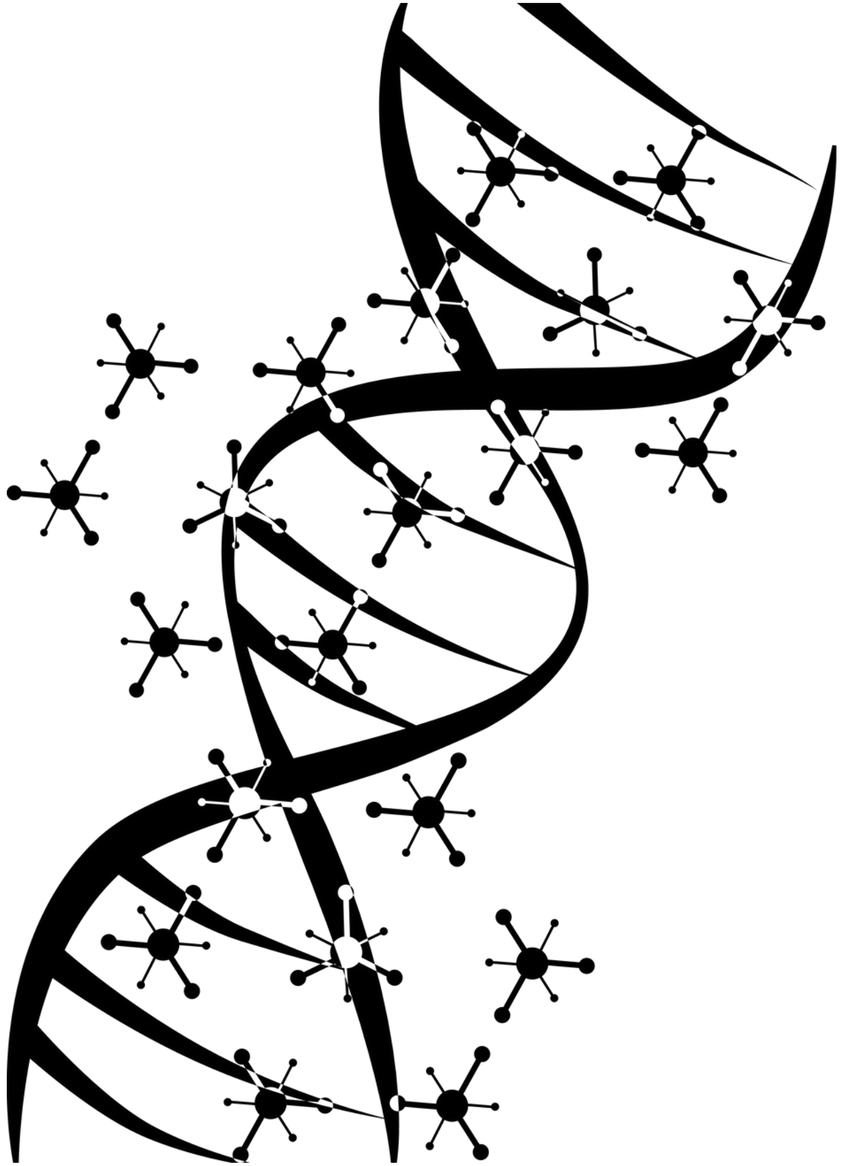


EVOLUTIONARY ALGORITHMS

Week 3, 06.02.2024

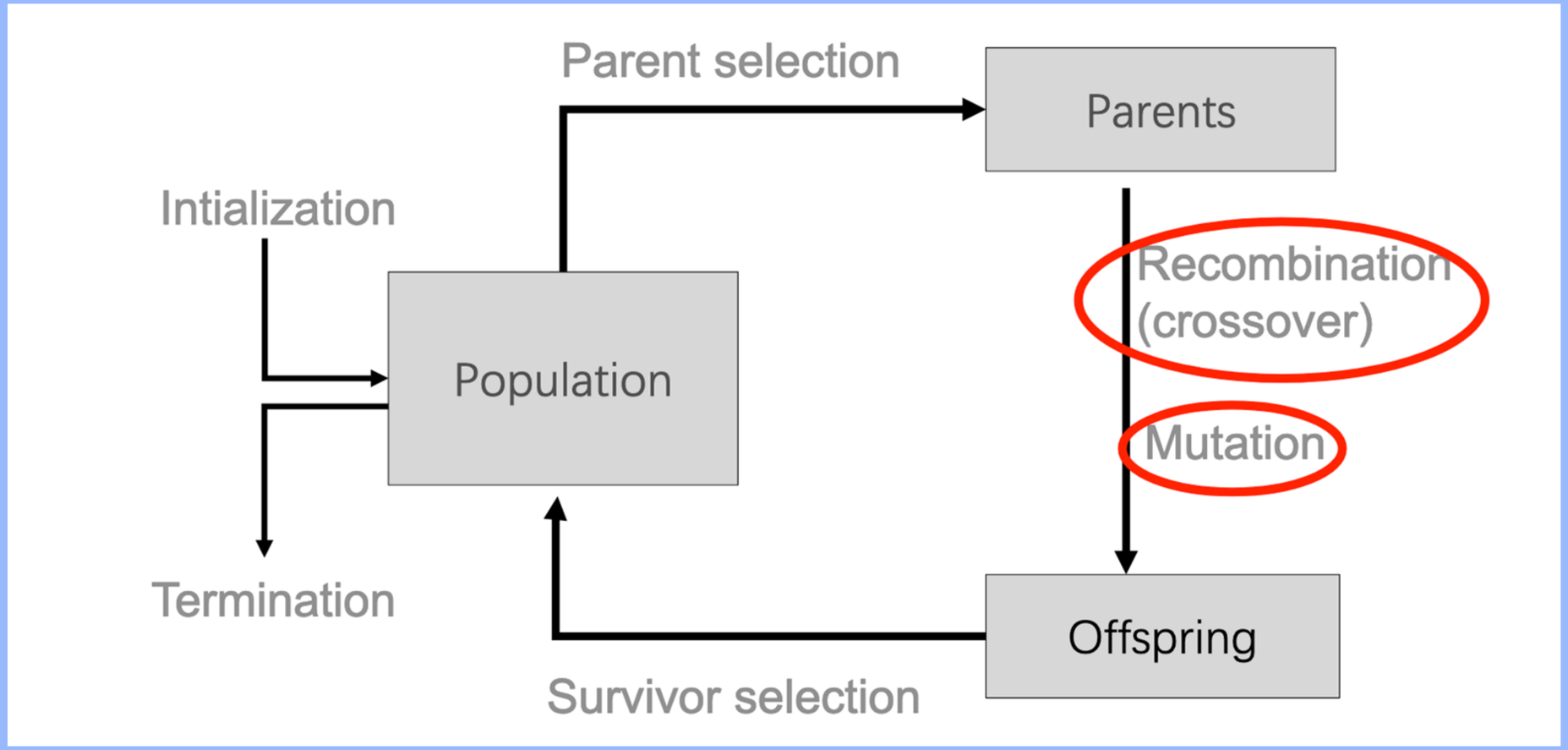
OUTLINE FOR THE SESSION

EA brief theory (recombination

mutation)

+ parallel exercises week 3

GENERAL SCHEME OF EA



BRIEF EXPLANATION OF OVERALL IDEA

Evolutionary Algorithms (EAs) are powerful optimization techniques inspired by the principles of natural selection.

Unlike traditional methods like Hill Climbing, EAs explore a population of potential solutions, mimicking the process of biological evolution.

Similar to hillclimbing neighbours approach

Hill Climbing can get stuck in local optima, limiting its ability to find the global optimum. EAs address this limitation by introducing a population-based approach.

EA IN PSEUDOCODE

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

ESSENTIAL PARTS OF EA REPRESENTATION

Population:

Representation: A collection of individuals or candidate solutions.
Structure: The population is often represented as an array or list, where each element corresponds to an individual genome.

Fitness Function:

Representation: A function that evaluates the quality of a solution.
Structure: The fitness function takes a genome as input and assigns a numerical value representing the solution's fitness or performance. It guides the selection process in the evolutionary cycle.

Genome(chromosomes):

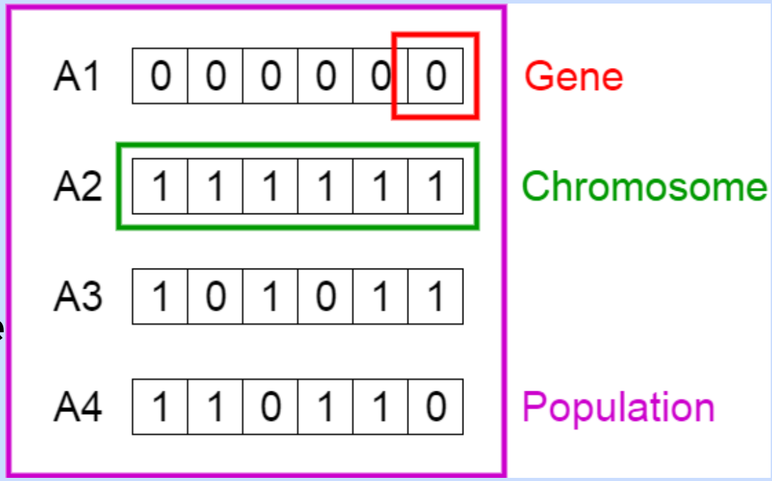
The encoded solution to the optimization problem.

The genome can be represented using various encoding schemes such as binary strings, real-valued vectors, integers, permutations, trees, or other structures depending on the nature of the problem.

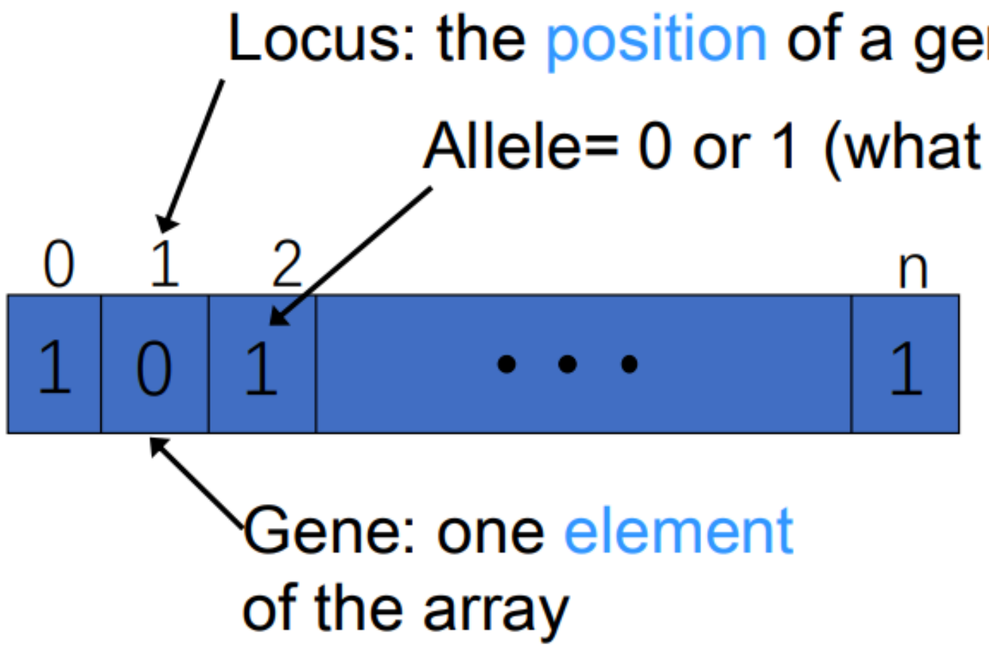
Initiallisation and **termination** criteria are essential parts part of the algorithm indicating start and finish

Genotype structure

Locus: the position of a gene
Allele = 0 or 1 (What values a gene can have)
Gene: one element of the array
Genotype: a set of gene
valuesPhenotype: What could be built/developed based on the genotype



Representation: EA terms



Genotype: a set of gene values

⇩

Phenotype: what could be built/developed based on the genotype

GENOTYPE VS PHENOTYPE

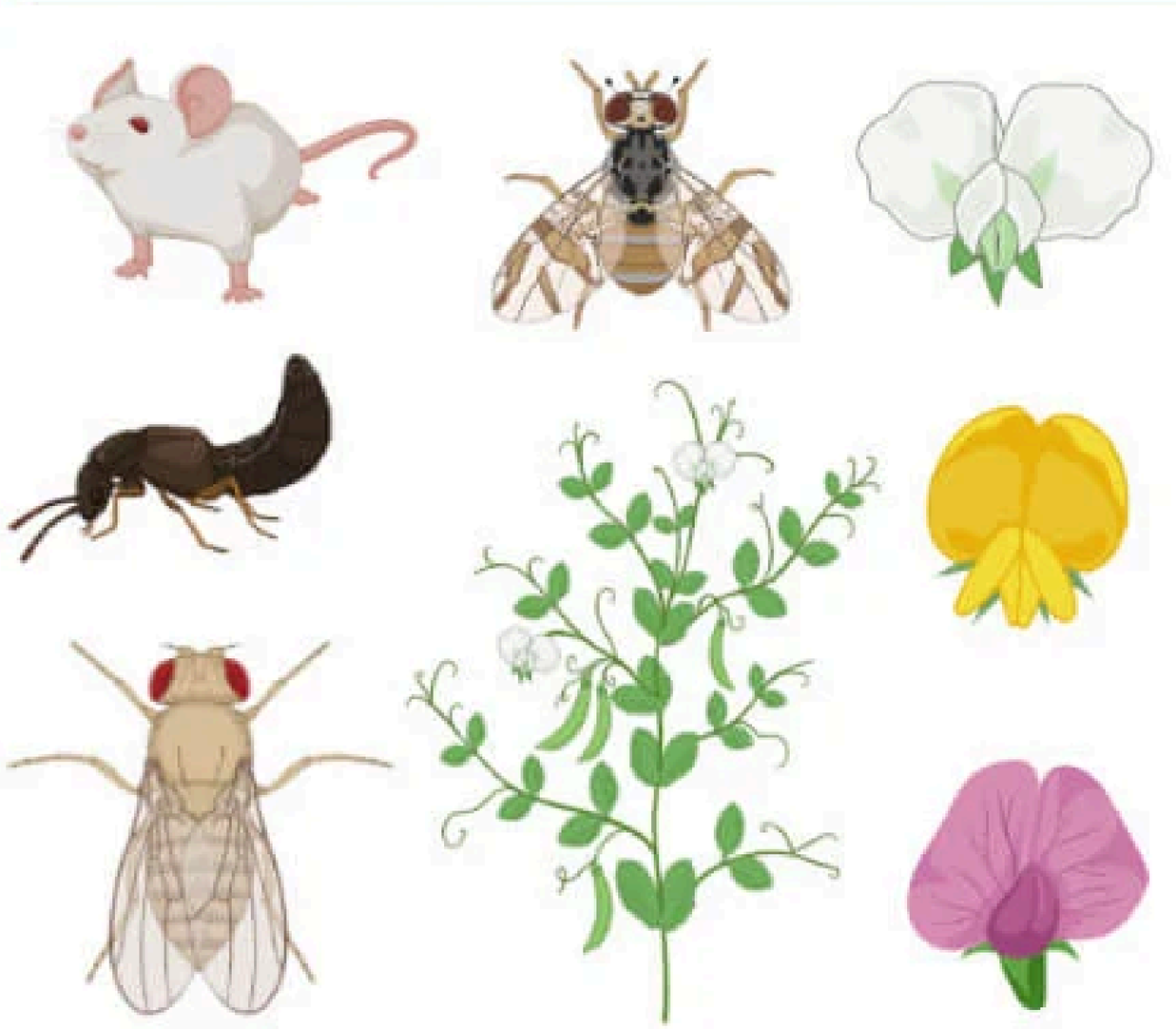
a set of gene values

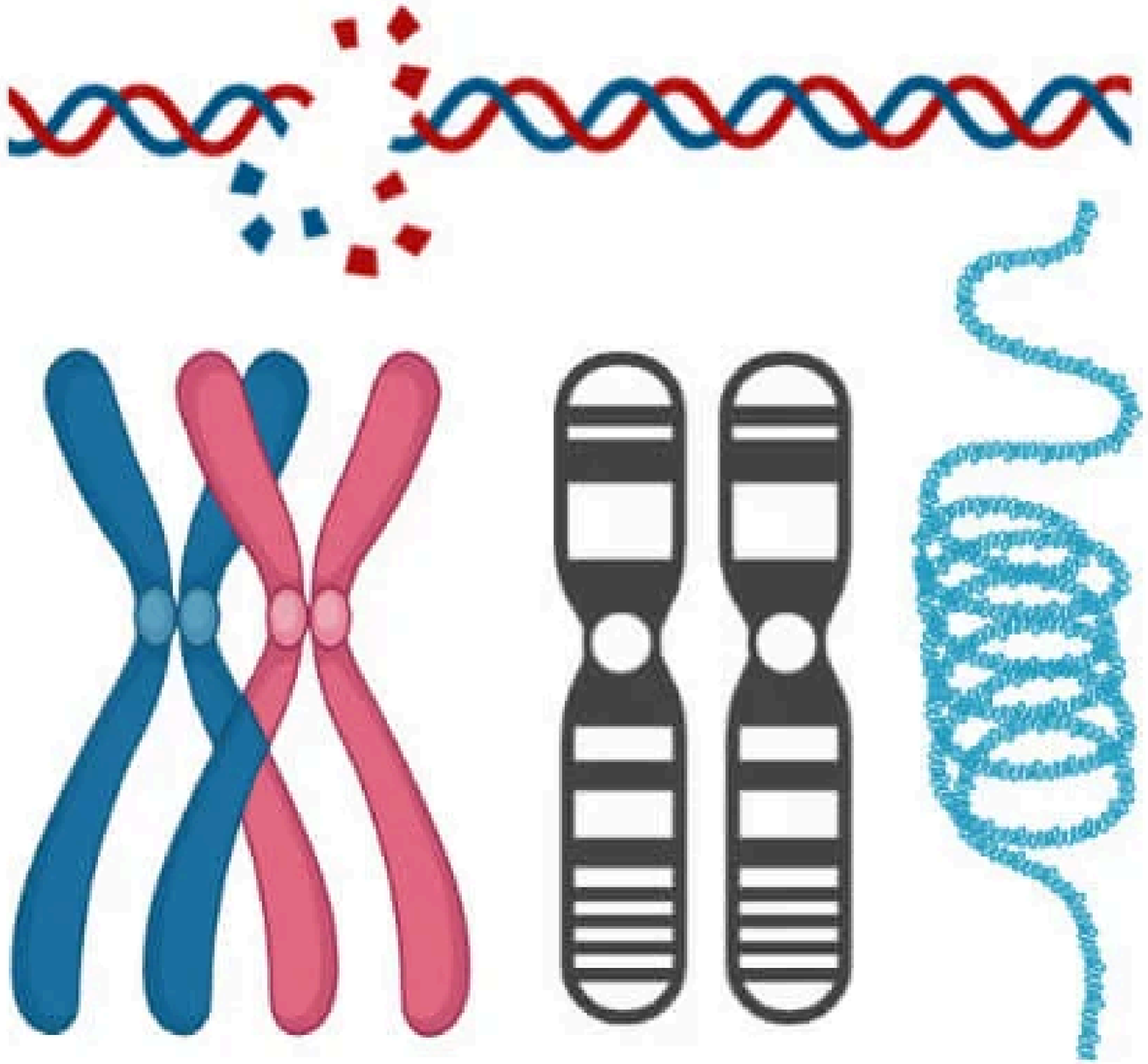
The genotype serves as the raw genetic material that undergoes genetic operations (crossover and mutation) during the evolutionary process. Example: For a binary string representation, the genotype might be a sequence of 0s and 1s: [1, 0, 1, 1, 0].

what could be built/developed based on the genotype

The phenotype is **what is evaluated by the fitness function** to determine the quality of an individual's solution to the problem. Example: If the genotype represents a binary string encoding, the decoded phenotype might be a corresponding integer or a real-valued number: 22.

Differences between Phenotype and Genotype





MOST COMMON REPRESENTATIONS OF GENOMES

Binary Representation:

strings of binary digits (0s and 1s).

Real-Valued Representation:

Genomes are represented as **vectors of real numbers**.
Suitable for optimization problems with continuous variables, like mathematical functions or parameter optimization.

Integer Representation:

Genomes are represented as **vectors of integers.**
Appropriate for problems where solutions must be whole numbers, such as certain combinatorial optimization tasks.

Permutation Representation:

Genomes are represented as permutations of a set of values.
Suitable for problems involving the ordering of elements, like the traveling salesman problem.

Tree-Based Representation:

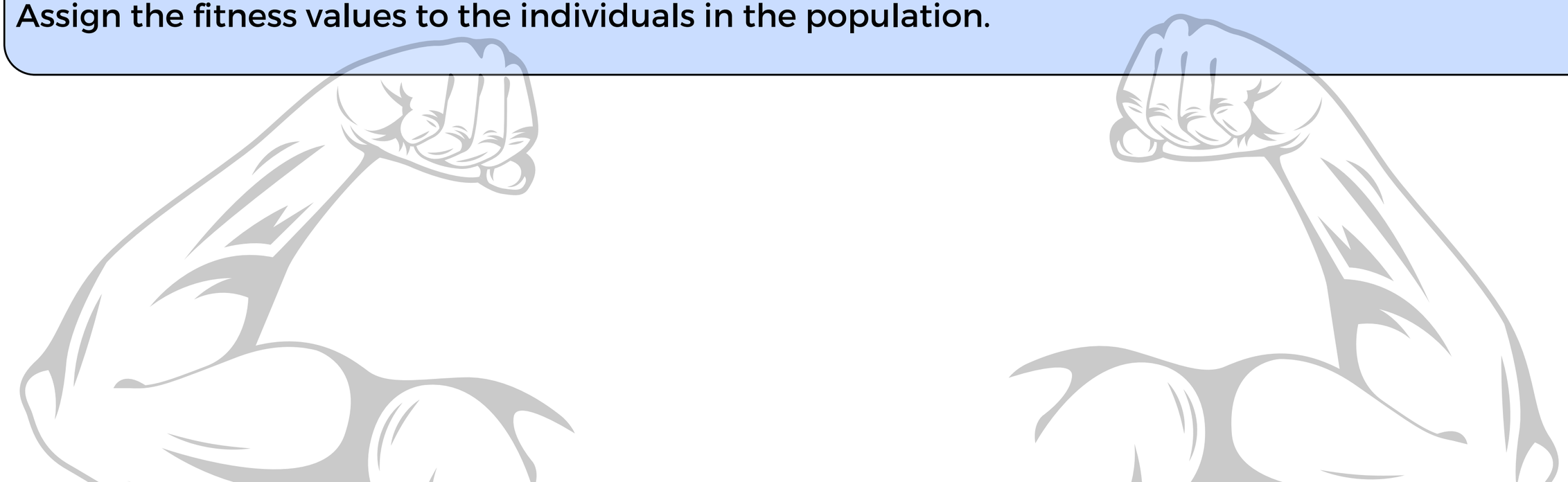
Genomes are represented as hierarchical tree structures.
Common in Genetic Programming (GP) and Genetic Algorithms (GA) for problems where the solution structure is essential.

SELECTION AND FITNESS FUNCTIONS

Fitness Function:

The fitness function evaluates the quality or performance of an individual, It assigns a numerical value indicating how well an individual satisfies the problem's objectives(no universal formula, depends on problem).

For each individual in the population, apply the fitness function to obtain a numerical fitness value.
Assign the fitness values to the individuals in the population.



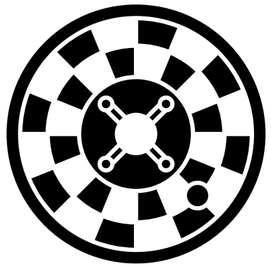
EXAMPLE OF FITNESS FUNCTION FOR TSP

TSP (Minimizing Total Distance in a Traveling Salesman Problem):

```
def tsp_fitness(individual, distance_matrix):
    """Fitness function for minimizing the total distance in the Traveling Salesman Problem."""
    total_distance = 0
    num_cities = len(individual)

    for i in range(num_cities - 1):
        city1, city2 = individual[i], individual[i + 1]
        total_distance += distance_matrix[city1][city2]

    # Return to the starting city
    total_distance += distance_matrix[individual[-1]][individual[0]]

    return -total_distance  # Negative because EA maximizes, and we want to minimize distance
```

SELECTION

the process of choosing individuals from the current population to act as parents for the creation of the next generation.

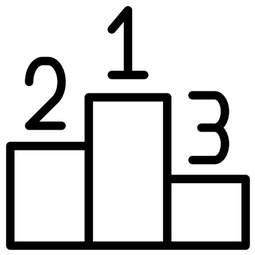


Roulette Wheel Selection:

Individuals are selected with a probability proportional to their fitness.
Fitness values are normalised to create a probability distribution.

Tournament Selection:

Randomly select a subset of individuals (a tournament) and choose the fittest individual from the subset.
Repeated until a sufficient number of parents are chosen.



Rank-Based Selection:

Individuals are ranked based on their fitness.
Selection is then based on the rank rather than absolute fitness.

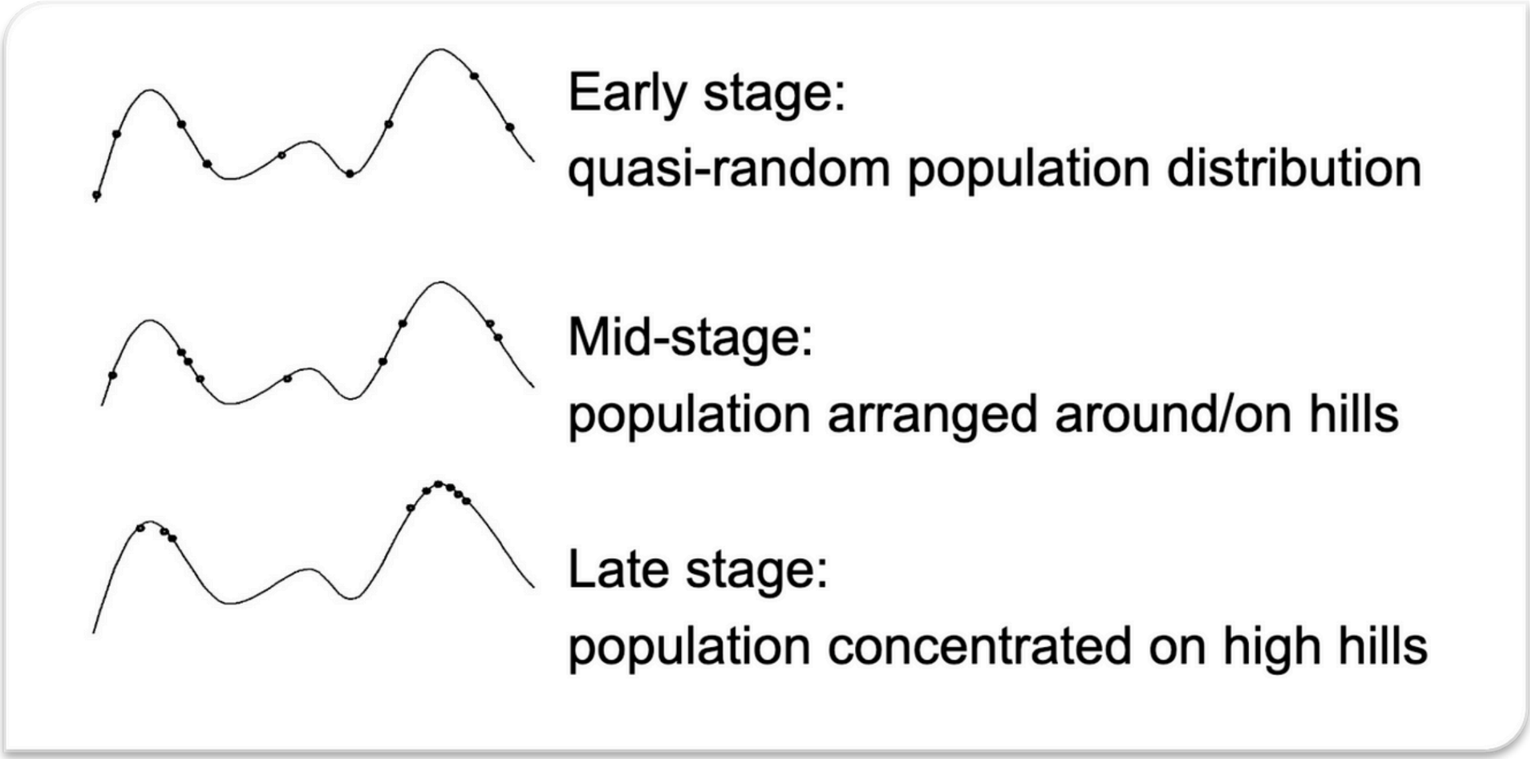
EVOLVING POPULATIONS

Diversity:

The population embodies a diverse array of potential solutions, fostering exploration across the solution space.

Variation - Unleashing Diversity:

Genetic operations inject diversity into the population, promoting the exploration of different solution pathways.



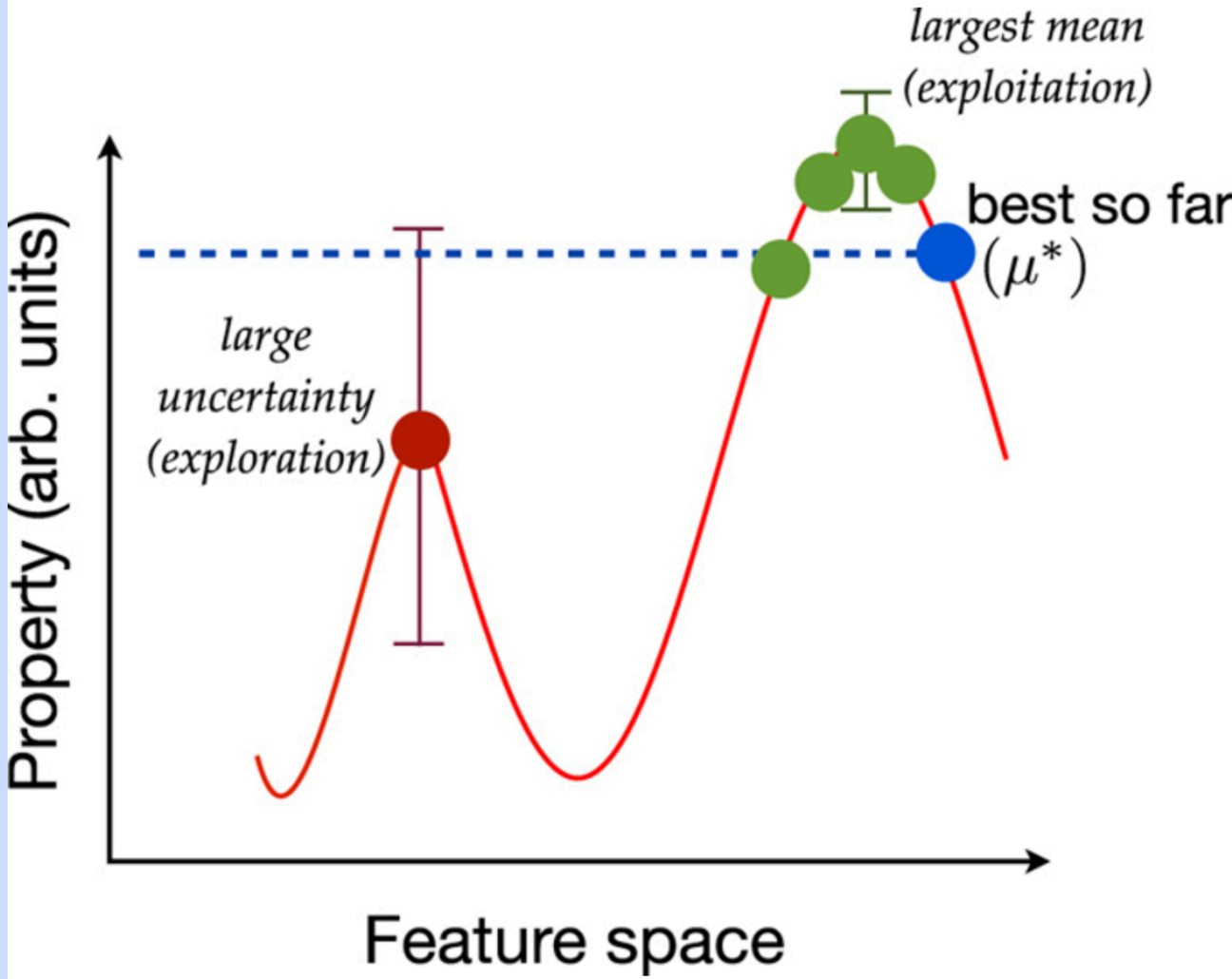
EXPLORATION VS EXPLOITATION

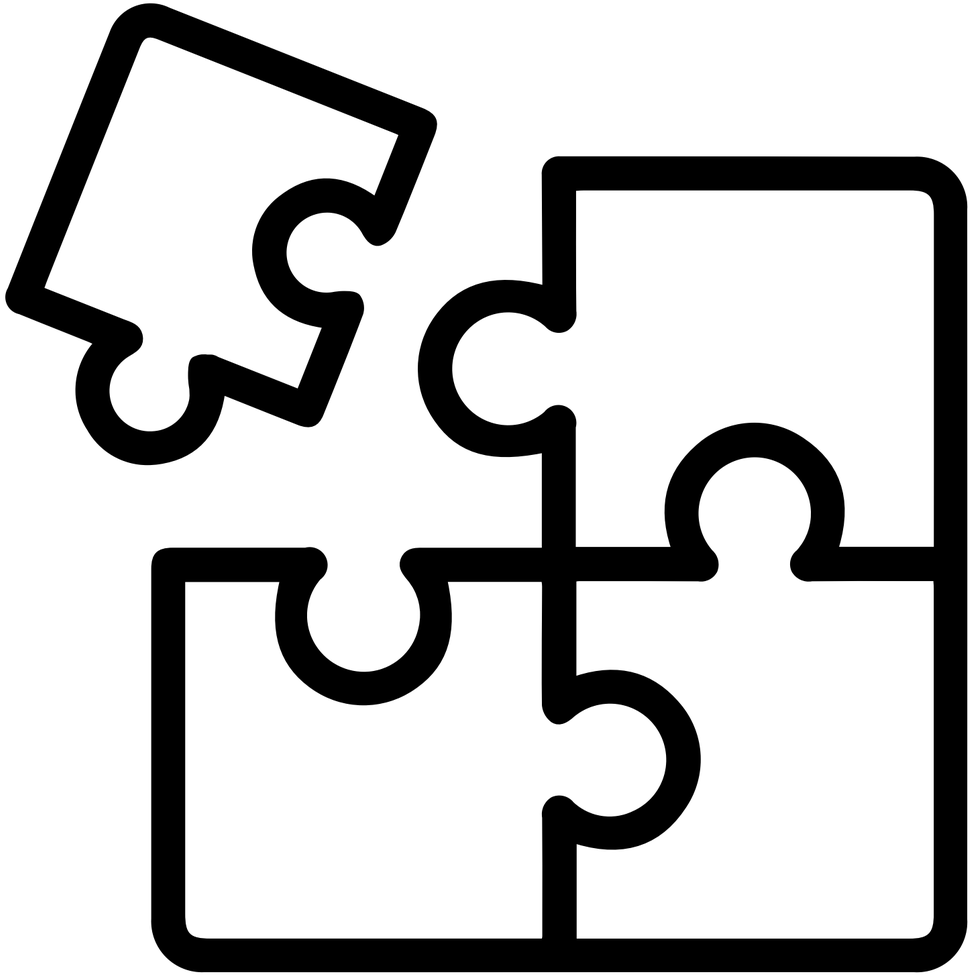
Exploration

Increasing population
diversity by genetic operators
mutation
recombination
Push towards novelty

exploitation

Decreasing population diversity
by selection
of parents
of survivors
Push towards quality





VARIATION OPERATIONS

VARIATION OPERATIONS

Main rule

Arity - number of inputs

Arity = 1 is mutation

Arity > 1 is recombination

Crossover (Recombination):

Single-Point Crossover
Multi-Point Crossover
Uniform Crossover
Edge recombination
Partially Mapped Crossover (PMX)
Order Crossover (OX)

Mutation:

Bit Flip Mutation
Random Resetting
Gaussian Mutation
Swap Mutation
Inversion

MUTATION

introduces small random changes to the genetic material of individuals in the population

Mutation helps maintain diversity in the population and can lead to the exploration of new regions in the solution space

MUTATION FOR BINARY REPRESENTATIONS

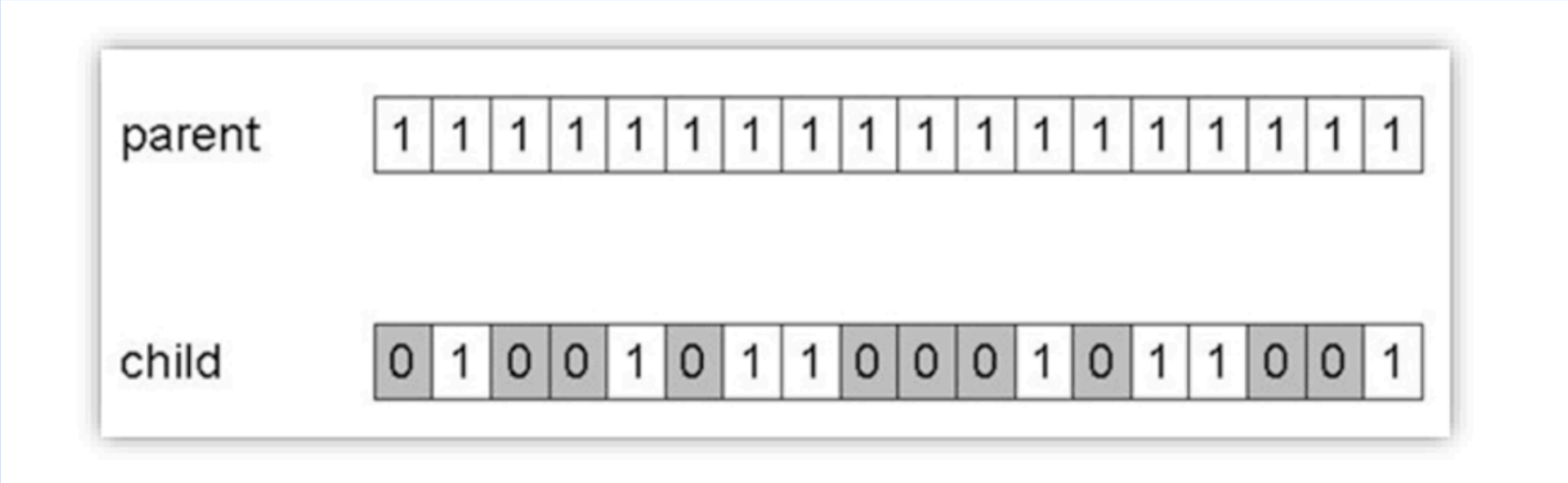
Bit-flip with probability p for each bit

Select Random Bits:

Randomly select a set of bits in the binary representation.

Flip the Selected Bits:

Flip (invert) the values of the selected bits.



INTEGER MUTATIONS

Creep Mutation:

small incremental changes to the values of randomly selected genes (integer variables) in the chromosome.

Increment (or decrement) the values of the selected genes by a small amount.

Example:

Original Chromosome: [4, 7, 12, 9, 5]
Randomly Selected: * *

Creep Mutated Chromosome: [4, 6, 12, 10, 5]

Random Resetting Mutation:

introduces more drastic changes compared to Creep Mutation.
Replaces the values of the selected genes with new random values.

Example:

Original Chromosome: [4, 7, 12, 9, 5]
Randomly Selected: * *

Random Resetting Chromosome: [4, 2, 12, 7, 5]

UNIFORM MUTATION FOR REAL VALUED AND FLOAT

draw random value from interval [min, max]

independently changing the value of each gene with a certain probability. It introduces uniform random variations to each gene.

Exaample:

Original Chromosome: [2.5, -1.8, 0.7, 3.2, -4.5]
Mutation Probability: 0.3
Mutation Range: [-1.0, 1.0]

Mutated Chromosome: [2.5, -0.5, 1.2, 3.2, -3.7]

NON-UNIFORM MUTATION FOR REAL VALUED AND FLOAT

adjusts the magnitude of mutation based on the generation number or other parameters. It allows the mutation to decrease over time, potentially converging to a more exploitative strategy.

Example:

Original Chromosome: [2.5, -1.8, 0.7, 3.2, -4.5]
Mutation Probability: 0.3
Initial Mutation Magnitude: 1.0
Generation Number: 5

Mutated Chromosome: [2.5, -1.6, 0.8, 3.0, -4.3]

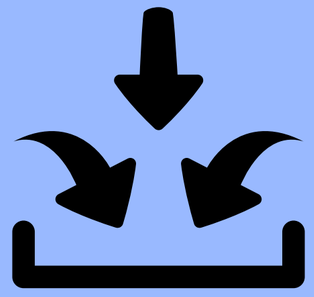
Add value drawn from Gaussian distribution parent value
$Xi\_child = Xi\_parent + N(0, \sigma)$

MUTATIONS FOR PERMUTATIONS

Swap Mutation:
Randomly selects two positions in the permutation and swaps the elements at those positions.

Example:

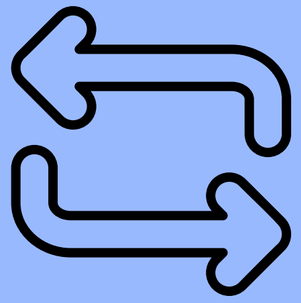
Original: [1, 2, 3, 4, 5]
After Swap Mutation: [1, 5, 3, 4, 2]



Insert Mutation:
Randomly selects a position in the permutation and moves the element at that position to another randomly chosen position.

Example:

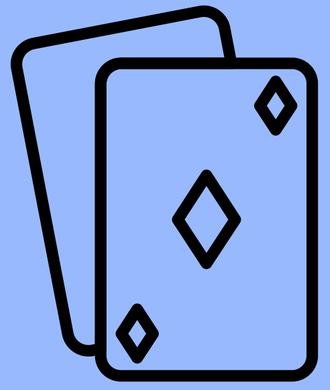
Original: [1, 2, 3, 4, 5]
After Insert Mutation: [1, 4, 2, 3, 5]

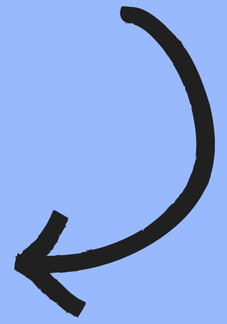


Scramble Mutation:
Description: Randomly selects a segment of the permutation and shuffles the elements within that segment.

Example

Original: [1, 2, 3, 4, 5]
After Scramble Mutation: [1, 3, 2, 4, 5]





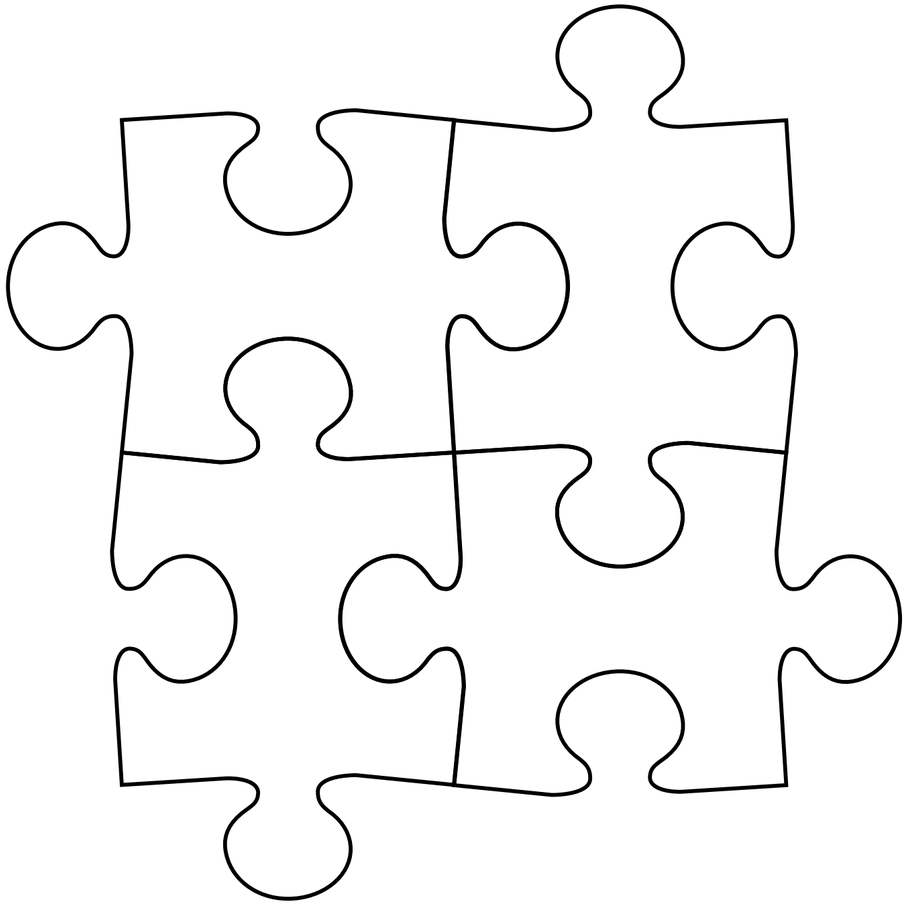
Inversion Mutation:
Inverts the order of a segment of the permutation.

Example:

Original: [1, 2, 3, 4, 5]
After Inversion Mutation: [1, 4, 3, 2, 5]

RECOMBINATION (CROSSOVER)

It simulates the process of genetic recombination or mating in biological evolution by combining genetic material from two parent individuals to produce one or more offspring.



CROSSOVERS FOR BINARY

Single-Point Crossover:
a random crossover point is chosen, and the genetic material beyond that point is swapped between the two parents.

Parent 1: 01011011
Parent 2: 11001010
Crossover Point: ↑

Offspring 1: 0101|1010
Offspring 2: 1100|1011

Uniform Crossover:
treats each bit independently, and for each bit, there's a probability of it being inherited from Parent 1 or Parent 2.

Parent 1: 01011011
Parent 2: 11001010

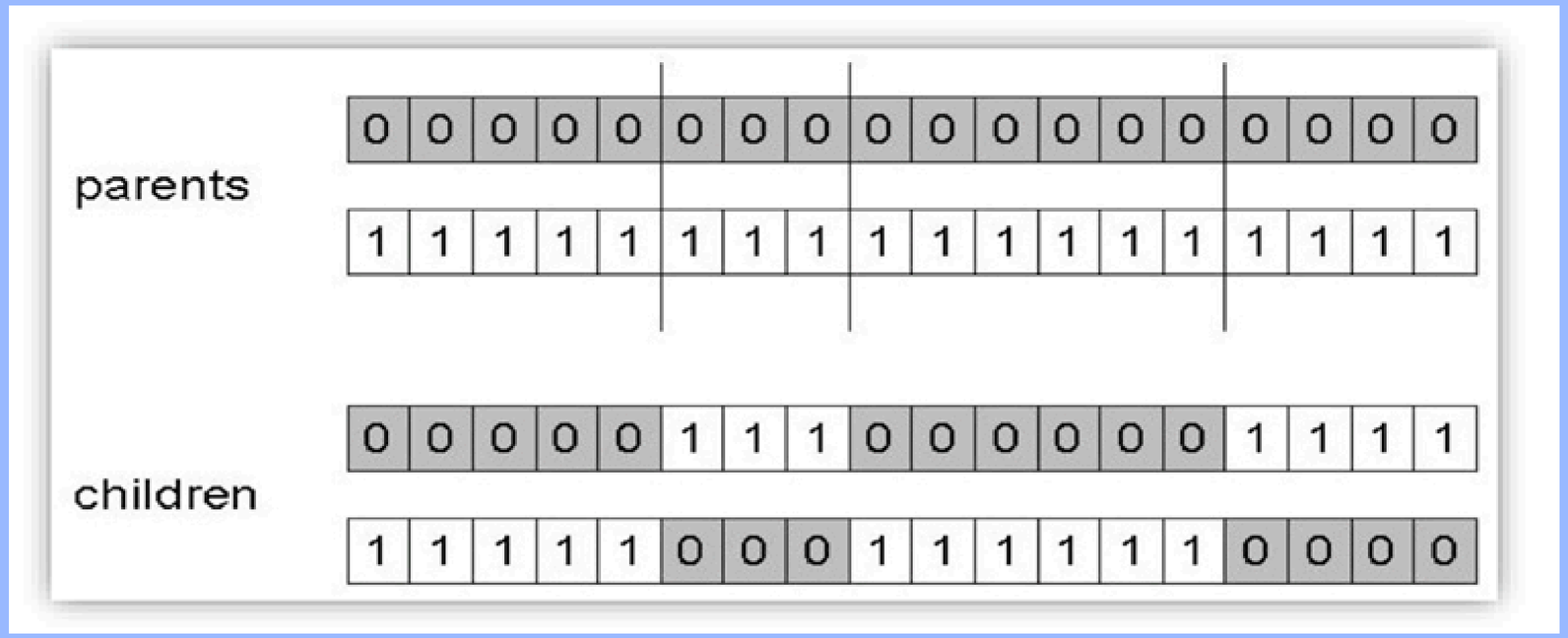
Mask: 11011001

Offspring 1: 1101|1001
Offspring 2: 0100|0100
In this example, the mask indicates which bits are inherited from Parent 1 (bit=1) or Parent 2 (bit=0).

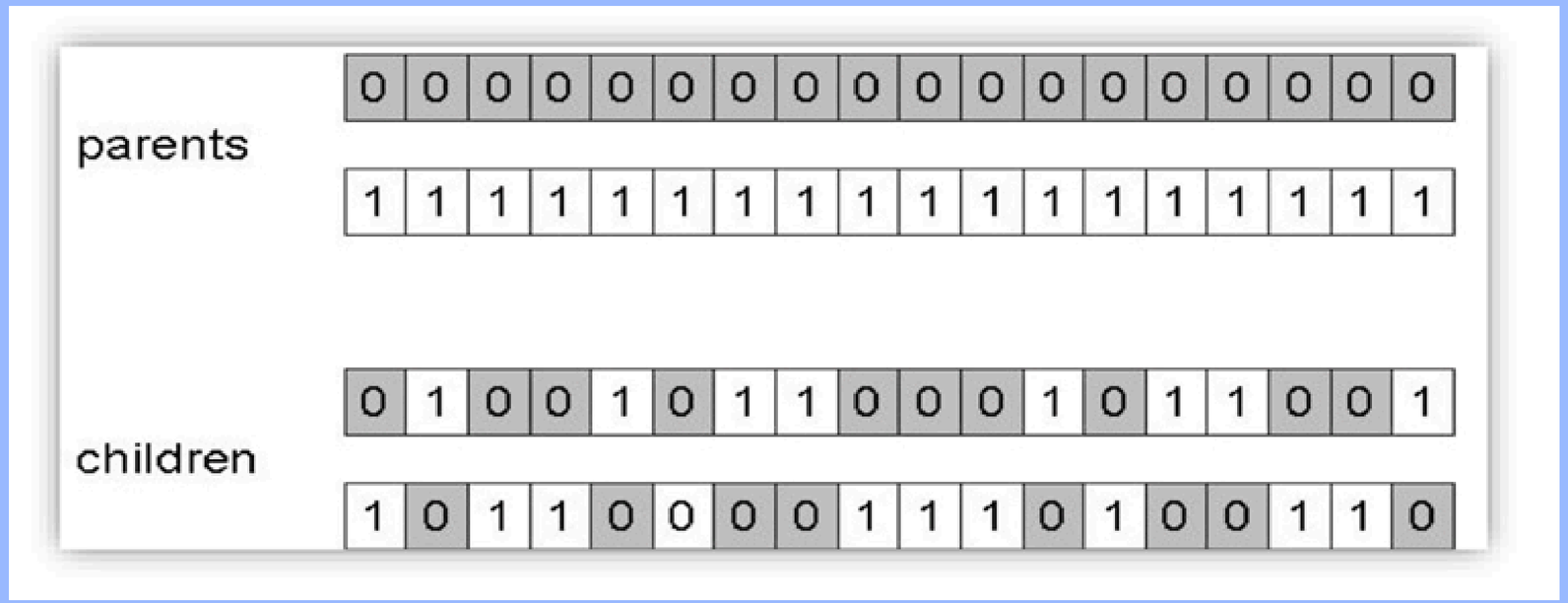
Multi-Point Crossover(n-point crossover):
Multi-Point Crossover involves selecting multiple crossover points, and the genetic material between these points is swapped between the parents.

Parent 1: 01011011
Parent 2: 11001010
Crossover Points: ↑ ↑

Offspring 1: 0101|0010
Offspring 2: 1100|1101



N-point
Choose n-points
split parents at these points



Uniform
For each index, choose parent at random
Other child gets Allele from the other parent

CROSSOVER FOR INTEGERS

N-Point Crossover:

Select N Random Crossover Points:

Choose N random positions along the chromosomes of both parents.
Alternate Crossover Segments

Example:

Parent 1: [1, 2, 3, 4, 5, 6, 7, 8]
Parent 2: [8, 7, 6, 5, 4, 3, 2, 1]
Crossover Points: ↑ ↑ ↑

Offspring 1: [1, 2, 3, 5, 4, 3, 2, 1]
Offspring 2: [8, 7, 6, 4, 5, 6, 7, 8]

.

Uniform Crossover:

Create a binary mask of the same length as the chromosome. Each bit represents whether the corresponding gene is inherited from Parent 1 (0) or Parent 2 (1).

Apply the binary mask to both parents to create offspring.

Example:

Parent 1: [1, 2, 3, 4, 5, 6, 7, 8]
Parent 2: [8, 7, 6, 5, 4, 3, 2, 1]
Binary Mask: [0, 1, 0, 1, 0, 1, 0, 1]

Offspring 1: [1, 7, 3, 5, 4, 3, 2, 1]
Offspring 2: [8, 2, 6, 4, 5, 6, 7, 8]

In this example, a binary mask is created, and each gene in the offspring is selected from either Parent 1 or Parent 2 based on the mask

CROSSOVER FOR REAL VALUED REPRESENTATIONS

Discrete Recombination:

(N-point crossover)
For Each Gene in the Chromosome:
Independently choose one of the parent genes with a probability of 0.5.
Create an offspring by selecting genes based on the choices made for each gene.

Example:

Parent 1: [2.5, -1.8, 0.7, 3.2, -4.5]
Parent 2: [1.0, 3.0, 2.0, 4.0, -2.0]

Offspring: [2.5, 3.0, 0.7, 4.0, -2.0]

In this example, each gene in the offspring is independently chosen from either Parent 1 or Parent 2 with equal probability.

Intermediate Recombination:
involves taking the weighted average of the corresponding genes from the parents to create the offspring. The weight of each parent's gene is determined by a mixing ratio (alpha), where alpha is a value between 0 and 1.

Calculate the weighted average of the corresponding genes from both parents using the mixing ratio (alpha).
Use the weighted averages to create the offspring.

Example:

Parent 1: [2.5, -1.8, 0.7, 3.2, -4.5]
Parent 2: [1.0, 3.0, 2.0, 4.0, -2.0]
Mixing Ratio (alpha): 0.7

Offspring: [1.85, 0.6, 1.65, 3.52, -3.25]

$X_{child\_1} = \alpha X_{parent\_1} + (1 - \alpha) X_{parent\_2}$
$X_{child\_2} = (1 - \alpha) X_{parent\_2} + \alpha X_{parent\_1}$
α Average of parents, if $\alpha = 0.5$

CROSSOVER FOR PERMUTATIONS - PMX

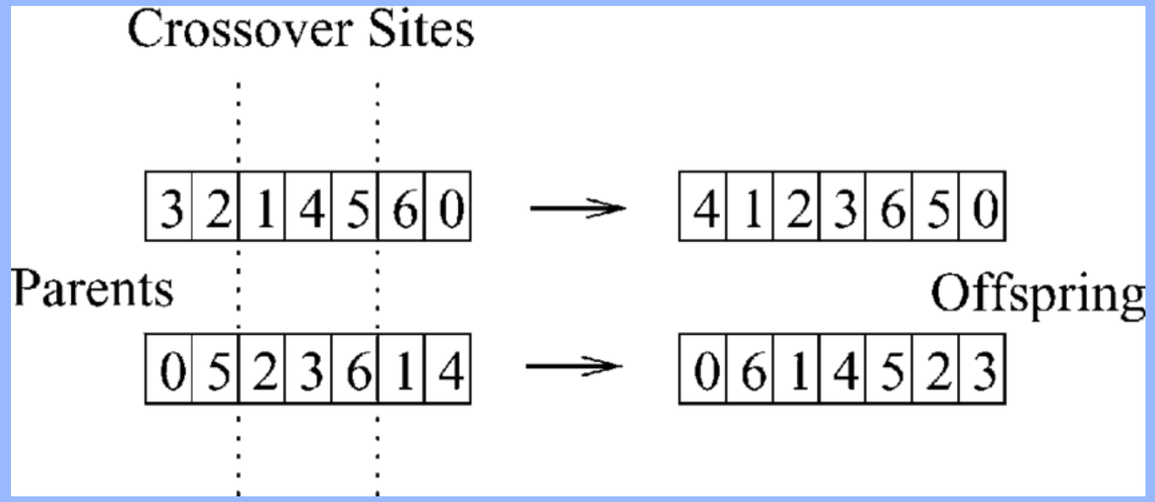
Partially Mapped Crossover (PMX):
It creates offspring by **preserving the relative order of selected elements** from both parents while filling in the remaining positions based on the information from the parents.

- Two random crossover points are chosen.
- The segment between the crossover points is copied from one parent to the corresponding positions in the offspring.
- The remaining positions are filled by mapping the values from the other parent, ensuring no duplication.

Example:

Parent 1: 1 2 3 | 4 5 6 7 8 9
Parent 2: 5 7 2 | 1 8 3 6 4 9
Crossover Points: ↑ ↑

Offspring 1: 5 7 3 | 4 1 2 6 8 9
Offspring 2: 1 2 7 | 5 8 3 6 4 9



PMX is **designed to maintain the relative order of the selected elements**, making it particularly useful for problems where the order of elements is crucial.

EDGE RECOMBINATION CROSSOVER

is often applied to permutation problems such as the Traveling Salesman Problem (TSP). It creates offspring by combining the edges of both parents, attempting to preserve as many edges as possible.

- Create a list of edges for each parent based on the order of elements in the permutation.
- Start with an initial city (commonly chosen randomly).
- Select the next city based on the common edge with the current city in either parent.
- Repeat until all cities are included in the offspring.

Example:

Parent 1: 1 2 3 4 5
Parent 2: 2 4 5 1 3

Offspring: 1 2 4 5 3

Edge Recombination emphasizes preserving edges present in both parents, leading to diverse and potentially high-quality solutions in TSP-like problems.

PERMUTATION REPRESENTATIONS: CONSERVING ORDER

Order Crossover (OX):
creates offspring by preserving a subset of genetic material from the parent individuals while filling in the remaining positions based on the information from the parents.

- Two random crossover points are chosen.
- The segment between the crossover points is copied from one parent to the corresponding positions in the offspring.
- The remaining positions are filled by copying genetic material from the other parent, excluding elements already present in the copied segment.

Parent 1: 1 2 3 | 4 5 6 7 8 9
Parent 2: 9 3 7 | 8 2 6 5 1 4
Crossover Points: ↑ ↑

Offspring 1: 9 3 2 | 4 5 6 7 8 1
Offspring 2: 1 2 7 | 8 9 3 6 5 4

Order Crossover aims to maintain the relative order of elements while introducing diversity from both parents.

Cycle Crossover:
creates offspring by identifying cycles of genetic material between the parent individuals.

- Identify cycles of genetic material between the parents using alternating positions.
- Copy genetic material from one parent for odd cycles and from the other parent for even cycles.
- Fill in the remaining positions by copying genetic material from the other parent.

Parent 1: 1 2 3 4 5
Parent 2: 2 4 5 1 3

Cycles: (1 2 4) (3 5)

Offspring: 1 2 4 5 3

Cycle Crossover tends to preserve blocks of genetic material from both parents and is known for producing diverse offspring.