Lancaster University

SCC361: Artificial Intelligence

Week 6: Search Algorithms

Genetic Algorithms

Dr Bryan M. Williams
School of Computing and Communications, Lancaster University
Office: InfoLab21 C46 Email: b.williams6@lancaster.ac.uk

Be sure to check in to all timetabled sessions using Attendance Check-in

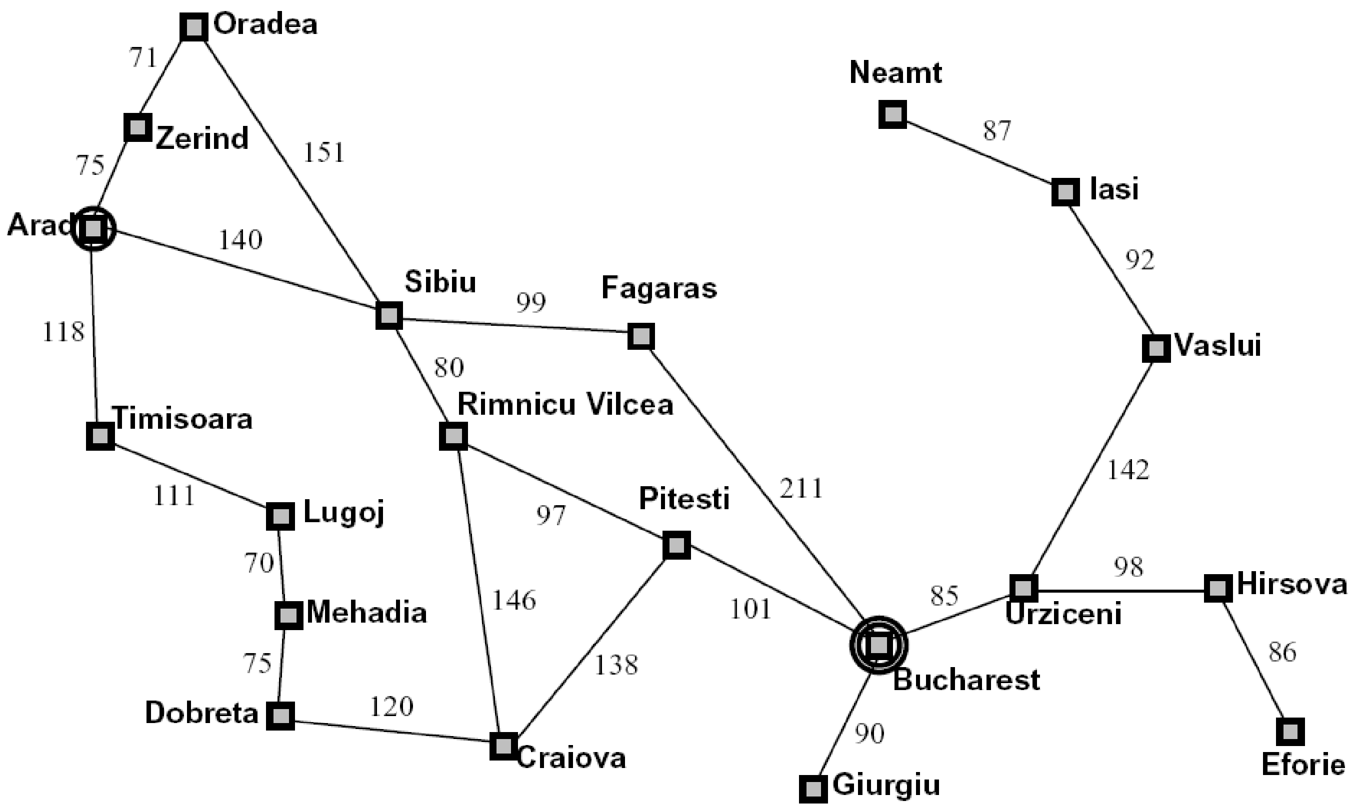
To check in:

- Check the **Attendance Hub** in iLancaster
- Click **Check In**
- Wait for the "You are checked in" confirmation page
- Here is a the demo

Please DO NOT leave a timetabled session without your attendance being registered

Search Algorithms

Search Algorithms Example 1



Shortest Path Problem

Search Algorithms Example 2

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

8-puzzle problem

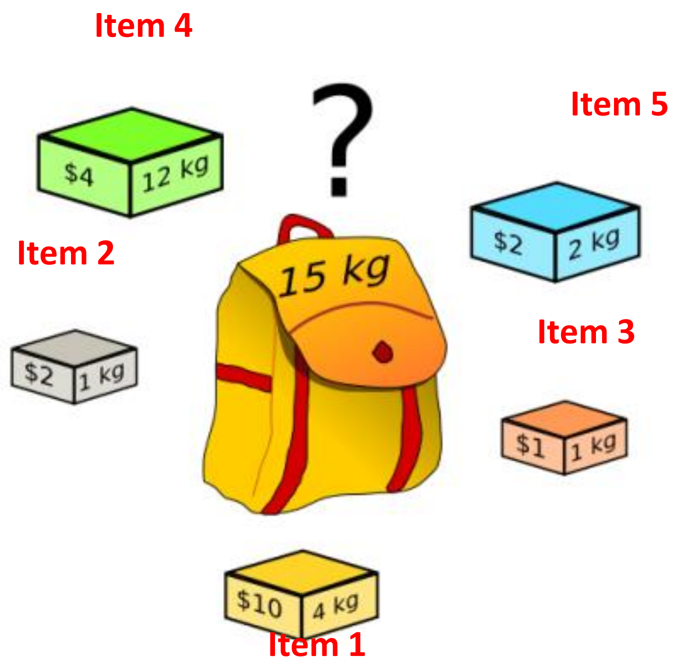
Search Algorithms Example 3

0-1 Knapsack Problem

- Set of items with given
  - value
  - weight
- Have a given capacity of the knapsack

- Aim: Maximise the value of the items in the knapsack without exceeding the total capacity

- This is an **NP-Complete** problem



Search Algorithms Example 4

Traveling Salesman Problem

- Famous touring problem
- Set of cities with given distances between them
- Traveling salesman must visit all cities
- Each city must be visited exactly once
- The aim is to find the shortest path, i.e. a sequence of cities to minimise travel distance
- The problem is NP-Complete

Search Algorithms Example 5

System of Nonlinear Equations

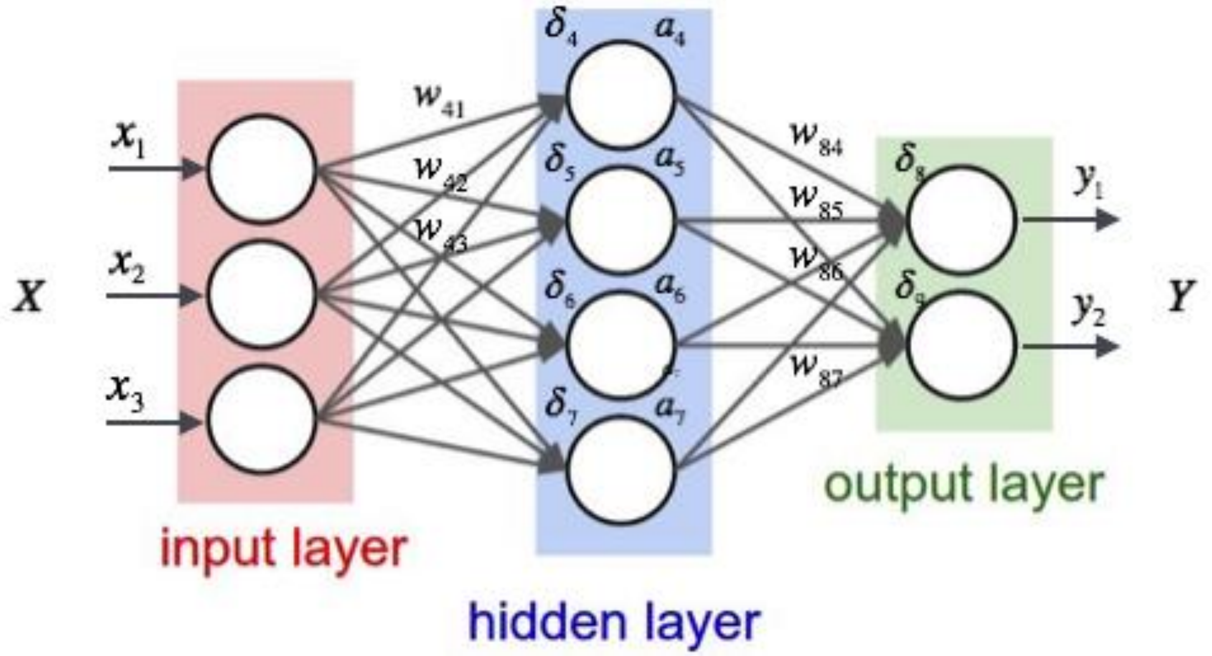
$$
\begin{aligned}
f_1(x) &= x_1 - 0.2543 - 0.1832x_4x_3x_9 \\
f_2(x) &= x_2 - 0.3784 - 0.1628x_1x_{10}x_6 \\
f_3(x) &= x_3 - 0.2716 - 0.1696x_1x_2x_{10} \\
f_4(x) &= x_4 - 0.1981 - 0.1559x_7x_1x_6 \\
f_5(x) &= x_5 - 0.4417 - 0.1995x_7x_6x_3 \\
f_6(x) &= x_6 - 0.1465 - 0.1892x_8x_5x_{10} \\
f_7(x) &= x_7 - 0.4294 - 0.2118x_2x_5x_8 \\
f_8(x) &= x_8 - 0.0706 - 0.1708x_1x_7x_6 \\
f_9(x) &= x_9 - 0.3450 - 0.1961x_{10}x_6x_8 \\
f_{10}(x) &= x_{10} - 0.4265 - 0.2147x_4x_8x_1
\end{aligned}
$$

We can consider this problem as

$$\min\left(f_1(x), f_2(x), \dots, f_n(x)\right)$$

Search Algorithms Example 6

Neural Network Weights



Determine the weights w_{ij} of a neural network

Search Algorithm

Uniform Search Methods (Brute Force)

- Breadth-first
- Depth-first
- Uniform Cost
- Iterative Deepening Depth-First
- Uniform Cost
- Bidirectional Cost

Informed Search Methods (Injecting Heuristic)

- Best-first Search (Greedy)
- A*

Genetic Algorithms

- A genetic algorithm is a guided random search strategy
- Inspired by Darwin’s theory of natural evolution
- The fittest survives, and has a greater chance of breeding and passing forward its genetic information
- Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal solution

Natural Selection

The process of natural selection starts with the selection of fittest individuals from a population.

They produce offspring/children which inherit the characteristics of the parents and will be added to the next generation.

If parents have better fitness, their offspring/children will be better than parents and have a better chance at surviving.

This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

This notion can be applied for a search problem

Consider a set of solutions for a problem

Select a set of best ones out of them

Outline of Basic Genetic Algorithm

1: Start: Generate random population of n chromosomes

2: Fitness: Evaluate the fitness of each chromosome

3: New population: Create a new population by repeating:

- **A: Selection:** Select two parent chromosomes based on their fitness
- **B: Crossover:** With a crossover probability, cross over the parents to form new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
- **C: Mutation:** With a mutation probability, mutate new offspring at each locus (position in chromosome).
- **D: Accepting:** Place new offspring in a new population
- **E: Fitness:** Evaluate the fitness of each chromosome

4: Replace: Generate a new population for a further run of algorithm

5: Test: If the end condition is satisfied, stop, and return the best solution in current population

6: Loop: Go to step 3

Basic Genetic Algorithm Outline

Lancaster University

Current population
8 Chromosomes

- Chromosome#1 90
- Chromosome#2 20
- Chromosome#3 40
- Chromosome#4 40
- Chromosome#5 10
- Chromosome#6 80
- Chromosome#7 60
- Chromosome#8 50

Iterating through step 3

New population
8 Chromosomes

- Chromosome#9 40
- Chromosome#10 50
- Chromosome#11 60
- Chromosome#12 50
- Chromosome#13 20
- Chromosome#14 90
- Chromosome#15 90
- Chromosome#16 90

Basic Genetic Algorithm Outline

4: Replace: Generate a new population for a further run of algorithm

Current population 8 Chromosomes	Mixed population 8 Chromosomes	New population 8 Chromosomes
Chromosome#1 90	Chromosome#6 80	Chromosome#9 40
Chromosome#2 20	Chromosome#7 60	Chromosome#10 50
Chromosome#3 40	Chromosome#1 90	Chromosome#11 60
Chromosome#4 40	Chromosome#10 50	Chromosome#12 50
Chromosome#5 10	Chromosome#11 60	Chromosome#13 20
Chromosome#6 80	Chromosome#14 90	Chromosome#14 90
Chromosome#7 60	Chromosome#15 90	Chromosome#15 90
Chromosome#8 50	Chromosome#16 90	Chromosome#16 90

Basic Genetic Algorithm Outline

5 and 6: If the end condition is not satisfied, go to Step 3

Current population
8 Chromosomes

- **Chromosome#6 80**
- **Chromosome#7 60**
- **Chromosome#1 90**
- **Chromosome#10 50**
- **Chromosome#11 60**
- **Chromosome#14 90**
- **Chromosome#15 90**
- **Chromosome#16 90**

Iterating through step 3

Chromosones, genes, alleles

Population

- A set of chromosones is referred to as a **population**

Chromosome

- The string, which is a candidate solution to the search problem, is referred to as a **chromosome/ individual**

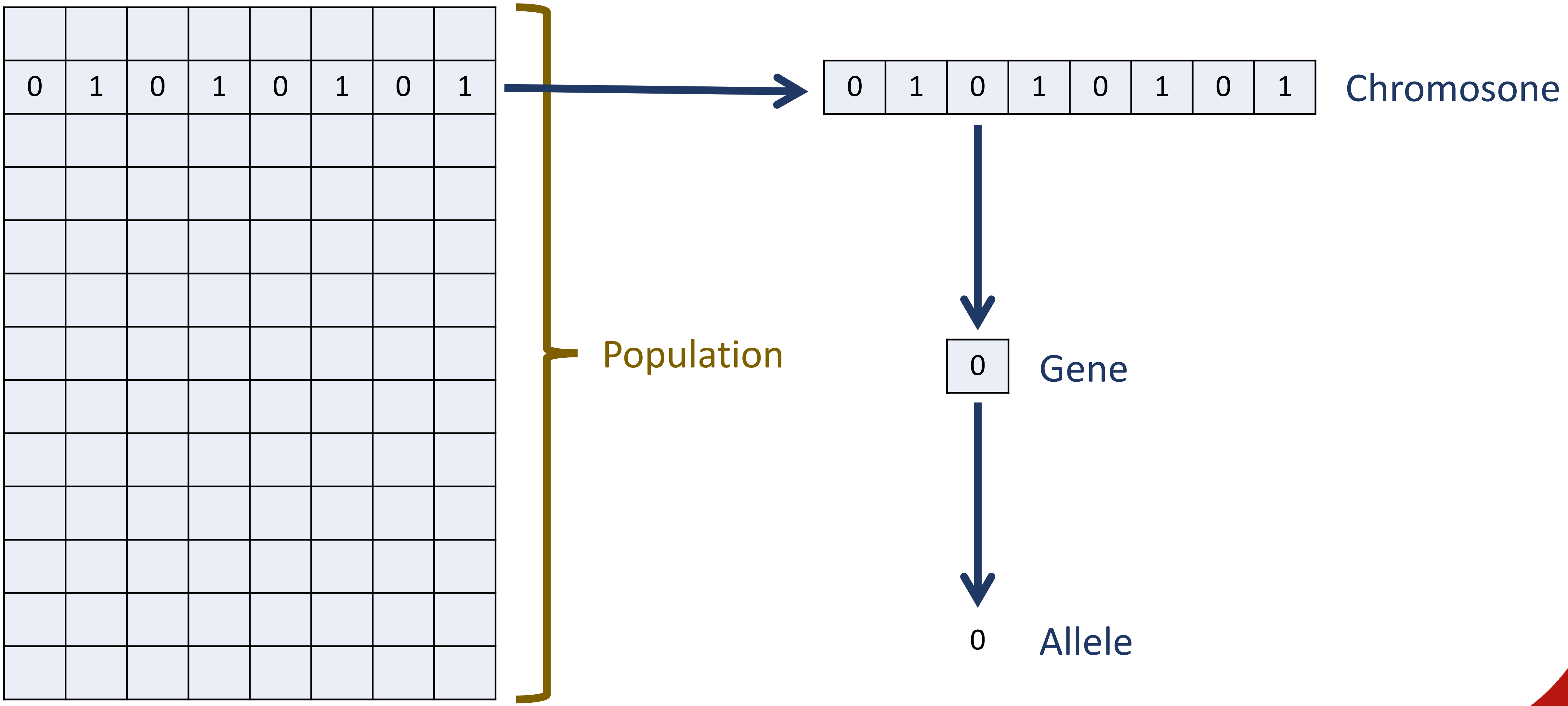
Genes

- A chromosome is characterized by a set of parameters (variables) known as **genes**

Alleles

- The values of genes are called **alleles**

Chromosones vs genes vs alleles



Chromosone Encoding

Popular Encoding Methods

- Binary Encoding
- Value Encoding
- Permutation Encoding
- Tree Encoding

Binary Encoding

Binary enoding is the most common to represent chromosones

It was first used because of relative simplicity

Each chromosone is a string of bits: 0 or 1

- Chromosone 1

1	1	0	0	1	1	0	1	0	1	0	0	0	1

- Chromosone 2

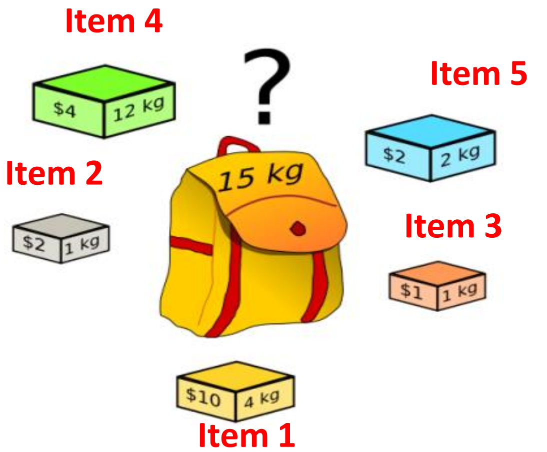
0	1	1	0	1	0	1	0	1	1	1	1	0	1

Meaning of the string can vary

- Each bit can represent a characteristic of the solution
- The whole string can represent number(s)

Example: 0-1 Knapsack Problem

- Set of items with given value and size
- The knapsack has given capacity
- Select items to maximise the value of items in knapsack, but do not exceed total capacity
- Each bit says if the corresponding item is in knapsack



Item 1	Item 2	Item 3	Item 4	Item 5
1	1	1	0	1

Chromosome

Value Encoding

Can be used in problems where values are used, e.g. real numbers

Each chromosome is a **string of some values**

Values can be anything connected to the problem

- Numbers

1.31	2.27	9.6	2.21	7.03	6.22

- Characters

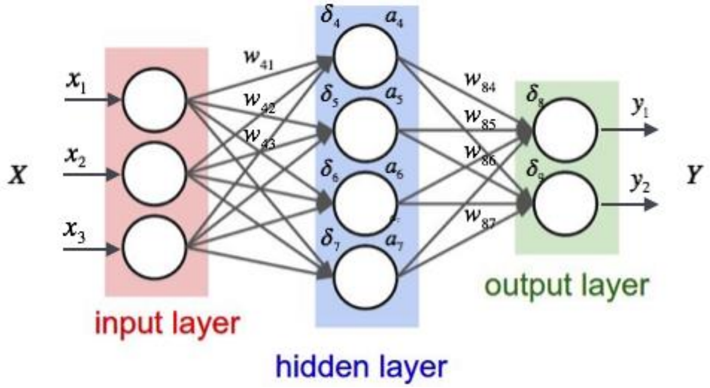
A	G	W	V	J	U	K	R	B

- Objects

←	→	↓	←	↓	←	↑	←	↑

Example: Finding weights for a neural network

- Given a neural network with a specific architecture
- Find weights to train the network to a desired output
- Real values in chromosones represent corresponding weights for inputs



Permutation Encoding

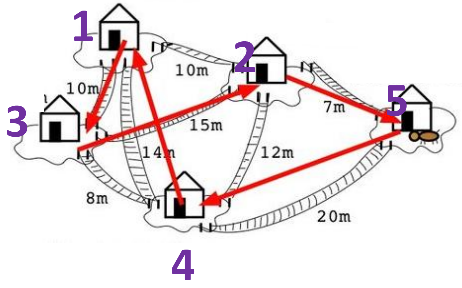
The permutation encoding can be used in ordering problems such as Travelling Salesman Problem (TSP) or Task Ordering problem.

Every chromosome is a string of numbers, which represent numbers in a sequence

The values must not be repeated in a single chromosome

Example: Travelling Salesman Problem

- Set of cities with given distances between them
- Must visit all cities
- Each city must be visited exactly once
- Chromosome gives order of cities to visit



1	3	2	5	4

- The aim is to find the shortest path, i.e. a sequence of cities to minimise travel distance

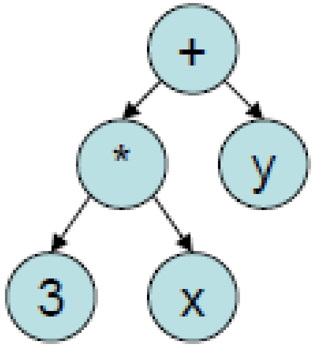
Tree Encoding

Mainly used for evolving programs or expressions

Every chromosome is a tree of some objects, such as functions or commands in programming language.

Example: Finding a function from given values

- Some input and output values are given
- Task is to find a function, which will give the best (closest to desired) output to all inputs
- Chromosome are functions represented in a tree



(+ (* 3 x) y)

Outline of Basic Genetic Algorithm

1: [Start] Generate random population of n chromosomes

2: [Fitness] Evaluate the fitness of each chromosome

3: [New population] Create a new population by repeating:

- **A: [Selection]** Select two parent chromosomes based on their fitness
- **B: [Crossover]** With a crossover probability, cross over the parents to form new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
- **C: [Mutation]** With a mutation probability, mutate new offspring at each locus (position in chromosome).
- **D: [Accepting]** Place new offspring in a new population
- **E: [Fitness]** Evaluate the fitness of each chromosome

4: [Replace] Generate a new population for a further run of algorithm

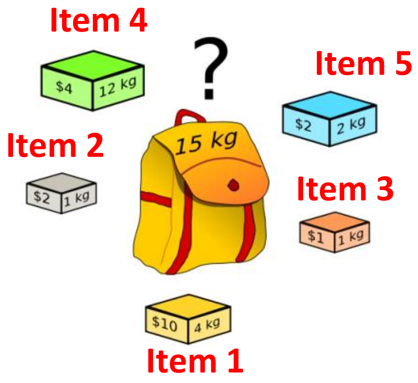
5: [Test] If the end condition is satisfied, stop, and return the best solution in current population

6: [Loop] Go to step 3

Generating Random Population

Example 1: 0-1 Knapsack Problem

- Set of items with given value and weight
- The knapsack has given capacity
- Select items to maximise the value of items in knapsack, but do not exceed total capacity
- Each bit says if the corresponding item is in knapsack



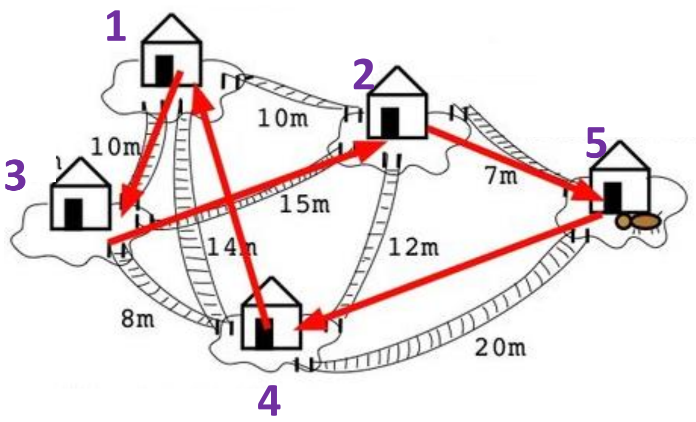
	Item 1	Item 2	Item 3	Item 4	Item 5
Chromosome 1	1	1	1	0	1
Chromosome 2	0	0	1	0	1
Chromosome 3	1	0	1	0	0
⋮			⋮		
Chromosome n	0	0	1	0	0

The number of chromosomes n is a parameter.

Generating Random Population

Example 2: Travelling Salesman Problem

- Traveling salesman must visit all cities
- Chromosome gives the order of cities to visit
- Each city must be visited exactly once
- Aim is to find the shortest path, i.e. a sequence of cities to minimise travel distance



	City 1	City 2	City 3	City 4	City 5
Chromosome 1	1	2	3	4	5
Chromosome 2	1	3	2	5	4
Chromosome 3	3	1	2	4	5
⋮			⋮		
Chromosome n	5	4	2	1	3

The number of chromosomes n is a parameter.

Outline of Basic Genetic Algorithm

1: [Start] Generate random population of n chromosomes

2: [Fitness] Evaluate the fitness of each chromosome

3: [New population] Create a new population by repeating:

- **A: [Selection]** Select two parent chromosomes based on their fitness
- **B: [Crossover]** With a crossover probability, cross over the parents to form new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
- **C: [Mutation]** With a mutation probability, mutate new offspring at each locus (position in chromosome).
- **D: [Accepting]** Place new offspring in a new population
- **E: [Fitness]** Evaluate the fitness of each chromosome

4: [Replace] Generate a new population for a further run of algorithm

5: [Test] If the end condition is satisfied, stop, and return the best solution in current population

6: [Loop] Go to step 3

Fitness Function

If the correct answer is known, fitness is a distance metric toward correct answer

If correct answer is unknown, fitness is estimator of the value of the solution

Combinations of multiple goals into a single numeric function can be difficult

Evolution can only be as good as the fitness function

Each problem has its own fitness function

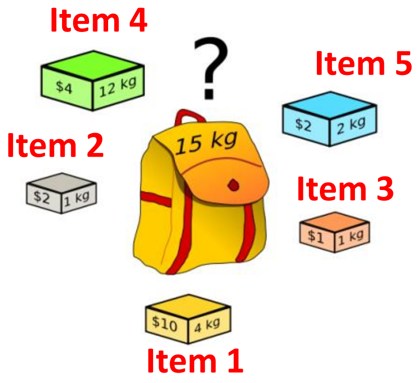
Generic Requirements: The fitness function should be

- Clearly defined
- Implemented efficiently
- Generate intuitive results, i.e. best/ worst candidates have best/worst scores

Fitness Function

Example 1: 0-1 Knapsack Problem

- Set of items with given value and weight
- The knapsack has given capacity
- Select items to maximise the value of items in knapsack, but do not exceed total capacity
- Each bit says if the corresponding item is in knapsack



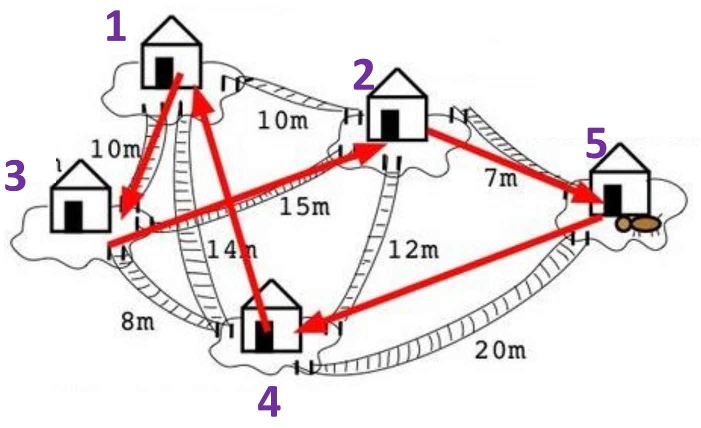
	Item 1	Item 2	Item 3	Item 4	Item 5	Weight (kg)	Profit ($)
Chromosome 1	1	1	1	0	1	$4+1+1+0+2=8$	$10+2+1+0+2=15$
Chromosome 2	0	0	1	0	1	$0+0+1+0+2=3$	$0+0+1+0+2=3$
Chromosome 3	1	0	1	0	0	$4+0+1+0+0=5$	$10+0+1+0+0=11$
⋮			⋮				
Chromosome n	0	0	1	0	0	$0+0+1+0+0=1$	$0+0+1+0+0=1$

We define fitness as the **profit value**

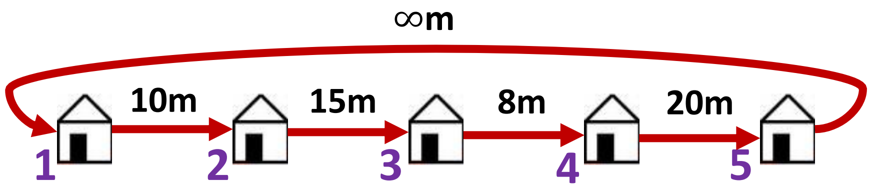
Fitness Function

Example 2: Travelling Salesman Problem

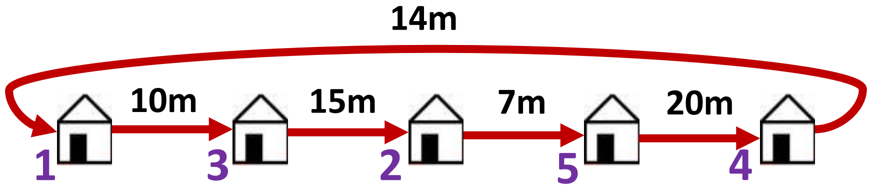
- Traveling salesman must visit all cities
- Chromosome gives the order of cities to visit
- Each city must be visited exactly once
- Aim is to find the shortest path, i.e. a sequence of cities to minimise travel distance



	City 1	City 2	City 3	City 4	City 5
Chromosome 1	1	2	3	4	5
Chromosome 2	1	3	2	5	4
Chromosome 3	3	1	2	4	5
⋮	⋮				
Chromosome n	5	4	2	1	3



Fitness = $\frac{1}{\infty} = 0$



Fitness = $\frac{1}{66^2} \approx 2.296 \times 10^{-4}$

We define fitness as

$$\frac{1}{distance^2}$$

Outline of Basic Genetic Algorithm

1: [Start] Generate random population of n chromosomes

2: [Fitness] Evaluate the fitness of each chromosome

3: [New population] Create a new population by repeating:

- **A: [Selection]** Select two parent chromosomes based on their fitness
- **B: [Crossover]** With a crossover probability, cross over the parents to form new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
- **C: [Mutation]** With a mutation probability, mutate new offspring at each locus (position in chromosome).
- **D: [Accepting]** Place new offspring in a new population
- **E: [Fitness]** Evaluate the fitness of each chromosome

4: [Replace] Generate a new population for a further run of algorithm

5: [Test] If the end condition is satisfied, stop, and return the best solution in current population

6: [Loop] Go to step 3

Selection

The first genetic operation in the reproductive phase

The objective selection is to choose the **fitter individuals** in the population that will create individuals for the next generation

Selection procedures can be broadly classified into two classes:

- Ordinal Selection
  - Tournament Selection
  - Linear Rank Selection
  - Truncation Selection
- Fitness Proportionate Selection
  - Roulette Wheel Selection
  - Stochastic Universal Selection

Tournament Selection

In tournament selection, s chromosomes are randomly selected from the population

- S is the tournament size, i.e. number of randomly selected individuals
- Most widely-used value is s=2

Tournament selection selects the best chromosome from this group

There are two variations:

- With replacement
- Without replacement

Imagine we are picking items from a bag

- With replacement: after picking s chromosomes, we put them back in the bag
- Without replacement: after picking s chromosomes, we don’t put the winner back in the bag

Tournament Selection

Example

- Consider a population with 5 chromosomes/ individuals ($n = 5$), with fitness values (f_i) as follows
- Tab Line 1
- Tab Line 2
- White space

With replacement: After picking $s = 2$ chromosomes, we put them back in the bag

1st Run

Chromosome #	2
Fitness (f_i)	18

Chromosome #	5
Fitness (f_i)	26

- Which chromosome is better? Chromosome 5

2nd Run

Chromosome #	2
Fitness (f_i)	18

Chromosome #	3
Fitness (f_i)	14

- Which chromosome is better? Chromosome 2

Get married

Tournament Selection

Con

- If the tournament size s is larger, weak individuals have a smaller chance to be selected. Why?

Pros

- Efficient to code, unlike Fitness Proportionate Selection Methods
- Works on parallel architectures

Roulette Wheel Selection

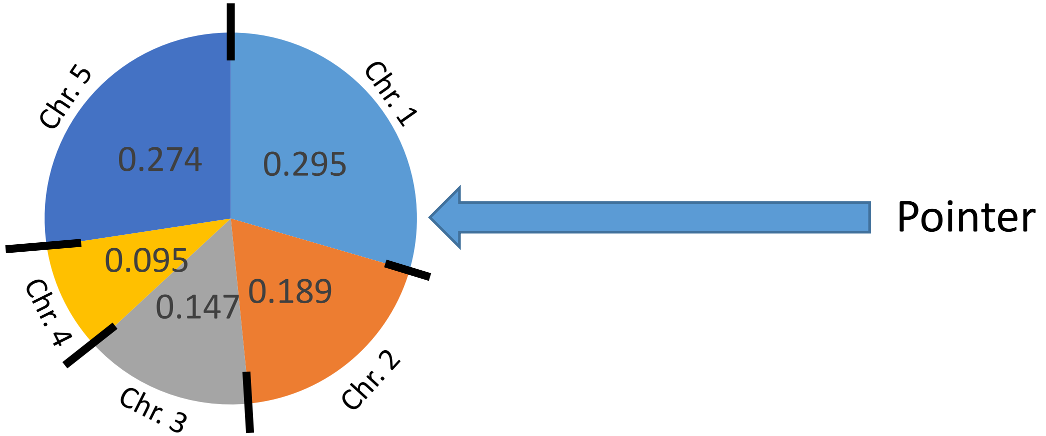
Roulette wheel is one of the simplest and traditional stochastic selection approaches

Proposed by Holland

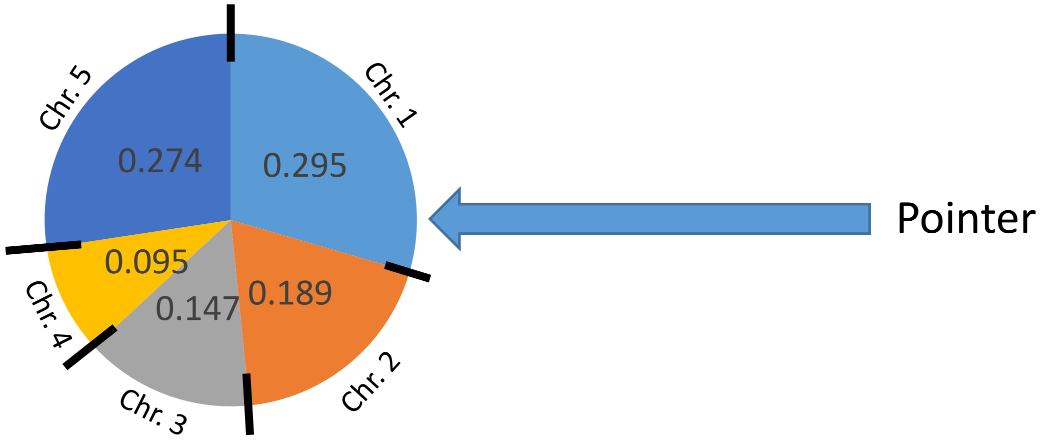
Selects the chromosomes based on probability proportional to the fitness

All the chromosomes (individuals) in the population are placed on the roulette wheel according to their fitness value

- The bigger the value, the larger the segment



Roulette Wheel Selection



The chromosome corresponding to the segment on which the roulette wheel stops is selected

The process is repeated until the desired number of chromosomes is selected

Chromosomes with higher fitness have more probability of selection

Roulette Wheel Selection

Example 1

- Consider a population with 5 chromosomes/ individuals ($n = 5$)
- Fitness values are given in the table below
- Compute the probability p_i of selecting each member of the population:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

$$\sum_{j=1}^{n} f_j = 28 + 18 + 14 + 9 + 26 = 95$$

Chromosome #	1	2	3	4	5
Fitness (f_i)	28	18	14	9	26
Probability	$p_1 = \frac{28}{95}$	$p_2 = \frac{18}{95}$	$p_3 = \frac{3}{95}$	$p_4 = \frac{4}{95}$	$p_5 = \frac{5}{95}$
Probability (p_i)	0.295	0.189	0.147	0.095	0.274

Roulette Wheel Selection

- Compute the cumulative probability q_i of each member of the population:

$$q_i = \sum_{j=1}^{i} p_j$$

Chromosome #	1	2	3	4	5
Fitness (f_i)	28	18	14	9	26
Probability (p_i)	0.295	0.189	0.147	0.095	0.274
Cumulative Probability	0.295	$0.295 + 0.189$	$0.484 + 0.147$	$0.631 + 0.095$	$0.726 + 0.274$

Roulette Wheel Selection

Chromosome #	1	2	3	4	5
Fitness (f_i)	28	18	14	9	26
Probability (p_i)	0.295	0.189	0.147	0.095	0.274
Cumulative Probability (q_i)	0.295	0.484	0.631	0.726	1.000

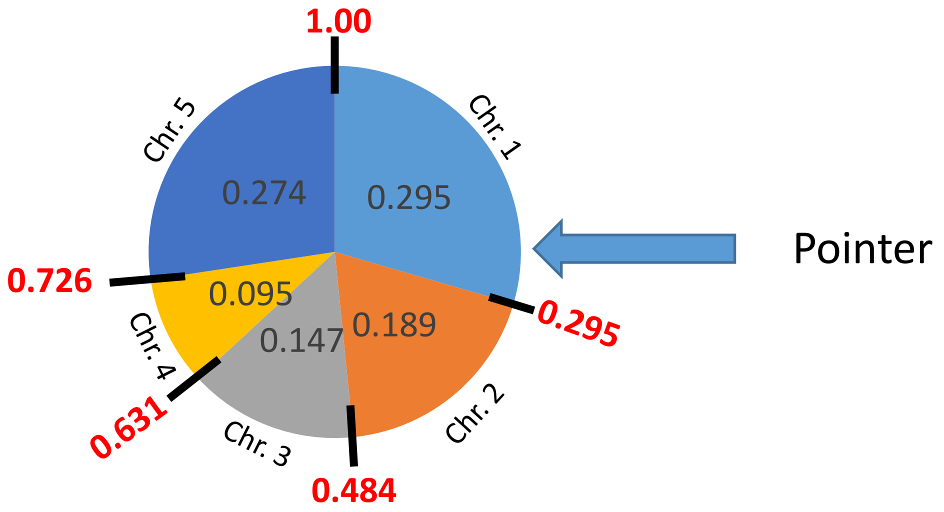
- Final Step: Generate a uniform random number r where $0 < r \leq 1$, e.g. $r = 0.585$, then the third chromosome is selected
- Repeat this step to select m parents (usually $m = 2$)

Roulette Wheel Selection

Illustration

- Final Step: Generate a uniform random number r where $0 < r \leq 1$, e.g. $r = 0.585$, then the third chromosome is selected
- Repeat this step to select m parents (usually $m = 2$)

Chromosome #	1	2	3	4	5
Fitness (f_i)	28	18	14	9	26
Probability (p_i)	0.295	0.189	0.147	0.095	0.274
Cumulative Probability (q_i)	0.295	0.484	0.631	0.726	1.000



Outline of Basic Genetic Algorithm

1: [Start] Generate random population of n chromosomes

2: [Fitness] Evaluate the fitness of each chromosome

3: [New population] Create a new population by repeating:

- **A: [Selection]** Select two parent chromosomes based on their fitness
- **B: [Crossover]** With a crossover probability, cross over the parents to form new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
- **C: [Mutation]** With a mutation probability, mutate new offspring at each locus (position in chromosome).
- **D: [Accepting]** Place new offspring in a new population
- **E: [Fitness]** Evaluate the fitness of each chromosome

4: [Replace] Generate a new population for a further run of algorithm

5: [Test] If the end condition is satisfied, stop, and return the best solution in current population

6: [Loop] Go to step 3

Crossover

Also called **Recombination**

After selection, chromosomes are recombined (crossed over) to create new, hopefully better, chromosomes

Crossover is performed with **high probability**

Many crossover operators used in the literature are problem-specific

Examples include:

- k-point Crossover
- Uniform Crossover
- Uniform Order-based Crossover
- Order-based Crossover
- Partially Matched Crossover (PMX)
- Cycle Crossover (CX)

Crossover Probability

In most recombination operators, two individuals are recombined with a probability p_c, called the **crossover probability**

The value of p_c can be set **experimentally**

A uniform random number r is generated

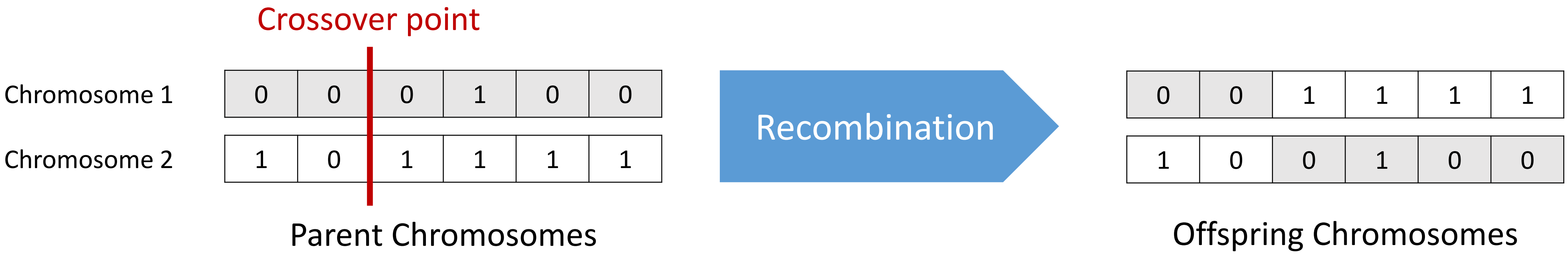
- If $r \leq p_c$, the two randomly selected individuals undergo recombination
- If $r > p_c$, the two offspring are simply copies of their parents

k-point Crossover ($k = 1$)

1-point and **2-point** crossovers are the simplest and most widely-used crossover methods

Example: 1-point crossover

- Two parents are randomly selected
- Crossover point is generated randomly



k-point Crossover ($k = 2$)

1-point and **2-point** crossovers are the simplest and most widely-used crossover methods

Example: 2-point crossover

- Two parents are randomly selected
- Crossover point is generated randomly

Crossover point

Chromosome 1	0	0	0	1	0	0
Chromosome 2	1	0	1	1	1	1

Parent Chromosomes

Recombination

0	0	1	1	1	0
1	0	0	1	0	1

Offspring Chromosomes

Uniform Crossover

Uniform crossover is another common recombination operator

For each position, exchange alleles with a given probability, known as the swapping probability (typically 0.5)

Example: Uniform crossover

- Random numbers are generated, number equal to length of chromosome
- **values $\geq$ swapping probability:** allele is swapped
- **values $<$ swapping probability:** allele is not swapped

Random numbers	0.9	0.4	0.1	0.8	0.1	0.6
Swap?	Y	N	N	Y	N	Y

Parent Chromosomes

Chromosome 1	0	0	0	1	0	0
Chromosome 2	1	0	1	1	1	1

Recombination

Offspring Chromosomes

1	0	1	1	0	1
0	0	0	1	1	0

Outline of Basic Genetic Algorithm

1: [Start] Generate random population of n chromosomes

2: [Fitness] Evaluate the fitness of each chromosome

3: [New population] Create a new population by repeating:

- **A: [Selection]** Select two parent chromosomes based on their fitness
- **B: [Crossover]** With a crossover probability, cross over the parents to form new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
- **C: [Mutation]** With a mutation probability, mutate new offspring at each locus (position in chromosome).
- **D: [Accepting]** Place new offspring in a new population
- **E: [Fitness]** Evaluate the fitness of each chromosome

4: [Replace] Generate a new population for a further run of algorithm

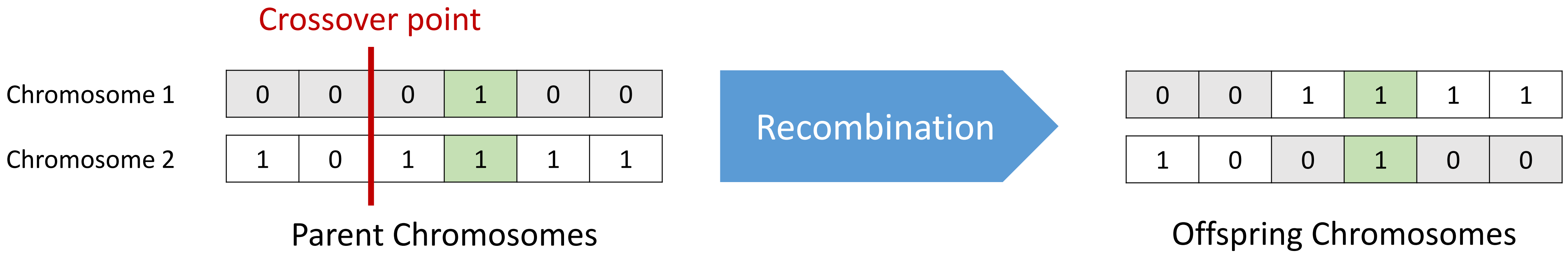
5: [Test] If the end condition is satisfied, stop, and return the best solution in current population

6: [Loop] Go to step 3

Mutation

If we use a crossover operator, such as one-point crossover, we may get better and better chromosomes. But:

- If the two parents (or worse, the entire population) have the same allele at a given gene, then one-point crossover will not change that
- That gene will have the same allele forever



- Mutation is designed to overcome this problem to **add diversity to the population** and ensure that it is possible to explore the entire search space
- Mutation can prevent problems with local minima
- Mutation is performed with **low probability** (e.g. 0.2)

Mutation

One of the most common mutations is the **bit-flip** mutation for binary encoding

Chromosome	1	0	0	1	0	0	Mutation	Offspring	1	0	0	0	0	0

Mutation with **Permutation Encoding**

- Swap: two locations are selected at random, and their values are exchanged

Chromosome	6	2	4	3	1	5	Mutation	Offspring	6	5	4	3	1	2

- Flip: two locations are selected at random, and the values between the locations are flipped

Chromosome	6	2	4	3	1	5	Mutation	Offspring	6	1	3	4	2	5

Mutation

Lancaster University

Mutation with Value Encoding

- **Real Numbers:** add/ subtract a small random number

Chromosome	1.29	5.68	2.86	4.11	Mutation	Offspring	1.29	5.68	2.73	4.11

- **Alphabet symbols:** advance one place

Chromosome	A	F	D	E	Mutation	Offspring	A	G	D	E

Outline of Basic Genetic Algorithm

1: [Start] Generate random population of n chromosomes

2: [Fitness] Evaluate the fitness of each chromosome

3: [New population] Create a new population by repeating:

- **A: [Selection]** Select two parent chromosomes based on their fitness
- **B: [Crossover]** With a crossover probability, cross over the parents to form new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
- **C: [Mutation]** With a mutation probability, mutate new offspring at each locus (position in chromosome).
- **D: [Accepting]** Place new offspring in a new population
- **E: [Fitness]** Evaluate the fitness of each chromosome

4: [Replace] Generate a new population for a further run of algorithm

5: [Test] If the end condition is satisfied, stop, and return the best solution in current population

6: [Loop] Go to step 3

Outline of Basic Genetic Algorithm

1: [Start] Generate random population of n chromosomes

2: [Fitness] Evaluate the fitness of each chromosome

3: [New population] Create a new population by repeating:

- **A: [Selection]** Select two parent chromosomes based on their fitness
- **B: [Crossover]** With a crossover probability, cross over the parents to form new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
- **C: [Mutation]** With a mutation probability, mutate new offspring at each locus (position in chromosome).
- **D: [Accepting]** Place new offspring in a new population
- **E: [Fitness]** Evaluate the fitness of each chromosome

4: [Replace] Generate a new population for a further run of algorithm

5: [Test] If the end condition is satisfied, stop, and return the best solution in current population

6: [Loop] Go to step 3

End Condition

The population converges when:

- either 90% of the chromosomes in the population have the same fitness value
- OR the number of generations is greater than a fixed number
- OR the average fitness value of a population remains fixed for several iterations

Genetic Algorithm Parameters

- Population Size
- Encoding Choices
- Crossover Probability
- Mutation Probability
- Replacement Strategies
- Fitness Function
- End Condition

GA Pros and Cons

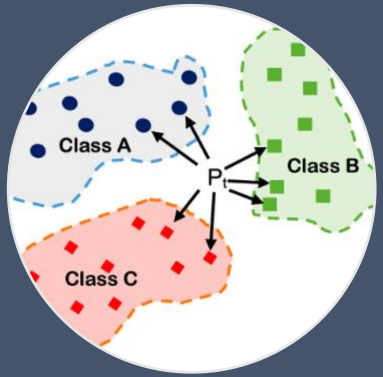
Advantages

- Parallelism
- Less likely to get stuck in local extrema than some other optimisation methods
- Relatively easy to implement

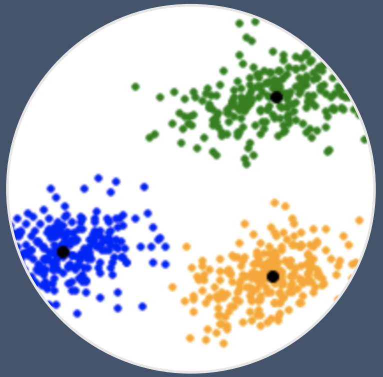
Disadvantages

- Choosing an encoding and fitness function can be difficult
- Computation time
- No guarantee of a solution
- Strong dependence on parameters

Next Four Weeks: Fundamental ML

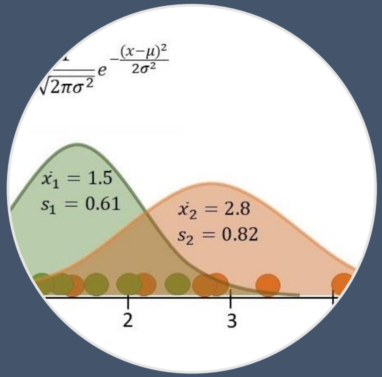


KNN



K-Means

Decision Trees



Naïve Bayes

Generic Algorithms

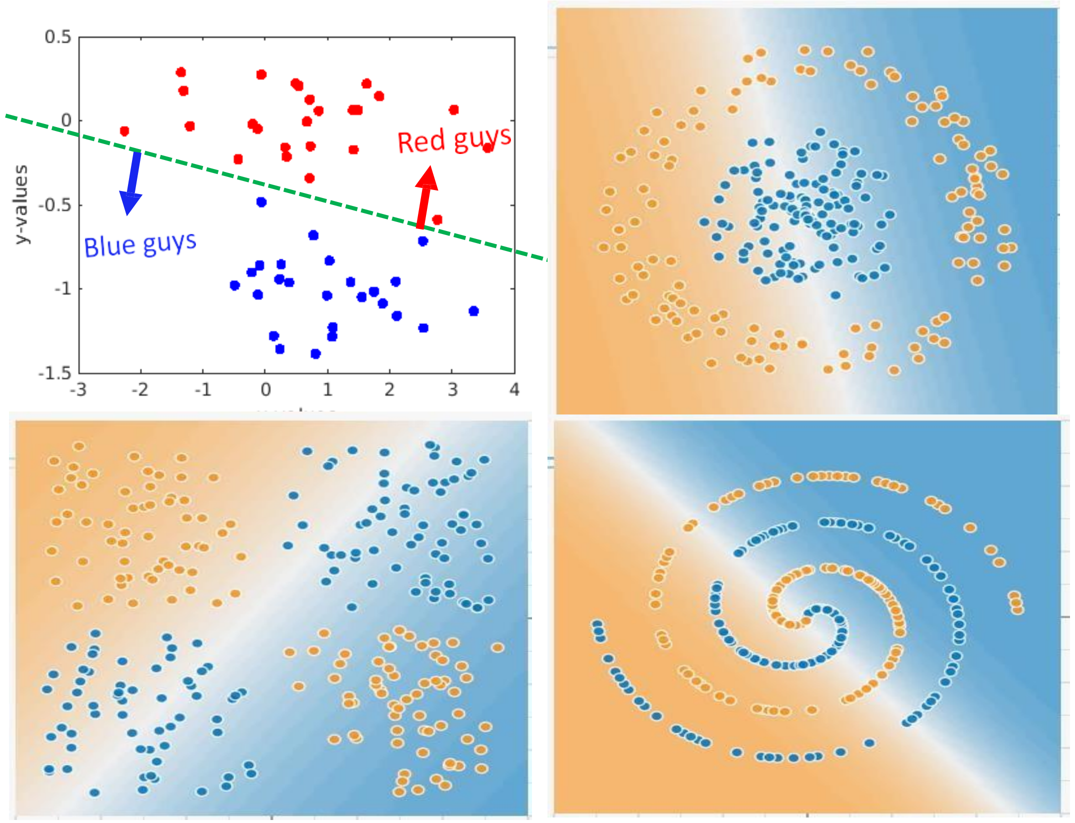
Coming up: Remaining Problems

Typical datasets are large in size

Data points are not usually well separated by lines

Real world datasets are complex

Classic ML Algorithms struggle to learn from this data



Remaining Problem

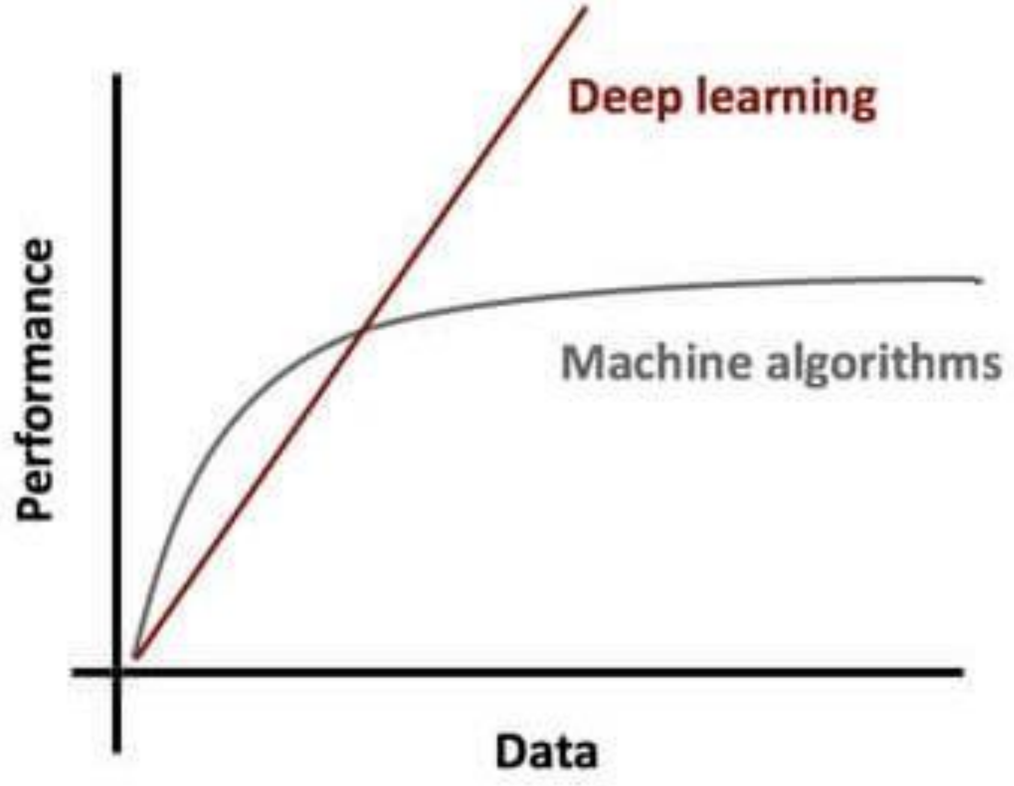
Typical datasets are large in size

Data points are not usually well separated by lines

Real world datasets are complex

ANNs can be scaled to perform well with these difficult data spaces.

Deep neural networks learn better by refining decisions at each layer



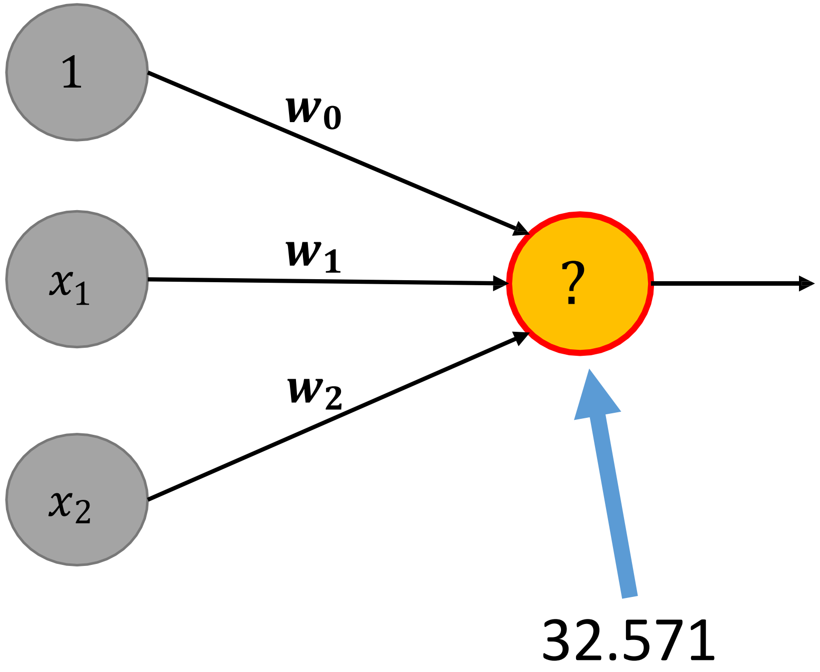
Source: https://en.wikipedia.org/wiki/Neuron

Hypothesis Functions

$$h(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2$$

Given these values for the variables of the hypothesis function

- What will be the outcome of the hypothesis function?
- Will it **rain** or **not**?



x_1	9
x_2	0.83
w_0	1.512
w_1	3.674
w_2	-2.418

Source: [Wikipedia Activation Functions]

Convolutional Neural Networks

- CNNs are usually applied to computer vision tasks i.e. building models that can 'read and understand' images
- Application area includes
  - Image and video recognition
  - Medical image analysis
  - Recommender systems
  - etc
- Key components include:
  - Image convolution
  - Pooling (max pooling)