random_tree() is a linear time and space C++ implementation able to create trees of up to a billion nodes for genetic programming and genetic improvement experiments. A 3.60GHz CPU can generate more than 18 million random nodes for GP program trees per second.
Why do we care?

• Tree universal data structure
• Space of trees is huge but sampling it is not simple
• Lessons
  – Linear O(n) 1999 code fine for trees of 100 nodes not usable with trees of a million nodes
  – Solving problem with real pseudo random number generators (PRNGs)
    • although we are in a state of sin (Von Neumann), call it random from now on.
Back Ground 1996-2000

• Genetic Programming needs random trees
  – Random start to the population
  – Subtree replacement mutation

• John Koza propose “ramped half and half”

• Walter Bohm and Andreas Geyer-Schulz [FOGA 4] and Hitoshi Iba [PPSN 96] suggest uniform random trees.

• 1997 I implement Iba’s in Andy Singleton’s GPquick

• Sean Luke [2000] says not fast
  – Who cares trees tiny trees, over head small.
Back Ground 2019

- Applying Genetic Improvement to speed up existing GP system: **COW62**
  - To speed up eval of very bloated big $10^8$ trees
- Only need binary trees (for now)
- Need lots of different big trees like those evolved by Genetic Programming (GP)
- Want random trees since:
  - Humungous GP trees take weeks to evolve
  - Without compression gigabytes each
- Existing (linear code) takes far too long.
- Code rewritten
Evolved GP Trees

- Picture binary tree
- Picture GP population evolving
Binary GP Trees
Binary GP Trees

101 nodes
Binary GP Tree

1001 nodes
Binary GP Tree

10001 nodes
GP population Evolving

6-Mux 500 binary trees (run 100)

Flajolet parabola
Generation 100

Tree size vs. Tree depth

langdon/talks/gecco2017 video
Mathematics of Large Trees

Combinatorics of large trees hard but well studied

Robert Sedgewick and Philippe Flajolet

[analysis of algorithms]

Number of binary trees is Catalan number

\[ N = \frac{(\text{size}-1)}{2} \]

Large \( N \) limit

\[
T_N = \frac{1}{N+1} \binom{2N}{N} = \frac{4^N}{\sqrt{\pi N^3}} \left(1 + O\left(\frac{1}{N}\right)\right).
\]

\[
2\sqrt{\pi N} + O(N^{1/4+\epsilon}) \approx \sqrt{(2\pi \text{ size})}
\]
Distribution of binary trees

Exponential $\approx 2^{\text{depth}}$
Parabola $\approx \text{depth}^2/2\pi$
Linear $2^{\text{depth}+1}$

W. B. Langdon, UCL
Number of possible binary trees

Distribution of Binary Tree Shapes

$log_{10}$ number of binary trees

Universe = $10^{80}$

Tree size
Large Evolved GP Trees

- Genetic Programming trees not random but random shape.
- Random trees good enough to test and time each new mutated eval() function.
Sampling Large Trees

• Sampling trees is **hard**
• Mathematical cheat: sample random permutation of size n, translate chosen permutation into tree
• Proof that sample permutation uniformly at random, so have sampled random trees uniformly at random.
1997 (Small) random Trees

- Original implementation followed Iba’s technical report ETL-TR-95-35 and existing random permutation C code.
- Not efficient. Linear time $O(n)$ but not fast. Instead deal with hard maths by keeping code modular.
- Also needed to deal with functions with up to 4 arguments (arity 0..4)
Sampling Permutation for Binary Tree

• Original code slow because general
• Simplify for binary tree
  – n/2 functions, n/2+1 leafs
• Start with empty sequence int dyck[n]
• Deterministically load n/2 node(a=2) and n/2+1 leafs (arity=0).
• Randomise sequence, dyck
• New: use Knuth shuffle
Sampling Permutation for Binary Tree

- Use Knuth shuffle to randomise sequence
- Prove that Knuth shuffle samples uniformly at random, hence have random permutation, hence will have random tree
- With real PRNG, need to help Knuth?
- Alternate 0,2 initial sequence (don’t start with all 0 at one end). Then shuffle.
- Number of random trees far bigger than number of PRNG sequences.

W. B. Langdon, UCL
Randomising sequence for real

Knuth shuffle unbalanced start

Knuth shuffle uniform start
Permutation into Tree via onedom()

• Sampling trees is hard
  – Odd language dyck sequence, 1-dominated
• For binary trees, sequence in n/2+1 square
  – Add leaf move horizontally.
  – Add function move vertically.
  – All random permutations move from start to End (albeit by different routes, next slide).
• Exactly one rotation makes route a valid tree.
• Routine onedom (1-dominated) converts random permutation into corresponding random tree.
Onedom() permutation into tree

Add leaf move horizontally.
Add function move vertically.

All random permutations move from start to end (albeit by different routes).

Onedom scans sequence. For each (x,y) find distance below diagonal. Keep last max. Reorder sequence to start at max.

Permutation into Tree

- It turns out to be easier to trace route backwards
- From end: Add E, 6, D, C, 5, B, A, 4, 3, 2, 1
Permutation into Tree

- It turns out to be easier to trace route backwards
- From end: Add E,6,D,C,5,B,A,4,3,2,1
Tree into GP Program

- Label tree with GP functions and leafs
  - If function (2) chose random function
  - If leaf (0) chose leaf at random
- Return GP tree and its depth
Depth an additional benefit

- Some data structures in eval() depend on depth of tree
- The distribution of random trees is known in advance so can conservatively set tree depth from known size of tree.
  - depth < mean + multiple of standard deviation
  - depth < 10×√(size)+100
- Estimated tree depth much smaller than worst case (size/2) but still wasteful.
- New code returns exact depth.
Does it work
Large Random Trees

![Graph showing the relationship between tree depth and tree size for Flajolet's method and Random_tree()](image)
Speed of Large Random Trees

![Graph showing the speed of large random trees with a linear relationship between tree size and time. The graph indicates that 18,600,000 nodes can be processed per second for Random_tree().]
UCL Computer Science Research Notes

- UCL CS RN more than 20 years history
Conclusions

• Can generate large trees quickly
  – up to 2 billion

• Original system was linear $O(n)$, still $O(n)$
  but now usable for big trees
  – big O notation only gets you so far

• Applied to do fitness testing of genetic
  programming system GPquick
  – next week COW62

• Scope for further speed up if wanted

• C++ code available via www

• depth $\approx \sqrt{2\pi \text{size}}$
GI 2020

Genetic Improvement of Software
http://www.human-competitive.org/
Workshop at ICSE 2020
- Position papers (1 or 2 pages)
- Research papers (up to 8 pages)
Submissions due 22 January
https://icse20-gi8.hotcrp.com/

Humies
http://www.human-competitive.org/
Awards for Human-Competitive results
$10,000 prizes
Presentation at GECCO-2020 in Cancun, Mexico
send email before 29 May to goodman@msu.edu
Discussion

Dr. David Clark suggested reference [1]

[1] Robert Feldt and Simon M. Poulding, Finding test data with specific properties via metaheuristic search, in International Symposium on Software Reliability Engineering (ISSRE 2013), 350-359. doi:
Genetic Programming

W. B. Langdon

CREST

Department of Computer Science
The Genetic Programming Bibliography

http://gpbib.cs.ucl.ac.uk/

13435 references, 12000 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

RSS Support available through the Collection of CS Bibliographies.

A web form for adding your entries.
Co-authorship community. Downloads

A personalised list of every author’s GP publications.

blog

Search the GP Bibliography at
http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html