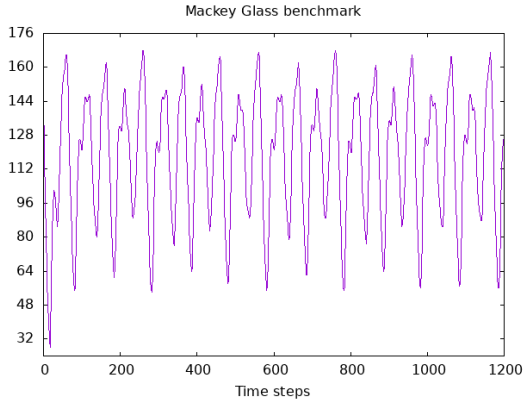
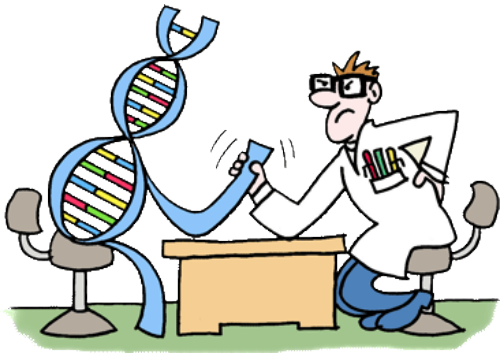
CREST

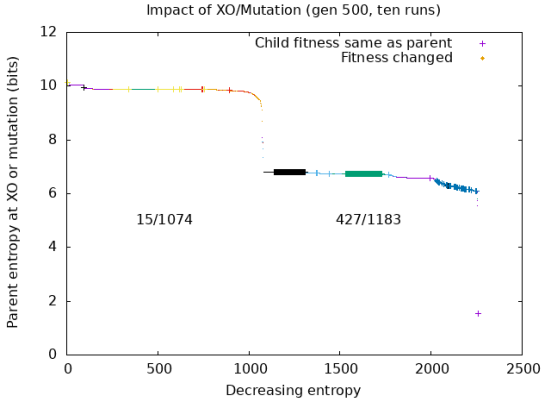
Recording

Long Term Evolution Experiments with Linear Genetic Programming

Software Systems Engineering (SSE) Seminar, 17 Feb 2026







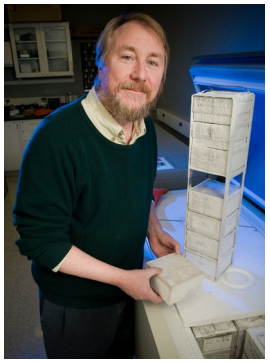
$10000 "Humies"
Submit by 29 May

W. B. Langdon, UCL

Long Term Evolution Experiments with Linear Genetic Programming

- Invited book chapter *Advances in Linear Genetic Programming*
- Run Linear Genetic Programming to a 100,000 generations
  - Exploit AVX512 (GI 2026) and multicore
  - Improvement continues but rate falls with program size
- Program's information content (entropy)
  - initially = entropy of inputs
  - falls to near entropy of desired output distribution
  - Almost all instructions do not change entropy

Long Term Evolution Experiment LTEE in Biology



- Rich Lenski's Long Term Evolution Experiment LTEE 80000 generations of bacteria E.coli shows **continued evolution**. (Homo Sapiens 9300 generations old)
  - Experiment running since 1988 (38 years)
    - 1988 University of California, Irvine
    - 1991 Michigan State University
    - 2022 University of Texas at Austin
    - 2025 return to Michigan State University. 82,000+ generations
- What happens in computer based evolution?

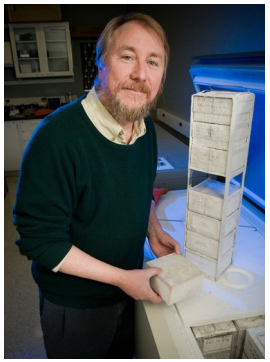
  Long-Term Evolution Experiment with Genetic Programming
  Artificial Life, 2022 28(2) pp 173-204

  Talk 4 March 2025
  http://crest.cs.ucl.ac.uk/W.Langdon/langdon_4-mar-2025.mp4
- Run with Linear Genetic Programming: Study Entropy

  Long Term Evolution Experiments with Linear Genetic Programming.
  In Recent Advances in Linear Genetic Programming. Springer, 2026.

Long Term Evolution Experiment LTEE with GP

- Rich Lenski's Long Term Evolution Experiment LTEE 80000 generations of bacteria E.coli shows **continued evolution**. (Homo Sapiens 9300 generations old)



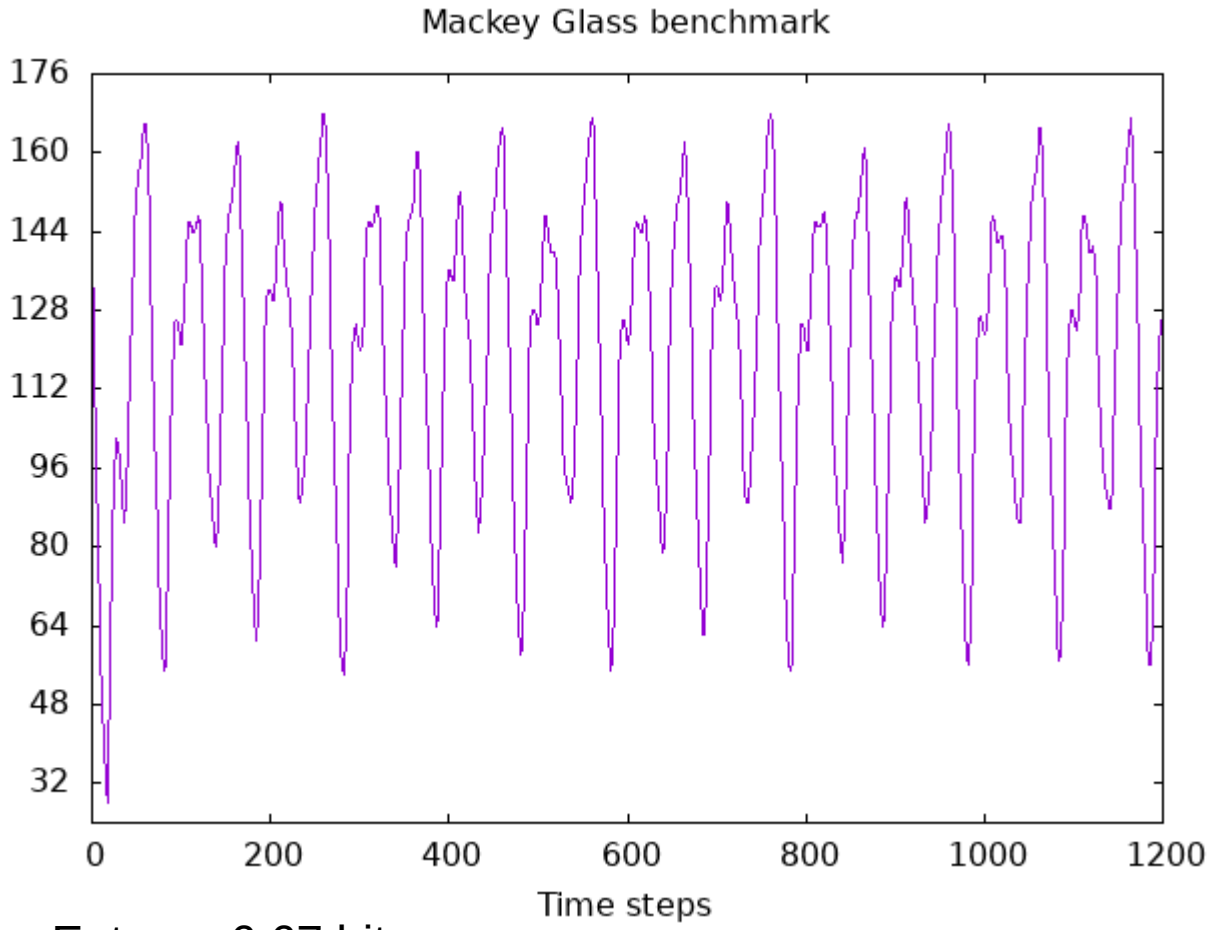
- Run with Linear Genetic Programming: Study Entropy
  - Run up to 100,000 generations 4 million nodes (E.coli 4.6 million base pairs)
  - Run experiments in ≤ 12 days
  - Run linear genetic programming ten times

Long Term Evolution Linear Genetic Programming

- Peter Nordin's GPengine https://github.com/wblangdon/GPengine
- 500 random initial programs 1-14 instructions
- Evaluate programs on 1201 Mackey-Glass test cases
- Fitness RMS error predicting next timestep
- Steady state overlapping generations
- Tournament selection
  - Four randomly selected. Best 2 parents ⇛ 2 children. Children replace losers
- Crossover and mutation
- Eight short integers (bytes) registers
- Four arithmetic: + - * protected division (y!=0)? x/y : 0
- 100,000 generations, ≤ 4 million instructions
- GI 2026 optimise AVX parallel fitness evaluation

Predict Chaotic Time Sequence

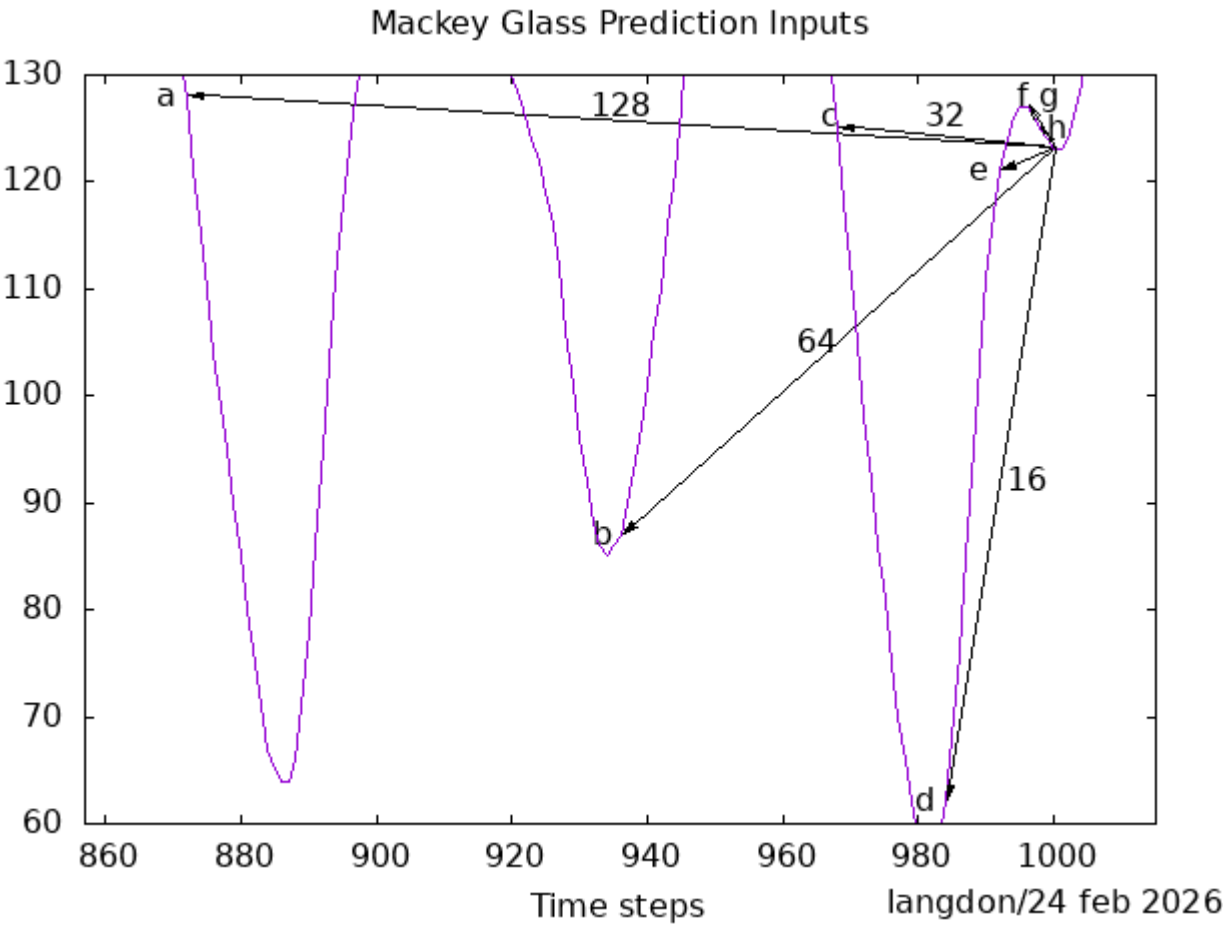
IEEE Mackey-Glass prediction benchmark sequence is chaotic,
so inherently not predictable.
Nonetheless, using earlier values, it can be approximated.



Entropy 6.67 bits

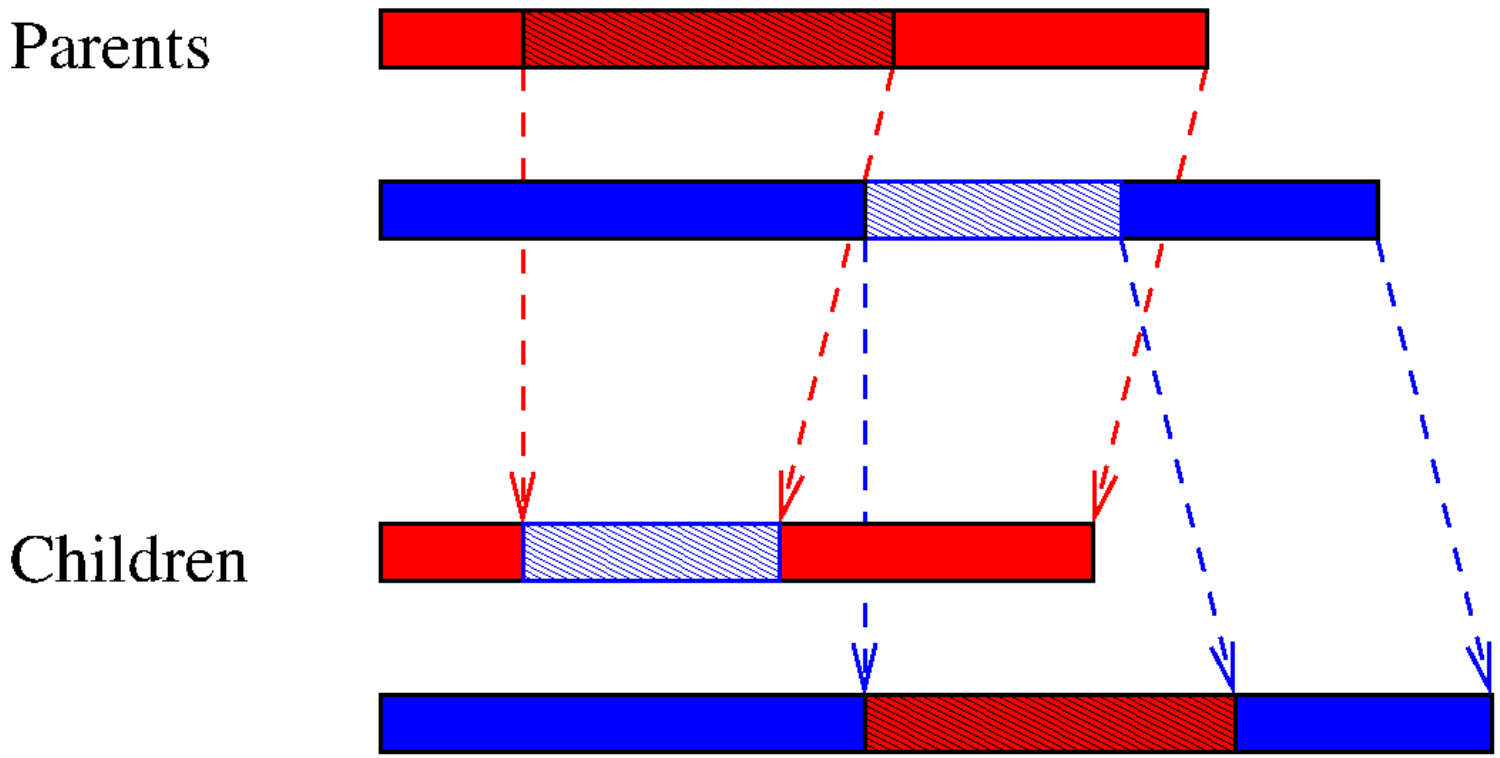
https://github.com/wblangdon/GPengine

Predict Chaotic Time Sequence

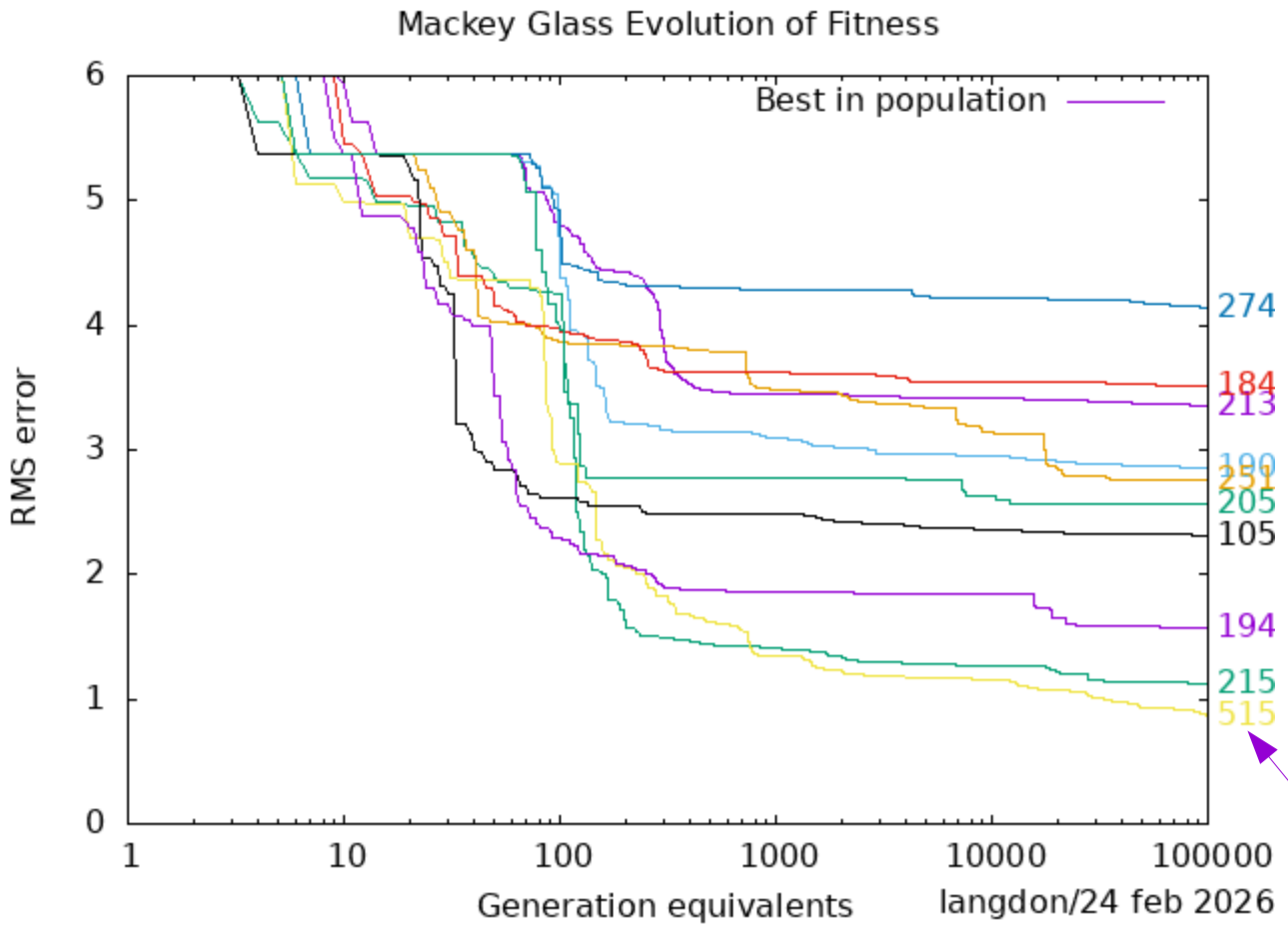


Eight 8-bit (0-255) inputs:
a 128, b 64, c 32, d 16, e 8, f 4, g 2, h 1
time steps behind

Crossover: Two Parents ⇒ Two Children

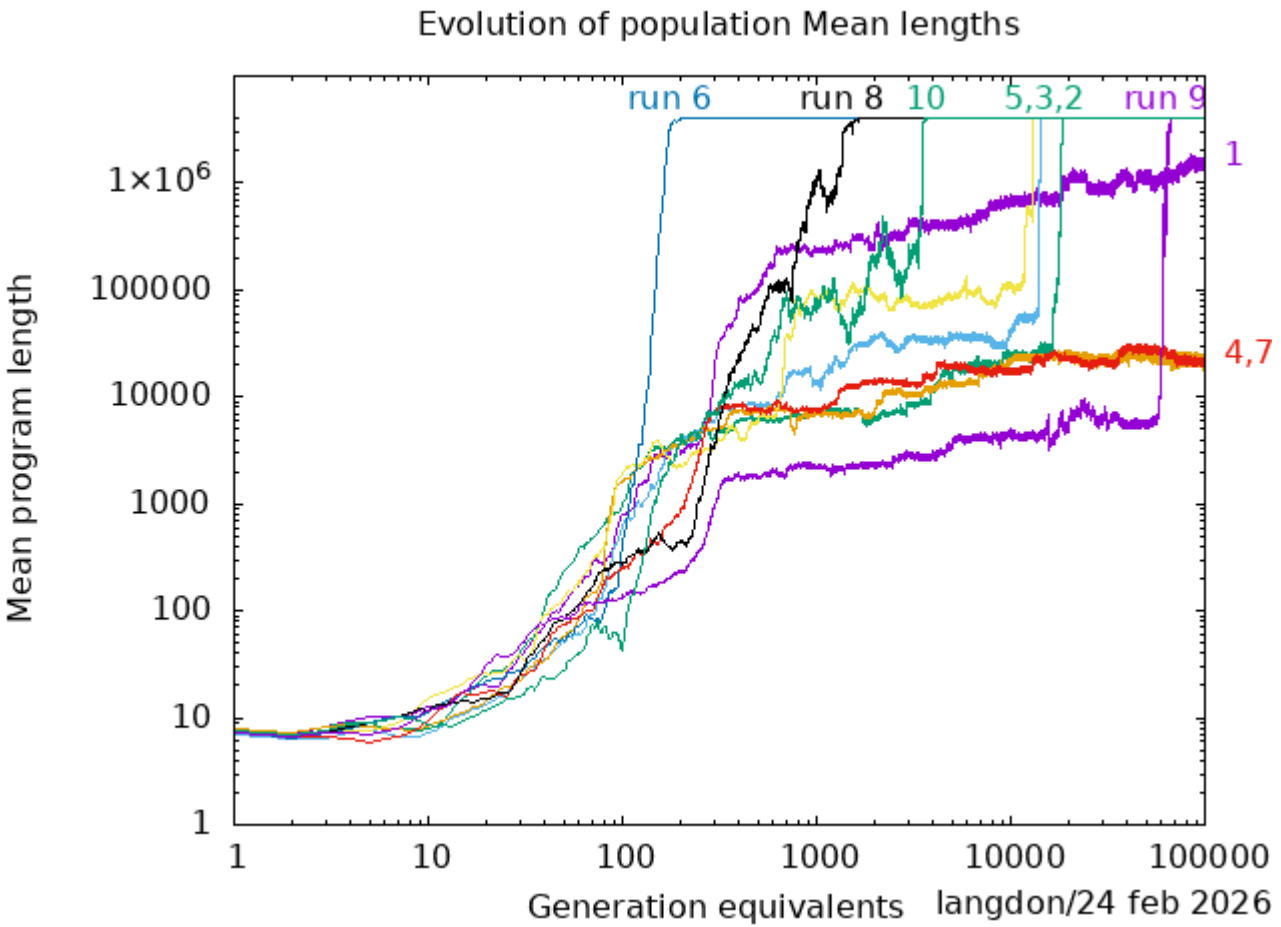


Evolution of Fitness (ten runs)

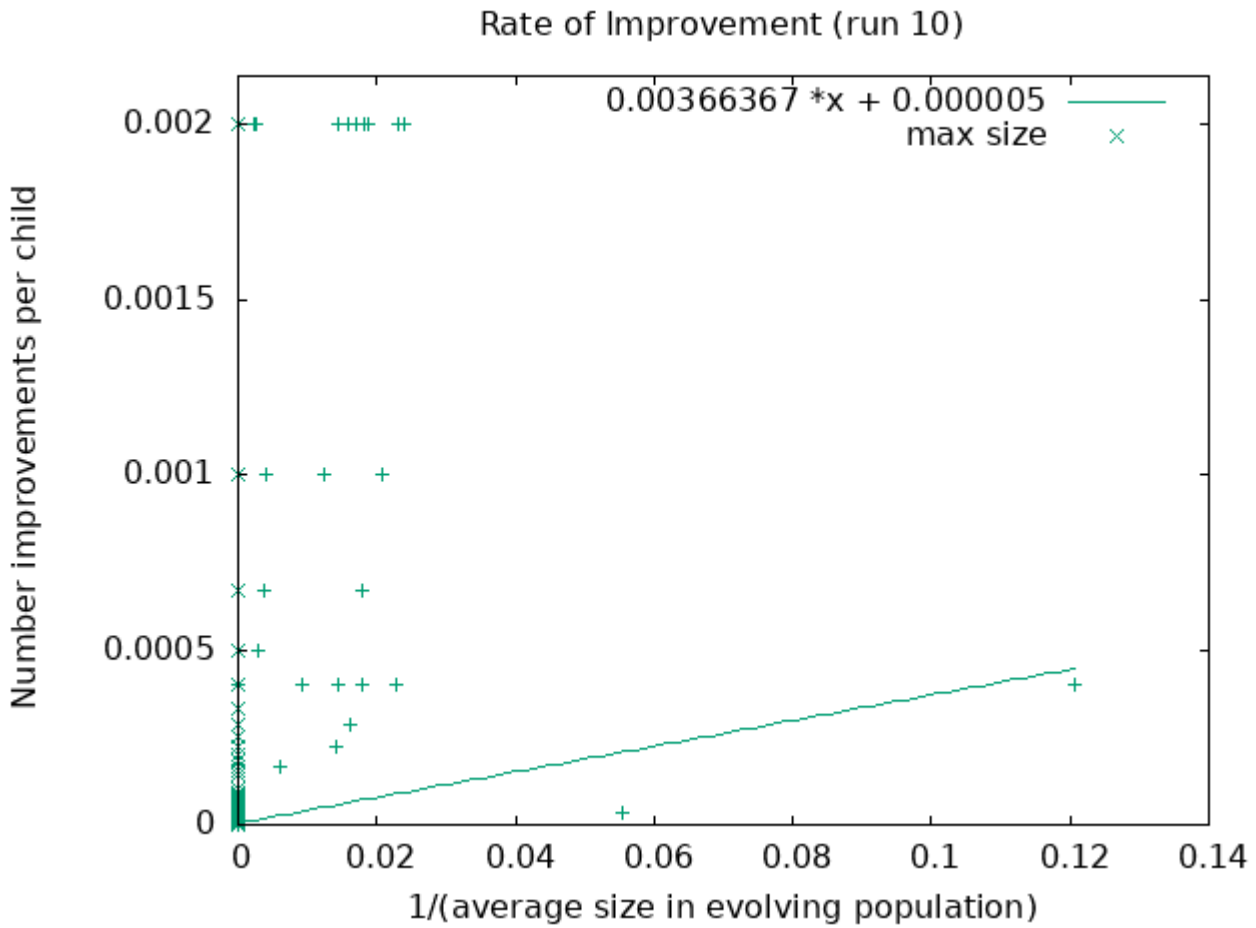


Number of fitness improvements in run

Evolution of Program Lengths (ten runs)



Rate of Innovation proportional to Length^{-1}



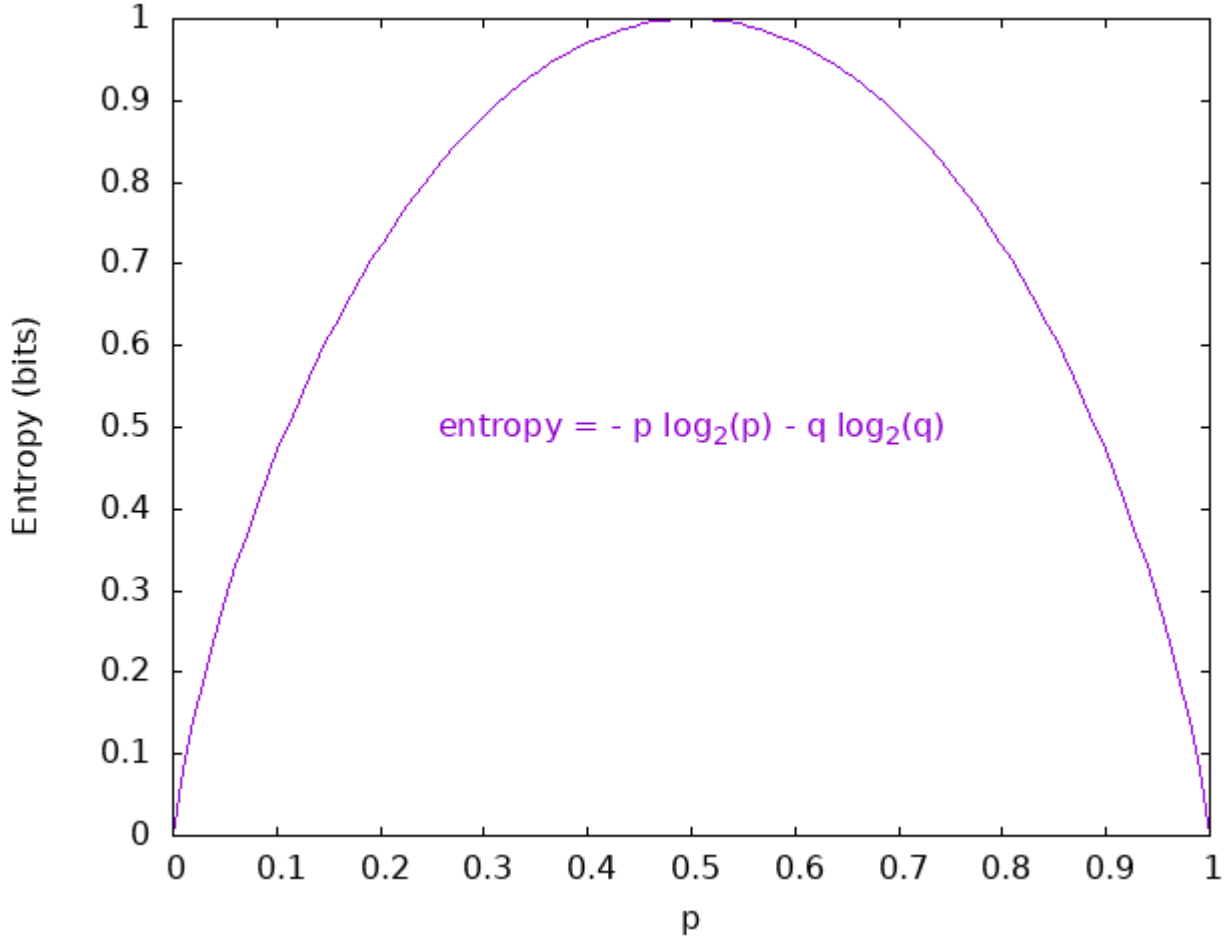
Long Term Evolution Linear Genetic Programming

- As expected, can use Peter Nordin's Linear GPengine to approximate time series.
  - Using genetic improvement (GI 2026) we optimised its expression evaluation. 64 fitness cases in parallel
  - Plus up to 19 cores in parallel.
- As expected, programs get longer.
- Innovation continues, rate falls as programs get longer
  - Bulk of program not improved
  - Fixed length sensitive region near program's inputs
- Why?
- Information theory: entropy within executing program

Entropy = - Σ_i p_i $\log_2(p_i)$

- Use entropy to quantify idea of information held inside a program as it runs
- Entropy is defined over probability distributions
- Entropy = - Σ_i p_i $\log_2(p_i)$
- Can view $-\log_2(p_i)$ as how surprising event i is
  - Eg a very rare event, $p_i = 1/1024$, $-\log_2(1/1024) = 10$
  - View entropy as average surprise.
- Here we consider distributions over the test inputs.
  - Eg for Mackey-Glass we have 1201 test cases
  - Consider how often does event i occur across the test suite
  - eg event i = when executing instruction 200, how often do registers a..h hold values 1,2,3,4,5,6,7,8
  - Get distribution of i by considering all values a..h have

Entropy two alternatives: min 0, max 1 bit

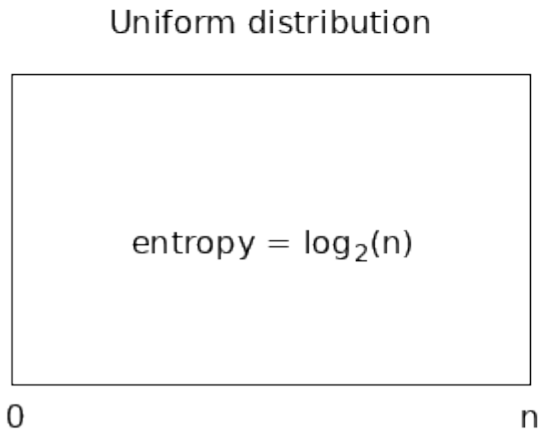


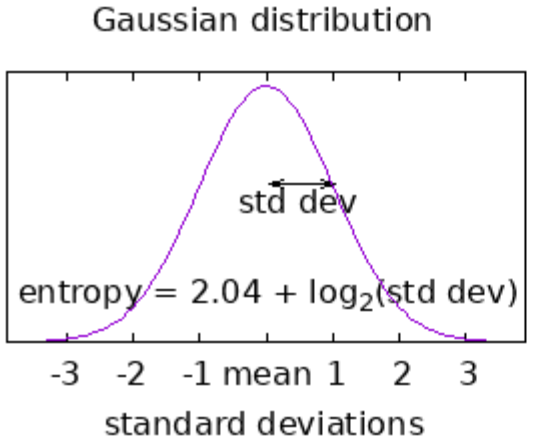
Information content, entropy = $-\Sigma\ p_i \log_2(p_i)$

Entropy = - Σ_i p_i $\log_2(p_i)$

- Entropy of uniform distribution of n values = $\log_2(n)$
  - $-\Sigma_{i=1}^{n}$ 1/n $\log_2(1/n)$ = $-\log_2(1/n)$ = $\log_2(n)$
- Uniform distribution 0 to 255, entropy = $\log_2(256)$ = 8 bits
- Entropy of Gaussian distribution standard deviation **σ**

  = 2.04 + $\log_2(\sigma)$
- In practice entropy is a robust slow moving function.
  A real distribution may have an entropy near that of a Gaussian distribution with the same standard deviation.





Information inside program. Entropy during execution

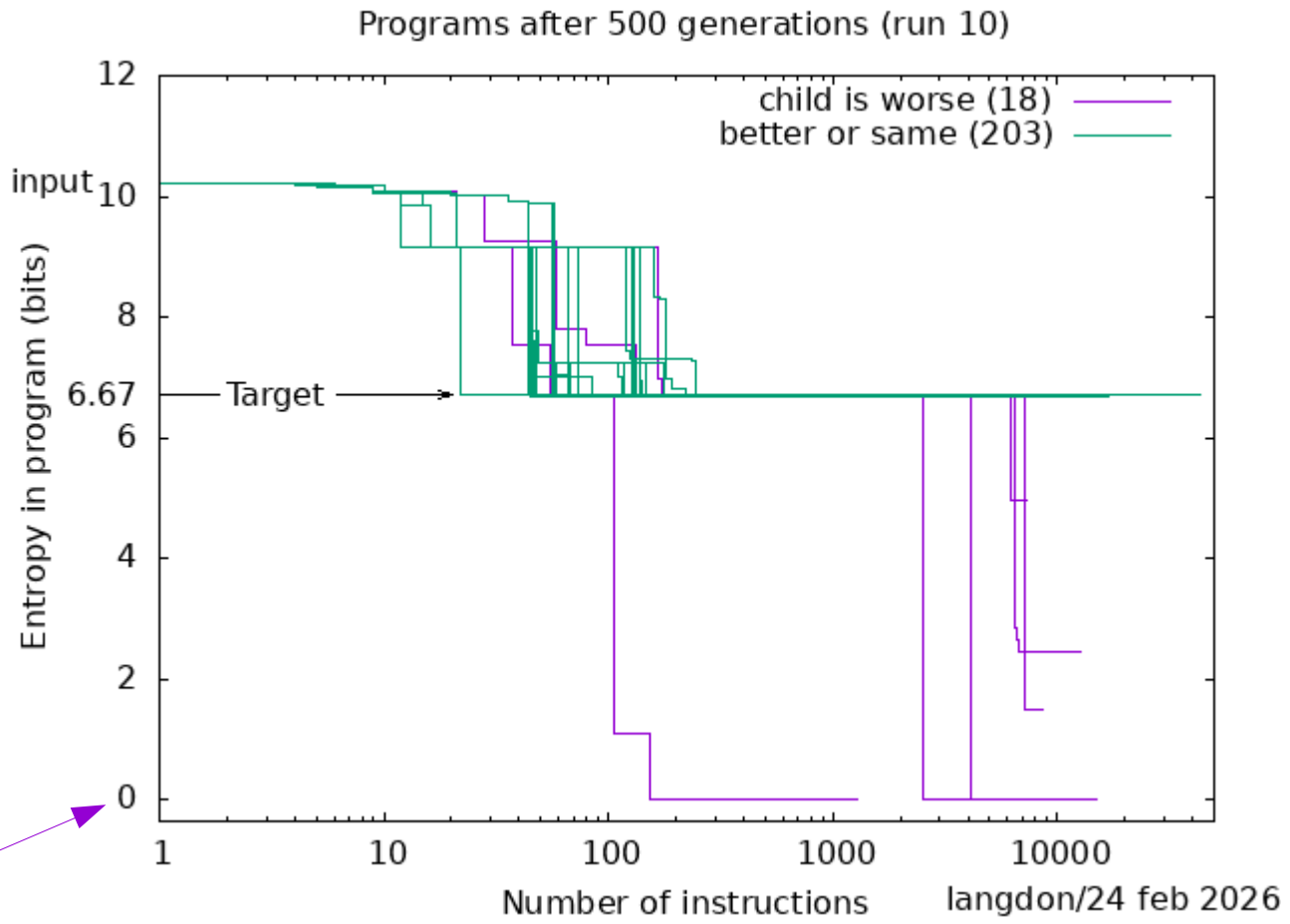
- Program run on 1201 test cases
- Start program by loading test inputs into registers
- Entropy at start of execution = entropy of distribution of inputs
- Internal information is distribution of program's memory states
- Output values also have a distribution
- Deterministic batch program writes answer at end and terminates; no more information added whilst running
- Final entropy ≤ start entropy
  - Intrinsic test oracle.
  - Automated watch dog (WDT)
    - eg send email or reboot if entropy violates ≤
- Output entropy ≤ final entropy ≤ input entropy

Entropy during execution

- Program run on 1201 test cases (30 are identical). Entropy of inputs = 10.22 bits (< $\log_2(1201) = 10.23$)
  - Could just run 1171 unique inputs
- Here programs state held in eight 8-bit registers.
  Max storage 64 bits (entropy 64, uniform $n=2^{64} = 1.8\ 10^{19}$)
- As program runs registers are over written so number and distribution of unique memory patterns changes, so entropy changes (it can only decrease).
- At end information in program must ≥ information in target, otherwise cannot hope for right answer.
- NB if internal entropy < target, program cannot be good.
  - Test oracle. Stop early.
- True in general not just genetic programming

- Consider fit programs after 500 generations

Entropy as programs execute (gen 500 run 10)

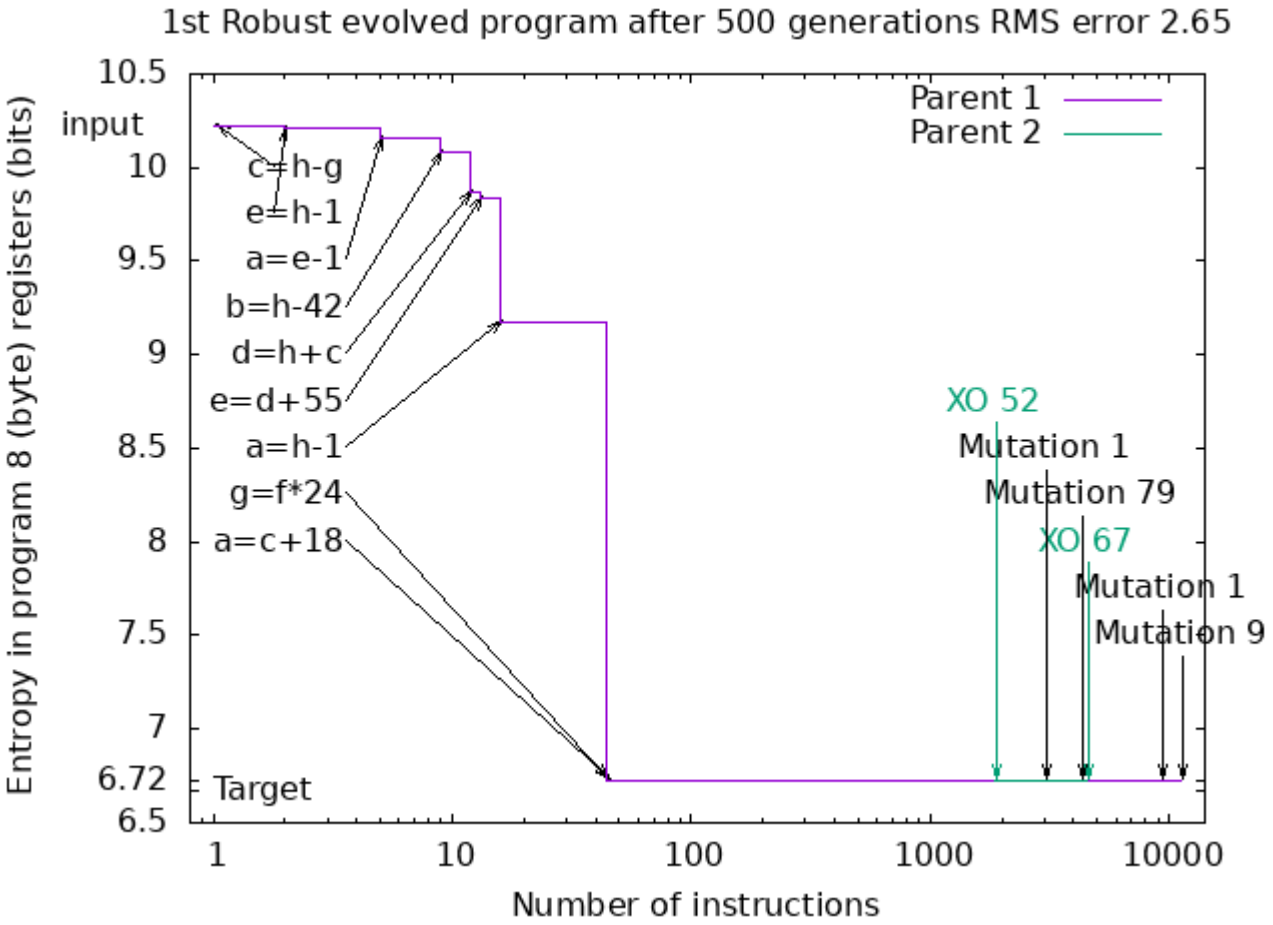


0 ie same memory pattern on all 1201 test cases.
Output is a constant.
No ability to predict.

Child is parts of 2 Parents and Mutations

- A fit program from generation 500
- Start of child and its execution identical to 1st parent
- Crossover inserts random part of 2nd parent
- This example, 4 random instruction mutations
- Here, none of this changes internal memory distribution (entropy unchanged).
- Also child's fitness is same as parent

Child's entropy as parent (gen 500 run 10)

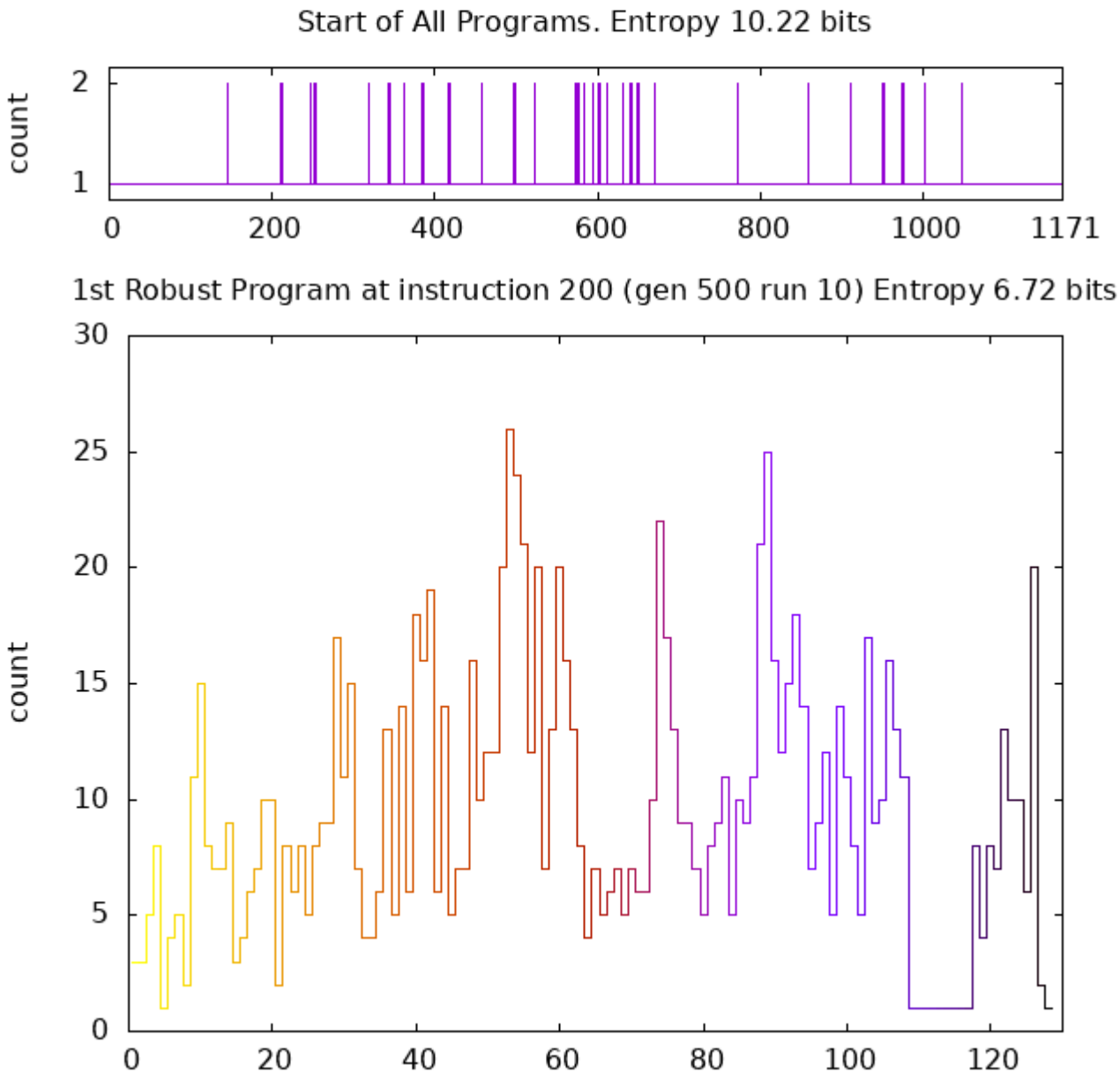


Only 8 of 11,326 instructions loose entropy

Child is parts of 2 Parents and Mutations

- A fit program from generation 500
- After 45 instructions 1171 unique memory states reduced to 128.
  - Could just continue execution on 128 test cases
- Some 8x8 bit memory patterns more often than others
- Entropy of distribution is 6.72 bits
  - Very close to entropy of target (6.67 bits)
  - Standard deviation of distribution is σ=32.54
    (entropy of Gaussian σ=32.54 is $2.04+\log_2(32.54) = 7.07$)

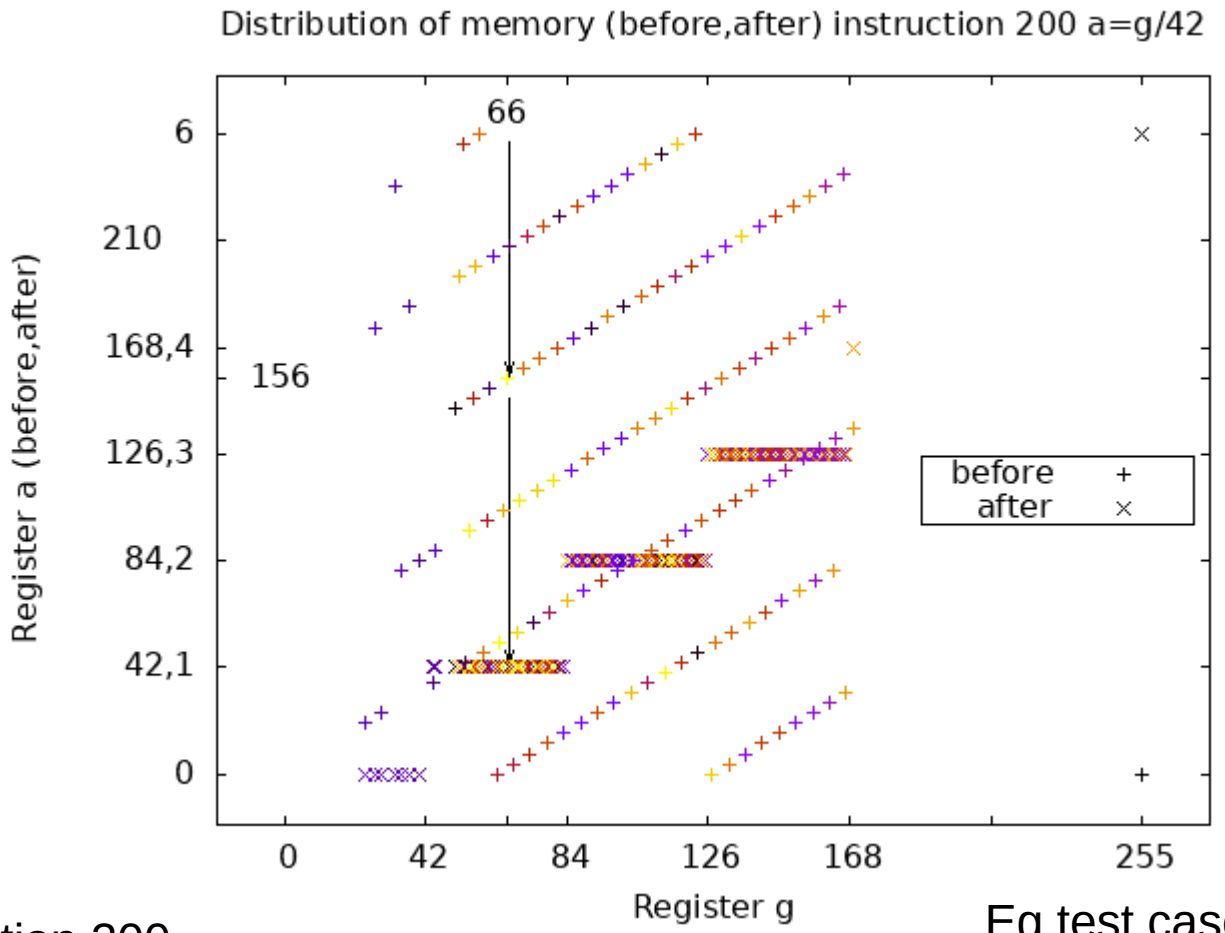
Memory content distribution during execution



Irreversible yet Isentropic, a=g/42

- Randomly choose instruction 200, a=g/42
- Register a, range 0 to 252 becomes 0 to 6
- No other registers changed
- Values in memory changed BUT number (128) of memory patterns when running 1201 test cases same. Also distribution of memory values unchanged. So entropy the same (6.72 bits).

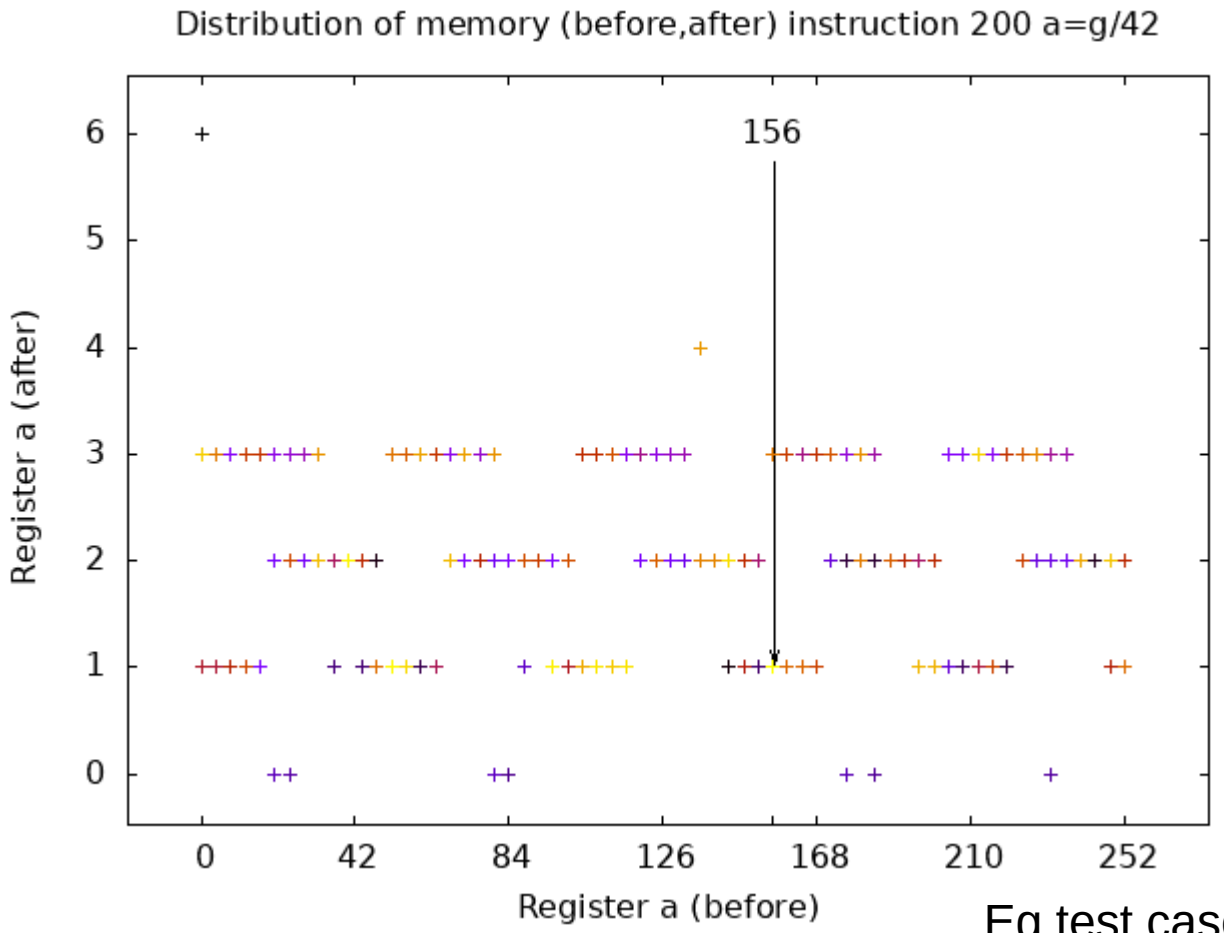
Memory content instruction 200 a = g/42



By instruction 200 registers a and g are strongly related

Eg test case 1. a was 156, g=66. Now a=1

Memory content instruction 200 a = g/42

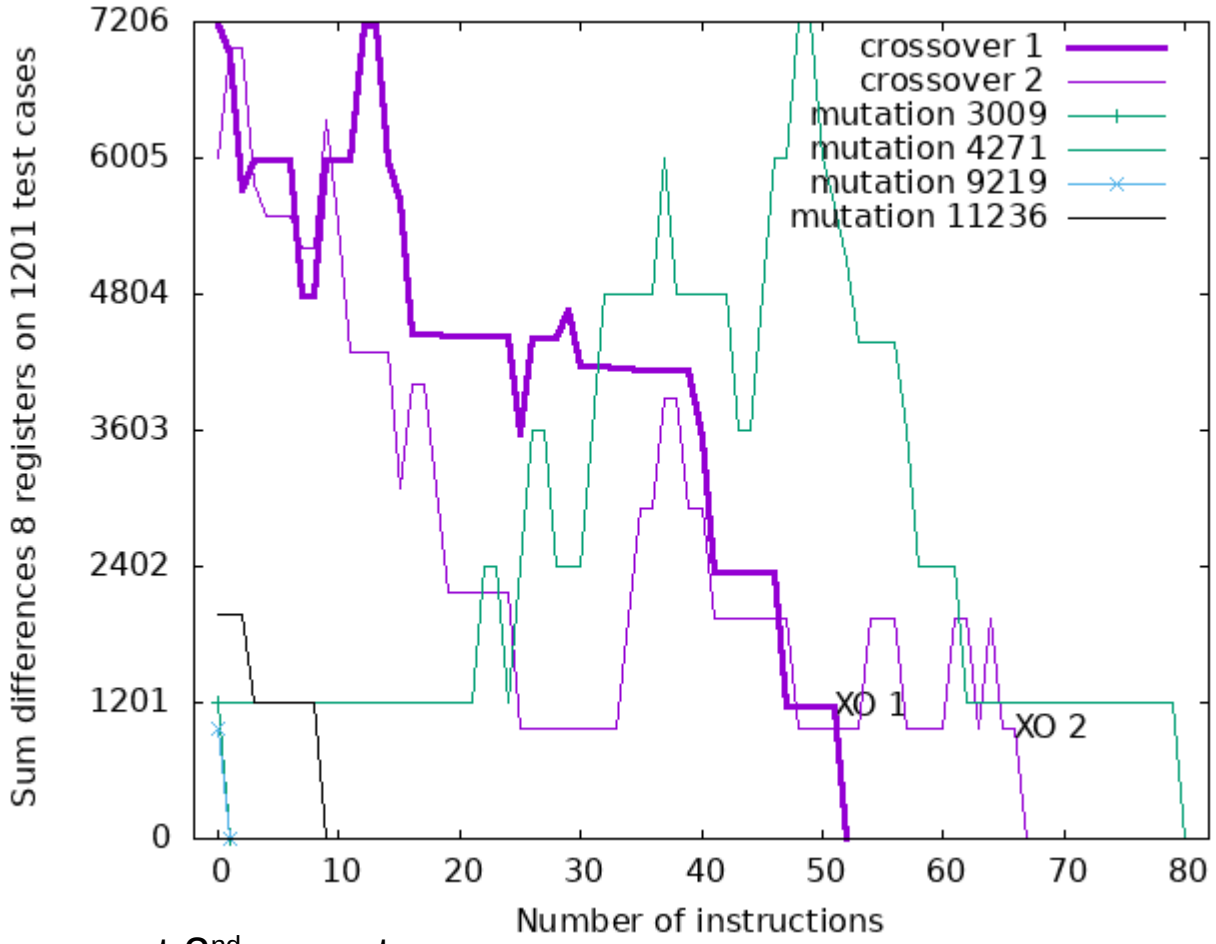


Eg test case 1.
a was 156,
g=66.
Now a=1

Crossover boundaries & Point mutations

- Child composed on two parent's code and four mutations, yet gives identical answers
- 1st crossover (XO1) child starts executing random part of 2nd parent.
  - Disruption lasts 52 instructions BUT after is identical
- 2nd crossover (XO2) child returns to executing random part of 1st parents code.
  - Disruption lasts 67 instructions BUT then identical to 1st parent
- 4 mutations, each cause random code changes.
  - Disruption 1 to 79 instructions long.

Temporary Disruption at Crossover and 4 Mutations

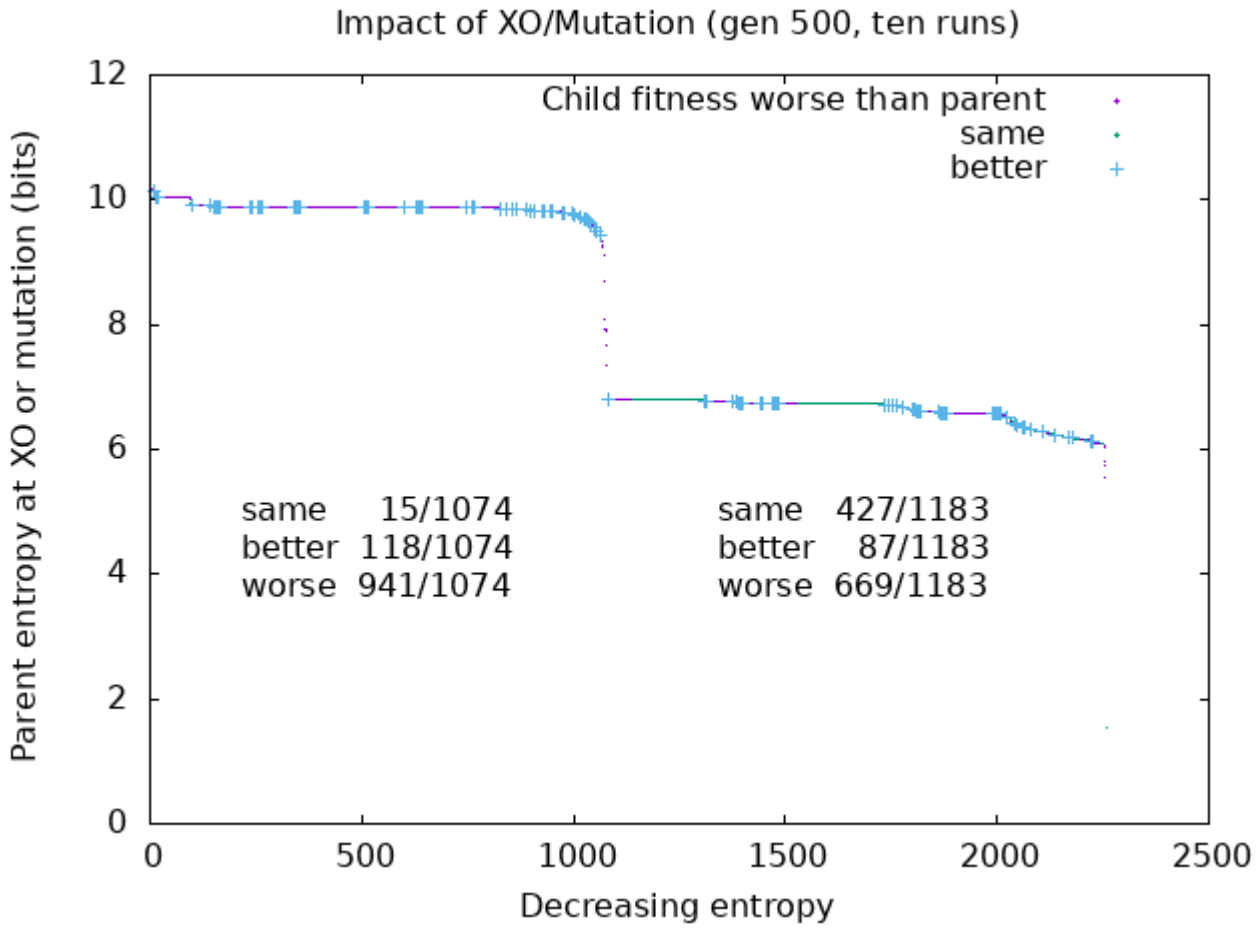


XO1 differences wrt 2nd parent

Innovation rate 1/length

- Resilient low entropy tail suggests crossover and mutations in tail tend not to improve fitness.
- If there is a more-or-less fixed sized sensitive region, chance crossover or mutation in it would fall in proportion to program length.
- Linear regression of all ten runs suggests size of sensitive region varies between runs.
- But entropy analysis suggests crossover or mutation in high entropy code more likely to change fitness

Fitness Disruption in High Entropy Code (gen 500, 10 runs)



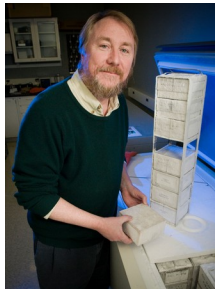
By generation 500,
If 1st genetic change where memory distribution entropy ≈ input entropy
then almost all crossovers change child output.

If entropy ≈ target entropy
36% crossovers no change.

Long Term Evolution Experiments with Linear Genetic Programming

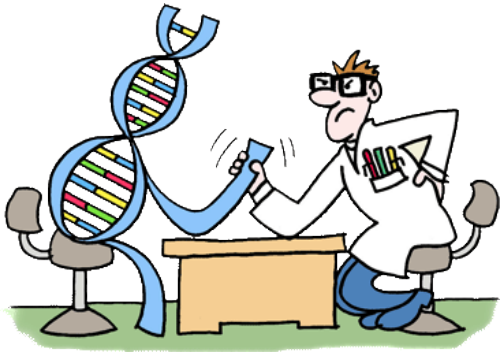
- Long term evolution experiments are possible in days not years
- Long term innovation is possible but slows with increase in size
- Linear GP is sensitive near inputs (rather than outputs tree GP)
- Expect similarities with hand made programs, feedforward ANN
- Good programs loose entropy towards their desired output
- Entropy loss might be an implicit test oracle
  - Detect unexpected behaviour
  - Terminate early bad programs
  - Trigger watch dog circuit
  - Specify desired program before it is (automatically) written
- Clustering memory patterns during execution might reduce testing
- Be ambitious

Take nobody's word for it

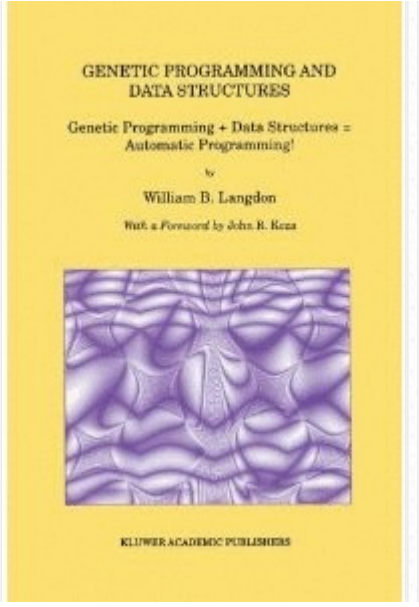


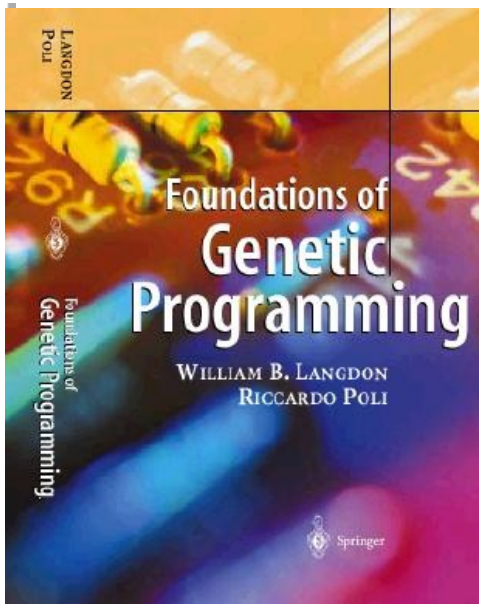
LTEE

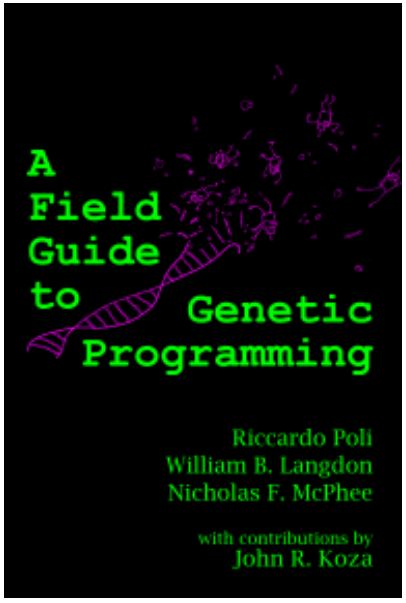
Do the impossible



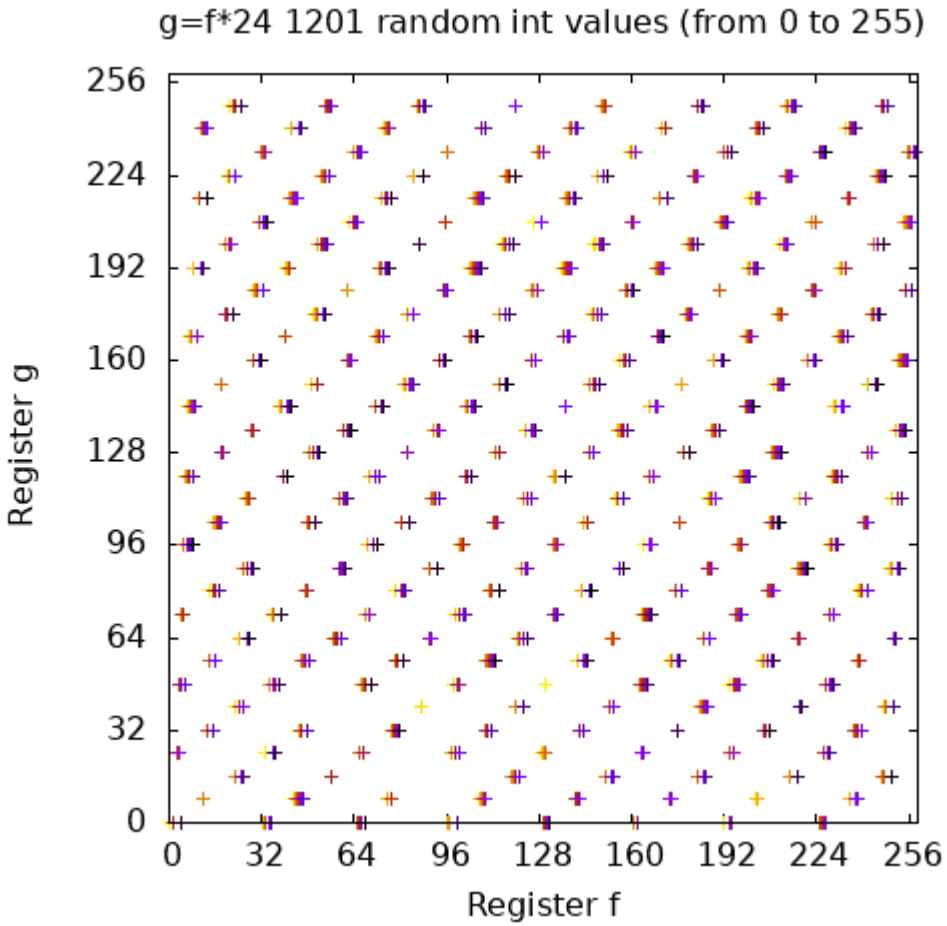
$10000 Humies
submit by
Friday 29 May 2026







Random Distribution g=f*24 (high entropy conserved)



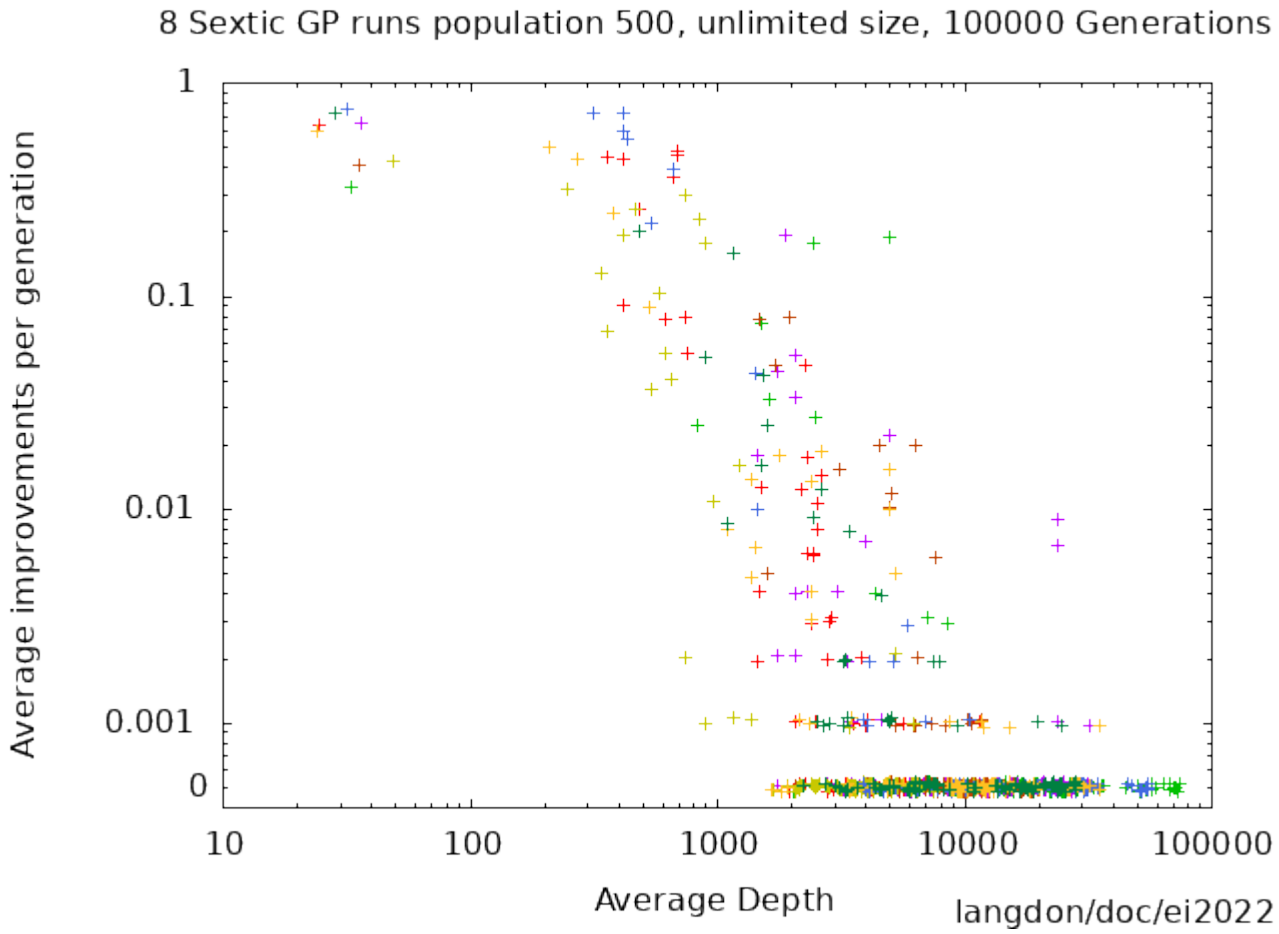
Input 1201 eight random byte values (entropy 10.2234)
g = f*24 (no entropy change).

A little noise added to spread data horizontally.

Tree GP Long Term Evolution Experiments

- **LTEE** shows E.Coli continued innovation 80000 generations
- Genetic Programming continued fitness improvement a million generations BUT GP slows as expressions get deeper
  - Impact of mutations lost, e.g. due to rounding error
  - In deep integer trees 92% to 99.97% of evaluation changes have no effect
- Exponential decay with depth
  - Need to be close to error for tests to find them
  - On average <7 more than 50% errors detected
- Each case may differ in detail but information theory says entropy loss will occur and will lead to robustness

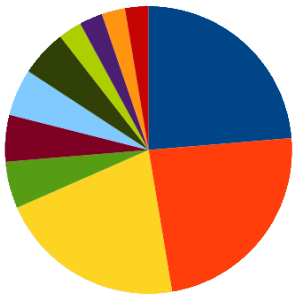
Deeper programs harder to evolve



As the GP populations evolve they find thousands of improvements but at a slower rate as the trees get deeper. Note log scales.

Conclusions: entropy loss gives robust code

- Importance of reminding everyone of GP achievements
- Genetic Programming can be creative, GP is in use
- Computing is not reversible, causing entropy loss, which gives failed disruption propagation, 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
  - Shallow code unit tests preferred to deep system tests
- Be ambitious



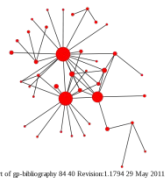
Do the impossible

The Genetic Programming Bibliography

18431 references, 19000 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

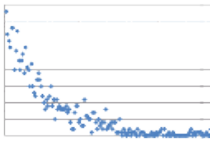
Co-authorship community.



Downloads

Downloads by day

A personalised list of every author's GP publications.



Your papers

blog

Googling GP bibliography, eg:
software testing site:gpbib.cs.ucl.ac.uk

Text search