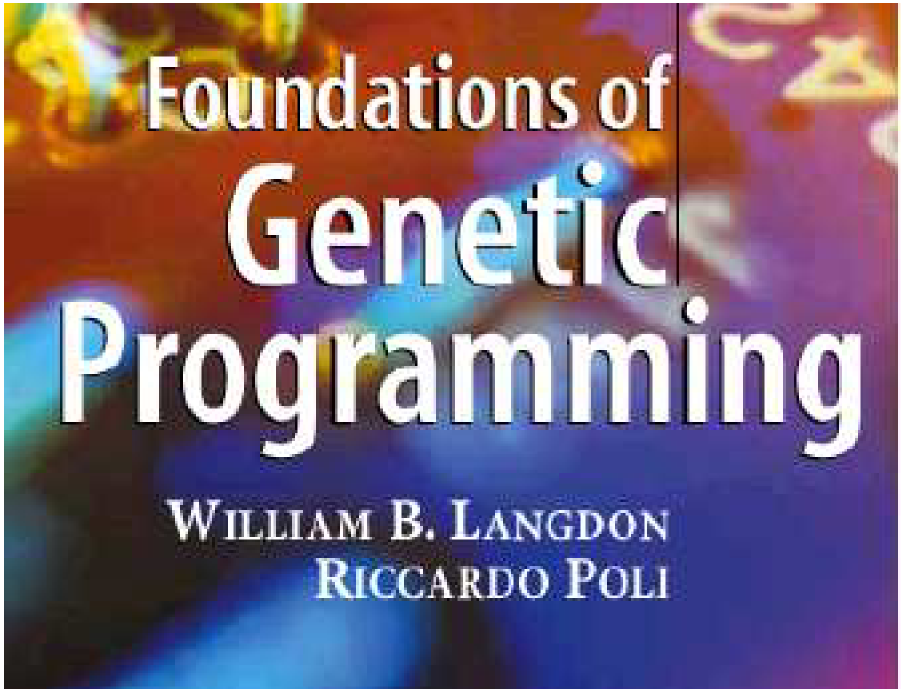


W. B. Langdon
Computer Science,
University College, London

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*

  GECCO 2001-2003 given in two parts with Riccardo Poli. Slides for 2003 available via `ftp://cs.ucl.ac.uk/genetic/papers/fogp_slides/`

- **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 $\rightsquigarrow$ Valleys and seas are good points.

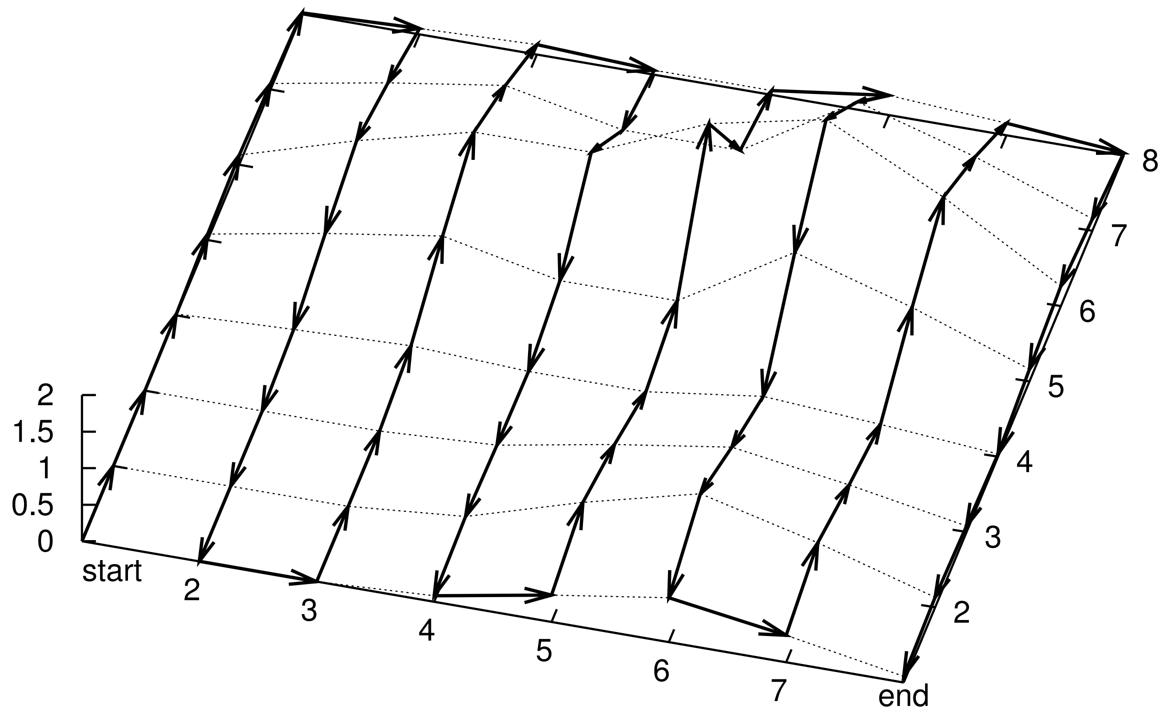
- Enumeration, exhaustive exploration

- 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

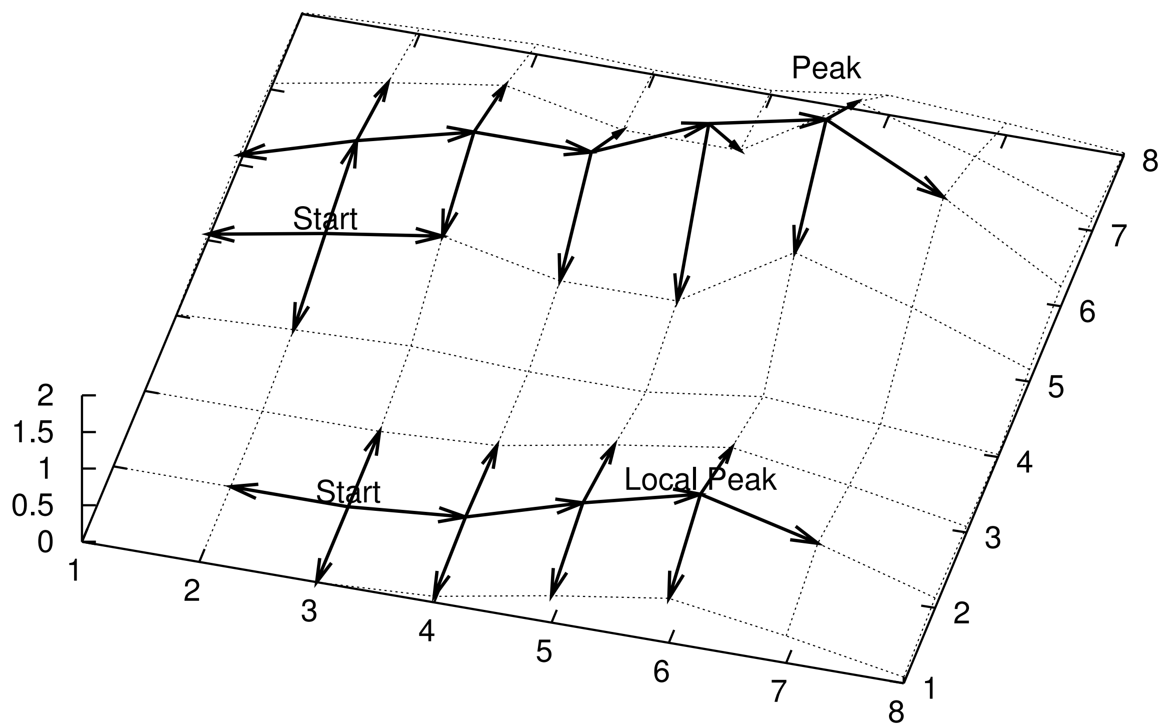
Enumeration



Explore whole search space is systematic fashion.

(Monte Carlo: sample search space at random)

Hill Climbing



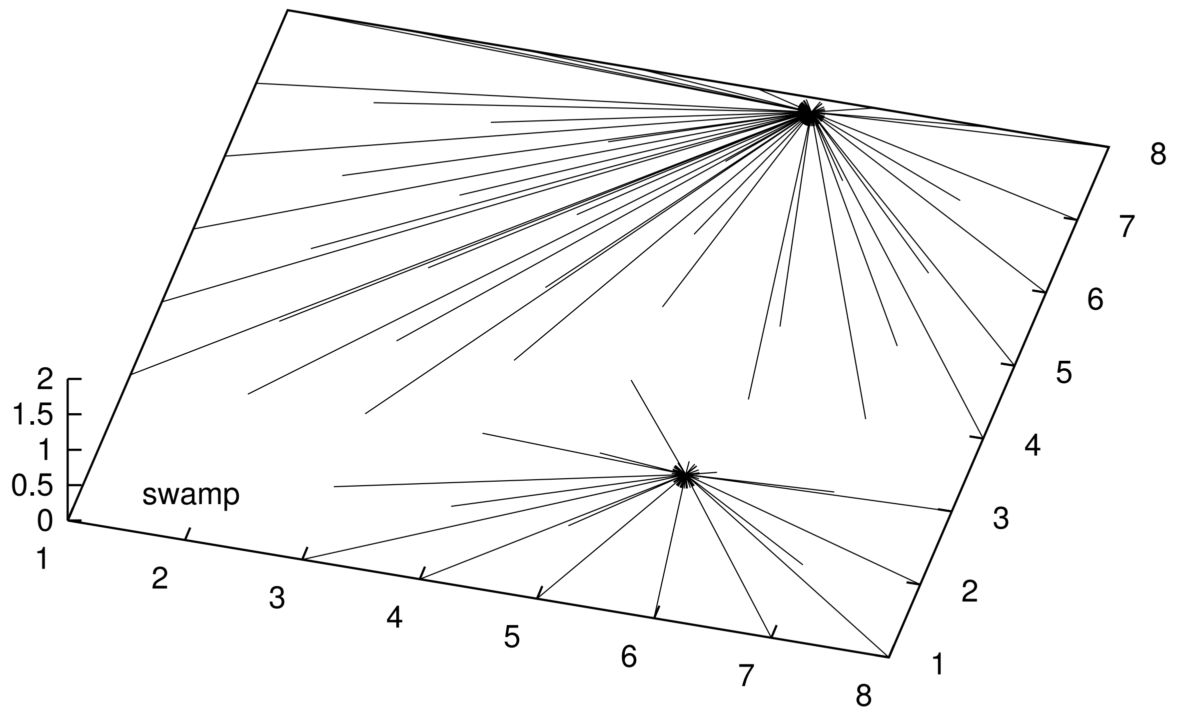
Choose start point at random

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.)

Basins of Attraction



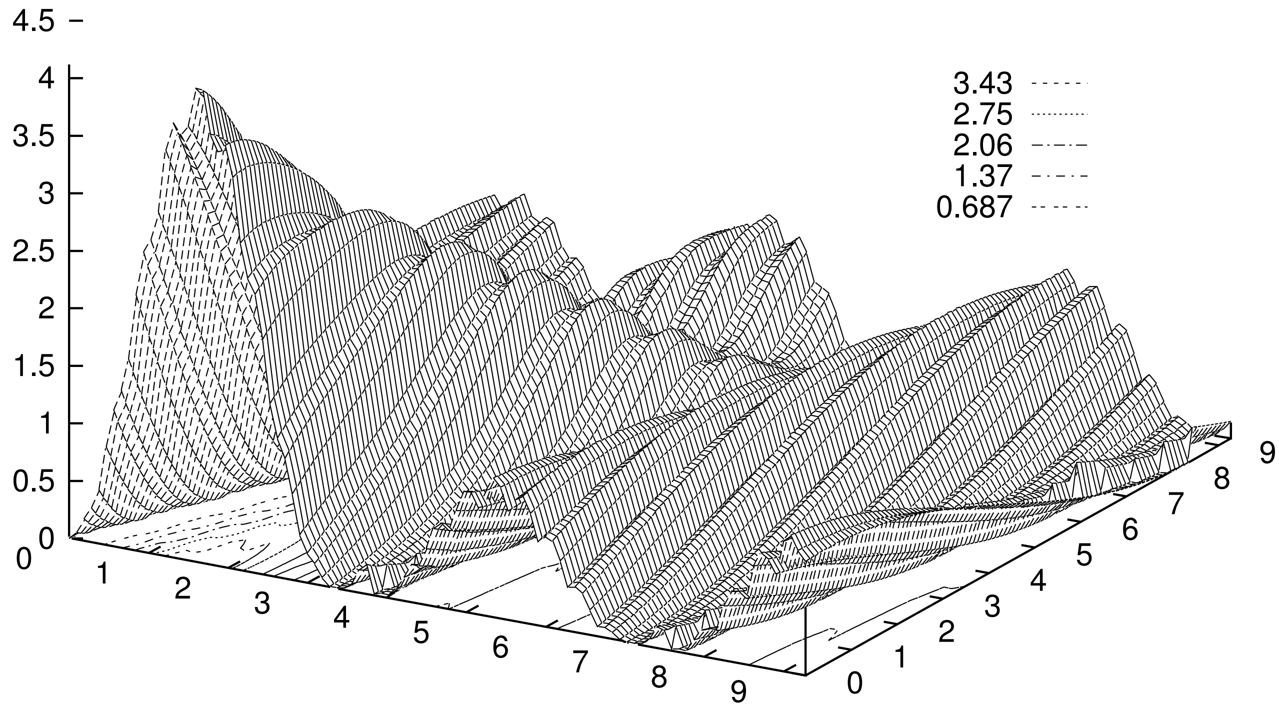
Two major basins of attraction.

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).

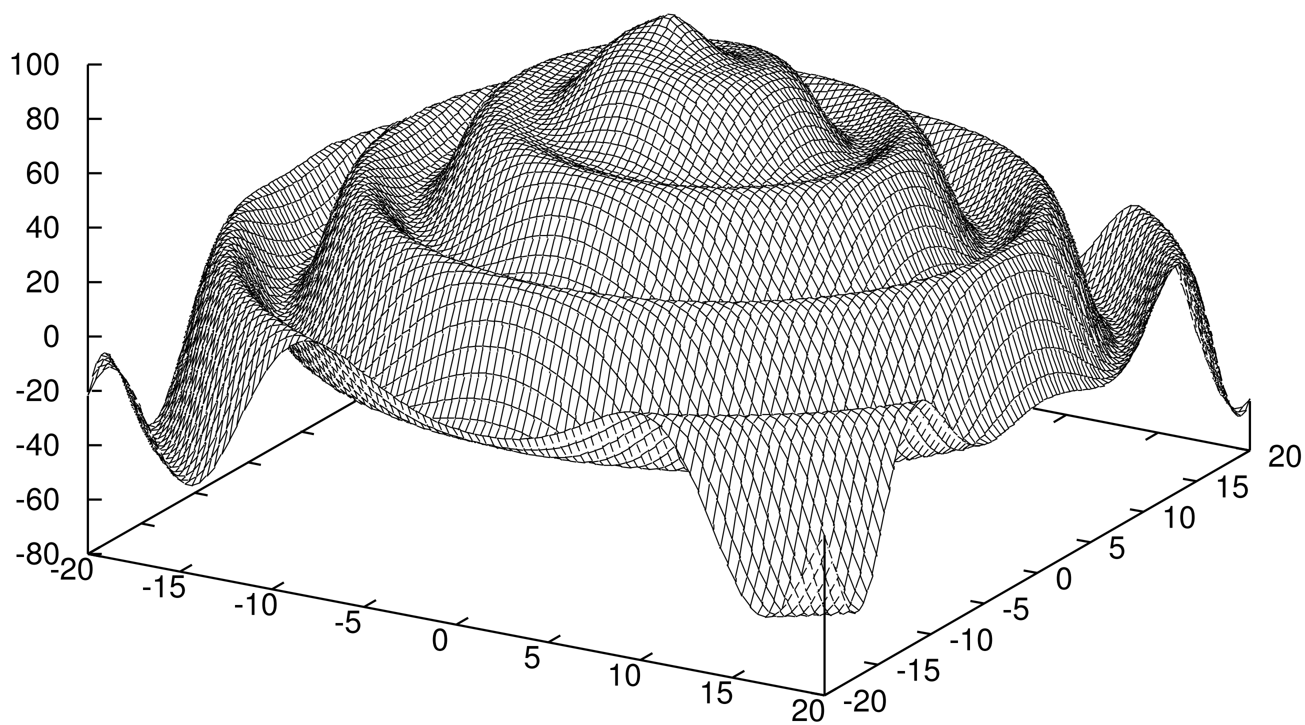


Rugged landscapes with many hills

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

Long Paths



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

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.

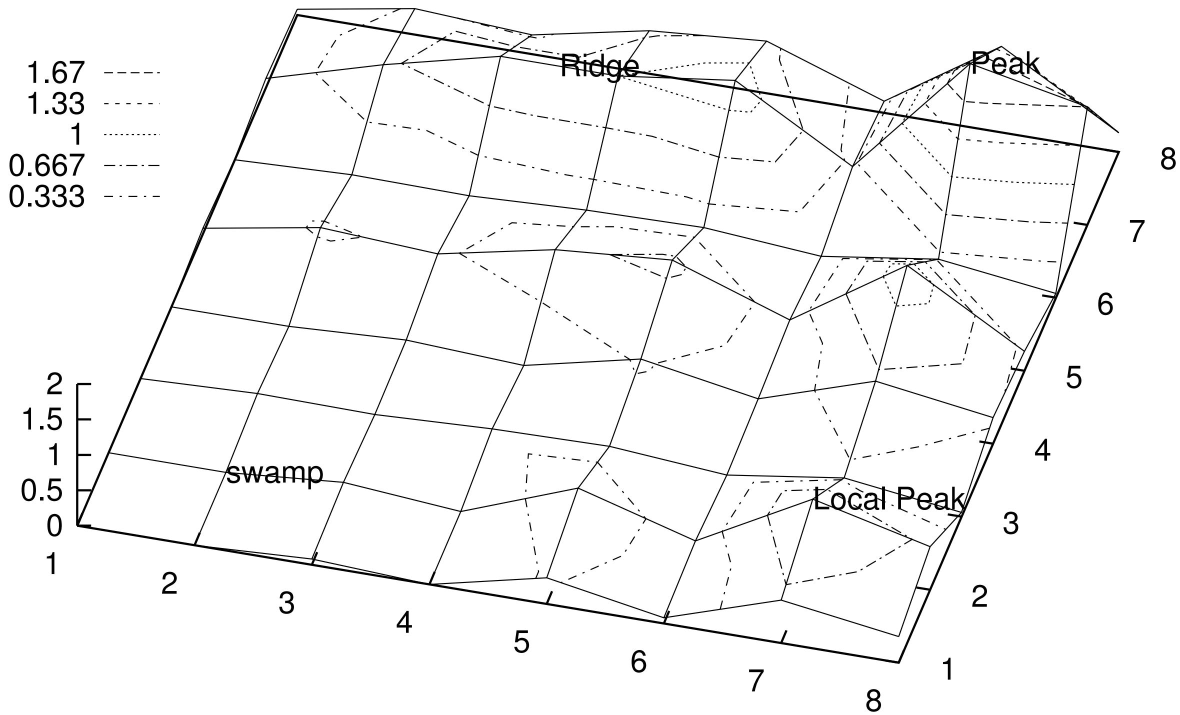
On detecting a local peak, start exploring again from a new randomly chosen start point.

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.

Change of representation



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

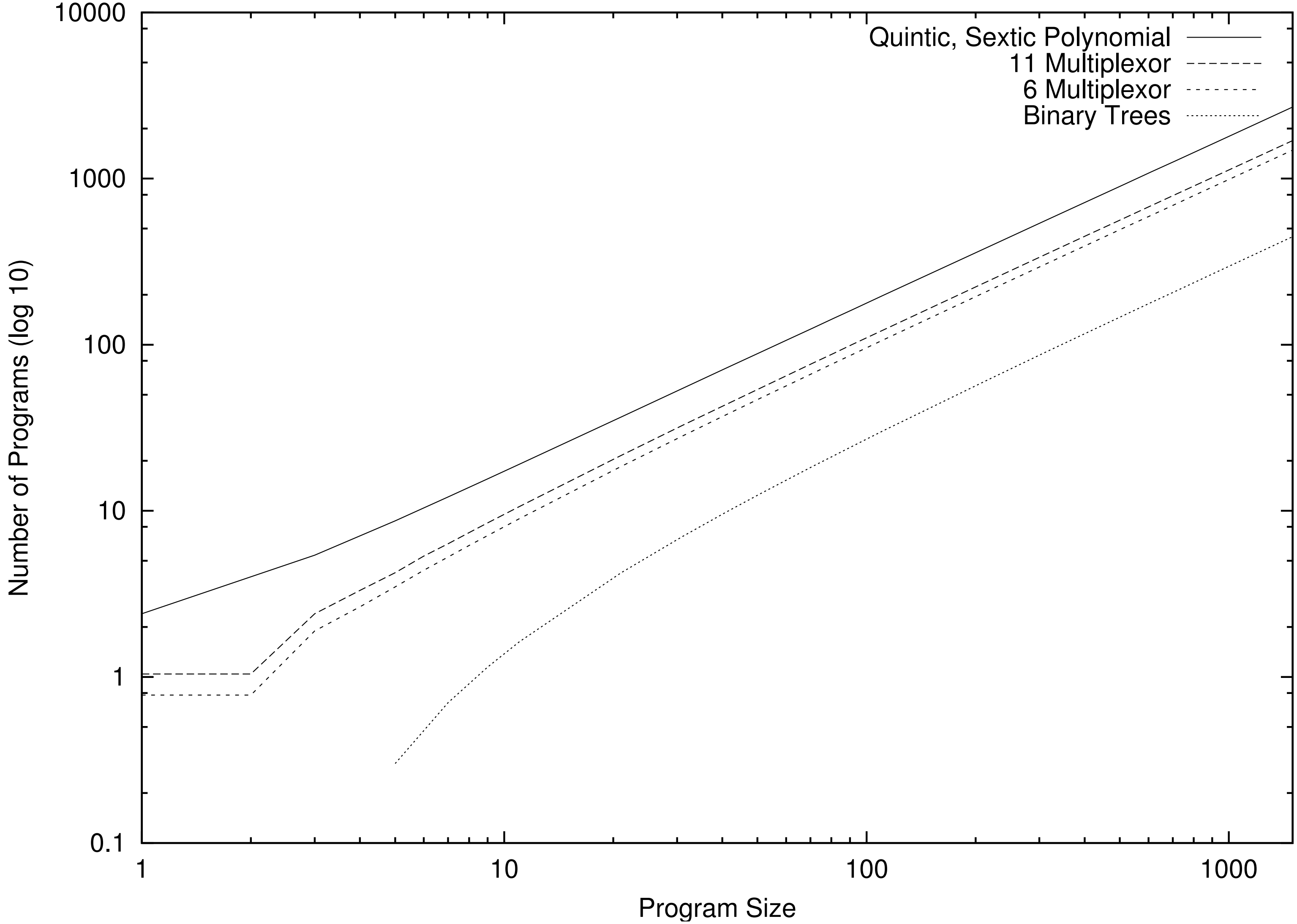
  Good for two dimensions and small mutations.

  Hard to visualise discrete search spaces or crossover
- Fixed landscape assumption, what of time varying?
  "Effective fitness" landscape changes as population moves across it

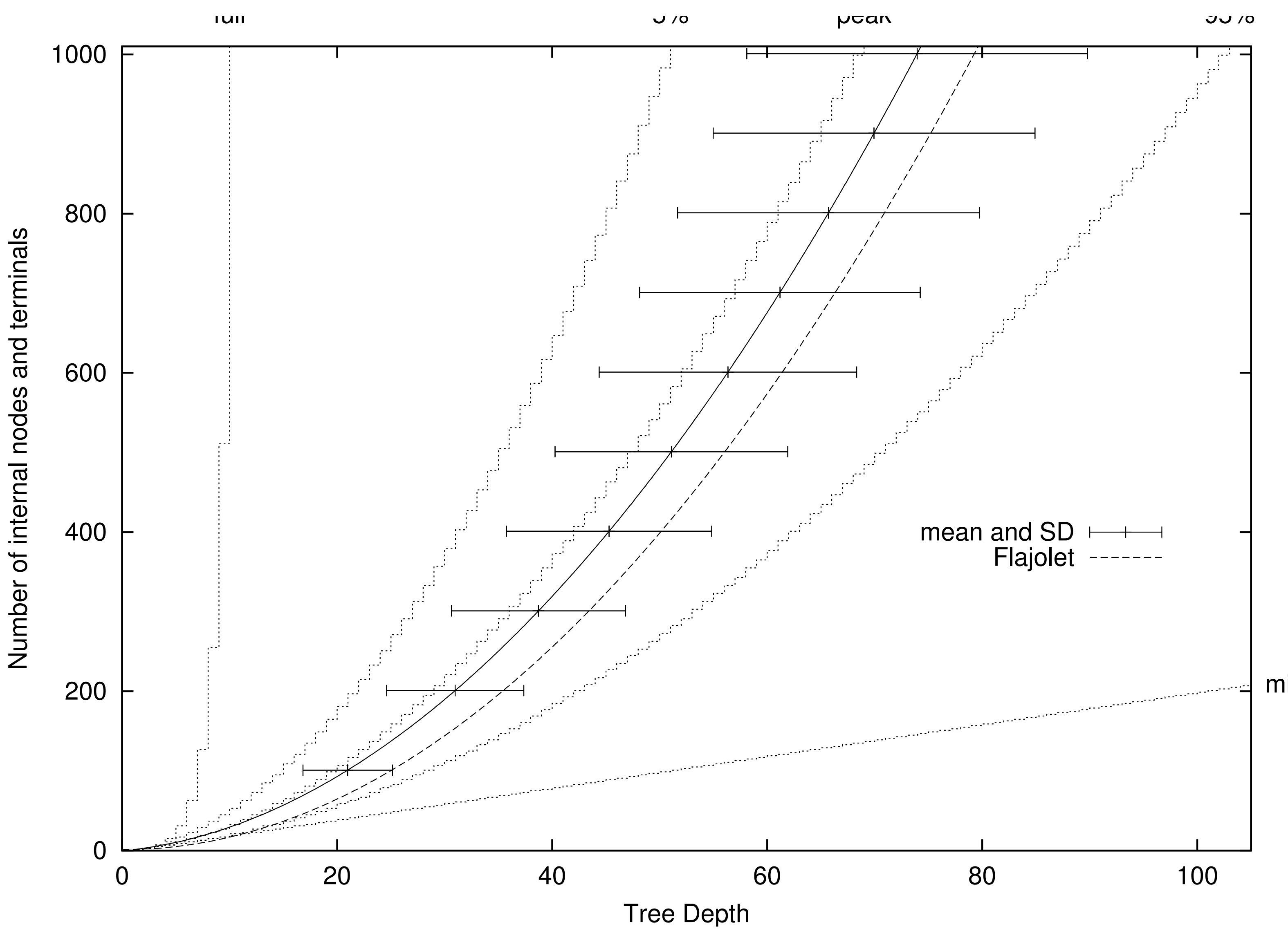
Scaling of Program Fitness Spaces

- 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?

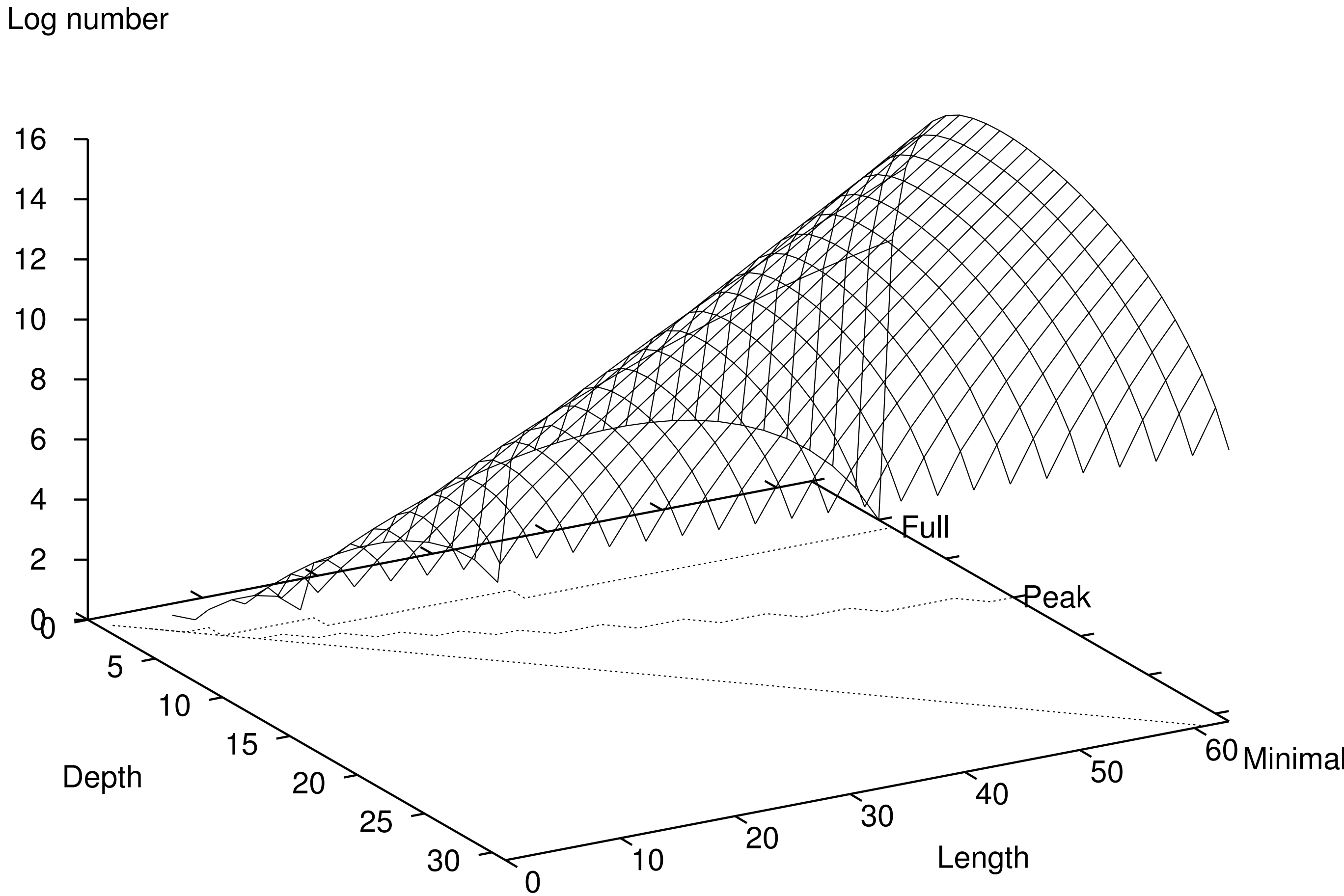
Number of programs v. size, various problems



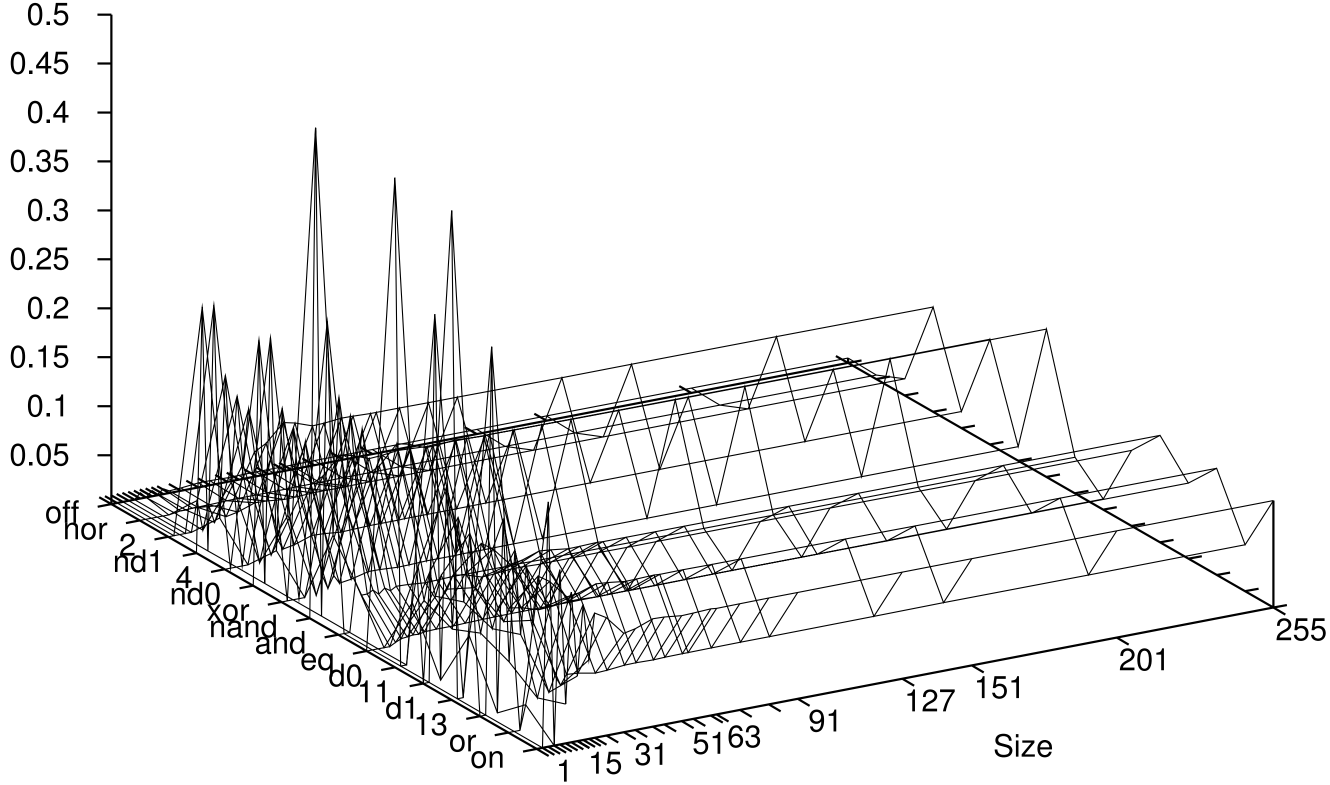
Distribution of Binary Trees by size and height



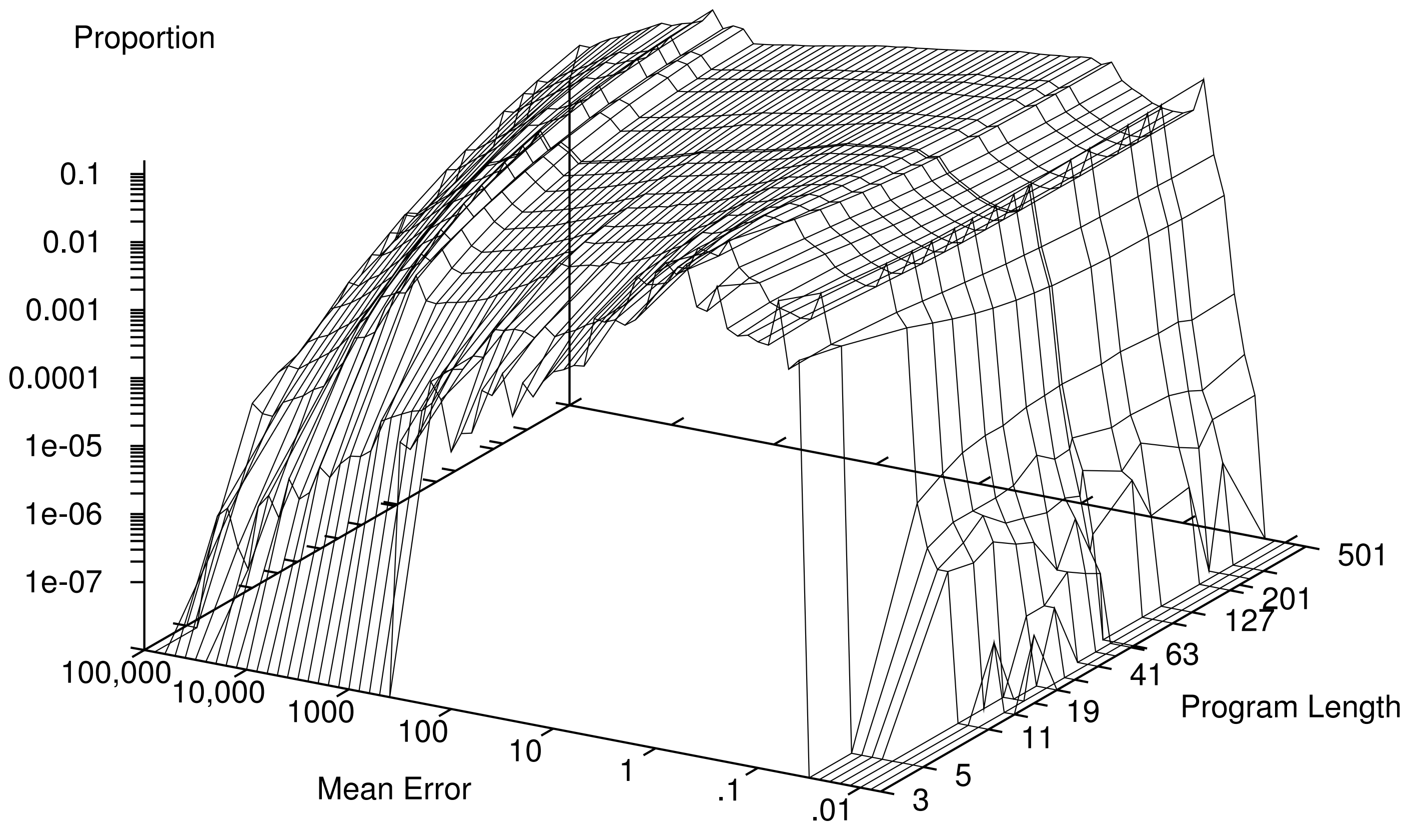
Distribution of Binary Trees by size and height



Proportion of NAND trees:
2 input logic function

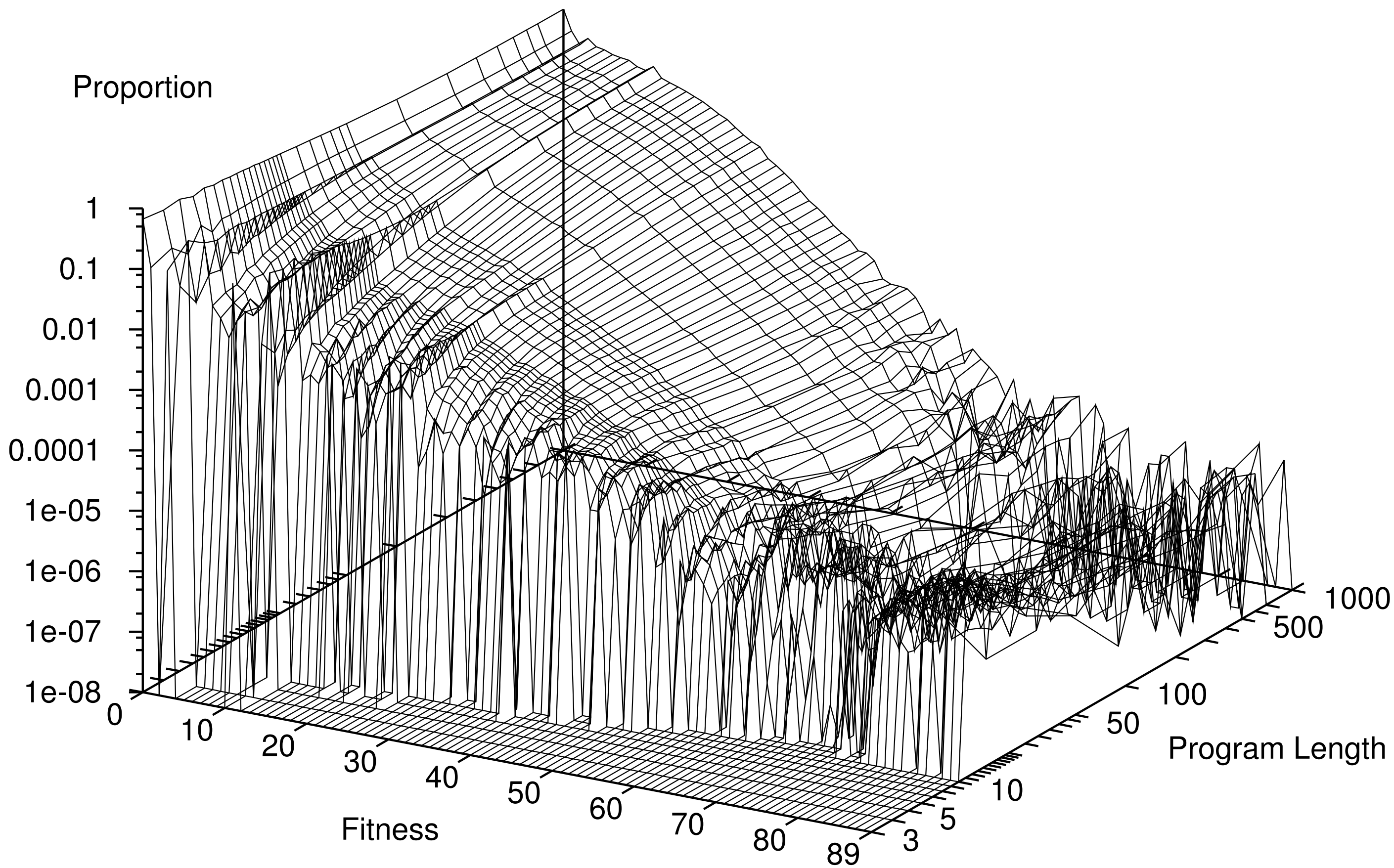


Fitness Sextic Polynomial

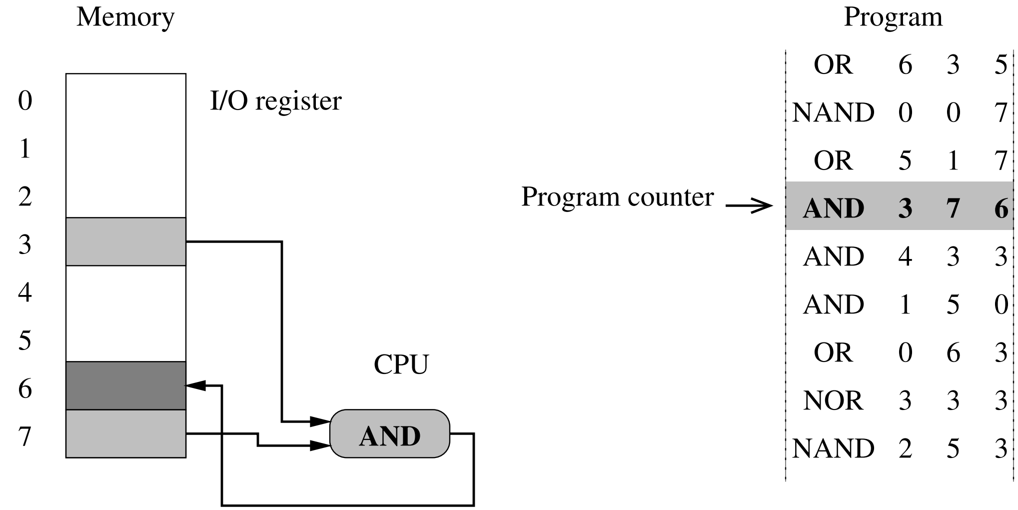


(constants and input equally sampled)

Artificial Ant



Linear Model of Computer



- Input and output registers part of memory.
- Memory initialy 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 $\leadsto$ 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 $M =$ probability of transition from state i 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 $\rightleftharpoons$ 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

- $|\text{probability vector}| = 2^n$ $\qquad v = \underbrace{0, 0, \ldots, 1, 0, \ldots, 0}_{2^n \text{ elements}}$

- At any time t in one state $i \Rightarrow v_i^t = 1$ and $v_{\not i}^t = 0$

- 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 $u =$ 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 initialy loaded with an input
- Program's answer is given by R_0

An Illustrative Example: Instruction Set

- There are $2^2 = 4$ states ($R_1 R_O = 00, 01, 10, 11$)

- There are eight instructions

  Eight transformation (4×4) matrices

$R_0 \leftarrow$ AND
$$\begin{matrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$R_1 \leftarrow$ AND
$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$R_0 \leftarrow$ NAND
$$\begin{matrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix}$$

$R_1 \leftarrow$ NAND
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

$R_0 \leftarrow$ OR
$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$R_1 \leftarrow$ OR
$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$R_0 \leftarrow$ NOR
$$\begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{matrix}$$

$R_1 \leftarrow$ NOR
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

- Example $R_0 \leftarrow$ AND

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

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

  $R_1 = 1, R_0 = 0$

  I.e. 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.

$$M = 1/8 \begin{pmatrix} 4 & 2 & 2 & 0 \\ 2 & 4 & 0 & 2 \\ 2 & 0 & 4 & 2 \\ 0 & 2 & 2 & 4 \end{pmatrix}$$

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 λ and corresponding eigenvectors E of M are

$$
\begin{array}{llrrrrl}
\lambda_{00}{=}1/2 & (& 0 & -1 & 1 & 0 &) \\
\lambda_{01}{=}1/2 & (& -1 & 0 & 0 & 1 &) \\
\lambda_{10}{=}1 & (& 1 & 1 & 1 & 1 &) \\
\lambda_{11}{=}0 & (& 1 & -1 & -1 & 1 &)
\end{array}
$$

  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, λ_2
- The smaller λ_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. ($N = 1$ so $2^{N2^N} = 4$ possible functions).

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

Two outputs (0 and 1) both equally likely.

$$\begin{array}{cccc} \text{CLEAR} & \text{NOP} & \text{TOGGLE} & \text{SET} \\ \begin{matrix}1&0&0&0\\1&0&0&0\\1&0&0&0\\1&0&0&0\end{matrix} & \begin{matrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{matrix} & \begin{matrix}0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0\end{matrix} & \begin{matrix}0&0&0&1\\0&0&0&1\\0&0&0&1\\0&0&0&1\end{matrix} \end{array} \quad \begin{array}{c} \text{M} \\ 1/4\begin{pmatrix}2&0&0&2\\1&1&1&1\\1&1&1&1\\2&0&0&2\end{pmatrix}\end{array}$$

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 [Langdon, 2002; Langdon, 2003a].

2. Bit flip. Flip bit$_i$ and NOP. Reversible but not universal [Langdon, 2002; Langdon, 2003a].

3. Any non reversible . [Langdon, 2002a; 2002b; 2003a].

4. Any reversible [Langdon, 2003b].

5. CCNOT (Toffoli gate). Reversible and universal [Langdon, 2003b].

6. The "average" computer [Langdon, 2002a; 2002b; 2003a].

7. AND, NAND, OR, NOR. Not reversible but universal [Langdon, 2002a; 2002b; 2003a].

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.

$\frac{1}{4}(N+1)(\log(m)+4)$, $l \leq (15 + 2.3\ m)/\log I$, $l \leq \frac{1}{2}N(\log(m)+4)$.

Non Reversible Programs – Limiting Fitness Distribution is Zero

Linear systems, where the inputs are not write protected, on average loose 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.

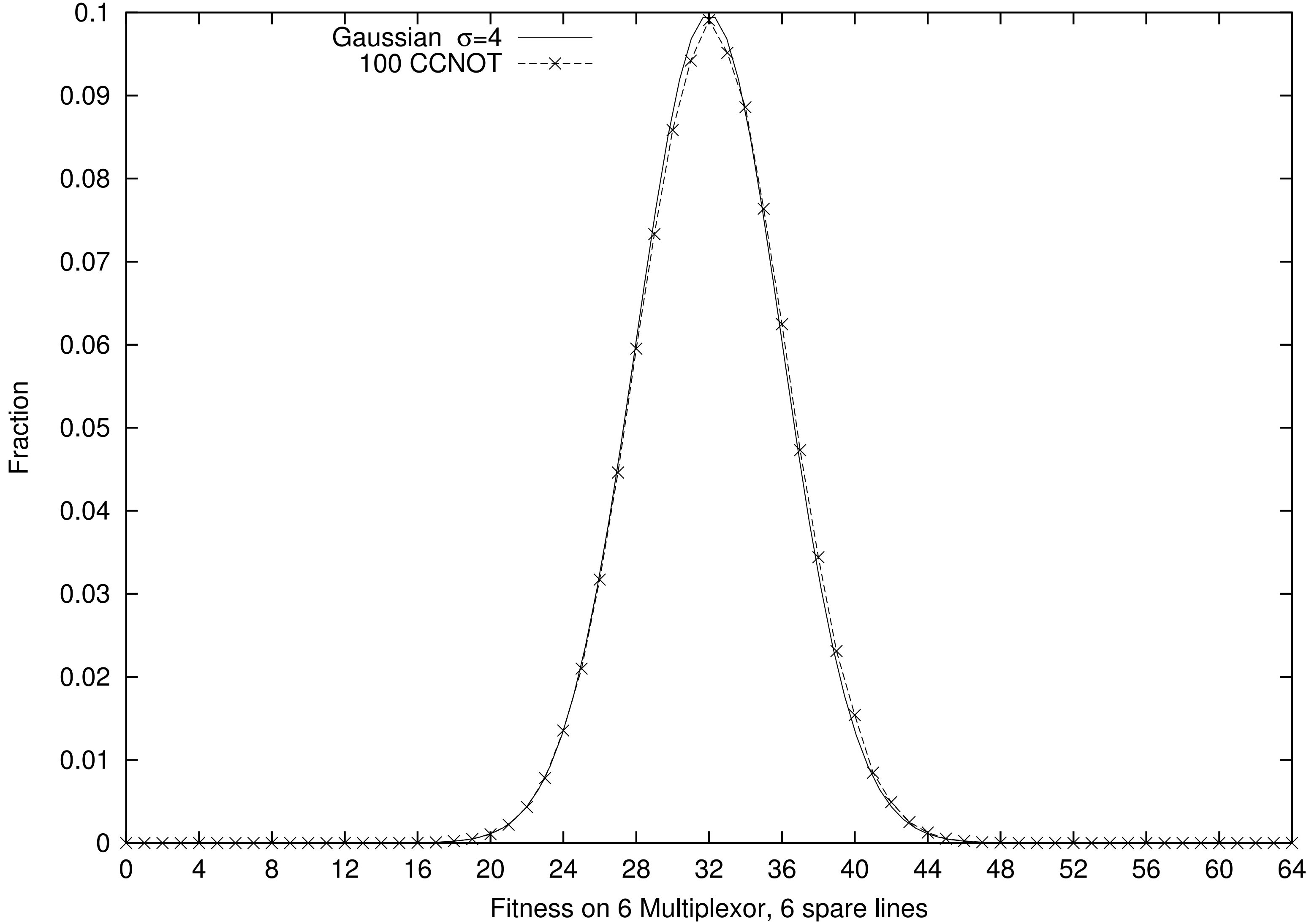
With a Hamming distance fitness function, fitness follows a Normal (Gaussian) distribution.

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 the their fitnesses is Normally distributed [Langdon, 2003b].

Distribution of Reversible Program



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

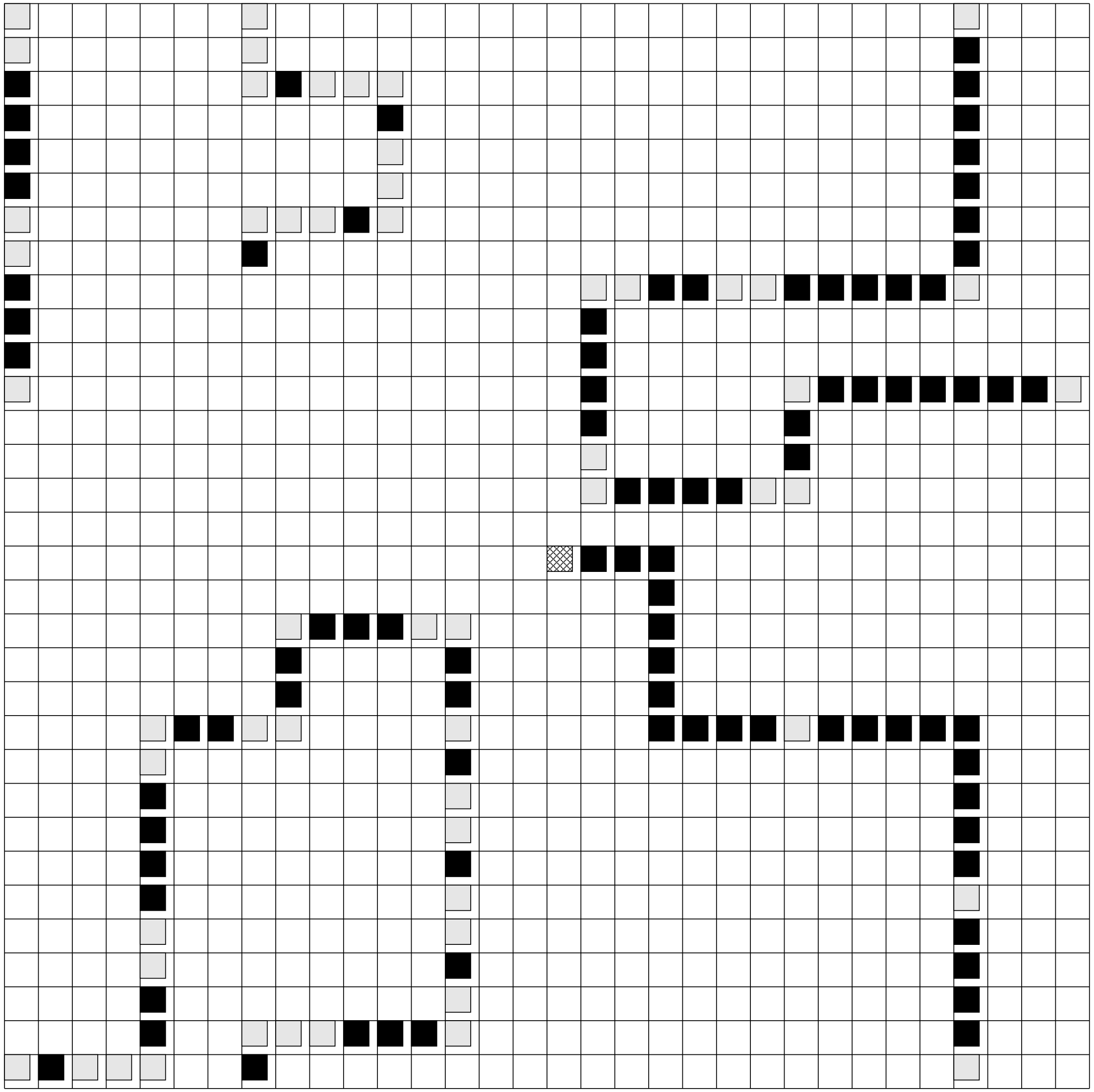
- Systematic exploration of program space
- Its size and the number of solutions
- Rugged landscape $\Rightarrow$ *hard to hill climb*
- Looking for Building Blocks $\Rightarrow$ *hard for crossover*
- Short solutions $\Rightarrow$ *symmetry not being exploited*
- Conclusions

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

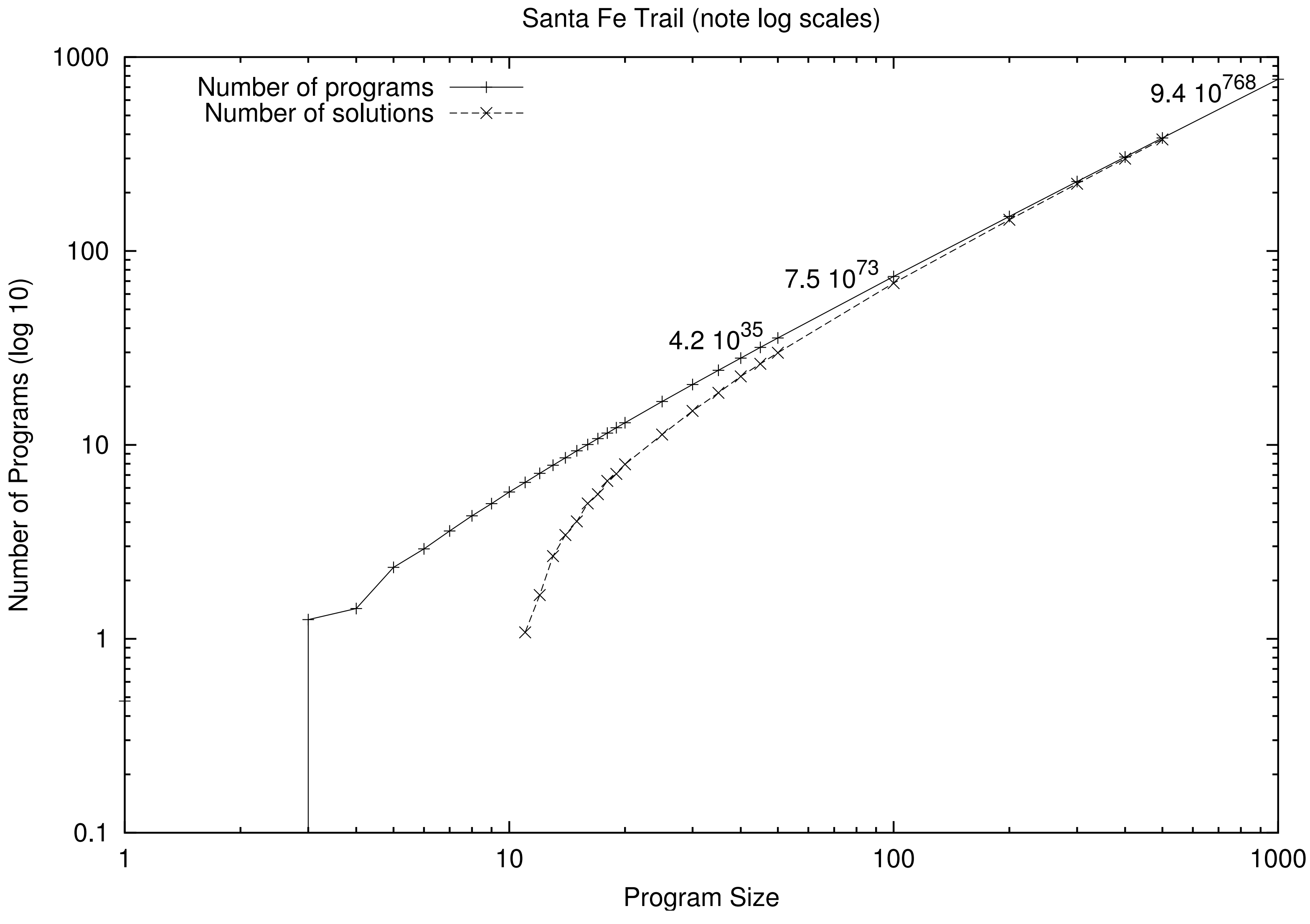
Santa Fe Trail



Ant starts from centre (hashed square)

Size and Number of Solutions

Cf. slide 18



“Effort” to Solve Santa Fe Trail

Method		Effort/1000
Random (len=18)		400
Random (len=25)		1,200
Random (len=50)		2,700
Random (len=500)		4,700
Ramped-half-and-half		15,000
Koza GP	[Koza, 1992, page 202]	450
Size limited, EP	[Chellapilla, 1997]	136
GP	[Langdon and Poli, 1997b]	450
Subtree Mutation	[Langdon and Poli, 1998a]	426
Simulated Annealing	50%–150%	748
	Subtree-sized	435
Hill Climbing	50%–150%	955
	Subtree-sized	1,671
Strict Hill Climbing	50%–150%	186
	Subtree-sized	738
Population (data for best)	50%–150%	266
	Subtree-sized [Langdon, 1998a]	390
PDGP		336

Analysing Fitness Landscape

1. Hill climbing landscape

2. GP schema fitness

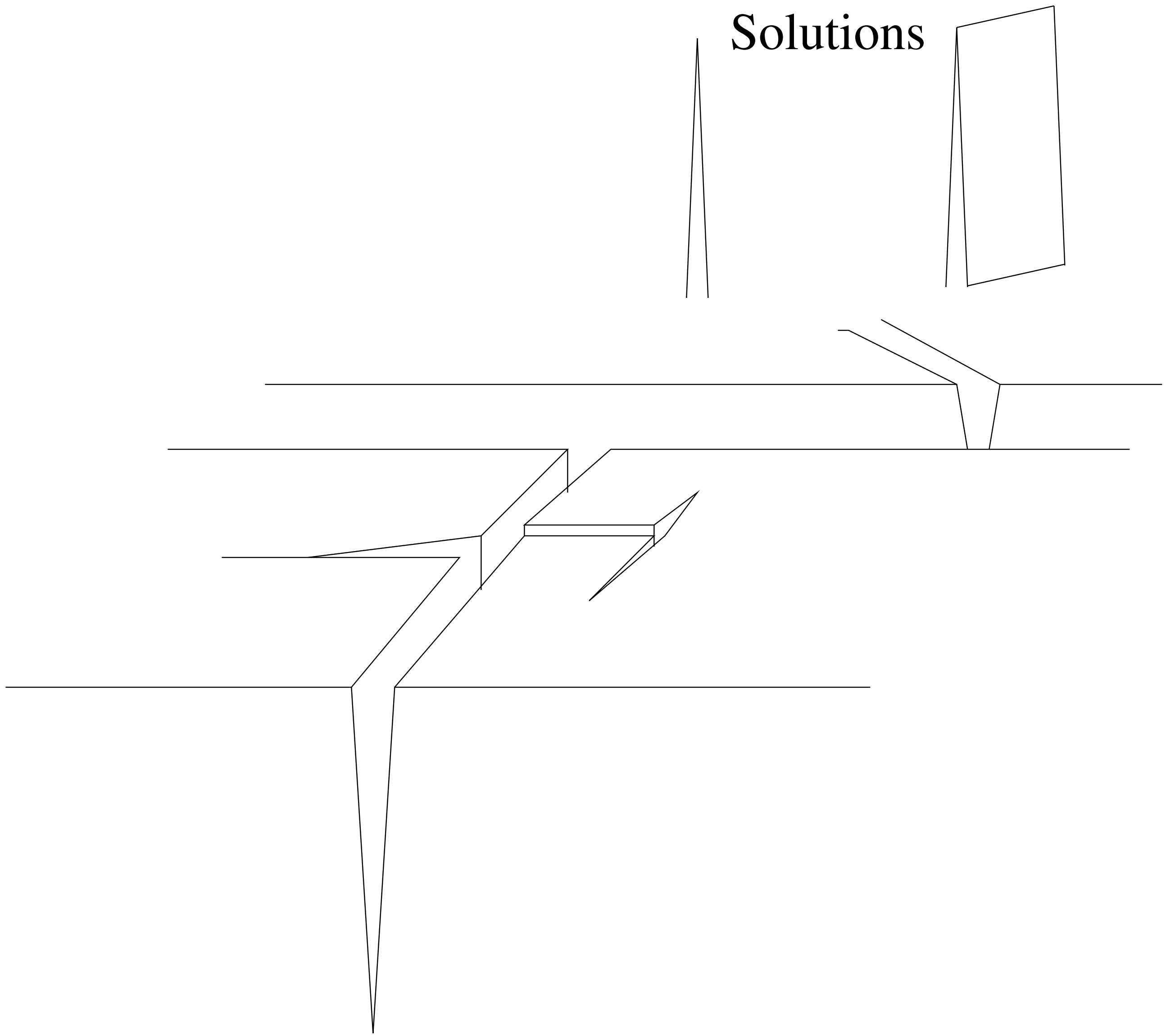
- Both use point mutation, no change in size or shape

Hill Climbing – Karst Landscape

- Many individuals of intermediate fitness don't have fitter neighbour
- len=11 solutions, no neighbour fitness>36, most <24
- length ≤ 14 solutions are (almost) isolated
- sampling $> 14 \Rightarrow$ similar
- neighbours with same fitness rises ≈ 1.5 length

  plateau, fitness gives no guidance

Karst Landscape



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

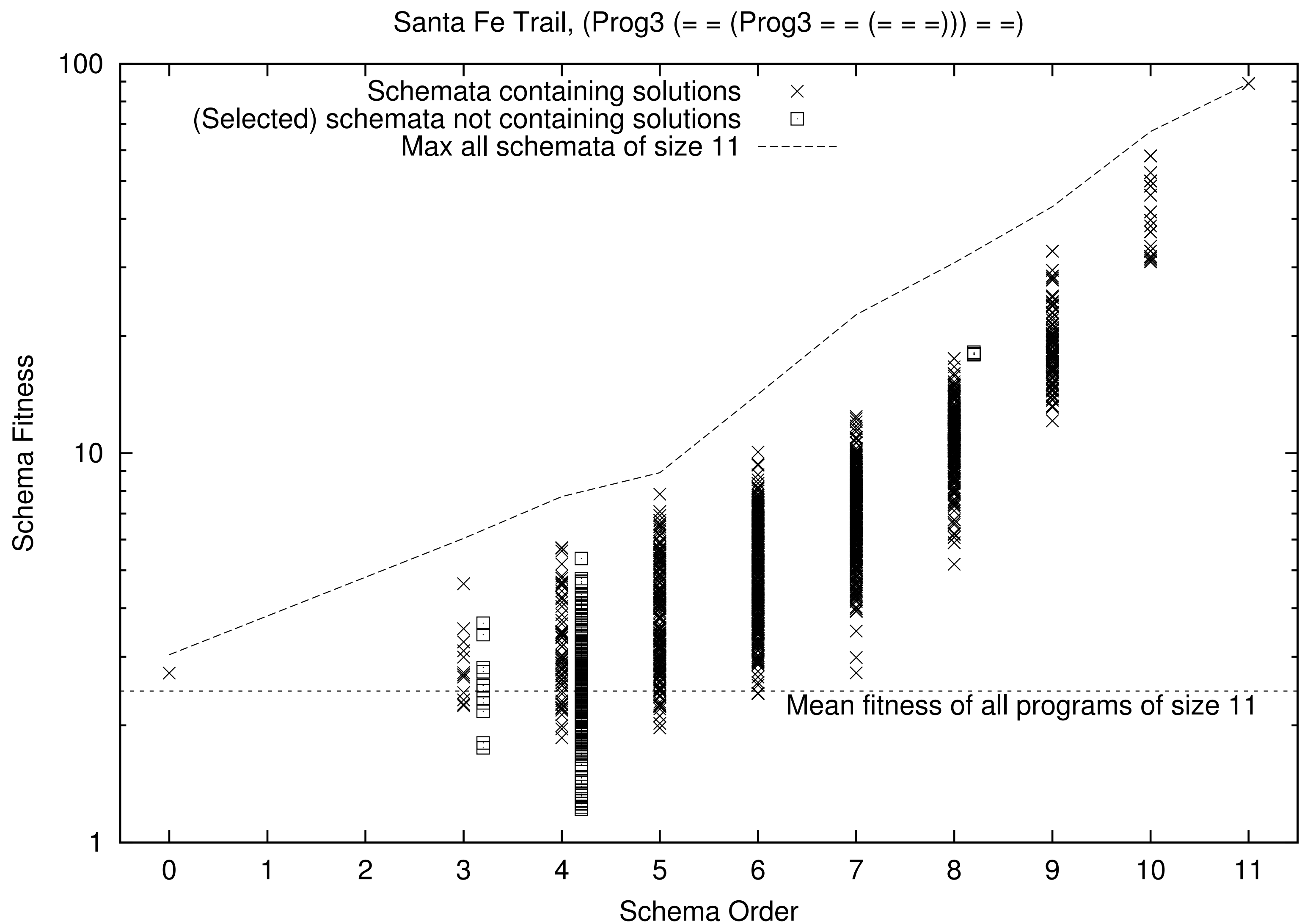
Looking for Building Blocks

- BB small components of solution with above average fitness

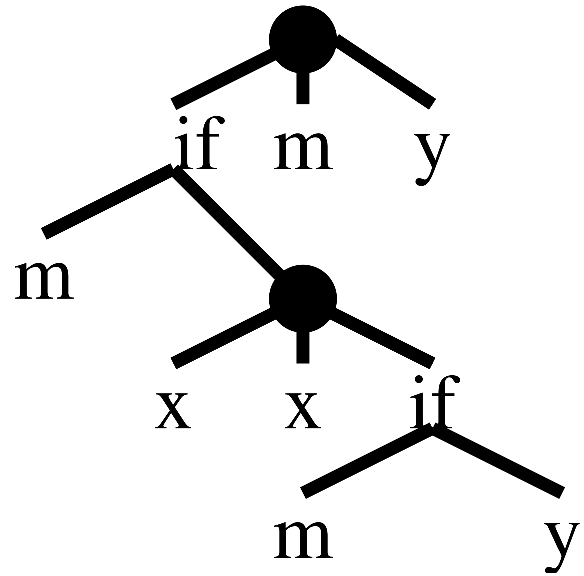
- Length ≤ 13

- 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



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 ≈ 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

  1. Initial generations deceptive. Price's Theorem
  2. Later difficult because a) depth limit and strong selection confine crossover so only *improvements* are accepted b) little crossover activity near root

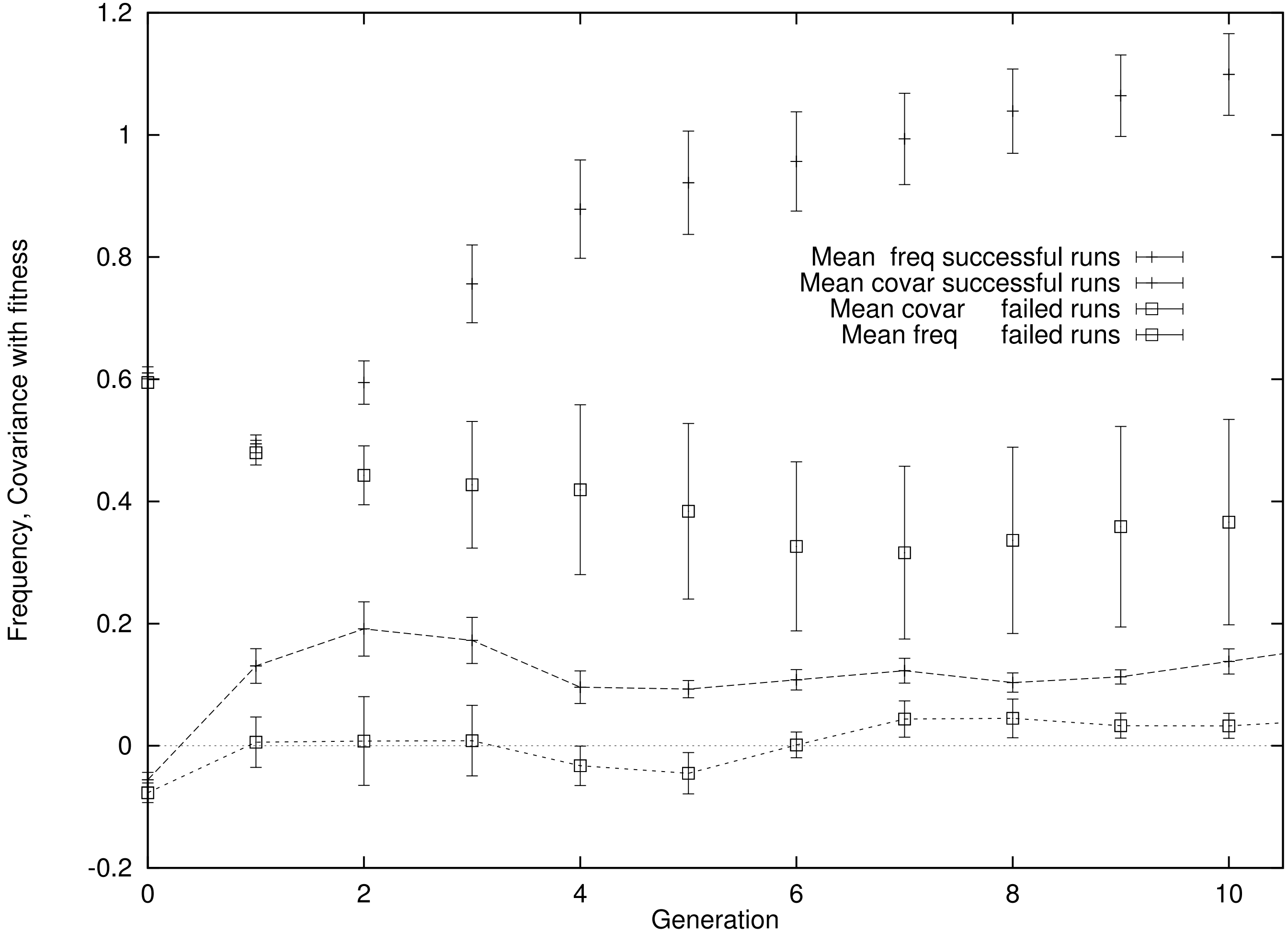
     Hill climbing model $\Rightarrow$ time $O(2^{2D})$. Cf. 58 and 59

     Search space $O\left(2^{2^{D+1}}\right)$

MAX Problem Parameters

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

Predicting which MAX runs will fail

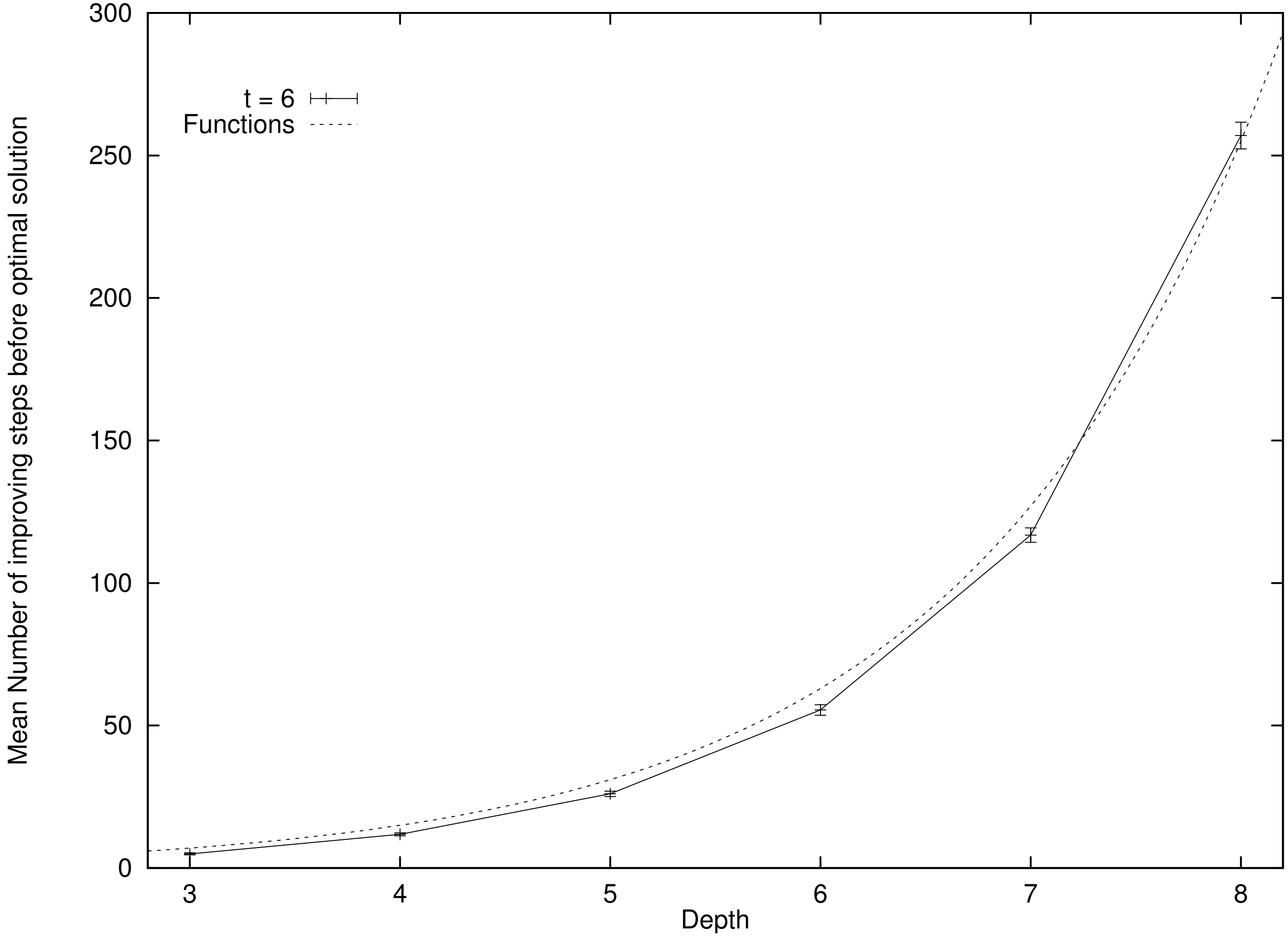


Initialy + gives better score than ×, so selection removes ×. This is especially important near root.

Covariance with fitness and frequency for × in the second level of the tree.

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

How Many Steps to Solution

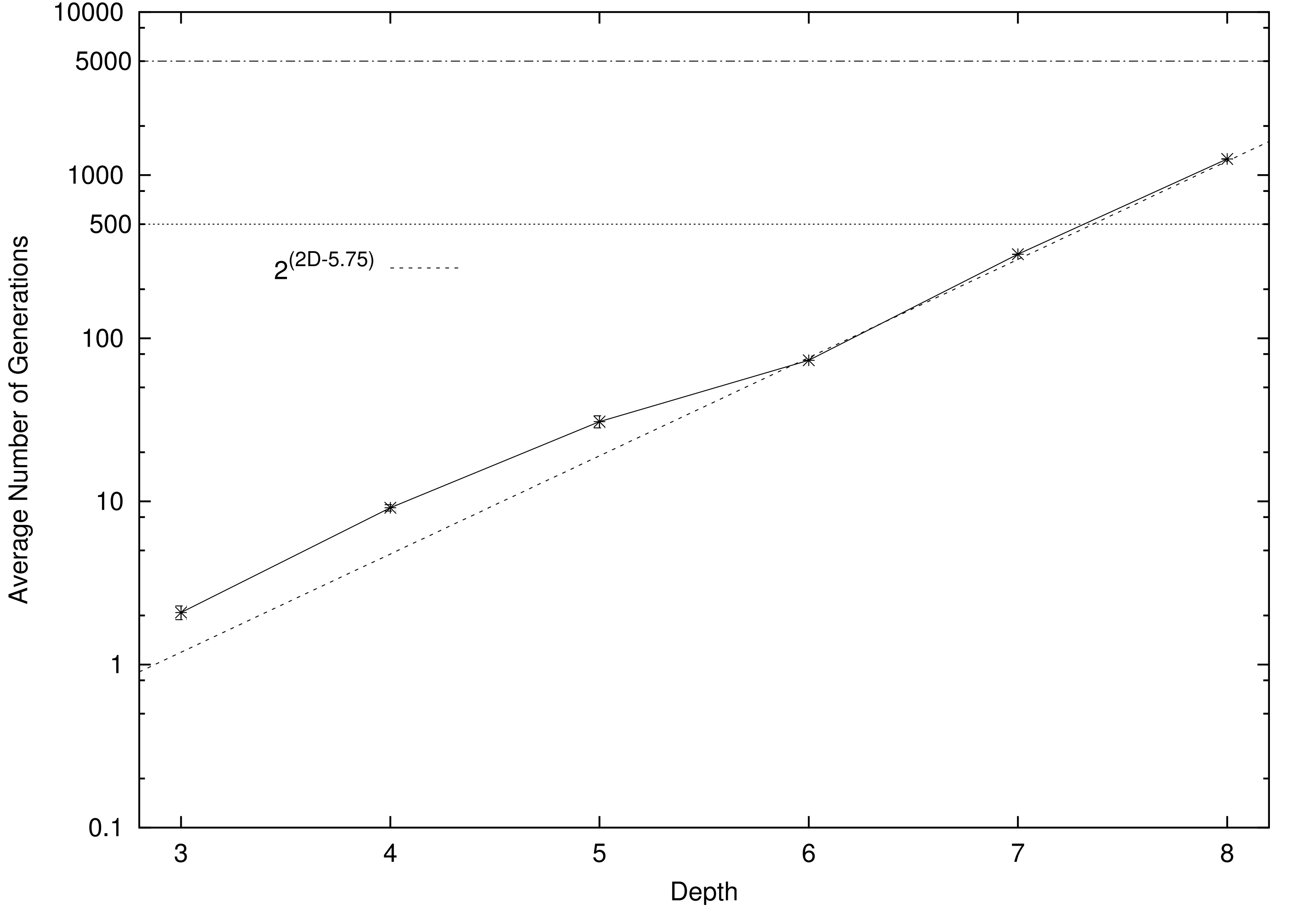


Time for crossover to replace + by $\times$ near root. Model 2. p55 $O(2^{2D})$ is reasonable.

Improvements made before optimal solution in successful runs.

Number of steps $\approx$ functions in tree

Mean number of generations to solve (successful 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 initialy 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.

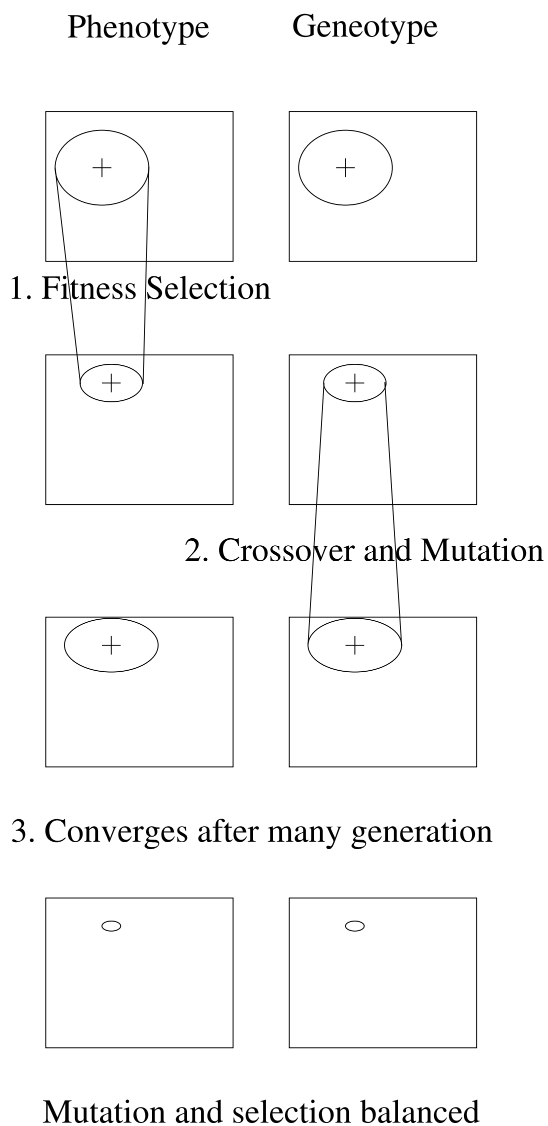
- Hard because only crossovers which make immediate improvements are accepted into the population. Many improvements are needed from random start. C.f. GA long path problems.

Bloat = Convergence

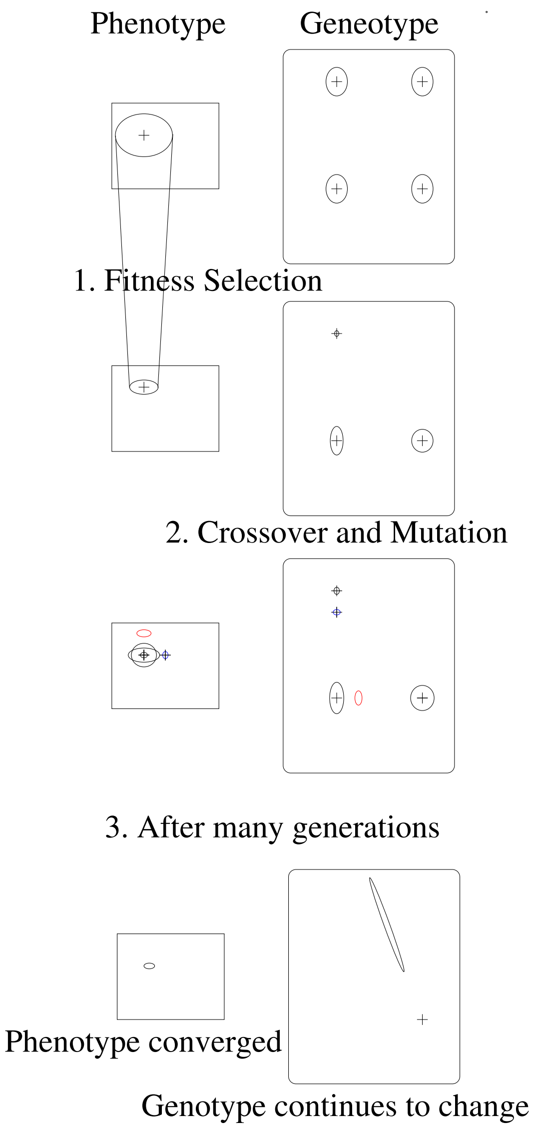
- Convergence in GP
- Evolution of Size and Shape
- No worse than parabolic growth prediction (binary trees)

Convergence

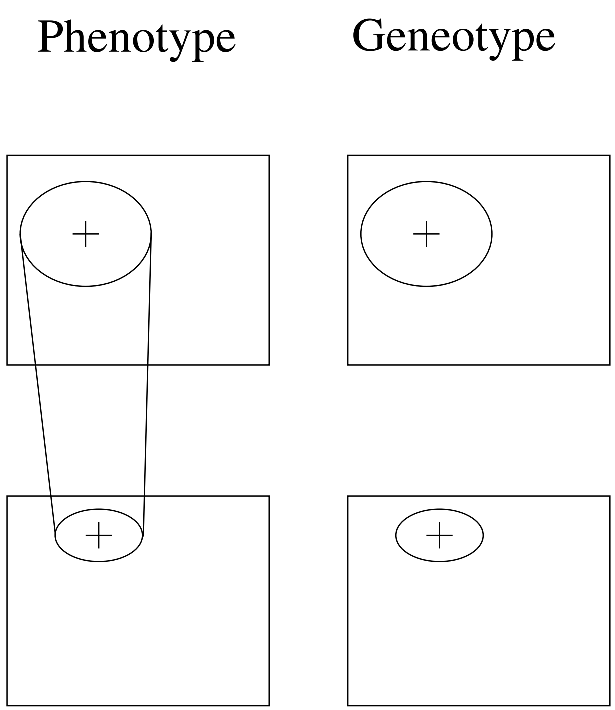
Genetic Algorithms

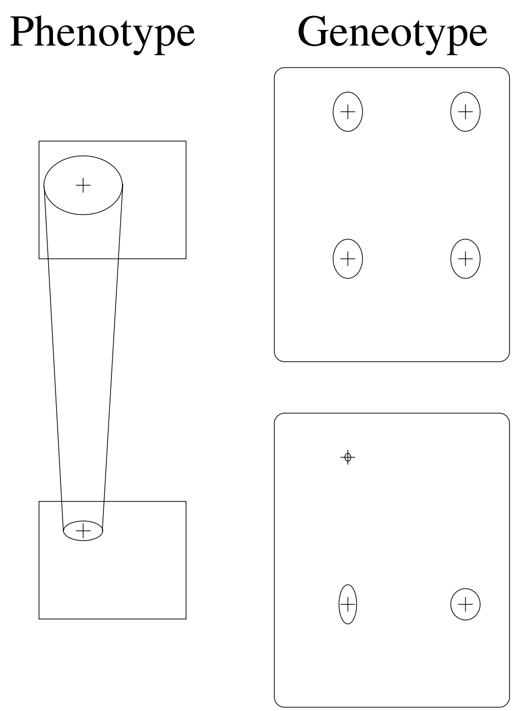


Genetic Programming



Fitness Selection Acts on Phenotype





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

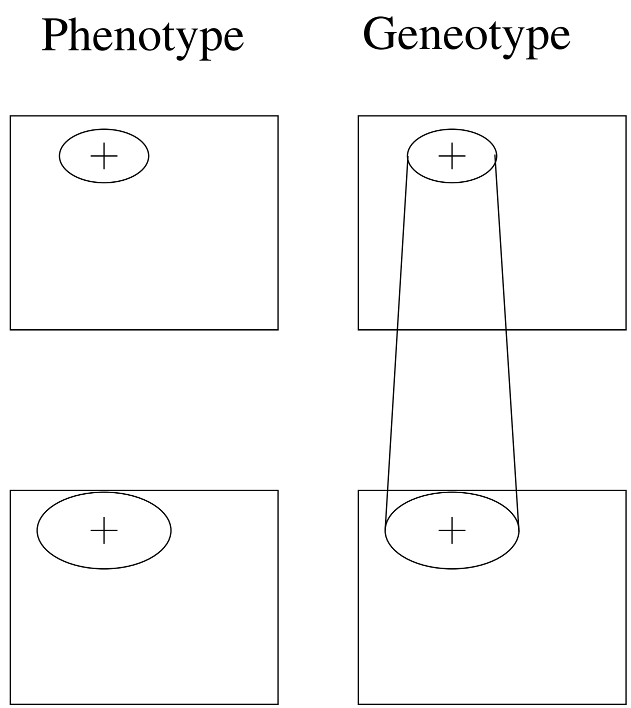
GP 1-to-many mapping

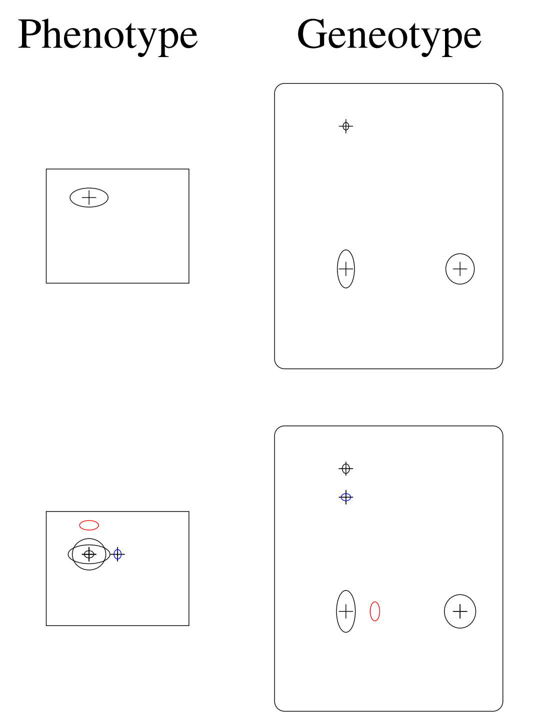
Spread of phenotypes (unequally) reduced

1:1 mapping, identical reduction in genotypes

Complex mapping, uneven reduction in genotypes

Crossover and Mutation Spread the Genotype

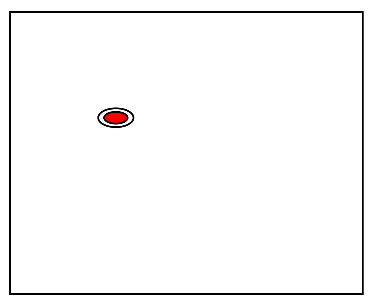


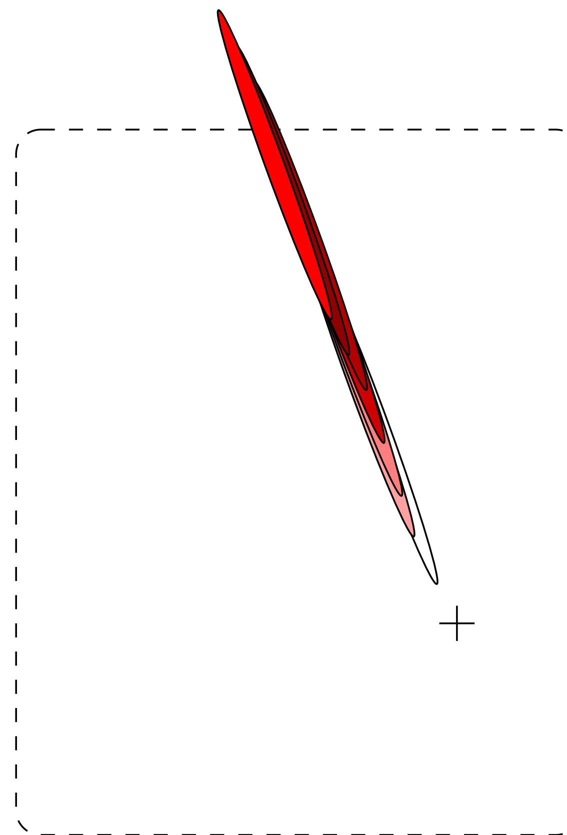


fixed mapping, spread of phenotype

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

GP Convergence Genotype Continue 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).

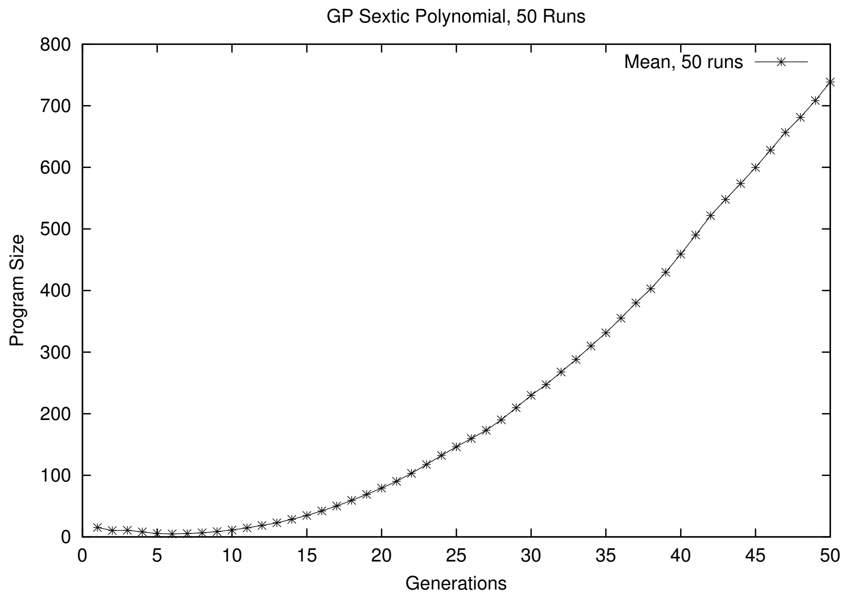
Genotype cluster does not stabilise but continues to evolve from a single point. The population's ancestor, i.e. the individual program where most of its genetic material came from.

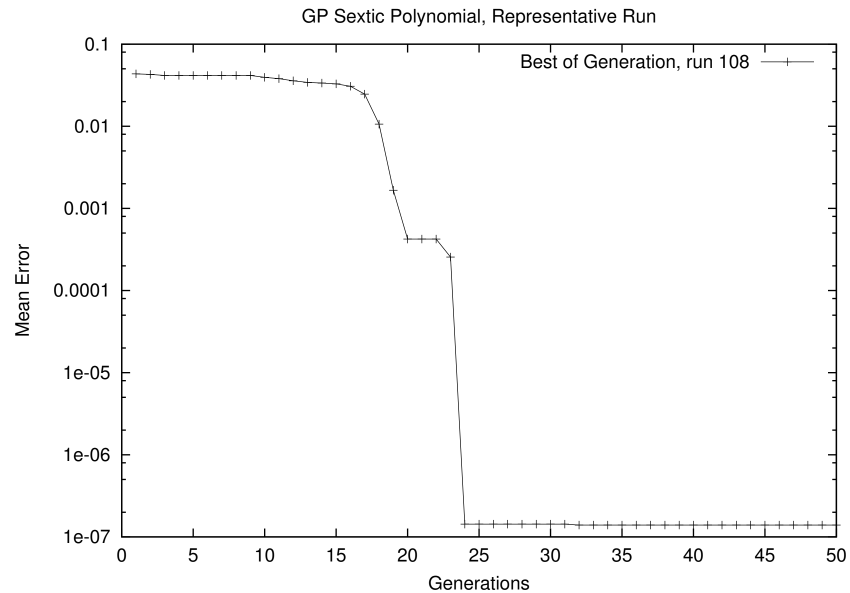
Since each fit child's genotype tends to be bigger than its parents there is a progressive increase in size, which we know as bloat.

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 ≈ 10 generations
- No limit to increase??

Experimental Evidence



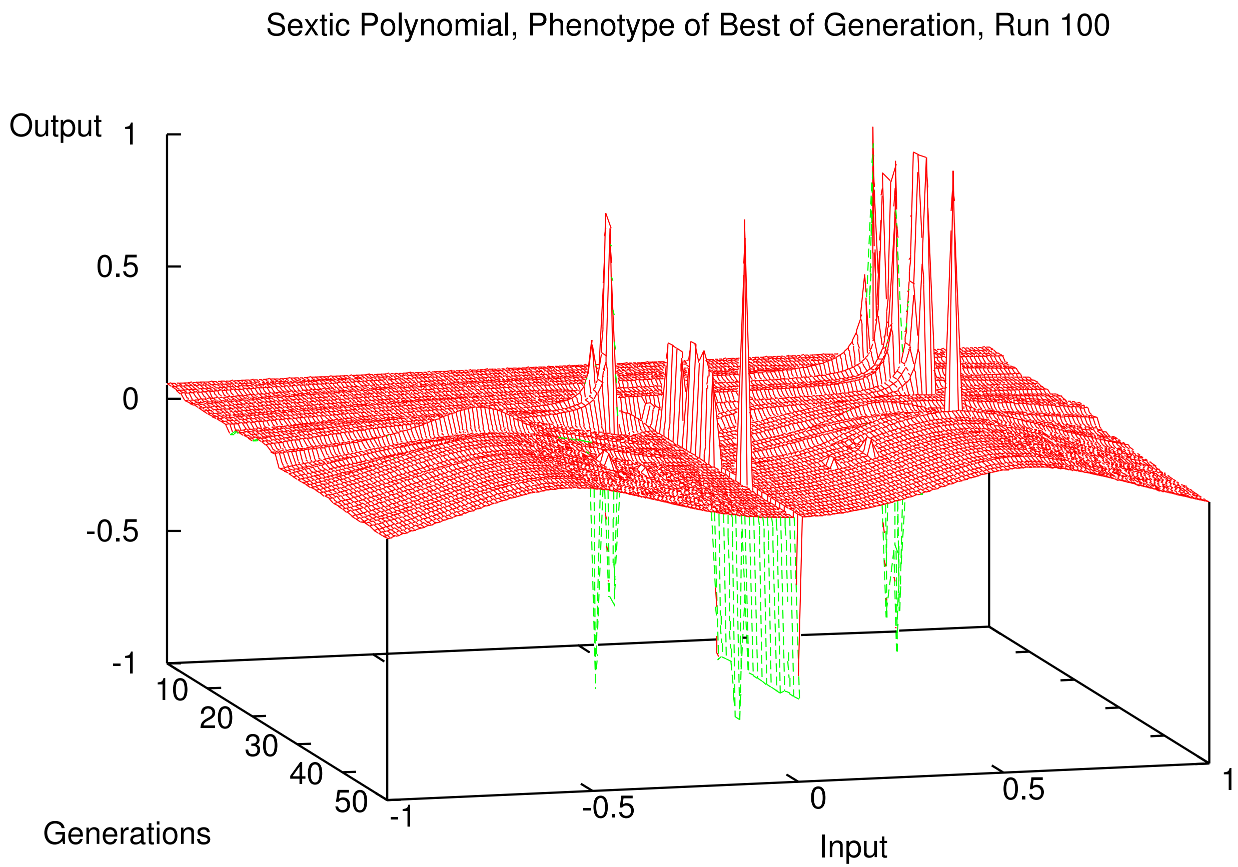


Size increase in most generations

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



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

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

Fitness Needed for Bloat

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

$$\Delta q = \frac{\text{Cov}(z,q)}{\overline{z}} \qquad \text{[Price, 1970]}$$

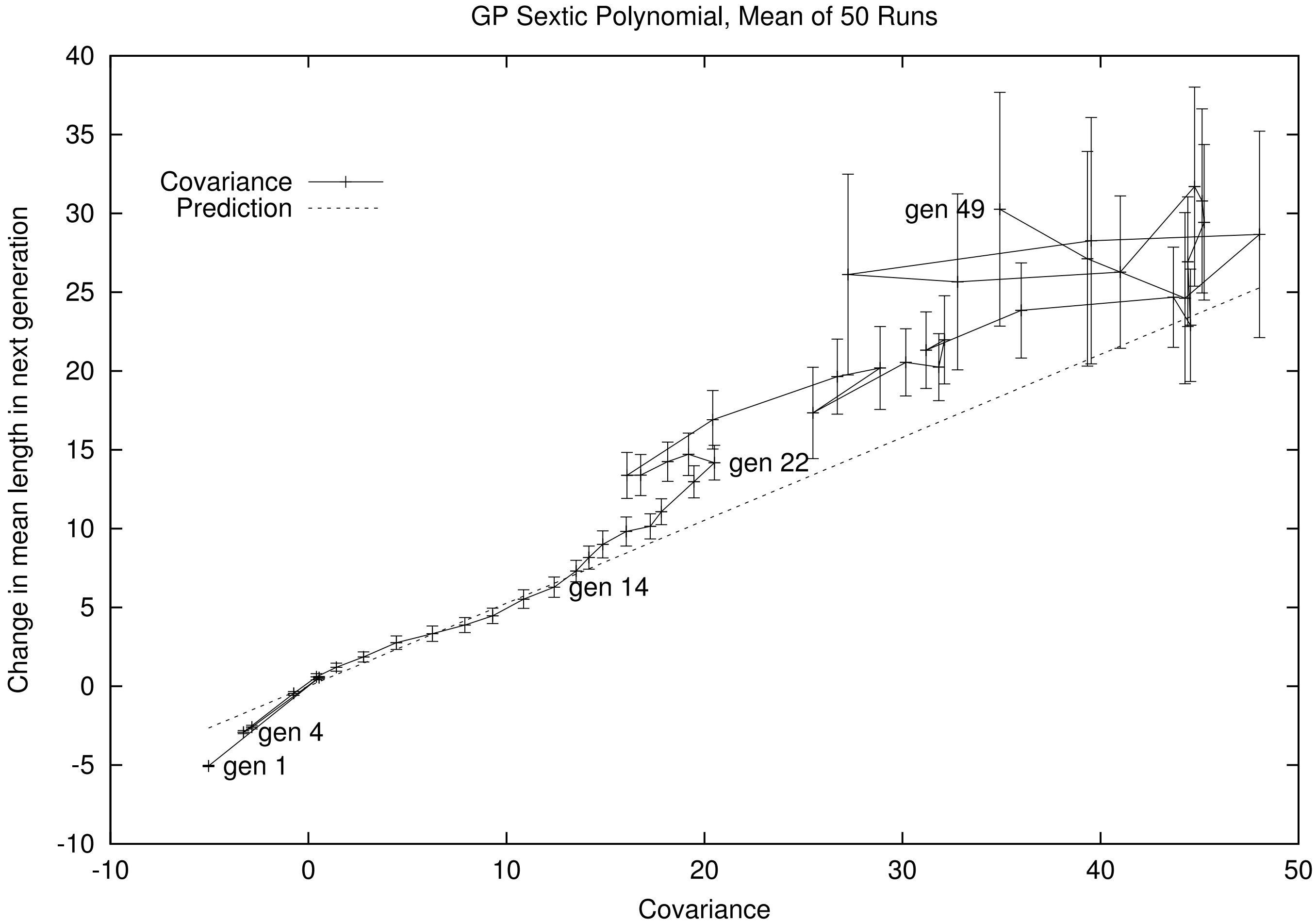
Holds if genetic operations are random with respect to gene.

Applies to program size in GP with crossover and mutation operators which have no size bias [Langdon *et al.*, 1999]

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

$$E\Delta\text{size} \approx \frac{t}{\overline{z}}\text{Cov}((r/p)^{t-1}, \text{size}) \qquad \text{[Langdon and Poli, 1998a]}$$

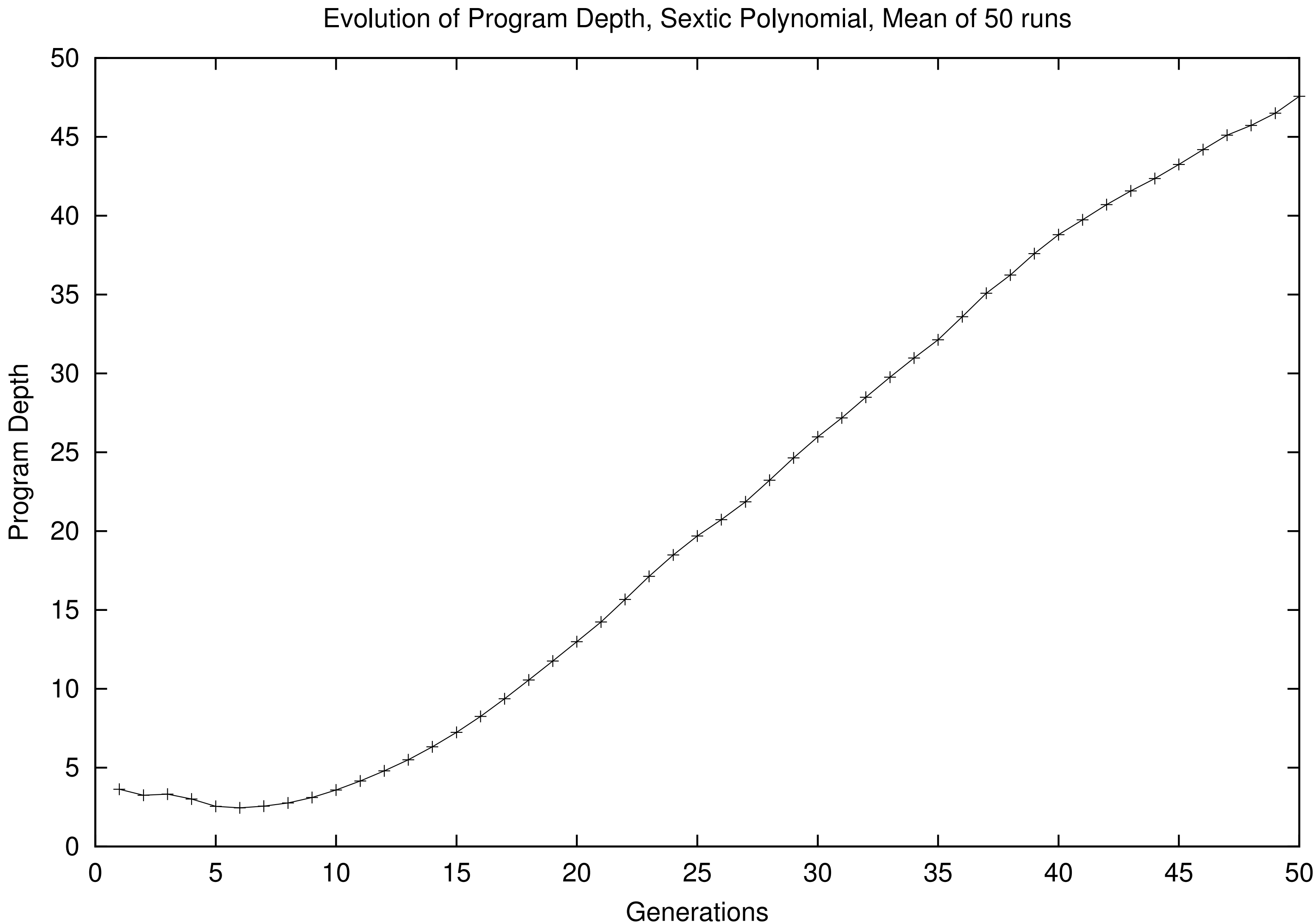
Covariance of Size and Fitness



Covariance of fitness and program size gives change in mean size from one generation to the next

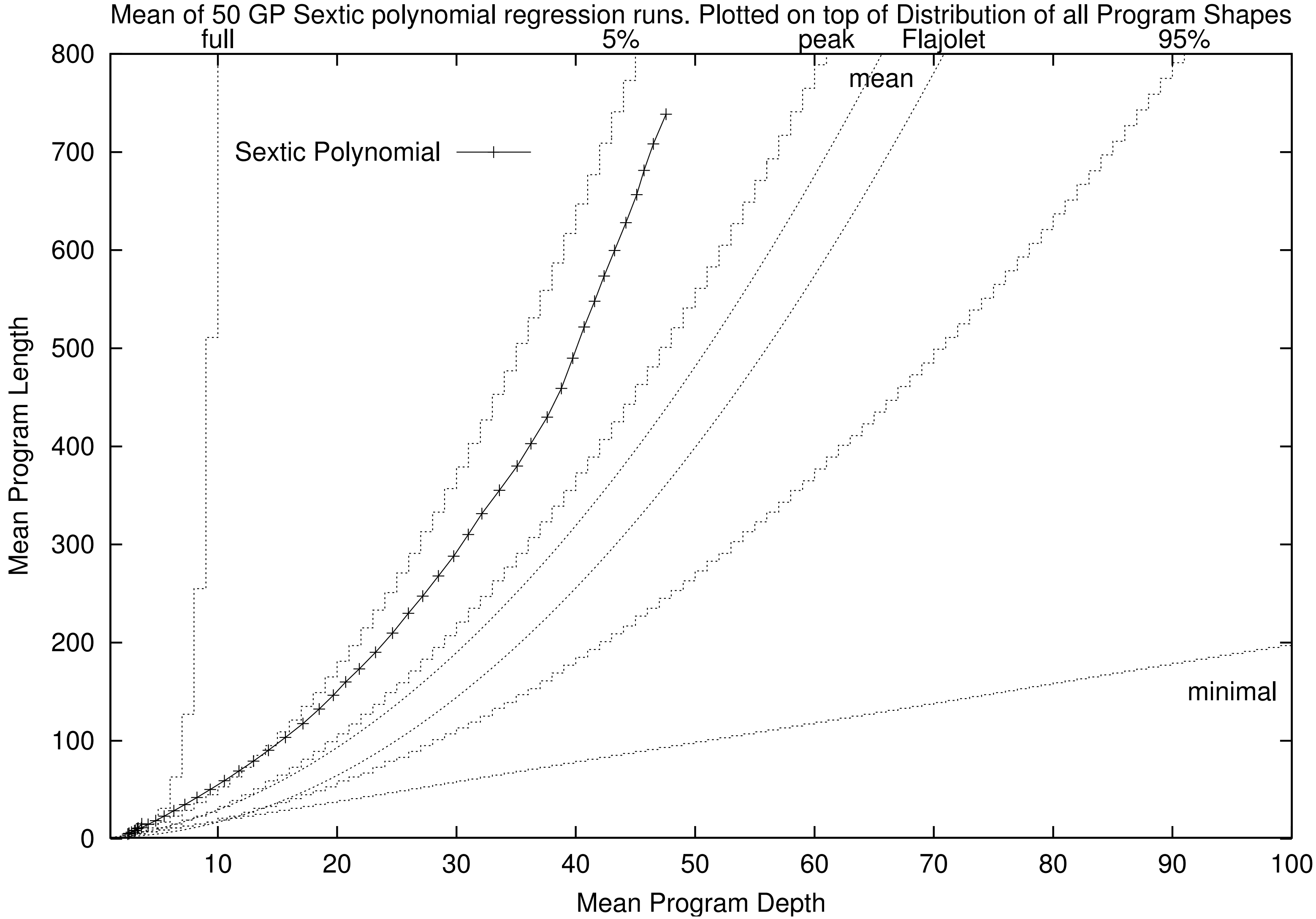
Positive increase (e.g. bloat) requires positive covariance, i.e. fitness variation in current generation

Linear Increase in Depth (Standard Crossover)



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

Evolution of Shape



Note movement from bushy (full) trees towards random trees

Note only bushy half of search space is used

Sub-quadratic Growth in Binary Trees

- Predicted $\lim_{t\to\infty}$ program size $= O(t^2)$
- Measured bloat $O(t^{1.2-1.5})$ $t \leq 50$ generations
- Test $O(t^2)$ 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 $= 2\sqrt{\pi(\text{internal nodes})}$ (ignoring terms O($N^{1/4}$) [Flajolet and Oldyzko, 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]

  Δdepth $= 0.5 \ldots 2.2$ per generation

  Variable between problems and individual runs

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

  - If $\lim_{t\to\infty}$ depth $\approx 2\sqrt{\pi\ \lfloor \text{size}/2 \rfloor}$
    $\lim_{t\to\infty}$ size $= O(\text{depth}^2) = O(\text{gens}^2)$

  - Fitting a power law to ridge (50–500) yields
    size $=$ O(gens$^{1.3}$)

Experiments

- Hundreds of generations, size = million on rapidly bloating populations
  - continuous: symbolic regression quartic polynomial [Koza, 1992]).
  - Discrete: 6-multiplexer (binary function set)

    64 fitness cases in parallel, submachine code GP [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

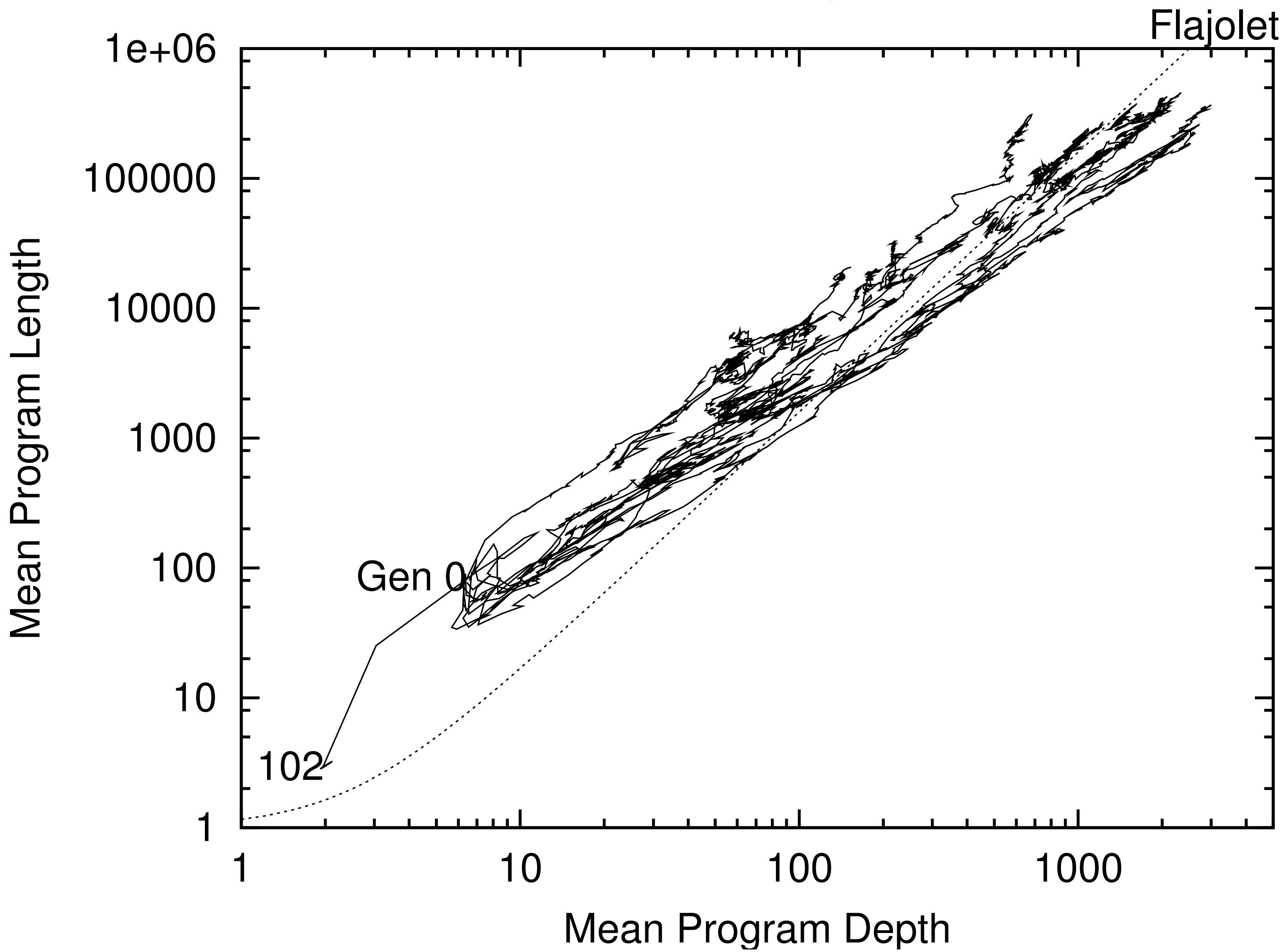
  In all runs most new generations do not find better fitness
  I.e. changes in size and shape are due to bloat

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

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

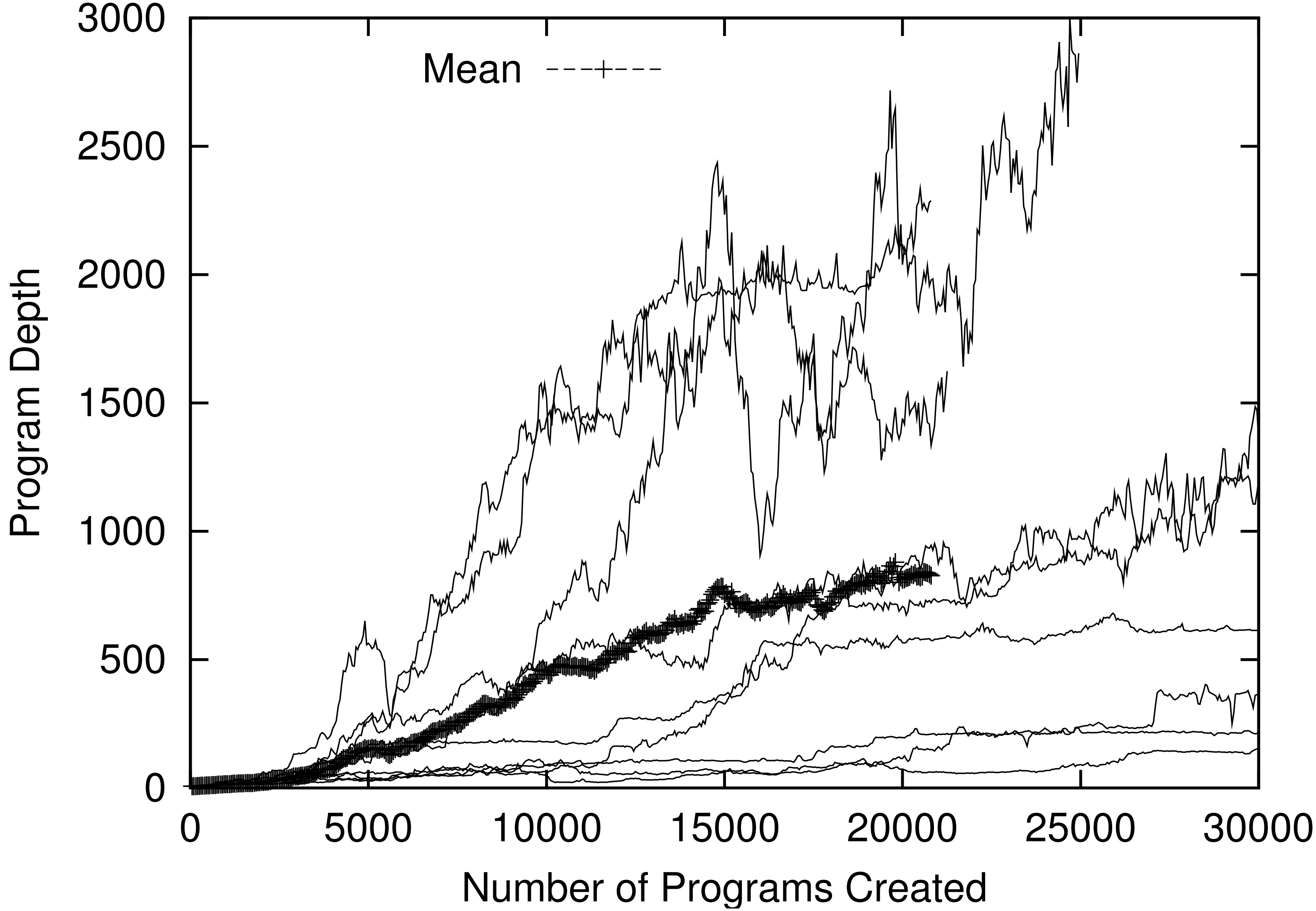
- The size power law varies widely. On average starts near 1.0 (generations 12–50) and steadily rises to 1.9 (12–400).

Evolution of Tree Shape: Quartic

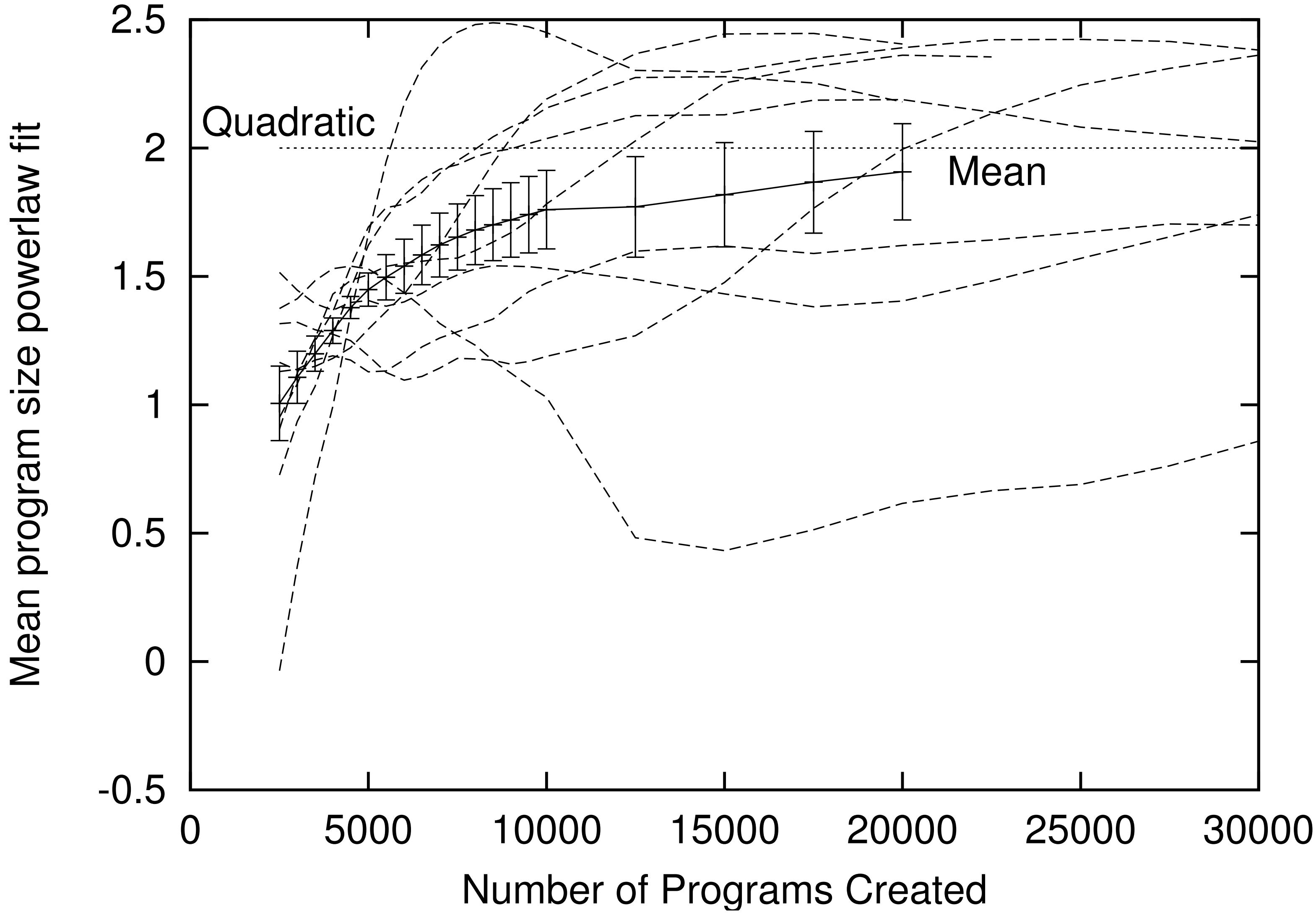


Note log scales

Mean Tree Depth: Quartic symbolic regression



Evolution Power Law Coefficient: Quartic



Nine bloating runs. Error bars show standard error

Results: Boolean Benchmark

- In all runs most new generations do not find better programs

  $\Rightarrow$ changes in size and shape are due to bloat

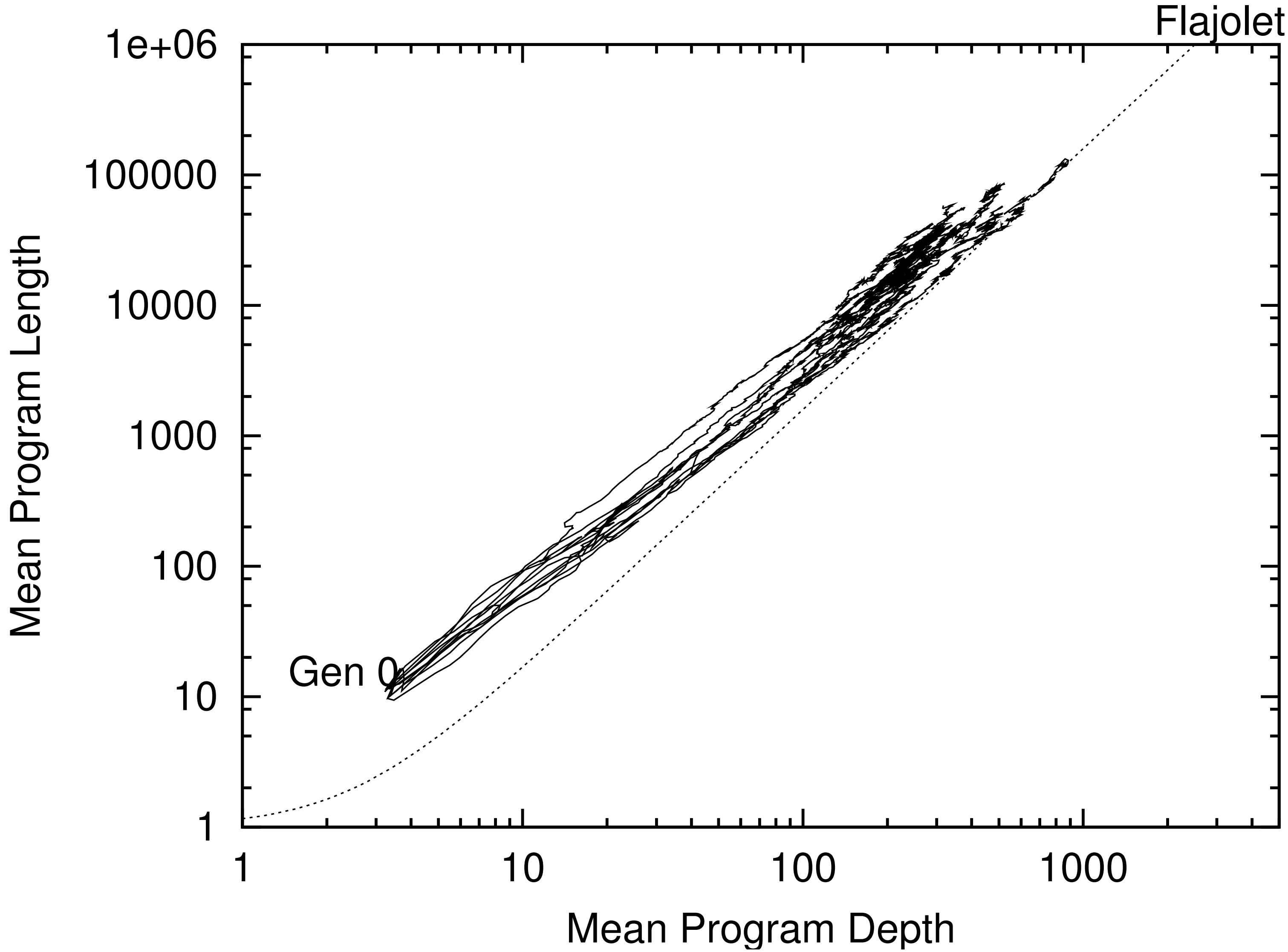
- On average each population evolves to lie close to the ridge and moves along it, slide 85

- 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.

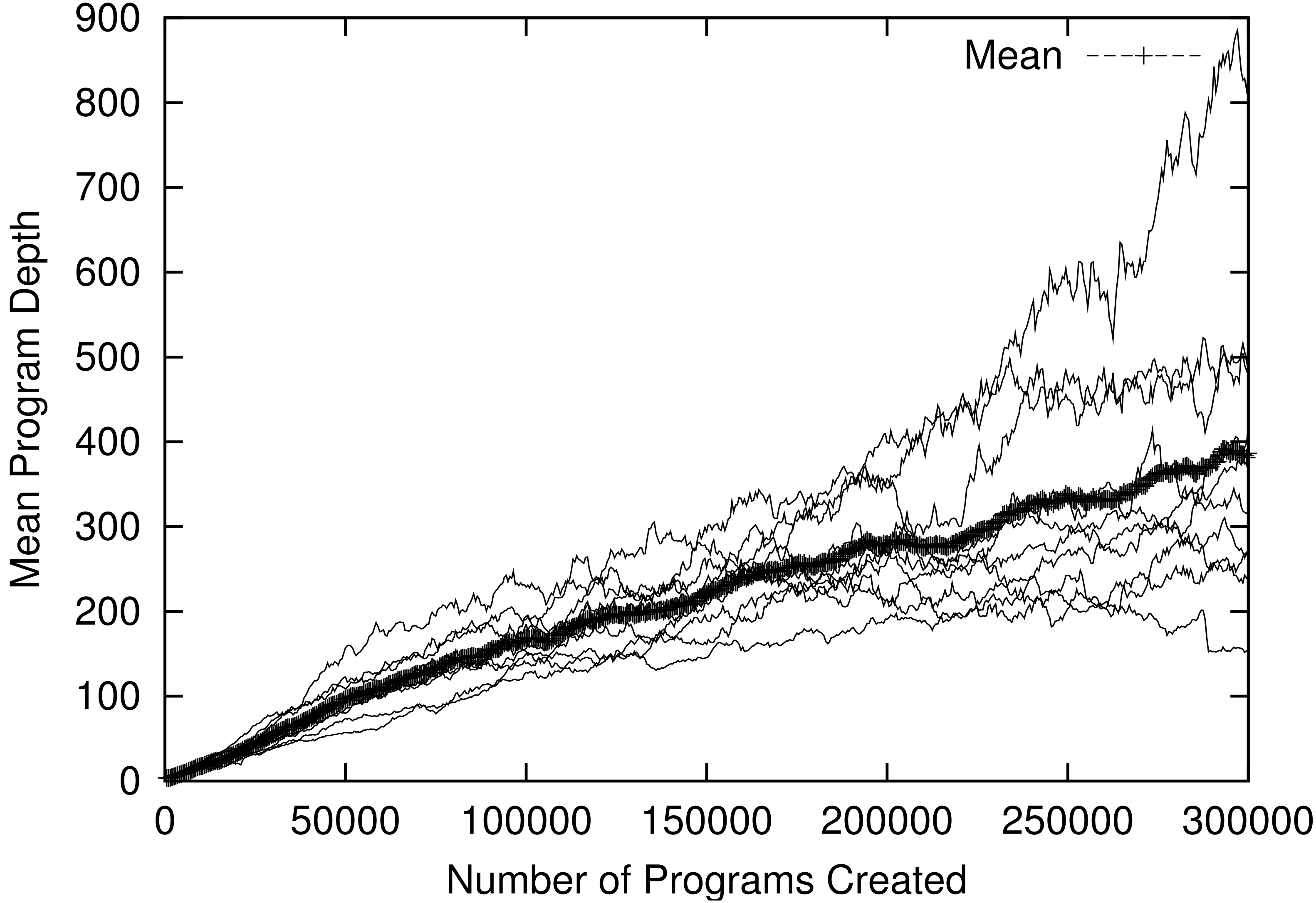
  On average starts at 1.25 and rises. By the end of the runs (generations 12–600) it reaches 1.5.

Evolution of tree Shape: binary 6-multiplexor

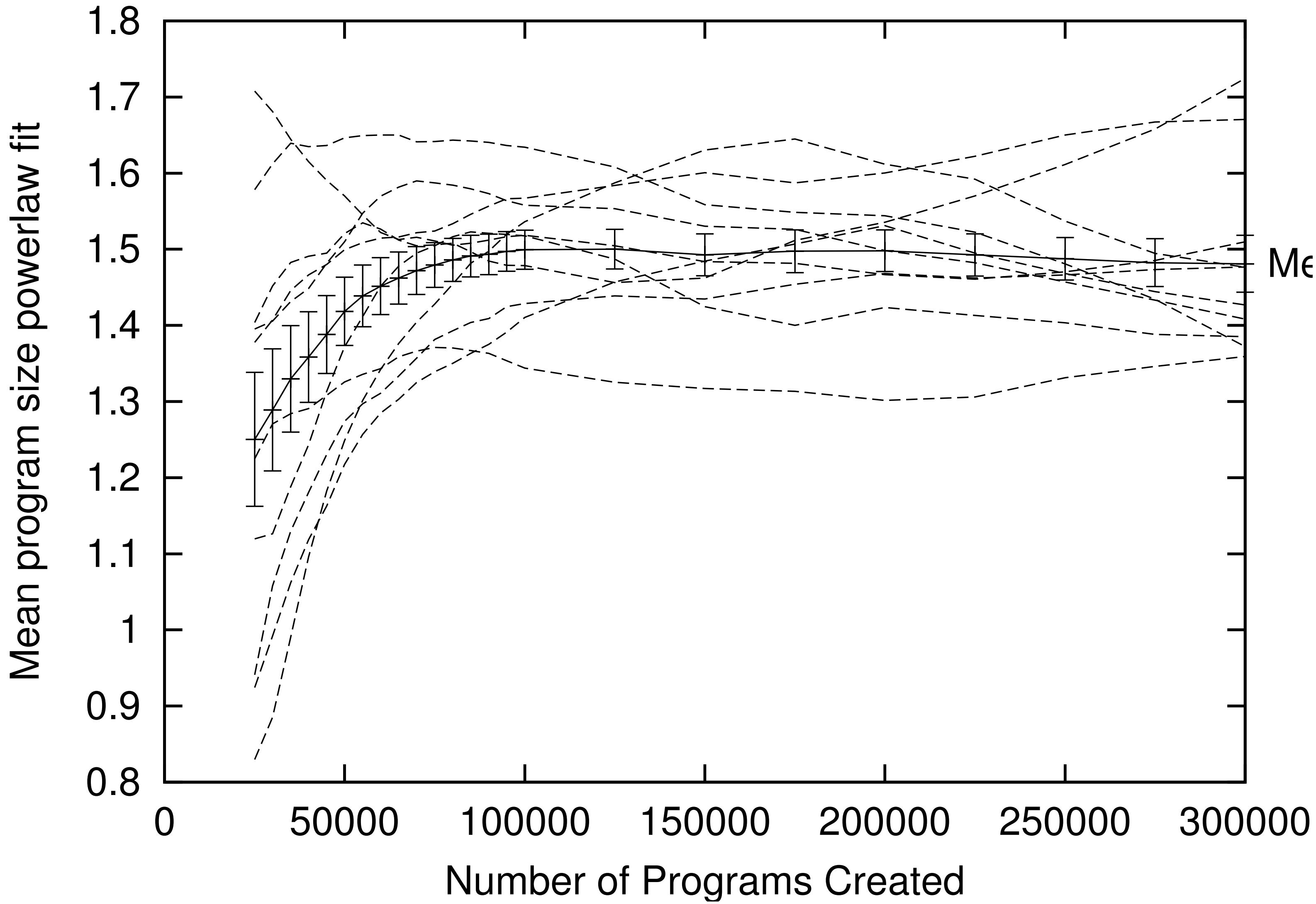


Note log scales

Evolution of tree Depth: binary 6-multiplexor



Evolution of power law coefficient: binary 6-multiplexor



Error bars indicate standard error

Why Quadratic Limit is Not Reached

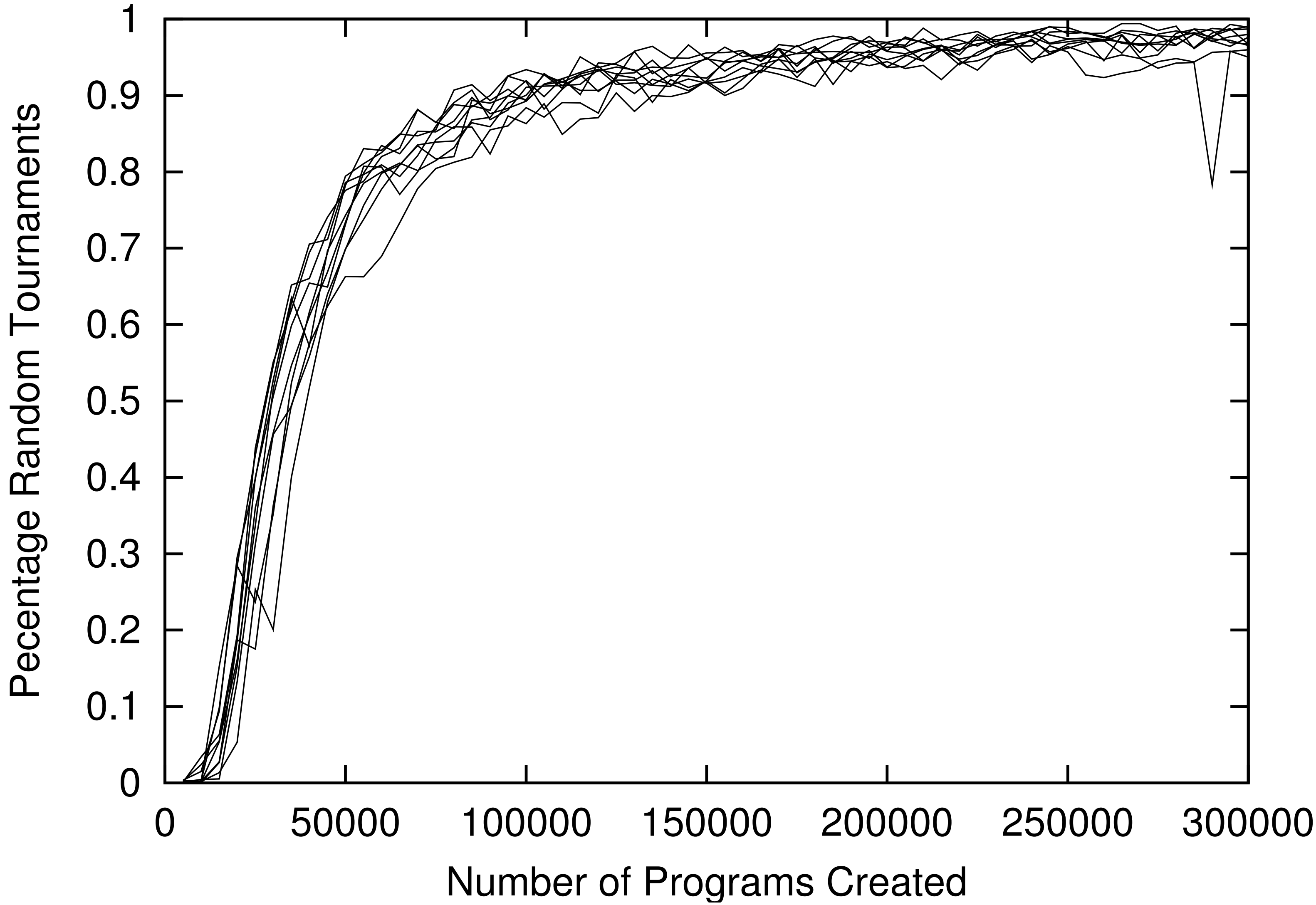
- Standard crossover may cease to be disruptive when the programs become very large

- 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% [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 ≈ 1 level per generation

- Predict average evolution of size, depth and shape

  Continuous $\lim_{g\to\infty}$ mean size $=$ O(generations$^{2.0}$)
  Discrete mean size $\leq$ O(generations$^{2.0}$)

  (Wide variation in population and between runs)

  New type of GP fitness convergence in discrete case

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

[Angeline, 1994] Peter John Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 4, pages 75–98. MIT Press, 1994.

[Angeline, 1998] Peter J. Angeline. Subtree crossover causes bloat. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 745–752, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[Blickle, 1996] Tobias Blickle. Evolving compact solutions in genetic programming: A case study. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pages 564–573, Berlin, Germany, 22-26 September 1996. Springer-Verlag.

[Chellapilla, 1997] Kumar Chellapilla. Evolutionary programming with tree mutations: Evolving computer programs without crossover. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 431–438, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[Christensen and Oppacher, 2002] Steffen Christensen and Franz Oppacher. An analysis of Koza's computational effort statistic for genetic programming. In James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 182–191, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.

[Daida *et al.*, 2003] Jason M. Daida, Adam M. Hilss, David J. Ward, and Stephen L. Long. Visualizing tree structures in genetic programming. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta,

M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1652–1664, Chicago, 12-16 July 2003. Springer-Verlag.

[Ehrenburg, 1996] Herman Ehrenburg. Improved direct acyclic graph handling and the combine operator in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 285–291, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[Ekart and Nemeth, 2001] Aniko Ekart and S. Z. Nemeth. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 2(1):61–73, March 2001.

[Feller, 1970] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, 3rd edition, 1970.

[Flajolet and Oldyzko, 1982] Philippe Flajolet and Andrew Oldyzko. The average height of binary trees and other simple trees. *Journal of Computer and System Sciences*, 25:171–213, 1982.

[Gathercole and Ross, 1996] Chris Gathercole and Peter Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 291–296, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[Handley, 1994] S. Handley. On the use of a directed acyclic graph to represent a population of computer programs. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pages 154–159, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[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, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, page 428, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[Iba *et al.*, 1994] Hitoshi Iba, Hugo de Garis, and Taisuke Sato. Genetic programming using a minimum description length principle. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 12, pages 265–284. MIT Press, 1994.

[Keijzer, 1996] Maarten Keijzer. Efficiently representing populations in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 13, pages 259–278. MIT Press, Cambridge, MA, USA, 1996.

[Kinnear, Jr., 1993] Kenneth E. Kinnear, Jr. Evolving a sort: Lessons in genetic programming. In *Proceedings of the 1993 International Conference on Neural Networks*, volume 2, pages 881–888, San Francisco, USA, 28 March-1 April 1993. IEEE Press.

[Kinnear, Jr., 1994] Kenneth E. Kinnear, Jr. Alternatives in automatic function definition: A comparison of performance. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 6, pages 119–141. MIT Press, 1994.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[Koza, 1994] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[Langdon and Nordin, 2000] W. B. Langdon and J. P. Nordin. Seeding GP populations. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 304–315, Edinburgh, 15-16 April 2000. Springer-Verlag.

[Langdon and Poli, 1997a] W. B. Langdon and R. Poli. An analysis of the MAX problem in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco

Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 222–230, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[Langdon and Poli, 1997b] W. B. Langdon and R. Poli. Fitness causes bloat. In P. K. Chawdhry, R. Roy, and R. K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag London, 23-27 June 1997.

[Langdon and Poli, 1998a] W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 37–48, Paris, 14-15 April 1998. Springer-Verlag.

[Langdon and Poli, 1998b] W. B. Langdon and R. Poli. Genetic programming bloat with dynamic fitness. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS*, pages 96–112, Paris, 14-15 April 1998. Springer-Verlag.

[Langdon and Poli, 1998c] W. B. Langdon and R. Poli. Why ants are hard. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[Langdon and Poli, 2002] W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[Langdon *et al.*, 1999] William B. Langdon, Terry Soule, Riccardo Poli, and James A. Foster. The evolution of size and shape. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA, June 1999.

[Langdon, 1998a] W. B. Langdon. The evolution of size in variable length representations. In *1998 IEEE*

International Conference on Evolutionary Computation, pages 633–638, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.

[Langdon, 1998b] William B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston, 24 April 1998.

[Langdon, 1999] W. B. Langdon. Scaling of program tree fitness spaces. *Evolutionary Computation*, 7(4):399–428, Winter 1999.

[Langdon, 2000] William B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119, April 2000.

[Langdon, 2002] W. B. Langdon. Convergence rates for the distribution of program outputs. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 812–819, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[Langdon, 2003a] W. B. Langdon. Convergence of program fitness landscapes. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1702–1714, Chicago, 12-16 July 2003. Springer-Verlag.

[Langdon, 2003b] W. B. Langdon. The distribution of reversible functions is Normal. In Rick L. Riolo and Bill Worzel, editors, *Genetic Programming Theory and Practise*, chapter 11, pages 173–188. Kluwer, 2003.

[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*

Algorithms VII, pages 183–202, Torremolinos, Spain, 4-6 September 2003. Morgan Kaufmann.

[Langdon, 2004] W. B. Langdon. Global distributed evolution of L-systems fractals. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming, Proceedings of EuroGP'2004*, volume 3003 of *LNCS*, pages 349–358, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.

[Luke, 2003] Sean Luke. Modification point depth and genome growth in genetic programming. *Evolutionary Computation*, 11(1):67–106, 2003.

[McPhee and Miller, 1995] Nicholas Freitag McPhee and Justin Darwin Miller. Accurate replication in genetic programming. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[McPhee and Poli, 2001] Nicholas Freitag McPhee and Riccardo Poli. A schema theory analysis of the evolution of size in genetic programming with linear representations. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 108–125, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.

[Nordin and Banzhaf, 1995] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, PA, USA, 15-19 July 1995. Morgan Kaufmann.

[Poli and Langdon, 1999] Riccardo Poli and William B. Langdon. Sub-machine-code genetic programming. In Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 13, pages 301–323. MIT Press, Cambridge, MA, USA, June 1999.

[Price, 1970] George R. Price. Selection and covariance. *Nature*, 227, August 1:520–521, 1970.

[Reeves and Rowe, 2003] Colin R. Reeves and Jonathan E. Rowe. *Genetic Algorithms–Principles and Perspectives: A Guide to GA Theory*. Kluwer Academic Publishers, 2003.

[Rosca, 1997] Justinian P. Rosca. Analysis of complexity drift in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 286–294, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[Smith and Harries, 1998] Peter W. H. Smith and Kim Harries. Code growth, explicitly defined introns, and alternative selection schemes. *Evolutionary Computation*, 6(4):339–360, Winter 1998.

[Soule and Foster, 1997] Terence Soule and James A. Foster. Code size and depth flows in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 313–320, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[Soule and Foster, 1998] Terence Soule and James A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4):293–309, Winter 1998.

[Soule and Heckendorn, 2002] Terence Soule and Robert B. Heckendorn. An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 3(3):283–309, September 2002.

[Zhang and Mühlenbein, 1995] Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

[Zhang and Mühlenbein, 1996] Byoung-Tak Zhang and Heinz Mühlenbein. Adaptive fitness functions for dynamic growing/pruning of program trees. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming*

2, chapter 12, pages 241–256. MIT Press, Cambridge, MA, USA, 1996.

[Zhang, 1997] Byoung-Tak Zhang. A taxonomy of control schemes for genetic code growth. Position paper at the Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97, 20 July 1997.