Genetic Algorithms and Genetic Programming

Lecture 8: (20/10/09)

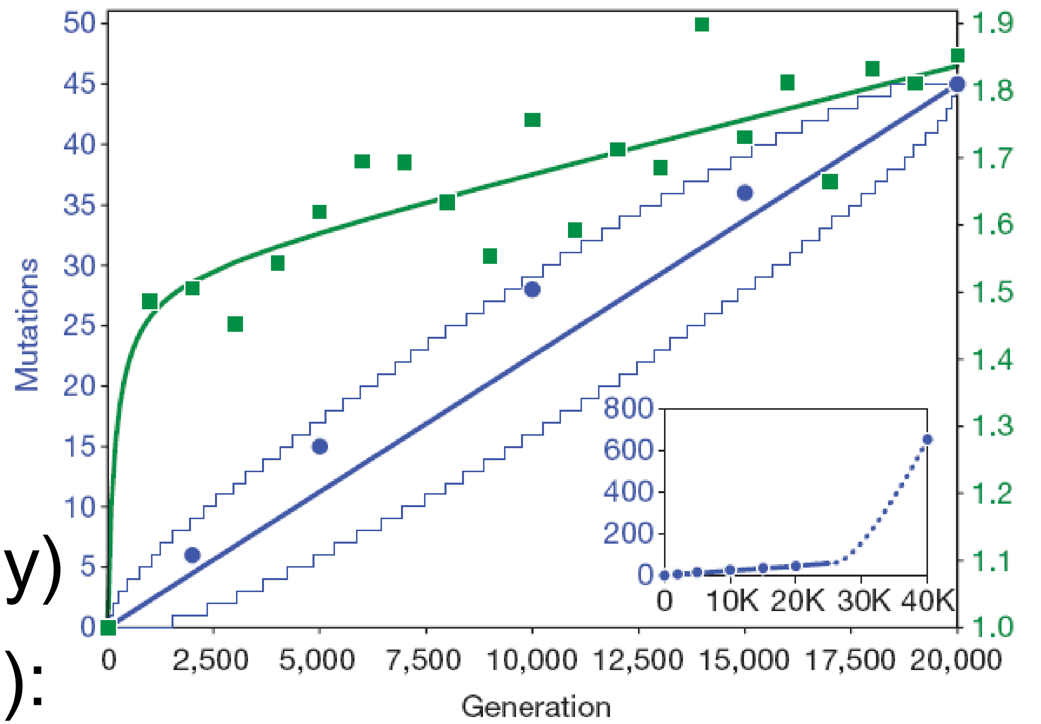
Genetic programming

School of informatics

Michael Herrmann

michael.herrmann@ed.ac.uk, phone: 0131 6 517177, Informatics Forum 1.42

Recent Progress in Evolution Theory



- 40000 generations (1988-2009: 21 years!)
- Initial population: 12 strains of E. coli
- Constant conditions (restricted glucose supply)
- first half (20000 generat.): 45 mutations often related to life span and efficience
- second half (20000 generat.): 653 mutations in some strains but without any obvious effects on fitness
- Conclusions: Relations between mutation rate and fitness are more complex than expected

J. Barrick et al. (2009) Genome evolution and adaptation in a long term expeiment with Escherichia coli. Nature 08480.

Overview

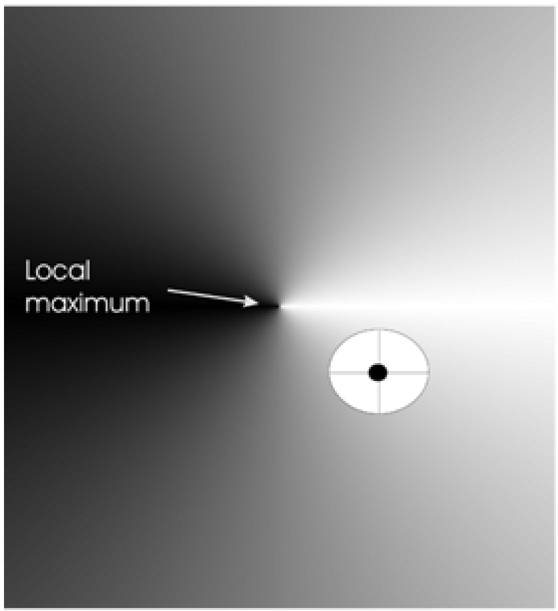
1. Introduction: History
2. The genetic code
3. The canonical genetic algorithm
4. Examples & Variants of GA
5. The schema theorem
6. Hybrid algorithms
7. Evolutionary robotics
8. Genetic Programming
9. GP: Examples and applications

Evolutionary algorithms

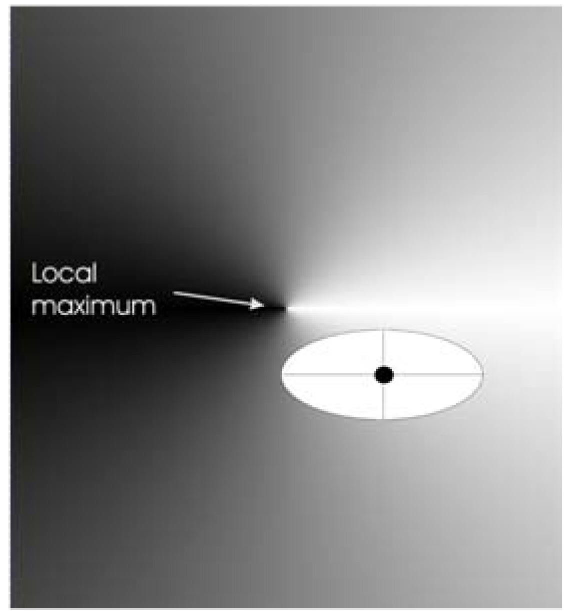
	genotype (encoding)	mutation/ crossover	phenotype (applied to)
Genetic algorithm	strings of binary or integer numbers	e.g. 1-point for both with p_m, p_c	optimization or search of optimal solutions
Genetic programming	strings of binary or integer numbers	e.g. 1-point for both with p_m, p_c	computer programs for a computational problem
Evolutionary programming	real numbers	mutation with self-adaptive rates	parameters of a computer program with fixed structure
Evolution strategy	real numbers	mutation with self-adaptive rates	optimization or search of optimal solutions

Multidimensional mutations in ES

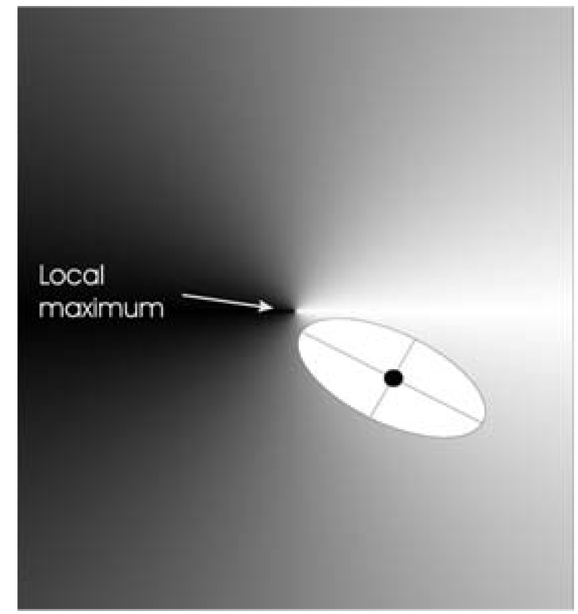
Uncorrelated mutation with one σ



Uncorrelated mutation with L σ_i's



Correlated mutations



Correlated mutations: $y = x + \mathcal{N}(0, C')$

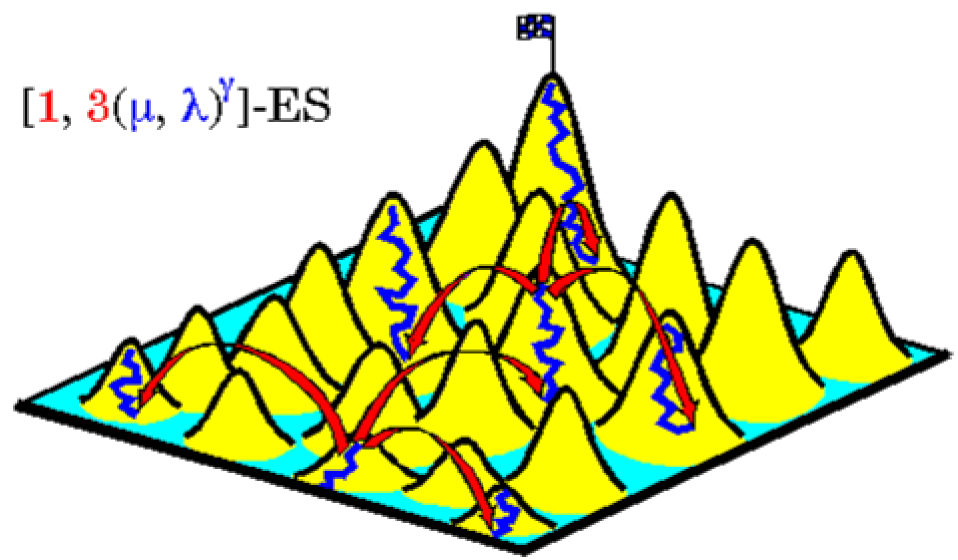
x stands for the vector $(x_1, \ldots, x_n)$

C' is the covariance matrix C after mutation of the σ values

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing. Evolution Strategies

Nested Evolution Strategy

- Hills are not independently distributed (hills of hills)
- Find a local maximum as a start state
- Generate 3 offspring populations (founder populations) that then evolve in isolation
- Local hill-climbing (if convergent: increase diversity of offspring populations)
- Select only highest population
- Walking process from peak to peak within an "ordered hill scenery" named *Meta-Evolution*
- Takes the role of crossover in GA



http://www.bionik.tu-berlin.de/intseit2/xs2mulmo.html

Genetic Programming

- Genetic programming now routinely delivers high-return human-competitive machine intelligence.
- Genetic programming is an automated invention machine.
- Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology.

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)
- If happy with the behaviour of the best program produced then stop.
- . . . but how?

How?

- What language should the candidate programs be expressed in?
- C, Java, Pascal, Perl, Lisp, 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?

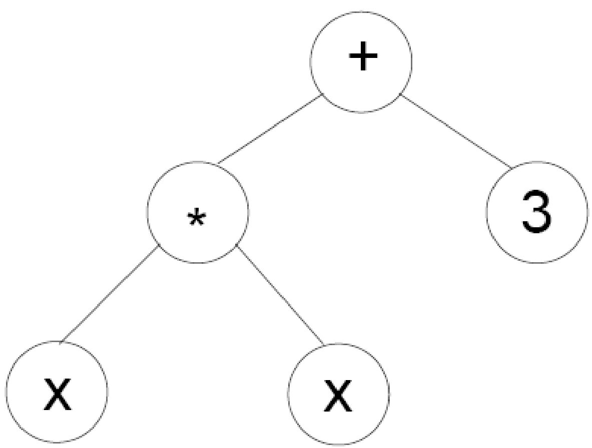
Koza: evolving LISP programs

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:

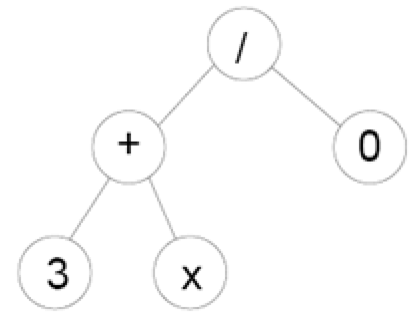
$f(x) = x^2 + 3$ $\qquad f(x) = (+\ \ (*\ \ x\ \ x)\ \ 3)$

$f(x) =$



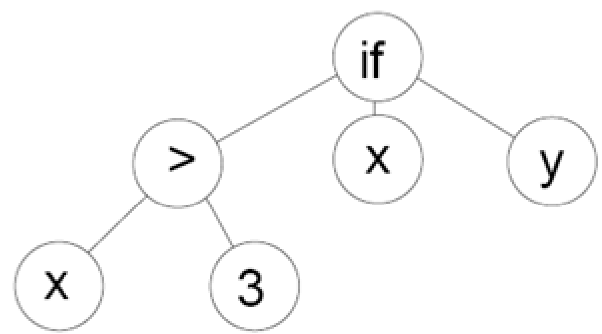
Here, $+$ and $*$ are function symbols (non-terminals) of arity 2, x and 3 are terminals.
Given a random bag of both, we can make programs.

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**.

Defaults: Choose a reasonable set of symbols, ignore arguments, double arguments, return max_n etc.

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.

y=f(x)	Input: x	p_i: y	Output (supposed)
	0	5	5
	1	6	9
	2	13	24
	4	69	157
	8	517	4079
	16	4101	405
	...	...	...

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

Fitness Function

For the fitness function we could use

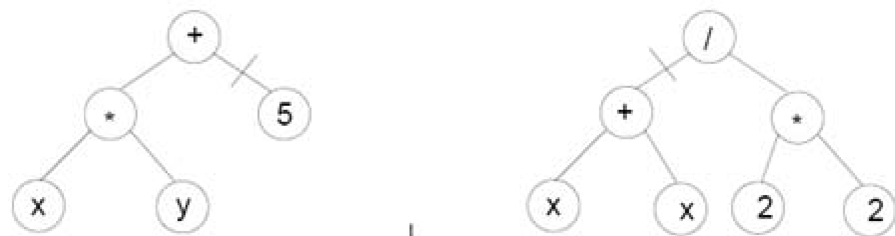
$$\text{raw} = \Sigma_{\text{fitness cases}} \mid \text{GP}_j - y_j \mid$$

$$\text{adjusted} = \frac{1}{1 + \text{raw}}$$

$$\text{normalised} = \frac{\text{adjusted}_i}{\Sigma_j \text{adjusted}_j}$$

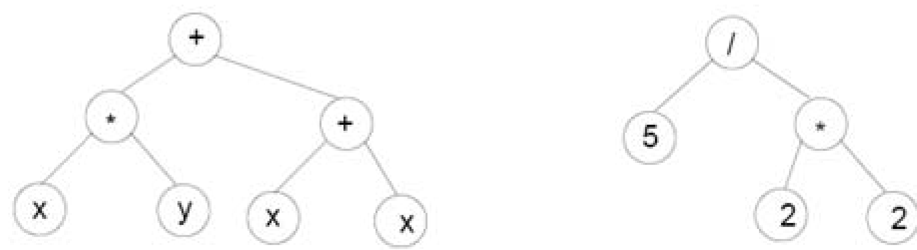
where *j* are the fitness cases, so most fit is 1, least fit is 0.

Crossover



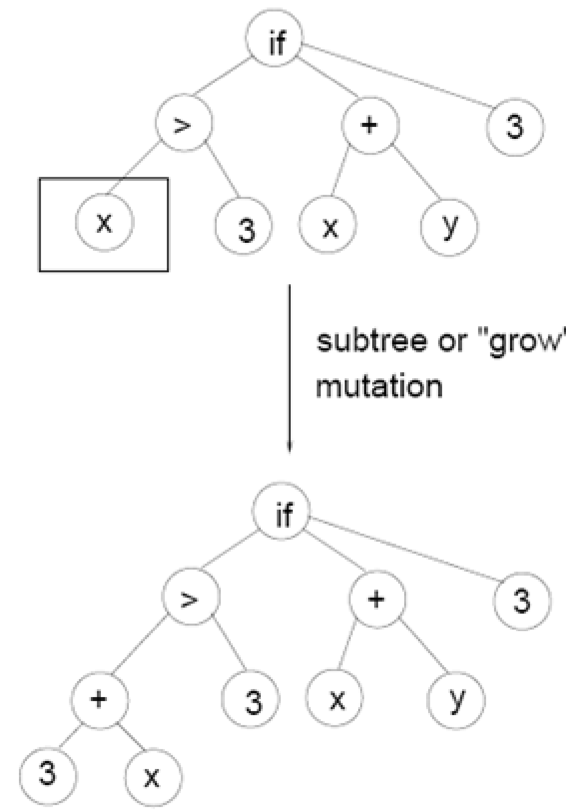
How can we cross two programs?

subtree crossover



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, e.g. hoist, shrink

shrink: replace a subtree by one of its terminals
hoist: use only a subtree as a mutant

or: vary numbers, exchange symbols, exchange subtrees, …

GP Algorithm

1. Choose a set of functions and terminals for the program you want to evolve:
   - non-terminals e.g.: if, /,* , +, −, sqrt, <, >…
   - terminals e.g.: x, y , −10, −9, . . . , 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.

Example parameter values: population size = 10000
crossover rate = 0.9
Selection: Fitness proportionate

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.

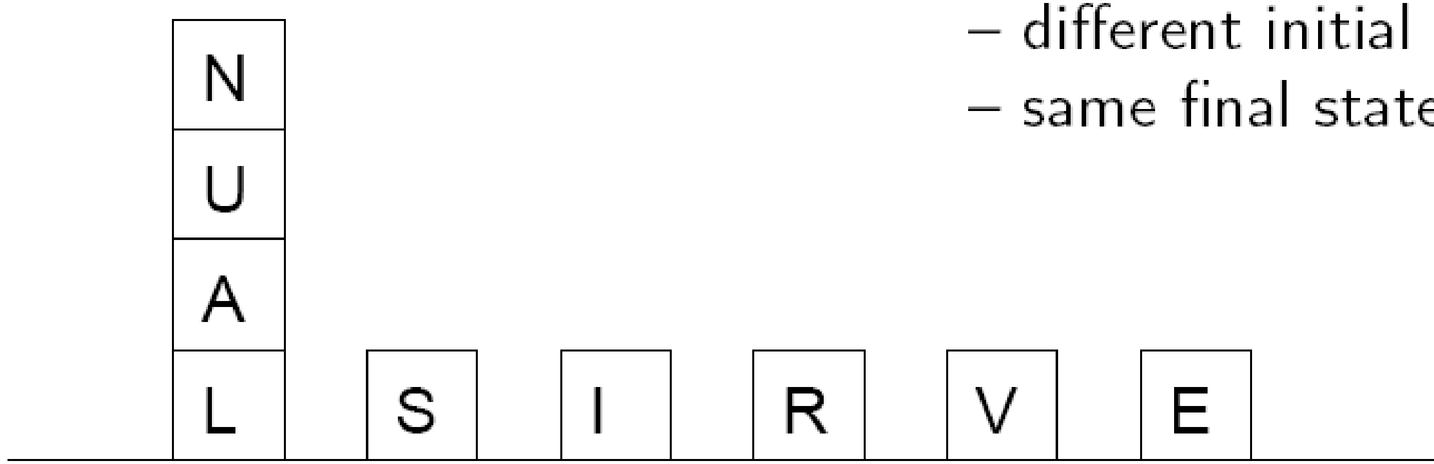
Planet	A	P
Venus	0.72	0.61
Earth	1.00	1.00
Mars	1.52	1.84
Jupiter	5.20	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

Kepler's third law: Square of the period P of the planet proportional to cube of semimajor axis A ($P = A^{3/2}$).

Learning to Plan

A planning problem (Koza):

Initial state:



Koza's data set:

166 fitness cases

– different initial states

– same final state

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

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

Objective:

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

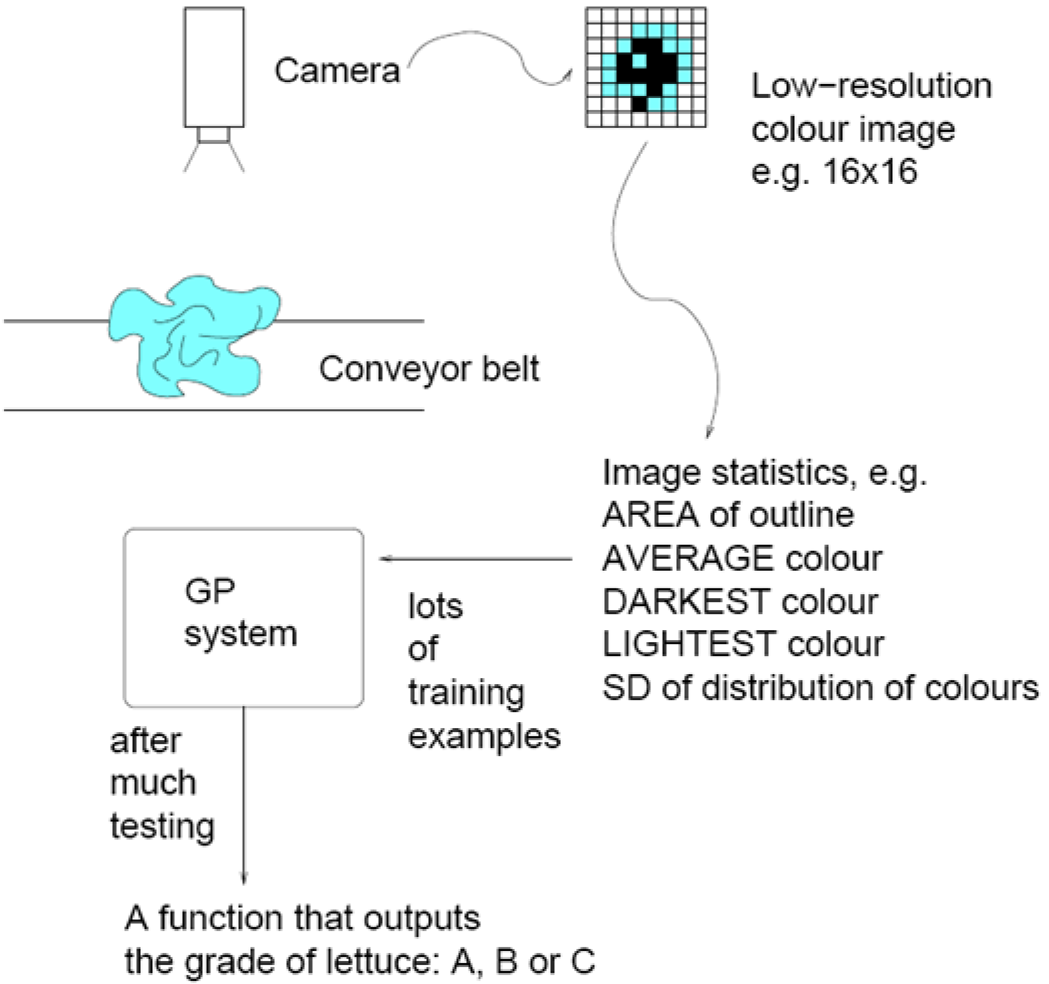
Functions: IF-FOOD-AHEAD, PROGN2, PROGN3

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



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
- Generation of financial trading rules
- Designing new electronic circuits
- Data mining: Creating functions that “fit” well to data
- Controllers for simulated creatures, predator-prey

see: http://www.genetic-programming.org
http://www.geneticprogramming.us

Open Questions/Research Areas

- Scaling up to more complex problems and larger programs
- Using large function and terminal sets.
- 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

- J. Koza 1990, especially pp 8–14, 27–35, 42–43 (paper linked to web page)

Outlook

- More Genetic Programming

Evolving neural networks for control

Grammars and Robotics

- Grammatical encoding of the linkage matrix (here for XOR)

$$S \rightarrow \begin{matrix} A & B \\ C & D \end{matrix} \rightarrow \begin{matrix} c & p & a & a \\ a & c & a & e \\ a & a & a & a \\ a & a & a & b \end{matrix} \rightarrow \begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

(S A B C D | A c p a c | B a a a e . . .)

Generate linkage matrix from the grammar. If at the end of rewriting there are still non-terminal nodes, that node is "dead" – not connected.

• Develop chromosome (genotype) into network (phenotype) and train for fixed no. of training episodes.

• Fitness = error at end of training

Problems with direct encoding

Fixed connections: as size of matrix grows, chromosome size grows

Can't encode repeated patterns, esp. with internal structure

Takes a long time to generate high-performing networks

Advantages of grammatical encoding

Can represent large connectivity matrices in compact form

Shorter encoding, faster search

Variable topologies including recurrent connections

Better on encoder/decoder problem than direct encoding

Evolving Neural Network Behaviours

- Previous examples rely on *training data*
- What if we haven't got any?
- Example: a neural network which controls a mobile agent which is trying to achieve some goals in a dynamic environment.
- No good example of behaviour is available; or we wish to try a range of possible behaviours to see which is best.
- A fitness function is available based on goal achievement.

Evolving Neural Network Behaviours

General approach:

- Decide on how to represent inputs to and outputs from the neural network.
- Decide on a neural network architecture: might need to try a range of possibilities.
- Decide on a simulation which tests the NN's behaviour.
- Decide on a fitness function which tests how well the NN did in the simulation.
- All the usual GA stuff: chromosome representation, crossover, mutation, population size, etc.

Example: Evolving Communication

This is an example from Artificial Life: the study of computer generated "life" forms. (Matthew Quinn, University of Sussex)

- Khepera robots controlled by evolved neural networks
- Group task: robots move together as far as possible – like dancing
- 8 sensor nodes, 4 motor nodes, hidden nodes
- Evolved thresholds, weights, decay parameters, size, connectivity of network
- **Co-evolution**: select two robots from population, rate them for fitness *as a pair*
- Initial result: leaders and followers emerge
- Only get a working pair 50% of the time, BUT....

Example: Evolving Communication

- After a while a new single species emerges
- This behaviour uses communication based on simple movement:
  - both agents (A and B) rotate anti-clockwise
  - one agent (B) becomes aligned first and moves towards the other agent
  - agent B moves backward and forward while staying close to A
  - when A becomes aligned, it becomes the leader: it reverses its direction and is followed by B
- Very similar to movement communication used in social insects (e.g. dancing in honey bees)