Genetic Algorithms and Genetic Programming Lecture 9

Gillian Hayes

24th October 2006

Genetic Programming

- The idea of Genetic Programming
- How can we make it work?
- Koza: evolving Lisp programs
- The GP algorithm
- Example: symbolic regression
- Example: learning to plan
- Example: the Santa Fe trail
- Open questions for GP

Evolving Programs

Is it possible to create computer programs by evolutionary means?

- Let P_0 be a population of randomly generated programs, p_i
- For each p_i, run it on some input and see what it does. Rate it for fitness based on how well it does.
- Breed the fitter members of P_0 to produce P_1
- When happy with the behaviour of the best program produced, stop.

. . . but **how?**

Some Problems

- What language should the candidate programs be expressed in?
  C; Java; Pascal; Perl?
  Machine code?
- How can you generate an initial population?
- How can you run programs safely? Consider errors, infinite loops, etc.
- How can you rate a program for fitness?
- Given two selected programs, how can they be bred to create offspring?
- What about subroutines, procedures, data types, encapsulation, etc.
- What about small, efficient programs?

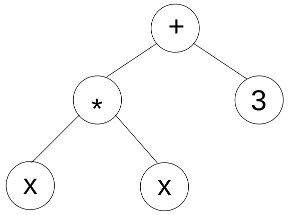
Koza: evolving LISP programs

Lisp: functional language: $f(x,y)$ is written $(f\ \ x\ \ y)$
$10-(3+4)$ is written $(-\ \ 10\ \ (+\ \ 3\ \ 4))$

Lisp programs can be represented as trees:

$f(x)=x^2+3$ $\qquad f(x)=(+\ \ (*\ \ x\ \ x)\ \ 3)$

$f(x)=$

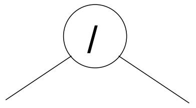


Here, $+$ and $*$ are function symbols (non-terminals) of arity 2, x and 3 are terminals. Given a random bag of both, we can make programs.

Random Programs and Closure

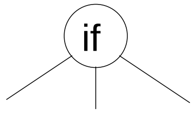
If we generate a random program:

How can we avoid an error?



Another random program:

How can we evaluate this?



All function calls return a result - **closure**.

Fitness Cases

How do we rate a program for fitness?

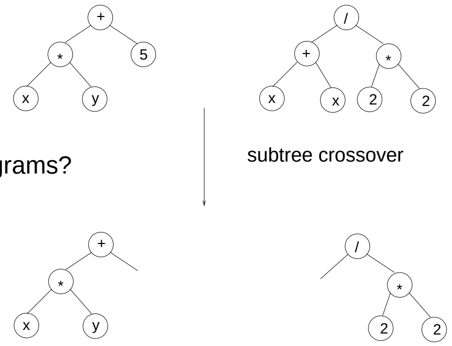
Answer: run it on some "typical" input data for which we know what the output should be. The hope is the evolved program will work for all other cases. (cf. Inductive Logic Programming, Supervised Machine Learning.)

Input:	x	y:	Output
	0	5	
	1	6	
	2	13	
Example: $y=f(x)$	4	69	
	8	517	
	16	4101	
	⋮	⋮	

Fitness: how close does p_i get to these perfect values?

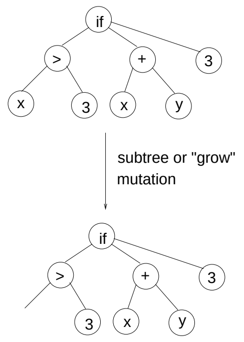
Crossover

How can we cross two programs?



Koza's original (1988-92) GP system used only crossover, to try to demonstrate that GP is "more than mutation"

Mutation



How can we mutate a program?

Lots of other forms of mutation are possible (see GAGP notes, Chapter 9, p.9:5).

GP Algorithm

1. Choose a set of functions and terminals for the program you want to evolve:
   e.g. non-terminals: if, $/$, $*$, $+$, $-$, $\sqrt{}$
   terminals: X, Y, -10, -9, $\ldots$, 9, 10
2. Generate an initial random population of trees of maximum depth d
3. Calculate the fitness of each program in the population using the chosen fitness cases.
4. Apply selection, subtree crossover (and subtree mutation) to form a new population.

Koza parameters: population size = 10,000 crossover rate = 0.9
generational algorithm fitness proportionate selection

GP Example

Symbolic regression on planetary orbits (Kepler's law). Given a set of values of independent and dependent variables, come up with a function that gives the values of the dependent variables in terms of the values of the independent variables.

Planet	A	P
Venus	0.72	0.61
Earth	1.00	1.00
Mars	1.52	1.84
Jupiter	5.20	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

Kepler's third law: Square of the period P of the planet proportional to cube of semimajor axis A

Assume $P = f(A)$. Then f can contain:

Non-terminals: $+, -, *, /, \sqrt{}$

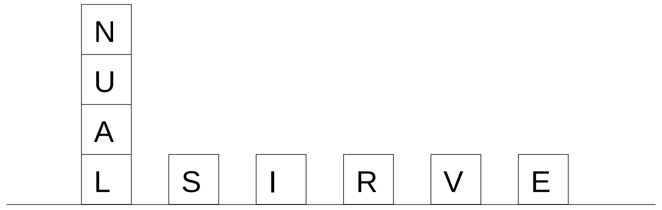
Terminals: A

- Can GP solve this problem? (yes)
- Can it solve any others? (yes) (see Koza paper for other examples)

Learning to Plan

A planning problem (Koza):

Initial state:



Goal state: a single stack that spells out the word "UNIVERSAL"

Koza's data set:

166 fitness cases
– different initial states
– same final state

Aim: to find a program to transform **any** initial state into "UNIVERSAL"

Learning to Plan

Terminals:

CS – returns the current stack's top block

TB – returns the highest correct block in the stack (or NIL)

NN – next needed block, i.e. the one above TB in the goal

Functions:

MS(x) – move block x from table to the current stack. Return T if does something, else NIL.

MT(x) – move x to the table

DU(exp1, exp2) – do exp1 until exp2 becomes TRUE

NOT(exp1) – logical not

EQ(exp1, exp2) – test for equality

Planning Results

Generation 0: (EQ (MT CS) NN)
– 0 fitness cases

Generation 5: (DU (MS NN) (NOT NN))
– 10 fitness cases

Generation 10:

```
(EQ (DU (MT CS) (NOT CS))
    (DU (MS NN) (NOT NN)))
```
– 166 fitness cases

Koza shows how to amend the fitness function for efficient, small programs: combined fitness measure rewards correctness AND efficiency (moving as few blocks as possible) AND small number of tree nodes (parsimony)

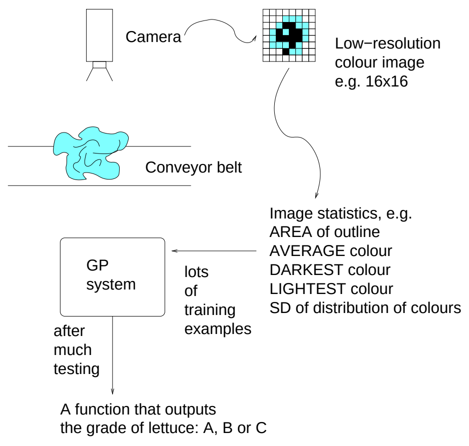
The Santa Fe Trail

Object: to evolve a program which eats all the food on a trail without searching too much when there are gaps in the trail.
Sensor can see the next cell in the direction it is facing
Terminals: MOVE, LEFT, RIGHT
Functions: IF-FOOD-AHEAD, PROGN2, PROGN3
Program:

```
(if-food-ahead move
    (progn3 left
        (progn2 (if-food-ahead move right)
            (progn2 right (progn2 left right)))
        (progn2 (if-food-ahead move left)
            move)
    )
)
```

Fitness: amount of food collected in 400 time steps (say).

GP: a practical example


Camera
Low–resolution colour image e.g. 16x16
Conveyor belt
Image statistics, e.g.
AREA of outline
AVERAGE colour
DARKEST colour
LIGHTEST colour
SD of distribution of colours
GP system
lots of training examples
after much testing
A function that outputs the grade of lettuce: A, B or C

Grading lettuces . . . uses proprietary form of GP, by Evis Technologies GmbH, Vienna.

Much faster and more accurate than humans.

GP: Some Other Examples

- Predicting electricity demand (suppliers can buy from each other, e.g. see www.elexon.co.uk)
- Generation of financial trading rules (e.g. see www.algorithmics.com)
- Designing new electronic circuits
- Data mining: creating functions that “fit” well to data
- Controllers for simulated creatures, predator-prey

Open Questions

- Will the technique scale up to more complex problems and larger programs?
- Will GP work if the function and terminal sets are large?
- How well do the evolved programs generalise?
- How can we evolve nicer programs?
  - size
  - efficiency
  - correctness
- What sort of problems is GP good at/ not-so-good at?
- How does GP work? etc.

Reading: (skim) Koza 1990 paper linked to web page
GAGP lecture notes chapter 9