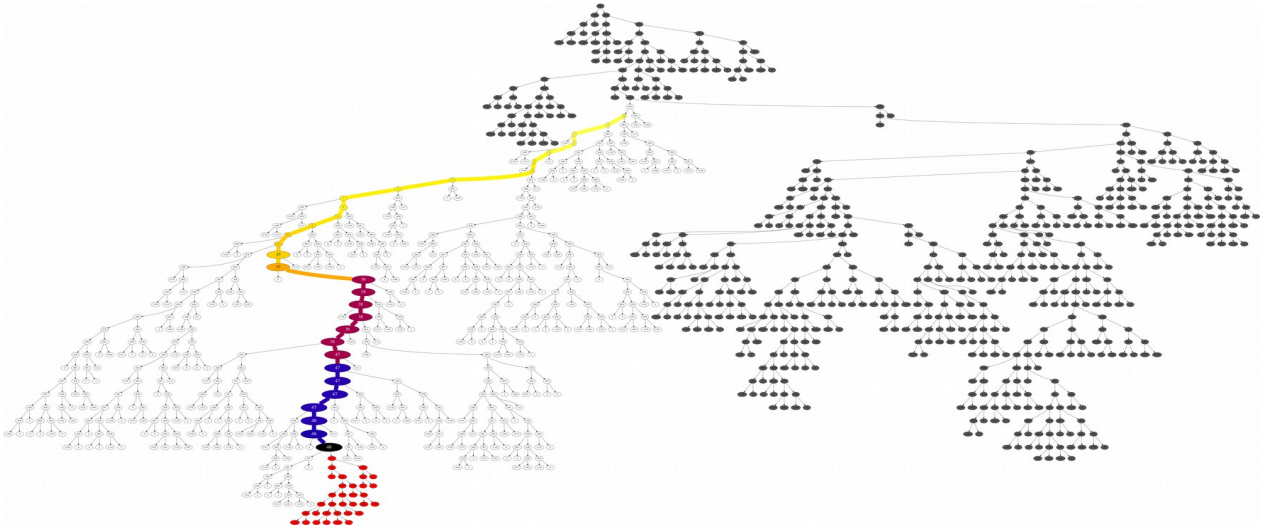
Dissipative Polynomials

Workshop on Landscape-Aware Heuristic Search

GECCO 2021

https://doi.org/10.1145/3449726.3463147

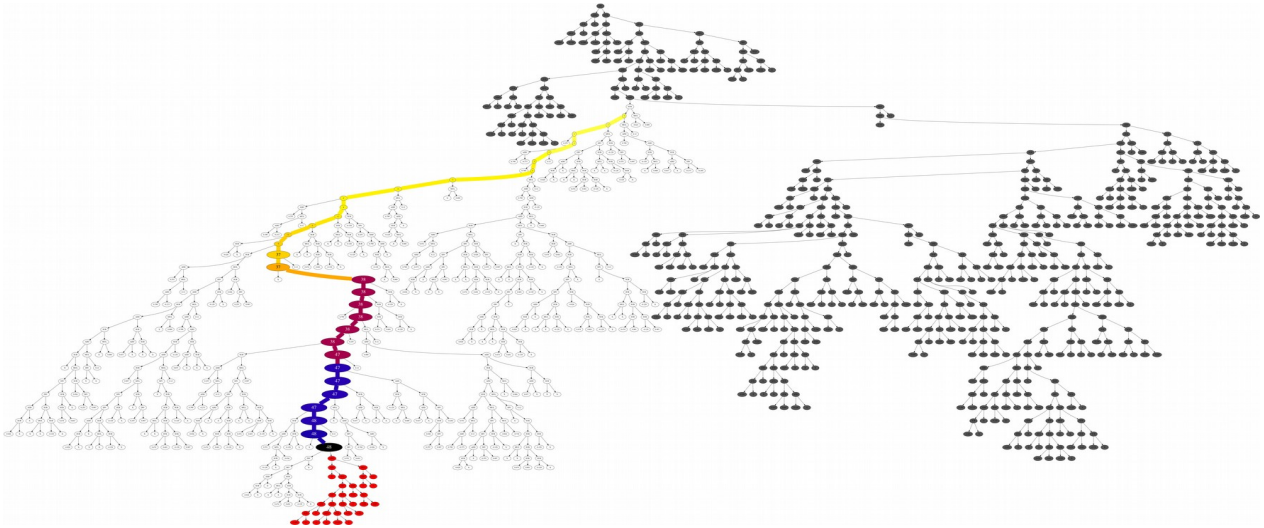
W. B. Langdon



Dissipative Polynomials

Workshop on Landscape-Aware Heuristic Search
GECCO 2021

W. B. Langdon

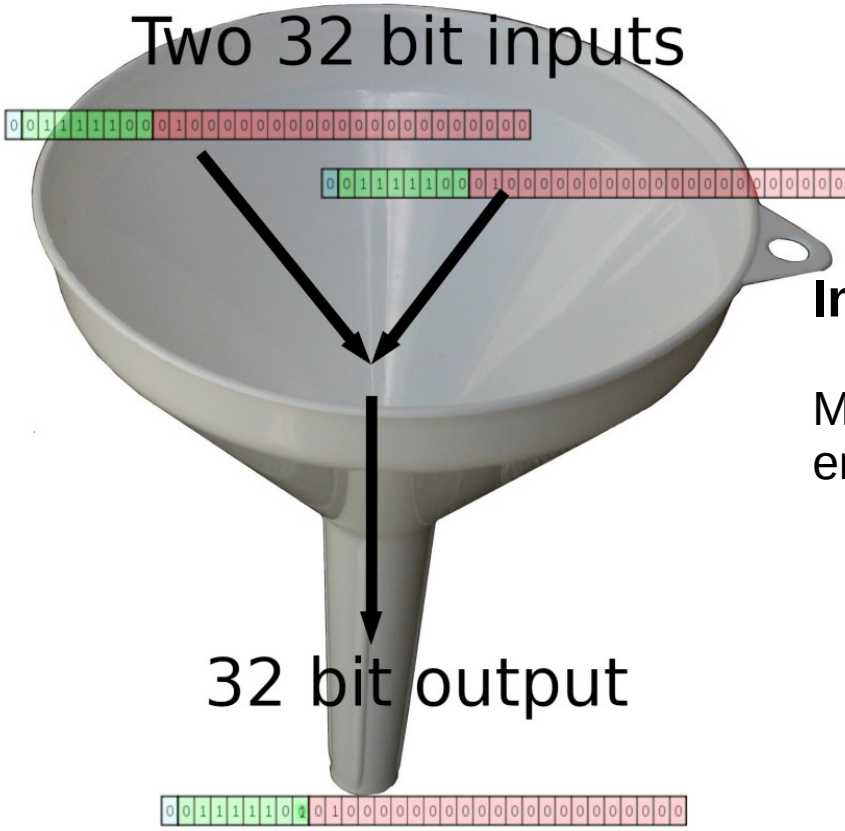


Computation as Information Flow

- Computer operators, eg addition, are irreversible. Meaning input state cannot be inferred from outputs.
  - Information is progressively lost.
  - Functions are dissipative.
- Most information about changes before a long sequence of operations is lost.
- Most errors (run time or syntax) make no difference
- If genetic programming trees are deep enough the GP landscape becomes smooth

Information Funnel

Computer operators are irreversible. Meaning input state cannot be inferred from outputs. Information is lost



Information funnel

More information enters than leaves

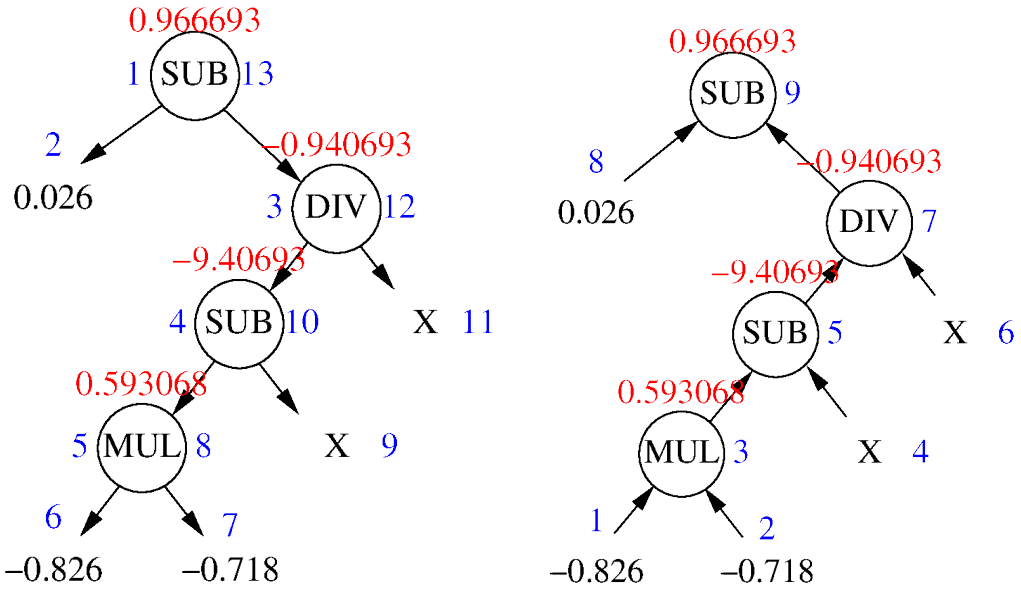
Genetic Programing

- Genetic programming example:
  symbolic regression,
  binary functions (add and multiply),
  for simplicity no (protected) division.
- Addition and multiplication give a polynomial
- Study genetic programming landscape by uniformly sampling
- Simulate impact of GP crossover by random crossovers between large GP trees
- Use incremental evaluation [EuroGP 2021] to trace impact of changes.
- Fitness change (number of different test cases) falls monotonically as we move away from mutation
- Difference in RMS fitness may go up or down with distance from change site but in most cases falls to zero.

Experiments: Genetic Programming

- Ordinary GP with pure functions without side effects
- Same argument holds for functional programming
- Imperative programming (e.g. C, Java) is dominated by side effects (i.e. memory) but suggest that
  - in deeply nested real programs in many (but not all) cases
  - impact of bugs passes through many irreversible steps
  - where its impact is diluted before reaching an observable output (e.g. a print statement)
  - With many steps, bug's impact may be totally invisible
- Without side effects GP tree can be evaluated in any order and the result (at every node) is identical.

Top Down = Bottom Up



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

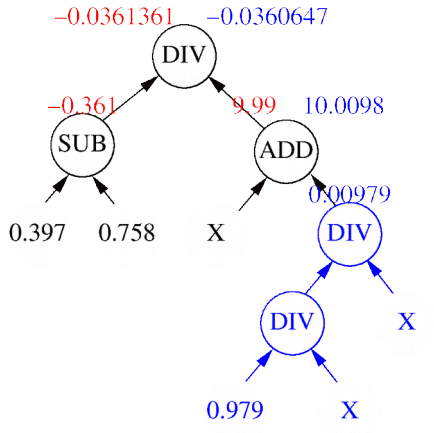
- Mum and child are identical except for mutated or crossed over subtree
- Use same test cases for mum and child
- Fitness evaluation of mum and child are identical except for the changed code and parts of tree which (recursively) calls the changed code.
- Use bottom up evaluation to trace changes along chain of nodes from crossover point to root node
- If on a test case, changed code returns identical value then the calling function will return an identical value
- If mum and child evaluations become identical at any point along calling chain to the root node, they will remain identical to the root node and the mutant's evaluation is identical to that of the original tree.

Genetic difference ≠ phenotypic difference

- If by chance inserted & removed subtrees are identical:
  - mum and child are identical and so have the same fitness
- If inserted subtree evaluates to same value as removed subtree on every test case:
  - mum and child (at root node) evaluate to same value on every test case
  - genetic difference => identical fitness
- What if the inserted subtree evaluates to different values to that given by remove subtree?
  - If we evaluate both child and mum starting at the change, there is a progressive fall in the number of test cases where the change is visible as we move towards the root node.

Evaluate both trees from change up

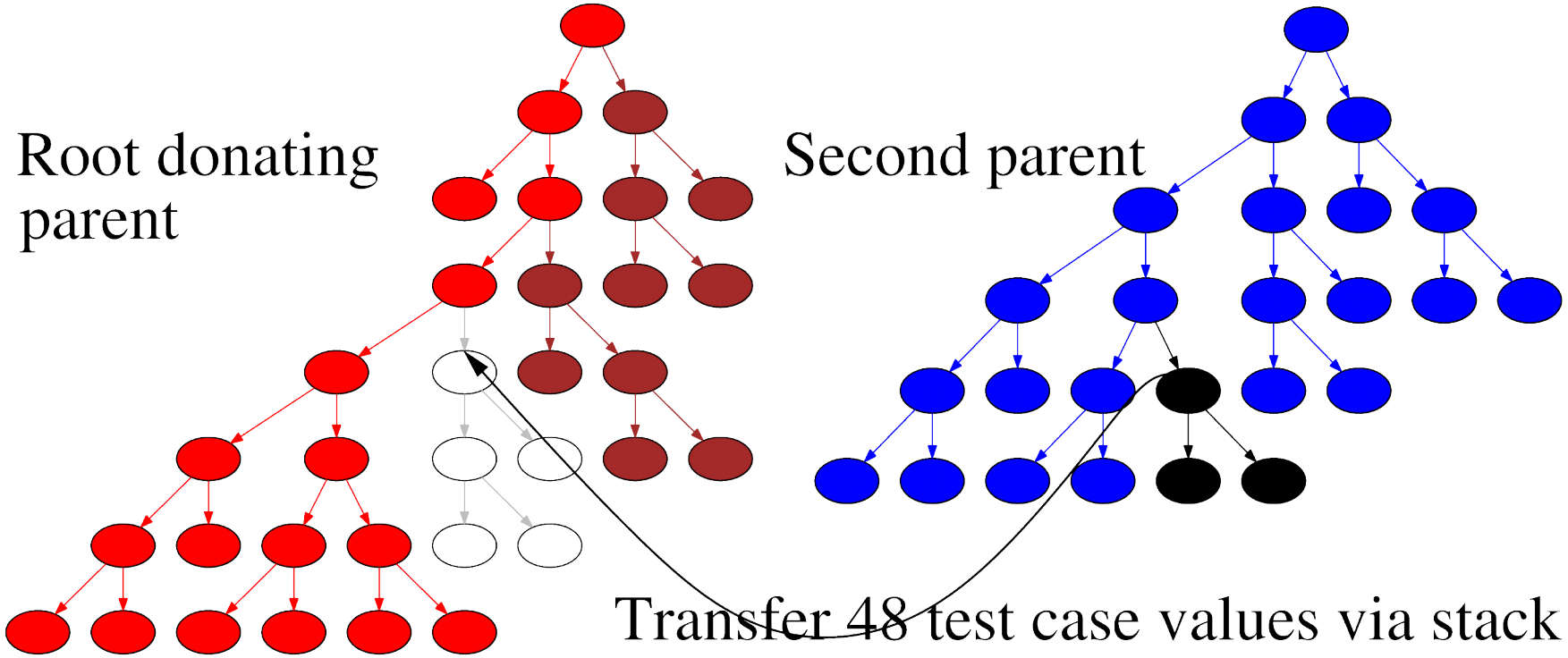
- Mum and child are identical above change, so use mum
- Fitness evaluation is identical except on route from change to root node.
- Evaluate both mum and child up this path.
- If they evaluate to identical values at any point then they evaluate to same value on the rest of the tree, including the root node:
  - semantic difference => identical fitness.



Evaluate mum in red. Evaluate child in blue.
Inserted code (DIV (DIV 0.979 X) X) in blue.
Here incremental evaluation proceeds 38 levels up the child tree before both mum and child evaluations are identical on all 48 test cases.

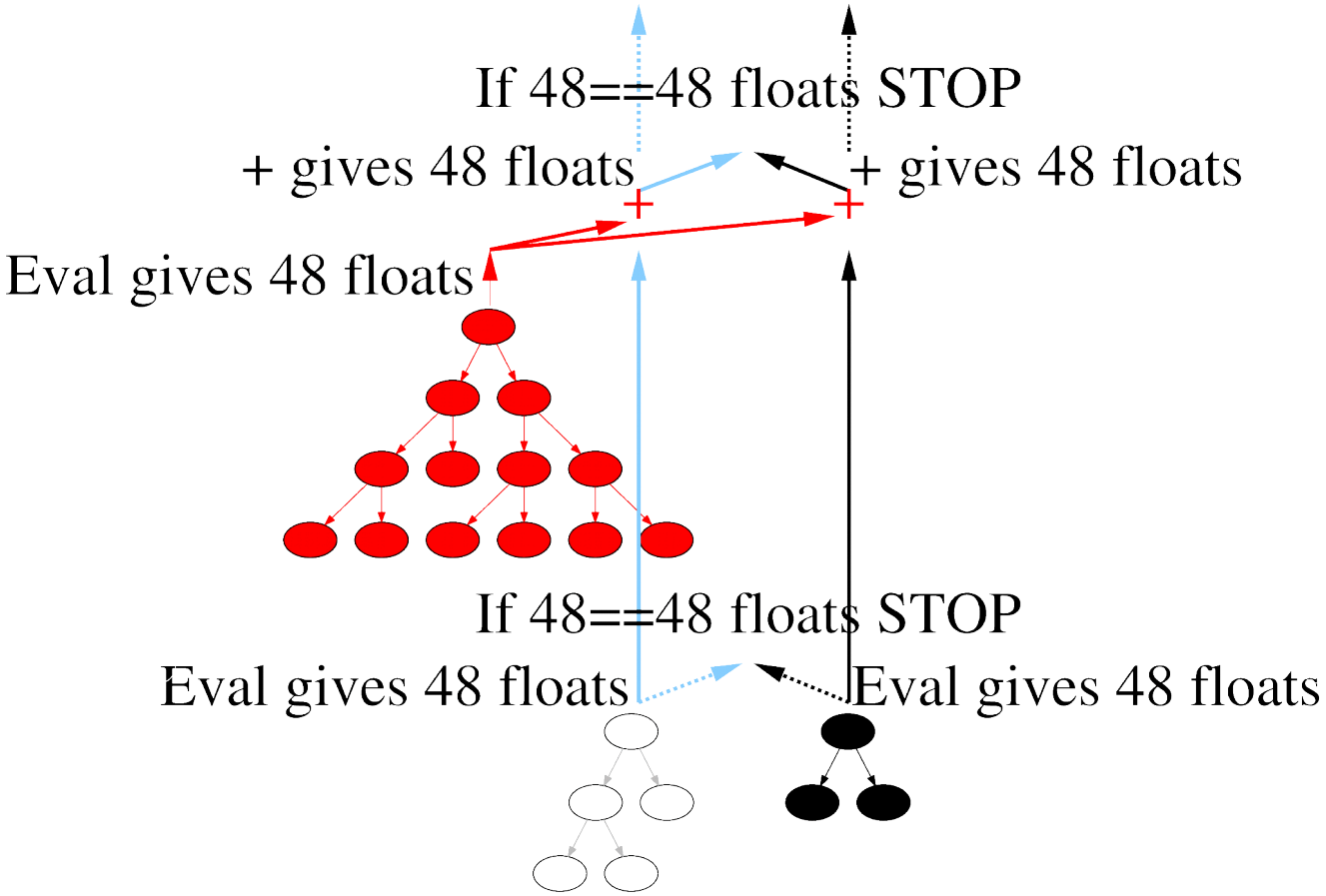
Functions lose information and so can give same output even with different inputs.

Incremental Fitness Evaluation



Subtree to be inserted (black) is evaluated on all test cases and values are transferred to evaluation of mum. Use incremental evaluation, so differences between original code (white subtree) and new (unborn) are propagated up 1st parent (mum) until either all differences are zero or we reach the root node.

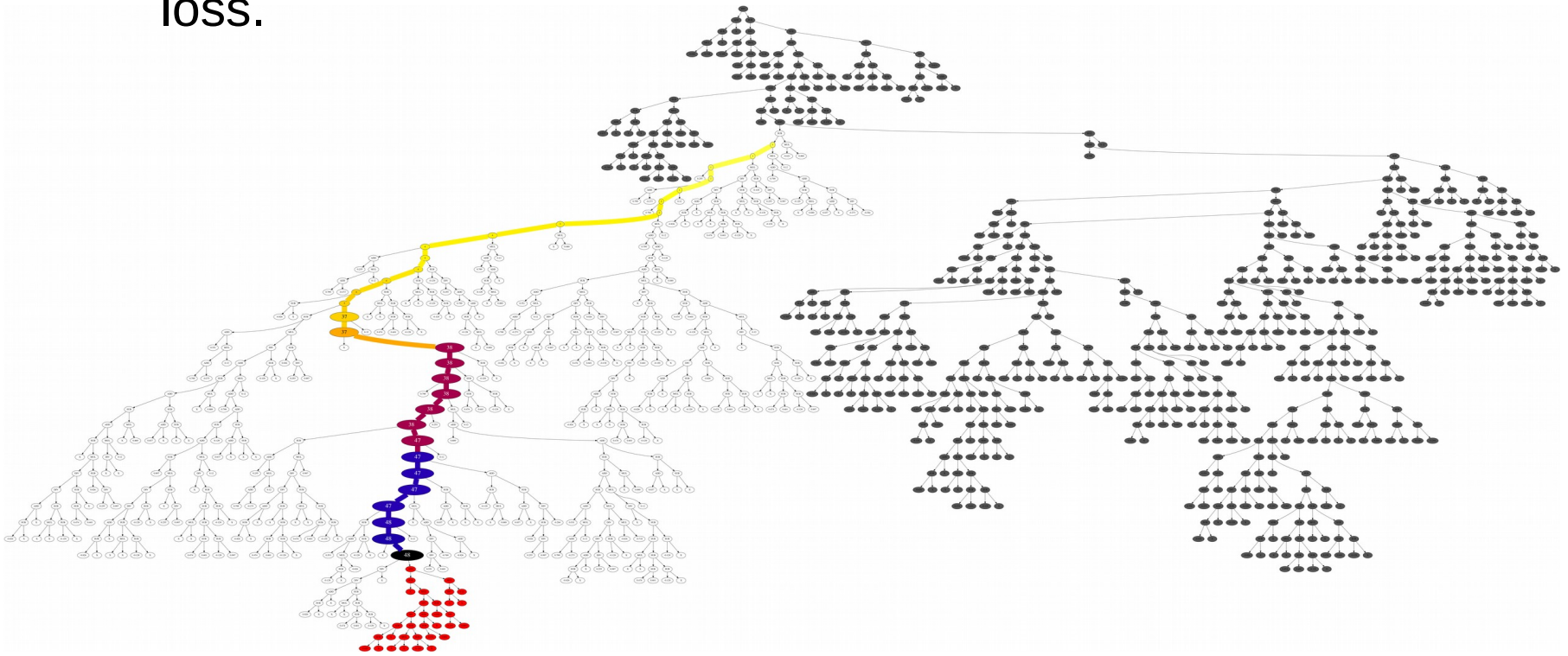
Interpret mum and child together



Evaluating the subtree to be removed from the mum (white) and the subtree to be inserted (black) on all (48) test cases. The interpreter proceeds up the mum's tree until either the evaluation on all test cases in the mum and the child are the same or it reaches the root node.

Evaluate both trees from change up

Changed code in red. Can stop fitness evaluation early as mum and child are phenotypically identical, due to information loss.

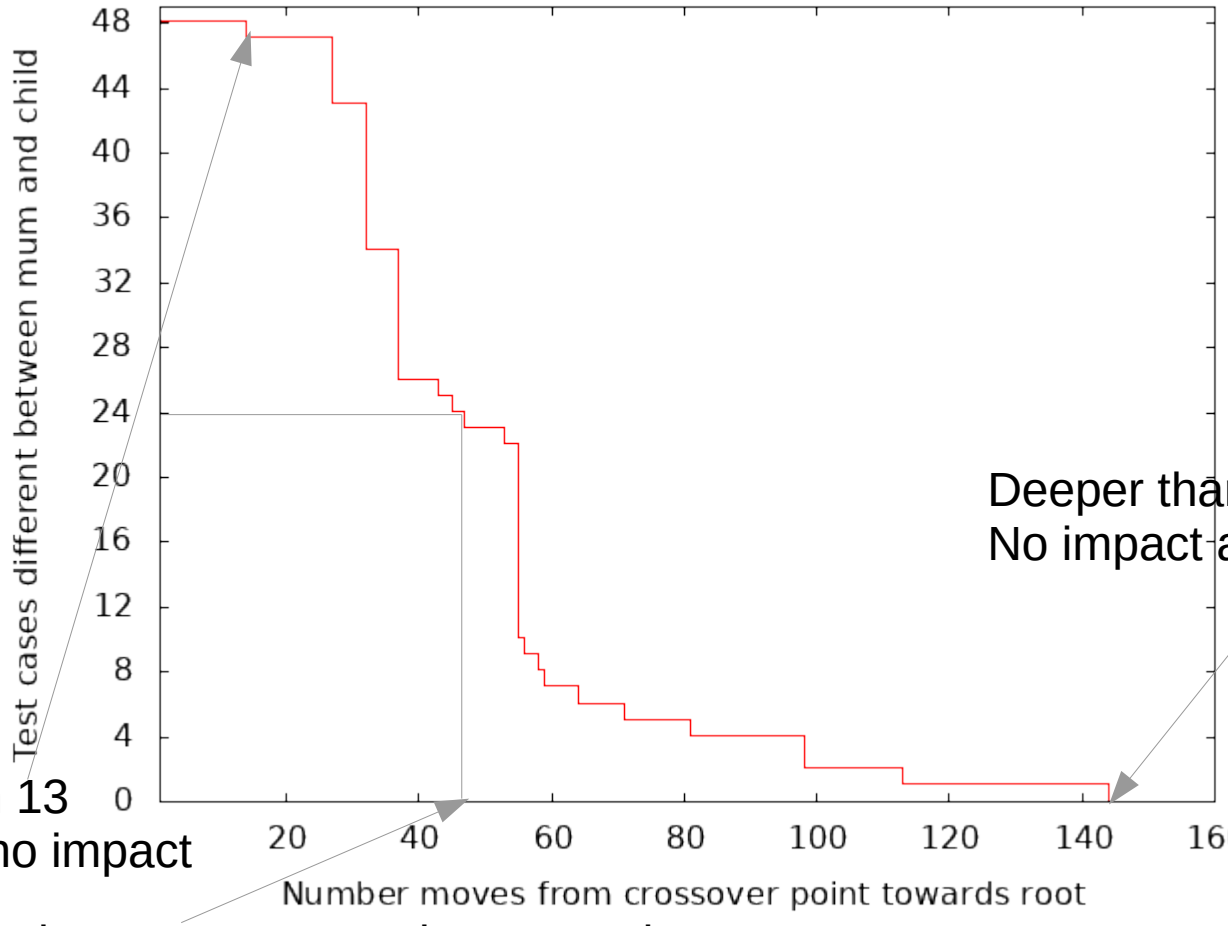


Example of bloated tree (chosen as it has no zeros, ie no “introns”).
Bloat due to information loss.

Node size gives number disrupted test cases.
Node colour gives change in evaluation value (log scale).

Side Effect Free: Disruption Falls Monotonically

Deeper disruption tends to have less impact on fitness



Deeper than 144
No impact at all

Deeper than 13
3 tests see no impact

Deeper than 44 ½ tests observe no impact

At each GP node: 32 bits + 32 bits => 32 bits
Information funnel. Information is lost.

Random sample 25001 nodes depth 383

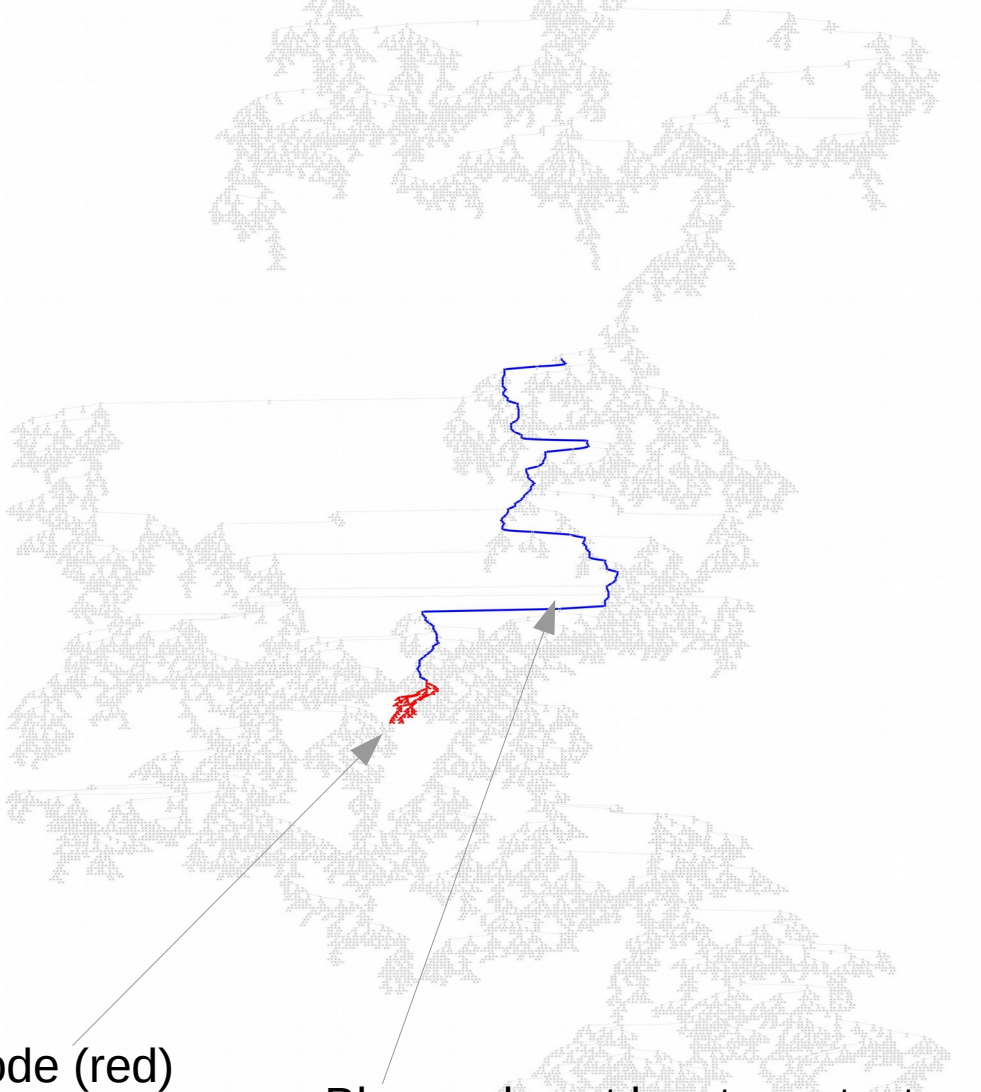
Deeper disruption tends to have less impact on fitness

Changed code (red)

Blue nodes at least one test case is different.
Change does not reach root node.

Random sample 25001 nodes

Deeper disruption tends to have less impact on fitness (fun 4 depth 491)

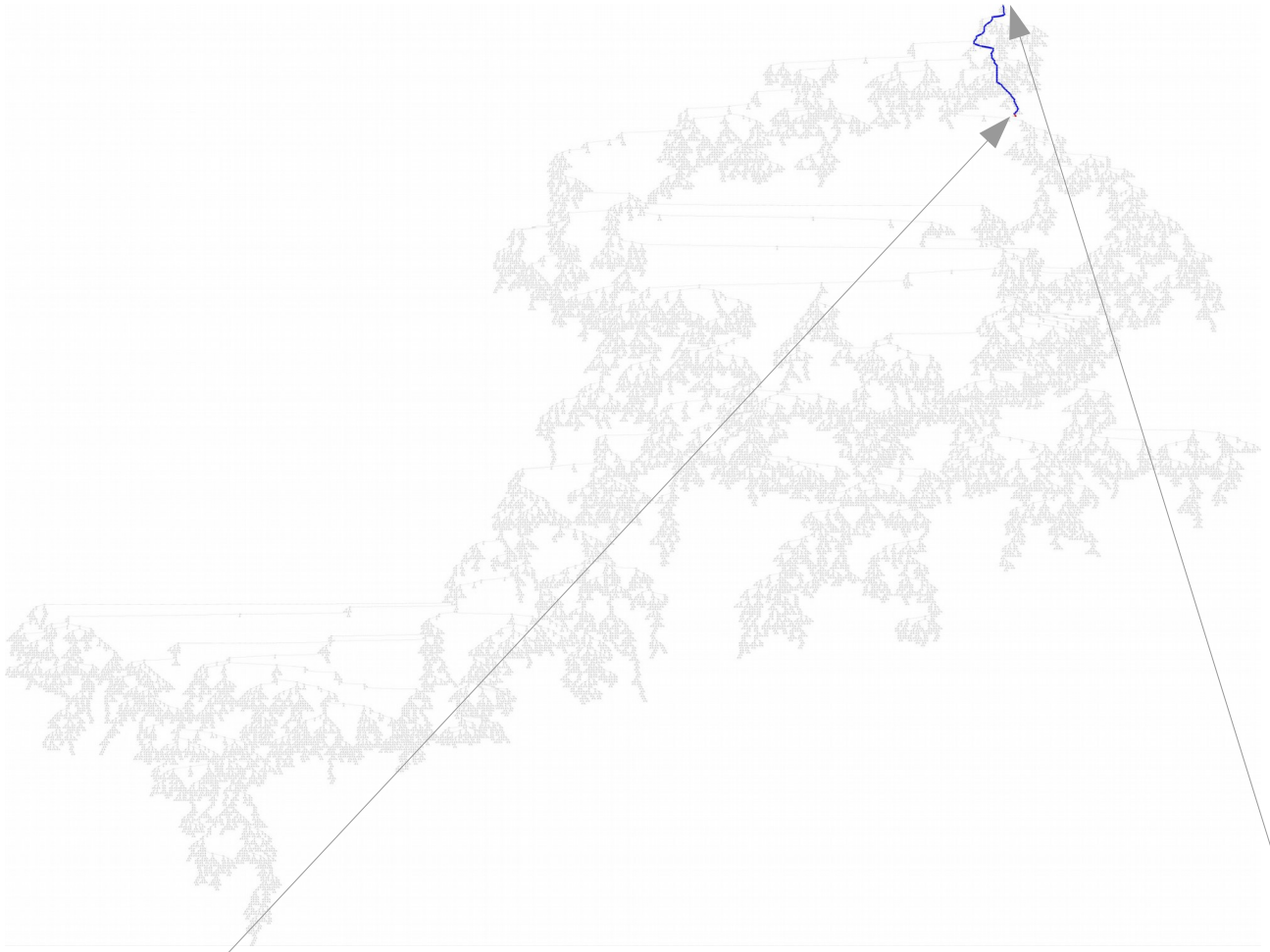


Changed code (red)

Blue nodes at least one test case is different.
Change does not reach root node.

Random sample 25001 nodes

Example where some of shallow disruption reaches root node (depth 390)

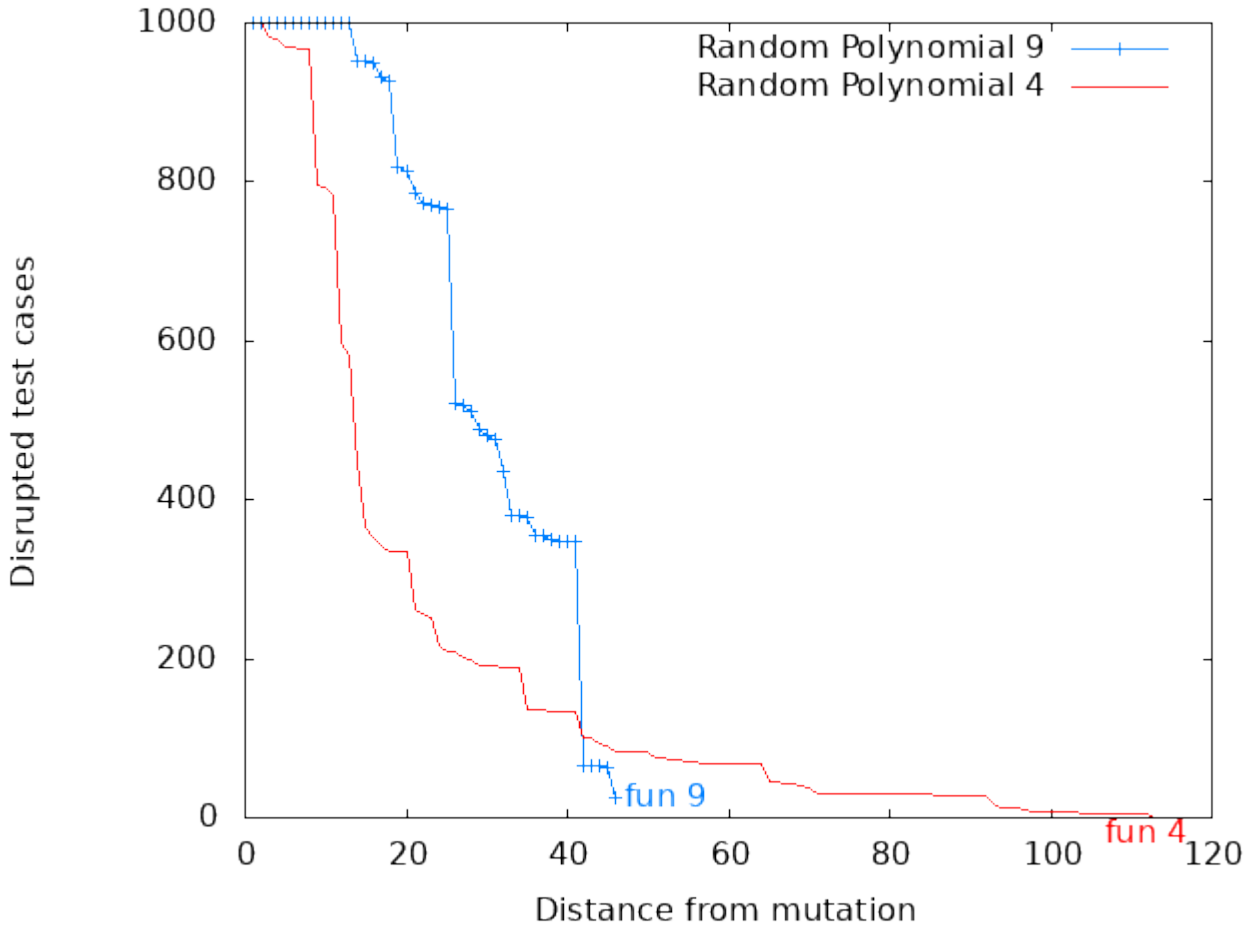


Changed code (red)

Blue nodes at least one test case is different.

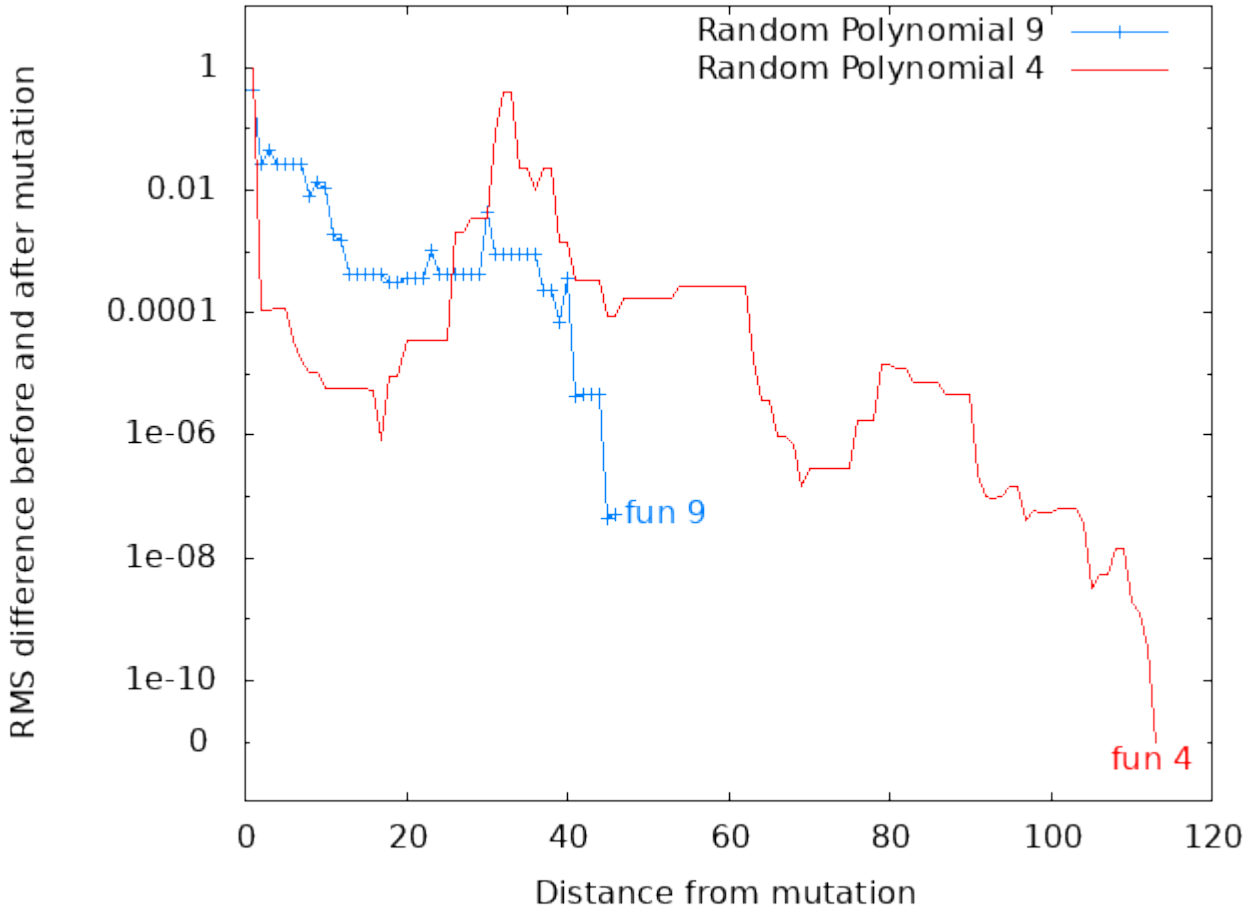
Monotonic fall in disrupted test cases

Only one of ten examples fails to reach zero disruption

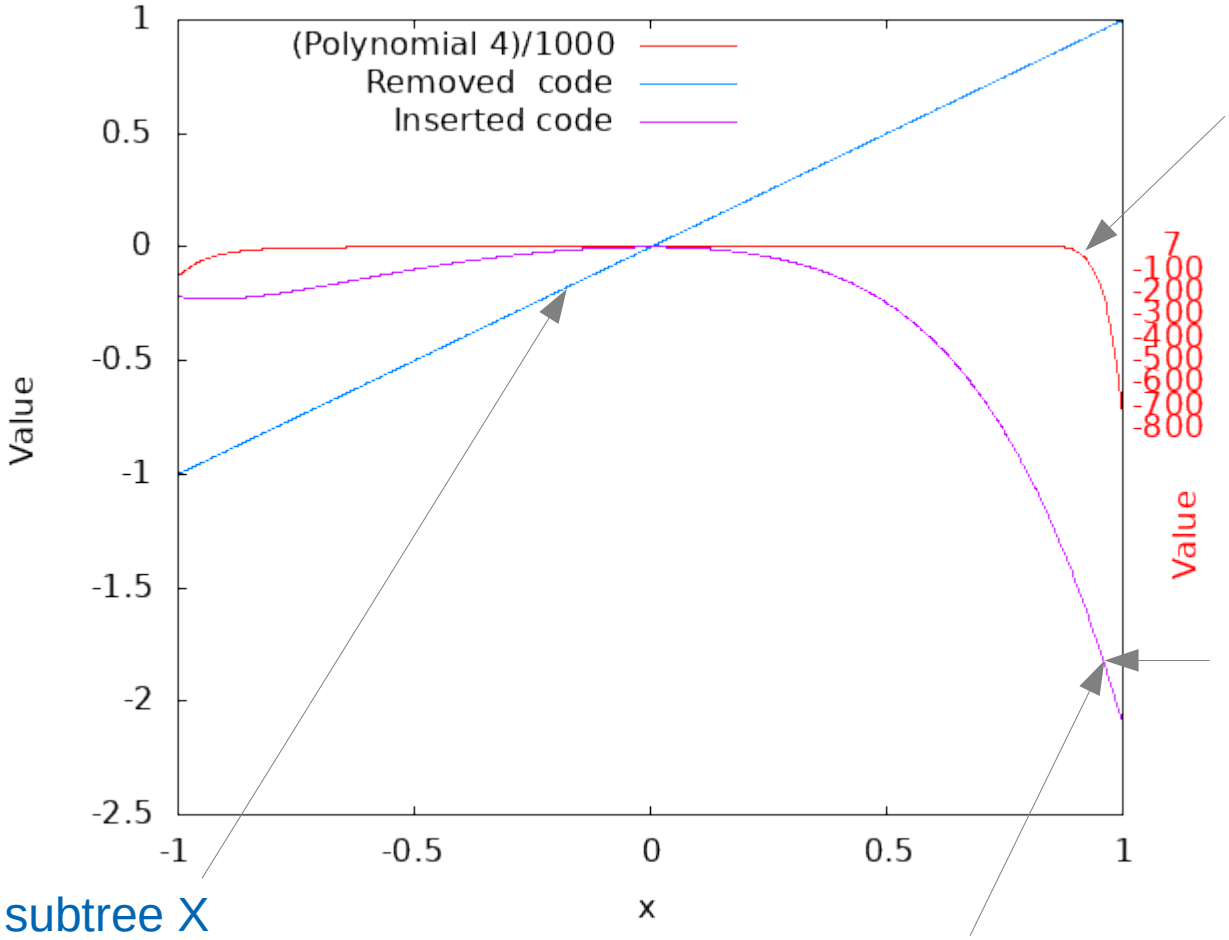


Average difference evaluation of mum v. child

Only one of ten examples fails to reach zero disruption



Random Polynomial 4

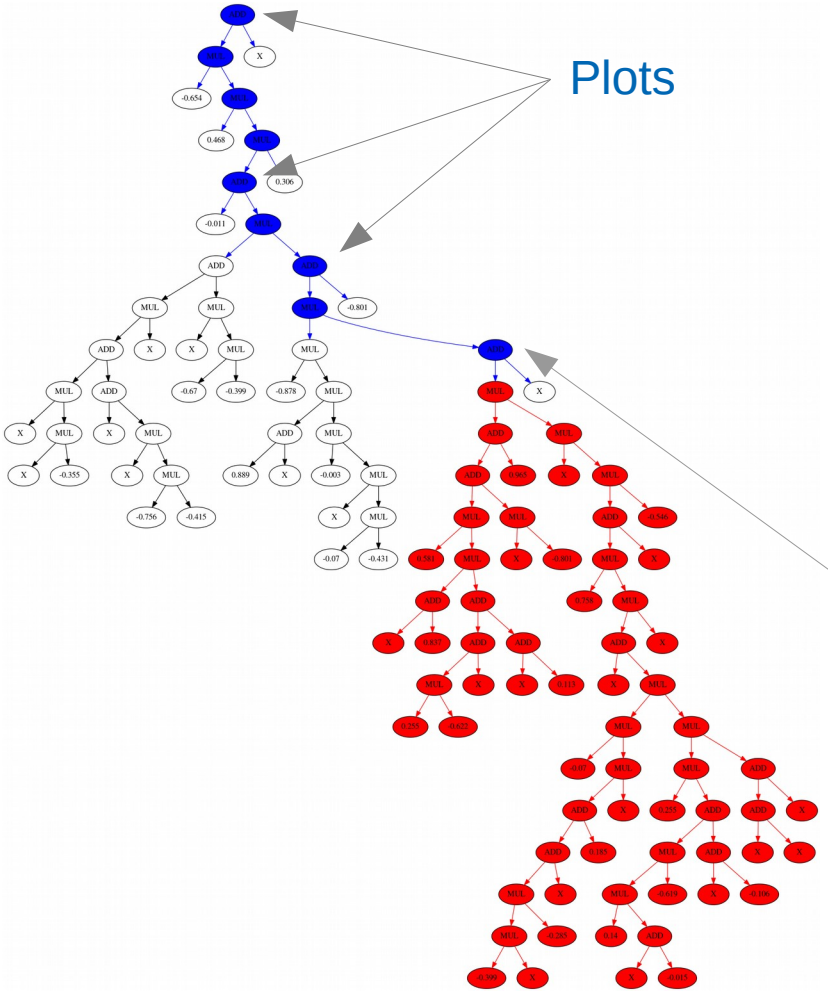


High order polynomial (rescaled). Non-zero at origin.

8th order inserted polynomial. Zero at origin

- Removed subtree X
- Insert subtree (MUL (ADD (ADD (MUL 0.581 (MUL (ADD X 0.837) (ADD (ADD (MUL 0.255 -0.622) X) (ADD X 0.113)))) (MUL X -0.801)) 0.965) (MUL X (MUL (ADD (MUL 0.758 (MUL (ADD X (MUL (MUL -0.07 (MUL (ADD (ADD (MUL (MUL -0.399 X) -0.285) X) 0.185) X)) (MUL (MUL 0.255 (ADD (MUL (MUL 0.14 (ADD X -0.015)) -0.619) (ADD X -0.106))) (ADD (ADD X X) X)))) X)) X) -0.546)))
- Note both are equal to zero at x=0

Random Change 4 (replacement tree)



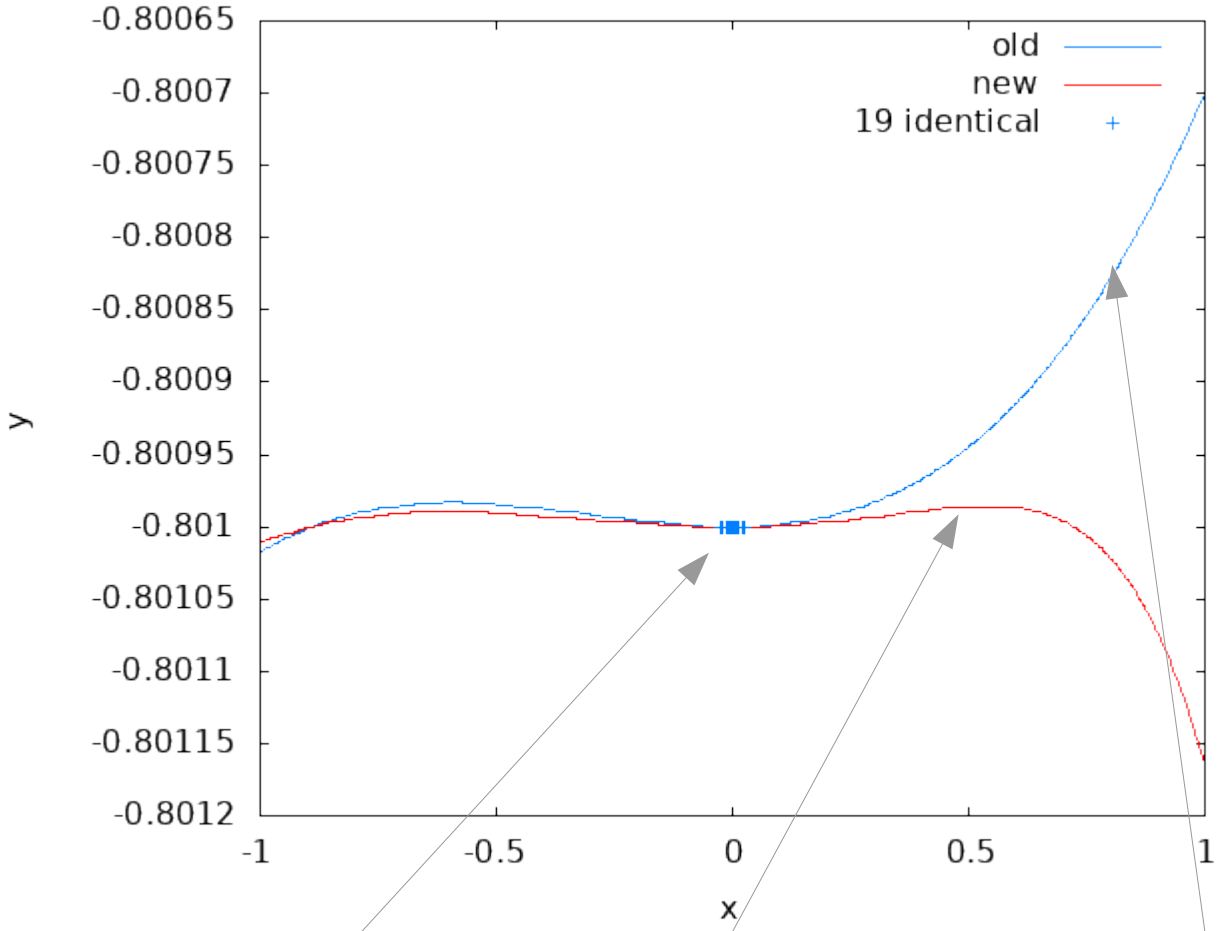
Replacement subtree and nearby original tree.

Evaluation of white nodes is identical.

Evaluation of blue nodes differs in new code.

Lowest blue node differs on 1000 test cases (but not test case x=0).

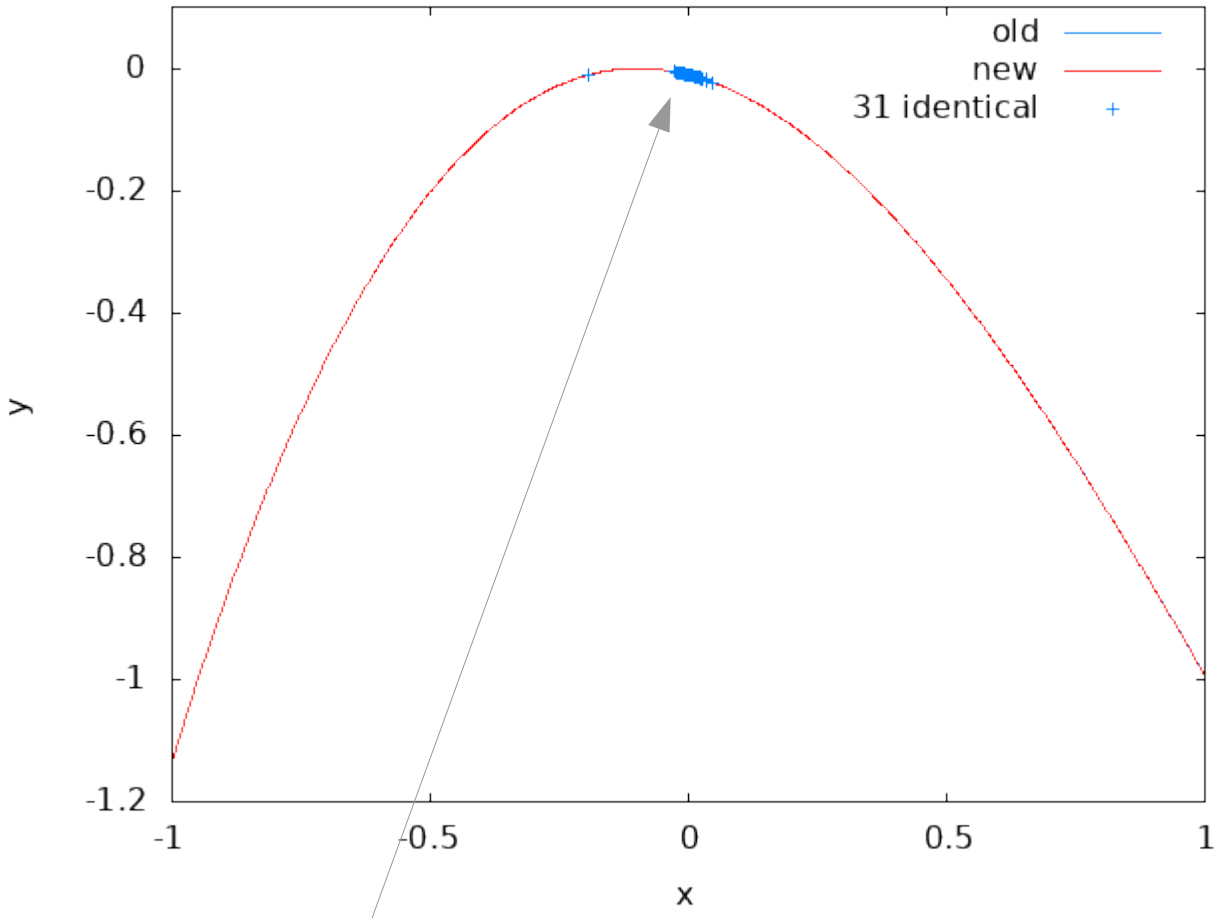
Impact of Change 4
Three levels (ADD -0.801) above crossover disruption



The new functionality (dashed line) closely follows the original for x < 0.2
At 19 points (+) they are identical.

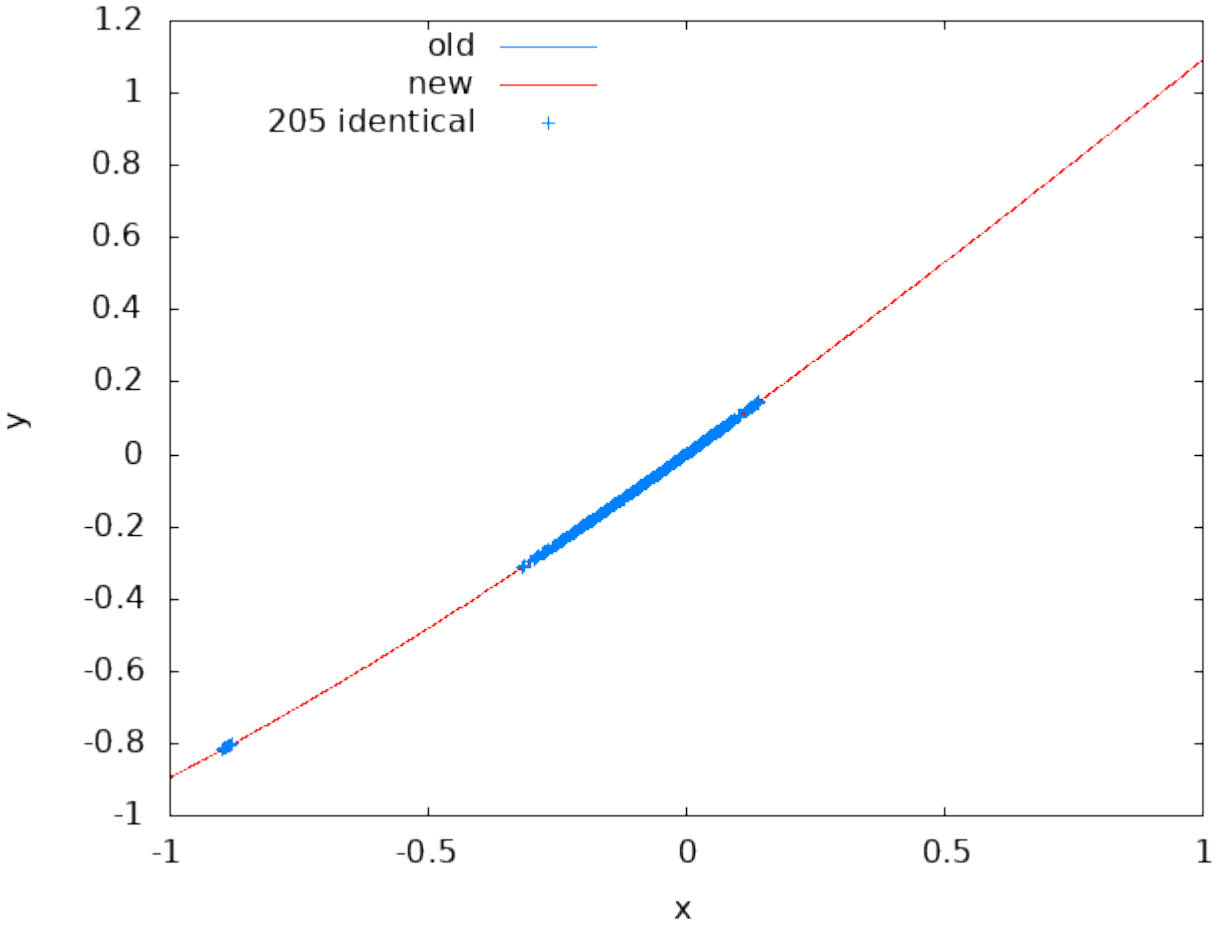
Once identical on a test case, will remain identical on that test case.

Impact of Change 4
Five levels (ADD -0.011) above crossover disruption



At 31 points (+) evaluation of old and new code is identical.

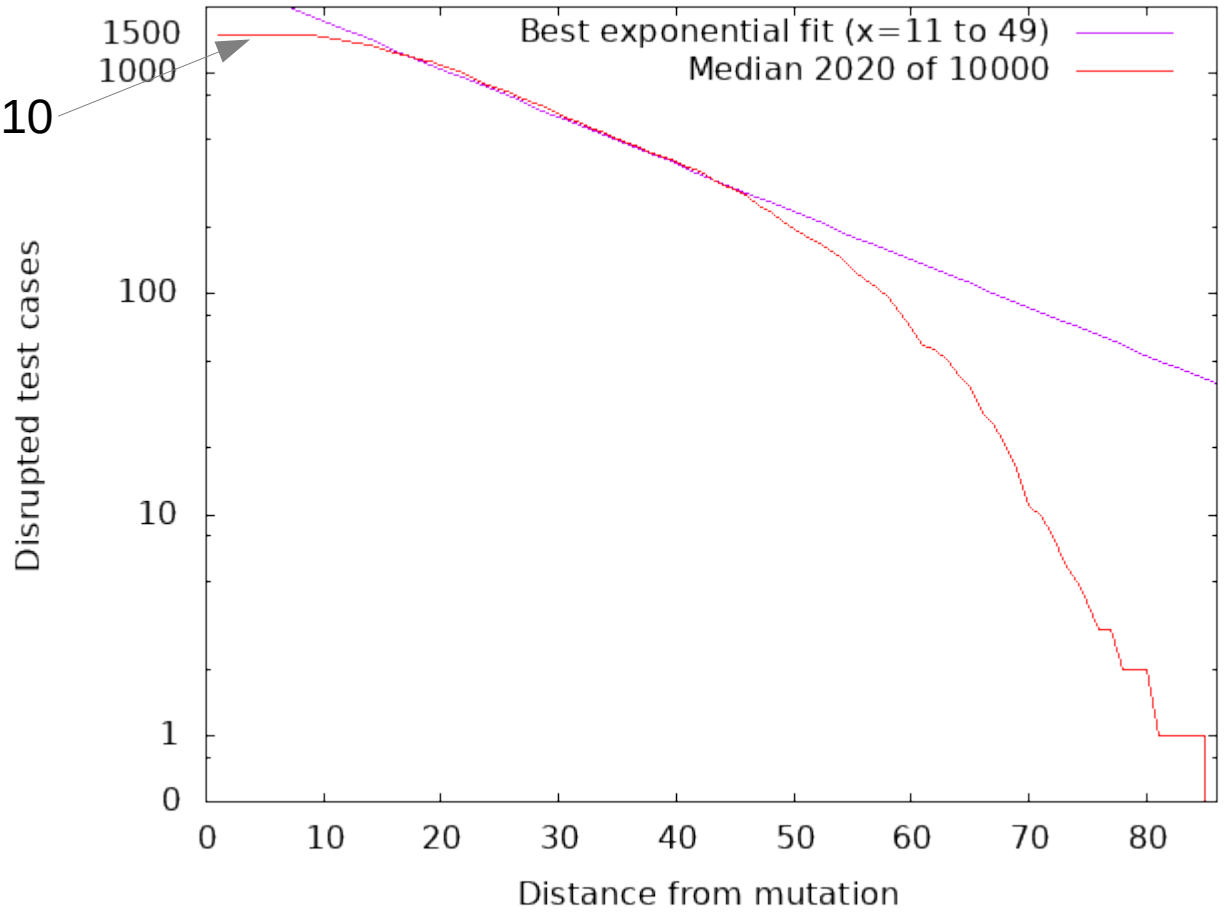
Impact of Change 4
Nine levels (ADD X) above crossover disruption



For 205 of 1001 points (+) evaluation of old and new code is identical. After transiting a total of 113 irreversible functions (32 bit floating point 53 additions and 60 multiplications), the replacement of X with a subtree of 67 nodes makes no difference at all.

Exponential Fall in Disruption 1501 tests

Flat region 1-10



- Average number of disrupted tests where change is eventually totally concealed.
- For our polynomials, bigger test values are more disruptive (harder to hide).

Issues

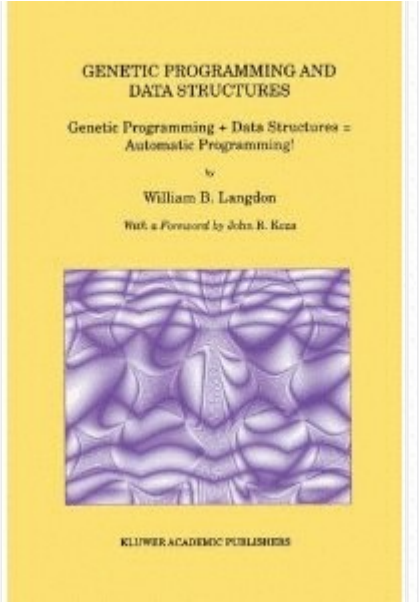
- Exponential decay in number of disrupted test cases suggests effectiveness of test suite of n tests rises only slowly with number of tests, Log(n)
  - But can this be proved
- Some mutations not being totally concealed
  - Can we characterise them?
  - Should we use them more or less in GP?
  - Can we characterise the tests needed to find them
- How much does this generalise to other types of GP
- Can lessons on mutations and testing be used in Software Engineering

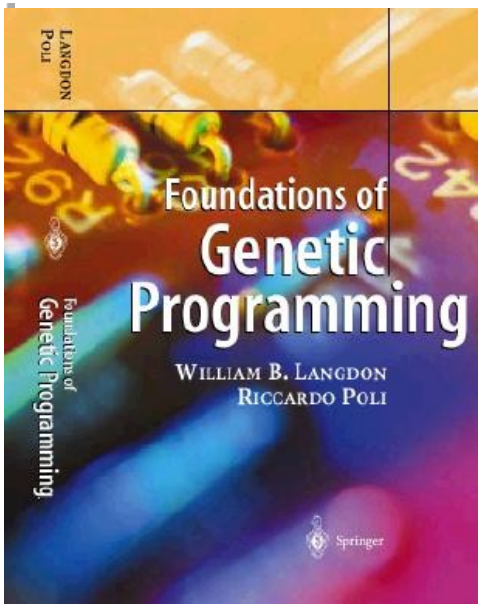
Conclusion Deep nesting hides errors

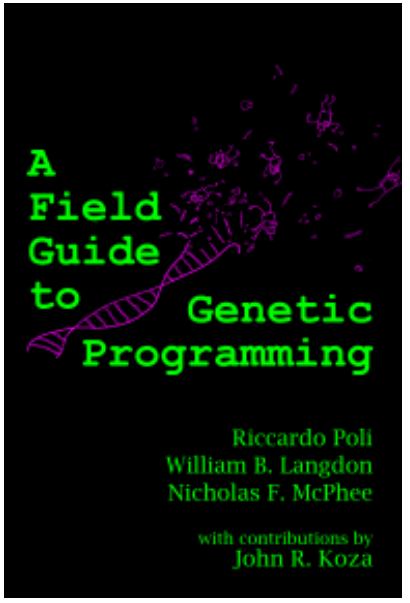
1) More fitness test cases have only small effect, log(n)
   - 1000 test cases only marginally more effective than 48
2) Deeper crossover or mutation may have less effect
   - Design your new crossover & mutation operators
3) Some functions lose information faster, eg division
   - Some tests more effective, here |x| > 1
4) If no disruption gets to root, crossover/mutation no effect
   - fitness identical => GP pop converges & evolution stops
5) If on some test cases, disruption does not reach root, crossover/mutation has less impact on fitness:
   - Information loss gives smoother fitness landscape.
6) Software is not fragile because progressive information loss dissipates many errors on many tests.

Genetic Programming

W. B. Langdon





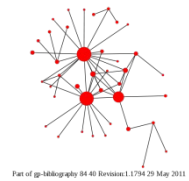


The Genetic Programming Bibliography

14543 references, 13000 authors

Make sure it has all of your papers!

E.g. email W.Langdon@cs.ucl.ac.uk or use | Add to It | web link

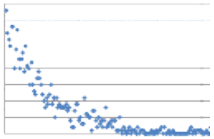


Co-authorship community.
Downloads

Downloads by day

A personalised list of every author's GP publications.

blog



Your papers

Googling GP bibliography, eg:
Evolutionary Medicine site:gpbib.cs.ucl.ac.uk