Control Evolutionary Robustness

2nd GECCO workshop on Graph-based Genetic Programming, graphgp.com
Room 104, Sunday 14 July 2024, 16:10-18:00

“Information theory says deep nested crossovers and mutations do not change fitness.

W. B. Langdon, UCL

38% GP papers on Applications

“When I was your age I could think of six impossible things before breakfast.”
Do the impossible

Be shallow
Be close to fitness environment

Genetic Improvement tutorial Monday 15 July
“Sustaining Evolution for Shallow Embodied Intelligence”
Evolutionary Robustness

- Applications of Genetic Programming, 38% of GP papers
- Creativeness of Evolutionary Computing [CACM June 2024]
- Run GP to a million generations
  - A trillion GP operations per second
  - Exploits lost of disruption in deep trees (10000 levels, 1e9 nodes)
- Information Theory explains Failure of Disruption to Propagate
- FDP so crossover/mutation little impact in deeply nested trees
- Evidence that deep mutations have little impact in C/C++
- Implications of too much robustness
  - C++ deeper code harder to test, improve, optimise, repair
  - Sustained long term evolution needs limited depth of nesting:
    - Open architecture with many small shallow programs
Applications of Genetic Programming

• Question is GP in use raised in recent celebration of 30 years since Koza’s 1992 GP book [“Jaws 30”, GP+EM]. Industrial use not published but:

• Applications 38% of 2023 GP bibliography papers
Creativeness of Evolutionary Computing

• Evolutionary Computing (EC) was ignored
• Last year *Communications of the ACM* had many LLM articles. Including claim in 2019 “no working examples of creative AI”
• Many examples of EC creative AI, just a few:
  - Koza’s “invention machine” GP4 2003, “Creative Evolutionary Systems” Bentley+Corne
  - Whole conference series dedicate to it EvoMusArt
• Need papers, discussion, etc., outside evolutionary computing venues

W. B. Langdon, UCL
Long-Term Evolution Experiment with Genetic Programming

- Rich Lenski's Long Term Evolution Experiment LTEE 75000 generations of bacteria E.coli shows continued evolution. (Homo Sapiens about 9300 generations old)
  - Experiment running since 1988 (36 years), MSU
- What happens in GP?
  - Run up to 1 million generations 2 billion nodes
  - Run experiments in days/weeks
Fitness improvement continues but slows

Evolution of mean absolute error in ten runs of Sextic polynomial with population of 500. Runs to 100,000 generations. Thousands of fitness improvements found. Note log scales.
Size and depth of the best individual in each of 100,000 generations for eight Sextic polynomial runs with population of 500.
When programs are large chance crossover hitting sensitive area near output falls in proportion to size. (Evolution gives different ratios between runs.) Smoothed by taking running averages over a 100 generations.
+1 Disruption, Fibonacci run 7, depth 33

red 16-20 test cases, blue 1 test cases

Output (root node)

Only disruption near root node reaches output
Exponential fall in fraction of run time disruption changing program output with depth

Test case $J=9$

Fraction run time disruption changing output

Distance (depth) between location of +1 disruption and output

W. B. Langdon
Exponential fall in fraction of run time disruption changing program output with depth

Deep floating point GP trees similar

W. B. Langdon
Long-Term Evolution Experiment with Genetic Programming

- Rich Lenski's Long Term Evolution Experiment (LTEE) shows continued evolution. 75,000 generations of bacteria E.coli shows continued evolution. (Homo Sapiens about 9,300 generations old)

- What happens in GP?
  - Run up to 1 million generations, 2 billion nodes

- In GP fitness improvement continues but slows

- Information theory explains crossover Disruption Fails to Propagate (FDP) to output, so fitness is unchanged.

- Only crossover or mutation near output impacts fitness. Rate of fitness improvement $O(1/\text{size})$

- True in any hierarchical system, shows up in C/C++

- Need evolvable code close to fitness environment
Deep Mutations have Little Impact

- PIE (Propagation Infection Execution) view of software bugs
  - **E** bug needs to be executed
  - **I** bug needs to change (disrupt) the program’s state (infection)
  - **P** the disruption needs to propagate to the program’s outputs

- If entropy loss causes Failure of the Disruption to Propagate FDP the error is invisible and the software is robust to it

- Information Theory says impact of disruptions lost with distance when nested

- Just seen deep GP trees are robust

- PARSEC, VIPS, vipsthumbnail benchmark deeper C code more robust to mutations
Computer operators are irreversible. Meaning input state cannot be inferred from outputs. Information is lost.

Information Funnel

Two 32 bit inputs

Information funnel

More information enters than leaves

32 bit output

W. B. Langdon, UCL
Information flow in five nested functions

Potential information loss at each (irreversible) function

Disruption may fail to reach output.

(No side effects.)

W. B. Langdon, UCL
Using Magpie to Sample C Mutations

• Genetic Improvement tool Magpie  https://github.com/bloa/magpie

• PARSEC suite of benchmarks to test parallel super computers for NASA. Mostly numeric but includes some image processing, including VIPS

• VIPS C image processing library 90,000 lines

• Chose vipsthumbnail, parallel multi-threaded, takes large image creates small image 128 pixels wide

• Use linux perf to profile vipsthumbnail select all VIPS functions perf reports

• Use GDB to select all functions called enroute to top CPU using function

• Remove unused functions (a few unused lines, eg if/switch case included)

• 90,000 => 7328 lines, in 37 C files. srcml => 37 XML files

• 1000 random Magpie mutations, measure their impact, measure execution depth
Magpie Mutating C

- Genetic Improvement Magpie https://github.com/bloa/magpie
- VIPS image thumbnail benchmark (use 37 files 7328 LOC)
1000 random Magpie VIPS mutants

- VIPS image thumbnail benchmark (use 37 files 7328 LOC)
  - try to exclude unused code
- Magpie mutating source code as XML, mostly syntax preserving, mostly compiles, runs, 526 give right answer
- 37 cases output wrong but no exception.
- Randomly choose 25 of 37, compare with 25 where mutant code is run, changes state but output is unchanged

<table>
<thead>
<tr>
<th>Compiled, ran correct output</th>
<th>526</th>
<th>Correct output</th>
<th>438</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mutation is identical to original code</td>
<td>88</td>
</tr>
<tr>
<td>Failed to compile</td>
<td>302</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failed to run correctly or gave incorrect output</td>
<td>164</td>
<td>exception</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td></td>
<td>output error</td>
<td>37</td>
</tr>
<tr>
<td>Magpie TypeError</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
25 v 25 Mutants. Deep less impact

- **25** mutants change execution but no change to output
- **25** mutants which change execution (without causing segfault) but change output output
Conclusions: Control Evolutionary Robustness

- Importance of reminding everyone of GP achievements
- Genetic Programming can be creative, GP is in use
- Information theory predicts failed disruption propagation FDP, which makes GP and software robust
- Excluding segfault etc., most C/C++ mutations nested >30 function calls deep did not change output
- In GP exponential decay with depth, impact of mutations lost
  - Need shallow code for prolonged evolution
  - “Sustaining Evolution for Shallow Embodied Intelligence”
- Be ambitious

Do the impossible
Genetic Programming

W. B. Langdon
To evolve large complex code, Must **AVOID** large fossil of dead code

- With **deep code** most crossovers and mutations make **no difference**.
- Leading to random drift
- Not directed evolution
- To avoid dead center evolving code must be near environment.

Large **dead** center
Thin evolving crust
The Genetic Programming Bibliography

17231 references, 17000 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

A personalised list of every author’s GP publications.

blog

Googling GP bibliography, eg:
Graph-based Genetic Programming site:gpbib.cs.ucl.ac.uk