(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(d0 IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1)) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 AND(d2 IF(d2 d1 a0))))))))
12	2.0	NOT(NOT(IF(IF(OR(IF(NOT(NOT(OR(IF(d3 a0 d2) IF(d0 d0 d0)))) d1 a0) OR(d2 OR(d1 IF(OR(IF(d2 d1 a0) OR(d2 a0)) OR(OR(a0 a0) NOT(a1)) OR(IF(d2 a0 a1) IF(a1 d2 NOT(OR(d3 a0)))))))) OR(OR(a0 a0) NOT(a1)) IF(d2 d3 a0)) AND(AND(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(d0 IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1)) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 AND(d2 IF(d2 d1 a0))))))))

```
13      1.0      NOT(NOT(IF(IF(d1 OR(OR(a0 a0) NOT(a1)) OR(AND(a0 d3) IF(d2 d1
a0))) AND(AND(IF(OR(OR(a1 a1) AND(a1 AND(a1 d1))) d0 OR(IF(OR(OR(a1 a1)
AND(a1 d1)) d2 d1) a1)) OR(d3 a0)) AND(IF(IF(OR(AND(IF(d1 AND(a0 d2) OR(d0
d1)) AND(AND(d1 a0) IF(a0 d0 a0))) OR(NOT(IF(IF(OR(IF(d2 d1 OR(d0 a0)) OR(d2
a0)) OR(OR(OR(AND(d2 d3) OR(d3 d2)) a0) NOT(a1)) OR(AND(d2 d3) OR(IF(d3 d0
d1) d2))) AND(AND(IF(OR(OR(a1 AND(a1 a0)) NOT(a0)) d0 d1) OR(d3 a0))
AND(OR(d0 a0) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 d0))))) a0)) OR(OR(a0
a0) NOT(a1)) IF(a1 a0 d1)) AND(AND(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) OR(d3
a0)) AND(OR(a0 d1) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 AND(d2 IF(a1
NOT(a1) d0)))))) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0 NOT(OR(d3 a0))))))))
14      0.0      NOT(NOT(IF(IF(OR(IF(NOT(NOT(OR(IF(d3 a0 d2) IF(d0 d0 d0)))) d1
a0) OR(IF(a1 d2 NOT(OR(d3 a0))) OR(d1 IF(OR(IF(d2 d1 a0) OR(d2 a0)) OR(a0 a0)
OR(IF(d2 a0 a1) IF(a1 d2 NOT(OR(d3 a0)))))))) OR(OR(a0 a0) NOT(a1)) IF(a1 d0
d1)) AND(AND(IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1) OR(d3 a0)) AND(OR(d0
IF(OR(OR(a1 a1) AND(a1 d1)) d0 d1)) OR(d1 d0))) NOT(OR(IF(d2 a0 a1) IF(a1 a0
AND(d2 IF(d2 d1 a0))))))))
```

Each line of output contains three parts: The iteration number, the fitness of the best individual (lower is better) and the tree definition of the best individual. In this particular example, an optimal solution (that classifies correctly all 64 instances is found at iteration 14. You will also see that the tree for the best solution is quite complex and hard to read.

Looking at the source code of Mux6.java

If you open Mux6.java you will see a nice example of how to use the EpochX framework. This file creates a class (Mux6) that derives from GPModel. GPModel is the interface that EpochX specifies for the genetic programming problems that can be plugged to the framework.

In the constructor of the class (lines 32-60) you will see first how to tell the framework about the variables of the problem and about the internal nodes of the tree. The last two lines are used to generate the data of the 6-bit multiplexer dataset.

Afterwards, the getFitness function (lines 63-84) evaluates individual trees. The function receives one GPCandidateProgram, and iteratively tests it on all instances of the dataset (contained in the inputs array). The setValue function of the Variable class is used to pass the value of the variables from the instances to the tree. Next thee tree is run and its result is compared to the correct output (outputs array), and the number of correctly predicted instances is computed.
The last line returns the fitness value of the tree, the number of misclassified examples.

Next, getReturnType specifies the result that the tree produces, Boolean variables. The generateOutputs auxiliary function is used to compute the right output for a given instance of the 6-bit multiplexer problem.

Finally, the main function is the one that sets up the parameters of the experiment (lines 118-125), then specifies what data will be printed at each iteration of the evolutionary cycle (lines 127-131) and finally runs the algorithm (line 134).

Changing parameters of the algorithm

You will need to re-compile the code to change the functioning of the algorithm (e.g. using Eclipse). Some of the things you can try are the following:

- Change the parameters of the system (population size, number of iterations, probabilities, tournament size).
- Change the set of Boolean operators of the trees (lines 43-46). The complete set of Boolean operators of the framework is at http://www.epochx.org/javadoc/1.4/org/epochx/epox/bool/package-summary.html
- Add a parsimony term to the fitness function to control bloat. Go to line 83 of the file and activate the commented code. Change the weight parameter (0.1) to something larger of smaller and see what effect it has. Combine playing with the bloat weight with the set of Boolean operators until you can find the shortest possible program.

Relation between this example and the coursework's problem.

Like the coursework assignment, here we are solving a machine learning and particularly a classification problem. The main difference is that in here there is no data file (as the examples are generated by the code in Mux6) and the definition of the problem (number of variables) is hard-coded. In the API of the coursework's framework you got functions that tell you the name, type and number of the attributes of the problem (classes Attributes and Attribute) and you got functions that give you the instances of the dataset (InstanceSet and Instance). Moreover, the dataset has continuous variables rather than Boolean ones. This means that both the set of terminals and operators of the tree would need to change. Take a look at the other GP examples of the framework (http://www.epochx.org/javadoc/1.4/index.html?org/epochx/gp/model/package-summary.html). The whole source code of the framework is available for download here: http://www.epochx.org/downloads.php.