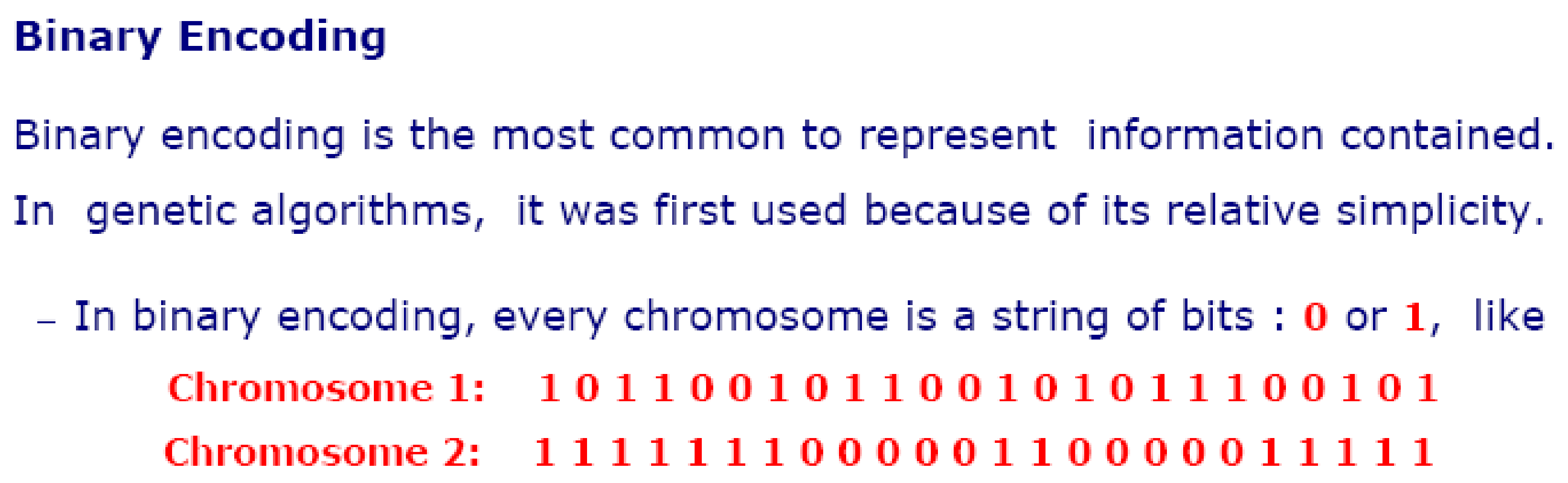
GENETIC ALGORITHMS

Encoding Schemes

Representing an Individual

- An individual is data structure representing the “genetic structure” of a possible solution.

- Genetic structure consists of an alphabet (usually 0,1)


Binary Encoding

Binary encoding is the most common to represent information contained.

In genetic algorithms, it was first used because of its relative simplicity.

– In binary encoding, every chromosome is a string of bits : 0 or 1, like

Chromosome 1: 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1

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

Example Problem: Knapsack problem

The problem: there are things with given value and size. The knapsack has given capacity. Select things to maximize the values.

Encoding: Each bit says, if the corresponding thing is in the knapsack

0-1 Knapsack Problem

- There are *n* items, each item has its own cost (c_i) and weight (w_i).
- There is a knapsack of total capacity *w*.
- The problem is to take as much items as possible but not exceeding the capacity of the knapsack.

This is an optimization problem and can be better described as follows.

Maximize :

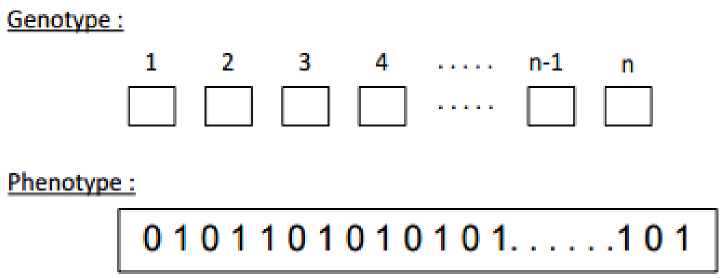
$$\sum_i c_i * w_i * x_i$$

Subject to

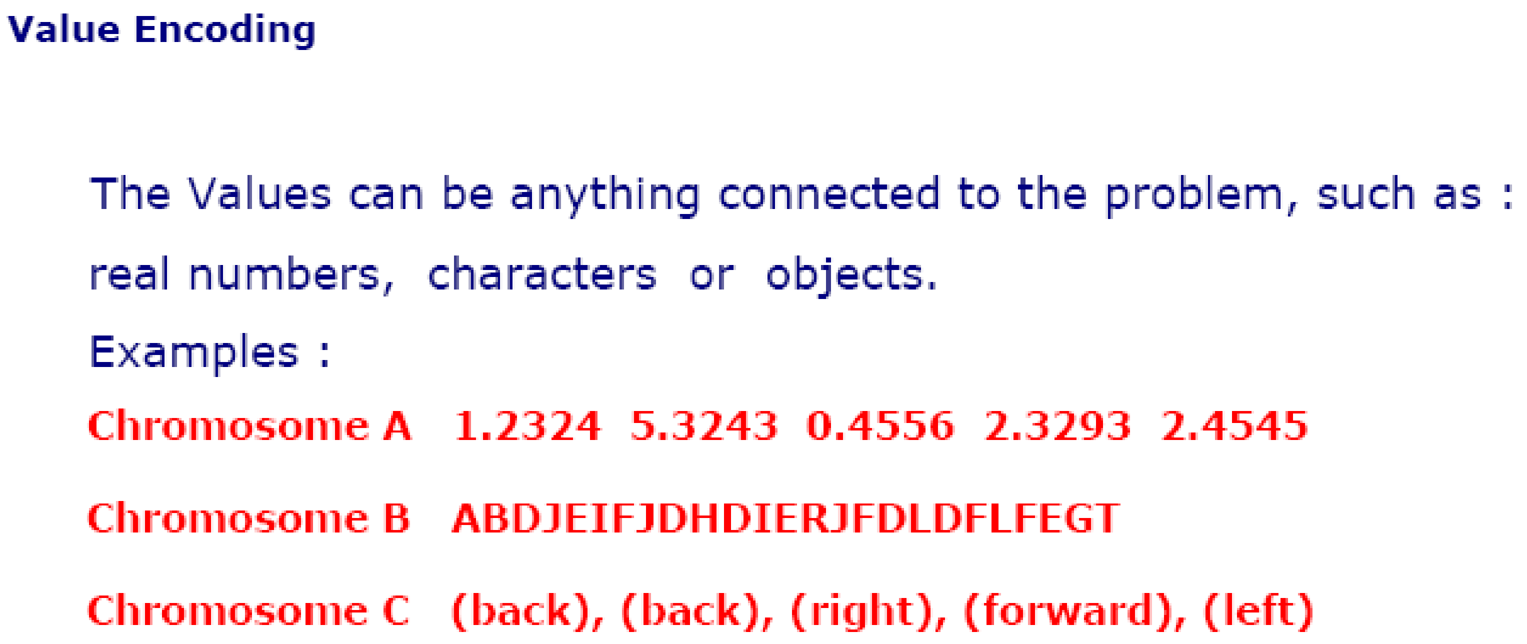
$$\sum x_i * w_i \leq W$$

where $x_i \in [0 \cdots 1]$

The encoding for the 0-1 Knapsack, problem, in general, for *n* items set would look as follows.



A binary string of n-bits



Value Encoding

The Values can be anything connected to the problem, such as :

real numbers, characters or objects.

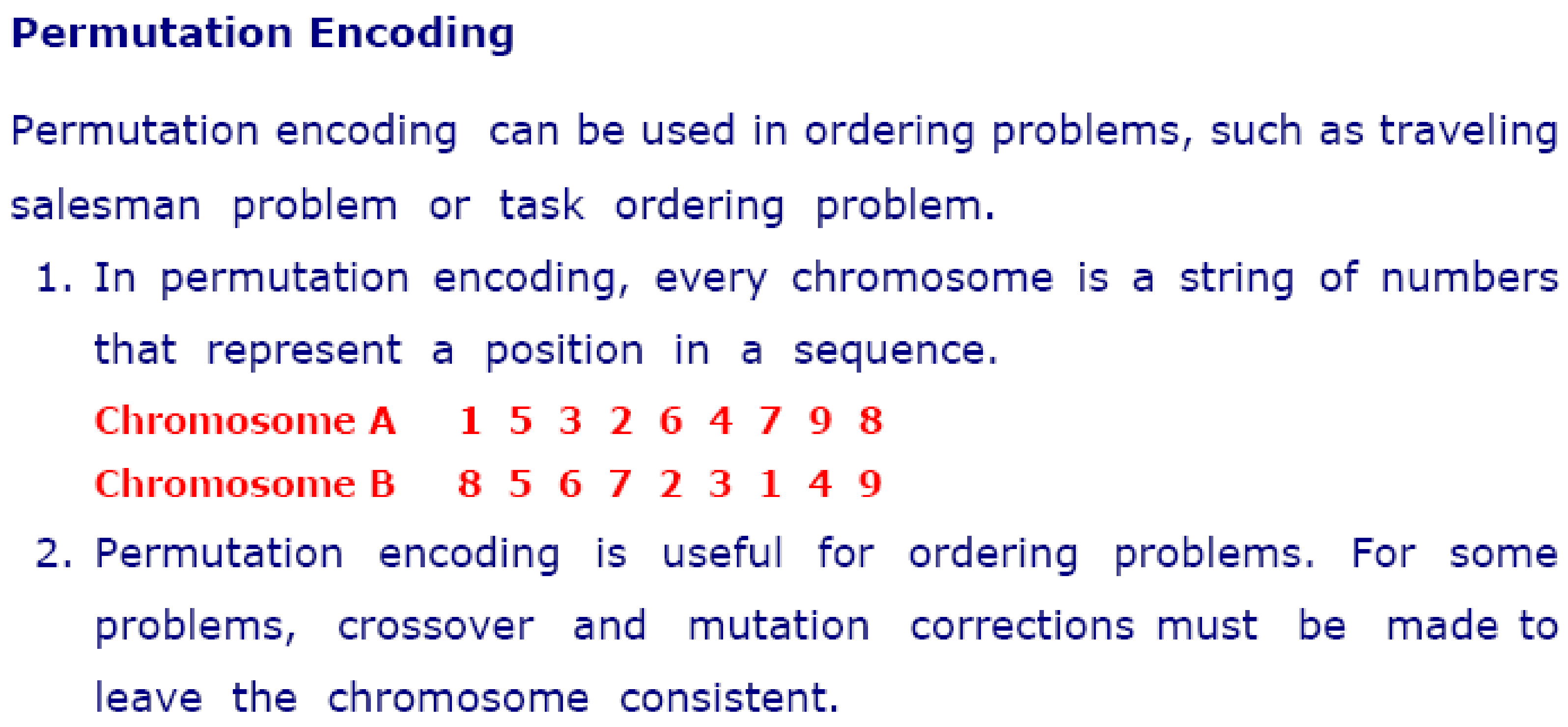
Examples :

Chromosome A 1.2324 5.3243 0.4556 2.3293 2.4545

Chromosome B ABDJEIFJDHDIERJFDLDFLFEGT

Chromosome C (back), (back), (right), (forward), (left)

Example: Finding weights for neural nets.
The problem: find weights for network
Encoding: Real values that represent weights



Permutation Encoding

Permutation encoding can be used in ordering problems, such as traveling salesman problem or task ordering problem.

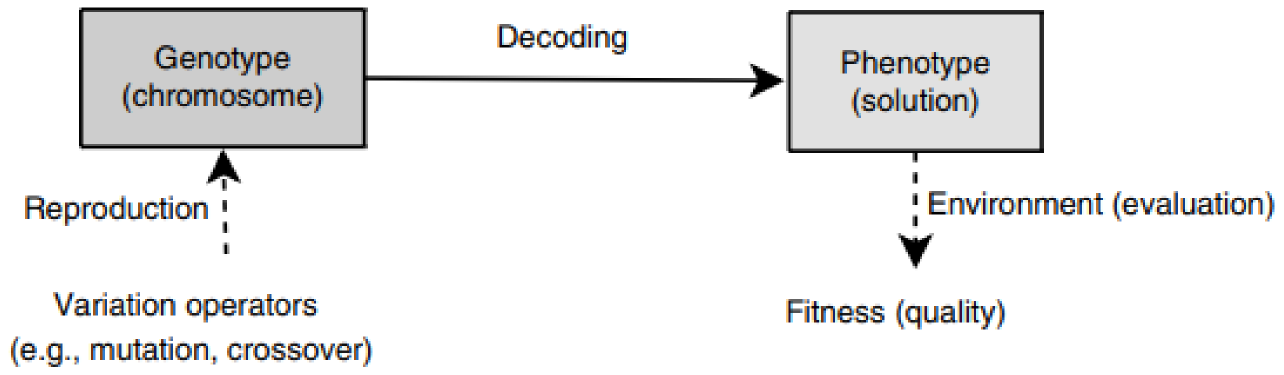
1. In permutation encoding, every chromosome is a string of numbers that represent a position in a sequence.

Chromosome A	1	5	3	2	6	4	7	9	8
Chromosome B	8	5	6	7	2	3	1	4	9

2. Permutation encoding is useful for ordering problems. For some problems, crossover and mutation corrections must be made to leave the chromosome consistent.

Example: Travelling salesman problem
The problem: cities that must be visited.
Encoding says order of cities in which salesman will visit.

Genotype vs Phenotype



Initialization

- Initialization is the process of making the first generation.
- During the algorithm our goal will be to improve them by imitating the nature.

How offspring are produced - Reproduction

- *Reproduction*- Through **reproduction**, genetic algorithms produce new generations of improved solutions by selecting parents with higher fitness ratings or by giving such parents a greater probability of being contributors and by using random selection

Selection Algorithms

Genetic Operators: Parent Selection Methods

In EAs, fitness assignment to individuals may take two different ways:

- **Proportional fitness assignment** in which the absolute fitnesses are associated with individuals.
- **Rank-based fitness assignment** in which relative fitnesses are associated with individuals. For instance, a rank in the population is associated with each individual according to its rank in a decreasing sorting of individuals.

Proportionate Selection

- In Proportionate Selection, individuals are assigned a probability of being selected based on their fitness:

$$p_i = f_i / \Sigma f_j$$

  - Where p_i is the probability that individual i will be selected,
  - f_i is the fitness of individual i, and
  - Σf_j represents the sum of all the fitnesses of the individuals with the population.
- This type of selection is similar to using a roulette wheel where the fitness of an individual is represented as proportionate slice of wheel. The wheel is then spun and the slice underneath the wheel when it stops determine which individual becomes a parent.

Roulette-wheel selection: Implementation

Input: A Population of size *N* with their fitness values
Output: A mating pool of size N_p

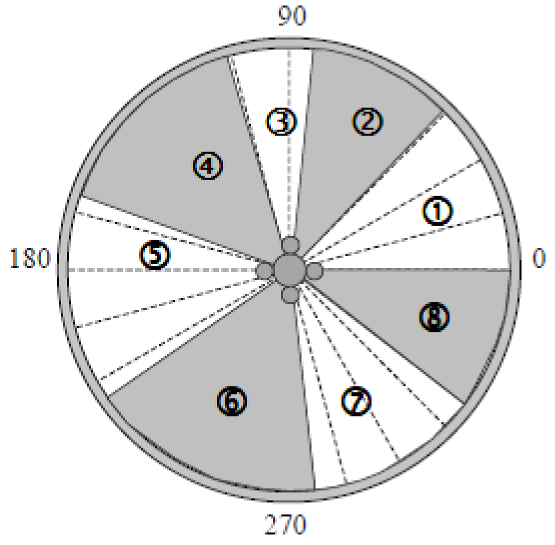
Steps:

- Sum up the fitness of all individuals, $f_T = \Sigma_i f_i$, $i = 1, \dots, N$.
- Multiply the fitness of each individual by 360 and divide it by the total fitness: $f_i' = (360.\, f_i)/f_T$, $i = 1, \dots, N$, determining the portion of the wheel to be assigned for each individual.
- Sort a random number in the interval (0,360] and compare it with the interval belonging to each individual. Select the individual whose interval contains the sorted number and place it in the new population **P**.

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{x}_1 = [010010010010]$$

Individual	Chromosome	Fitness	Degree	Portion of the Roulette
1	000010111101	6	46	(0,46]
2	100111111011	5	38	(46,84]
3	100001100101	3	23	(84,107]
4	100100010000	7	54	(107,161]
5	110101000010	7	54	(161,215]
6	110110001010	8	61	(215,276]
7	100101010110	6	46	(276,322]
8	111011000101	5	38	(322,360]



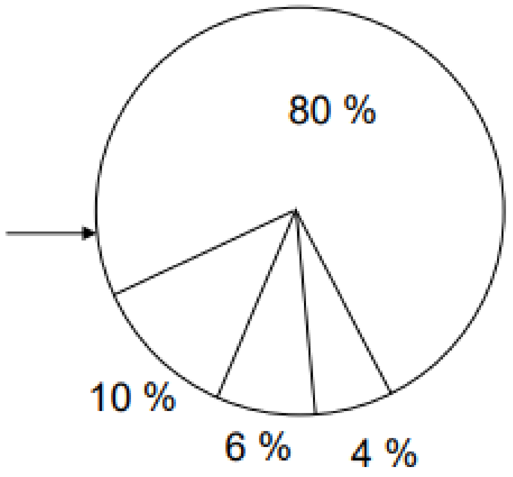
Assuming that $s = [230,46,175,325,275,300,74,108]$

$$P = [x_6, x_1, x_5, x_8, x_6, x_7, x_2, x_4]^T,$$

Drawback of Fitness Proportionate Selection

- Suppose, there are only four binary string in a population, whose fitness values are f_1, f_2, f_3 and f_4.
- Their values 80%, 10%, 6% and 4%, respectively.

Individual (i)	Fitness (fi)	RW (Area)
1	0.4	80 %
2	0.05	10 %
3	0.03	6 %
4	0.02	4 %



The observation is that the individual with higher fitness values will guard the other to be selected for mating. This leads to a lesser diversity and hence fewer scope toward exploring the alternative solution and also premature convergence or early convergence with local optimal solution.

Rank-based Selection

- To overcome the problem with Roulette-Wheel selection, a rank-based selection scheme has been proposed.
- The process of ranking selection consists of two steps.
  1. Individuals are arranged in an ascending order of their fitness values. The individual, which has the lowest value of fitness is assigned rank 1, and other individuals are ranked accordingly.
  2. The proportionate based selection scheme is then followed based on the assigned rank.

Note:

- The % area to be occupied by a particular individual i, is given by

$$\frac{r_i}{\sum_{i=1}^{N} r_i} \times 100$$

  where r_i indicates the rank of $i - th$ individual.
- Two or more individuals with the same fitness values should have the same rank.

Rank-based selection: Implementation

Input: A population of size N with their fitness values
Output: A mating pool of size N_p.

Steps:

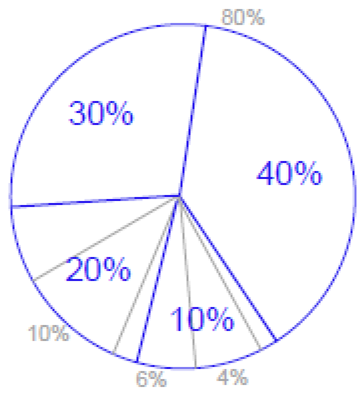
1. Arrange all individuals in ascending order of their fitness value.
2. Rank the individuals according to their position in the order, that is, the worst will have rank 1, the next rank 2 and best will have rank N.
3. Apply the Roulette-Wheel selection but based on their assigned ranks. For example, the probability p_i of the i-th individual would be

$$p_i = \frac{r_i}{\sum_{j=1}^{i} r_j}$$

4. Stop

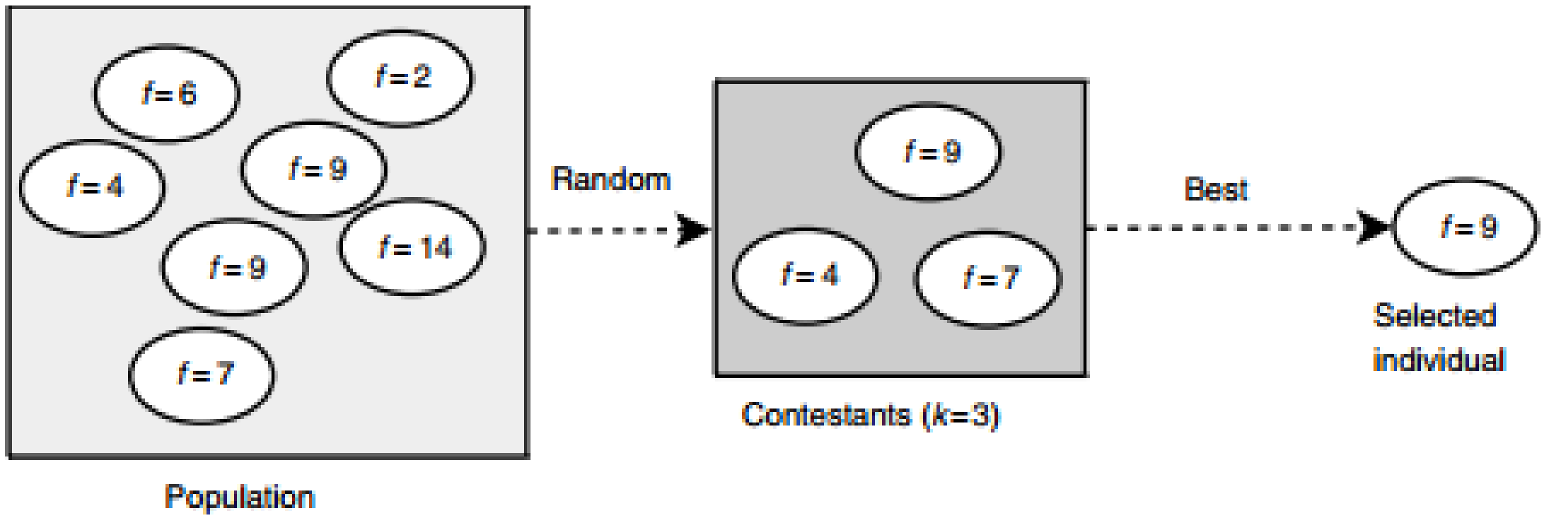
- Continuing with the population of 4 individuals with fitness values: $f_1 = 0.40$, $f_2 = 0.05$, $f_3 = 0.03$ and $f_4 = 0.02$.
- Their proportionate area on the wheel are: 80%, 10%, 6% and 4%
- Their ranks are shown in the following figure.

Individual (i)	Fitness (fi)	RW (Area)	Rank	RS (Area)
1	0.4	80 %	4	40 %
2	0.05	10 %	3	30 %
3	0.03	6 %	2	20 %
4	0.02	4 %	1	10 %



Tournament Selection

1. In this scheme, we select the tournament size *n* (say 2 or 3) at random.
2. We pick *n* individuals from the population, at random and determine the best one in terms of their fitness values.
3. The best individual is copied into the mating pool.
4. Thus, in this scheme only one individual is selected per tournament and N_p tournaments are to be played to make the size of mating pool equals to N_p.



$$N = 8, N_U = 2, N_p = 8$$

Input :

Individual	1	2	3	4	5	6	7	8
Fintess	1.0	2.1	3.1	4.0	4.6	1.9	1.8	4.5

Output :

Trial	Individuals	Selected
1	2, 4	4
2	3, 8	8
3	1, 3	3
4	4, 5	5
5	1, 6	6
6	1, 2	2
7	4, 2	4
8	8, 3	8

Effectiveness of any Selection Algorithm

Population diversity

- This is similar to the concept of exploration.
- The population diversity means that the genes from the already discovered good individuals are exploited while permitting the new area of search space continue to be explored.

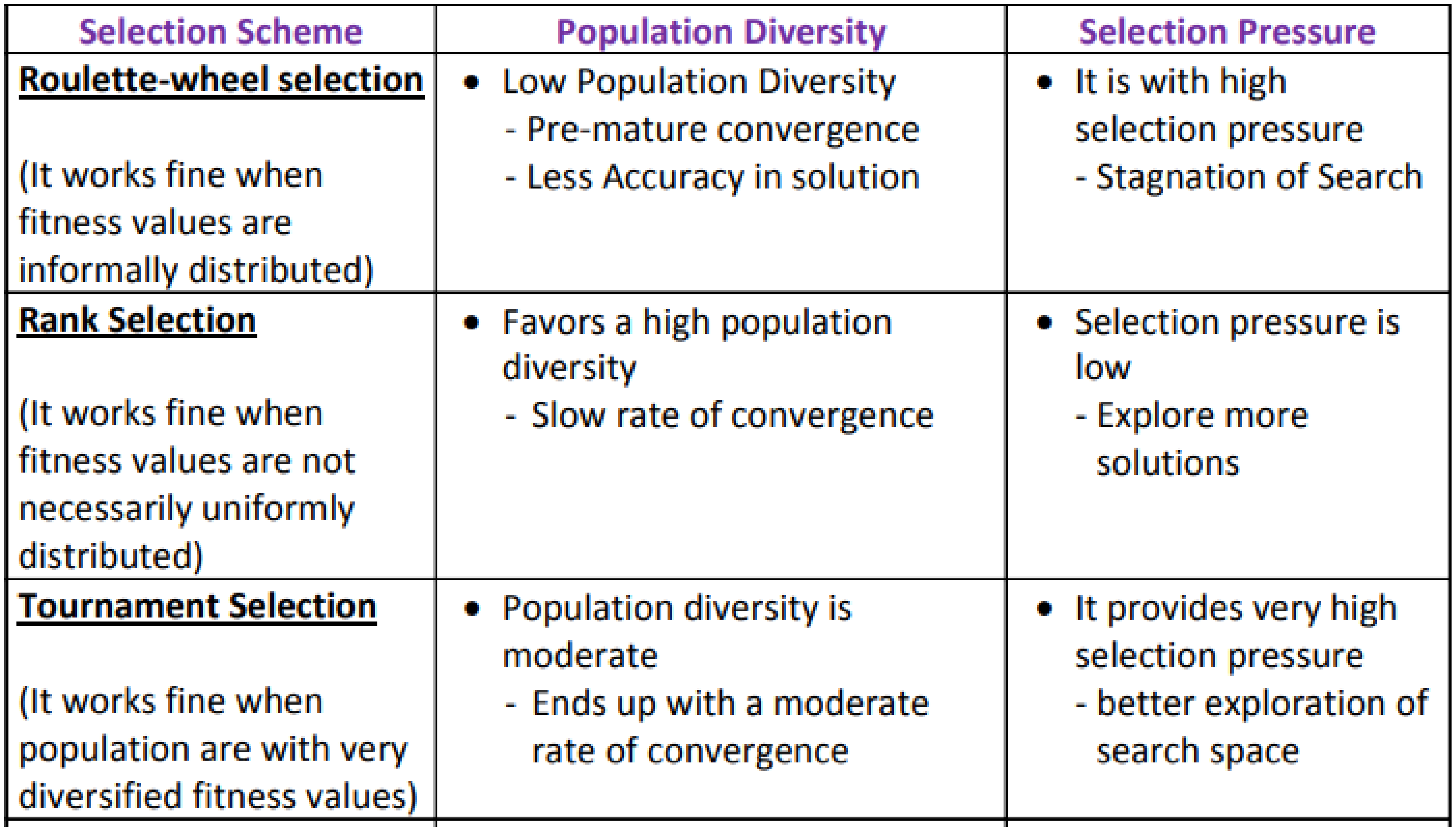
Selection pressure

- This is similar to the concept of exploitation.
- It is defined as the degree to which the better individuals are favoured.

Effectiveness of any Selection Algorithm

These two factors are inversely related to each other in the sense that if the selection pressure increases, the population diversity decrease and vice-versa. Thus,

1. **If selection pressure is HIGH**
   - The search focuses only on good individuals (in terms of fitness) at the moment.
   - It loses the population diversity.
   - Higher rate of convergence. Often leads to pre-mature convergence of the solution to a sub-optimal solution.
2. **If the selection pressure is LOW**
   - May not be able to drive the search properly and consequently the stagnation may occurs.
   - The convergence rate is low and GA takes unnecessary long time to find optimal solution.
   - Accuracy of solution increases (as more genes are usually explored in the search).

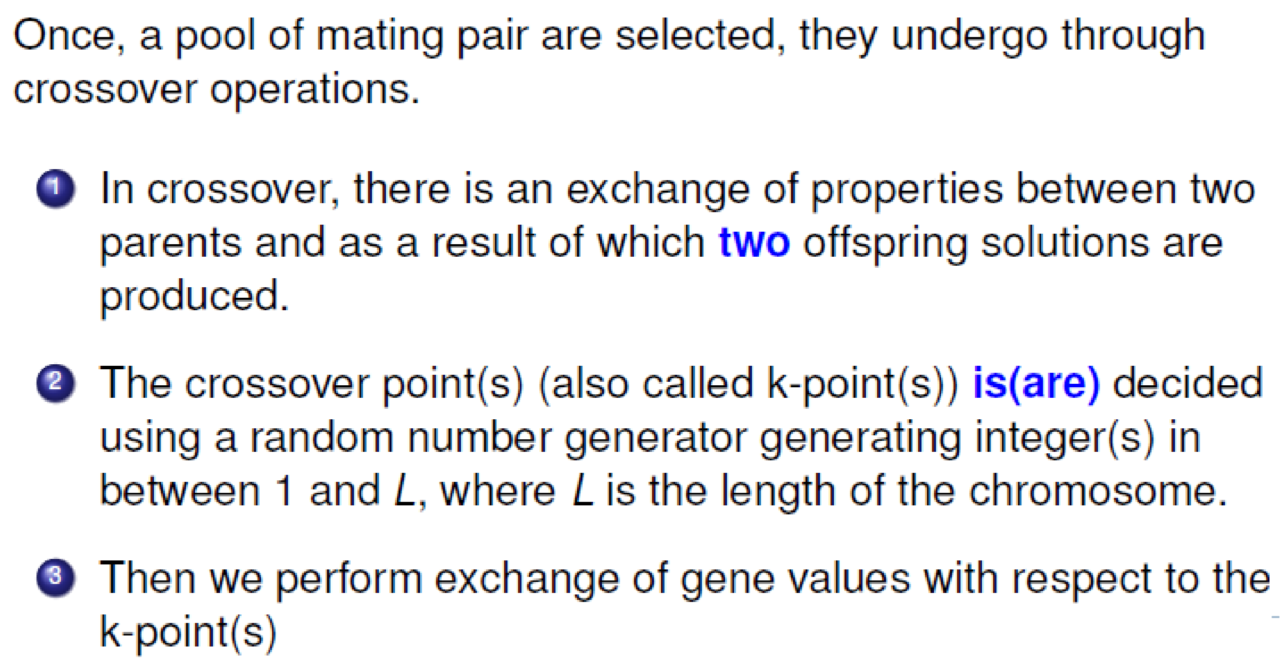


Selection Scheme	Population Diversity	Selection Pressure
Roulette-wheel selection (It works fine when fitness values are informally distributed)	• Low Population Diversity - Pre-mature convergence - Less Accuracy in solution	• It is with high selection pressure - Stagnation of Search
Rank Selection (It works fine when fitness values are not necessarily uniformly distributed)	• Favors a high population diversity - Slow rate of convergence	• Selection pressure is low - Explore more solutions
Tournament Selection (It works fine when population are with very diversified fitness values)	• Population diversity is moderate - Ends up with a moderate rate of convergence	• It provides very high selection pressure - better exploration of search space

Crossover Techniques

Genetic Procreation Operators

- Genetic Algorithms typically use two types of operators:
  - Crossover (Recombination), and
  - Mutation
- Crossover is usually the primary operator with mutation serving only as a mechanism to introduce diversity in the population.



Once, a pool of mating pair are selected, they undergo through crossover operations.

1. In crossover, there is an exchange of properties between two parents and as a result of which **two** offspring solutions are produced.
2. The crossover point(s) (also called k-point(s)) **is(are)** decided using a random number generator generating integer(s) in between 1 and L, where L is the length of the chromosome.
3. Then we perform exchange of gene values with respect to the k-point(s)

Crossover operators

Some important points must be taken into account in the design or use of a crossover operator:

- **Heritability**: The main characteristic of the crossover operator is heritability. The crossover operator should inherit a genetic material from both parents.
- **Validity**: The crossover operator should produce valid solutions. This is not always possible for constrained optimization problems.

Crossover Operators for Binary Coded GA and Value Coded GAs

- One point crossover
- Two-point crossover
- Multi-point crossover (also called n-point crossover)
- Uniform crossover (UX)
- Half-uniform crossover (HUX)
- Shuffle crossover
- Matrix crossover (Tow-dimensional crossover)
- Three parent crossover

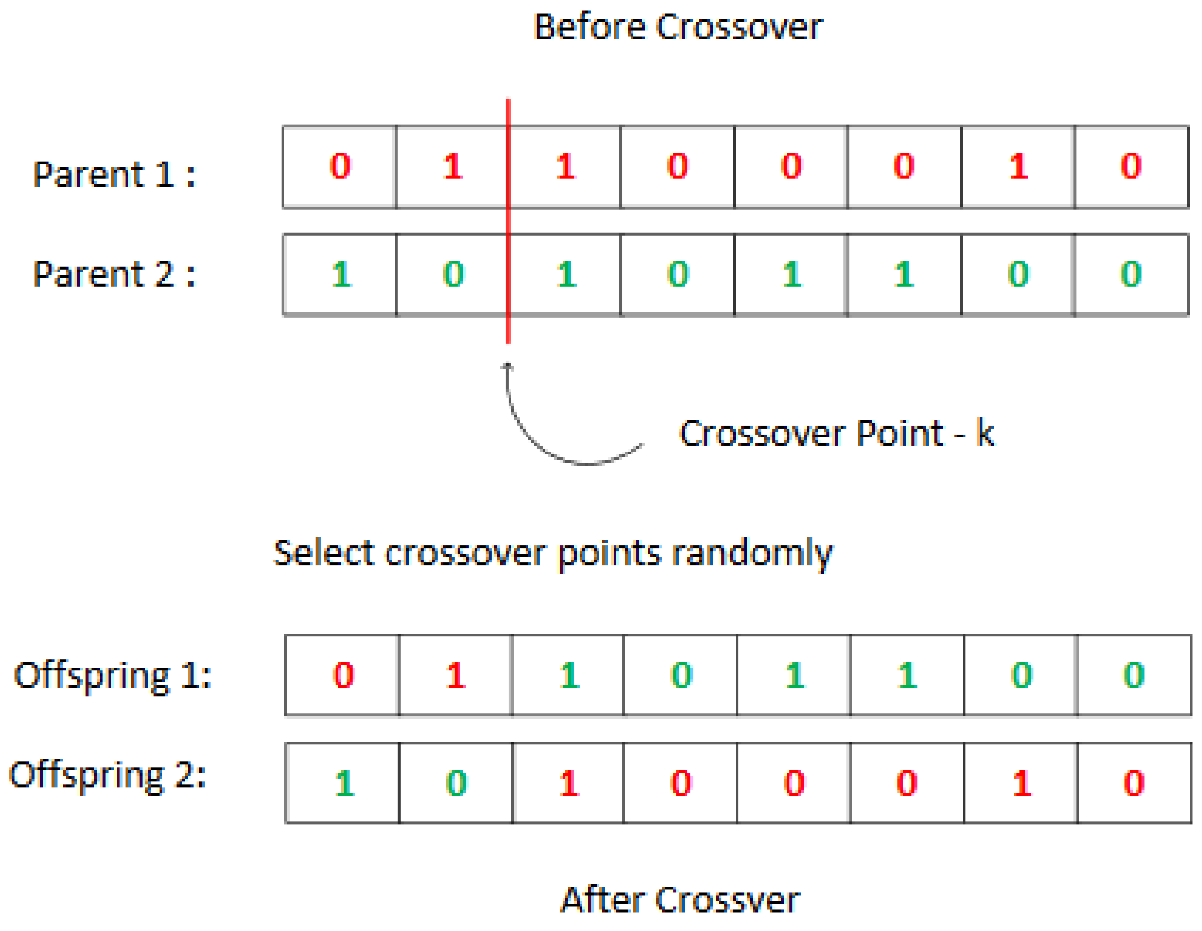
Crossover point(s) (also called k-point(s))is(are) decided using a random number generator generating integer(s) in between 1 and L, where L is the length of the chromosome.

Single-Point Crossover

- Given two parents, single-point crossover will generate a cut-point and recombines the first part of first parent with the second part of the second parent to create one offspring.
- Single-point crossover then recombines the second part of the first parent with the first part of the second parent to create a second offspring.
- Example:

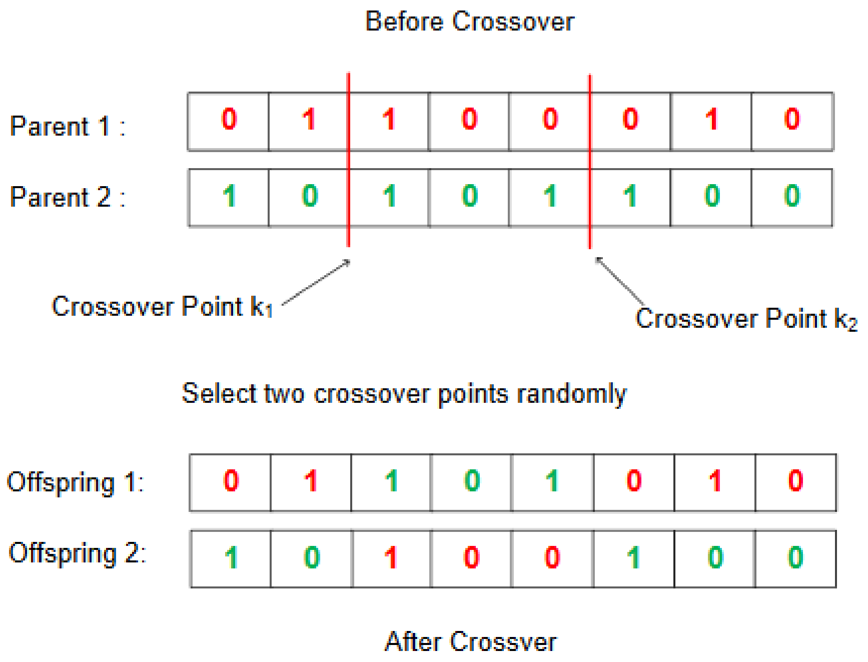
Parent 1:	**X X** | X X X X X
Parent 2:	Y Y | **Y Y Y Y Y**
Offspring 1:	**X X Y Y Y Y Y**
Offspring 2:	Y Y X X X X X

Single-Point Crossover: Example



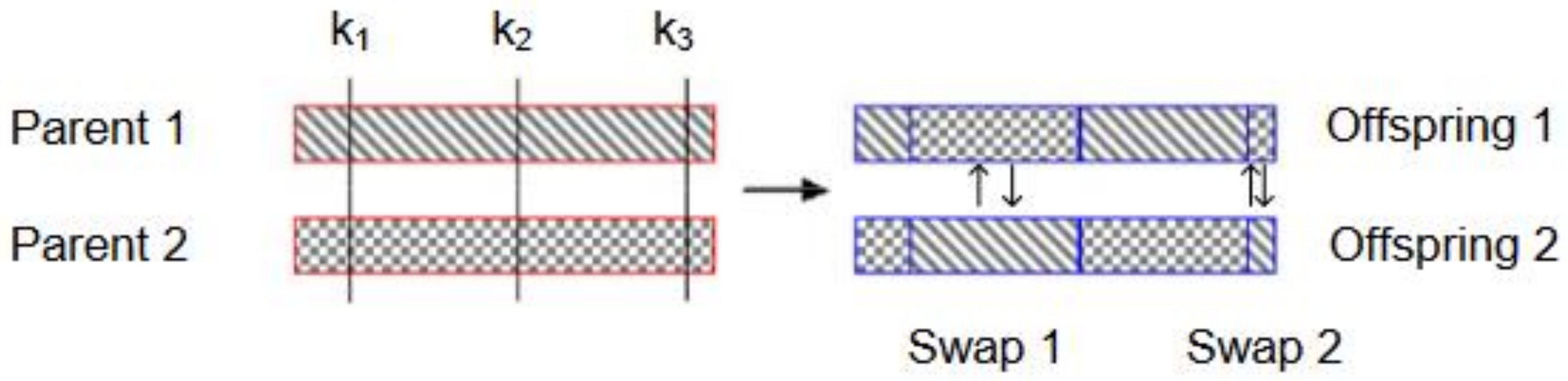
Genetic Algorithms: Two-Point Crossover

- Two-Point crossover is very similar to single-point crossover except that two cut-points are generated instead of one.
- The middle parts are swapped between the two strings.
- Alternatively, left and right parts also can be swapped.



Multi-point crossover

1. In case of multi-point crossover, a number of crossover points are selected along the length of the string, at random.
2. The bits lying between alternate pairs of sites are then swapped.



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.

Uniform Crossover: Example

Before Crossover

Parent 1 :	1	1	0	0	0	1	0	1
Parent 2 :	0	1	1	0	0	1	1	1
Mask	1	0	0	1	1	1	0	1
Offspring 1:	1	1	1	0	0	1	1	1
Offspring 2:	0	1	0	0	0	1	0	1

When there is a 1 in the mask, the gene is copied from Parent 1 else from Parent 2.

When there is a 1 in the mask, the gene is copied from Parent 2 else from Parent 1.

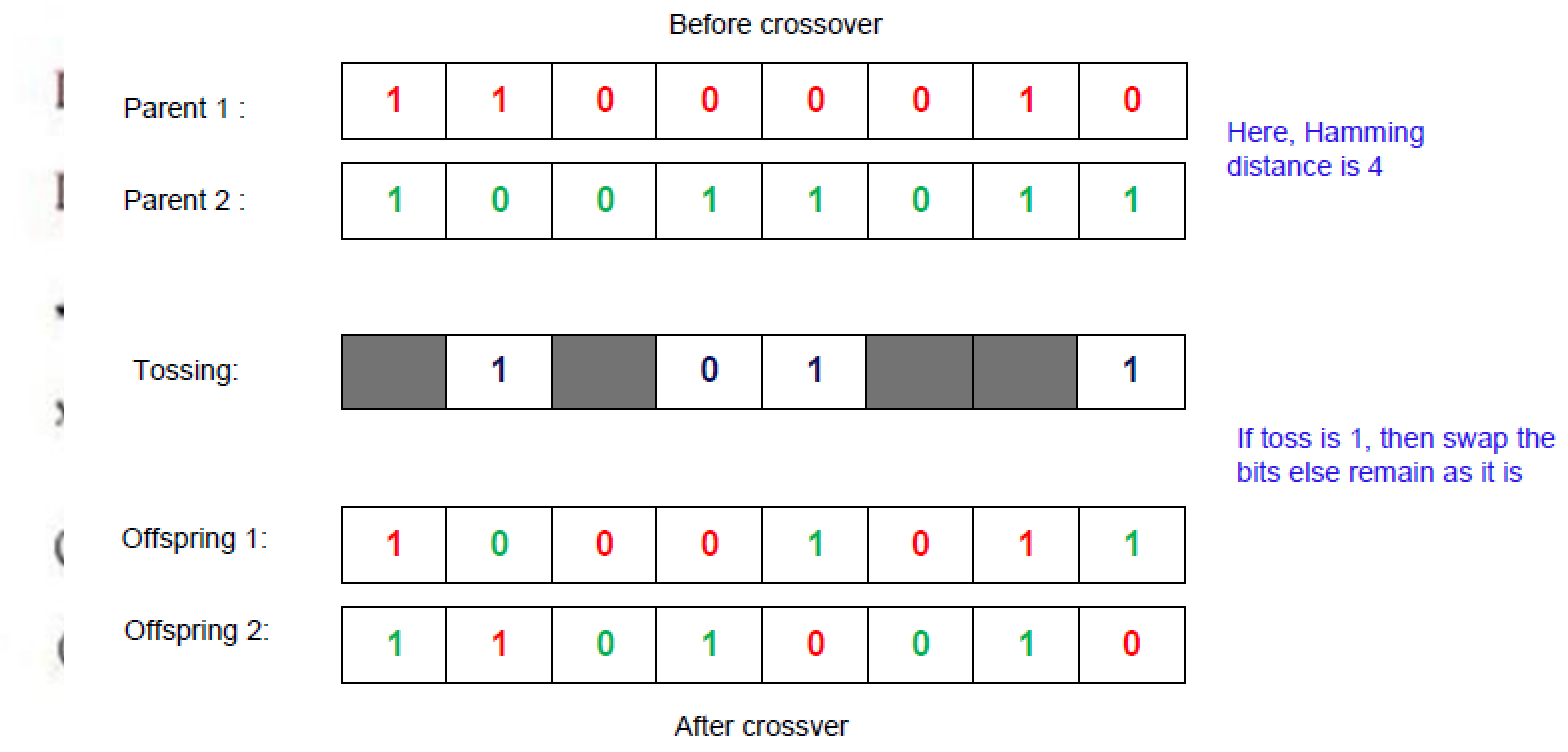
After Crossver

Half-uniform crossover (HUX)

- In the half uniform crossover scheme, exactly half of the non-matching bits are swapped.
  - Calculate the Hamming distance (the number of differing bits) between the given parents.
  - This number is then divided by two.
  - The resulting number is how many of the bits that do not match between the two parents will be swapped but probabilistically.
  - Choose the locations of these half numbers (with some strategies, say coin tossing) and swap them.

HUX Example

Hamming distance is 4 here, therefore number of swaps=4/2=2



Before crossover

Parent 1 :	1	1	0	0	0	0	1	0
Parent 2 :	1	0	0	1	1	0	1	1
Tossing:	▨	1	▨	0	1	▨	▨	1
Offspring 1:	1	0	0	0	1	0	1	1
Offspring 2:	1	1	0	1	0	0	1	0

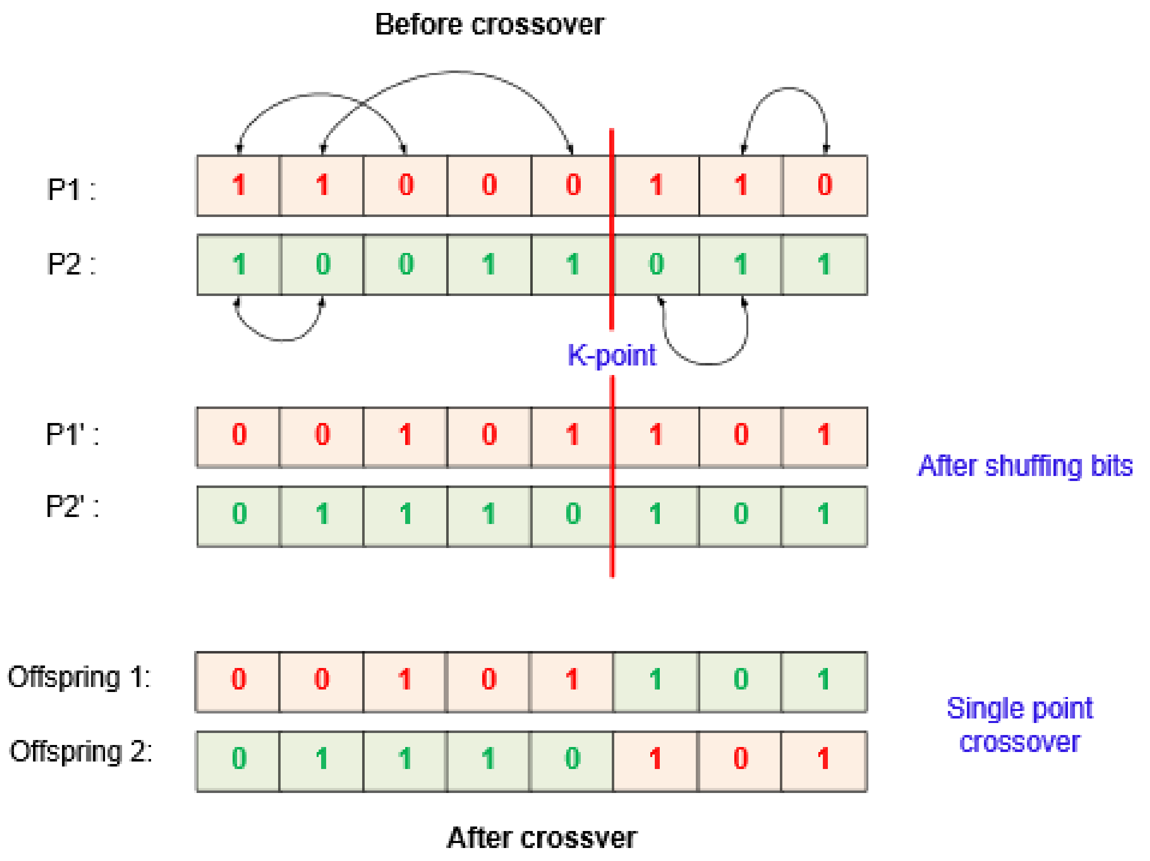
After crossver

Here, Hamming distance is 4

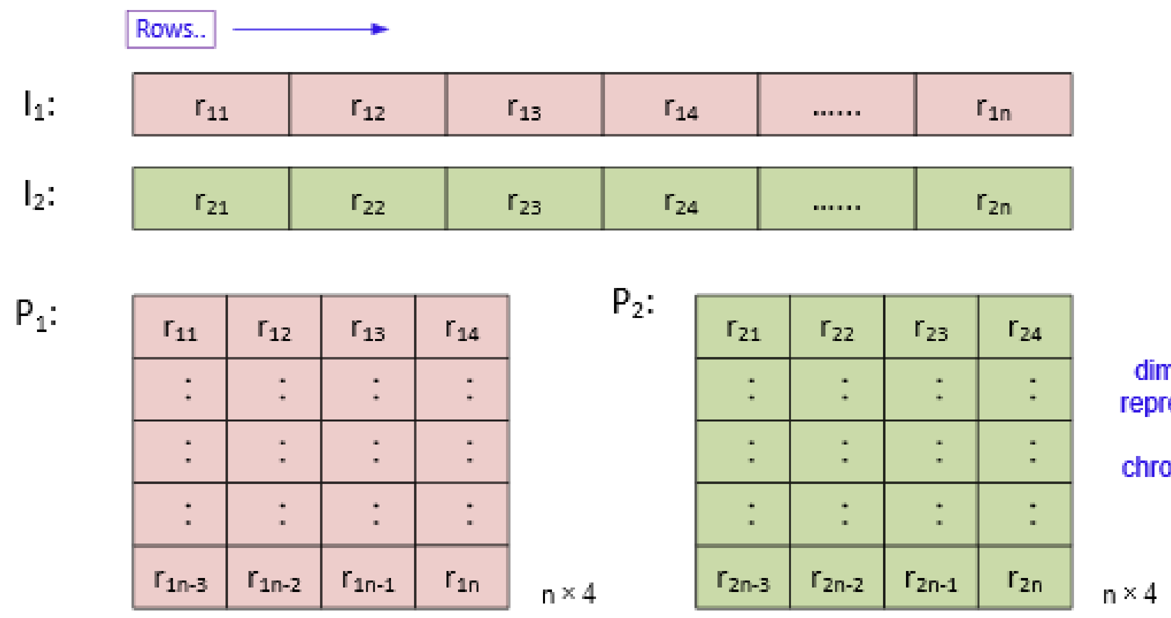
If toss is 1, then swap the bits else remain as it is

Shuffle crossover

- A single crossover point is selected. It divides a chromosome into two parts called schema.
- In both parents, genes are shuffled in each schema. Follow some strategy for shuflling bits
- Schemas are exchanged to create offspring (as in single crossover)

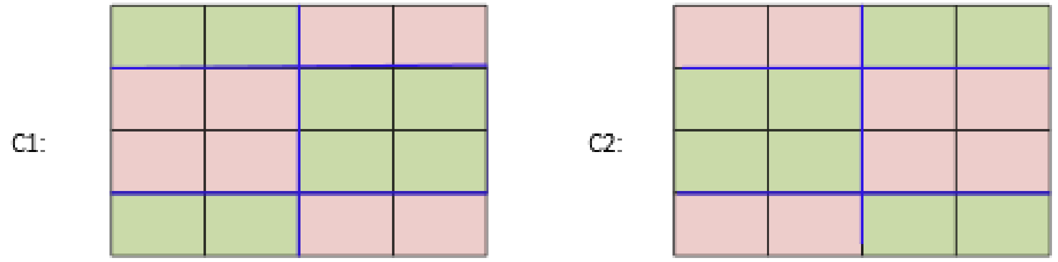


Matrix crossover



Two dimensianal representation of the chromosomes

Then matrices are divided into a number of non-overlapping zones



Two matrices are divided into a number of non-overlapping zones and shuffle between them

Mutation

After a crossover is performed, mutation takes place.

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next.

Mutation is an important part of the genetic search, helps to prevent the population from stagnating at any local optima. Mutation is intended to prevent the search falling into a local optimum of the state space.

The Mutation operators are of many type.

– one simple way is, **Flip Bit.**

– the others are **Boundary, Non-Uniform, Uniform, and Gaussian.**

The operators are selected based on the way chromosomes are encoded .

Flip Bit

The mutation operator simply inverts the value of the chosen gene.
i.e. **0** goes to **1** and **1** goes to **0**.
This mutation operator can only be used for binary genes.

Consider the two original off-springs selected for mutation.

Original offspring 1	1	1	0	1	1	1	1	0	0	0	0	1	1	1	1	0
Original offspring 2	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	0

Invert the value of the chosen gene as **0** to **1** and **1** to **0**

The Mutated Off-spring produced are :

Mutated offspring 1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0
Mutated offspring 2	1	1	0	1	1	0	1	1	0	0	1	1	0	1	0	0

Boundary

The mutation operator replaces the value of the chosen gene with either the upper or lower bound for that gene (chosen randomly).

This mutation operator can only be used for integer and float genes.

Uniform

The mutation operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene.

This mutation operator can only be used for integer and float genes.

Genetic Algorithms: Selection Who Survives

- Basically, there are two types of GAs commonly used.

  These GAs are characterized by the type of replacement strategies they use.
- A Generational GA uses a (μ,μ) replacement strategy where the offspring replace the parents.
- A Steady-State GA usually will select two parents, create 1-2 offspring which will replace the 1-2 worst individuals in the current population ***even if the offspring are worse than the individuals they replace***.

Genetic Algorithms: Example

- The SGA for our example will use:
  - A population size of 6,
  - A crossover usage rate of 1.0, and
  - A mutation rate of 1/7.
- Let's try to solve the following problem

$$f(x) = x^2, \text{ where } -2.0 \leq x \leq 2.0,$$

  - Let $l = 7$, therefore our mapping function will be
    - ***d(2,-2,7,c) = 4*decode(c)/127 - 2***

If we were to use binary-coded representations we would first need to develop a mapping function form our genotype representation (binary string) to our phenotype representation (our CS). This can be done using the following mapping function:

$$d(ub,lb,l,chrom) = (ub-lb)\ decode(chrom)/2^l-1 + lb$$

Genetic Algorithms: An Example Run (by hand)

- Randomly Generate an Initial Population

	Genotype	Phenotype	Fitness
Individual 1:	1001010	0.331	Fit: ?
Individual 2:	0100101	- 0.835	Fit: ?
Individual 3:	1101010	1.339	Fit: ?
Individual 4:	0110110	- 0.300	Fit: ?
Individual 5:	1001111	0.488	Fit: ?
Individual 6:	0001101	- 1.591	Fit: ?

Genetic Algorithms: An Example Run (by hand)

- Evaluate Population at t=0

	Genotype	Phenotype	Fitness
Individual 1:	1001010	0.331	Fit: 0.109
Individual 2:	0100101	- 0.835	Fit: 0.697
Individual 3:	1101010	1.339	Fit: 1.790
Individual 4:	0110110	- 0.300	Fit: 0.090
Individual 5:	1001111	0.488	Fit: 0.238
Individual 6:	0001101	- 1.591	Fit: 2.531

Genetic Algorithms: An Example Run (by hand)

- Select Six Parents Using the Roulette Wheel

	Genotype	Phenotype	Fitness
Individual 6:	0001101	- 1.591	Fit: 2.531
Individual 3:	1101010	1.339	Fit: 1.793
Individual 5:	1001111	0.488	Fit: 0.238
Individual 6:	0001101	- 1.591	Fit: 2.531
Individual 2:	0100101	- 0.835	Fit: 0.697
Individual 1:	1001010	0.331	Fit: 0.109

Genetic Algorithms: An Example Run (by hand)

- Create Offspring 1 & 2 Using Single-Point Crossover

	Genotype	Phenotype	Fitness
Individual 6:	00|01101	- 1.591	Fit: 2.531
Individual 3:	11|01010	1.339	Fit: 1.793
Child 1 :	0001010	- 1.685	Fit: ?
Child 2 :	1101101	1.433	Fit: ?

Genetic Algorithms: An Example Run (by hand)

- Create Offspring 3 & 4

	Genotype	Phenotype	Fitness
Individual 5:	1001|111	0.488	Fit: 0.238
Individual 6:	0001|101	- 1.591	Fit: 2.531
Child 3 :	1011100	0.898	Fit: ?
Child 4 :	0001011	- 1.654	Fit: ?

Genetic Algorithms: An Example Run (by hand)

- Create Offspring 5 & 6

	Genotype	Phenotype	Fitness
Individual 2:	010|0101	- 0.835	Fit: 0.697
Individual 1:	100|1010	0.331	Fit: 0.109
Child 5 :	1101010	1.339	Fit: ?
Child 6 :	1010101	0.677	Fit: ?

Genetic Algorithms: An Example Run (by hand)

- Evaluate the Offspring

	Genotype	Phenotype	Fitness
Child 1 :	0001010	- 1.685	Fit: 2.839
Child 2 :	1101101	1.433	Fit: 2.054
Child 3 :	1011100	0.898	Fit: 0.806
Child 4 :	0001011	- 1.654	Fit: 2.736
Child 5 :	1101010	1.339	Fit: 1.793
Child 6 :	1010101	0.677	Fit: 0.458

Genetic Algorithms: An Example Run (by hand)

Population at t=0

	Genotype	Phenotype	Fitness
Person 1:	1001010	0.331	Fit: 0.109
Person 2:	0100101	- 0.835	Fit: 0.697
Person 3:	1101010	1.339	Fit: 1.793
Person 4:	0110110	- 0.300	Fit: 0.090
Person 5:	1001111	0.488	Fit: 0.238
Person 6:	0001101	- 1.591	Fit: 2.531

Is Replaced by:

	Genotype	Phenotype	Fitness
Child 1 :	0001010	- 1.685	Fit: 2.839
Child 2 :	1101101	1.433	Fit: 2.054
Child 3 :	1011100	0.898	Fit: 0.806
Child 4 :	000**1**011	- 1.654	Fit: 2.736
Child 5 :	1101010	1.339	Fit: 1.793
Child 6 :	1010101	0.677	Fit: 0.458

Genetic Algorithms:
An Example Run (by hand)

- Population at t=1

	Genotype	Phenotype	Fitness
Person 1:	0001010	- 1.685	Fit: 2.839
Person 2:	1101101	1.433	Fit: 2.054
Person 3:	1011100	0.898	Fit: 0.806
Person 4:	0001011	- 1.654	Fit: 2.736
Person 5:	1101010	1.339	Fit: 1.793
Person 6:	1010101	0.677	Fit: 0.458

Common Parameters of GA

- **Mutation probability pm:** A large mutation probability will disrupt a given individual and the search is more likely random. Generally, small values are recommended for the mutation probability ($pm \in [0.001, 0.01]$). Usually, the mutation probability is initialized to 1/k where k is the number of decision variables. Hence, on average, only one variable is mutated.
- **Crossover probability pc:** The crossover probability is generally set from medium to large values (e.g., [0.3, 0.9]).
- **Population size:** The larger is the size of the population, the better is the convergence toward "good" solutions. However, the time complexity of EAs grows linearly with the size of the population.