

Fall 2011
Instructor: B. Ross

COSC5P71 GP: Assignment 1

Due date: Tuesday, October 4, start of class.
Lates: Friday, October 7, 9:00am (-25% deduction).
Goal: Introduction to genetic programming.

Hand in: A report about your experiments. Printed copies of your source and parameter files (only the code you have written or edited; please do not include “kernel” system source code). Use Excel to create performance graphs, to be included in your report. Also submit electronic copies of all data and report files (use “submit5p71” on sandcastle).

System: You are free to use any GP system you want (lilGP, Open Beagle, ECJ, etc.). The system that will be discussed in class is lilGP 1.1. This is a C-based GP system with the same functionality as Koza’s GP system in his first book. It also has many additional enhancements (typing, multiple populations, etc.). ECJ and Beagle are preferable to lilGP, however, as they are more contemporary, better maintained, have more features, and are better documented. See the end of the assignment for the location of these systems.

A. Symbolic regression

This question will introduce you to genetic programming. You are to take an existing symbolic regression example included with your GP system, compile it for the platform you are using, and execute it to get some results. The lilGP, ECJ, and Open Beagle systems all include this example. Once you get it running, modify the training initialization setup, by reading in the training points from a text file. You should have the total number of points to process and the name of the text file as parameters in the “input file” user parameter file. This will let you run your system on new data sets, without having to recompile the GP system. (Note that this modification will be useful for part B below).

Try the system a few times, on new data you have put into a new training file. Also try it on a few variations of languages. For example, try it on the default language, as well as a “weaker” language (perhaps only plus, minus, and multiply).

Decide on 2 interesting comparative experiments to do, using the same training data. You might compare the GP language, or selection strategy, or fitness evaluation scoring, or.... Do 20 runs using each of these experiments. Your report should compare these two experiments. The average of all the population averages and “best” at the end of each run should be computed for each experiment, and put into a table. Performance graphs showing the overall run averages of the population average and best per generations should also be plotted in Excel, and included.

B. Pima Indians Diabetes Data Set

The Pima Indians Diabetes data set is a well known challenging pattern recognition problem from the UCI Machine Learning Repository:

<http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes>

You are to use genetic programming to evolve a rule that predicts whether a patient has diabetes. The data set has 768 cases, all with the following numeric attributes:

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (μ U/ml)
6. Body mass index ($\text{weight in kg}/(\text{height in m})^2$)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

The Class variable (9) is treated as a boolean: 0 (false), 1 (true – tested positive for diabetes).

Here are two rows from the database:

```
3, 78, 50, 32, 88, 31.0, 0.248, 26, 1
10, 115, 0, 0, 0, 35.3, 0.134, 29, 0
```

The second example has some missing data (0's), and you may wish to remove such examples from consideration. Your GP program will evolve a decision expression that uses the first 8 attributes, to determine the 9th class. (Of course, don't give your GP program access to the 9th value, because it is the solution!). Since all the values are numeric, it is not necessary to use strongly typed GP. However, strong typing can still be useful.

Here is a recommended approach to follow.

1. Read the database into a large table, one row per example. You should randomly shuffle the rows. Then split the table into 2 independent sets of examples – a training set, and a testing set. The exact size of these sets is up to you; you might initially try a 50-50 split. Be aware that machine learning experiments are usually affected by the size of training sets. For example, sometimes smaller training sets are more effective. However, with this problem, you probably have the same number of training examples for true and false cases. Note that there are twice the number of false cases, compared to true cases, in the data set.
2. Define a set of language primitives to be used by GP. These primitives should work sensibly on the input data. A possible set of primitives to consider are:
 - a. functions: arithmetic operators, relationals (less-than, greater-or-equal-to), min and max, if-then-else,...
 - b. terminals: attribute variables (8), and ephemeral random constants.

3. The top-level wrapper that executes each classifier program works as follows. The program will try to predict whether a given case is positive for diabetes or not. The 8 attribute values, from a row in the table, are copied into a set of 8 variables accessible by the GP program primitives, and which will take the form of leaf terminals within the tree. The program is executed by the GP system interpreter using these values. The resulting answer is recorded. The fitness function will then compare this answer to the known solution from the example table, and the fitness score will be appropriately updated (see next point). So in other words, the following pseudo-code shows the general execution strategy:

```
float numpreg, plasma, thick, ...;
int ans;

loop for i=0 to N-1 : // N training examples from table
    numpreg = training[i][0];
    plasma = training[i][1];
    thick = training[i][2];
    (... etc.)
    ans = execute_tree(t); // t is the current GP tree
// now the value of ans can be compared to the real answer,
// perhaps stored in integer array ans[i].
```

4. Fitness is based upon the number of correct classifications. It might simply be a count of the number of correct classifications over the N training examples. A score of N is perfect. During the run, if you ever find a GP expression that reaches N, you can terminate the run at that point, because it can't do better than that.
5. At the end of the run, you must cross-validate (test) the best solution obtained on the testing set (ie. all the examples not used for training). This gives an idea of the generality of your evolved solution when given new examples it has never seen. Record the test performance of each run. This information will be included in the report.
6. Do multiple runs (perhaps 20), each with different random number seeds and shuffled data sets. Collate the performance of your runs together. Your report should have a table summarizing all the runs, for both the training scores and testing scores. You should report the average best solution scores, as well as the scores for the overall best solution from all runs. Include a *confusion matrix* with each experiment. Also include performance graphs showing the population fitness and best fitness plotted over generations (averaged over for all the runs).
7. Your report should describe all relevant aspects of your experiment. **The goal of the report is to give enough information so that someone else could reproduce your experiments.** Describe your problem set (and its source), and describe what your GP program is attempting to do. Be sure to describe your GP language, all GP parameters, and your fitness evaluation methodology. Include source code of programs for your 2 best rules obtained. If they are large, they can be put into an appendix of your report. Include a discussion on your results. Tables and graphs do not necessarily speak for themselves! Also include a conclusion section that summarizes the strengths or weaknesses of your experiment.

The report format should therefore be:

1. Introduction: problem description
2. Experiment details:
 - a. parameters for GP
 - b. fitness function
 - c. GP language
 - d. format of data
 - e. variations of experiments
3. Results
 - a. tables of results (best, avg, etc.)
 - b. graphs
4. Analytical discussion of results
5. Conclusions
6. Bibliography (if necessary)

Comments: Run your experiments multiple times with different random number seeds. For those using Lilgp, it creates a lot of data output, not all of which will be useful to you. Feel free to experiment with different GP parameter settings for the experiments. Use the Java-based lilgpMonitor utility to observe the performance of your runs.

Be sure to include reasonable inline comments in your program code!

Some relevant folders and web sites:

ML Repository and Pima diabetes data:

<http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes>

Location of lilGP software:

www.cosc.brocku.ca/Offerings/5P71/

(also see BST lilgp – a strongly-typed lilgp enhanced and debugged by Brock students; windows-ready).

Open Beagle: **<http://beagle.gel.ulaval.ca/index.html>**

ECJ: **<http://www.cs.umd.edu/projects/plus/ec/ecj/>**

Confusion matrix:

http://en.wikipedia.org/wiki/Confusion_matrix