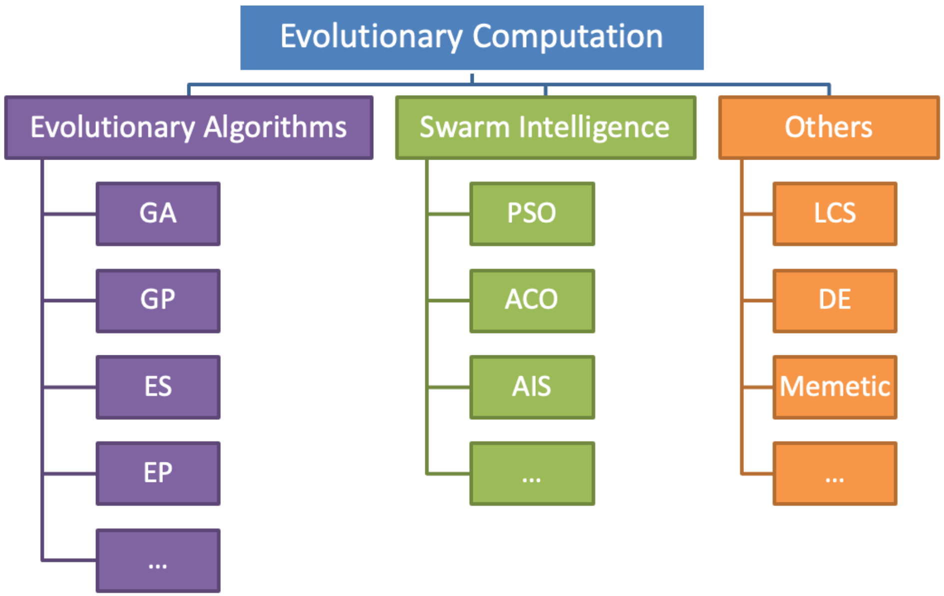
Evolutionary Computation and Learning

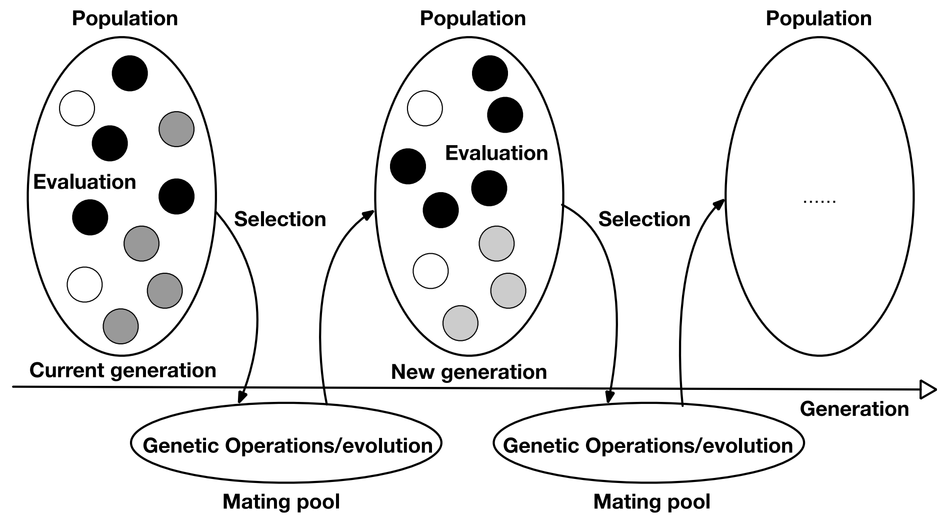
Genetic Programming 1:

Basic, Regression and Classification

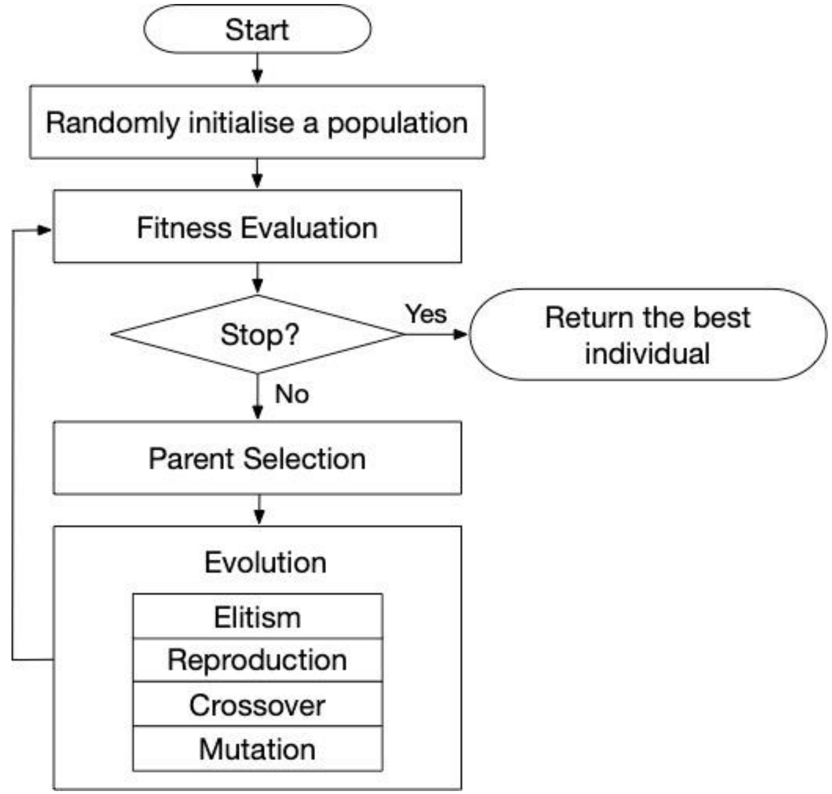
Represented by: Dr. Vahid Ghasemi

Review---Evolutionary Computation





Review---Evolutionary Algorithms



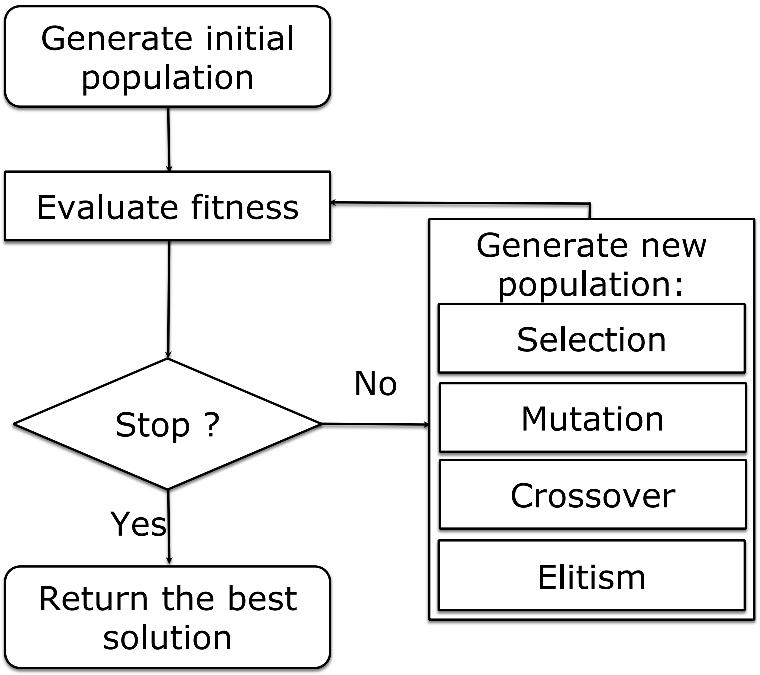
– Start from a randomly initialised population
– Improve the qualities of individuals generation by generation
– Such as GA and GP

Outline

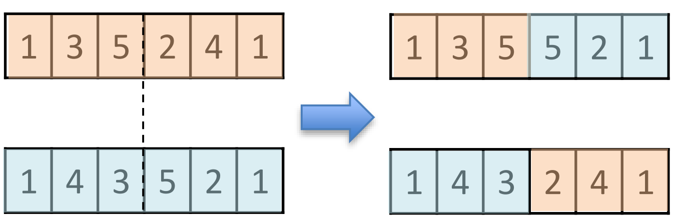
- From GA to GP
  - Representation
  - Terminals and Functions
  - Program Generation
  - Genetic Operators
  - Fitness Function
- GP for Regression and Classification

Review---A Simple GA

- Initialise a population of chromosomes
- **Repeat until** stopping criteria are met:
  - Fitness evaluation of each individual
  - Construct an empty new population
  - Do elitism (copy top individuals)
  - **Repeat until** the new population is full:
    - Select **two** parents from the population
    - Apply crossover to the two parents to generate two children
    - Each child has a probability (mutation rate) to undergo mutation
    - Put the two (mutated) children into the new population
  - **End Repeat**
- **End Repeat**

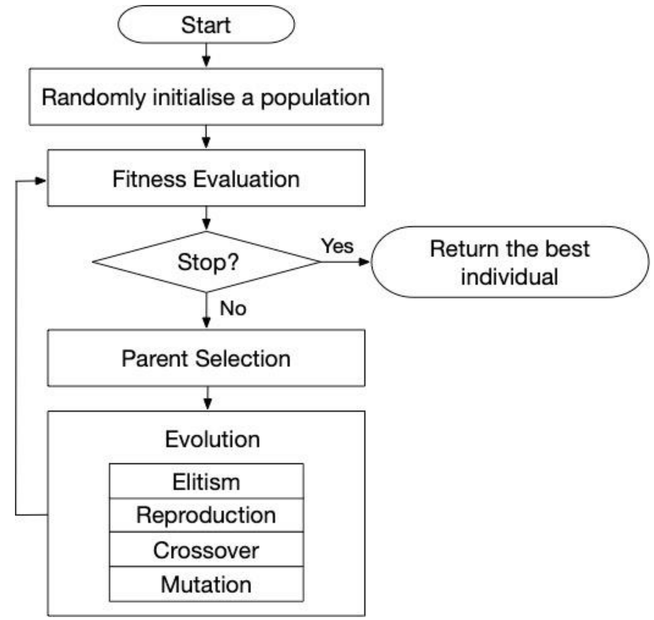


Crossover:

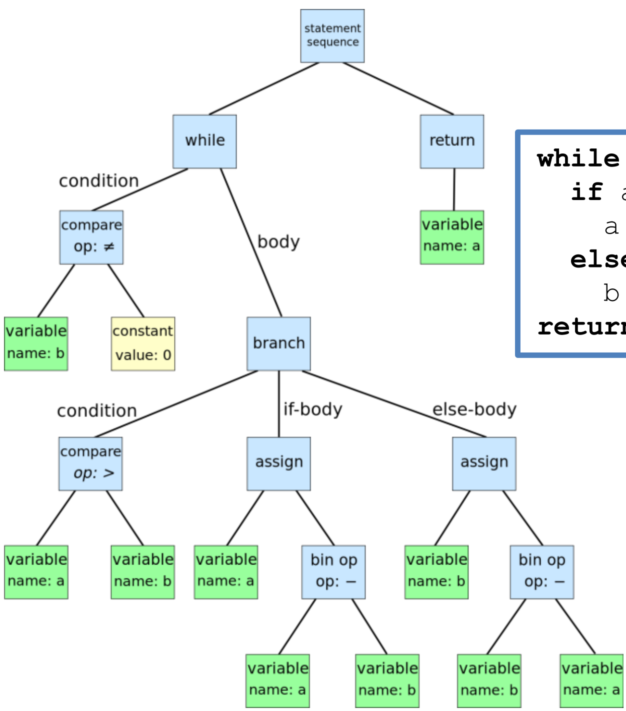


Genetic Programming (GP)

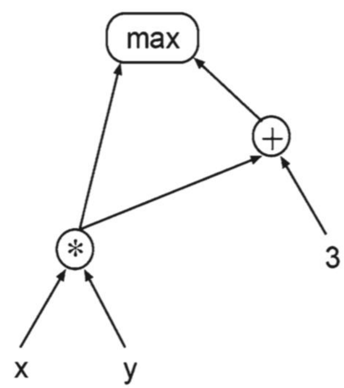
- A type of evolutionary algorithm
  - Evolve **computer programs** rather than solutions



- **Representation** of computer programs
  - **Tree-like**, graph-like, linear, …




```
while b ≠ 0
  if a > b
    a := a − b
  else
    b := b − a
return a
```

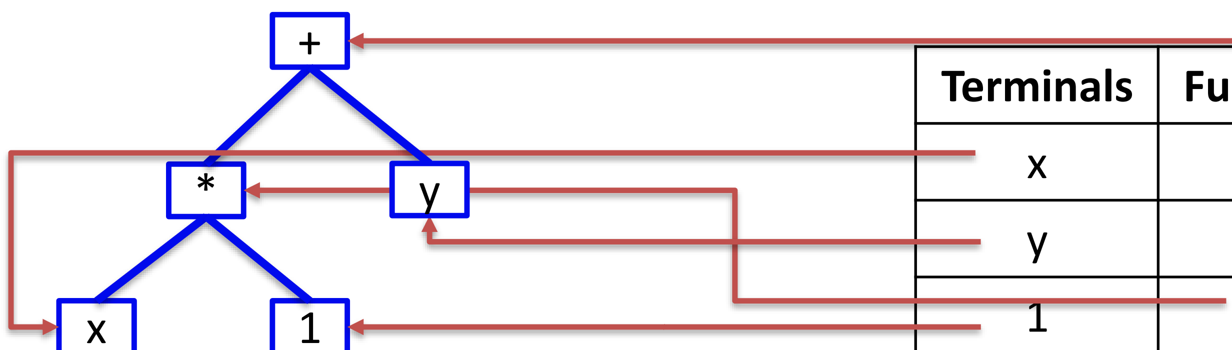


$\max(x * y, x * y + 3)$

```
r[3] = r[1] / 1.3;
r[1] = r[2] * -5.5;
r[0] = sqrt(10);
r[3] = r[1] + r[1];
r[1] = log(r[3]);
r[1] = r[3] >= r[0];
r[0] = abs(r[2]);
r[0] = if r[1] < 0 then
          r[0] else r[3];
```

GP Program Generation

- **Individual generation** (Tree-based representation)
  - **Terminal set**: inputs of the program and constants, no argument, form the leaf nodes,
  - **Function set**: operators to the inputs and intermediate results of the program (e.g. +, -, max, …), form the non-leaf nodes
- Start from the **root** node
- For each node, **randomly sample from the terminal/function set**
  - If sampling from the terminal set, then stop this branch
  - If sampling from the function set, create the child nodes, and **recursively sample** the child nodes



$x * 1 + y$

Terminals	Functions
x	+
y	-
1	*
	/

GP Program Generation

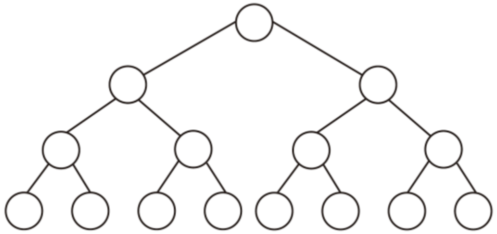
- **Parameters**
  - Min and Max depth of the generated tree
    - The depth of a node is the number of nodes traversed from the root node to it
    - depth(root) = 0
  - Max program size (number of nodes) in the generated tree
  - Max depth is more commonly used

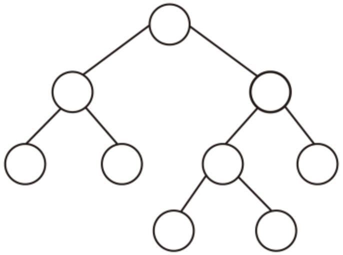
- Full Method
  - Start from the root node (depth 0)
  - If the depth of the generated node is smaller than the max depth, sample from the function set ONLY
  - If the depth of the generated node equals the max depth, sample from the terminal set ONLY
  - This ensures that full, entirely balanced trees are constructed.

GP Program Generation

- Grow Method
  - Start from the root node (depth 0)
  - If the depth of the generated node is smaller than the max depth, sample from both the terminal set and function set
  - If the depth of the generated node equals the max depth, sample from the terminal set ONLY
  - If a terminal is selected, the branch with this terminal is terminated and the generation process moves on to the next non-terminal branch in the tree

- Ramp-half-and-half
  - Mainly used for GP population initialisation
  - Half population is generated by grow, the other half by full



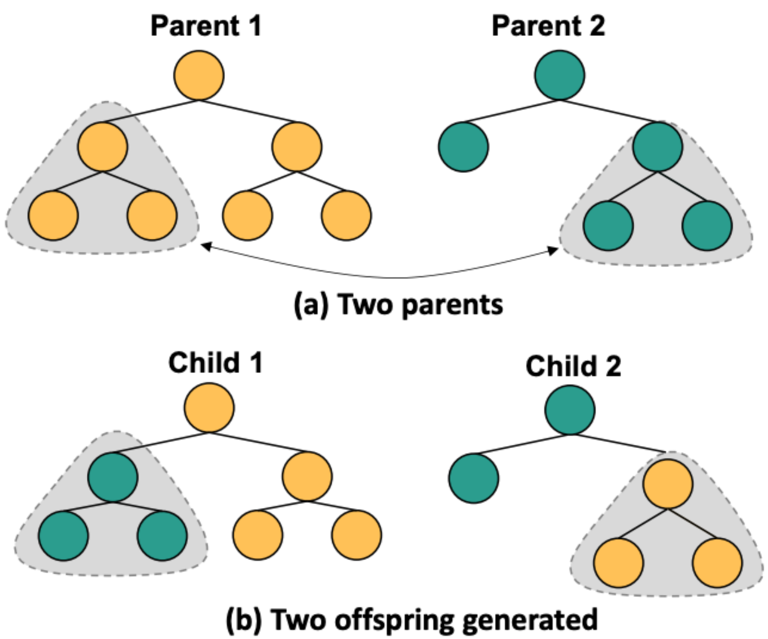


GP Terminal and Function Selection

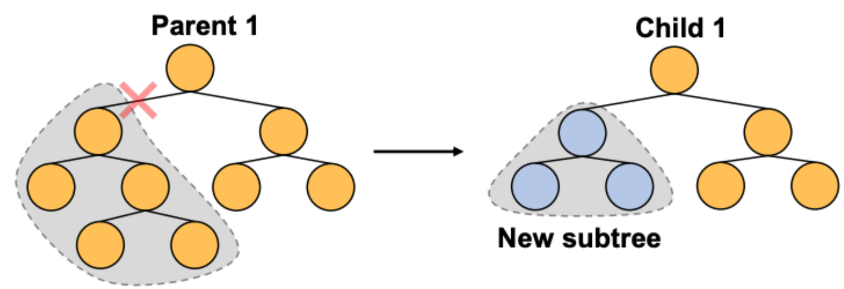
- **Proper terminal and function sets** are critical for the success of GP
  - Sufficiency
  - Closure
- Sufficiency: There must be some combination of terminals and function symbols that can solve the problem
  - If the target program is to calculate $\log(x) + 2^y$, but the function set is $\{+, -, *, /\}$, then not sufficient
- Closure: Any function can accept any input value returned by any function (and any terminal).
  - If the function set includes $AND(boolean, boolean)$ and $+$, then not closure, since we may have AND taking the real-value inputs.

Genetic Operators in GP

- **Genetic Operators** (tree-based)
  - Crossover: randomly select a sub-tree from each parent, and swap them
  - Mutation: randomly select a sub-tree from the parent, and replace the sub-tree with a randomly generated sub-tree (e.g., grow)
  - Reproduction: copy the parent directly



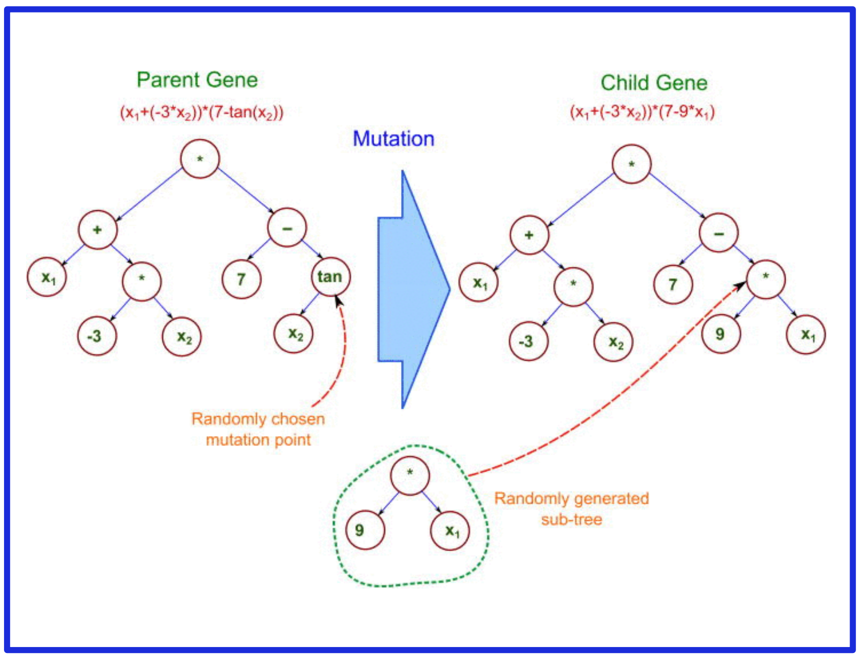
Crossover

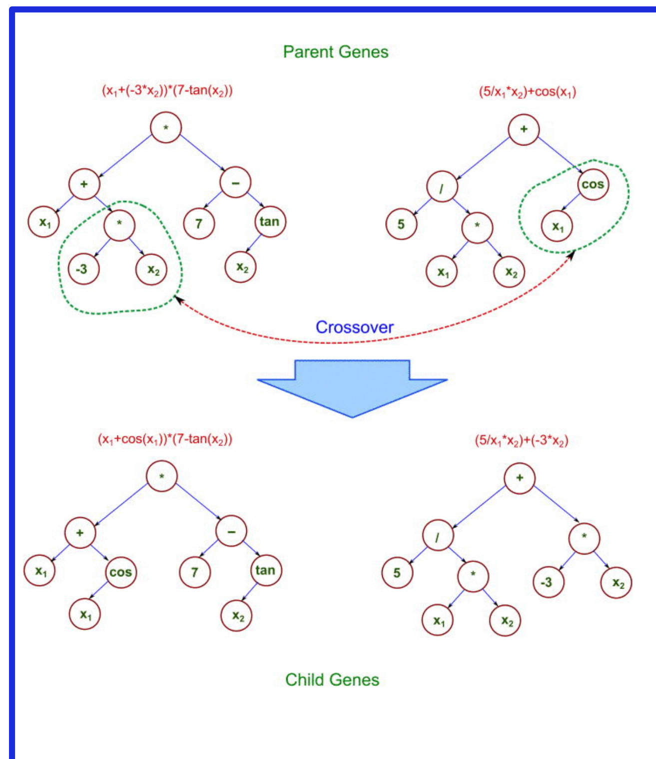


Mutation

Genetic Operators in GP

- **Genetic Operators** (tree-based)
  - Crossover: randomly select a sub-tree from each parent, and swap them
  - Mutation: randomly select a sub-tree from the parent, and replace the sub-tree with a randomly generated sub-tree (e.g., grow)
  - Reproduction: copy the parent directly





Parameters in GP

- Parameters in Crossover
  - Probabilities of each sub-tree to be selected (terminals, non-terminals)
- Parameter in Mutation
  - Probabilities of each sub-tree to be selected
  - Parameters for generating the new sub-tree
    - Full/Grow
    - Min/Max depth
- Crossover/Mutation/Reproduction rates
  - These rates should add up to 100%
- Other parameters
  - Population size, stopping criteria
  - Min/Max depth/program size
  - Parent selection
  - …

A Basic GP Algorithm

- Initialise a GP population by Ramp-Half-and-Half
- **Repeat until** stopping criteria is met:
  - Evaluate each GP individual;
  - Construct an empty offspring population;
  - **Repeat until** the offspring population is full:
    - Elitism (copy top individuals)
    - Parent selection (2 for crossover, 1 for mutation and reproduction, typical size tournament selection)
    - Generate offspring by crossover/mutation/reproduction and add into the offspring population
  - **Return** the best GP individual

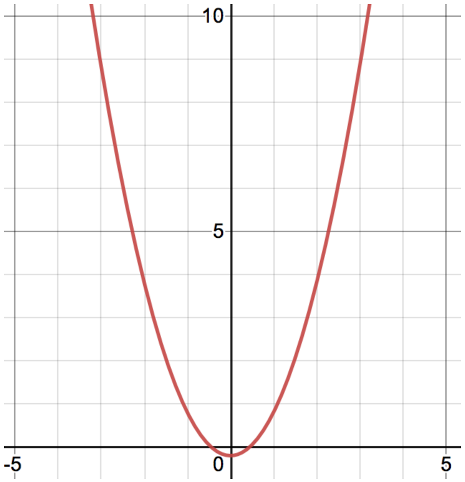
Typical GP Parameter Setting

- **Population size**: 1000
- **Generations**: 50
- **Elitism**: top 5/10
- **Crossover/Mutation/Reproduction rates**: 85%/15%/0%
  - Can use Elitism of top 1, and ~5% reproduction
- **Initialisation min/max depth**: 2/6
- **Max depth**: 17 (or smaller depending on your need)
- Grow method for generating mutation sub-trees, min/max depths are 2 and 6

GP for Symbolic Regression

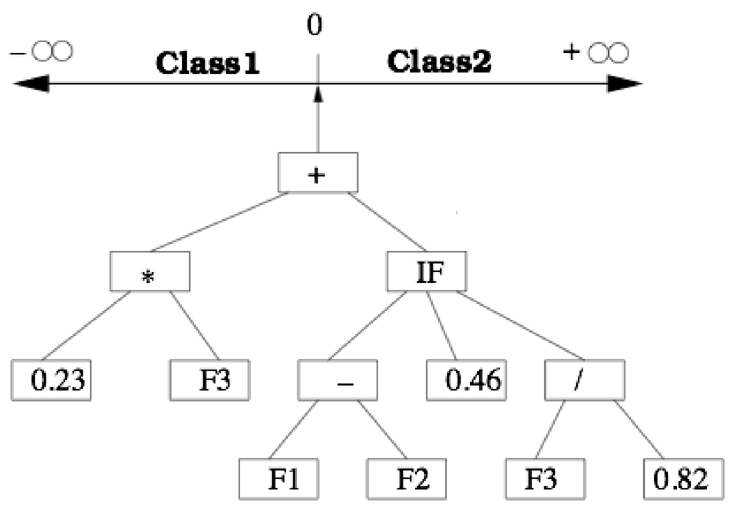
- Fitness Cases:
  - The patterns or examples in other learning paradigms such as neural networks are called fitness cases in GP.
- Two different sets of fitness cases: training cases for learning and test cases for performance evaluation.
  - Define terminal set $\{x_1, x_2, \dots, x_n, w\}$
  - Define function set $\{+, -, *, /, \log, \ \dots\}$
  - Define the fitness function
    - Mean squared error $\sum_i (gp(\overrightarrow{x_i}) - y_i)^2$
    - Can consider regularisation (generalisation performance)

x	y
1	0.8
2	3.8
3	8.8
…	…



GP for Binary Classification

- Given a set of training data (feature vector and class label)
- Evolve GP program in the same way as regression
- Translate the final real-valued output into class prediction
  - Fixed/Predefined boundary? Fixed order of class?
  - Can we change the boundary after training and during test?

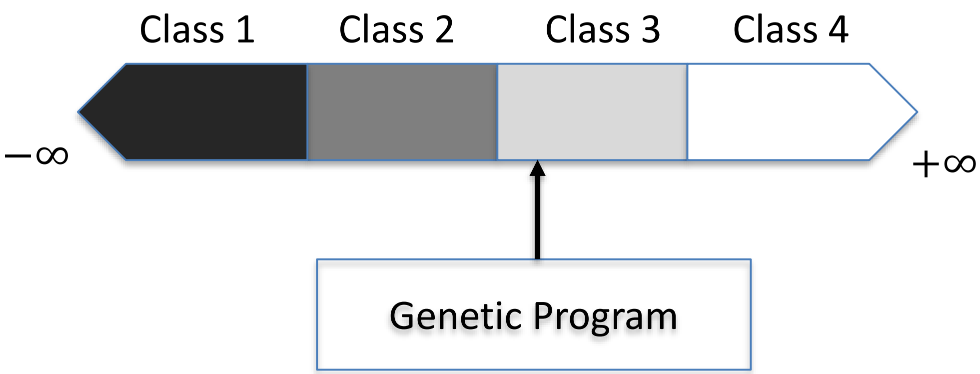


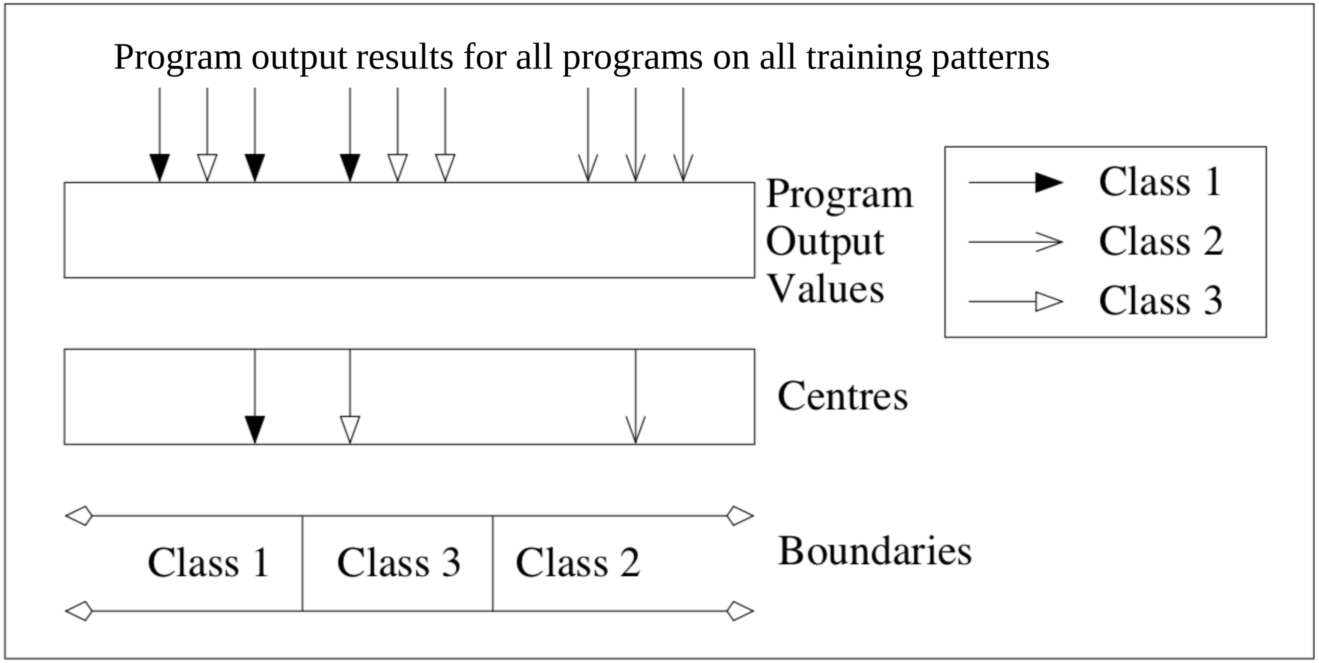

```
Genetic Program: (+ (* 0.23 F3)
                    (IF (- F1 F2) 0.46 (/ F3 0.82))
                 )
```

```
if ProgOut < 0  then  Class1  else   Class2;
```

GP for Multi-Class Classification

- Multiple boundaries
  - Fixed/Predefined boundaries?
  - Fixed order of classes?
  - Interval for each class?
- Dynamic boundaries
  - Set the boundary based on the centre of different classes (average of the outputs)

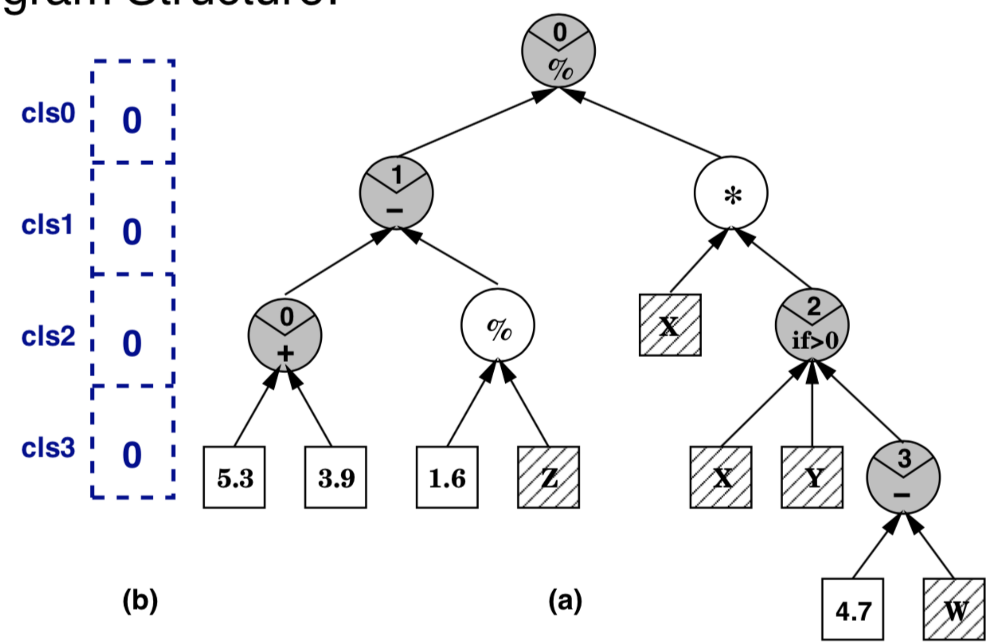




GP for Multi-Class Classification

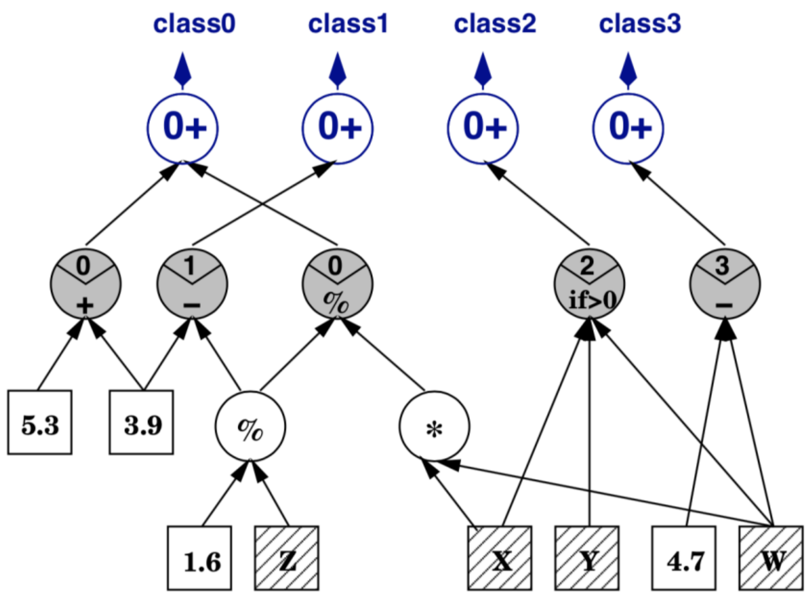
- Multiple trees, each for a class (kinds of converting multi-class classification to binary classification)
  - Tree 1: Class 1 vs non-class 1
  - Tree 2: Class 2 vs non-class 2
  - …
- A single tree with multiple outputs
  - Voting from multiple outputs

Program Structure:

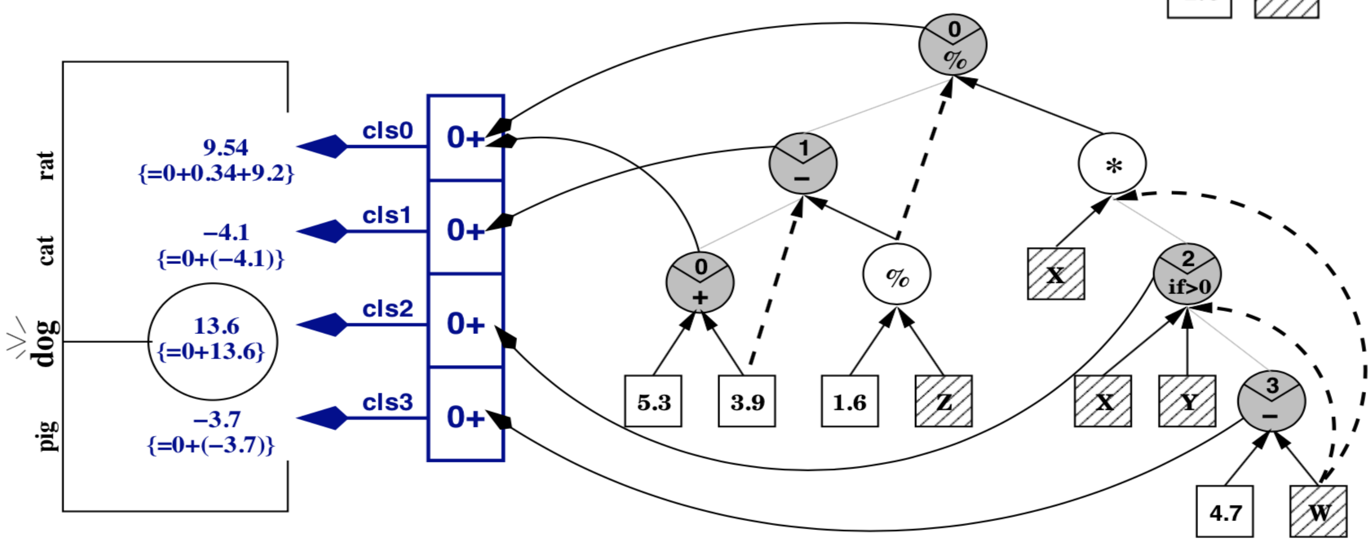


GP for Multi-Class Classification (Modi Tree, 2004)

- Calculate the outputs
  - Adjust the inputs of the nodes
- Vote for the classes
- Predict for the class with most votes
- Like neural network!



- Consider input feature vector:
  - [V,U,W,X,Y,Z]=[0.6,5.7,8.4,2.8,13.6,0.2]

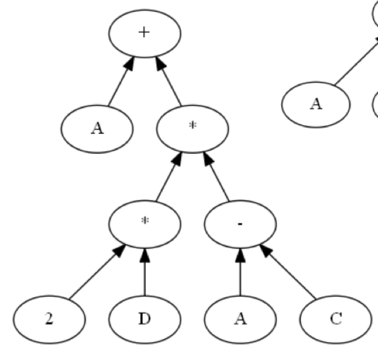


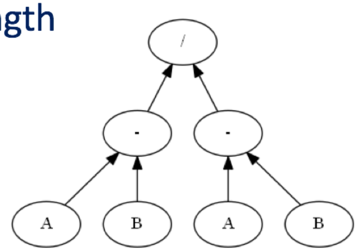
GP vs GA

Genetic Algorithm	Genetic Programming
➢Bit string representation	➢Tree-like structure
➢Fixed in length	➢Vary in length
➢Inflexible	➢Flexible

1	0	1	1	0	1	0	1	1	0

0	0	0	1	0	1	1	1	0	1





Summary

- Genetic Programming
  - Terminals and functions
  - Program generation
  - Crossover and mutation
  - Fitness function
- GP for symbolic regression
- GP for classification
  - Binary
  - Multi-class

➢ Next week

- Genetic Programming 2: Automatic Algorithm/Heuristic Design