NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

MSAI-6124 Neuro Evolution & Fuzzy Intelligence

Week 5 – Part 2

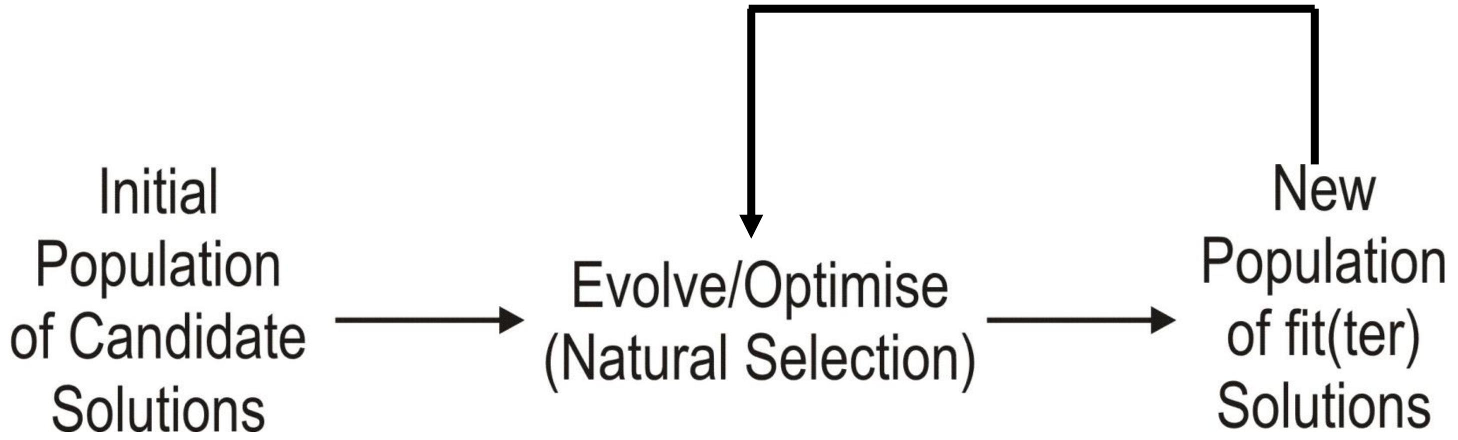
Genetic Algorithm (GA)

Dr WANG Di

wangdi@ntu.edu.sg

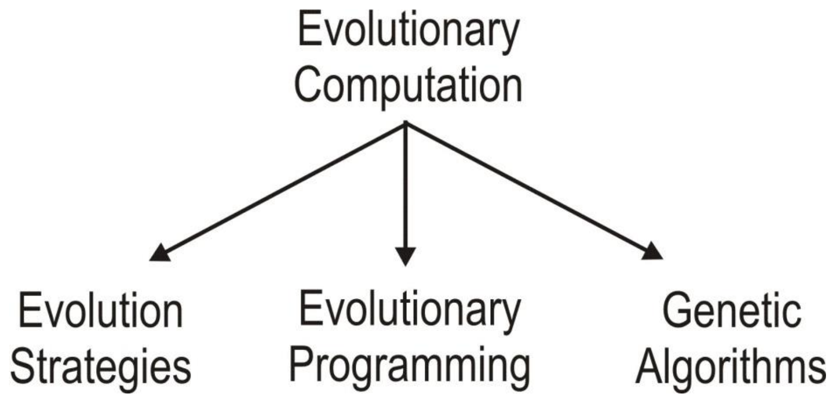
Evolutionary Computation

- Computational systems that use natural evolution (universal Darwinism) as an optimisation mechanism for solving engineering problems


Initial Population of Candidate Solutions
Evolve/Optimise (Natural Selection)
New Population of fit(ter) Solutions

A Brief History

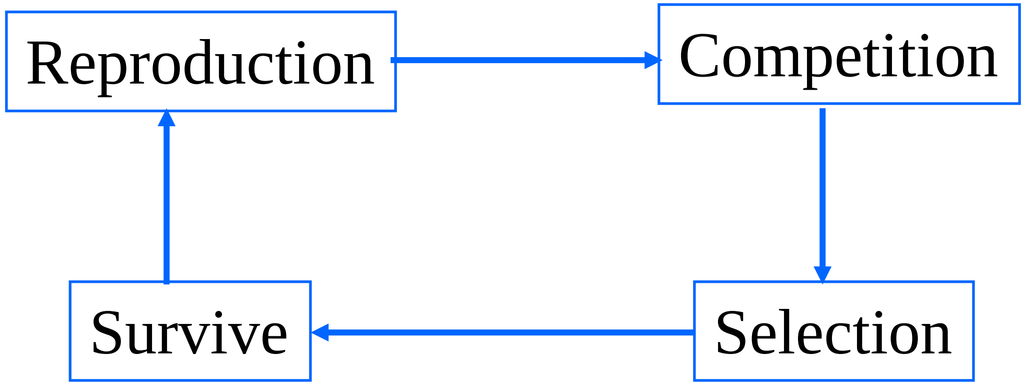
- **Evolution Strategies**: Real value parameter optimisation for device models (1965-1975)
- **Evolutionary Programming**: Evolvable state-transition diagrams (FSM) to produce fit solutions for specific tasks (1966)
- **Genetic Algorithms**: Abstraction and formalisation of natural-adaptation mechanisms for general purpose computations (1962)
  - As opposed to problem-specific algorithm development



- Other independent efforts for evolution-inspired algorithms
  - Simulated Annealing, 1970-1985
  - Ant Colony Optimization (ACO), 1992
  - Particle Swarm, 1995
  - Artificial Bee Colony (ABC), 2005

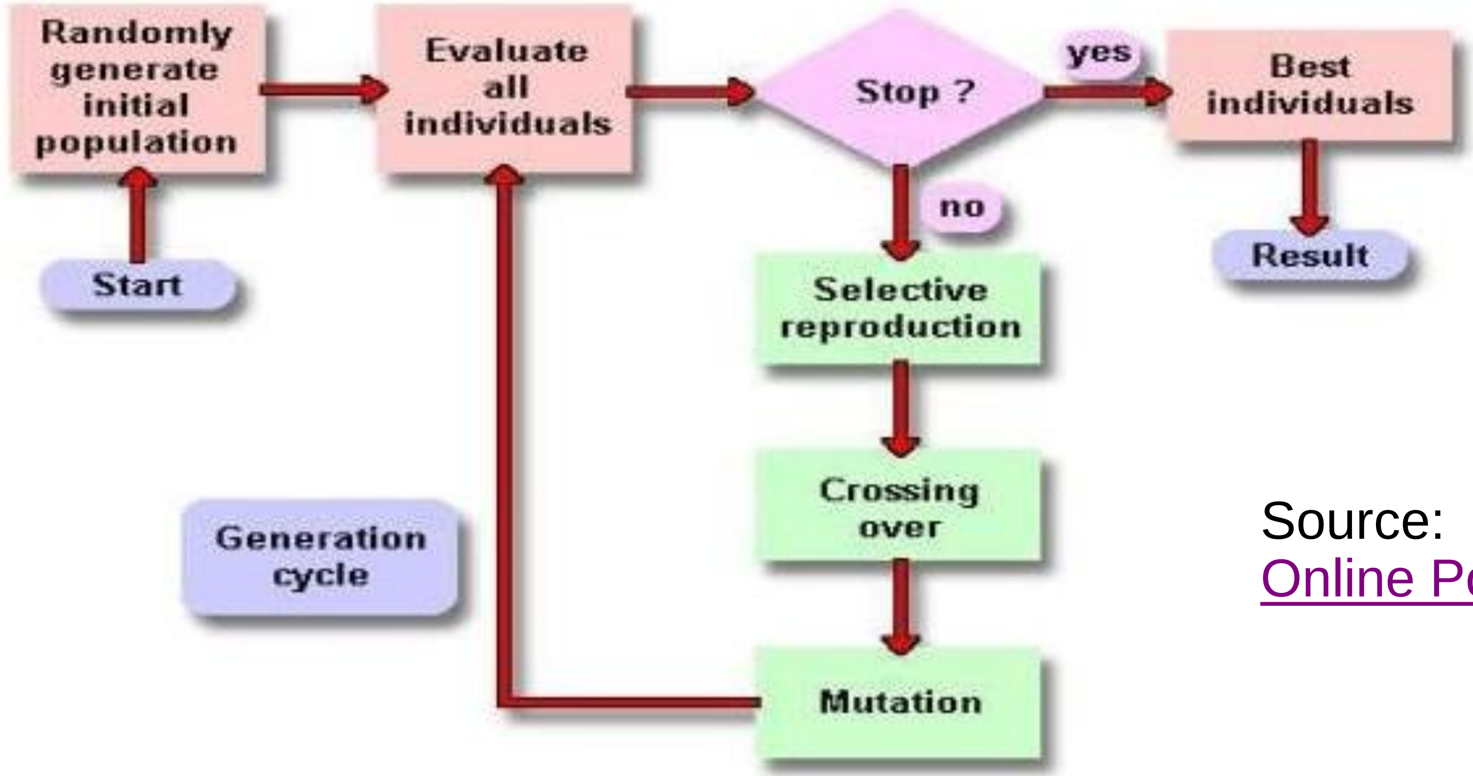
Genetic Algorithm (GA)

- Based on Darwinian Paradigm



- Intrinsically a robust search and optimization mechanism

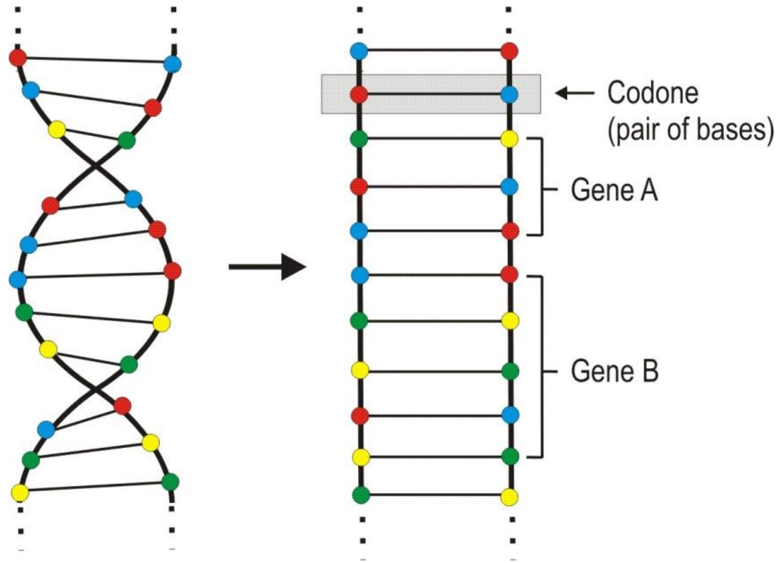
Conceptual Diagram of GA



Source:
Online Post

Biological Foundations of GA

- Living organisms consist of cells
- Each cell contains one or more chromosomes (DNAs & RNAs)
- A set of chromosomes provide the organism blueprint
- A chromosome is divided conceptually in genes (functional blocks of DNA)
- A (set of) gene(s) encodes a protein – a trait (e.g. ,eye color)
- Alleles are the possible encodings of a gene (blue, green, red, yellow)
- Locus is the position of a gene in the chromosome

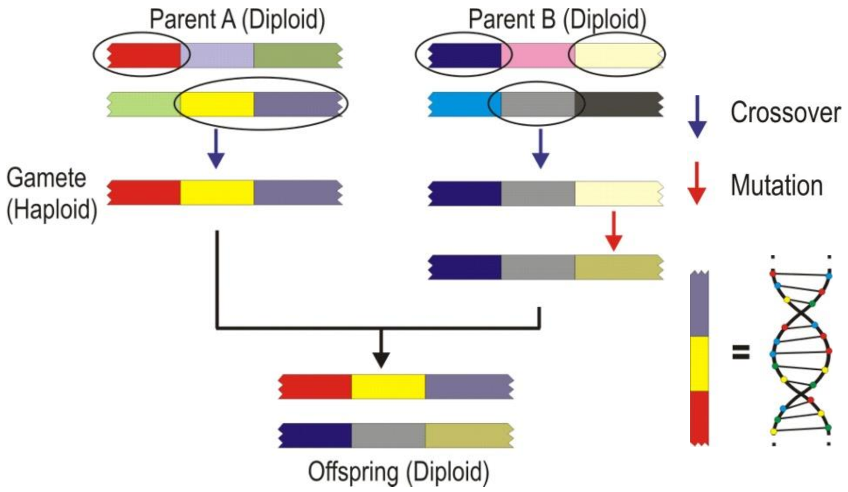


Biological Foundations of GA

- Genome: Complete collection of chromosomes (genetic material)
- Genotype is a particular set of genes (encoded in chromosomes) in the genome that represent the genetic material of an individual
- Phenotype refers to the physical and mental characteristics related to a genotype (eye color, intelligence, height, hair type, etc.) of an individual

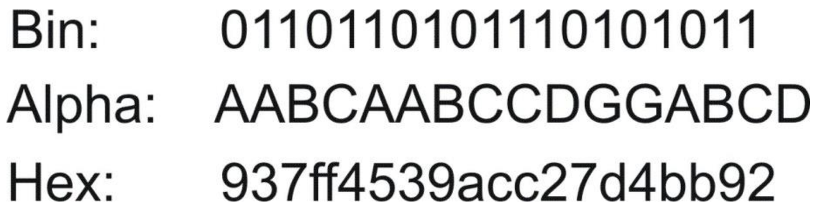
Biological Foundations of GA

- Organisms whose chromosomes appear in pairs (most sexually reproducing species) are called diploid, if not they are called haploid
- During sexual reproduction, genetic recombination (crossover) occurs whereby chromosomes exchange sets of genes to produce a gamete (haploid)
- Mutation is the product of copying errors in the recombination process (biochemecal action, radiation, etc.)
- Genetic fitness refers to the probability that a new organism will survive to reproduce (viability) or the number of offspring an organism has (fertility)



Computational GA Model

- A chromosome is a string representation of a candidate solution to a problem

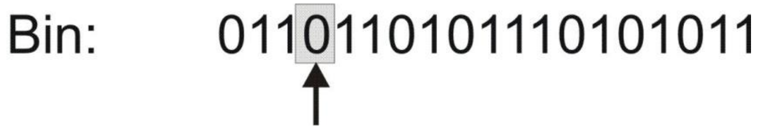


Bin: 0110110101110101011

Alpha: AABCAABCCDGGABCD

Hex: 937ff4539acc27d4bb92

- The genes are single digits or subsets of digits at specific locations in the chromosome string

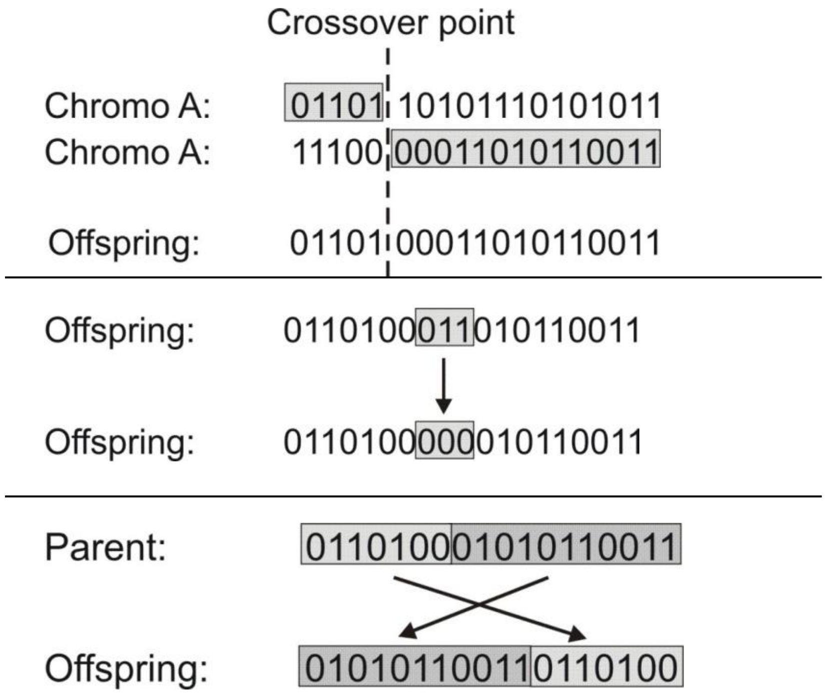


Bin: 0110110101110101011

- An allele refers to the possible values a gene can take (0/1 if binary, a-z if alpha, 0-9 if decimal)

Computational GA Model

- Crossover exchanges substrings between chromosomes
- Mutation replaces a gene value with another from its allele
- Inversion swaps the head with the tail of the chromosome at a locus



Crossover point
Chromo A: 01101 10101110101011
Chromo A: 11100 00011010110011
Offspring: 01101 00011010110011
Offspring: 0110100011010110011
Offspring: 0110100000010110011
Parent: 0110100 01010110011
Offspring: 01010110011 0110100

Key Concepts in GA

- Population of individuals
  - Each represents a feasible solution to the formulated problem
- An individual is characterized by a fitness function
  - A higher fitness value normally means a better solution
- Based on their fitness values, parents are selected to reproduce offspring for a new generation
  - Fitter individuals have more chance to reproduce
  - New generation has same size as old generation; old generation dies
- Offspring has combination of properties of two parents
- If well designed, population will converge to optimal solution

```
BEGIN
  Generate initial population;
  Compute fitness of each individual;
  REPEAT /* New generation /*
    FOR population_size / 2 DO
      Select two parents from old generation;
       /* biased to the fitter ones */
      Recombine parents for two offspring;
      Compute fitness of offspring;
      Insert offspring in new generation
    END FOR
  UNTIL population has converged
END
```

Linkage between GA and Given Problem

- Coding or Representation
  - String with all parameters

→ Encoding of the problem solution

- Fitness function
  - Parent selection

→ Higher fitness value means better solution

- Reproduction
  - Crossover
  - Mutation

→ No direct linkage: Just need to ensure the resulting chromosome is still valid in the problem space

- Convergence
  - When to stop

→ May stop when the problem solution is satisfactory enough

Coding in GA

- Parameters of the solution (genes) are concatenated to form a string (chromosome)
- All kind of alphabets (binary numbers, characters, real numbers, etc.) can be used for a chromosome
- Order of genes on a chromosome can be important
- Generally, different coding strategies for a solution are possible
- Good coding is probably the most important factor for the performance of a GA
- In many cases, many possible chromosomes do not code for feasible solutions

Encoding Procedures

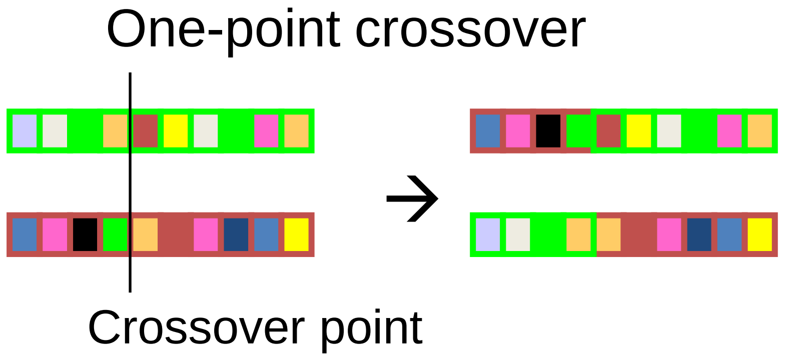
- Design alternative → individual (chromosome)
- Single design choice → gene
- Design objectives → fitness
- Example: Schedule *n* jobs on *m* processors to minimize the overall completion time
  - Design alternative: Job i (i=1,2,…n) is assigned to processor j (j=1,2,…,m)
  - Single design choice: A vector **x** of length *n* such that $x_i = 1, \ldots, \text{or } m$
  - Design objectives: To minimize the job completion time
  - Fitness value: The inverse of the maximal span of any processor

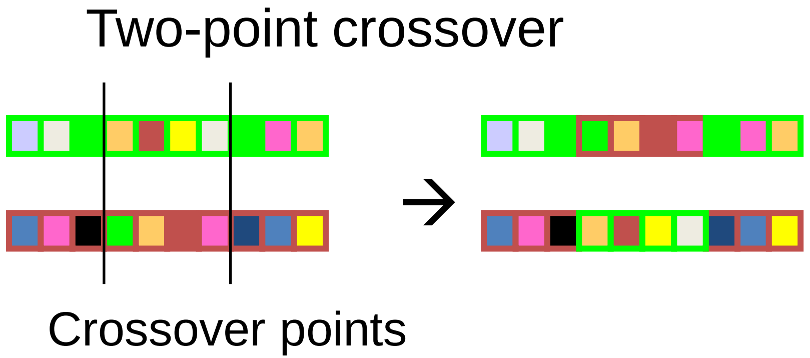
Reproduction Operations

- Crossover
  - Two parents produce two offspring
  - There is a chance that the chromosomes of the two parents are copied unmodified as offspring
  - Otherwise, the chromosomes of the two parents are randomly recombined (crossover) to form offspring
  - Generally, the chance of crossover is between 0.6 and 1.0
- Mutation
  - There is a chance that a gene of a child is changed randomly
  - Generally, the chance of mutation is low (e.g., 0.001)

Crossover Operations

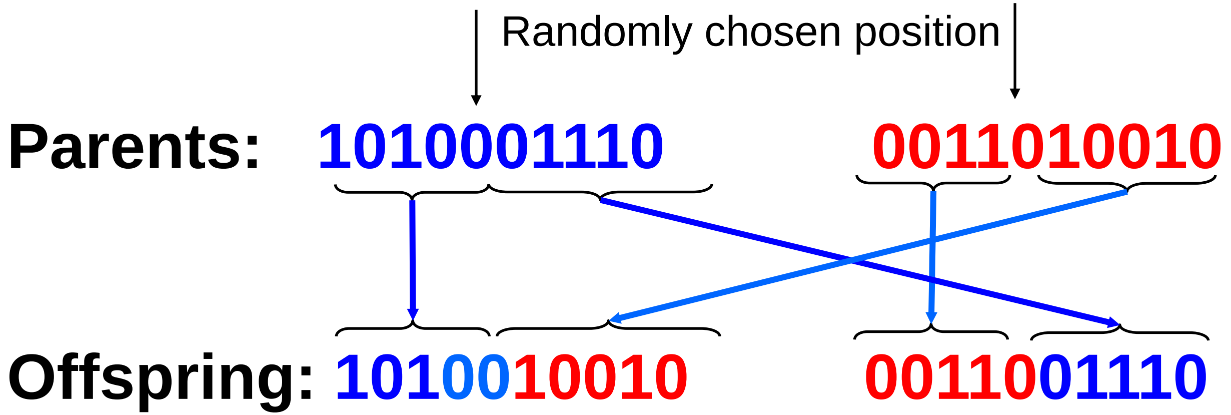
- Generating offspring from two selected parents
  - Single-point (One-point) crossover
  - Two-point crossover, Multi-point crossover
  - Uniform crossover





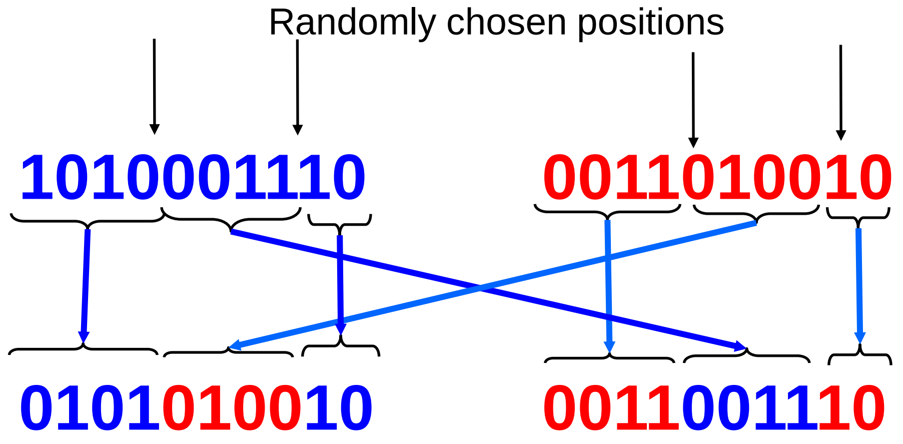
One-point Crossover

- Randomly choose one position in both chromosomes
- Child 1 inherits the head of chromosome of parent 1 with the tail of chromosome of parent 2
- Child 2 inherits the head of 2 with the tail of 1


Randomly chosen position
Parents: 1010001110 0011010010
Offspring: 1010010010 0011001110

Two-point Crossover

- Randomly choose two positions in both chromosomes
- Avoids that genes at the head and genes at the tail of one chromosome are always split when recombined



Parents: 1010001110 0011010010

Offspring: 0101010010 0011001110

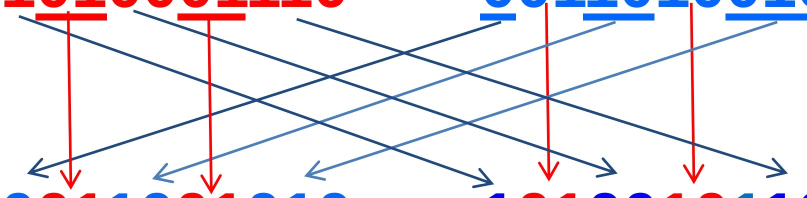
- Easily extensible to multi-point crossover

Uniform Crossover

- A random mask is generated
- The mask determines which bits are copied from one parent and which from the other parent
- Bit density in mask determines how much material is taken from the other parent (takeover parameter)

Mask: **0110011000** **(randomly generated)**

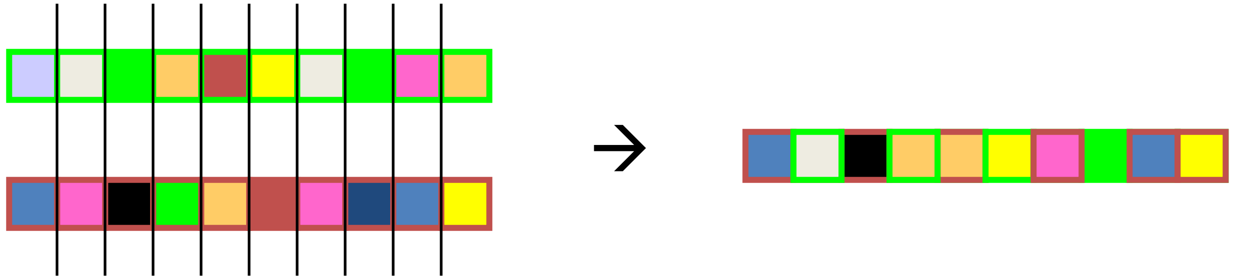
Parents: **1010001110** **0011010010**



Offspring: **0011001010** **1010010110**

Crossover Operators – Comparison

- Uniform crossover:



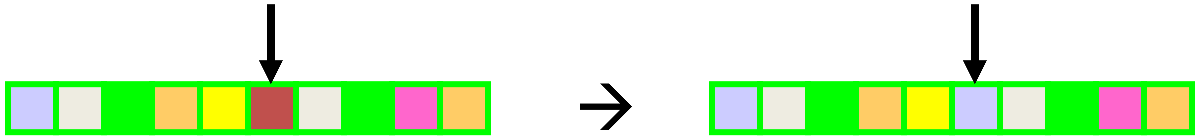
- Is uniform crossover better than one-point or multi-point?
  - Tradeoff to be leveraged
    - Exploration: Introduction of new combination of features
    - Exploitation: Keep the good features in the existing solution

Problems with Crossover

- Depending on the coding strategy, simple crossovers can have high chance to produce illegal offspring
  - E.g., in TSP (Travelling Salesman Problem) with simple binary or path coding, most offspring will be illegal because not all cities will be in the offspring and some cities will be there more than once
- Uniform crossover can often be modified to avoid this problem
  - E.g., in TSP with simple path coding:
    - Where mask is 1, copy cities from one parent
    - Where mask is 0, choose the remaining cities in the order of the other parent

Mutation Operation

- Generating new offspring from single parent/offspring



- Maintaining the diversity of the individuals
  - Crossover can only explore the combinations of the current gene pool
  - Mutation can "generate" new genes

Other Reproduction Operators

- Other control parameters: **population size, crossover probability, mutation probability**
  - Problem specific
  - Increase population size
    - Increase diversity and computation time for each generation
  - Increase crossover probability
    - Increase the opportunity for recombination but also disruption of good combination
  - Increase mutation probability
    - Closer to random search
    - Help to introduce new gene or reintroduce the lost gene
- Usual reproduction approach
  - Use crossover operator to generate new chromosomes
  - Then use mutation operator on new chromosomes

Survivor Selection

- Strategy
  - Always keep the best ones, with caution
  - Elitism: Select best performing parents and keep them (unchanged) in the new generation
- Dilemma:
  - Too strong fitness selection bias may lead to sub-optimal solution
  - Too little fitness bias selection may result in unfocused and meandering search
  - Usual approach: Define an elitism ratio, e.g., 10%

Parent Selection

- Chance to be selected as parent is proportional to fitness
- Roulette wheel selection:
  - Select *n* parents by rotating the wheel *n* times
- Tournament selection:
  - Select *k* individuals each time, and run a tournament (competition) among them
- To suppress the dominance of highly fit species, can apply scaling and ranking of the fitness values

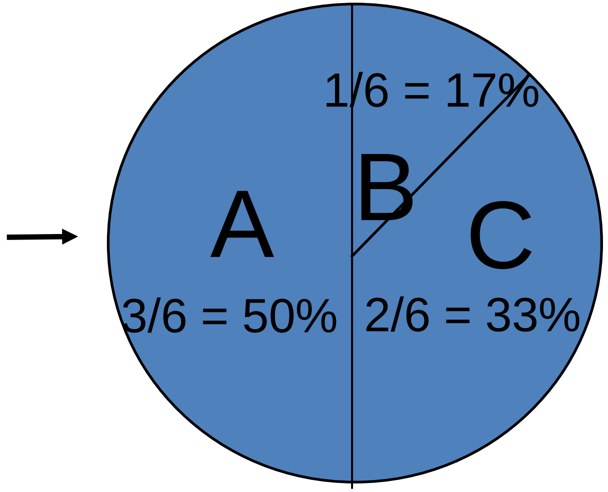
Roulette Wheel Selection

- Main idea: Fitter individuals get higher chances
  - Chances proportional to fitness values
  - Implementation: Roulette wheel technique
    - Assign a part of the roulette wheel to each individual
    - Spin the wheel n times to select n individuals

fitness(A) = 3

fitness(B) = 1

fitness(C) = 2



Roulette Wheel Selection

- Sum the fitness of all chromosomes, call it *T*
- Generate a random number N between 1 and *T*
- Return chromosome whose fitness added to the running total is equal to or larger than N
- Chance to be selected is exactly proportional to fitness

```
Chromosome   :  1  2  3  4  5  6
Fitness:        8  2  17 7  4  11
Running total:  8  10 27 34 38 49
N (1 ≤ N ≤ 49):       23
Selected:             3
```

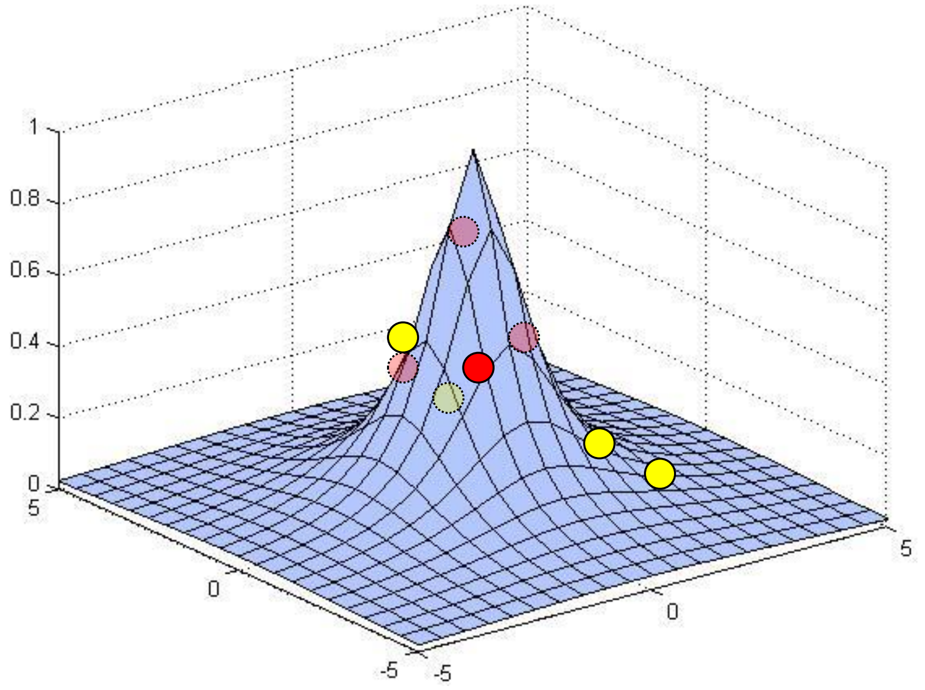
Tournament Selection

- Roulette wheel selection relies on the global population statistics
  - Highly fit individuals quickly dominate the population
- Tournament selection procedure:
  - Pick *k* members at random then select the best among them
  - Repeat to select more individuals
  - Work well even when the fitness function is not carefully designed

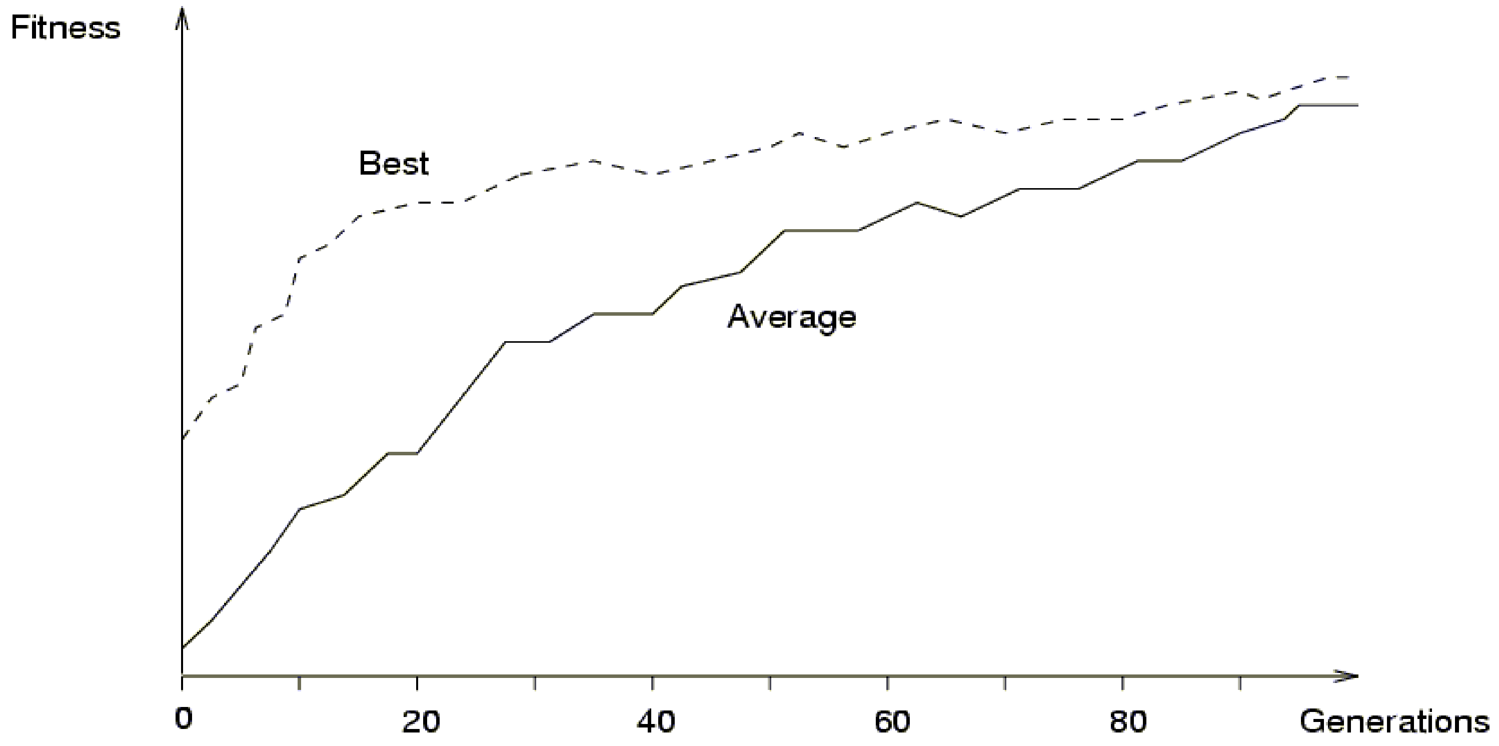
Example: Search Space & Fitness Landscapes

$$F(x,y) = \frac{1}{1 + x^2 + y^2}$$

	X	Y
C1	-1	2
C2	-2	3
C3	1.5	0
C4	0.5	-1



Example of Convergence in GA



JAVA Demos of GA

- GA Viewer

http://www.rennard.org/alife/english/gavgb.html

- GA Experimenter

http://www.oursland.net/projects/PopulationExperiment/

- Simulated World Evolution

http://math.hws.edu/xJava/GA/

Python Resources

- Genetic Algorithm (GA) Introduction with Example Code https://pub.towardsai.net/genetic-algorithm-ga-introduction-with-example-code-e59f9bc58eaf

- 5 Genetic Algorithm Applications Using PyGAD https://blog.paperspace.com/genetic-algorithm-applications-using-pygad/

Incorporate GA into FNN

- GA can be used to tune the fuzzy membership functions used in Fuzzy Neural Networks (FNNs) (see example here)

- GA can also used to tune other parameters of FNNs, such as network architecture (see examples here and here)