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

W.B.Langdon@cs.ucl.ac.uk
http://www.cs.ucl.ac.uk/staff/W.Langdon
14:00–15:50 Bill Langdon

• Tutorial based on *Foundations of Genetic Programming*


• 1. Introduction

• 2. Fitness Landscapes

• 7. and 8. The Genetic Programming Search Space
  *New* rates of convergence and limits.

• 9. Empirical: Santa Fe Ant

• 10. The MAX problem

• 11. Genetic Programming Convergence and Bloat

• Conclusions
Fitness Landscapes

- Metaphor: Search space like countryside (2 dimensions!!) height is fitness.
  Mountains have high fitness
  Metaphor can be inverted \(\sim\) Valleys and seas are good points.

- Enumeration, exhaustive exploration

- Shortsighted Hill climbing. Local optima, swamps and plateaus. Basins of attraction.

- Simulated annealing, uphill but downward steps probabilistically allowed. Chance depends exponentially upon ratio of height of backward step and current “temperature”

- Genetic Algorithm, metaphor fails with crossover
Explore whole search space is systematic fashion.

(Monte Carlo: sample search space at random)
Choose start point at random

Find local gradient

Follow local gradient up hill

May get stuck at top of small hill (known as “local optima”, “false peaks”, “deceptive peaks” etc.)
Two major basins of attraction.

In this case higher peak also has larger area from which a hill climber will reach it.

In swamp explorer has no gradient to guide him.
Smooth or Rough Landscape

Smooth: hill climber moves rapidly to single hill top (Fuji).

Rugged landscapes with many hills

If hills isolated by deep valleys the hill climber worse than if interconnected by mountain ridges.

But if the ridges are narrow and lead downwards our explorer will still have difficulties

W. B. Langdon
Long Paths

In this landscape the local gradient may eventually lead to the summit but the path is much longer than the direct path.

W. B. Langdon
Other users of Landscape Poetry

Smarter explorers may not be as short sighted and so make assumptions about the smoothness of the landscape to make large jumps towards were they calculate the peak “should be” (if their assumptions are correct).

In simulated annealing, sometimes allowed to climb down hill.

On detecting a local peak, start exploring again from a new randomly chosen start point.

Tabu search can be thought of as keeping a “tabu” list of places the explorer should not revisit.

In A* (and other AI search) once a search is underway it may be possible to exclude large areas of the search space (by using heuristics i.e. knowledge about the problem)

For example find cheapest route between two cities, A* stops exploring any partial route which already costs more than a viable route it has previously found.
Landscape using binary coding. This is the same landscape as slides 4, 5 and 6 but using a binary coding rather than a Grey coding.

The captions (“Peak” etc.) refer to the original figures. Recoding the parameters changes the topology of the landscape and, in this example, introduces more local peaks.
Fitness Landscape Failings

- overlook long range correlation (conceal useful regularities)

- only two horizontal dimensions

- Single objective, can’t deal with both speed and battery life

- GA etc. often binary parameters not continuous

- “Landscape” depends upon representation and operators
  Good for two dimensions and small mutations.
  Hard to visualise discrete search spaces or crossover

- Fixed landscape assumption, what of time varying?
  “Effective fitness” landscape changes as population moves across it
Scaling of Program Fitness Spaces

- Genetic Programming stochastic search for programs

- What is known about the space of all programs

- Above threshold, proportion of functions of each type independent of length

- Experimental evidence, tree based GP

- Proof linear, e.g. machine code GP

- summary tree based GP

- So what?

W. B. Langdon
Number of programs v. size, various problems

![Graph showing the relationship between the number of programs and program size for different problems such as Quintic, Sextic Polynomial, 11 Multiplexor, 6 Multiplexor, and Binary Trees. The x-axis represents program size on a logarithmic scale, while the y-axis represents the number of programs also on a logarithmic scale. The graph illustrates how the number of programs increases as the program size grows, with different line styles representing each problem type.]
Distribution of Binary Trees by size and height

Tree Depth

Number of internal nodes and terminals

5% peak

95% full

Minimal

Mean and SD

Flajolet

W. B. Langdon
Distribution of Binary Trees by size and height
Proportion of NAND trees:

2 input logic function
Fitness Sextic Polynomial

(constants and input equally sampled)

W. B. Langdon
Artificial Ant

![Graph showing the relationship between fitness, program length, and proportion.](image)
• Input and output registers part of memory.
• Memory initially zero (except input register).
• Linear GP program is a sequence of instructions.
• CPU fetches operands from memory.
• Performs operation.
• Writes answer into memory (overwriting previous contents).
• Programs stops after $l$ instructions.
• Final answer in output register.
Why Interest in Random Programs?

- Consider all programs, of a chosen length.

- Create a large number of random programs, measure their properties

- We are sampling the search space of all possible programs

- Bigger sample $\sim$ better estimate of actual

- We are interested in Markov processes because analysis (rather than experiment) can give provable general results a) in the limit and b) the rate at which practical systems approach this limit.
Why are Random Programs Markov?

• A Markov process only depends on current state

• When a program is check pointed, its state is saved
  It can be restarted, without ill effect, if its state (i.e. content of memory) is restored and it restarts from the same point.
  I.e. what happens later only depends on current memory

• At each time step $t$ a Markov process is in a state, $i$
  Randomly chose another state $j$ for the next time step, $t + 1$.
  Process is Markov if probabilities associated with each transition do not change with time, only depend on current state.
  Matrix $M = \text{probability of transition from state } i \text{ to } j$.
  $M$ does not change with time.

• Executing a random program is a Markov process, whose state is the contents of the computer’s memory.
Proof Linear: Model of Computer

- State of computer given by contents of memory

- All memory, registers but exclude PC

- $N$ memory bits $\Rightarrow 2^N$ states

- Execution $\equiv$ state $\rightarrow$ next state

- In general state $\neq$ next state but allow state $=$ next state

- Computer designed so all states accessible

- Symmetric instruction set, state $\leftrightarrow$ next state
Proof Linear: Execution of computer program

- state$_0$ given by inputs

- Program = sequence of instructions, change state

- program $l$ states long

- terminates at state state$_{l-1}$

- program itself need not be linear
  
  branches, loops, function calls OK provided executes random instructions
Instructions as Transformation Matrices

- |probability vector| = $2^n$

  \[ v = \underbrace{0, 0, \ldots, 1, 0, \ldots, 0}_{2^n \text{ elements}} \]

- At any time $t$ in one state $i \Rightarrow v^t_i = 1$ and $v^t_j = 0$

- Each instruction = $2^n \times 2^n$ matrix

- $v^{t+1} = v^t N$

- Every $N_{ij} = 0$ or 1, N is stochastic

*Stochastic matrices have the property that each of their elements are not negative and the elements in each add up to one. “Stochastic” does not mean they are random!
All Programs

- All possible programs of $l$

  average vector $u = \text{Mean of all } v$

  $u^{t+1} = u^t M$ where $M$ is average instruction matrix

- $u$ is Markov, $M$ is stochastic

  At least one $M_{ii} \neq 0$

  period of state $i = 1$ i.e. it will be aperiodic [Feller, 1970]

  Greatest common divisor (g.c.d) of all states $= 1$

- All states can be reached $\Rightarrow M$ irreducible

- Irreducible ergodic Markov chain $\Rightarrow \lim_{t \to \infty} u^t = u^\infty$

  independent of the starting state (i.e. the program’s inputs)
An Illustrative Example

- Two Boolean registers $R_0$ and $R_1$
- Each initially loaded with an input
- Program’s answer is given by $R_0$
An Illustrative Example: Instruction Set

- There are $2^2 = 4$ states ($R_1R_0 = 00, 01, 10, 11$)

- There are eight instructions

Eight transformation (4 × 4) matrices

\[
\begin{align*}
R_0 &\leftarrow \text{AND} & R_1 &\leftarrow \text{AND} & R_0 &\leftarrow \text{NAND} & R_1 &\leftarrow \text{NAND} \\
1 0 0 0 & 1 0 0 0 & 0 1 0 0 & 0 0 1 0 \\
1 0 0 0 & 0 1 0 0 & 0 1 0 0 & 0 0 0 1 \\
0 0 1 0 & 1 0 0 0 & 0 0 1 0 & 0 0 1 0 \\
0 0 0 1 & 0 0 0 1 & 0 0 1 0 & 0 1 0 0 \\
\end{align*}
\]

\[
\begin{align*}
R_0 &\leftarrow \text{OR} & R_1 &\leftarrow \text{OR} & R_0 &\leftarrow \text{NOR} & R_1 &\leftarrow \text{NOR} \\
1 0 0 0 & 1 0 0 0 & 0 1 0 0 & 0 0 1 0 \\
0 1 0 0 & 0 0 0 1 & 1 0 0 0 & 0 1 0 0 \\
0 0 0 1 & 0 0 1 0 & 0 0 1 0 & 1 0 0 0 \\
0 0 0 1 & 0 0 0 1 & 0 0 1 0 & 0 1 0 0 \\
\end{align*}
\]
• Example $R_0 \leftarrow \text{AND}$

$R_1 = 1, R_0 = 0$ \quad u = (0 \ 0 \ 1 \ 0)$

$v = uM = (0 \ 0 \ 1 \ 0) \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = (0 \ 0 \ 1 \ 0)$

$R_1 = 1, R_0 = 0$

I.e. $\text{AND}(0,1) = 0$, so $R_0$ is set to 0 while $R_1$ is unchanged

• If we use each of the instructions with equal probability the Markov transition matrix is the average of all 8, i.e.

$$M = \frac{1}{8} \begin{pmatrix} 4 & 2 & 2 & 0 \\ 2 & 4 & 0 & 2 \\ 2 & 0 & 4 & 2 \\ 0 & 2 & 2 & 4 \end{pmatrix}$$
An Illustrative Example: Limiting Probabilities

- The limiting distribution \( u_\infty = 1/4(1,1,1,1) \) is given by the eigenvector corresponding the largest eigenvalue (which always has the value 1).

The eigenvalues \( \lambda \) and corresponding eigenvectors \( E \) of \( M \) are

\[
\begin{align*}
\lambda_{00} &= 1/2(0 \quad -1 \quad 1 \quad 0) \\
\lambda_{01} &= 1/2(-1 \quad 0 \quad 0 \quad 1) \\
\lambda_{10} &= 1 (1 \quad 1 \quad 1 \quad 1) \\
\lambda_{11} &= 0 (1 \quad -1 \quad -1 \quad 1)
\end{align*}
\]

Note since \( M \) is symmetric the other eigenvalues are also real.
Rate of Convergence and the Threshold

• The Rate of convergence is dominated by the second largest (absolute magnitude) eigenvector of $M$, $\lambda_2$

• The smaller $\lambda_2$ is the faster the actual distribution of functions converges to the limiting distribution

• I.e. the smaller is the threshold

• Threshold size $\approx -1/ \log |\lambda_2|$

Convergence rate depends crucially on type of computer and size of its memory [Langdon, 2002].
Extend to Functions

We have proved distribution of outputs tends to limit.

Formally need to extend this to the distribution of functions.

There is a limiting distribution of program functionality.

Uniform distribution of outputs \( \not\Rightarrow \) uniform distribution of functions.
Functions Example

One Boolean register. \( (N = 1 \text{ so } 2^{N^2} = 4 \text{ possible functions}) \).

Suppose our machine has 4 instructions: CLEAR, NOP, TOGGLE, SET.

Two outputs (0 and 1) both equally likely.

\[
\begin{array}{cccc}
\text{CLEAR} & \text{NOP} & \text{TOGGLE} & \text{SET} \\
1 0 0 0 & 1 0 0 0 & 0 0 0 1 & 0 0 0 1 \\
1 0 0 0 & 0 1 0 0 & 0 0 1 0 & 0 0 0 1 \\
1 0 0 0 & 0 0 1 0 & 0 1 0 0 & 0 0 0 1 \\
1 0 0 0 & 0 0 0 1 & 1 0 0 0 & 0 0 0 1 \\
\end{array}
\]

\[ \begin{pmatrix} 1/4 \\ 2 0 0 2 \\ 1 1 1 1 \\ 1 1 1 1 \\ 2 0 0 2 \end{pmatrix} \]

The limiting distribution (eigenvector with eigenvalue=1) of the functions is

\[ 1/2 (1 0 0 1) \]  

I.e. 50% CLEAR and 50% SET (not uniform).
What is the Limiting Distribution?

The limit depends upon the computer type. If we restrict ourselves, the eigenvalues and eigenvectors of the Markov matrices may already be known or maybe we can discover them.

1. Cyclic. Increment, decrement and NOP. Reversible but not universal \cite{Langdon2002, Langdon2003a}.

2. Bit flip. Flip bit $i$ and NOP. Reversible but not universal \cite{Langdon2002, Langdon2003a}.

3. Any non reversible . \cite{Langdon2002a, Langdon2002b, Langdon2003a}.

4. Any reversible \cite{Langdon2003b}.

5. CCNOT (Toffoli gate). Reversible and universal \cite{Langdon2003b}.

6. The “average” computer \cite{Langdon2002a, Langdon2002b, Langdon2003a}.

7. AND, NAND, OR, NOR. Not reversible but universal \cite{Langdon2002a, Langdon2002b, Langdon2003a}.

Program Outputs Limiting Distribution

In general the distribution of outputs of any computer will converge to a limiting distribution but programs may need to be exponentially long.

The cyclic computer shows not only is the upper bound exponential but that it can be reasonably tight in that exponentially long programs can be required for the distribution to be close to the limit. \[ l > 0.8 \frac{3}{4\pi^2} 2^{2N} \]

However bit flip, average and four Boolean computers show in some cases the output distribution of much smaller programs is close to the limit.

\[ \frac{1}{4}(N+1)(\log(m)+4), \quad l \leq (15 + 2.3 \ m)/\log I, \quad l \leq \frac{1}{2}N(\log(m)+4). \]
Non Reversible Programs –
Limiting Fitness Distribution is Zero

Linear systems, where the inputs are not write protected, on average lose information. This means in the limit the fraction of programs implementing interesting functions goes to zero.

I.e. almost all non reversible linear programs return one of \(2^m\) constants.

In general programs need to be exponentially long for fitness distributions to converge. In cyclic computers the upper bound is tight but in some cases (e.g. AND NAND OR NOR) programs can be much smaller and still be close to the limiting distribution.
Reversible Program –
Limiting Fitness Distribution is Gaussian

In the limit of long programs, with large reversible computers both every output and every possible (i.e. reversible) function are equally likely.

With a Hamming distance fitness function, fitness follows a Normal (Gaussian) distribution.

This means almost all programs have near average fitness.

And the fraction of solutions is exponentially small (but bigger than zero).

CCNOT gates show reversible programs need not be desperately big before their fitnesses is Normally distributed [Langdon, 2003b].
Distribution of Reversible Program

Fitness on 6 Multiplexor, 6 spare lines

Gaussian $\sigma=4$

100 CCNOT

W. B. Langdon
Convergence of Effect of Mutation

In general the effect on the outputs of a single point mutation falls at least as quickly as $l^{-1}$ but the bound on the convergence threshold length is exponential in the number of fitness tests [Langdon, 2003a].

However in two cases (cyclic and bit flip), if we consider changes in fitness, the impact of mutation on fitness is independent of program size. I.e. convergence is instantaneous rather than requiring exponentially long programs.

The fitness impact of point mutation on the “average” computer falls as $l^{-1}$ but the bound on the convergence threshold length is exponential in the size of the computer.

The cyclic and bit flip computers are simple enough to allow analysis of the time to solution (quadratic or faster).
Summary: Big Random Tree Programs

- Above a threshold, distribution of performance is independent of tree size.

- Most trees are asymmetric. The chance of finding a leaf near the root is \( \approx 50\% \).

- Even if instruction set is symmetric, some functions are more likely than others.

- Solutions to problems where the function set requires them to be bushy will be rare.

- The number of solutions grows exponentially with size.
So what?

- Generally random instructions “lose information”. Unless inputs are protected, almost all long programs are constants. Write protecting inputs linear GP like tree GP.

- “Random Trees” a few inputs near root. May be good for Data Mining, where some inputs are more important. Other cases each input is equally important. Need bushy trees. E.g. parity more common in full trees.

- Depth limit promotes near full trees rather than random Size limit promotes random trees

- Density of solutions indication problem difficulty

- No point searching above threshold?

- Predict where threshold is? Ad-hoc or theoretical.
Conclusions

• Size and shape of search space

• Experimental evidence, tree based GP

• Proof linear

• Proof tree (in FoGP book)

• Number of solutions grows exponentially with size

cs.ucl.ac.uk/genetic/gp-code/

ntrees.cc and rand_tree.cc
Santa Fe Ant Trail

- Systematic exploration of program space
- Its size and the number of solutions
- Rugged landscape $\Rightarrow$ hard to hill climb
- Looking for Building Blocks $\Rightarrow$ hard for crossover
- Short solutions $\Rightarrow$ symmetry not being exploited
- Conclusions
Artificial Ant following the Santa Fe Trail

Complex, iteration and side-effects

- 89 food pellets, twisting trail, 32 × 32
- Move, Left, Right
- IfFoodAhead
- Prog2, Prog3
- fitness = food eaten
Santa Fe Trail

Ant starts from centre (hashed square)
Size and Number of Solutions

Cf. slide 18

Santa Fe Trail (note log scales)

- Number of programs
- Number of solutions

Program Size

Number of programs

Number of solutions

W. B. Langdon
### “Effort” to Solve Santa Fe Trail

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort/1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random (len=18)</td>
<td>400</td>
</tr>
<tr>
<td>Random (len=25)</td>
<td>1,200</td>
</tr>
<tr>
<td>Random (len=50)</td>
<td>2,700</td>
</tr>
<tr>
<td>Random (len=500)</td>
<td>4,700</td>
</tr>
<tr>
<td>Ramped-half-and-half</td>
<td>15,000</td>
</tr>
<tr>
<td>GP [Langdon and Poli, 1997b]</td>
<td>450</td>
</tr>
<tr>
<td>Subtree Mutation [Langdon and Poli, 1998a]</td>
<td>426</td>
</tr>
<tr>
<td>Simulated Annealing</td>
<td>748</td>
</tr>
<tr>
<td>Subtree-sized</td>
<td>435</td>
</tr>
<tr>
<td>Hill Climbing</td>
<td>955</td>
</tr>
<tr>
<td>Subtree-sized</td>
<td>1,671</td>
</tr>
<tr>
<td>Strict Hill Climbing</td>
<td>186</td>
</tr>
<tr>
<td>Subtree-sized</td>
<td>738</td>
</tr>
<tr>
<td>Population (data for best)</td>
<td>266</td>
</tr>
<tr>
<td>PDGP</td>
<td>336</td>
</tr>
</tbody>
</table>
Analysing Fitness Landscape

1. Hill climbing landscape

2. GP schema fitness

- Both use point mutation, no change in size or shape
Hill Climbing – Karst Landscape

• Many individuals of intermediate fitness don’t have fitter neighbour

• len=11 solutions, no neighbour fitness $> 36$, most $< 24$

• length $\leq 14$ solutions are (almost) isolated

• sampling $> 14 \Rightarrow$ similar

• neighbours with same fitness rises $\approx 1.5$ length plateau, fitness gives no guidance
Karst Landscape

Solutions

Solutions surrounded by deep moats.

Plateaus – deep ravines
Karst Limestone Landscape

Local optima off in the distance (behind lake).

Plateaus – deep ravines.

Global optima\(^a\) thousands of miles away.

\(^a\)Not shown

* The Geological Survey of Ireland
Looking for Building Blocks

- BB small components of solution with above average fitness

- Length $\leq 13$

- Deception
  - length, longer programs fitter than those of length 18
  - Fittest tree shapes no solutions
  - Fittest schema no solutions
  - Some components (fixed size/shape) of solutions below average fitness
Fitness of Schema Containing Solutions, Length=11

Santa Fe Trail, (Prog3 (= = (Prog3 = = ( = = =))) = =)
Solutions of length 11

$x$ and $y$ can be either Left or Right and the three arguments of the root can be rotated, giving 12 solutions.

Solutions of size 12, 13 and 14 are similar
Conclusions

• Benchmark problem, features typical of real programs?
  Details important. Toroid and time limit give “dumb” programs high score

• Systematic exploration of 889 $10^6$ programs

• Neighbourhood analysis, hill climbing search difficult
  Schema analysis, deceptive, no building blocks found. (Problem too simple?)
  GA hard $\Rightarrow$ randomisation techniques with size $\approx 18$

• Ramped $\frac{1}{2}$-and-$\frac{1}{2}$ need not be a good form of random search

Tree counting C++ code and 3916 ants available via cs.ucl.ac.uk cf. slide 41
Max

- What is the MAX Problem
  - Find tree with max value within limited size
  - Function set \{*,+\}, terminal set \{0.5\}
  - Depth limit \(D\)
  - Max value \(4^{2^{D-3}}\), \(2^{D-3}\) optimal trees

- Exponentially hard (but GP \(\ll\) random) because
  - Initial generations deceptive. Price's Theorem
  - Later difficult because a) depth limit and strong selection confine crossover so only improvements are accepted b) little crossover activity near root
  - Hill climbing model \(\Rightarrow\) time \(O(2^{2D})\). Cf. 58 and 59
  - Search space \(O\left(2^{2^{D+1}}\right)\)
**MAX Problem Parameters**

<table>
<thead>
<tr>
<th>Objective:</th>
<th>Find a program that returns the largest value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitives:</td>
<td>$+, \times, 0.5$</td>
</tr>
<tr>
<td>Max depth</td>
<td>3 ... 8 (NB root node is depth 0)</td>
</tr>
<tr>
<td>Init depth</td>
<td>5 or Max depth</td>
</tr>
<tr>
<td>Fitness:</td>
<td>Value of tree</td>
</tr>
<tr>
<td>Selection:</td>
<td>Tournament group size of 2 to 8, generational plus elitism.</td>
</tr>
<tr>
<td>Parameters:</td>
<td>Pop $= 200$, $G = 500$, 99.5% crossover, no mutation. Crossover points selected uniformly between nodes.</td>
</tr>
</tbody>
</table>
Predicting which MAX runs will fail

Covariance with fitness and frequency for $\times$ in the second level of the tree.

Means of successful and unsuccessful runs. (D = 5).

Initialy $\dagger$ gives better score than $\times$, so selection removes $\times$. This is especially important near root.
How Many Steps to Solution

Mean Number of improving steps before optimal solution

Time for crossover to replace $+$ by $\times$ near root. Model \[2\]. p55 $O(2^{2D})$ is reasonable.

Improvements made before optimal solution in successful runs.

Number of steps $\approx$ functions in tree

W. B. Langdon
Mean number of generations to solve
(succesful runs)
MAX Problem Conclusions

- Can understand how GP solves MAX problem by analysis

- MAX is a hard problem, but GP exponentially better than random search

- Hard because initially deceptive. Fitness selection drives population in wrong direction.

  Using Price’s covariance theorem in first few generations can predict which runs will fail many generations later.

- Hard because only crossovers which make immediate improvements are accepted into the population. Many improvements are needed from random start. C.f. GA long path problems.
Bloat = Convergence

- Convergence in GP
- Evolution of Size and Shape
- No worse than parabolic growth prediction (binary trees)
Convergence

Genetic Algorithms

1. Fitness Selection

2. Crossover and Mutation

3. Converges after many generation

Mutation and selection balanced

Genetic Programming

1. Fitness Selection

2. Crossover and Mutation

3. After many generations

Phenotype converged

Genotype continues to change
Fitness Selection Acts on Phenotype

GA 1-to-1 mapping genotype-to-phenotype

GP 1-to-many mapping

Spread of phenotypes (unequally) reduced

1:1 mapping, identical reduction in genotypes

Complex mapping, uneven reduction in genotypes
Crossover and Mutation Spread the Genotype

fixed mapping, spread of phenotype

Most genotype slightly changed, map to (nearly) original ellipse. Some more diverse, new (small) ellipses, map to new phenotype ellipses.
Genetic Programming Phenotype Convergence

Phenotype converged

Genotype continues to change
Some GP genotypes resist crossover and mutation more and “breed true”. I.e. more of their offspring have the same phenotype. If it is fit, these genotypes quickly dominates.

Population convergences to contain just the descendents of one phenotype-genotype mapping (a bit like GA).

Genotype cluster does not stabilise but continues to evolve from a single point. The population’s ancestor, i.e. the individual program where most of its genetic material came from.

Since each fit child’s genotype tends to be bigger than its parents there is a progressive increase in size, which we know as bloat.
What is Bloat

- Tendency for programs to increase in size without a corresponding increase in fitness

- In the absence of counter measures always(?) happens

- Trees and linear

- Often size decrease in first 1..3 generations

- Steady increase (max, average, standard deviation) after \( \approx 10 \) generations

- No limit to increase??
Experimental Evidence

Size increase in most generations
No change in best fitness in most generations

Smooth “average” curves conceal wide variation between runs. Also wide variation within each population [Langdon et al., 1999]
Convergence of Phenotype

Sextic Polynomial, Phenotype of Best of Generation, Run 100

Plot of output of evolved program from a range of inputs (excludes training points)

Note similarity of behaviour (i.e. phenotype) of nearby generations

W. B. Langdon
Fitness Needed for Bloat

Expected change in frequency of a gene $\Delta q$ in the population from one generation to the next $= \text{covariance of the gene's frequency in the original population with the number of offspring } z \text{ produced by individuals in that population, divided by the average number of children } \bar{z}$

$$\Delta q = \frac{\text{Cov}(z,q)}{\bar{z}}$$  \cite{Price, 1970}

Holds if genetic operations are random with respect to gene.

Applies to program size in GP with crossover and mutation operators which have no size bias \cite{Langdon et al., 1999}

With tournaments $t$, fitness is given by ranking $r$ in the population (of size $p$). If $p \gg 1$ \cite{Price, 1970} can be approximated:

$$E \Delta \text{size} \approx \frac{t}{\bar{z}} \text{Cov}((r/p)^{t-1}, \text{size})$$ \cite{Langdon and Poli, 1998a}
Covariance of fitness and program size gives change in mean size from one generation to the next

Positive increase (e.g. bloat) requires positive covariance, i.e. fitness variation in current generation
Linear Increase in Depth (Standard Crossover)

Note on average linear growth in tree depth during bloat
($\approx 1$ level per generation)

W. B. Langdon
Evolution of Shape

Mean of 50 GP Sextic polynomial regression runs. Plotted on top of Distribution of all Program Shapes

Note movement from bushy (full) trees towards random trees

Note only bushy half of search space is used

W. B. Langdon
Sub-quadratic Growth in Binary Trees

- Predicted $\lim_{t \to \infty}$ program size $= O(t^2)$

- Measured bloat $O(t^{1.2-1.5})$ for $t \leq 50$ generations

- Test $O(t^2)$ for 600 generations, size $10^6$

- Theory

- Experiments

- Conclusions
Theory

• If program size $\gg$ problem and fitness level dependent threshold, distribution of fitness does not change with length

• Above threshold, number of programs with fitness $f$ of size $l$ is distributed $\propto$ total number of programs of size $l$

• Total number of programs grows exponentially with size

• Most programs are near mean depth $= 2\sqrt{\pi}$ (internal nodes) (ignoring terms $O(N^{1/4})$ [Flajolet and Oldzyko, 1982], cf. slide 73)
Rate of Bloat

- In a variety of problems linear increase in mean depth, cf. slide 72 and [Daida et al., 2003]

\[ \Delta \text{depth} = 0.5 \ldots 2.2 \text{ per generation} \]

Variable between problems and individual runs

- If population remains near ridge, size can be predicted from depth

  - If \( \lim_{t \to \infty} \text{depth} \approx 2\sqrt{\pi \left\lfloor \text{size}/2 \right\rfloor} \)

    \[ \lim_{t \to \infty} \text{size} = O(\text{depth}^2) = O(\text{gens}^2) \]

  - Fitting a power law to ridge (50–500) yields

    \[ \text{size} = O(\text{gens}^{1.3}) \]
Experiments

- Hundreds of generations, size = million on rapidly bloating populations
  - continuous: symbolic regression quartic polynomial \[ \text{[Koza, 1992]} \).
  - Discrete: 6-multiplexer (binary function set)
    64 fitness cases in parallel, submachine code GP \[ \text{[Poli and Langdon, 1999]} \).
# Quartic Symbolic Regression

**Objective:** Find a program that produces the given value of the quartic polynomial \( x^2(x + 1)(x - 1) = x^4 - x^2 \) as its output when given the value of the one independent variable, \( x \), as input.

**Terminal set:** \( x \) and 250 floating point constants chosen at random from 2001 numbers between -1.000 and +1.000.

**Functions set:** \( + - \times \% \) (protected division)

**Fitness cases:** 10 random values of \( x \) from the range -1...1.

**Fitness:** The mean, over the 10 fitness cases, of the absolute value of the difference between the value returned by the program and \( x^4 - x^2 \).

**Hits:** The number of fitness cases (between 0 and 10) for which the error is less than 0.01.

**Selection:** Tournament group size of 7, non-elitist, generational.

**Wrapper:** none

**Pop Size:** 50

**Max program:** \( 10^6 \) program nodes

**Initial pop:** Created using “ramped half-and-half” with depths between 8 and 5 (No uniqueness requirement).

**Parameters:** 90% one child crossover, no mutation. 90% of crossover points selected at functions, remaining 10% selected uniformly between all nodes.

**Termination:** Maximum number of generations 600 or maximum size limit exceeded.
# Binary 6-Multiplexor

| Objective: | Find a Boolean function whose output is the same as the Boolean 6 multiplexor function |
| Terminal set: | D0 D1 D2 D3 A0 A1 |
| Functions set: | AND OR NAND NOR |
| Fitness cases: | All the $2^6$ combinations of the 6 Boolean arguments |
| Fitness: | number of correct answers |
| Selection: | Tournament group size of 7, non-elitist, generational |
| Pop size: | 500 |
| Max program: | $10^6$ program nodes |
| Initial pop: | Ramped half-and-half max depth between 2 and 6 |
| Parameters: | 90% one child crossover, no mutation. 90% of crossover points selected at functions, remaining 10% selected uniformly between all nodes. |
| Termination: | Maximum number of generations $G = 50$ or exceeding size limit |
Results: Continuous

- 9 of 10 bloat (1 trapped at local optima in generation 7)
  
  At least 400 generations
  
  3 runs reach 1,000,000 limit before 600 generations
  
  In all runs most new generations do not find better fitness
  I.e. changes in size and shape are due to bloat

- Each population close to the ridge and moves up it,

- Depth varies widely between runs. However mean of all ten
  runs increases $\approx 2.4$ levels per generation

- The size power law varies widely. On average starts near
  1.0 (generations 12–50) and steadily rises to 1.9 (12–400).
Evolution of Tree Shape: Quartic

Note log scales
Mean Tree Depth: Quartic symbolic regression

Program Depth vs. Number of Programs Created
Evolution Power Law Coefficient: Quartic

Nine bloating runs. Error bars show standard error

W. B. Langdon
Results: Boolean Benchmark

- In all runs most new generations do not find better programs
  \[ \Rightarrow \text{changes in size and shape are due to bloat} \]

- On average each population evolves to lie close to the ridge and moves along it, slide 85

- Mean population depth varies between runs but the mean of all ten runs increases at about 0.6 levels per generation, slide 86

- Power law coefficient of programs v. generations varies widely between runs.
  
  On average starts at 1.25 and rises. By the end of the runs (generations 12–600) it reaches 1.5.
Evolution of tree Shape: binary 6-multiplexor

Note log scales

W. B. Langdon
Evolution of tree Depth: binary 6-multiplexor

Mean Program Depth

Number of Programs Created

W. B. Langdon
Evolution of power law coefficient:

binary 6-multiplexor

Error bars indicate standard error
Why Quadratic Limit is Not Reached

• Standard crossover may cease to be disruptive when the programs become very large

• In 6-multiplexor there are whole generations when every program in the population has the same fitness

• Therefore the selection pressure driving bloat falls as the populations grow in length. Cf. slide 70 and [McPhee and Poli, 2001].

Which is why the quadratic limit is not reached
Evolution of Selection: binary 6-multiplexor

Fraction of tournaments where all 7 candidates have the same fitness (smoothed)
Discussion

• Ridge divides search space in half. In fact the region searched is much less than 50% \cite{Daida et al., 2003}.

• Can predict when program size or depth restrictions will be effective

In practise limits are quickly reached

but may be beneficial in some problems

Even Parity v. Santa Fe artificial ant

• Other genetic operators and non-tree GP have different bloat behaviour

• Benchmarks here simple but subtree crossover ineffective on programs of $10^6$

  Many smaller trees? Different genetic operators?
Bloat Conclusions

- Bloat explained as evolution towards popular tree shapes
  Subtree crossover leads to growing $\approx 1$ level per generation

- Predict average evolution of size, depth and shape
  Continuous $\lim_{g \to \infty} \text{mean size} = O(\text{generations}^{2.0})$
  Discrete mean size $\leq O(\text{generations}^{2.0})$

  (Wide variation in population and between runs)

  New type of GP fitness convergence in discrete case

  Memory $O(\text{gens}^{1.2-2.0})$ or $\leq O(\text{gens})$ (DAGs)
  Run time $O(\text{gens}^{2.2-3.0})$ or $= O(\text{gens}^{2.0})$ (DAG caches)

- Understanding bloat provides insights into GP dynamics
  Understand GP biases $\rightsquigarrow$ new operators, better biases

- GP theory developed, tested, Works! (in part)
Conclusions

- *Foundations of Genetic Programming* covers many other topics

- Fitness Landscapes metaphor

- Fitness search space convergence

- Santa Fe Ant. GA hard, no regularities?

- Max. Qualitative prediction

- Bloat. Manifestation of GP convergence (of phenotype) Sub-quadratic growth predicted and tested


[Hooper and Flann, 1996] Dale Hooper and Nicholas S. Flann. Improving the accuracy and robustness of genetic programming through expression simplification. In John R. Koza, David E. Goldberg,


[Langdon, 2003c] W. B. Langdon. How many good programs are there? How long are they? In Kenneth A. De Jong, Riccardo Poli, and Jonathan E. Rowe, editors, Foundations of Genetic


