VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

AIML231/DATA302 —Techniques in Machine Learning

Week 10 - Evolutionary Computation

Dr Qi Chen

School of Engineering and Computer Science

Victoria University of Wellington

Qi.Chen@vuw.ac.nz

Why Need Evolutionary Computation?

- We have discussed several methods and algorithms in ML
- But they have limitations:
  - Local optima
  - Unreasonable assumptions
  - Needs to predefine/fix the structure/model of the solution, and only learns the parameters/coefficients
  - Many parameters to learn (high-dimensional optimisation)
- Evolutionary Computation (EC) is one technique that can avoid some of the problems

Evolutionary Computation and Learning

- In computer science, **evolutionary computation** is a family of "*nature inspired*" AI algorithms for global optimisation.

- In technical terminology, they are a family of population-based trial-and-error problem solvers with a metaheuristic or stochastic optimisation character.

- **Evolutionary Learning** is the use of evolutionary computation methods for tackling machine learning tasks

- *Source: https://en.wikipedia.org/wiki/Evolutionary_computation*

EC Techniques

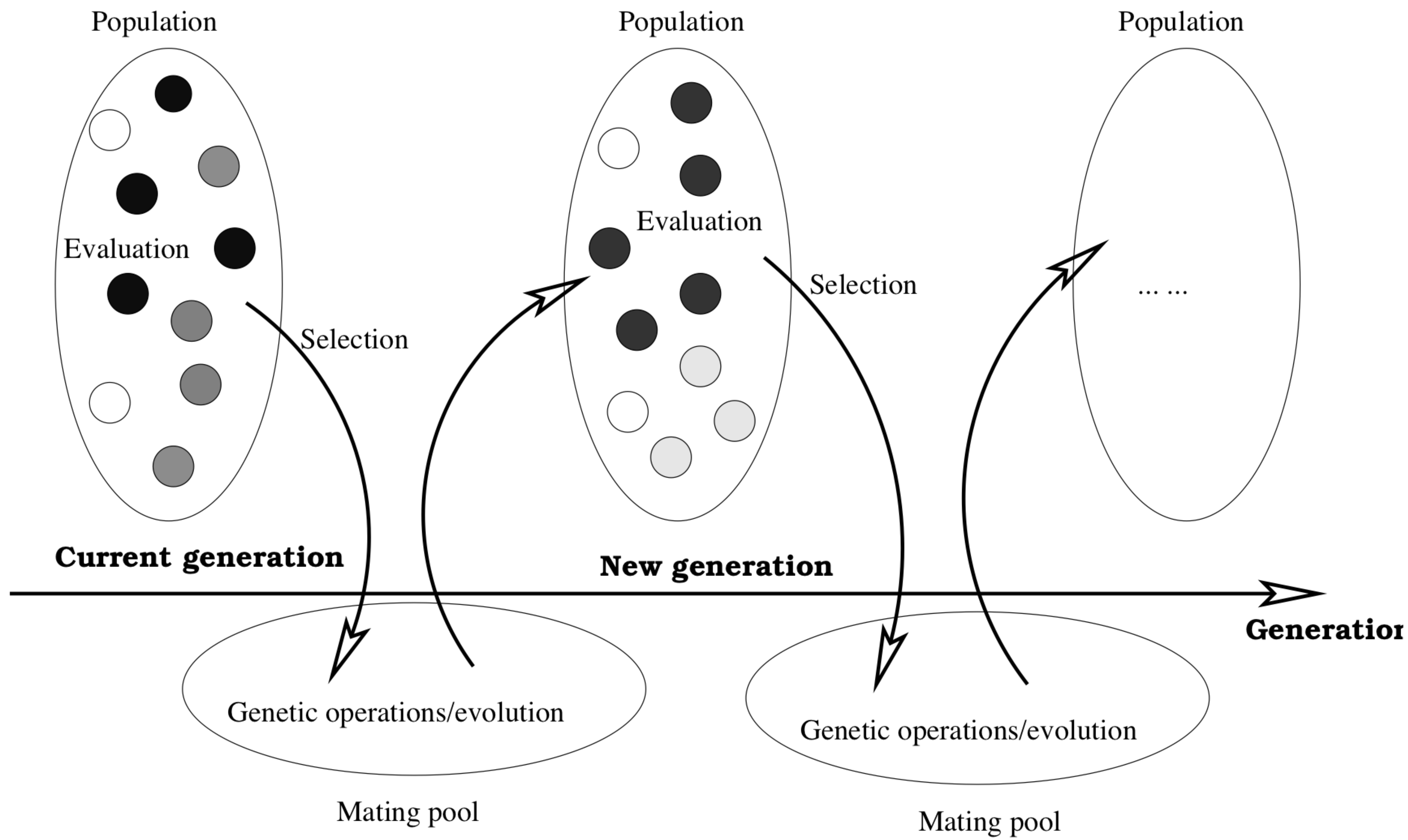
- Evolutionary algorithms (EAs)
  - Genetic algorithms (the biggest branch)
  - Evolutionary programming
  - Evolutionary strategies
  - Genetic Programming (Koza, 1990s, fast growing area)

- Swarm intelligence (SI)
  - Ant colony optimisation
  - Particle swarm optimisation (PSO)
  - Artificial immune systems

- Other techniques
  - Differential evolution
  - Estimation of distribution algorithms
  - …

Evolutionary Algorithms

- Search for the best individual by evolving a *population* with reproduction (e.g. crossover, mutation)



Evolutionary Search

- Search space of **candidate solutions**
  - Not space of partial solutions
  - Modify whole solutions rather than extending partial solutions

- Genetic beam search
  - Keep track of a set of good solutions
  - Not all candidate solutions, unlike best first or A*
  - Not only the best candidates, unlike in hill climbing or gradient descent

- *Combine* good candidates to construct new candidates
  - Can modify candidates in isolation (mutation)
  - Or different candidates can *interact* in evolution (crossover)

Key Characteristics

- One (or more) populations of *individuals*
- Dynamically changing populations due to the birth and death of individuals (through crossover, mutation, …)
- A *fitness function* which reflects the ability of an individual to survive and reproduce ("survival of the fittest")
- Variational inheritance: offspring closely resemble their parents, but are not identical

- Final solution (individual): the one with the best *fitness*
- Fitness could be accuracy, cost, error, …

Key Design Questions

- Representation
  - How can we represent individuals (solutions)?
- Evaluation
  - How can we evaluate individuals (fitness function)?
  - A fitter individual should have a better objective value (e.g. smaller error)
- Selection
  - How to select individuals into the mating pool (selection scheme)?
  - Fitter individuals should be more likely to survive/reproduce
  - Selection pressure
- Genetic Operators
  - How to generate new individuals (crossover, mutation operators)?
  - Children inherit strong parts of parents
  - Maintain diversity (jump out of local optima)
- Other parameters
  - population size, mating pool size, stopping criteria, …

Individual Representation

- Problem dependent

- Binary string (e.g. feature selection)

1	0	0	1	1

0	0	0	1	1

- Continuous vector (e.g. ANN weight optimisation)

-0.73	0.10	0.35	-0.06	0.23

-0.13	0.10	0.35	-0.06	-0.29

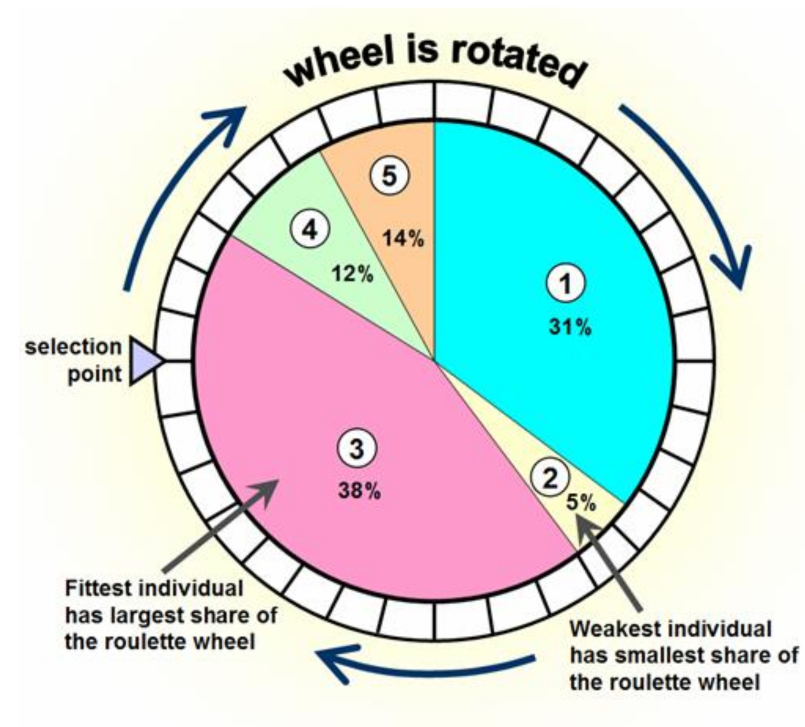
Fitness Evaluation

- **Fitness function**: reflect the quality of individuals
  - Must correspond to optimality property
  - Must be computable
  - Smoothness:
    - Small changes to candidate -> small changes to quality/fitness
    - Large changes to candidate -> large changes?

- Depending on the problem, the fitness function could be:
  - the larger, the better --- maximisation
  - the smaller, the better --- minimisation

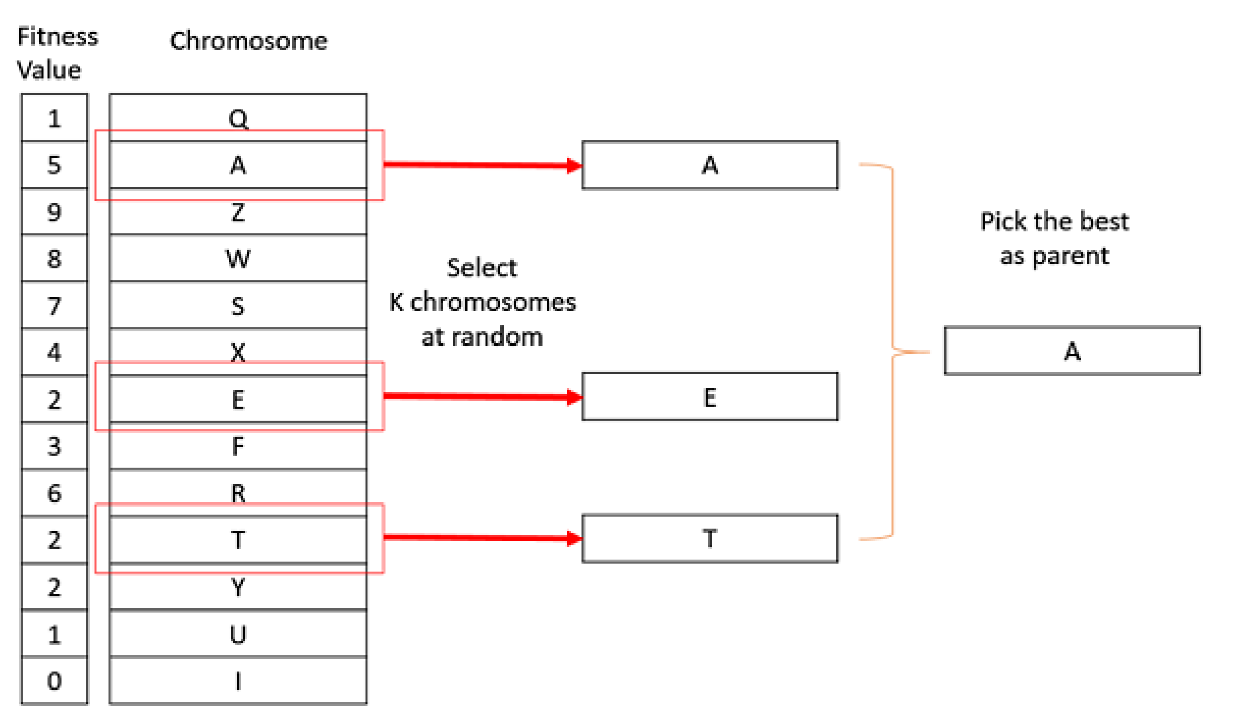
Selection

- Uniform selection
  - Each individual has the same chance to be selected



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

- K-tournament selection



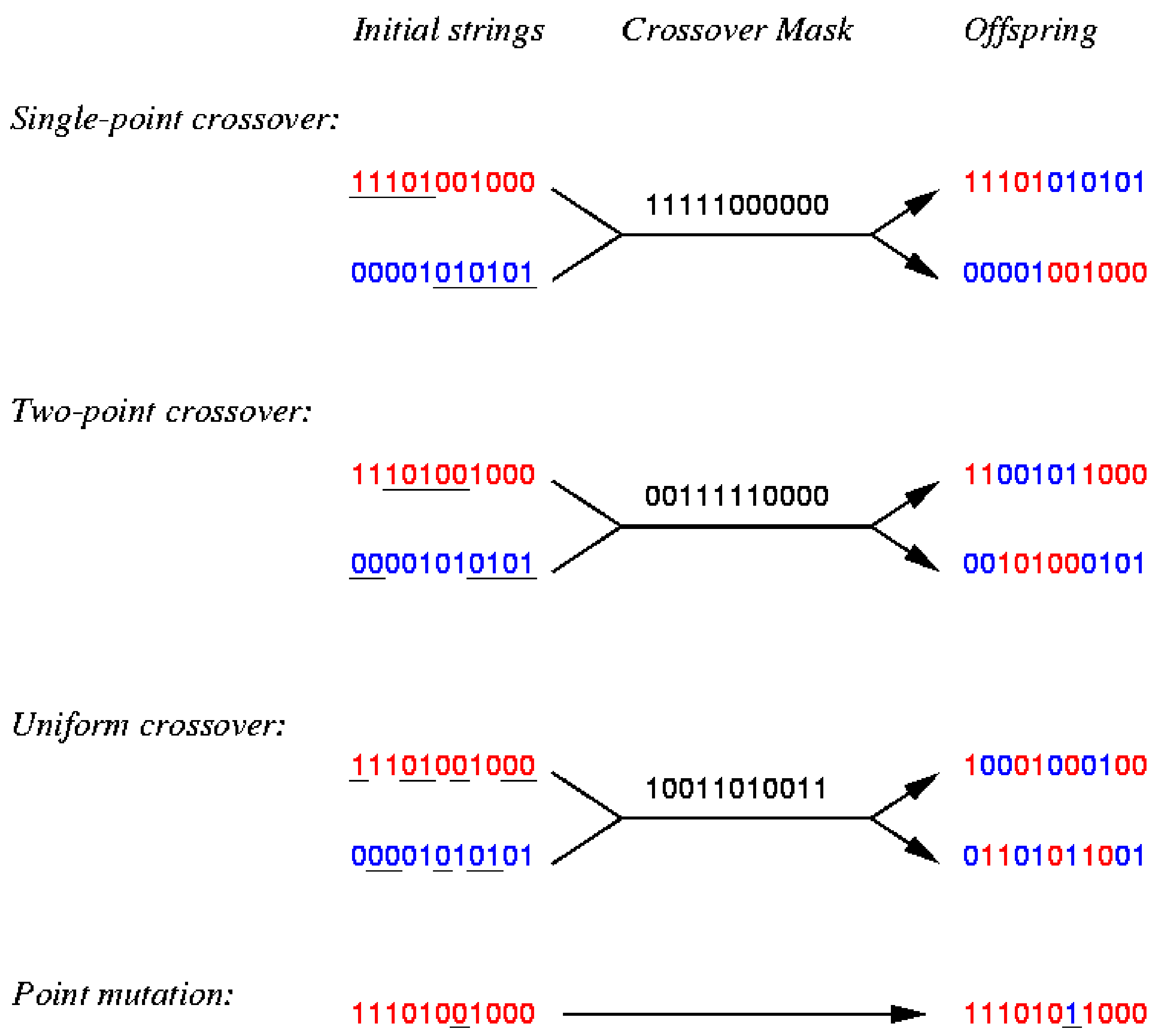
- Truncate selection
  - Select the best k individuals

Genetic Operators

- Depends on the problem – individual representation
  - Swap a bit of a binary vector
  - Resample an element of a continuous vector
  - Shuffle a part of a sequence
  - …
- A representative: Genetic Algorithms

Genetic Algorithm

- Representation: binary string
- An individual is also called a chromosome



- Other representations as well: continuous vector, permutation, …

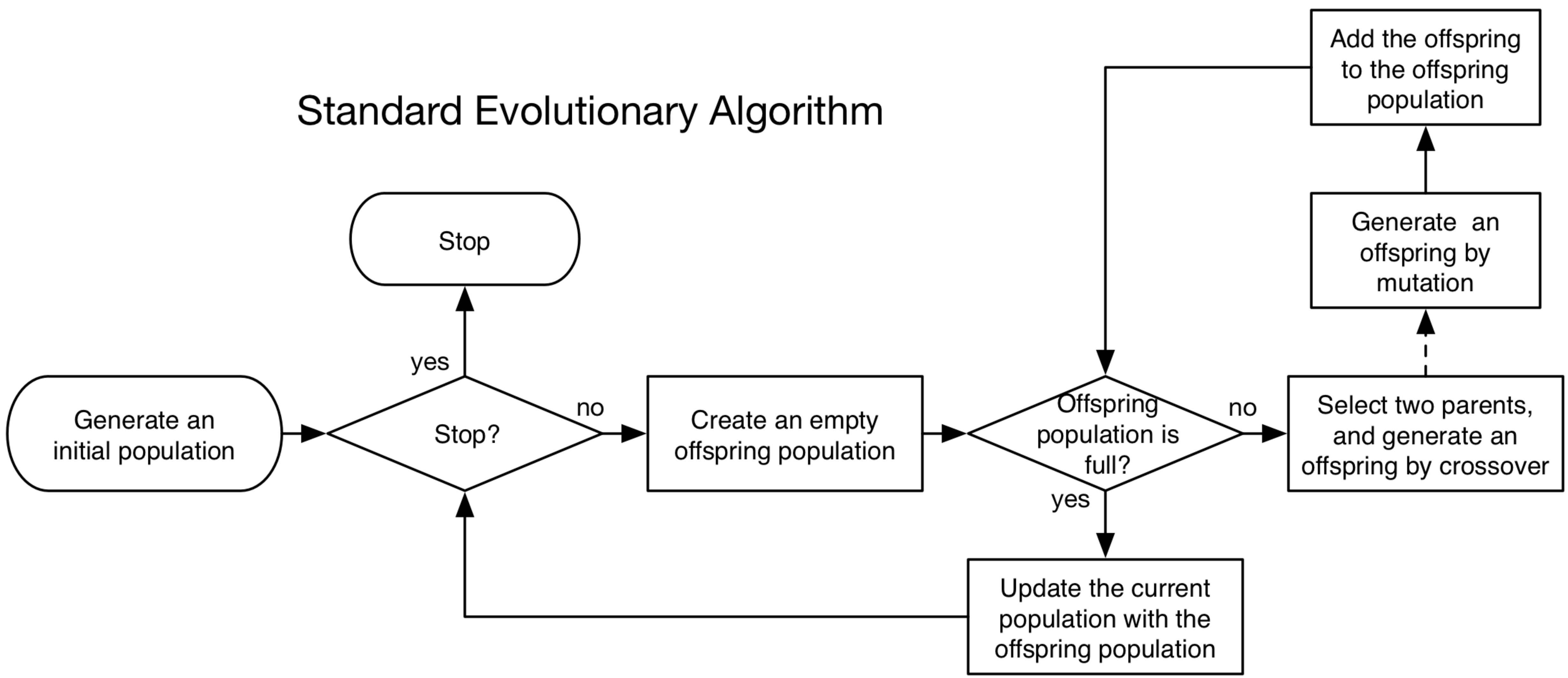
A Basic Genetic Algorithm

- Randomly initialise a population of chromosomes

- **Repeat until** stopping criteria are met:
  - Construct an empty new population
  - **Repeat until** the new population is full:
    - Select **two** parents from the population by roulette wheel selection
    - Apply crossover to the two parents to generate two children
    - Each child has a probability (mutation rate) to undergo mutation
    - Put the two children into the new population
  - **End Repeat**
  - Move to the new population (new generation)
- **End Repeat**

- Output the best individual from the final population

A Basic Genetic Algorithm



A Simple GA Example

- **OneMax Problem**
  - Target to (11111…1)
  - More zeros means worse: far away from the target
  - Simple “benchmark” problem!

- Representation: bit string
- Fitness function: $1 + \sum_i x_i$ (the larger the better)
- Crossover: single-point crossover
- Mutation: point mutation

- ***Assume our algorithm does not know the problem or fitness function!***

A Simple GA Example

- 10 bits (Optimal fitness = 11)
- population size = 20
- mutation rate = 0.25 (25%)
- Run for 10 generations

```
Generation: 0 Average Fitness: 4.1 Best Fitness: 7
Generation: 1 Average Fitness: 5.4 Best Fitness: 8
Generation: 2 Average Fitness: 6.2 Best Fitness: 9
Generation: 3 Average Fitness: 7.2 Best Fitness: 9
Generation: 4 Average Fitness: 6.6 Best Fitness: 9
Generation: 5 Average Fitness: 7.2 Best Fitness: 10
Generation: 6 Average Fitness: 7.9 Best Fitness: 11
Best solution: [1 1 1 1 1 0 1 1 1 0] with fitness: 11
```

Genetic Programming

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

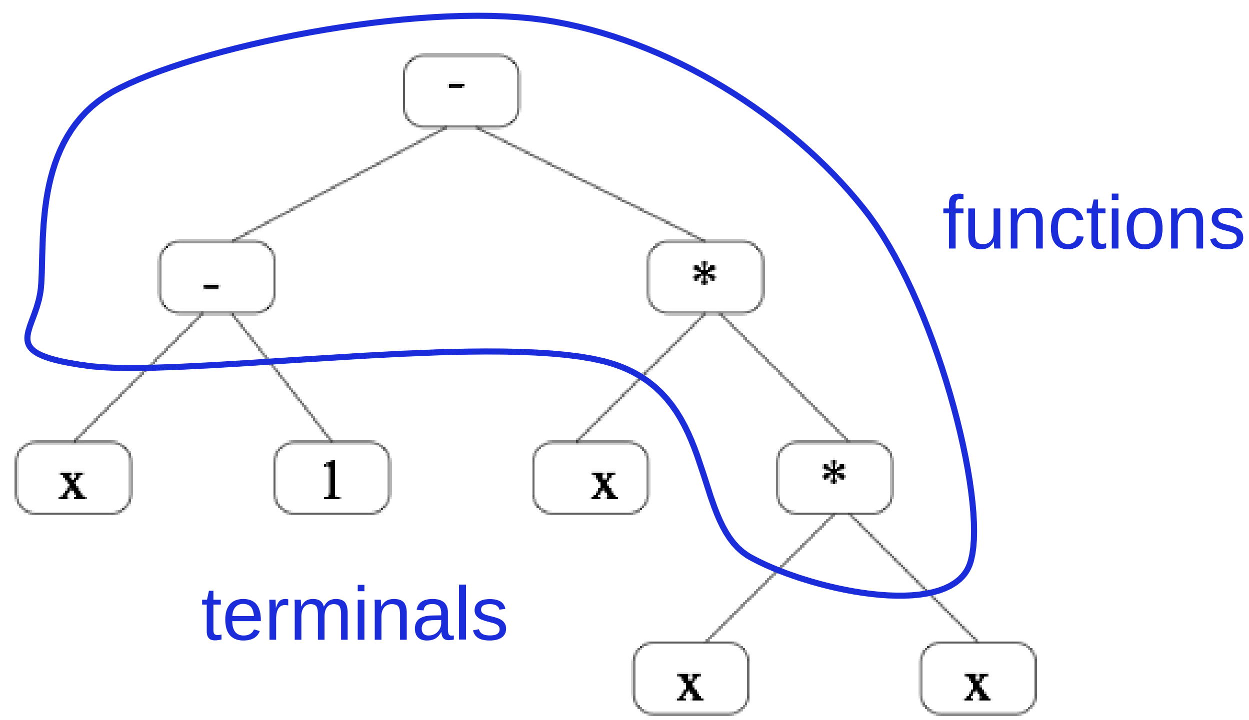
- GP uses a similar evolutionary process to the general evolutionary algorithms (e.g. GAs)
  - GA uses bit strings to represent solutions;
    GP uses tree-like structures that can represent computer programs
  - GA bit strings use a fixed length representation;
    GP trees can vary in length
  - The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

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

Genetic Algorithm

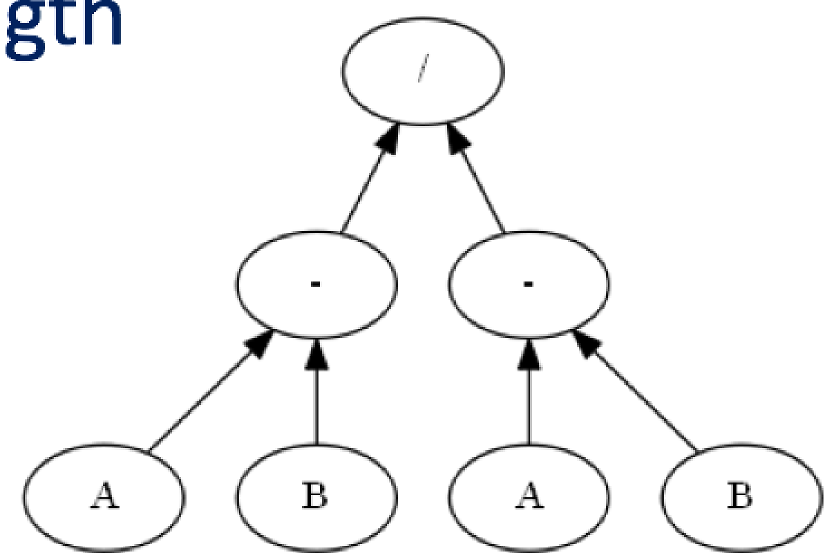
- Bit string representation
- Fixed in length
- Inflexible

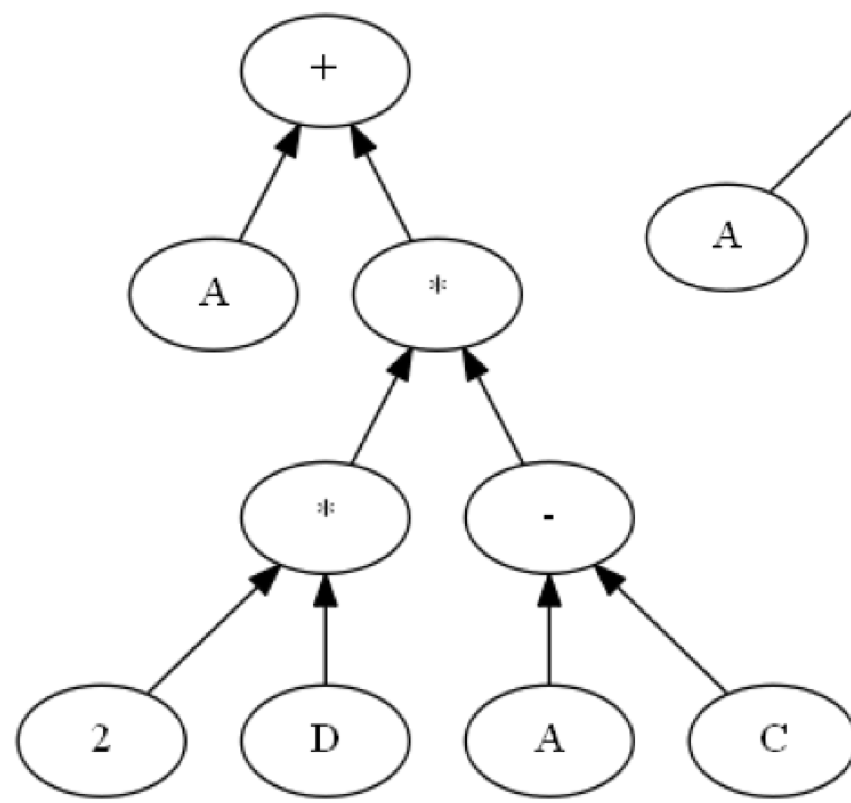
1	0	1	1	0	1	0	1	1	0

0	0	0	1	0	1	1	1	0	1

Genetic Programming

- Tree-like structure
- Vary in length
- Flexible





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.

Summary

- Evolutionary computing overview
- Main idea and process
- Representations of candidate solutions
- Selection and genetic operators
- Genetic algorithms
- Genetic programming (GP)
- Other EC algorithms and techniques