The Genetic and Evolutionary Computation Conference

Handling Bloat in GP

Sara Silva

INESC-ID Lisboa, Portugal
CISUC, Coimbra, Portugal

sara@kdbio.inesc-id.pt
sara@dei.uc.pt

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

Instructor Biography

- BSc and MSc in Informatics
  - Faculty of Sciences of the University of Lisbon, Portugal
- PhD in Informatics Engineering
  - Faculty of Sciences and Technology of the University of Coimbra, Portugal
- Research using different methodologies
  - Neural Networks
  - Genetic Algorithms
  - Genetic Programming
- Participation in several interdisciplinary projects
  - Remote Sensing and Forest Science
  - Epidemiology and Biomedical Informatics
- Main research interests in GP
  - Bloat, Overfitting and Complexity
  - Making GP a user-friendly tool for solving real life applications

Agenda

Bloat
- History and Definition
- Theories and Methods

Crossover Bias

Operator Equalisation
- Idea and Initial Results

DynOpEq
- How Does It Work
- Benchmark Results
- Efficiency Problems

break

MutOpEq
- Differences from DynOpEq
- Benchmark Results

Real World Results
- Drug Discovery
- Remote Sensing

Open Questions
- Pros and Cons of OpEq options
- Bloat vs Overfitting vs Complexity

Future
- Improvements and Extensions
- The next chapter on OpEq

Objectives of the Tutorial

For the participants
- Get an overview of the past bloat research
- Get acquainted with Crossover Bias
- Learn how to implement Operator Equalisation (OpEq)
- Understand the implications of different options
- Think about the open questions
- Think about possible improvements

For the presenter
- Gather ideas,
  suggestions,
  and criticisms
  from the participants!

Bloat
Early History

1992 John Koza: edited the final solutions to remove pieces of redundant code; imposed a depth limit of 17 on the trees created by crossover

1994 Peter Angeline: adopted the name *introns*; noted they provided neutral points for crossover; based on their importance for genetic algorithms, remarked that

> "**it is important then to not impede this emergent property as it may be crucial to the successful development of genetic programs**"

5

Bloat
Pros and Cons

Pros Code compression and parsimony (effective code is shorter! why?)
Protection against genetic operators (but is it really useful?)
Artificial introns beneficial to linear GP (but not tree-based GP)

Cons Exhaustion of computational resources
(storage, evaluation and swapping of useless code)
Stagnation of effective search
Poor readability of the solutions

6

Bloat
Definition

Excessive code growth without a corresponding improvement in fitness

A measure to **quantify bloat** was recently introduced in: "Measuring Bloat, Overfitting and Functional Complexity in Genetic Programming" by Vanneschi, Castelli, Silva **GECCO-2010**

Bloat is not specific to GP.
It affects all progressive search techniques based on variable-length representations and using static evaluation functions

7

Bloat
Theories

Hitchhiking, 1994 (Tackett) Based on genetic algorithms, where unfit building blocks propagate in the population because they join highly fit building blocks. Introns in GP propagate because they are hitchhikers.

Defense Against Crossover, 1994-1998 (Altenberg, Blickle and Thiele, McPhee and Miller, Nordin and Banzhaf, Smith and Harries, Soule, etc) Genetic operators seldom create better individuals than their parents. Offspring who have the same fitness as their parents have a selective advantage. Introns provide code where changes will not affect fitness.

Developed mostly in the context of linear GP. Difference between inviable code and unoptimized code is important in tree-based GP.

Alternative names: Replication Accuracy Theory, Intron Theory, Protection Theory.

8

Bloat
Theories

Removal Bias, 1998 (Soule and Foster) Based on defense against crossover. To maintain fitness the removed branch must be contained within the inviable region, while the inserted branch can have any size.

Fitness Causes Bloat, 1998 (Langdon and Poli) The first theory not blaming introns as the cause of bloat. Because genetic operators are destructive, maintaining fitness is advantageous. There are many more longer ways than shorter ways to represent the same program, so a natural drift occurs to longer programs.

Alternative names: Solution Distribution, Diffusion Theory, Drift, Nature of Search Spaces, Entropy Random Walk.

→ **Beyond a certain program length, the distribution of fitness converges to a limit, 2002 (Langdon and Poli)**

9

Bloat
Theories

Modification Point Depth, 2003 (Luke) When a genetic operator modifies an individual, the deeper the modification point the smaller the change in fitness. Small changes are less likely to be disruptive, so there is a preference for deeper modification points, and consequently a preference for larger trees.

Alternative names: Depth-correlation, or Depth-based Theory.

- All these theories "make sense"
- If you remove the search for fitness the reasons for bloat disappear (if selection is random, there is no code growth with "normal" genetic operators)
- But we cannot avoid the search for fitness!

10

Bloat
Theories

Crossover Bias, 2007 (Dignum and Poli)

Most genetic operators, in particular standard subtree crossover, do not add or remove any amount of genetic code from the population, they simply swap it between individuals. So the average program length in the population is not changed by crossover.

There is a bias of many genetic operators, in particular crossover, to create many small, and consequently unfit, individuals.

→ **When these small unfit individuals are engaged in competition for breeding, they are always discarded by selection in favor of the larger ones.** This is what increases the average program length.

11

Bloat
Methods

Bloat control is possible at different levels of the evolutionary process:

Evaluation
Parametric Parsimony Pressure, Tarpeian

Selection
Multi-Objective Optimization, Special Tournaments

Breeding
Special Genetic Operators

Survival
Size/Depth Limits, Operator Equalisation (size=length)

Others
Code Editing, Dynamic Fitness, Other Types of GP

12

Bloat

Methods

Evaluation

Parametric Parsimony Pressure

The fitness of an individual is a function of its raw fitness and its size/length, penalizing larger individuals. Some techniques apply adaptive pressure.

Pros

Can speed the evolution and produce very compact solutions

Cons

Tends to converge on local optima
Very dependent on parameters
(which depend on the problem and on the stage of the evolution)

13

Bloat

Methods

Selection

Special Tournaments – Double Tournament

The winners of a first tournament are engaged in a second tournament. The first is based on fitness and the second on size, or vice versa. In the size tournament the smaller individual wins with probability *D*. ($0.5 < D < 1$)

Pros

One of the best methods until recently

Cons

Difficult to find correct setting for *D*
(same problem as with parametric parsimony pressure)

14

Bloat

Methods

Breeding

Special Genetic Operators – Homologous Crossover

Selects the crossover node on the first parent randomly, like in standard subtree crossover. Selects the crossover node on the second parent so that the swapped nodes are similar in structure and position in the tree.

Pros

Effectively controls bloat

Cons

Weak exploration of the search space
Requires a larger population and larger initial individuals
Requires mutation

15

Bloat

Methods

Survival

Size/Depth Limits – Fixed Limits

Whenever crossover creates an individual that breaks the fixed predetermined size/depth limit, the individual is rejected and 1) one of its parents is accepted instead, 2) crossover is repeated with the same parents, or 3) crossover is repeated with new parents.

Pros

Effectively prevents bloat beyond a certain point

Cons

The fixed limit is arbitrary
Option 1 actually speeds bloat until the limit is reached

16

Bloat
Methods

Survival

Size/Depth Limits – Dynamic Limits

Works like the Fixed Limits, except that the limit is not static. The initial limit is set to a very low value, and only increased whenever that is needed to accept a new best-of-run individual.

Pros

Does not allow code growth unless it is necessary
Allows enough code growth to solve very complex problems

Cons

For some problem types bloat still happens
(typically very hard regression problems)

17

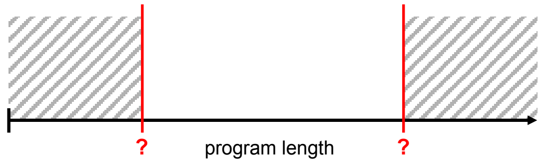
Bloat
Methods

Survival

Operator Equalisation

Beyond a certain program length, the distribution of fitness converges to a limit (Langdon and Poli, 2002)

Many small unfit individuals are created by crossover, and then discarded by selection in favor of the larger ones (Dignum and Poli, 2007)



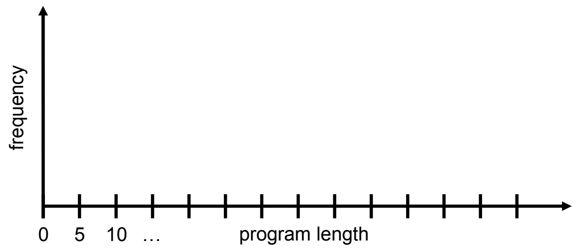
18

Operator Equalisation
initial idea

Program Length

~~Operator~~ Equalisation
(Dignum and Poli, 2008)

Control the distribution of program lengths inside the population, biasing the search towards the desired lengths.

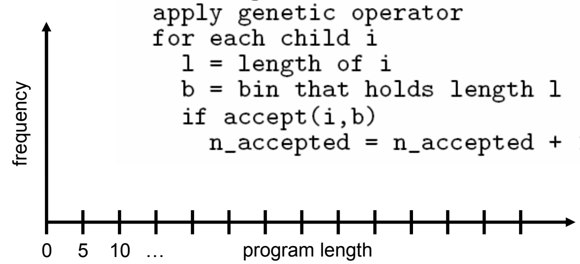


19

Operator Equalisation
initial idea

generation of a new population:

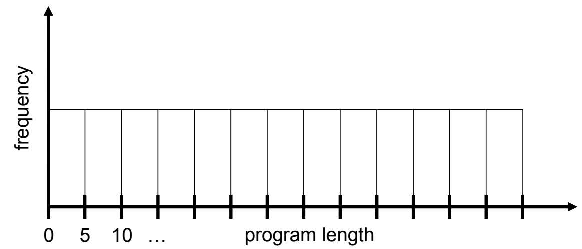
```
n_accepted = 0
while n_accepted < pop_size
  select parents
  apply genetic operator
  for each child i
    l = length of i
    b = bin that holds length l
    if accept(i,b)
      n_accepted = n_accepted + 1
```



20

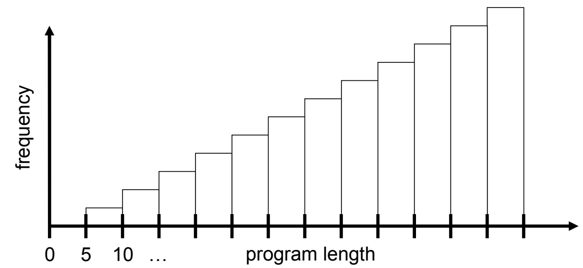
Operator Equalisation
initial idea

You can choose any target distribution…



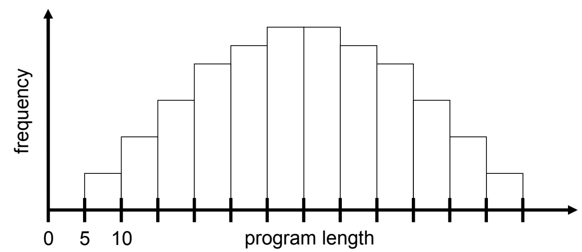
Operator Equalisation
initial idea

You can choose any target distribution…



Operator Equalisation
initial idea

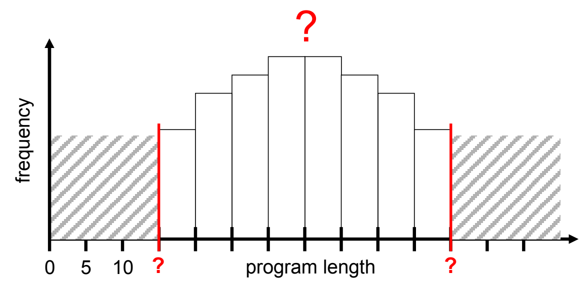
You can choose any target distribution…



Operator Equalisation
initial idea

Limitations of the initial idea:

- **Fixed number of bins**
- **Fixed predetermined target distribution**

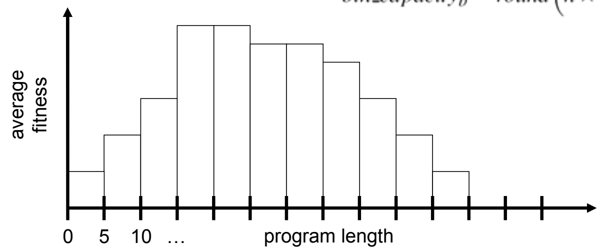


DynOpEq
calculating the target

DynOpEq (Dynamic Operator Equalisation)
(Silva and Dignum, 2009)

Target is fitness proportional
- Based on the average bin fitness on the previous generation

$$bin\_capacity_b = round\left(n \times (\bar{f}_b / \sum_i \bar{f}_i)\right)$$



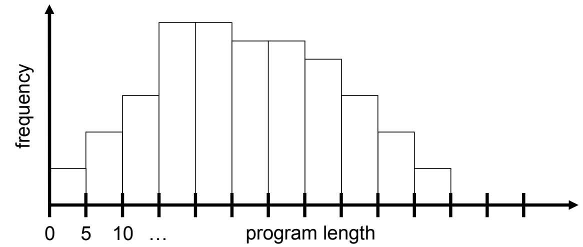
25

DynOpEq
calculating the target

DynOpEq (Dynamic Operator Equalisation)
(Silva and Dignum, 2009)

Target is dynamic
- Self adapted every generation

Example
Symbolic Regression
Generation 5



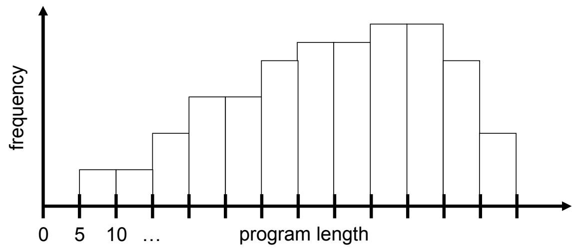
26

DynOpEq
calculating the target

DynOpEq (Dynamic Operator Equalisation)
(Silva and Dignum, 2009)

Target is dynamic
- Self adapted every generation
- Variable number of bins

Example
Symbolic Regression
Generation 25

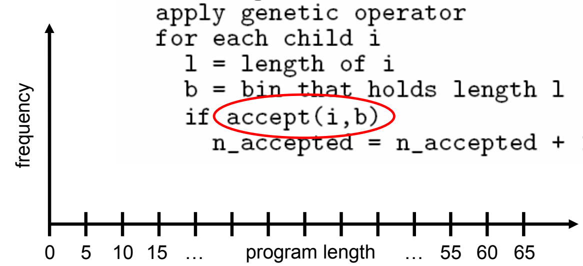


27

DynOpEq
following the target

Example – generation of a new population:

```
n_accepted = 0
while n_accepted < pop_size
  select parents
  apply genetic operator
  for each child i
    l = length of i
    b = bin that holds length l
    if accept(i,b)
      n_accepted = n_accepted + 1
```



28

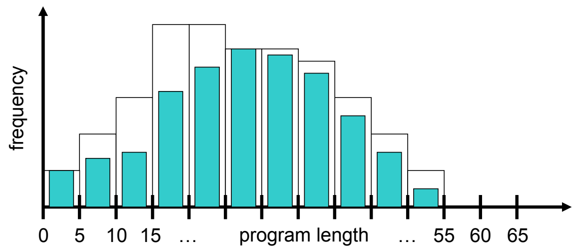
DynOpEq
following the target

Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: (5 example cases)

target
occupied



29

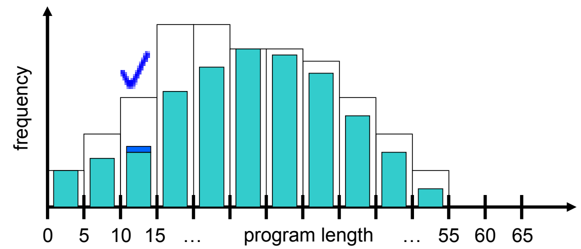
DynOpEq
following the target

Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: CASE 1 – length 12, fitness 50

target
occupied



30

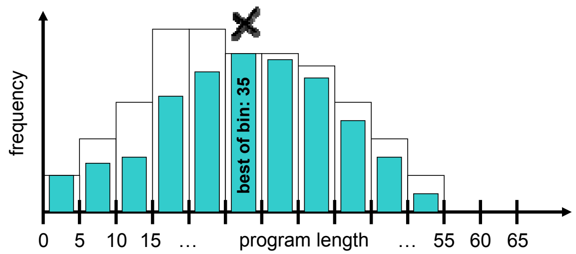
DynOpEq
following the target

Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: CASE 2 – length 28, fitness 40

target
occupied



31

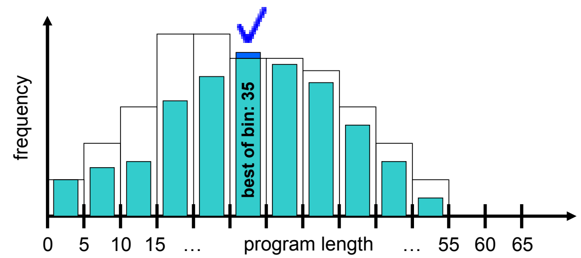
DynOpEq
following the target

Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: CASE 3 – length 28, fitness 33

target
occupied



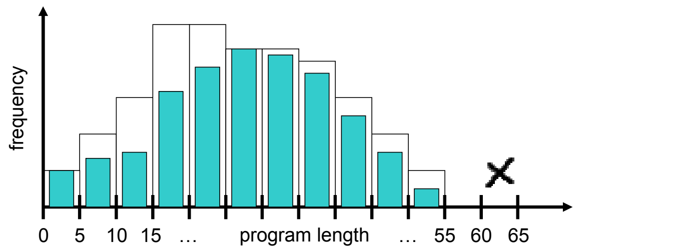
32

DynOpEq
following the target

Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: CASE 4 – length 63, fitness 30

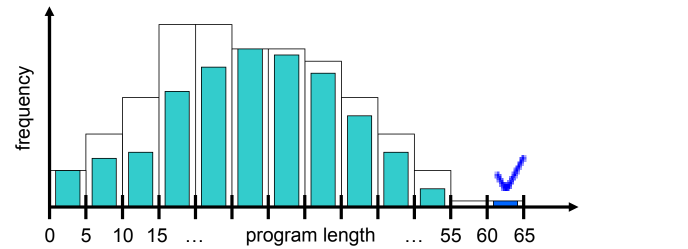


DynOpEq
following the target

Example – generation of a new population

Best fitness so far: 28 (lower is better)

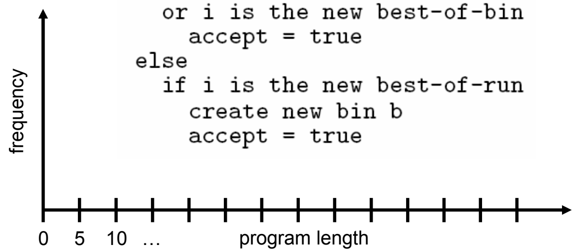
New individual: CASE 5 – length 63, fitness 28



DynOpEq
following the target

accept(individual i, bin b):

```
accept = false
if b exists
  if b is not full
  or i is the new best-of-bin
    accept = true
else
  if i is the new best-of-run
    create new bin b
    accept = true
```



DynOpEq
benchmark results - experiments

4 problems
- Symbolic Regression
- Artificial Ant
- 5-bit Even Parity
- 11-bit Multiplexer

3 bin widths
- 1, 5, 10

6 techniques
- No Limits
- Koza Max Depth 17
- Dynamic Limits (Depth)
- (Dyn)OpEq 1
- (Dyn)OpEq 5
- (Dyn)OpEq 10

1000 individuals
50 generations
30 runs

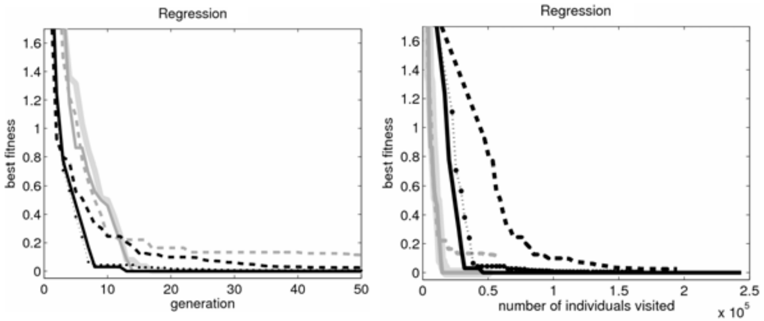
DynOpEq
benchmark results – 3 flavors

Results presented in 3 different ways:

- Optimistic view
- Pessimistic view
- Realistic view

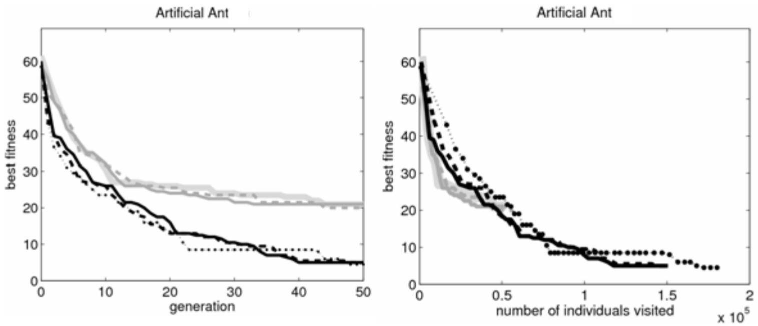
No Limits
Koza
Dyn Depth
Op Eq 1
Op Eq 5
Op Eq 10

DynOpEq
benchmark results – optimistic / pessimistic



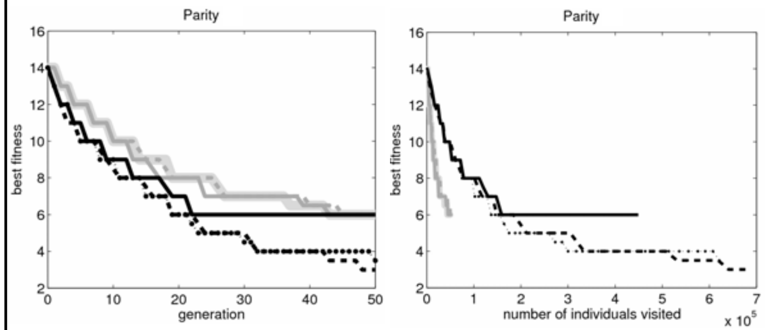
No Limits Koza Dyn Depth Op Eq 1 Op Eq 5 Op Eq 10

DynOpEq
benchmark results – optimistic / pessimistic



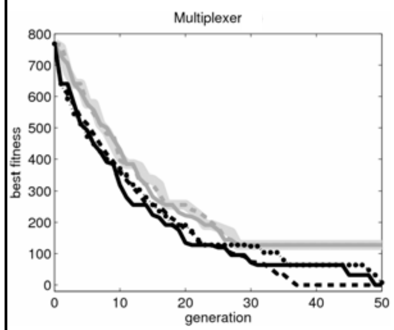
No Limits Koza Dyn Depth Op Eq 1 Op Eq 5 Op Eq 10

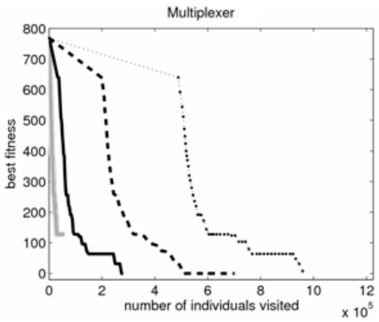
DynOpEq
benchmark results – optimistic / pessimistic



No Limits Koza Dyn Depth Op Eq 1 Op Eq 5 Op Eq 10

DynOpEq
benchmark results – optimistic / pessimistic





No Limits — Koza — Dyn Depth — Op Eq 1 — Op Eq 5 — Op Eq 10

DynOpEq

Very high number of rejections

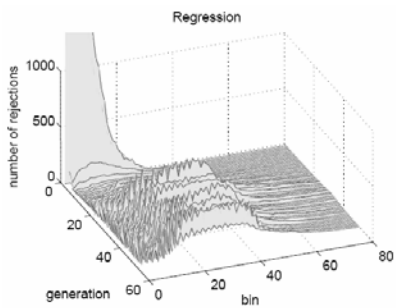
=

Efficiency problem

DynOpEq
rejections

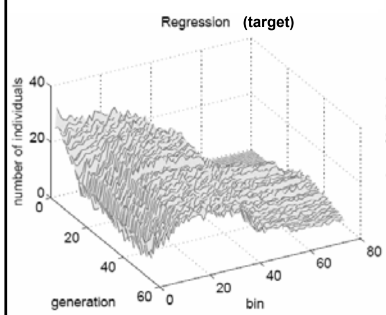
Regression:
<u>Bin width 1</u>
16% of rejections happen in the first 5 generations to individuals of length 1-10
<u>Bin width 5,10</u>
75-76%

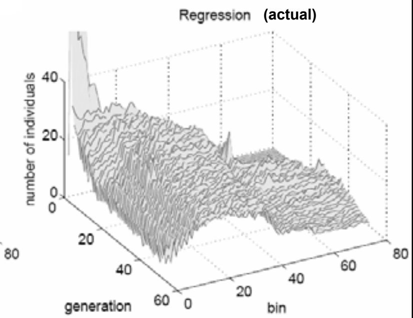
Multiplexer:
10%, 23%, 40%
Artificial Ant
and **Parity: <2%**



DynOpEq
target / actual length distribution

Many rejections occur to small individuals, in the beginning of the run…
… and they mostly occur within the target distribution





DynOpEq
rejections

Rejections falling outside the target

	Bin width		
	1	5	10
Regression	7.6%	0.5%	0.003%
Artificial Ant	5.0%	4.1%	4.6%
Parity	13.1%	4.9%	4.0%
Multiplexer	3.8%	2.8%	1.8%

Possible efficiency improvement:

- Evaluate only individuals falling outside the target
- Why this can be a problem

45

DynOpEq
benchmark results – realistic view

(regardless of the efficiency problem)

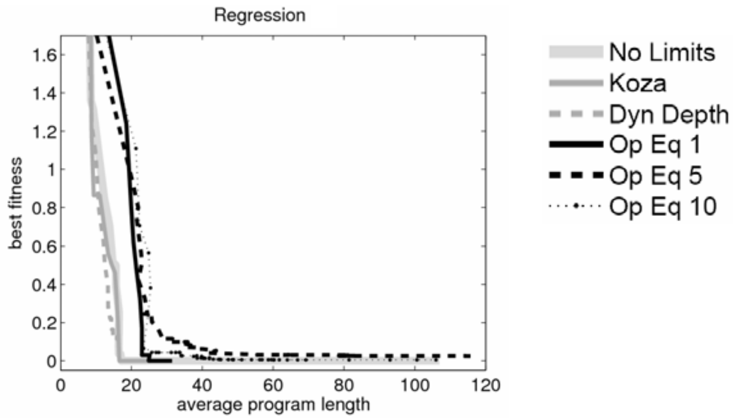
Most end users of GP want
simple and accurate solutions
regardless of how long it takes

The relationship between fitness and
program length is important!

This is what bloat is about

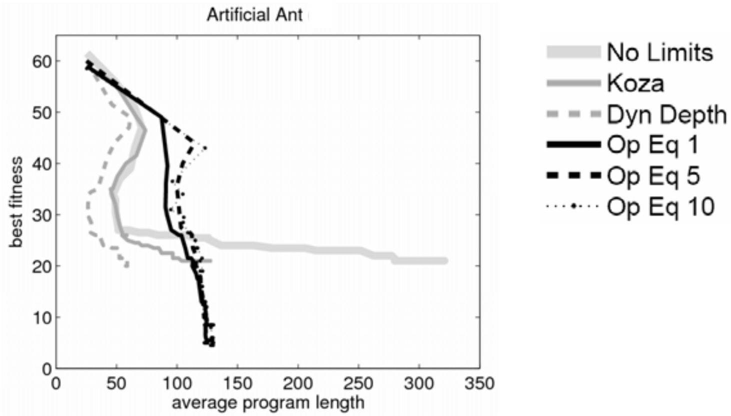
46

DynOpEq
benchmark results – realistic view



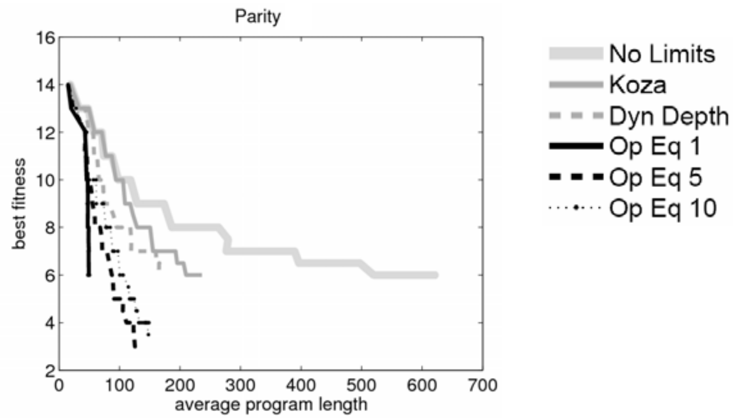
47

DynOpEq
benchmark results – realistic view



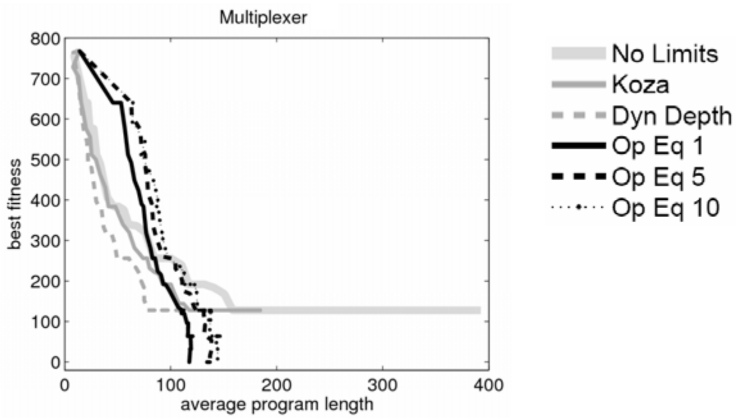
48

DynOpEq
benchmark results – realistic view



49

DynOpEq
benchmark results – realistic view



50

Operator Equalisation
summary

Operator Equalisation (Dignum and Poli, 2008)

- **Fixed predetermined target distribution**
- **Fixed number of bins**

Operator Equalisation (Silva and Dignum, 2009)

- **Dynamic self-adaptive target distribution**
- **Variable number of bins**

DynOpEq

51

MutOpEq

MutOpEq (Mutation-Based Dynamic Operator Equalisation)
(Silva and Vanneschi, 2009)

To avoid the efficiency problem, individuals are not rejected. Instead, they are mutated if necessary to fit the target.

The deeper the modification point the smaller the change in fitness.

Soft mutations:

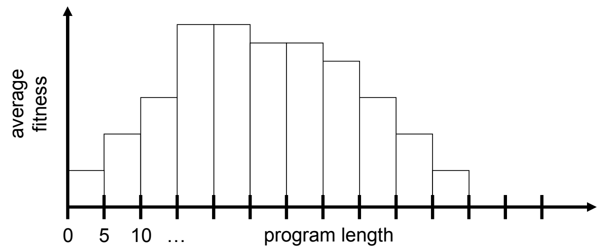
Shrink Choose a terminal branch and replace it with one of its terminal nodes (terminal branch = subtree of minimum depth, but not terminal)

Grow Choose a terminal node and replace it with a terminal branch where one of the arguments is the terminal node

52

MutOpEq
following the target

MutOpEq (Mutation-Based Dynamic Operator Equalisation)
(Silva and Vanneschi, 2009)



53

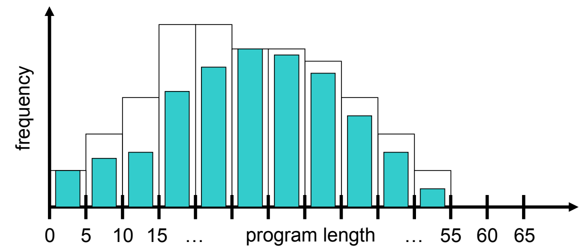
MutOpEq
following the target

Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: (5 example cases)

target
occupied



54

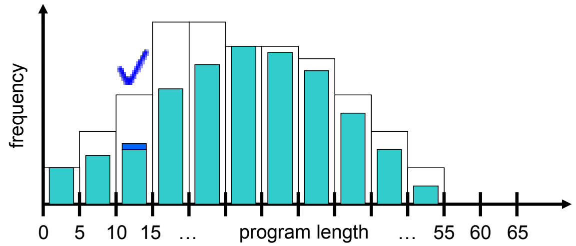
MutOpEq
following the target

Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: CASE 1 – length 12, fitness 50

target
occupied



55

MutOpEq
following the target

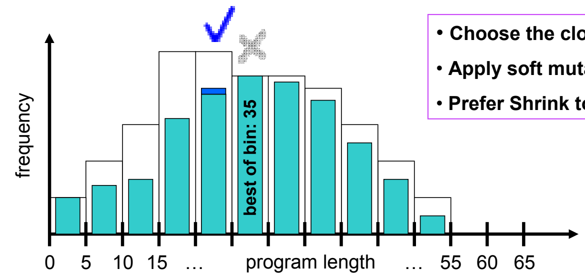
Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: CASE 2 – length 28, fitness 40

target
occupied

- **Choose the closest bin**
- **Apply soft mutations**
- **Prefer Shrink to Grow**



56

MutOpEq
following the target

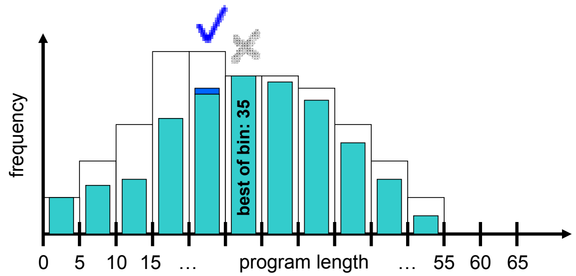
Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: CASE 3 – length 28, fitness 33

target

occupied



MutOpEq
following the target

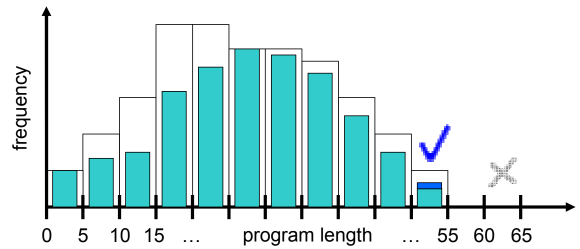
Example – generation of a new population

Best fitness so far: 30 (lower is better)

New individual: CASE 4 – length 63, fitness 30

target

occupied



MutOpEq
following the target

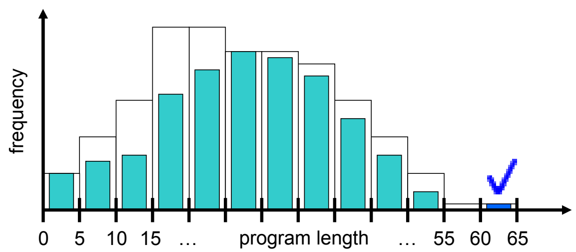
Example – generation of a new population

Best fitness so far: 28 (lower is better)

New individual: CASE 5 – length 63, fitness 28

target

occupied



Operator Equalisation
summary

Operator Equalisation (Dignum and Poli, 2008)

- **Fixed predetermined target distribution**
- **Fixed number of bins**

Operator Equalisation (Silva and Dignum, 2009)

- **Dynamic self-adaptive target distribution**
- **Variable number of bins**

DynOpEq

Operator Equalisation (Silva and Vanneschi, 2009)

- **No rejections**
- **Individuals are mutated to fit the desired length**

MutOpEq

Operator Equalisation
pseudocode

accept(individual i, bin b):

```
accept = false
if b exists
  if b is not full
  or i is the new best-of-bin
    accept = true
else
  if i is the new best-of-run
    create new bin b
    accept = true
```

accept unchanged
accept mutated

validate(individual i):

```
l = length of individual i
b = bin that holds individuals of length l
```

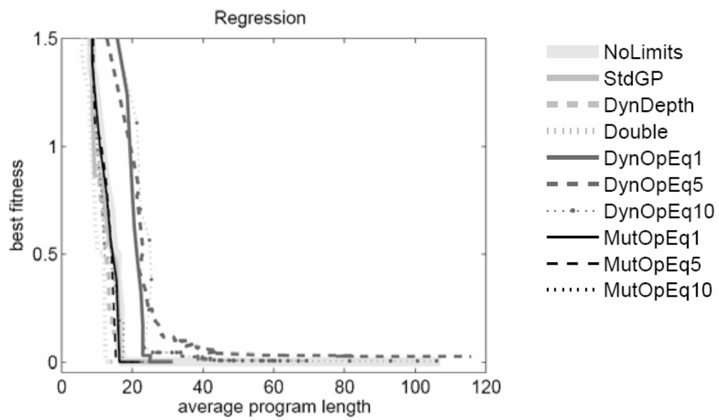

Operator Equalisation
pseudocode

validate(individual i):

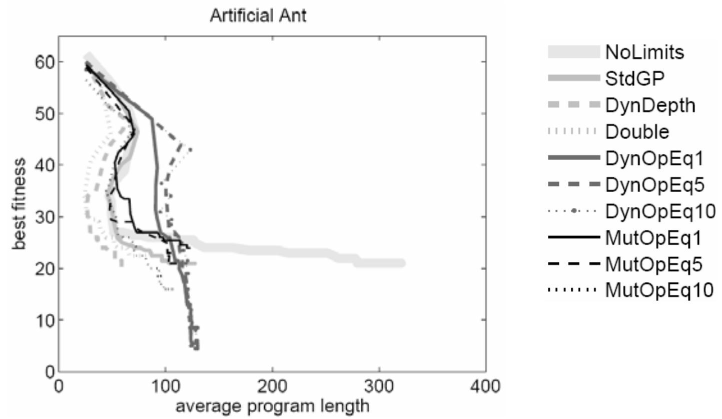
```
if b exists
   if b is full
      validate = false
      if DynOpEq and i is the new best-of-bin
         validate = true
      endif
   else
      validate = true
   endif
else if i is the new best-of-run
   create new bin
else
   validate = false
endif

if MutOpEq and validate = false
   closest_b = non-full bin closest to b
   mutate i until it fits closest_b
   validate = true
endif
```

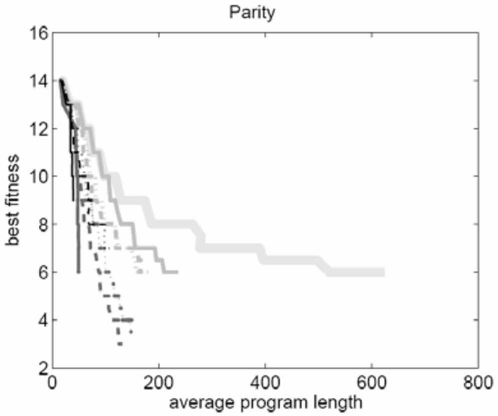

MutOpEq
benchmark results – realistic view



MutOpEq
benchmark results – realistic view

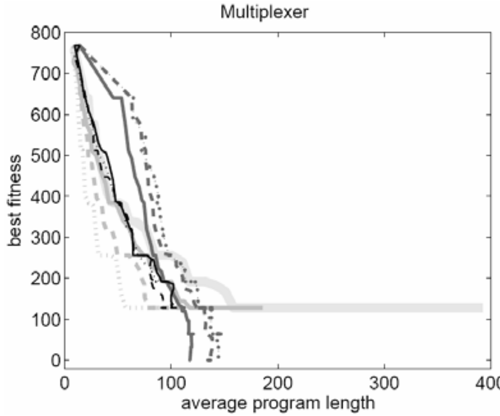


MutOpEq
benchmark results – realistic view



65

MutOpEq
benchmark results – realistic view



66

DynOpEq versus MutOpEq

- MutOpEq **runs faster** than DynOpEq
- MutOpEq **dynamics is different** from DynOpEq
- MutOpEq **learns slower** than DynOpEq

67

Real World Results
experiments - drug discovery

2 hard real-life problems

Symbolic Regression

Drug Discovery Applications

Bioavailability:
241 variables
359 samples

Toxicity:
626 variables
234 samples

Training (70%) + Test (30%) data
(30 random partitions)

Previous experiments identified
bloat and overfitting as problems

3 techniques

StdGP (max depth 17)

DynOpEq (no limits)

MutOpEq (no limits)

500 individuals
100 generations
30 runs…

68

Real World Results
experiments – satellite imagery

2 hard real-life problems
+
1 easy real-life problem

Symbolic Regression
Classification of Satellite Imagery
Identification of Burned Areas:
7 variables
3637 samples
(2 classes)
Training (70%) + Test (30%) data
(30 random partitions)
No previous work done on this data

3 techniques

StdGP (max depth 17)
DynOpEq (no limits)
MutOpEq (no limits)

500 individuals
200 generations
30 runs…

69

Real World Results
disclaimer

The following plots concern:

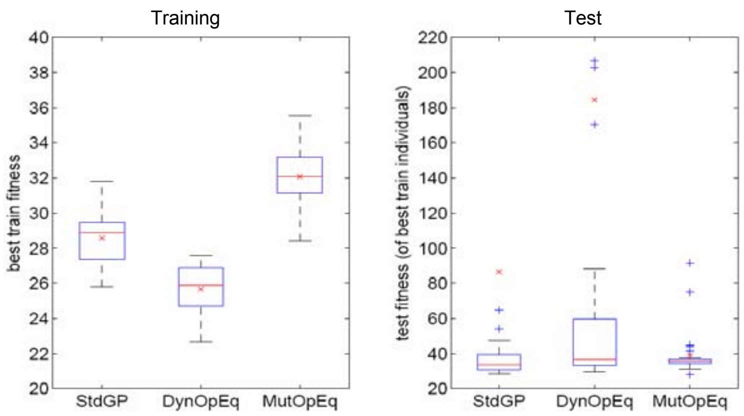
- Bloat (fitness, program length, relationship between them)
- Overfitting (training fitness, test fitness, evolution of both)

They ignore:

- Time spent on rejections in DynOpEq and StdGP
- Time spent on mutations in MutOpEq

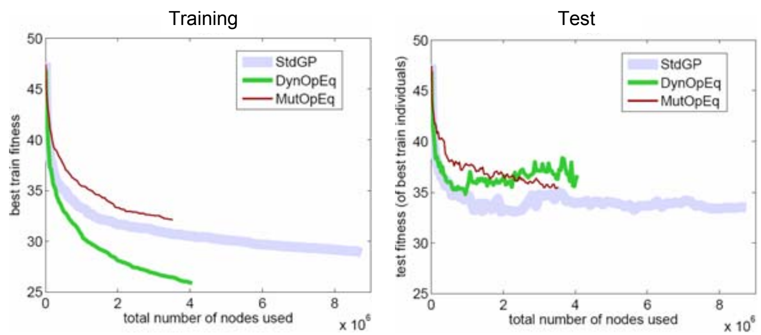
70

Real World Results
bioavailability



71

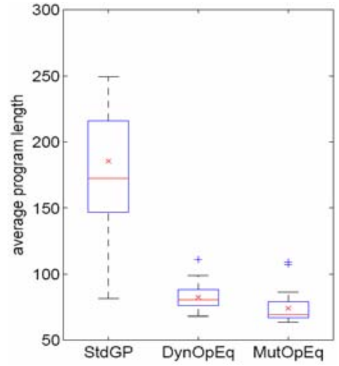
Real World Results
bioavailability

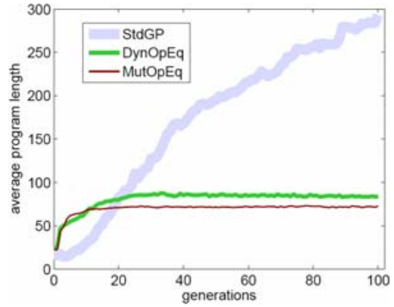


- DynOpEq overfits
- MutOpEq and StdGP do not overfit

72

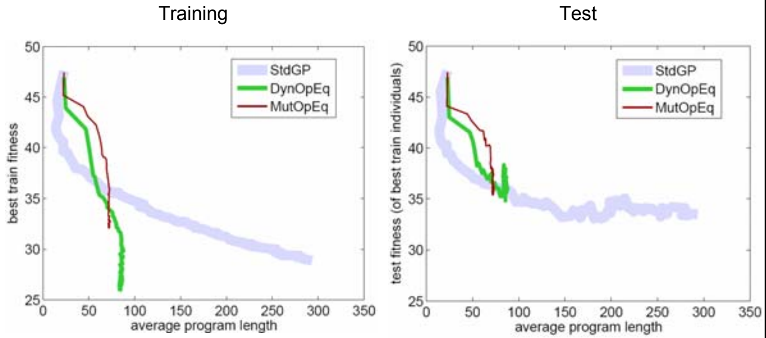
Real World Results
bioavailability





- StdGP bloats (?)
- DynOpEq and MutOpEq do not bloat

Real World Results
bioavailability



Real World Results
bioavailability – bloat versus overfitting

Eliminating bloat should have an impact on overfitting
“shorter solutions generalize better”

	Bloats	Overfits
StdGP	✓	✗
DynOpEq	✗	✓
MutOpEq	✗	✗

!!!

Is the effective code shorter?

Real World Results
bioavailability - actual solutions

After removing inviable code and simplifying the expressions:

MutOpEq, run 21, original length 75, training fitness 27.96, test fitness 32.93:

$$\left[2x_2 + \frac{x_2}{x_{30}} + x_{58} - (x_6 - x_{35})x_{91} + x_{231}\right] x_{30}^{-1} \left(\frac{x_2 + x_{231}}{x_{200}} + x_{143}\right)^{-1} + \frac{x_{30}x_{45}}{x_{149}} + \frac{x_{205}}{x_{143}} +$$
$$+ \left(x_{182} + \frac{x_{116}}{x_{131}}\right) x_{219}^{-1} - x_{164}\,(x_{160} - x_{219}) \left[x_{200} + x_{219}^{-1}\left(x_{110} - \frac{x_2}{x_{30}} - x_{58}\right)^{-1}\right]^{-1}$$

MutOpEq 27, original length 133, training fitness 33.43, test fitness 30.50:

$$x_{231} + x_{227} + \frac{x_{125}}{x_{124}x_{227}} + x_{91}x_{224} + 2\frac{x_{191}}{x_{218}} + 2\frac{x_{151}x_{222}}{x_{142}x_{227}} + 3\frac{x_{201}}{x_{91}x_{170}} + \frac{x_{142}x_{164}x_{97}}{x_{222}x_{224}} - 5x_{124} + \frac{x_{199}}{x_{32}x_{227}} +$$
$$+ \frac{x_{164}}{x_{227}x_{199}} + x_{164}x_{222}x_{225} - x_{91} + x_5 - x_{12} - x_{18} + x_{191}^{-1} + 2x_{199}^{-1} + \frac{x_{106}}{x_{97}}$$

DynOpEq 14, original length 129, training fitness 27.72, test fitness 35.76:

$$2x_{38} - \left(8x_{94} + 2x_{18} + 23x_{28} + \frac{x_{101}}{x_{221}^2} + \frac{x_{182}}{x_{221}} - 2x_{232}x_{41} + \frac{x_{38}}{x_{221}} - x_{232} - x_{26}^2x_{94} + x_{41} + \frac{x_{28}}{x_{221}}\right)$$
$$(x_{52} - x_{189} - x_{28})^{-1} + 2x_{26} - 3x_{18} + 2x_{50} + x_{230}$$

Real World Results
bioavailability - actual solutions

After removing inviable code and simplifying the expressions:

StdGP, run 8, original length 207, training fitness 27.46, test fitness 30.77:

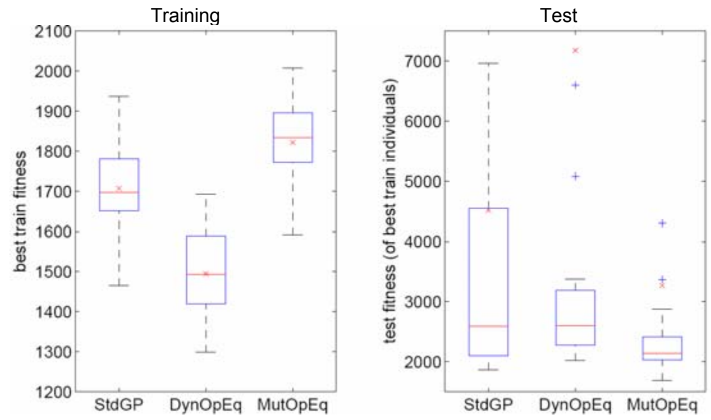
$$x_{17}+\Bigg[1+2\frac{x_{231}}{x_{212}x_{45}}+(2x_4+x_{231})\left(x_9-x_{61}-\frac{x_{82}}{x_4}\right)(x_4+x_{30})^{-1}+x_5x_{18}\left(\frac{x_5}{x_{212}}+2x_{17}\right)^{-1}x_{216}{}^{-1}x_{45}{}^{-1}+(x_4+x_{30})$$
$$\left(x_9-x_{61}-\frac{x_{82}}{x_{238}}\right)^{-1}+x_{212}+\frac{x_{17}}{x_{212}}+10x_{17}+3x_{30}+\frac{x_5}{x_{212}}+\frac{x_{17}}{x_{18}}+\left(\frac{x_{17}}{x_{18}}+x_{30}\right)(x_9-x_{118}-x_{216}{}^{-1})^{-1}+\frac{x_{17}}{x_4+x_{231}}+$$
$$+\frac{2x_4+x_{30}}{x_{216}}+\frac{x_4+4x_{231}+4x_{17}+2x_{37}+x_{30}}{x_{18}}+2x_{211}+2\frac{x_{17}}{x_{216}}+5x_4+2x_5+5x_{37}+4x_{231}\Bigg]x_{30}{}^{-1}-x_4$$

StdGP, run 12, original length 135, training fitness 28.50, test fitness 30.51:

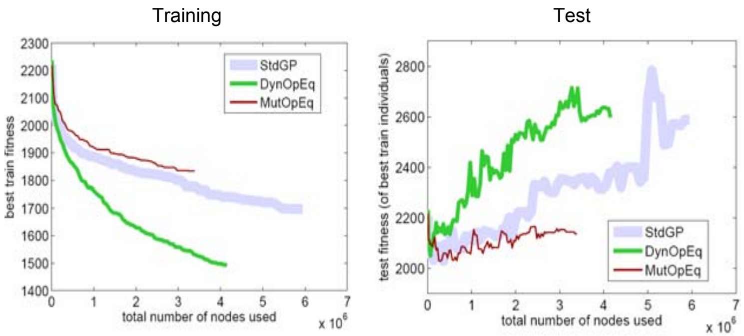
$$x_{45}-\Bigg[x_3+x_{64}-x_{46}-\left(\frac{x_{17}}{x_{37}x_{131}}-x_{30}-3x_{56}+x_{17}\right)\left(x_{37}-\left((x_{56}-x_{17})\left(\frac{x_{152}}{x_{37}}-x_{30}-x_{56}+x_{17}\right)^{-1}-x_{94}-x_{231}\right)\right.$$
$$\left.((x_{131}-x_{17})x_{200}-x_{17})^{-1}-x_{30}-x_{76}+x_{56}\right)^{-1}\Bigg]\left[\left(\frac{x_{17}}{x_{30}+x_{76}-x_{56}}-x_{17}\right)x_{37}{}^{-1}-x_{94}-x_{231}\right][(x_{152}-x_3)x_{200}-x_{17}]^{-1}$$
$$\left[\frac{x_{37}}{x_{231}-x_{56}\,(x_{64}-x_{46})x_{239}-x_{17}}-x_{239}+x_{121}-x_{30}\right]^{-1}$$

Is this the effective code?
It is not shorter for StdGP!

Real World Results
toxicity

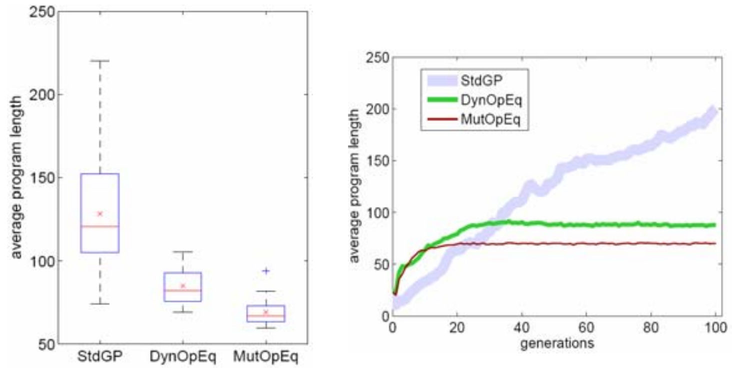


Real World Results
toxicity



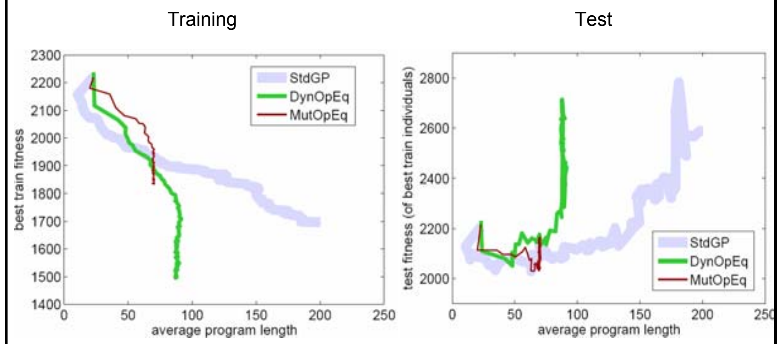
- All techniques overfit
- Overfitting proportional to learning

Real World Results
toxicity



- StdGP bloats (?)
- DynOpEq and MutOpEq do not bloat

Real World Results
toxicity



81

Real World Results
toxicity – bloat versus overfitting

Very hard problem and… still no bloat with OpEq!
(work in progress is revealing that Dynamic Limits and Double Tournament bloat on this problem)

	Bloats	Overfits
StdGP	✓	✓
DynOpEq	✗	✓
MutOpEq	✗	✓

Overfitting is simply proportional to learning?

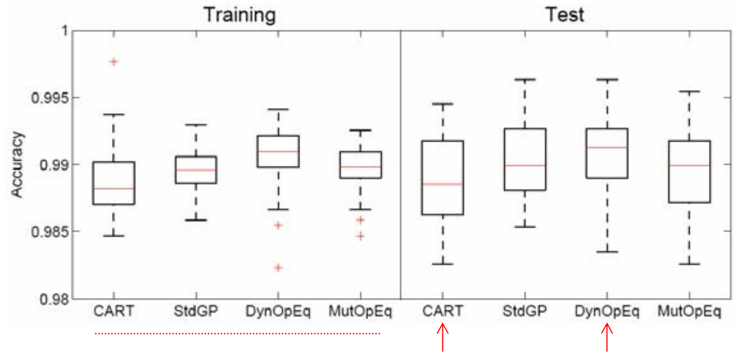
82

DynOpEq versus MutOpEq

- MutOpEq **runs faster** than DynOpEq
- MutOpEq **dynamics is different** from DynOpEq
- MutOpEq **learns slower** than DynOpEq
- MutOpEq **overfits less**

83

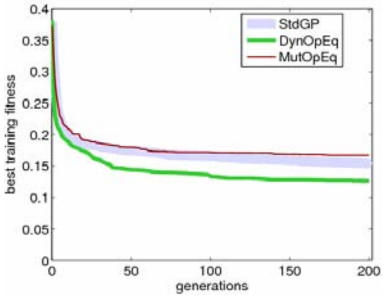
Real World Results
burned areas



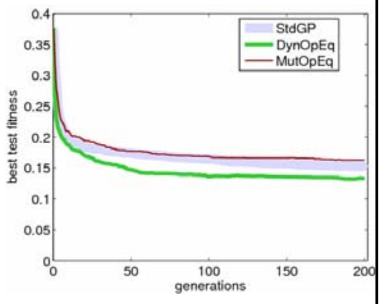
84

Real World Results
burned areas

Training

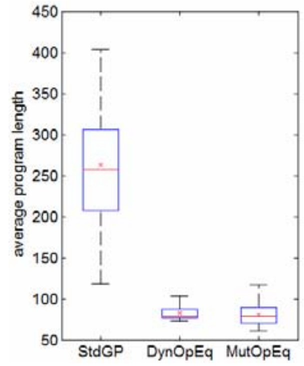


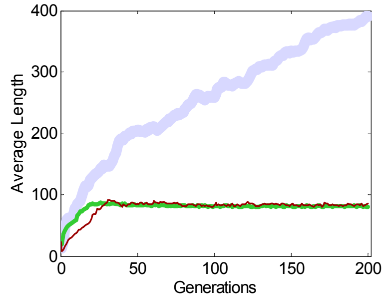
Test



• No overfitting at all

Real World Results
burned areas

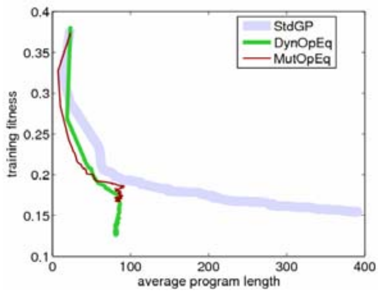




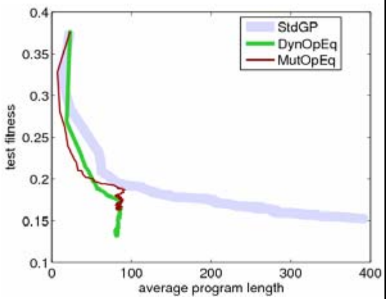
• StdGP bloats (?) • DynOpEq and MutOpEq do not bloat

Real World Results
burned areas

Training



Test



Real World Results
burned areas - actual solutions

After removing inviable code and simplifying the expressions:

Example of solution by StdGP:

$$y = x_6 - (x_2 + x_4) + (x_6 + x_7 - 3x_4)\Big(3x_6 - (x_1 + 3x_4) + x_1\big(2x_1 + x_6 + x_7 - (x_3 + 5x_4) + (7x_6 - 10x_1 - 4x_4 + 12x_7 - 2x_2 - 7x_3)(8x_1 - 3x_6 + x_4 - 3x_7)\big)\Big)$$

Example of solution by DynOpEq:

$$y = \frac{x_3 x_6}{x_7} + x_1 - x_4 + x_6^2 - \frac{x_6}{x_5} + \frac{x_5 x_6^3}{x_1 x_3^2}\left(\frac{1}{x_3} - \frac{x_7}{x_5}\right)\left(\frac{x_5}{x_3} + \frac{1 - \frac{x_3 x_5}{x_6^2 x_7}}{\frac{x_5^2}{x_7^2} + x_6 + x_7 - x_3 - \frac{x_6}{x_5}}\right)$$

Example of solution by MutOpEq:

$$y = x_2 - x_5 + x_3 x_6 + \frac{x_3}{x_6}$$ ← **Several short solutions were produced**

(rule to apply to all GP solutions: if $y < 0.5$ then $class = 0$ else $class = 1$)

DynOpEq versus MutOpEq

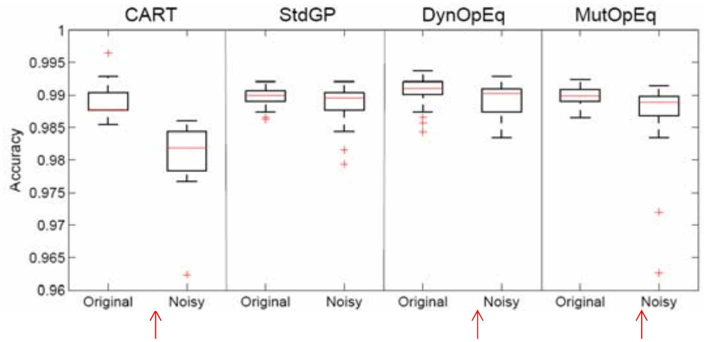
- MutOpEq **runs faster** than DynOpEq
- MutOpEq **dynamics is different** from DynOpEq
- MutOpEq **learns slower** than DynOpEq
- MutOpEq **overfits less**
- MutOpEq **produces short solutions more easily**

89

Real World Results

burned areas – noisy data

After changing the x_4 values ramdomly by 10%:



• StdGP is the only one showing generalization ability!

90

Open Questions

Eliminating bloat does not seem to help generalization at all.

So what is necessary for good generalization ability?

- The size of the effective code seems to be irrelevant
- Small size does not mean low complexity
- Complexity may mean lack of readability
- Complexity should be related to overfitting

"Measuring Bloat, Overfitting and Functional Complexity in Genetic Programming" by Vanneschi, Castelli, Silva
GECCO-2010

91

Open Questions

MutOpEq runs faster, overfits less, produces shorter solutions.

Is it really better than DynOpEq?

- Cons of MutOpEq

  MutOpEq is a slow learner because
  1) By not rejecting individuals, it is less selective than DynOpEq
  2) By mutating individuals, it may be making them worse

  MutOpEq mutates without first evaluating. It risks spoiling the perfect individual without even knowing it (never happens with DynOpEq)

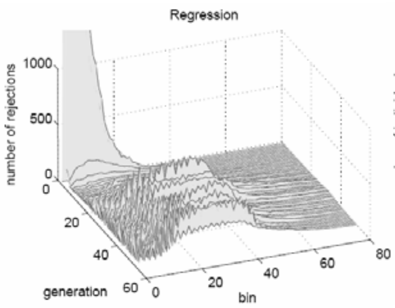
- Cons of DynOpEq (besides the obvious efficiency problem)

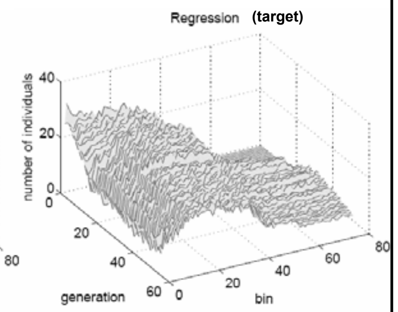
92

Open Questions

DynOpEq

Why was DynOpEq not good in simple Symbolic Regression?
Plots of bin width 1. Notice the jagged patterns.



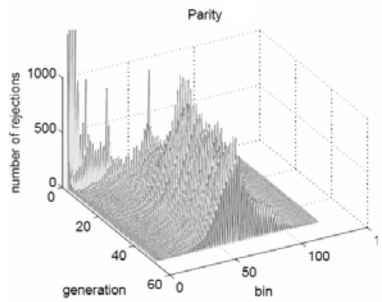


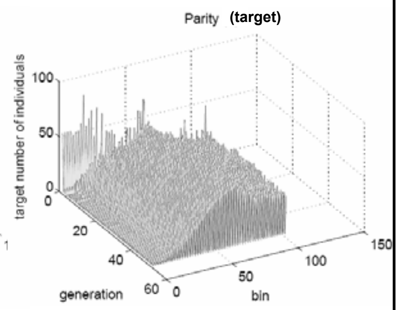
93

Open Questions

DynOpEq

Compare with the patterns of Parity. Bin width 1.



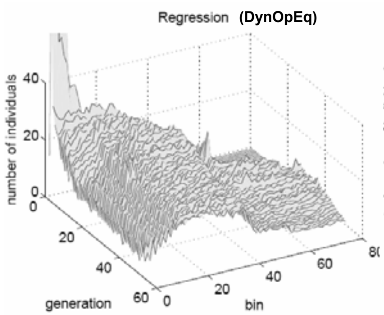


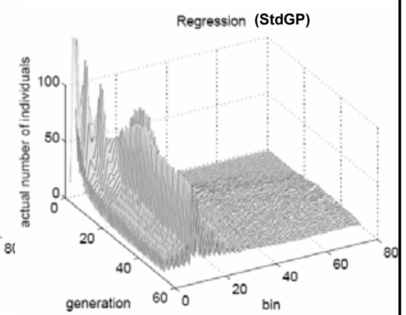
94

Open Questions

DynOpEq

Compare the actual distributions of DynOpEq and StdGP. Bin width 1.





95

Open Questions

DynOpEq

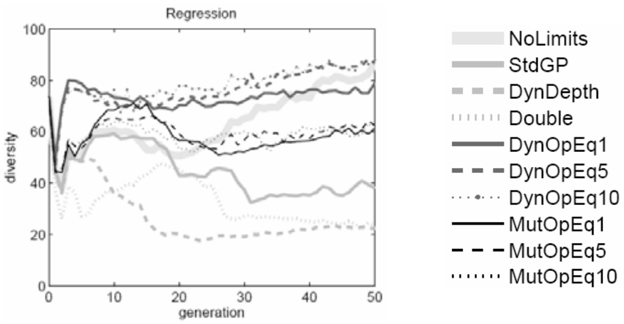
- The patterns of Parity are completely jagged because this problem uses only binary operators, so with bin width 1 some bins never receive individuals
- The patterns of Regression are also jagged, only slightly in DynOpEq, and very much in StdGP
  - Regression uses unary and binary operators, but apparently the individuals of even size are difficult to create, thus the jagged pattern
  - DynOpEq forces the creation of even size individuals by rejecting the odd size ones, whose bins fill up quickly. So the actual length distribution of DynOpEq is quite smooth when compared to StdGP

96

Open Questions

DynOpEq

• This forced creation of even size individuals increases genotypical diversity:

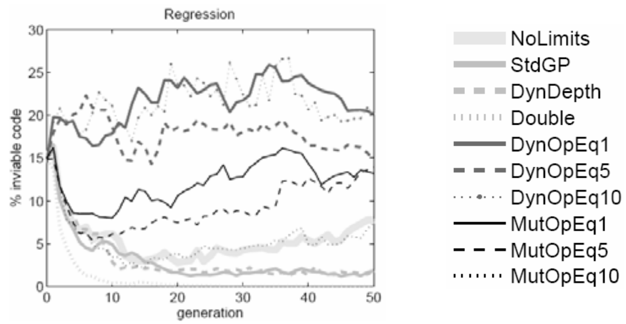


97

Open Questions

DynOpEq

• Unfortunately this seems to be achieved by allowing an atypical (in Regression) proliferation of inviable code:



98

Future Work

Improvements

How to improve the efficiency of DynOpEq?

- Evaluating only the individuals that fall outside the target is not a good idea:
  - Not overriding the target means waiting a long time for it to be filled
  - Also means slower learning
  - Not evaluating introduces the risk of rejecting the perfect individual

How to speed the learning of MutOpEq?

- Minimize impact on fitness:
  - Using smarter mutations, e.g. that act on inviable code, is expensive
  - Delay mutations: save individuals that do not fit the target, mutate only when there are enough for a new generation; maybe calculate fitness to decide whether to mutate or to replace by another individual, e.g. a parent

99

Future Work

Improvements

A hybrid DynOpEq / MutOpEq approach?

Each individual that does not fit the target can be:
- accepted
- mutated
- replaced
- rejected

Depending on:
- length and fitness
- current state of target

Any other suggestions? Please…? ☺

100

Future Work

Extensions

Incorporating OpEq into other elements of the evolutionary process:

Whenever the decision involves the size/length of the individual, let it involve "how well the individual fits the target" instead.

Examples:

Parametric Parsimony Pressure

Let fitness be a function of raw fitness and "how well the individual fits the target"

Double Tournament

Let the size/length tournament be replaced by a "how well the individual fits the target" tournament

101

Future Work

Extensions

Incorporating OpEq into other elements of the evolutionary process:

Whenever the decision involves the size/length of the individual, let it involve "how well the individual fits the target" instead.

Examples:

Special Genetic Operators

Choose the crossover point on the second parent so that the offspring "fits the target well"

Advantages: "how well the individual fits the target" dynamically changes along the run

Disadvantages: How to calculate such a measure?

102

Future Work

the next chapter…

The secret for the success of OpEq may ***not*** be:

- to follow a distribution target that is "proportional" to fitness
- to avoid the smaller individuals

It may be exactly the opposite:

- to ignore fitness
- to promote a continued search among ***all*** program lengths

"Reassembling Operator Equalisation – A Secret Revealed"
by Sara Silva
to appear in GECCO-2011

103

References / Suggested Reading

- S. Dignum, R. Poli, Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In Proceedings of GECCO-2007.
- S. Dignum, R. Poli, Operator equalisation and bloat free GP. In Proceedings of EuroGP-2008.
- W.B. Langdon, R. Poli, Foundations of genetic programming, 2002.
- R. Poli, W.B. Langdon, N.F. McPhee, A Field Guide to Genetic Programming, 2008.
- S. Silva, Controlling bloat: individual and population based approaches in genetic programming. PhD thesis, 2008.
- S. Silva, S. Dignum, Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP. In Proceedings of EuroGP-2009.
- S. Silva, L. Vanneschi, Operator equalisation, bloat and overfitting - a study on human oral bioavailability prediction. In Proceedings of GECCO-2009.
- L. Vanneschi, M. Castelli, S. Silva, Measuring Bloat, Overfitting and Functional Complexity in Genetic Programming, In Proceedings of GECCO-2010.
- S. Silva, Reassembling Operator Equalisation – A Secret Revealed. To appear in GECCO-2011.

104

Acknowledgements

- Leonardo Vanneschi and Stephen Dignum
- Project “EnviGP – Improving Genetic Programming for the Environment and Other Applications” (PTDC/EIA-CCO/103363/2008) from Fundação para a Ciência e a Tecnologia, Portugal
- You, who are still listening to me!
  Or maybe not…

Thank you!