Introduction to Artificial Intelligence

COMP307/AIML420

Evolutionary Computation 2: Genetic Programming (GP)

Dr Andrew Lensen

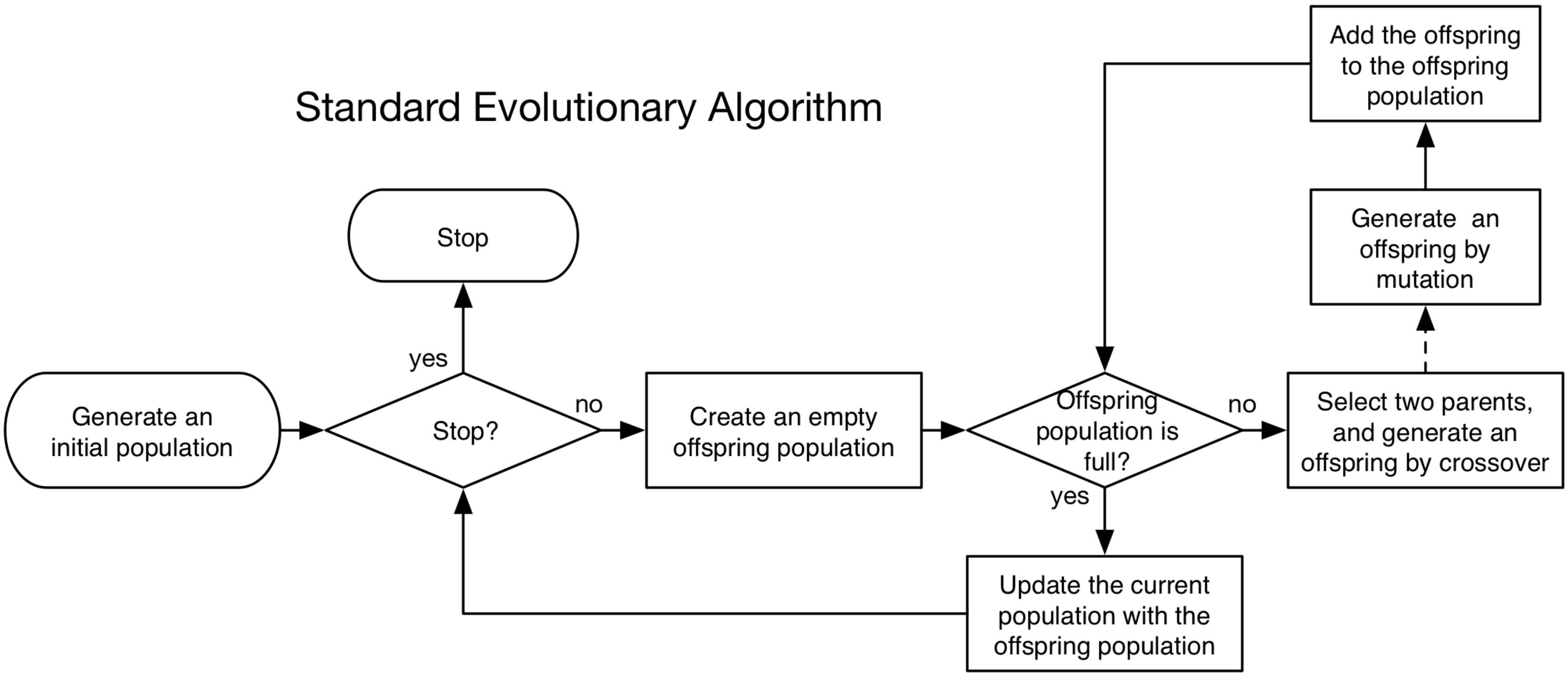
Andrew.Lensen@vuw.ac.nz

Outline

- GP representation
- Terminals and functions
- Program generation
- Genetic operators
- Fitness functions
- A basic GP algorithm
- Tackling a problem with GP

Genetic Programming

- GP follows the process of a standard evolutionary algorithm



Genetic Programming

- **Genetic programming (GP)** inherits properties from EC techniques (e.g. GAs) and automatic programming

- GP uses a similar evolutionary process to the general evolutionary algorithms (e.g. GAs)
  - GA uses bit strings to represent solutions;
    GP uses tree-like structures that can represent computer programs
  - GA bit strings use a fixed length representation;
    GP trees can vary in length
  - The term GP originates from the notion that computer programs can be represented as a tree-structured genome

- Automatically learning a set of computer programs for a particular task is a dream of computer scientists
- GP is such a technique that can help us achieve this goal

LISP S-Expressions

- Form of a LISP function:
  (*FUNCTION-NAME ARG1 ARG2 ARG3, …*)

  The arguments are evaluated, the function is applied to the arguments and then the output returned

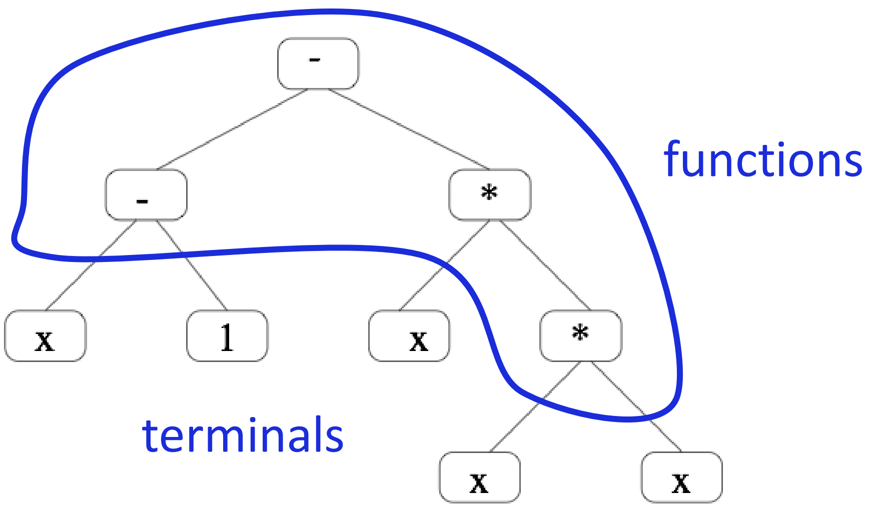
- (+ 1 2 3) evaluates to 6
- (+ (- 3 2) (* 2 4)) evaluates to (+ 1 8) which is 9
- (IF (> TIME 10) 3 4) evaluates to 3 if TIME is 11 or more

  and to 4 if time is 10 or less
- If TIME=20, what is the value of (+ 1 2 (IF (> TIME 10) 3 4))?

Programs as Tree Structures

- Representation: Tree Structures
- Programs are constructed from a *terminal* set & *function* set
- Terminals and functions are also called primitives

$$(x-1)-x^3$$



GA vs GP: Representation

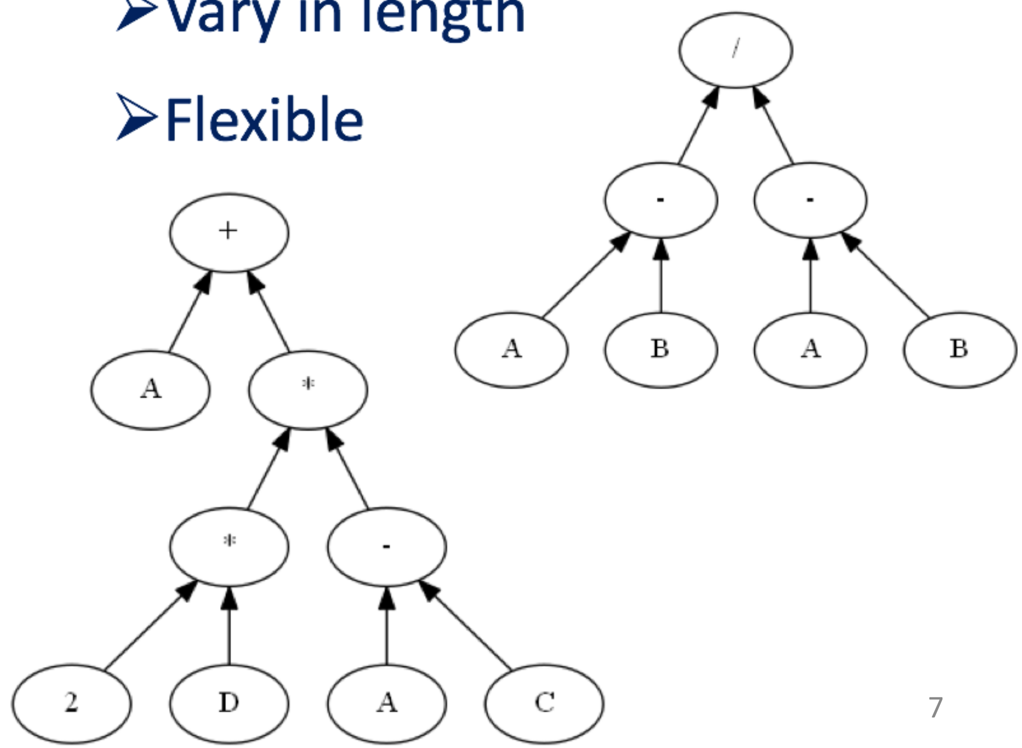
Genetic Algorithm	Genetic Programming

- Bit string representation
- Fixed in length
- Inflexible

1	0	1	1	0	1	0	1	1	0

0	0	0	1	0	1	1	1	0	1

- Tree-like structure
- Vary in length
- Flexible



Terminal Set

- A terminal set consists of a set of terminals:
  - attributes/features
  - "Constants" (randomly generated, but don't change)

- Terminals have no arguments & form the leaves of the tree

- Terminals represent the ***inputs*** of a GP program, i.e. input from the environment (a specific task)

- Attributes or features of a problem domain are usually used as terminals

Function Set

- A function set consists of a set of functions or operators
- Functions form the root and the internal nodes of the tree representation of a program

- Two kinds of functions: general functions, and domain-specific functions

- General functions:
  - Arithmetic functions: +, -, *, %.
  - Protected division (%): returns 0 (or 1) if denominator is 0 (*why?*)
  - Other functions: sin, cos, exp, log, abs, ...

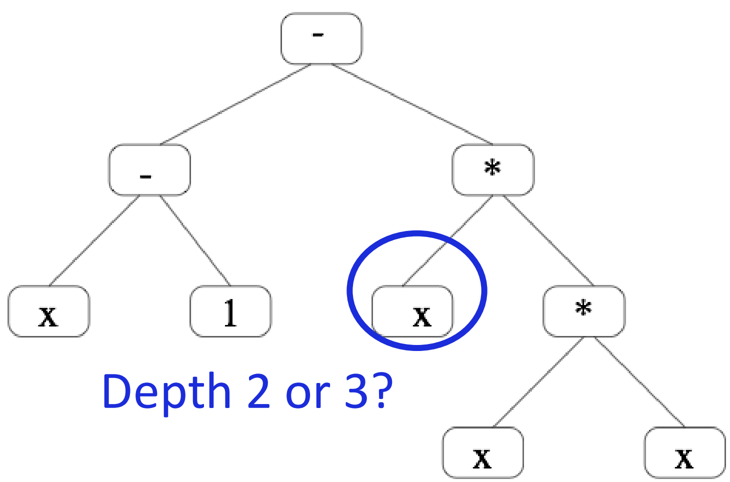
- Domain Specific functions: e.g. image processing operators

Sufficiency and Closure

- Selection of the functions and terminals is critical to success
- The terminal set and the function set should be selected to satisfy the requirements of *closure* and *sufficiency*
- Sufficiency: There must be some combination of terminals and function symbols that can solve the problem
- Closure: Any function can accept any input value returned by any function (and any terminal)
  - NB: “strongly-typed GP” violates this!

Program Generation

- For initialising a population, or performing mutation
- Maximum program size: the maximum depth of a tree
- Depth: The depth of a node is the minimum number of nodes that must be traversed from the root of the tree to it



Program Generation

- There are several ways of generating programs: full, grow, and ramped half-and-half

- Full method:
  - Functions are selected as the nodes of the program tree until a given depth is reached
  - Then terminals are selected to form the leaf nodes
  - This ensures that full, entirely balanced trees are constructed

Program Generation

- Grow method:
  - Nodes are selected from both functions and terminals
  - If a terminal is selected, the branch with this terminal is terminated and we move on to the next non-terminal branch in the tree

- Ramped half-and-half method:
  - Both the full and grow methods are combined
  - Half of the population generated for each depth value are created by using the grow method and the other half using the full method

- Ramped half-and-half is widely used in many GP systems
  - Good balance of the benefits of each!

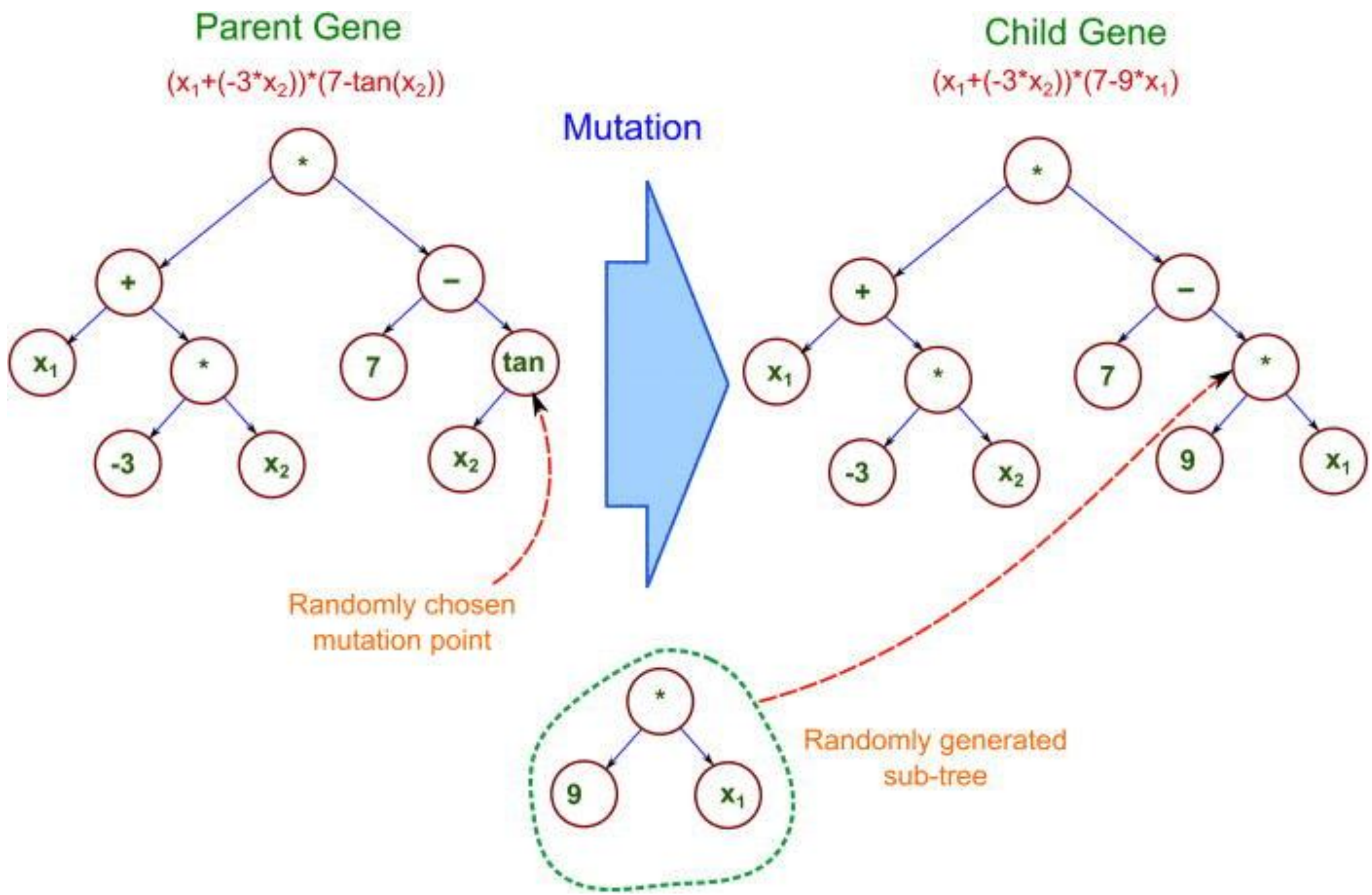
Genetic Operators in GP

- Evolution proceeds by updating the initial population by the use of genetic operators
  - An initial population usually has very bad fitness
  - Three main operators in GP: reproduction, mutation, and crossover

- Reproduction:
  - Simply copy a selected program to the new generation
  - Allow good programs to survive
  - Elitism: keep only the best!

- Mutation:
  - Operate on a single selected program
  - Remove a random subtree of the program
  - Generate a new subtree in the same place

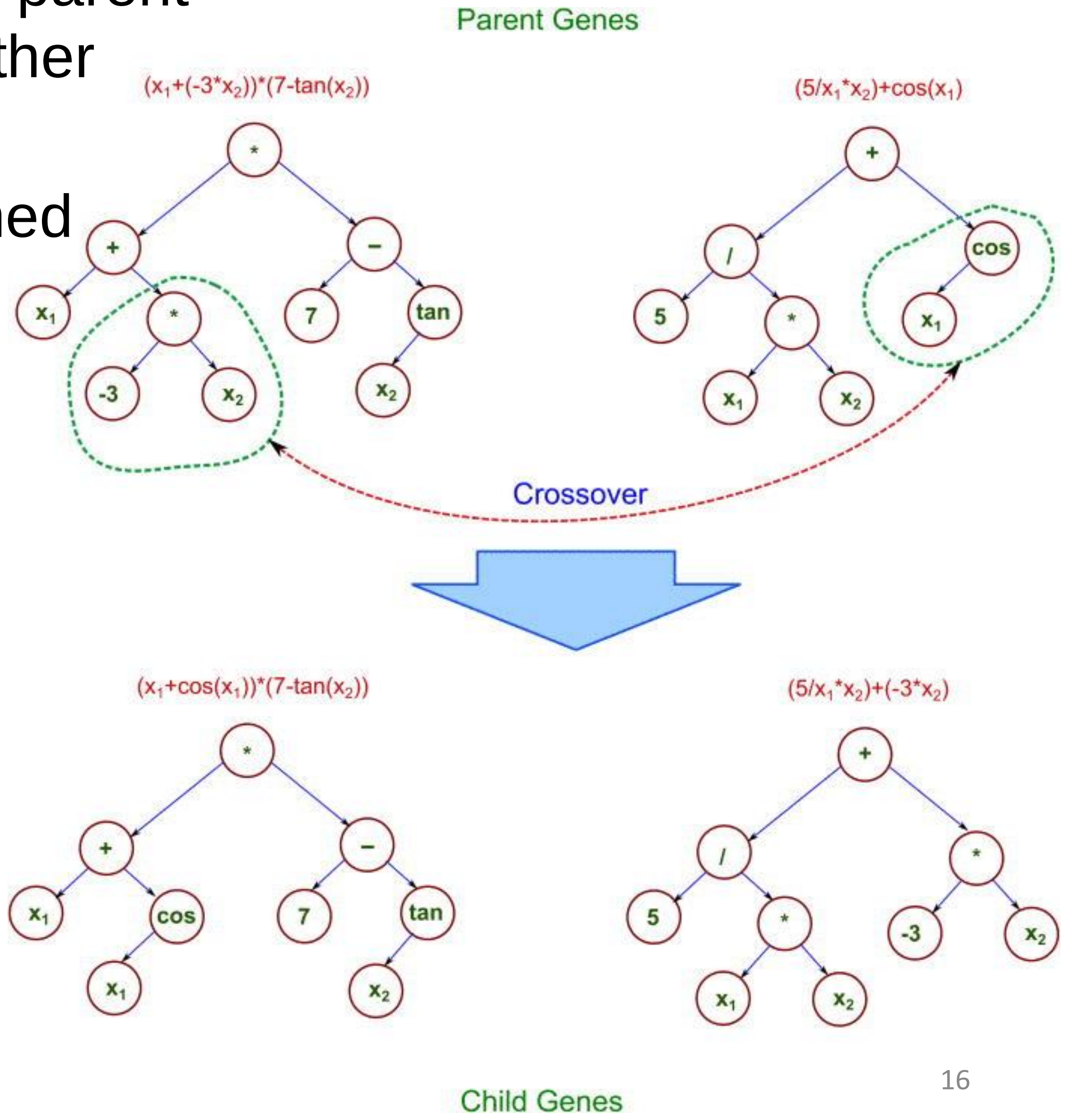
Mutation in GP



Crossover in GP

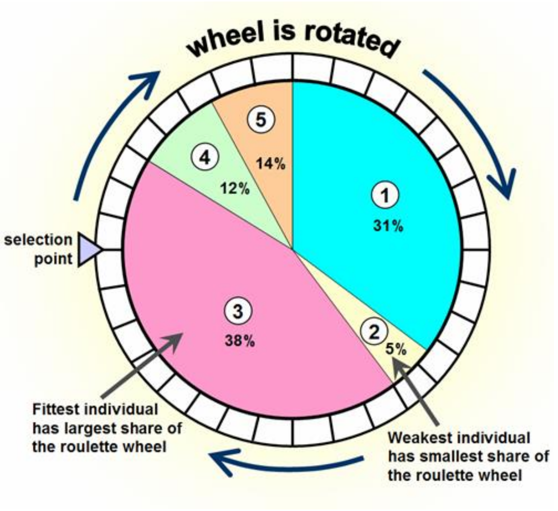
- Swap a subtree of one parent with a subtree of the other

- Put the two newly-formed programs into the next generation

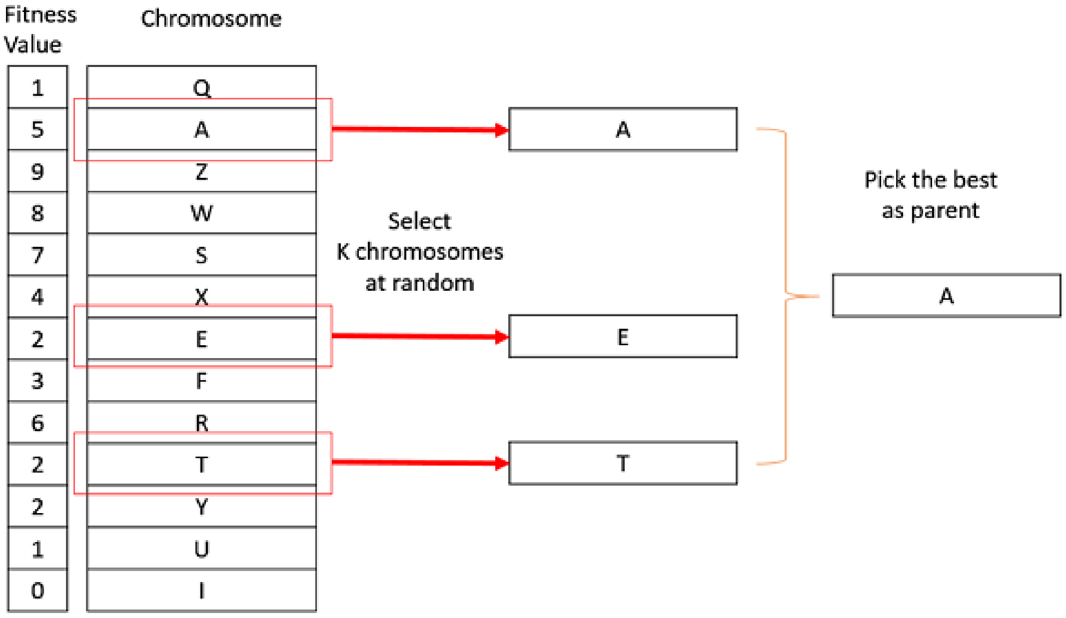


Selection (same as GAs!)

- Roulette wheel selection
  - The probability of being selected is proportional to the fitness
  - Assume fitness is maximized



- K-tournament selection



Fitness Evaluation

- The fitness of a program generated by the evolutionary process is evaluated according to the **fitness function**
- The fitness function should give graded and continuous feedback on how good a program is on the training set
- The fitness function plays a *very* important role in the evolutionary process and varies with the problem domain
- **Fitness cases**: instances used for fitness evaluation
  - Training cases: training instances used for learning
  - Test cases: test instances used for performance evaluation

Fitness Function Examples

- Image matching: the number of matched pixels
- Robot learning obstacle avoidance: the number of walls hit for a robot
- Classification task: the number of correctly classified examples, error rate, or classification accuracy
- GP-controlled gambling agent: the amount of money won
- Artificial life application: the amount of food found and eaten

A Basic GP algorithm

- Initialise the population

- Repeat until the stopping criteria is met:
  - Evaluate the fitness of each program in the current population
  - Create an empty new population
  - Repeat until the new population is full:
    - Select programs in the current generation (often *tournament selection*)
    - Apply genetic operators to the selected programs to generate offspring (*e.g. 80% crossover, 15% mutation, 5% reproduction*).
    - Insert the children programs into the new generation.

- Output the best individual program in the population.

Tackling a Problem with GP

- What terminals should be used in the program trees?
- What functions are needed to represent the program tree?
- What is the fitness function/measure?
- Parameters values for controlling the evolutionary process: e.g. what population size, tree depth and tournament size?
- When to terminate a run?
- Which genetic operators should be used, and how frequently should they be applied?

Summary

- Overview of EC (GA) process
- GP basics: representation, primitives, terminals, functions, fitness, genetic operators, selection
- GAs vs GP
- Basic GP algorithm
- Suggested reading:
- http://www.genetic-programming.com/
- https://en.wikipedia.org/wiki/Genetic_programming
- Next lecture: GP examples, for regression and classification