

Genetic Programming Theory

Riccardo Poli
Department of Computing and Electronic Systems
University of Essex

Copyright is held by the author/owner(s).
GECCO '08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-131-6/08/07.

July 2008

2

Overview

- Motivation
- Search space characterisation
 - How many programs?
 - Limiting fitness distributions
 - Implications
- GP search characterisation
 - Schema theory
 - Search bias
- Bloat
 - What's bloat?
 - Reasons
 - How to avoid it
- Conclusions

Motivation

July 2008

4

Understanding GP Search Behaviour with Empirical Studies

- We can perform **many GP runs** with a **small set of problems** and a **small set of parameters**
- We record the variations of **certain numerical descriptors**.
- Then, we **suggest explanations** about the behaviour of the system that are compatible with (and could explain) the empirical observations.

Problem with Empirical Studies

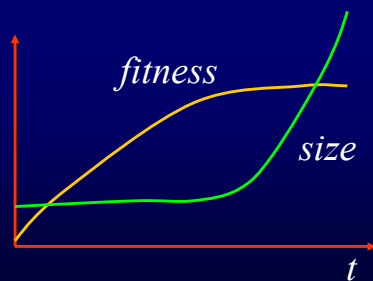
- GP is a complex adaptive system with **zillions of degrees of freedom**.
- So, any small number of descriptors can capture only a fraction of the complexities of such a system.
- **Choosing** which problems, parameter settings and **descriptors** to use is an **art form**.
- **Plotting the wrong data increases the confusion** about GP's behaviour, rather than clarify it.

Example: Bloat

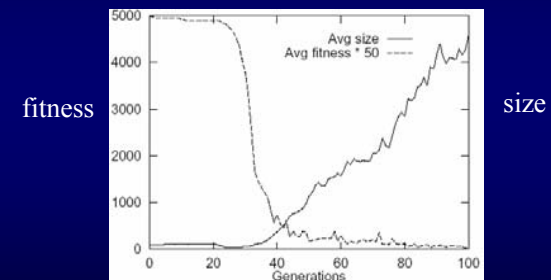
- What's bloat?
- Click [here](#) or [here](#) to find out

Bloat

- **Bloat** = growth without (significant) return in terms of fitness.



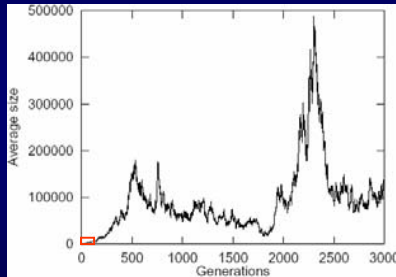
Fitness vs size



- Bloat exists and **continues forever**, right?

Why do we need theory?

- Empirical studies are rarely conclusive



- Qualitative theories can be incomplete

Search Space Characterisation

How many programs in the search space?

n_d = Number of trees of depth at most d

$$n_0 = |\mathcal{P}_0| \quad n_d = \sum_{a=0}^{a_{\max}} |\mathcal{P}_a| \times (n_{d-1})^a$$

Example

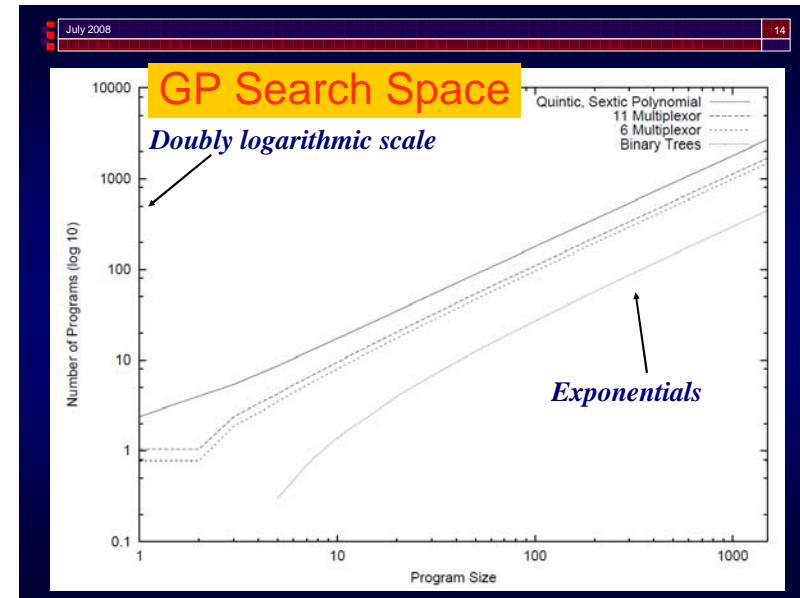
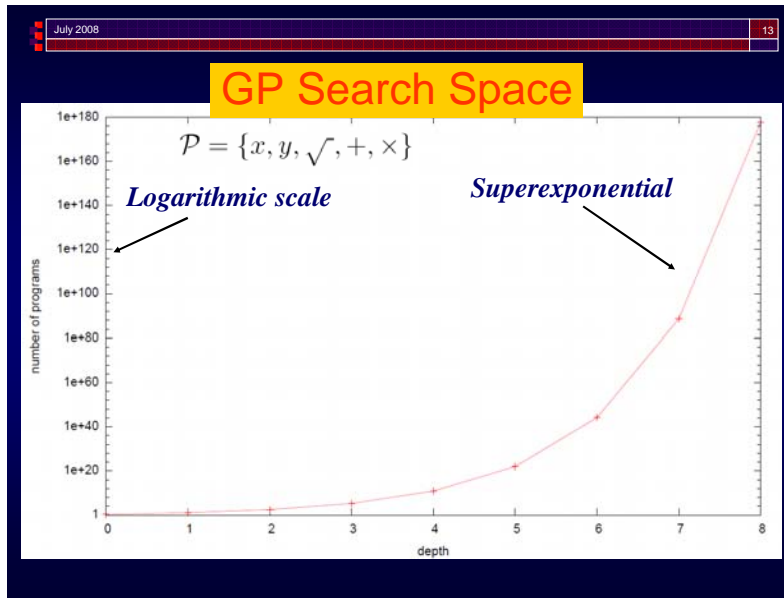
$$\mathcal{P} = \{x, y, \sqrt{}, +, \times\}$$

$$a_{\max} = 2, \mathcal{P}_0 = \{x, y\}, \mathcal{P}_1 = \{\sqrt{}\} \quad \mathcal{P}_2 = \{+, \times\}$$

$$n_0 = 2$$

$$n_1 = 2 + 1 \times (n_0) + 2 \times (n_0)^2 = 12$$

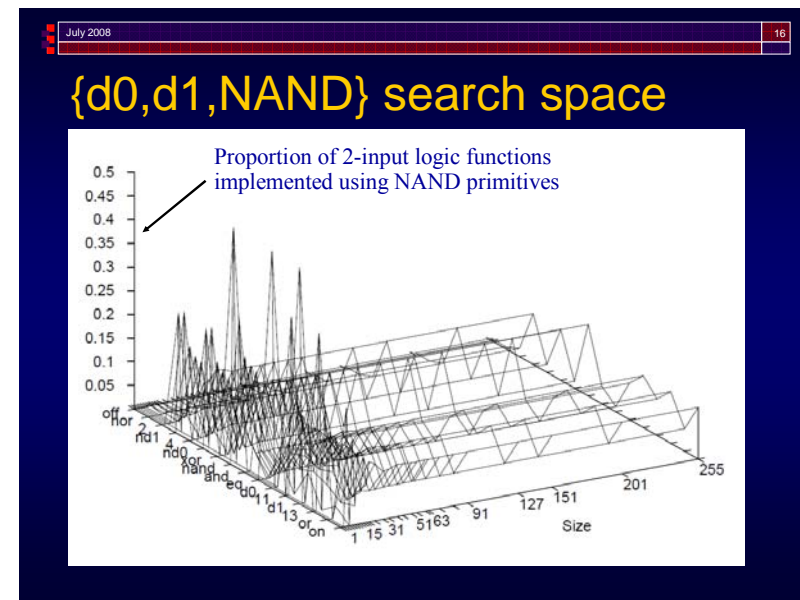
$$n_2 = 2 + 1 \times (n_1) + 2 \times (n_1)^2 = 302$$



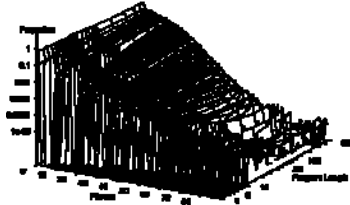
July 2008 15

GP cannot possibly work!

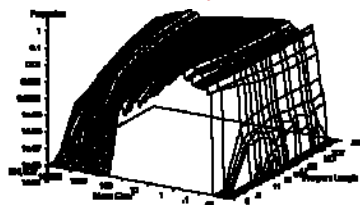
- ❑ The GP search space is immense, and so any search algorithm can only explore a tiny fraction of it (e.g. 10^{-1000} %).
- ❑ Does this mean GP cannot possibly work?
Not necessarily.
- ❑ We need to know the ratio between the size of solution space and the size of search space



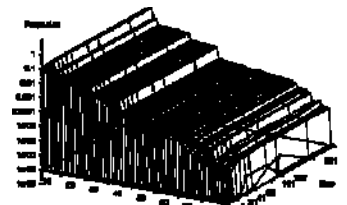
Proportion of Ant programs with each score



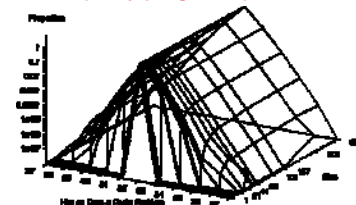
Error Distribution Sextic Polynomial Problem



Distribution of 3 bit Boolean Functions



Even-6 parity program space



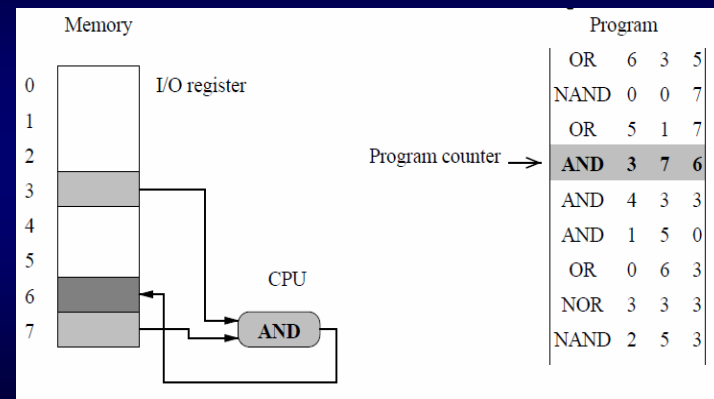
Limiting distribution

- Empirically it has been shown that as program length grows the **distribution of functionality reaches a limit**
- So, beyond a certain length, the proportion of programs which solve a problem is constant
- Since there are exponentially many more long programs than short ones, in GP

$$\frac{\text{size of the solution space}}{\text{size of the search space}} = \text{constant}$$

- Proofs?

Linear model of computer



States, inputs and outputs

- Assume n bits of memory
- There are 2^n states.
- At each time step the machine is in a state, s

Instructions

- Each instruction changes the state of the machine from a state s to a new s' , so instructions are maps from binary strings to binary strings of length n
- E.g. if $n = 2$, AND $m_0 m_1 \rightarrow m_0$ is represented as

m_0	m_1	m'_0	m'_1
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	1

=

0	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Behaviour of programs

- A program is a sequence of instructions
- So also the **behaviour of a program** can be described as a mapping from initial states s to corresponding final states s'

- For example,
 AND $m_0 m_1 \rightarrow m_0$
 NOP
 OR $m_0 m_1 \rightarrow m_0$
 AND $m_0 m_1 \rightarrow m_0$

→

m_0	m_1	m'_0	m'_1
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1

↓

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

July 2008 25

Does the behaviour tend to a limiting distribution?

Two primitives: AND $m0\ m1 \rightarrow m0$ OR $m0\ m1 \rightarrow m0$

Identity function (no instruction executed yet)

0 0 0 1 1 0 1 1

AND $m0\ m1 \rightarrow m0$ 1/2 1/2 OR $m0\ m1 \rightarrow m0$

0 0 0 1 0 0 1 1 0 0 1 1 1 0 1 1

A B

July 2008 26

0 0 0 1 0 0 1 1 A

AND $m0\ m1 \rightarrow m0$ 1/2 1/2 OR $m0\ m1 \rightarrow m0$

0 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1

A C

July 2008 27

0 0 1 1 1 0 1 1 B

AND $m0\ m1 \rightarrow m0$ 1/2 1/2 OR $m0\ m1 \rightarrow m0$

0 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1

C B

July 2008 28

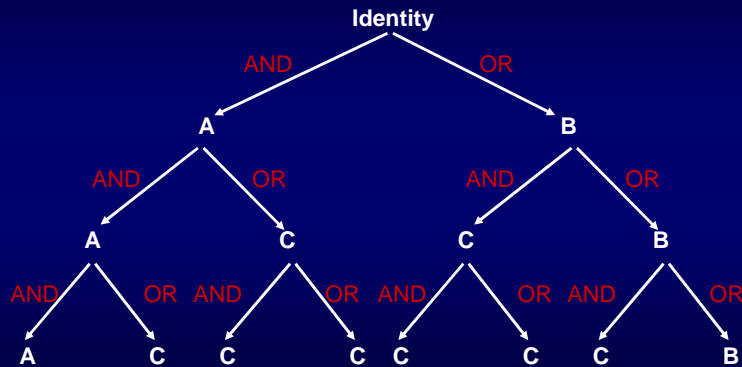
0 0 1 1 0 0 1 1 C

AND $m0\ m1 \rightarrow m0$ 1/2 1/2 OR $m0\ m1 \rightarrow m0$

0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1

C C

Probability tree



Distribution of behaviours

Program length	Behaviour A	Behaviour B	Behaviour C	Identity
0	0	0	0	1
1	$\frac{1}{2}$	$\frac{1}{2}$	0	0
2	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$	0
3	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{3}{4}$	0
4	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{7}{8}$	0
∞	0	0	1	0

Yes....

- ...for this primitive set **the distribution tends to a limit** where only behaviour **C** has non-zero probability.
- Programs in this search space tend to copy the initial value of m1 into m0.

Markov chain proofs of limiting distribution

- Using Markov chain theory WBL proved that a limiting distributions of functionality exists for a large variety of CPUs
- There are **extensions of the proofs** from linear to tree-based GP.
- See **Foundations of Genetic Programming** book for an introduction to the proof techniques.

So what?

- Generally **instructions lose information**. Unless inputs are protected, almost all long programs are constants.
- Write protecting inputs makes **linear GP more like tree GP**.
- No point searching above threshold?
- Predict where threshold is? Ad-hoc or theoretical.

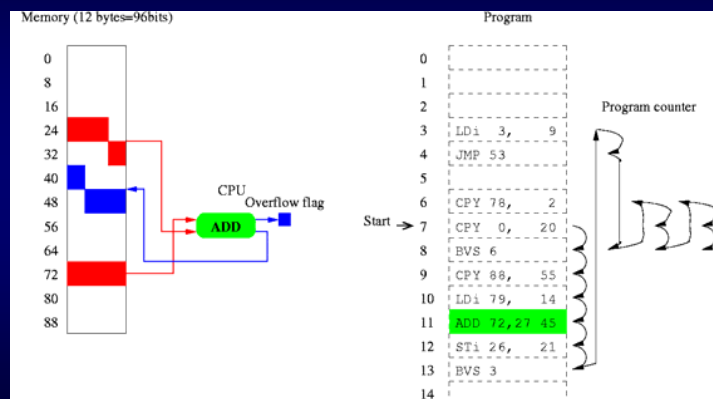
Implication of $|\text{solution space}|/|\text{search space}|=\text{constant}$

- GP can succeed if
 - the **constant** is not too small or
 - there is **structure** in the search space to guide the search or
 - the search operators are **biased** towards searching solution-rich areas of the search space
- or any combination of the above.

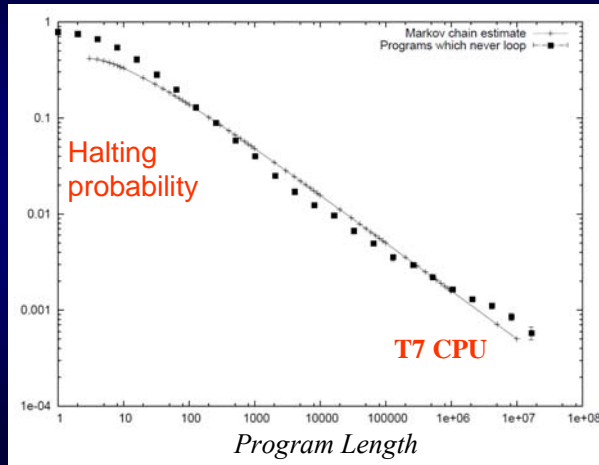
What about Turing complete GP?

- **Memory and loops make linear GP Turing complete**, but what is the effect search space and fitness?
- Does the **distribution of functionality** of Turing complete programs tend to a **limit** as programs get bigger?

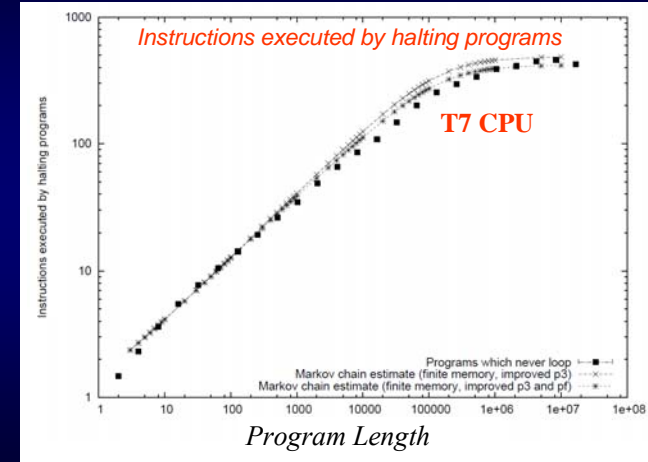
T7 Architecture



Almost all T7 Programs Loop



Halting programs have limited active code



T7 Turing complete GP cannot possibly work?

- If only halting programs can be solutions to problems, so
 $|\text{solution space}|/|\text{search space}| < p(\text{halt})$
- In T7, $p(\text{halt}) \rightarrow 0$, so,
 $|\text{solution space}|/|\text{search space}| \rightarrow 0$
- Since the search space is immense, GP with T7 seems to have no hope of finding solutions.

What can we do?

- Control $p(\text{halt})$
- Size population appropriately
- Design fitness functions which promote termination
- Repair
- Use result of program even if it is still running
-
- Any mix of the above

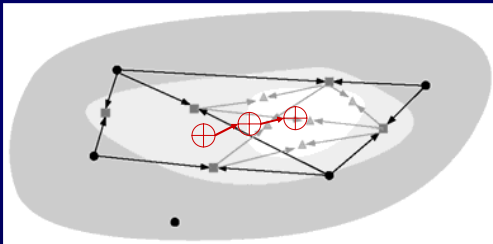
Limiting distribution of functionality for halting programs?

- The distribution of instructions in halting programs is the same as with a primitive set without jumps
- So, as the number of instructions executed grows, the **distribution of functionality of non-looping programs approaches a limit.**
- **Number of instructions executed, not program length**, tells us how close the distribution is to the limit

GP Search Characterisation

GA and GP search

- GAs and GP search like this:

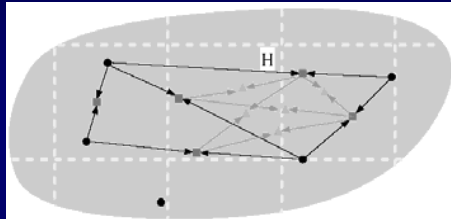


- How can we **understand** (characterise, study and predict) this search?

Schema Theories

- **Divide** the search space into **subspaces** (*schemata*)
- **Characterise** the schemata using *macroscopic quantities*
- **Model how and why** the individuals in the population **move** from one subspace to another (*schema theorems*).

Example



- The number of individuals in a given schema H at generation t , $m(H, t)$, is a good descriptor
- A *schema theorem* models mathematically how and why $m(H, t)$ varies from one generation to the next.

Exact Schema Theorems

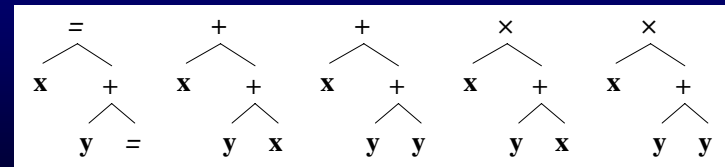
- The selection/crossover/mutation process is a random coin flip (Bernoulli trial). New individuals are either in schema H or not.
- So, $m(H, t+1)$ is a binomial stochastic variable.
- Given the success probability of each trial $\alpha(H, t)$, an **exact schema theorem** is

$$E[m(H, t+1)] = M \alpha(H, t)$$

Exact Schema Theory for GP with Subtree Crossover

GP Schemata

- Syntactically, a *GP schema* is a tree with some “don’t care” nodes (“=”) that represent exactly one primitive.
- Semantically, a *schema* is the set of all programs that match size, shape and defining nodes of such a tree.

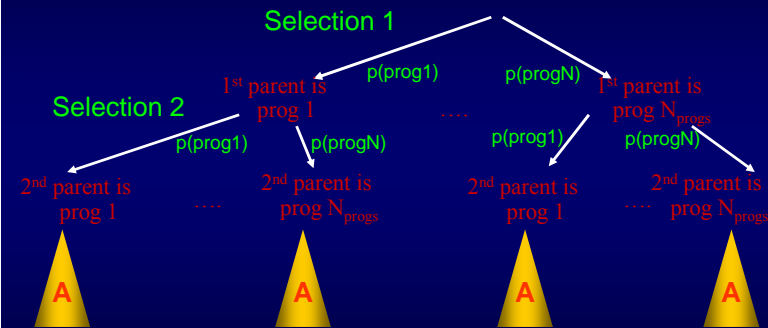


Creation of individuals via crossover is a compound event

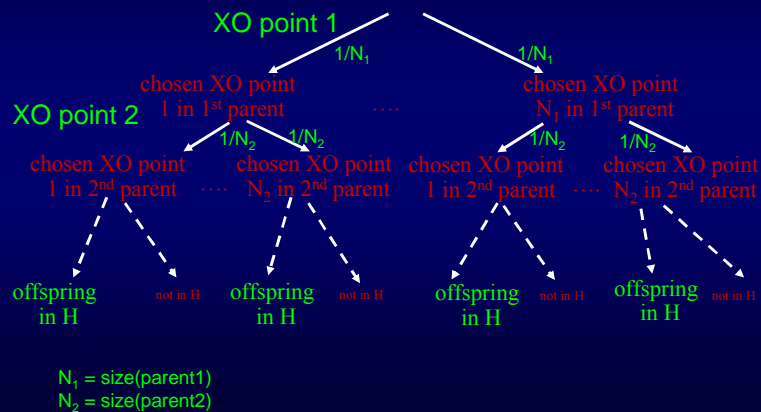
{create individual} =

{select parent 1,
select parent 2,
choose crossover point 1,
choose crossover point 2 }

Selection-Crossover Probability tree



Subtree



Microscopic schema model

$\alpha(H,t)$ = sum of products of probabilities
along paths leading to offspring in H

Problems:

- many paths \rightarrow many terms to evaluate (most=0)
- r.h.s. quantities are not about schemata
- model misses regularities in creation process

Can we do better?

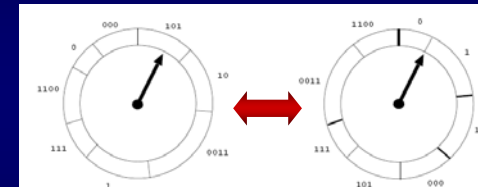
Regularities

- The process of **crossover point selection** is **independent** from the actual primitives in the parent tree.
- The **probability** of choosing a particular crossover point depends only on the actual **size and shape** of the parent.
- For example, the probability of choosing any crossover point in the program

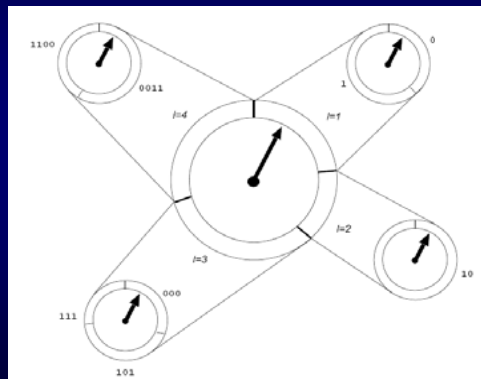
$$(+\ x\ (+\ y\ x))$$
 is identical to the probability of choosing any crossover point in

$$(AND\ D1\ (OR\ D1\ D2))$$

Fragmenting selection

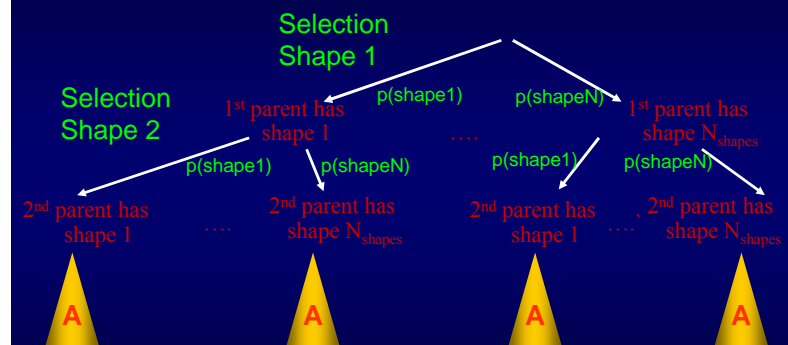


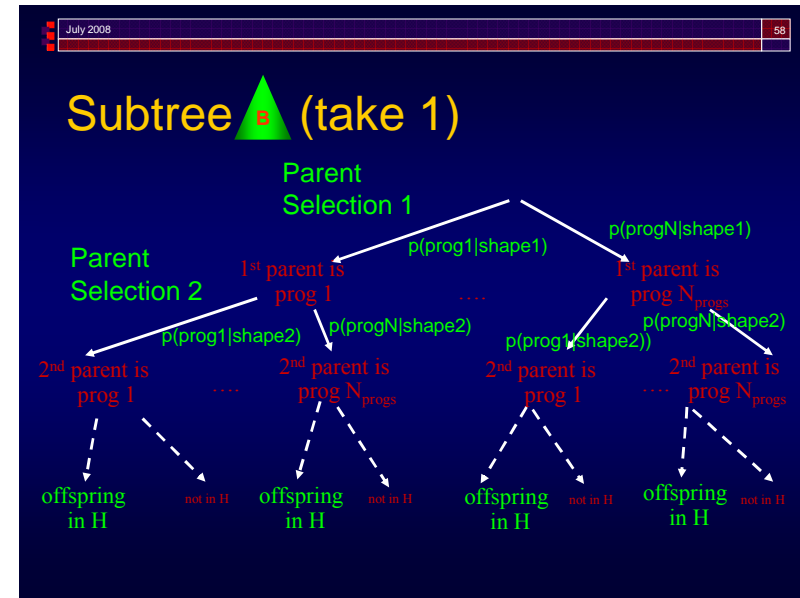
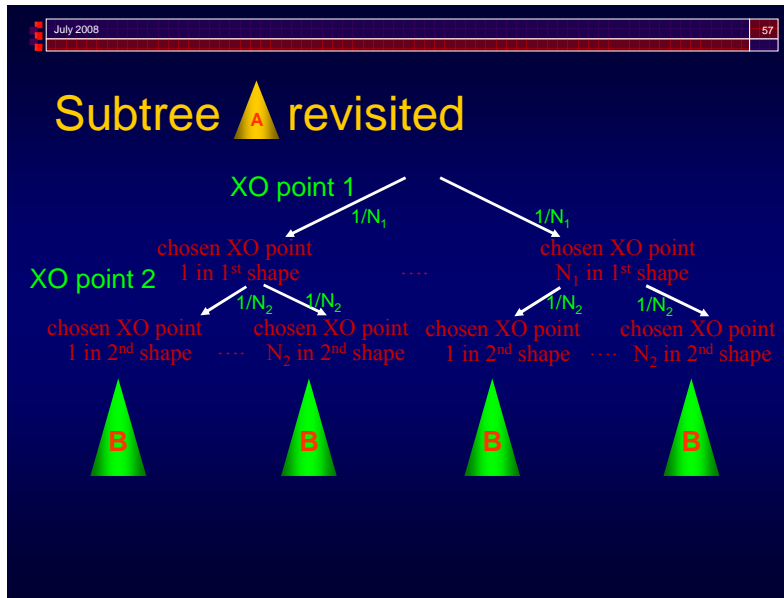
$\{\text{select parent}\} = \{\text{select size/shape, select individual of that size/shape}\}$



can be postponed

Selection-XO Probability Tree revisited

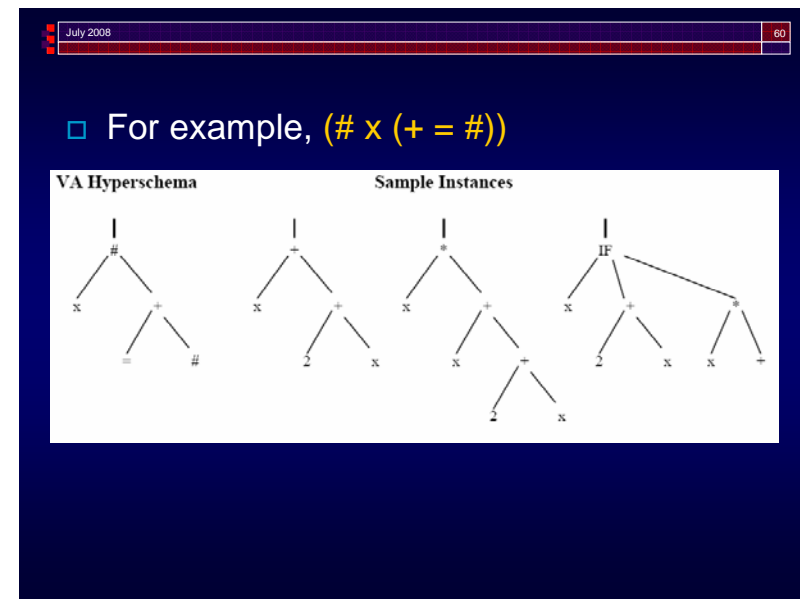




July 2008 59

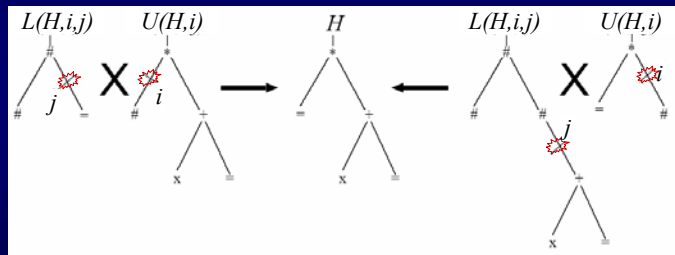
Variable Arity Hyperschemata

- A GP variable arity hyperschema is a tree with internal nodes from $F \cup \{=, \#\}$ and leaves from $T \cup \{=, \#\}$.
 - $=$ is a “don’t care” symbols which stands for exactly one node
 - $\#$ is a more general “don’t care” that represents either a valid subtree or a tree fragment depending on its arity



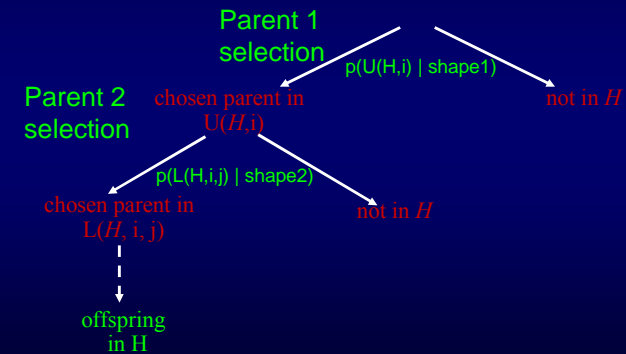
Upper and lower building blocks

Variable arity hyperschemata express which parents produce instances of a schema



Crossing over at points i and j any individual in $L(H,i,j)$ with any individual in $U(H,i) \rightarrow$ offspring in H

Subtree (take 2)



Bayes

$$p(U(H,i) | \text{shape1}) = \frac{p(U(H,i) \cap \text{shape1})}{p(\text{shape1})}$$

$$p(L(H,i,j) | \text{shape2}) = \frac{p(L(H,i,j) \cap \text{shape2})}{p(\text{shape2})}$$

Exact GP Schema Theorem for Subtree Crossover (2001)

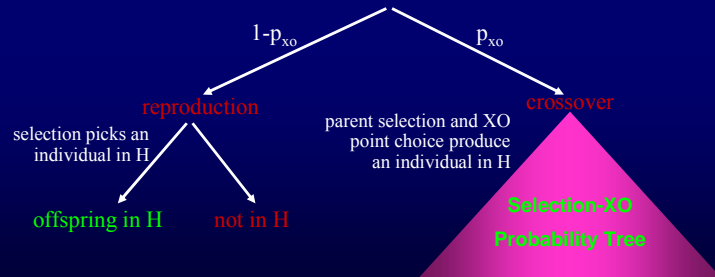
- Schema theorem for selection + 100% standard GP crossover

$$\alpha(H,t) = \frac{\sum_{k,t} N(G_k) N(G_l)}{\sum_{i \in H \cap G_k} \sum_{j \in G_l} p(U(H,t) \cap G_k, t) p(L(H,i,j) \cap G_l, t)}$$

shape1 shape2
XO points in shape1 XO points in shape2
size(shape1)= N_1 size(shape2)= N_2

To reproduce or not to reproduce ...

- Let us assume that also reproduction is performed.
- Creation probability tree for a schema H :



Exact GP Schema Theorem with Reproduction, Selection, Crossover

$$\alpha(H, t) = (1 - p_{xo})p(H, t) + p_{xo} \sum_{k,l} \frac{1}{N(G_k)N(G_l)} \sum_{i \in H \cap G_k} \sum_{j \in G_l} p(U(H, i) \cap G_k, t) p(L(H, i, j) \cap G_l, t)$$

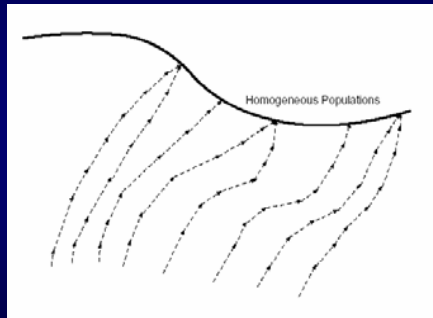
So what?

- A model is as good as the predictions and the understanding it can produce
- So, what can we learn from schema theorems?

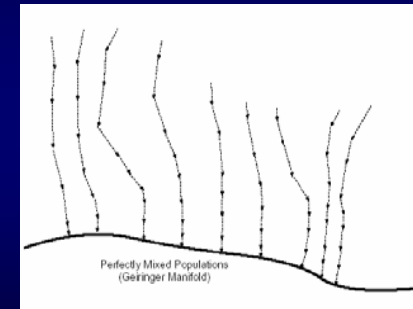
Lessons

- Operator biases
- Size evolution equation
- Bloat control
- Optimal parameter setting
- Optimal initialisation
- ...

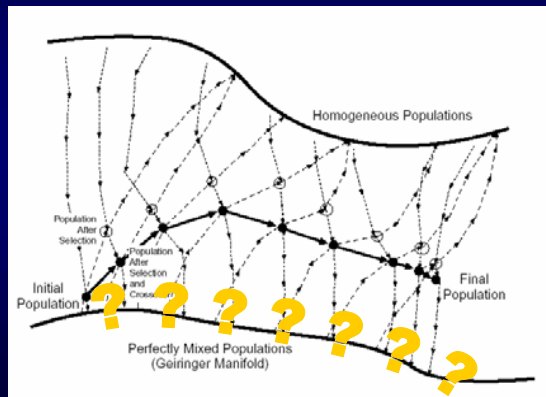
Selection Bias



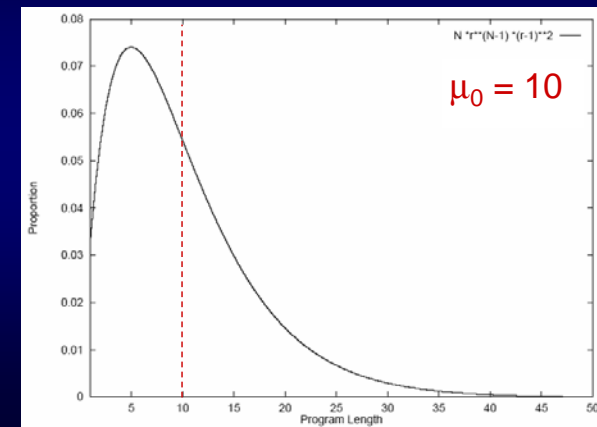
Crossover Bias



So where is evolution going?



Fixed-point of Schema Equations for Shapes (no fitness, linear GP)



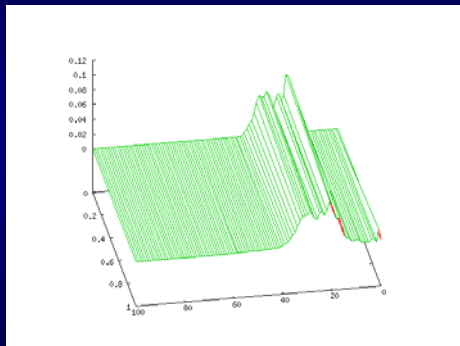
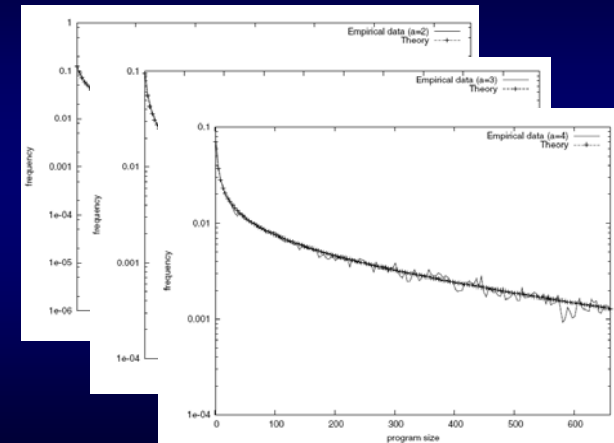
GP with subtree XO pushes the population towards a Lagrange distribution of the 2nd kind

$$\Pr\{n\} = (1 - ap_a) \binom{an+1}{n} (1 - p_a)^{(a-1)n+1} p_a^n$$

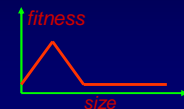
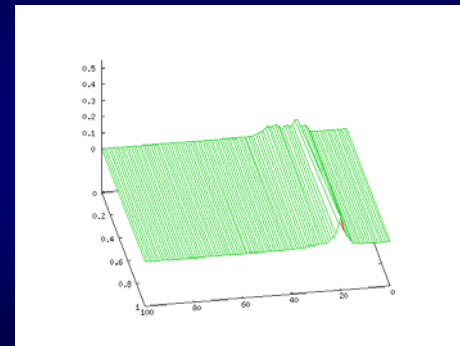
Proportion of programs with n internal nodes

Note: uniform selection of crossover points

□ Theory is right!



Can fitness override the XO bias?



Sampling probability under Lagrange

- Probability of sampling a particular program of size n under subtree crossover

$$p_{\text{sample}}(n) = \frac{(1 - ap_a)}{\mathcal{F}n \mathcal{T}^{(a-1)n+1}} \binom{an+1}{n} (1 - p_a)^{(a-1)n+1} p_a^n$$

- So, GP samples short programs much more often than long ones

Allele Diffusion

- The fixed-point distribution for linear, variable-length programs under GP subtree crossover is

$$\Phi(h_1 h_2 \dots h_N, \infty) = \Phi((=)^N, \infty) \times \prod_{i=1}^N c(h_i)$$

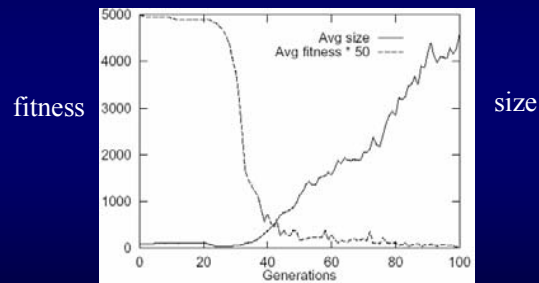
with

$$c(a) = \sum_{n \geq 0} \Phi((=)^n a, 0)$$

- Crossover attempts to push the population towards distributions of primitives where each primitive of a given arity is equally likely to be found in any position in any individual.
- The primitives in a particular individual tend not just to be swapped with those of other individuals in the population, but also to diffuse within the representation of each individual.
- Experiments with unary GP confirm the theory.

Bloat

Bloat



Ant Problem: Fitness

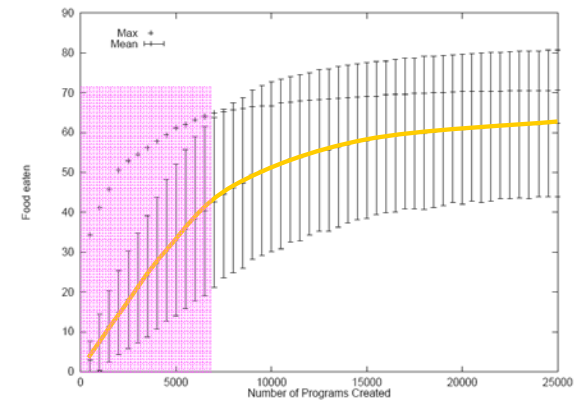


Figure 1: Evolution of maximum and population mean of food eaten. Error bars indicate one standard deviation. Means of 50 runs.

Ant Problem: Size

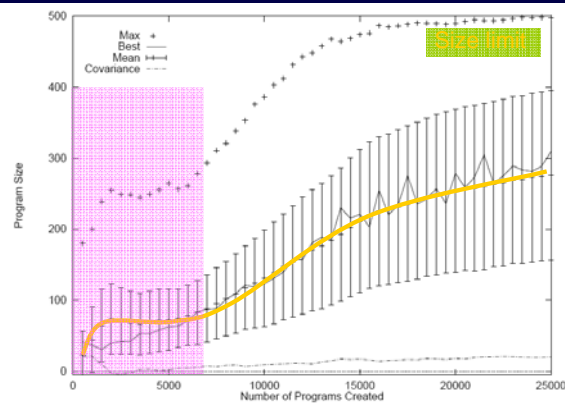
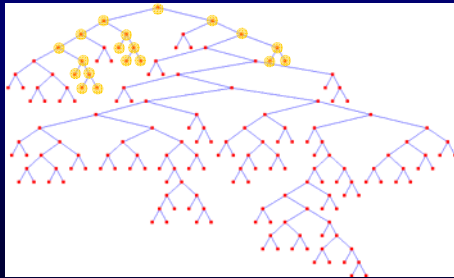


Figure 2: Evolution of maximum and population mean program length. Error bars indicate one standard deviation. Solid line is the length of the "best" program in the population, covariance of length and normalised rank based fitness shown dotted. Means of 50 runs.

WHY DOES IT HAPPEN?

Some evidence

- Most code in bloated programs is inactive.
- When a fit program is changed, it produces unfit offspring most of the time.
- Active code is like “icing on the cake”



Replication accuracy theory

- The success of a GP individual depends on its ability to have offspring that are functionally similar to the parent.
- So, GP evolves towards (bloated) representations that increase replication accuracy.

Removal bias theory

- Inactive code in a GP tree is low in the tree, forming smaller-than-average-size subtrees.
- Crossover events excising inactive subtrees produce offspring with the same fitness as their parents.
- On average the inserted subtree is bigger than the excised one, so such offspring are bigger than average.

More evidence: No Fitness = No Bloat

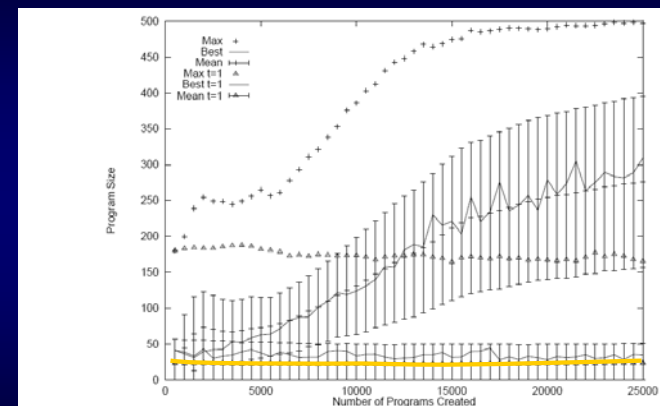


Figure 5: Evolution of maximum and population mean program length. Error bars indicate one standard deviation. Solid line is the length of the “best” program in the population. Means of 50 runs comparing tournament sizes of 7 and 1.

Nature of program search spaces theory

- Above a certain size, the distribution of fitnesses **does not vary with size**.
- Since there are **more long programs**, the number of long programs of a given fitness is greater than the number of short programs of the same fitness.
- Thus, over time **GP samples longer and longer programs** simply because there are more of them.

Size Evolution

- The *mean size* of the programs at generation t is

$$\mu(t) = \sum_l N(G_l) \Phi(G_l, t)$$

where





G_l = set of programs with shape l

$N(G_l)$ = number of nodes in programs in G_l

$\Phi(G_l, t)$ = proportion of population of shape l at generation t

- E.g., for the population:

x $(+ x y)$ $(- y x)$ $(+ (+ x y) 3)$

l	G_l	$N(G_l)$	$\Phi(G_l, t)$
1		1	1/4
2		3	2/4
3		5	1/4
4		5	0
⋮	⋮	⋮	⋮

$$\mu(t) = 1 \times \frac{1}{4} + 3 \times \frac{2}{4} + 5 \times \frac{1}{4} = 3$$

Size Evolution under Subtree XO

- In a GP system with symmetric subtree crossover

$$E[\mu(t+1)] = \sum_l N(G_l) p(G_l, t)$$

where

$p(G_l, t)$ = probability of selecting a program of shape l from the population at generation t

- The mean **program size evolves as if selection only was acting** on the population

Conditions for Growth

- Growth can happen only if

$$E[\mu(t+1) - \mu(t)] > 0$$

- Or equivalently

$$\sum_i N(G_i) [p(G_i, t) - \Phi(G_i, t)] > 0$$

- Or

$$\sum_{\ell} \ell (p(\ell, t) - \Phi(\ell, t)) > 0$$

- Size evolution eq. = Price's theorem

$$E[\Delta\mu] = \frac{\text{Cov}(\ell, f)}{\bar{f}(t)}$$

Problem for drift in program space hypothesis

- All very big programs sample same fitness distribution, so....
- Price's theorem applied to size says

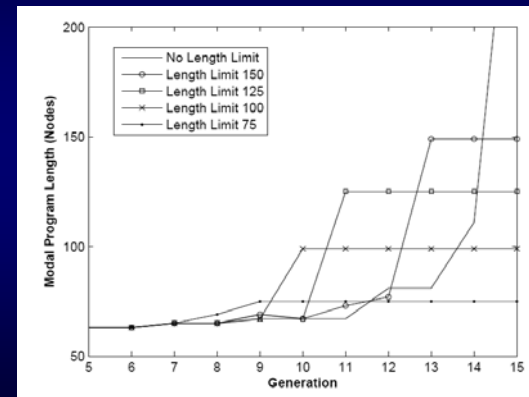
$$E[\Delta\mu] = \frac{\text{Cov}(\ell, f)}{\bar{f}(t)} = 0$$

i.e., no bloat for long programs

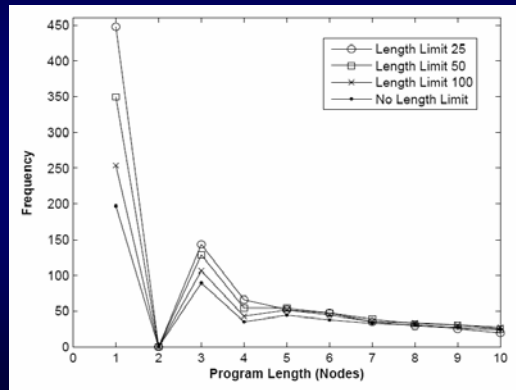
Crossover Bias Theory of Bloat

1. Crossover does not change the mean program size, on average, but...
2. It creates a population of individuals with a large proportion of small programs.
3. In most problems, very small programs are unfit, so they are ignored by selection. Thus...
4. Only larger programs will be picked as parents, hence increasing mean program size.

Size Limits Considered Harmful



Sampling Frequencies



Popsizes causes bloat

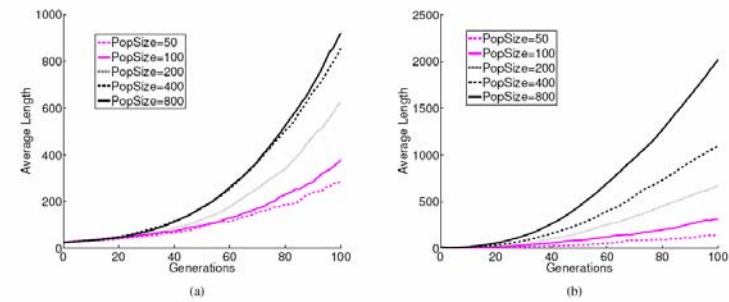


Figure 3: Average length of the individuals in the population against generations. Results are averages over 100 independent GP runs. (a): Artificial Ant on the Santa Fe trail. (b): Symbolic regression.

Are short programs unfit? Random Search

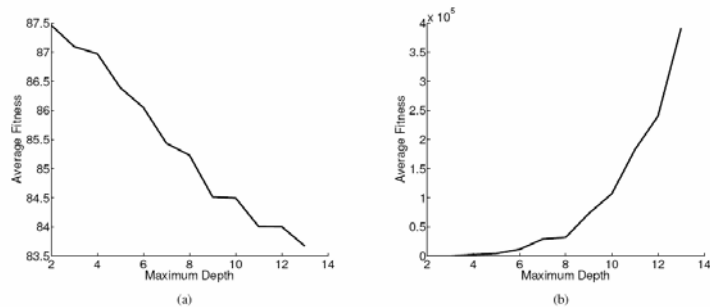


Figure 1: Average fitness vs. maximum depth over a sample of 4000 individuals generated with the Ramped Half and Half algorithm. All problems use minimization (the smaller the fitness, the better). (a): Artificial Ant on the Santa Fe trail. (b): Symbolic regression.

Are short programs unfit? Metropolis-Hastings

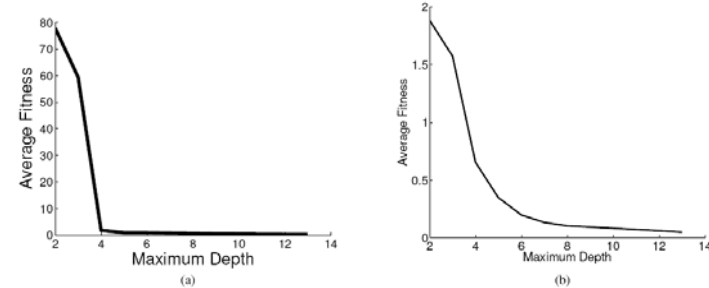
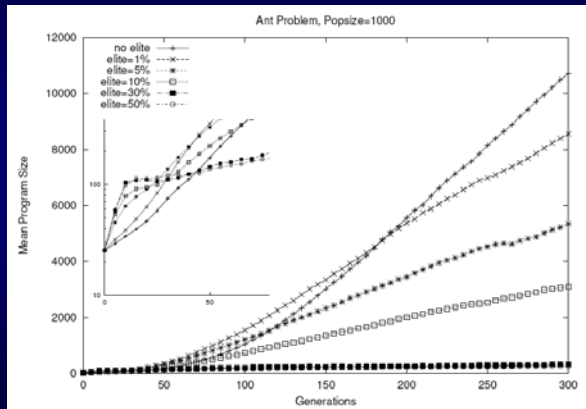


Figure 2: Average fitness vs. maximum depth over a sample of 4000 individuals generated with the Metropolis-Hastings algorithm (see the text for a definition of the acceptance criterion used). All problems use minimization (the smaller the fitness, the better). (a): Artificial Ant on the Santa Fe trail. (b): Symbolic regression.

Elitism slows down bloat



Revised Crossover Bias Theory

- Crossover bias causes sampling of short programs
- Short programs are less fit than long ones if you search for long enough based on fitness
- Whenever short unfit programs are sampled long programs have a selective advantage → bloat
- Anything that modifies the tail of the sampling distribution may have an effect on bloat: e.g. mean program size, arity of primitives, size limits, elitism.
- In finite populations the frequency of sampling short unfit programs varies with population size and mean program size. So, bloat is modulated by these.

Main techniques for limiting code bloat

- *Fixed size or depth limit*: Programs exceeding the limit are discarded and a parent is kept instead.
- This is very dangerous as it gives a fitness advantage to programs that tend to violate the constraint.

- *Parsimony pressure*: a term is added to the fitness function that penalises larger programs.
- Typically:

$$f_{\text{parsimony}}(\text{prog}) = f_{\text{raw}}(\text{prog}) - c * \text{size}(\text{prog})$$
 where c is a constant.

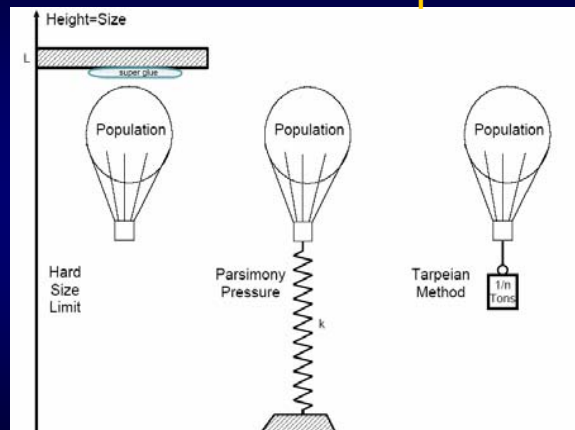
- *Modification of operators*: variation of the selection probability of crossover points by using explicitly defined introns, rejection of destructive crossover events, size-fair operators, MOO techniques, etc.
- For example, *point mutation* (applied with a fixed probability *per node*) has an **anti bloat effect**.

Tarpeian Bloat Prevention

- To **prevent growth** one needs
 - To **increase** the selection probability for **below-average-size programs**
 - To **decrease** the selection probability for **above-average-size programs**



Hot Air Balloon Metaphor



Covariant parsimony pressure

- Parsimonious fitness

$$f_p(x, t) = f(x) - g(\ell(x), t)$$

- Price:

$$E[\Delta\mu] = \frac{\text{Cov}(\ell, f_p)}{\bar{f}_p} = \frac{\text{Cov}(\ell, f) - \text{Cov}(\ell, g)}{\bar{f} - \bar{g}}$$

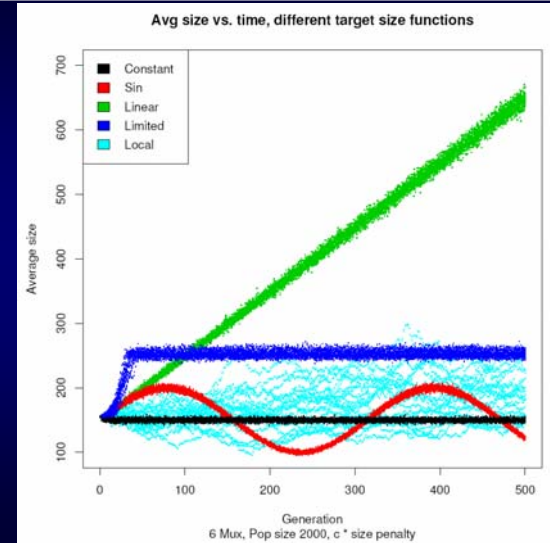
- Assume $g(\ell(x), t) = c(t)\ell(x)$

- No bloat if $c(t) = \frac{\text{Cov}(\ell, f)}{\text{Var}(\ell)}$

□ Target dynamics $\gamma(t)$

□ Power parsimony $g(\ell(x), t) = c(t)\ell(x)^k$

$$c(t) = \frac{\text{Cov}(\ell, f) - (\gamma(t+1) - \mu(t))\bar{f}}{\text{Cov}(\ell, \ell^k) - (\gamma(t+1) - \mu(t))E[\ell^k]}$$



Conclusions

Theory

- In the last few years the theory of GP has seen a formidable development.
- Today we understand a lot more about the nature of the GP search space and the distribution of fitness in it.
- Also, schema theories explain and predict the syntactic behaviour of GAs and GP.
- We know much more as to where evolution is going, why and how.

- Theory primarily provides explanations, but **many recipes for practice** have also been derived
- So, theory can **help design competent algorithms**
- Theory is hard and slow: **empirical studies are important** to direct theory and to corroborate it.