Genetic Programming

Sara Silva
INESC-ID Lisboa, IST/UTL, Portugal
CISUC, Univ.Coimbra, Portugal

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

Sara Silva – Biography

- BSc (1996) and MSc (1999) in Informatics
  - Faculty of Sciences of the University of Lisbon, Portugal
- PhD (2008) 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 contributions to GP
  - Dynamic Limits
  - Resource-Limited GP
  - Operator Equalisation in practice

Goals of this tutorial

To provide you with

motivation,
intuition
and
practical advice

about Genetic Programming

...and very few technical details!

Agenda

Motivation

Representation

Running GP
- Population Initialization
- Fitness Evaluation
- Selection for Breeding
- Genetic Operators
- Selection for Survival
- Stop Condition

Properties of Solutions

Summary

Demos

Problems & Open Questions
- Bloat
- Overfitting
- Complexity
- Multiclass Classification

Project

Motivation
Origins

Genetic Programming (GP) is the youngest paradigm inside the research area called Evolutionary Computation (EC).

Created by John Koza (first book published in 1992), GP has its origins in Genetic Algorithms (GAs). John Koza was a PhD student of John Holland, the father of GAs.

The crucial difference between GP and GAs is the representation of the individuals:

GA representation – fixed length numerical strings

GP representation – variable length structures containing whatever ingredients are needed to solve the problem

Motivation
Example – Symbolic Regression

Problem
Find an expression (f) that transforms a pair of numbers (x_1, x_2)
in a result such that: f(2,3) = 5, f(4,6) = 10, f(5,1) = 6
→ f (x_1, x_2) = x_1 + x_2

Problem
Find an expression (f) that transforms a triplet of numbers (x_1, x_2, x_3)
in a result such that: f(1,2,3) = 5, f(6,2,5) = 14, f(1,10,8) = 7
→ f (x_1, x_2, x_3) = x_1 - x_2 + 2 x_3

Problem
Find an expression (f) that transforms a 100-tuple of numbers (x_1,..., x_{100})
in a result such that: …

Motivation
Potential

The goal of GP is to evolve computer programs. Given all the elements of a programming language, GP has the potential to find the computer program that solves a particular problem.

Theoretically, GP can solve any problem whose candidate solutions can be measured and compared in terms of quality, or “how well they solve the problem”.

Motivation
Difficulties

Consider a GA using binary strings of length n. There are 2^n possible configurations of 0 and 1 bits. This is the size of the search space.

Now consider one of the symbolic regression problems presented earlier. How many different functions can be built using variables, constants and arithmetic operators? With a variable length representation, the search space is potentially unlimited.

Even if a maximum program length is established, how many different computer programs can be written using all the elements of a programming language?

Motivation
Applications

GP is not generally used to evolve computer programs, but it has been very successful in a vast number of applications.

- Modeling and regression
- Image and signal processing
- Time series prediction
- Control
- Medicine
- Biology and bioinformatics
- Arts and entertainment

...

... and almost anything else one can think of!

Motivation
Creativeness

John Koza and his "invention machine"

In 1995 John Koza saw his "invention machine" create a complex electronic circuit from scratch, i.e, by combining basic electronic components like resistors and capacitors.

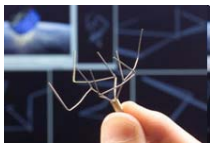
This circuit was a patented low-pass filter, a circuit used for cleaning up the signal passing through an amplifier.

Since then, GP has replicated many other previously patented items. New patents have been registered resulting from GP creativeness. In 2005 the invention machine earned one of the first patents ever granted to a non-human designer, for developing a system to make factories more efficient.

Motivation
Creativeness

GP often yields results that are not merely academically interesting, but competitive with the work developed by humans.

Humies – Annual Awards for Human-Competitive Results Produced by Genetic and Evolutionary Computation (http://www.genetic-programming.org/hc2010/cfe2010.html)



Antenna launched on NASA's Space Technology 5 mission

"What he got, several hundred generations later, appeared to be a mistake"

"It looked like a bent paper clip"

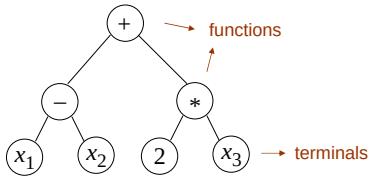
Representation
Common Types

Previous example:
Find and expression (…) such that (…)

→ f (x_1, x_2, x_3) = x_1 - x_2 + 2 x_3

Tree-based GP:

Parse tree



LISP-like expression

(+ (− x_1 x_2) (+ x_3 x_3))

Tree-based GP is the most popular because:

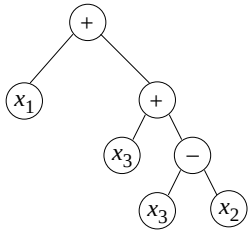
- Koza used/uses it
- Allows for simple genetic operators

Other types of GP:
Linear GP
Graph GP

Representation
Common Types

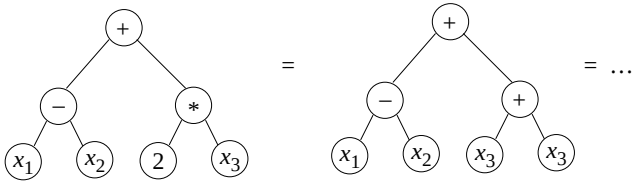
Previous example:
Find and expression (…) such that (…)

→ f (x_1, x_2, x_3) = x_1 - x_2 + 2 x_3



Tree-based GP:

Parse tree



Representation
Ingredients

Previous example:
Find and expression (…) such that (…)

→ f (x_1, x_2, x_3) = ~~x_1 - x_2 + 2 x_3~~

f (1, 2, 3) = 5
f (6, 2, 5) = 14
f (1, 10, 8) = 7

Function Set: { ~~+ , - , *~~ }

Terminal Set: { ~~x_1 , x_2 , x_3 , 2~~ }

Function Set: { + , - , * , / }

Terminal Set: { x_1 , x_2 , x_3 }

The success or failure of a GP run may depend on the setting of the function and terminal sets.

- if the ingredients are not enough to represent the optimal solution, the run will never reach it
- if there are too many superfluous ingredients, the search process will get lost in a too large, too complex search space

Representation
Closure

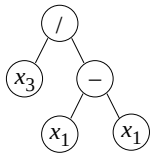
In GP, protection is preferred over repair procedures

Function Set: { + , - , * , / }

Terminal Set: { x_1 , x_2 , x_3 }

What's "wrong" with this **function set**?

What happens if GP tries to evaluate this individual?



To verify the closure property, any output argument of any of the functions / terminals must be a valid input for all the other functions.

Division must be "protected" to avoid division by zero.

Defining protected division a//b:
if b=0 then 1 else a/b

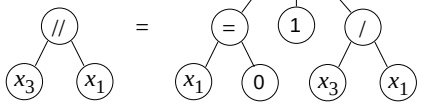
Other operators need it, like log. Others require protection from 0, negative numbers, overflowing, etc.

Representation
Closure

Function Set: { + , - , * , // }

Terminal Set: { x_1 , x_2 , x_3 }

Protected functions have side effects on the search space.



Very close values of x_1 may return very different results.

This example is also useful to introduce two advanced options of GP:

- Strongly-Typed GP (STGP)
- Automatically Defined Functions (ADFs)

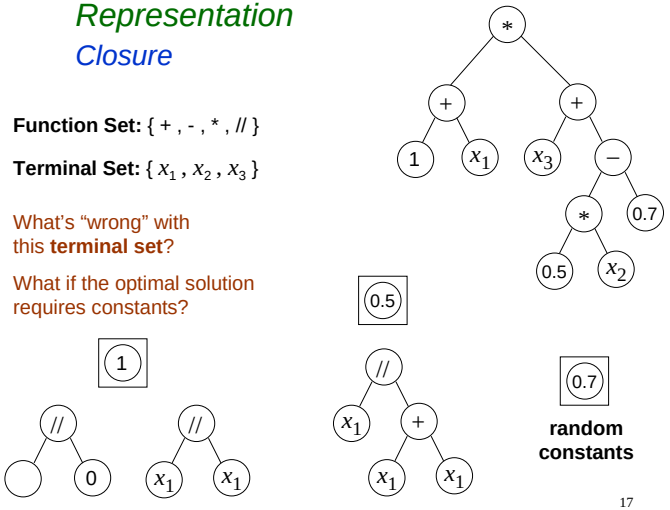
Representation
Closure

Function Set: { + , - , * , // }

Terminal Set: { x_1 , x_2 , x_3 }

What's "wrong" with this **terminal set**?

What if the optimal solution requires constants?



Representation
Practical Advice

When you don't know anything about the problem and possible solutions, use small function and terminal sets.

Add / replace functions and constants if learning is difficult (with a population that is already large).

If you know some ingredients that may help solve the problem, use them! GP needs all the help it can get.

Specialized ingredients are allowed / encouraged, like complex functions that perform specific tasks on the data.

Representation
Other examples

Parity problem
Return true (1) if number of 1's in the input data is even; return false (0) otherwise.

Data:

x_1, x_2, x_3	$f(x_1, x_2, x_3)$
0 , 0 , 0	1
0 , 0 , 1	0
0 , 1 , 0	0
0 , 1 , 1	1
…	…

Function Set: { and, or, nand, nor }

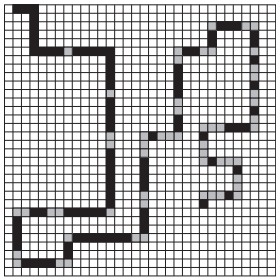
Terminal Set: { x_1 , x_2 , x_3 }

Adding 'xor' to the function set transforms many hard parity problems into easy problems!

This can be solved like a symbolic regression problem using logical operators.

Representation
Other examples

Artificial Ant problem
Evolve a strategy to follow a food trail.



Function Set: { if-food-ahead, prog2, prog3 }

Terminal Set: { left, right, move }

Maintains closure. Input/output arguments are current position and units of food eaten.

Example
(if_food_ahead (move)
(prog2 left move))

Running GP
Population Initialization

Initialization Methods

Initial trees are generated so they don't exceed a certain depth (typically 6). Koza described three initialization methods: Grow, Full, Ramped Half-and-Half.

Starting for the root of the tree, nodes are added until the leaves. The choice of nodes is mostly random among the function and terminal sets, obbeying the restrictions:

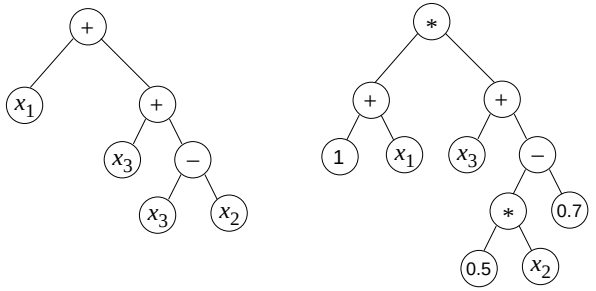
Grow - maximum depth cannot be exceeded

Full - tree must be full (all branches must reach maximum depth) - may not be appropriate for some function sets

Ramped - for each depth between 2 and the maximum depth, half of the trees are initialized with Grow and the other half with Full

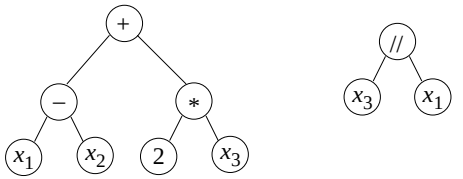
Running GP
Population Initialization

Examples of Grow and Full trees



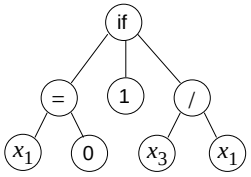
Running GP
Population Initialization

Examples of Grow and Full trees



Running GP
Population Initialization

Examples of Grow and Full trees



Running GP
Population Initialization

Population Diversity

Ramped Half-and-Half ensures a high structural (genotypical) diversity of the trees in the initial population. This may not translate into semantic (phenotypical) diversity.

In GP the initial diversity is not so important. In regression problems most initial individuals are very unfit and quickly discarded by selection, but GP has a remarkable ability to maintain and/or recover diversity, even without using mutation.

The population size (number of individuals) tends to be much more important than the initialization method. Typically GP uses larger populations than GAs, to cover the larger search space more effectively.

Running GP
Population Initialization

Premature convergence vs Stagnation

Premature convergence is a rare event in GP, but it may happen very early in the run in highly complex problems. In case it happens, the initial population should be allowed deeper trees. Increasing the number of individuals without increasing the initial depth may be useless.

Stagnation is a common event in GP which can be mistaken by premature convergence. The cause is not loss of diversity, but the proliferation of redundant code. The result is similar: difficulty in learning. To be addressed in **Problems & Open Questions**.

Advanced options like using multiple populations and niching techniques, common for preventing premature convergence, are not helpful in preventing stagnation.

Running GP
Fitness Evaluation

In GP the fitness value is usually a direct translation of the error, so the **lower** the fitness, the **better** the individual.

Typical fitness measures for symbolic regression problems:

- Absolute differences between expected and obtained results, summed for all samples of the data set
- Root mean squared error

Given the diversity and complexity of the problems that can be tackled by GP, evaluating the fitness may be a computationally expensive process. Multiobjective optimization can also used.

The fitness function is crucial for the success of GP. Ideally, it should promote small steps towards the optimum value. Use knowledge about the problem, if available.

Running GP
Fitness Evaluation

Artificial Ant problem

Evaluating an individual of the Artificial Ant problem involves simulating the behavior of the ant inside the food trail.

The evolved strategy is repeatedly applied until a certain number of time steps in reached. Each action of the ant counts as a time step. Koza used 600 time steps in his first book, but wrongly reported it as 400, so both values are commonly used.

The fitness is the amount of food pellets eaten during this time. It can also be measured as the amount of food pellets remaining in the trail after using all the time steps.

Running GP
Selection for Breeding

The amount of selective pressure used to select the parents of each next generation is determined by the method used.

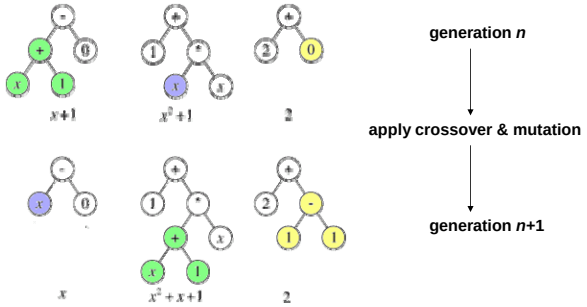
• **Roulette** methods impose a high selection pressure, which can be lowered by using SUS (Stochastic Uniform Sampling)

• **Tournament** methods allow a much finer control of selection pressure, by controlling the size of the tournaments. Some tournament methods (like Double Tournament) can be used to implement a multiobjective-like optimization

Lower selection pressure slows down convergence and promotes diversity.

Running GP
Genetic Operators

The most commonly used genetic operators in GP are standard subtree crossover and mutation.



Running GP
Genetic Operators

In GP, mutation is not generally used within crossover, but rather as an independent operator, or not used at all. Crossover alone can maintain / recover genotypical diversity, being sometimes regarded as a ‘macromutation’ operator.

Standard subtree crossover is a highly destructive operator. Other crossovers, more “well-behaved”, have been developed, but they tend to require diversity-preserving measures. One-point crossover was very important in developing the ‘building blocks’ theory for GP.

Genetic operators can be specifically crafted to deal with particular representations, and to include knowledge about the problem. STGP and ADFs also require special care.

Running GP
Selection for Survival

The selection for survival is independent from the selection for breeding. It happens after the application of the genetic operators and decides, from among parents and offspring, which individuals will be part of the new generation.

Elitist strategies guarantee the survival of the best individuals to the next generation. Too much elitism reduces diversity and promotes premature convergence.

Criteria other than fitness may be used to decide which individuals survive.

Steady-state evolution is sometimes used in GP.

Running GP
Stop Condition

Different stop conditions may be used in a GP system.
Some examples are:

- Number of generations
- Fitness reaching a certain value
- Fitness not improving for a number of generations
- Number of “hits” (data samples being handled correctly)
- Other criteria related to tree size, overfitting, etc.

Typically, GP uses fewer generations than GAs.

Properties of Solutions
Diversity

GP is a highly stochastic process, and different runs will provide different solutions. They may be different because:

1) The trees are structurally different, although they represent the same phenotype once the redundant code is removed

2) The trees represent different structures / genotypes but their phenotype does the same thing

3) The trees really represent different solutions, using the same, or even different, variables

Note that GP does automatic feature selection!

Properties of Solutions
Automatic Feature Selection

The features selected by GP may not be the same in each run, particularly in high dimensional difficult real-world problems.

Advantages
Robustness to difficult data. Can provide distinct alternative solutions for the same problem. Can reveal data redundancies. Results in creativeness and innovation.

Disadvantages
Does not provide “exact” solutions. Interpreting one solution may already be difficult – even more for a set of different solutions. Low GP credibility among practitioners.

Properties of Solutions
Examples – easy real world problem

$$y = x_2 - x_5 + x_3 x_6 + \frac{x_3}{x_6}$$

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

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

Properties of Solutions

Examples – difficult real world problem

$$2x_{58} - \left(8x_{94} + 2x_{18} + 23x_{28} + \frac{x_{101}}{x_{221}^2} + \frac{x_{182}}{x_{221}} - 2x_{252}x_{41} + \frac{x_{38}}{x_{221}} - x_{252} - x_{26}^2 x_{94} + x_{41} + \frac{x_{28}}{x_{221}}\right)$$
$$(x_{52} - x_{189} - x_{28})^{-1} + 2x_{26} - 3x_{18} + 2x_{50} + x_{230}$$

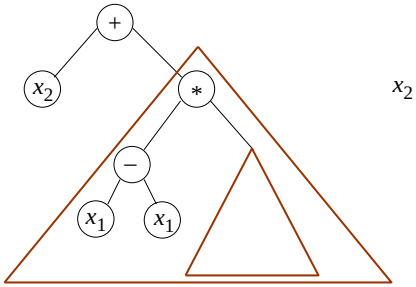
$$x_{231} + x_{227} + \frac{x_{125}}{x_{124}x_{227}} + x_{91}x_{224} + 2\frac{x_{191}}{x_{218}} + 2\frac{x_{151}x_{222}}{x_{142}x_{227}} + 3\frac{x_{201}}{x_{91}x_{170}} + \frac{x_{142}x_{164}x_{97}}{x_{222}x_{224}} - 5x_{124} + \frac{x_{199}}{x_{32}x_{227}} +$$
$$+ \frac{x_{164}}{x_{227}x_{199}} + x_{164}x_{222}x_{225}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

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

Properties of Solutions

Redundant Code - Introns

Inviable code

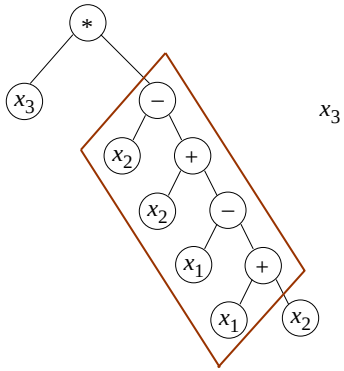


$x_2 + ((x_1 - x_1) * ( ))$

Properties of Solutions

Redundant Code - Introns

Unoptimized code



$x_3 * (x_2 - x_2 + x_1 - x_1 + x_2)$

Summary

Properties and Abilities

GP differs from GAs in the representation used for the individuals of the population. The set of ingredients used to build the solutions is the most influential element in a GP run.

The enormous versatility of the GP representation allows it to deal with any type of problem. As a last resort, GP can be instructed to evolve a computer program, but more realistic applications deal with more specific representations.

Given the “right” representation, GP is quite insensitive to most of the running parameters.

Summary
Properties and Abilities

The GP representation lends itself to high genotypical diversity, but which may not be present at the phenotypical level.

Proliferation of redundant code is common. Bloat.
To be addressed in **Problems & Open Questions.**

GP is a highly stochastic process. The solutions exhibit high variability among different runs. Feature selection is automatically performed.

Different runs = different solutions = different features selected

Summary
Advanced Options

Most advanced options of GAs have also been used in GP, like multiple populations, niching techniques, and multiobjective optimization.

Other advanced options are specific to the GP representation, like Strongly-Typed GP and Automatically Defined Functions.

Demos
GPLAB – A GP Toolbox for MATLAB

`http://gplab.sourceforge.net/`

GPLAB

A Genetic Programming Toolbox for MATLAB

by Sara Silva

home features download older versions new versions acknowledgements

Welcome to the homepage of

GPLAB - a Genetic Programming toolbox for MATLAB

(MATLAB is a product from The MathWorks)

I started developing GPLAB after searching for a free GP system for MATLAB and realizing there was none (which is not true any longer). After using it extensively for my own work I decided to release it, and the new versions that followed were the result of the many comments, suggestions, and additional code I have received from several users ever since - **thank you** all!

GPLAB includes most of the traditional **features** usually found in GP tools. It is able to accomodate a wide variety of usages, but its highly modular structure makes it a particularly versatile, generalist and easily extendable tool, highly suited for testing new elements and techniques in GP research.

If you are not a MATLAB user but are looking for an Evolutionary Computation tool, you may want to visit **this page** (a bit outdated, but still useful).

home features download older versions new versions acknowledgements

Last updated on June 8, 2009
© 2003-2009 Sara Silva

Demos
Examples

Symbolic Regression:

$f(x) = x^4 + x^3 + x^2 + x$

Function set: { +, –, * }
Terminal set: { x }

demo_reg1 (25 gens, 500 inds)
plots: fitness, diversity, best

demo_reg2 (25 gens, 500 inds)
function set: { +, -, *, // , sin, cos, log }

demo_reg3 (50 gens, 25 inds)
demo_reg4 (50 gens, 25 inds)
plots: + approximations

input	expected output
-1.0000	0.0000
-0.9000	-0.1629
-0.8000	-0.2624
-0.7000	-0.3129
-0.6000	-0.3264
-0.5000	-0.3125
-0.4000	-0.2784
-0.3000	-0.2289
-0.2000	-0.1664
-0.1000	-0.0909
0.0000	0.0000
0.1000	0.1111
0.2000	0.2496
0.3000	0.4251
0.4000	0.6496
0.5000	0.9375
0.6000	1.3056
0.7000	1.7731
0.8000	2.3616
0.9000	3.0951
1.0000	4.0000

Demos
Examples

3-Bit Even Parity:

return true (1) if number of 1's is even
return false (0) otherwise

Function set: { and, or, nand, nor }
Terminal set: { x_1, x_2, x_3 }

input			expected output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

demo_par1 (20 gens, 100 inds)
plots: fitness, best

demo_par2 (20 gens, 100 inds)
function set: { and, or, nand, nor, xor }

Demos
Examples

Artificial Ant:

return a strategy to follow a food trail

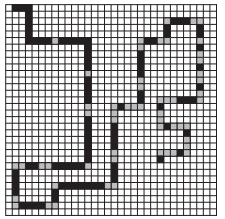
Function set: { if-food-ahead,
prog2, prog3 }
Terminal set: { move, left, right }

demo_ant1 (10 gens, 20 inds)
plots: fitness, best

load trained_ant
antsim(vars)
drawtree(vars.state.bestsofar.tree)

input

matrix of 1's and 0's
representing the food trail
1 = food (black)
0 = empty (white, grey)



Problems & Open Questions
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**"

Problems & Open Questions
Bloat

Definition

Excessive code growth without a corresponding improvement in fitness

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

Problems & Open Questions
Bloat

Pros and Cons

Pros Code compression and parsimony (effective code is shorter?)
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

Problems & Open Questions
Bloat

Theories

Based on introns

Code growth occurs as a protection against the destructive effects of genetic operators: introns provide nodes for neutral variations; size itself increases the chances that the crossover / mutation nodes are deeper into the tree, thus reducing the probability of serious disruption.

Based on drift

Code growth results from the structure of the search space: any stochastic search technique will tend to find the most common programs in the search space of the current best fitness – large programs are more common than small programs.

Problems & Open Questions
Bloat

Theories

Crossover Bias

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.

Problems & Open Questions
Bloat

Methods

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

Fitness Evaluation
Parametric Parsimony Pressure, Tarpeian

Selection for Breeding
Multi-Objective Optimization, Special Tournaments

Genetic Operators
Special Genetic Operators

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

Others
Code Editing, Dynamic Fitness, Other Types of GP

Problems & Open Questions
Bloat

Methods

Fitness 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)

Problems & Open Questions
Bloat

Methods

Selection for Breeding

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)

Problems & Open Questions
Bloat

Methods

Genetic Operators

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

Problems & Open Questions
Bloat

Methods

Selection for 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

Problems & Open Questions
Bloat

Methods

Selection for 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 in very hard regression problems)

Problems & Open Questions
Bloat

Methods

Selection for Survival

Operator Equalisation (Program Length Equalisation)

Based on the Crossover Bias theory, controls the distribution of program lengths by filtering which individuals are accepted in the population, based on their size/length and fitness.

Pros

Effectively eliminates bloat!!
Based on Dynamic Limits ideas, allows code growth when needed

Cons

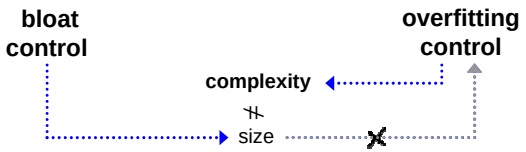
Two versions available: one is computationally expensive, the other is a slow learner

Problems & Open Questions
Overfitting and Complexity

"shorter solutions generalize better"

- Operator Equalisation eliminates bloat

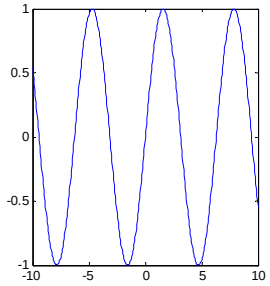
Problems & Open Questions
Overfitting and Complexity

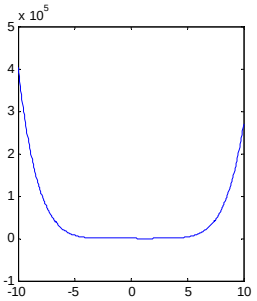


"shorter solutions generalize better"
simpler

- Operator Equalisation eliminates bloat, overfitting remains / increases
- Bloated solutions = Shorter effective code? Not in our problems
- For overfitting, size doesn't matter – complexity does

Problems & Open Questions
Mathematical vs Visual Complexity





$\sin(x)$

$$\frac{1}{3}x^6-\frac{2}{3}x^5+\frac{1}{3}x^4-\frac{2}{3}x^3+\frac{1}{3}x^2-\frac{2}{3}x+\frac{1}{3}$$

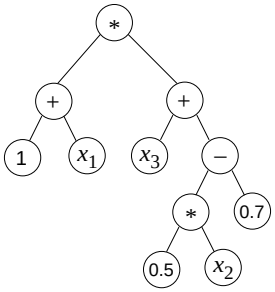
Problems & Open Questions
Multiclass Classification

Typical Limitation of GP

Each individual of tree-based GP returns only one value for each data sample.

How to perform, e.g., multiclass classification?

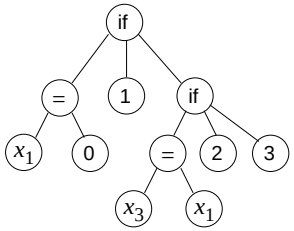
(note that a 2-class problem can be handled like a regression problem with application of a cutoff to the result)



Problems & Open Questions
Multiclass Classification

How to perform *n*-class classification?

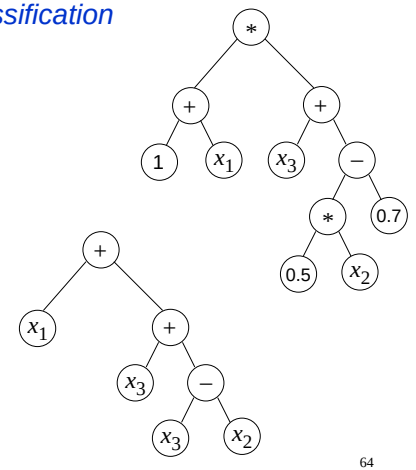
- Divide the problem into *n* 2-class problems
- Use if-then-else in the function set and constants in the terminal set as class identifiers (similar to decision trees)



Problems & Open Questions
Multiclass Classification

How to perform *n*-class classification?

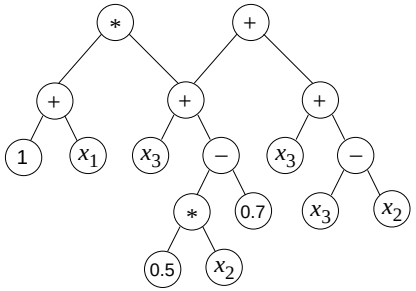
- Divide the problem into *n* 2-class problems
- Use if-then-else in the function set and constants in the terminal set as class identifiers (similar to decision trees)
- Use *n* trees to represent each individual



Problems & Open Questions
Multiclass Classification

How to perform *n*-class classification?

- Divide the problem into *n* 2-class problems
- Use if-then-else in the function set and constants in the terminal set as class identifiers (similar to decision trees)
- Use *n* trees to represent each individual
- Use trees with *n* roots (similar to Graph GP)



Project

EnviGP – Improving Genetic Programming for the Environment and Other Applications" (PTDC/EIA-CCO/103363/2008)

- Bloat
- Overfitting
- Complexity
- Interpretability of Solutions
- Multiclass Classification
- Applications in earth sciences and biomedical informatics

Project

Students welcome! 

- Sara Silva
- Susana Vinga
- PhD student
- MSc student

- Maria José Vasconcelos
- João M.N. Silva
- Marco Lotz

- Francisco B. Pereira
- MSc student

- Leonardo Vanneschi
- Mauro Castelli

Reading and Working Material

Books on Genetic Programming

A Field Guide to Genetic Programming, 2008
by Riccardo Poli, Bill Langdon, Nick McPhee (with contributions by John Koza)
http://www.gp-fieldguide.org.uk (PDF freely available)

Genetic Programming – an introduction, 1998
by Wolfgang Banzhaf et al.
Morgan Kaufmann

Genetic Programming – on the programming of computers by means of natural selection, 1992
by John Koza
MIT Press

Foundations of Genetic Programming, 2002
by William Langdon, Riccardo Poli
Springer

Reading and Working Material

Miscellaneous Links

Programação Genética – Darwin e o teu computador
http://academy.dei.uc.pt/page/artigos/proggenetica

Humies – Annual Awards for Human-Competitive Results Produced by Genetic and Evolutionary Computation
http://www.genetic-programming.org/hc2010/cfe2010.html

GPLAB – A Genetic Programming Toolbox for MATLAB
http://gplab.sourceforge.net

ECJ – Evolutionary Computation in Java
http://cs.gmu.edu/~eclab/projects/ecj/

Disciplus
http://www.rmltech.com/

Genetic Programming and Evolvable Machines
http://gpemjournal.blogspot.com/

For those of you still listening to me…

Thank you!

"*Genetic Programming is one of the few technological methods that has never killed a person*"

[Riccardo Poli, 2006]