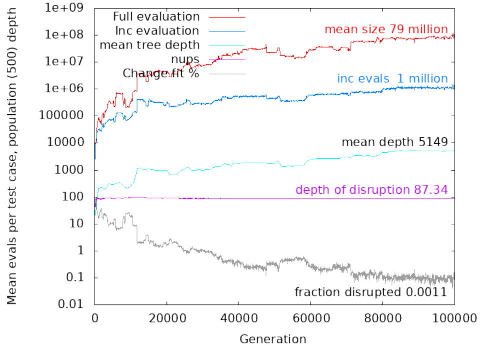
CREST

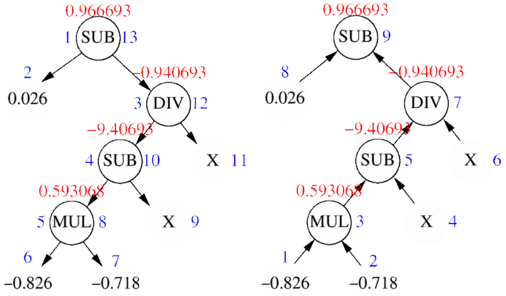
EPSRC

Incremental Evaluation in Genetic Programming

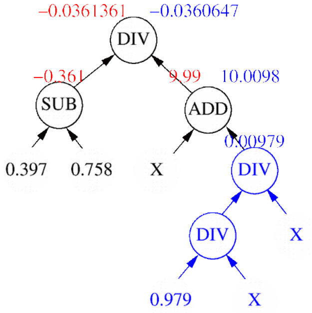
W. B. Langdon Department of Computer Science, University College London



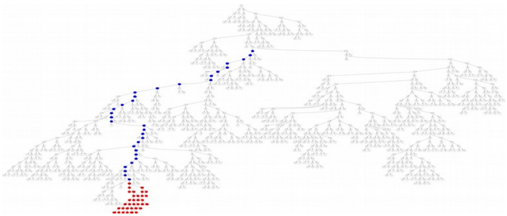
Convergence of sextic polynomial. Grey % parent's !=child's fitness. After gen 800 most children have identical fitness to mum, and on average (purple dotted line) incremental evaluation evaluates subtrees of depth≈100. The max saving in evaluation of GP operations on traditional (top red v. dashed blue) is 100 fold. Note log scale.



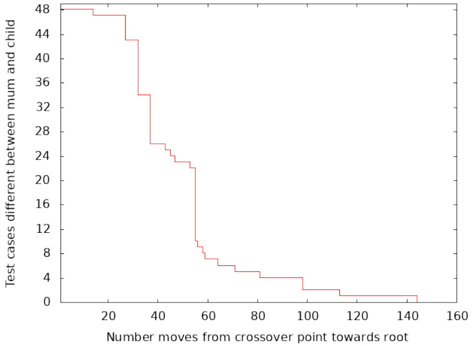
Left: Conventional top-down recursive evaluation of (SUB 0.026 (DIV(SUB (MUL -0.826 -0.718) X) X)). X=10. Blue integers indicate evaluation order, red floats are node return values. **Right**: an alternative ordering, starting with leaf -0.826 and working to root node. Both return exactly the same answer.



Incremental evaluation of fragment of child produced by crossover. Inserted subtree (DIV (DIV 0.979 X) X) in blue. Nodes common to both mum and child in black. Red floats (left) are mum node return values. Blue floats (right) are node return values in the child. (In both X=10.0). Notice (SUB 0.397 0.758) is not affected by the crossover and has the same value in parent and child and so is evaluated only once per test case.



New code in red. Disrupted nodes in blue. Crossover's change is not visible externally.



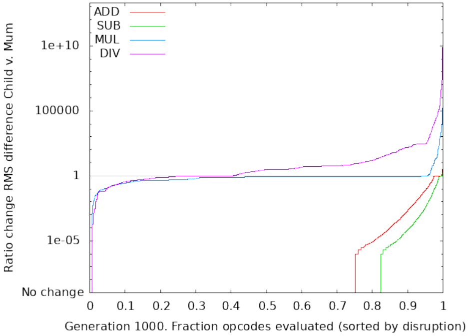
Incremental evaluation of first member of generation 1000. Number of test cases where evaluation in the root donating parent (mum) and its offspring are identical never falls.

Terminal set:	X, 250 constants between -0.995 and 0.997
Function set:	MUL ADD DIV SUB
Fitness cases:	48 fixed input -0.97789 to 0.979541 (randomly selected from -1.0 to +1.0).
	Target $y = xx(x-1)(x-1)(x+1)(x+1)$
Selection:	Tournament size 7 with
	fitness $= \frac{1}{48}\sum_{i=1}^{48} \lvert GP(x_i) - y_i \rvert$
Population:	500. Panmictic, non-elitist, generational.
Parameters:	Initial population ramped half and half [1] depth between 2 and 6.
	100% unbiased subtree crossover. At least 1000 generations

DIV is protected division `(y!=0)? x/y : 1.0f`

Equivalent of 571 billion GP operations per second on goldeneye, 16 core 3.8Ghz i7 with AVX-512
Cf. Koza 0.5 petaflop per day

C++ code
http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/gp-code/GPinc.tar.gz



Ratio of difference between mum and child before and after each function at generation 1000. Most linear functions (i.e. ADD and SUB) do not change difference. On average 86% MUL and 56% DIV decrease difference. Note log scale

INFORMATION THEORY EXPLAINS GP CONVERGENCE

All functions loose information. Without side effects, lost information cannot be restored. Disruption passes up tree but once lost on a test case cannot be restored. In deep trees impact does not reach root. Hence child behaves identically to its mother and therefore has the same fitness. Deep trees give GP a smooth landscape. Relatively insensitive, O(log n), to number of test cases.

EuroGP 2021, pp229-246, LNCS 12691
https://doi.org/10.1007/978-3-030-72812-0_15

w.langdon@cs.ucl.ac.uk