GECCO 2004

Tiny GP Contest Results

back to GECCO contests

Introduction

The aim of this competition is to implement the tiniest possible GP system capable of doing symbolic regression with the following characteristics:

a. terminal set includes floating point variables X1 to XN (with N<=10)

b. function set includes multiplication, protected division, subtraction and addition.

c. must read the fitness cases from file format given below

d. the system can be either generational or steady state. In the latter case a "generation" is to be interpreted as POPSIZE (see below) crossover/mutation events. Also, negative tournament should be used for the selection of the individuals to be replaced at each steady-state-GP iteration.

e. standard (Koza-style) subtree crossover must be implemented.

f. point mutation (that is, GA-like mutation, not subtree mutation) must be implemented.

g. tournament selection.

h. The following parameters must be implemented (but can be fixed at compile time if necessary) and printed when each run starts:

- MAX_LEN // the maximum length any GP program can take
- POPSIZE // the size of the population
- DEPTH // the maximum depth initial programs can have (0 represents just one terminal)
- GENERATIONS // the maximum number of generations allowed for a run
- CROSSOVER_PROB // the probability of crossing over (mutation probability will be 1 - CROSSOVER_PROB)
- PMUT_PER_NODE // the mutation probability per node (when mutation has been chosen as the variation operator)
- TSIZE // tournament size

i. the fitness function must be related to the sum of the absolute differences between the actual program output and the desired output for each fitness case.

j. The system must be able to handle up to 10 variables and up to 1000 fitness cases.

k. The initialisation method should be "Grow"

l. At each generation the following statistics/data should be calculated and printed:

- Generation number
- Average fitness of the individuals in the population.
- The fitness of the best individual in the population.
- The total number of primitives evaluated so far during evolution.
- The average program size in the current generation.
- The best individual in the population should be printed.

m. The random number generator will need to be seeded. It must be possible to do this from command line.

If the command line parameter is absent, the system must use the current time to seed the random number generator.

n. The name of the file containing the fitness cases must be passed to the system. It must be possible to do this from command line. If the command line parameter is absent, the system should assume the data are stored in the current directory in a file called "problem.dat".

o. If the total error made by the best program goes below 1.0e-5 the program should stop printing a message indicating success.

p. If the problem has not been solved after the maximum number of generations, the program should stop printing a message indicating failure.

q. A short document (in PDF format, max 3 pages) should accompany the submission of the source code to explain how this works, how to execute it, how to change the parameters, etc.

Winning Criteria (not ordered)

1. Clarity of implementation and accompanying documentation.
2. Readability and formatting of the code.
3. Number of lines of code.
4. Source file size.
5. Size of compiled version of program (if any) under gcc or Java 2 (v 1.4).
6. Memory footprint when running.
7. Degree to which the requirements indicated above have been met.

Judging Panel

The entries will be evaluated according to the above criteria by a panel of expert judges (t.b.a).

Input File Format

We use a plain ASCII file format.

```
The format is

N               // an integer representing the number of variables
NFITCASES       // an integer representing the number of fitness cases
                // must be <= 1000
FITNESSCASE1    // The fitness cases (one per line)
FITNESSCASE2
FITNESSCASE3
....

Each fitness case is of the form

X1 ... XN TARGET

where X1 to XN represent a set of input values for a program, while
TARGET represents the desired output for the given inputs.
```

Sample training set:

```
3
5
0.1     0       1       -1.2
0       1.3     0.1     0
1       0       -1      2.1
1.3     1.2     1       2
2       2       1       11
```