National University of Computer and Emerging Sciences

Lab Manual 05
Artificial Intelligence Lab

Department of Computer Science
FAST-NU, Lahore, Pakistan

Table of Contents

1 Objectives

After performing this lab, students shall be able to understand the following Python concepts and applications:

- ✔ Genetic Algorithm
- ✔ Problem solving using Genetic Algorithm

2 Task Distribution

Total Time	**170 Minutes**
Genetic Algorithm Introduction	25 Minutes
Application of Genetic Algorithm	25 Minutes
Exercise	120 Minutes
Online Submission	10 Minutes

3 Genetic Algorithm

3.1 History

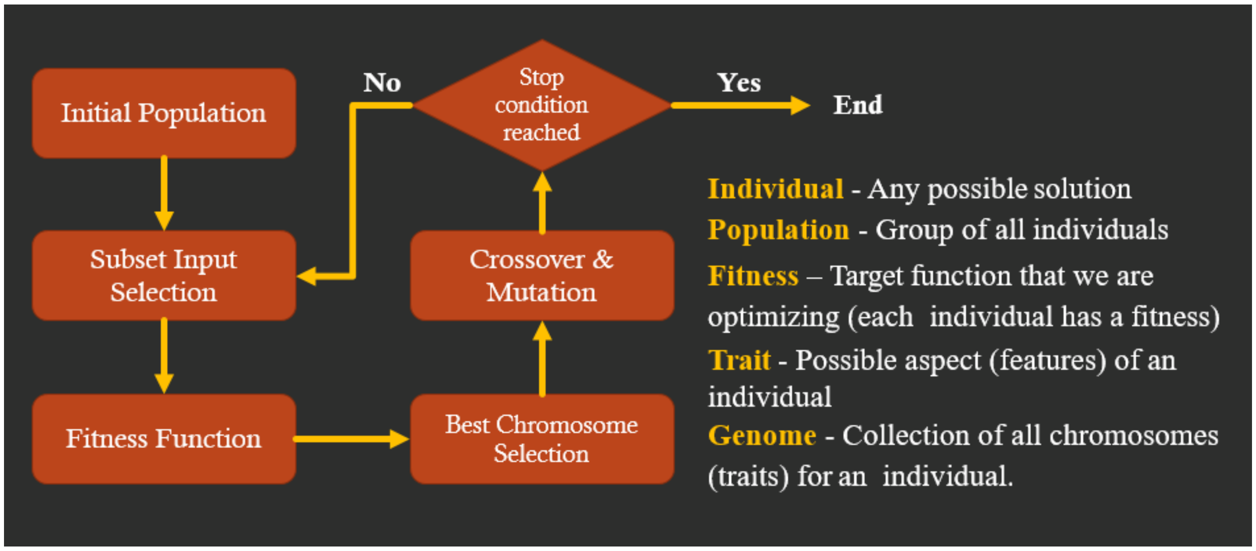
- As early as 1962, John Holland's work on adaptive systems laid the foundation for later developments.
- By the 1975, the publication of the book Adaptation in Natural and Artificial Systems, by Holland and his students and colleagues.
- Early to mid-1980s, genetic algorithms were being applied to a broad range of subjects.
- In 1992, John Koza has used genetic algorithm to evolve programs to perform certain tasks. He called his method "genetic programming" (GP).
- A genetic algorithm (or GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
- (GA)s are categorized as global search heuristics.
- (GA)s are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

3.2 What is Genetic Algorithm (GA)?

- The evolution usually starts from a population of randomly generated individuals and happens in generations.
- In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness), and modified to form a new population.

- The new population is used in the next iteration of the algorithm.
- The algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

3.3 Genetic Algorithm Flow



Individual - Any possible solution
Population - Group of all individuals
Fitness – Target function that we are optimizing (each individual has a fitness)
Trait - Possible aspect (features) of an individual
Genome - Collection of all chromosomes (traits) for an individual.

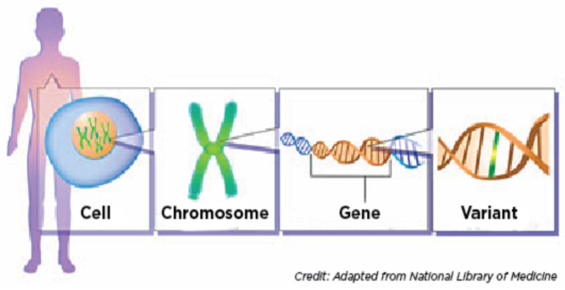
3.4 Population

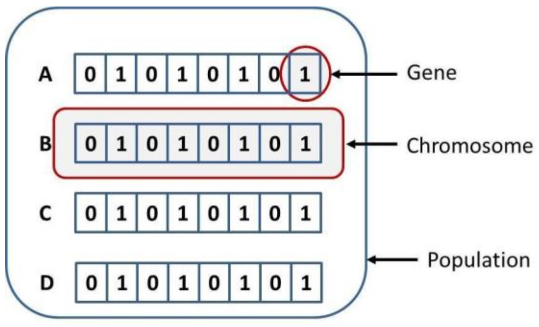
Biology

- Collection of individuals.

Algorithm

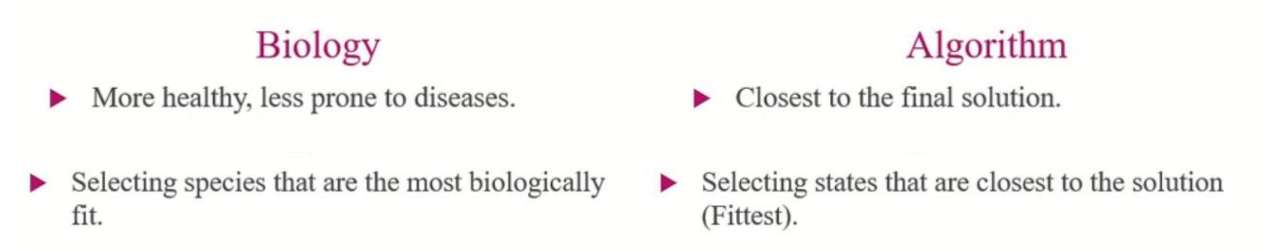
- Collection of states.





Gene, Chromosomes and Population

3.5 Fitness and Selection

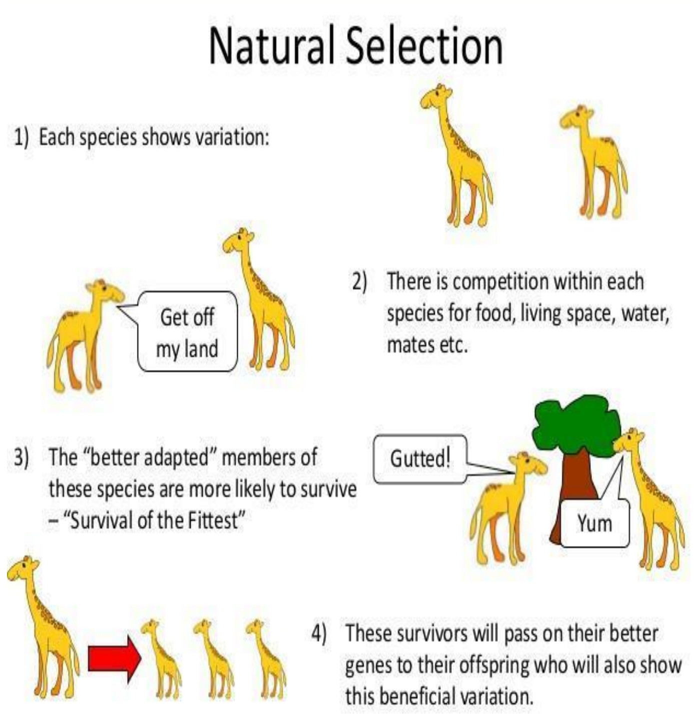


Biology

- More healthy, less prone to diseases.
- Selecting species that are the most biologically fit.

Algorithm

- Closest to the final solution.
- Selecting states that are closest to the solution (Fittest).



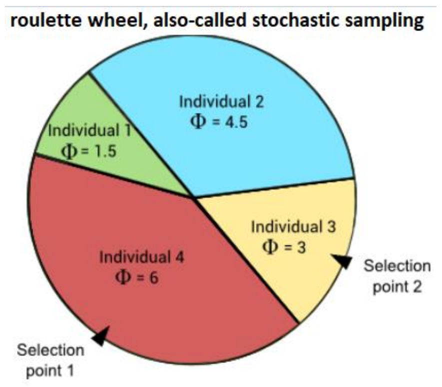
Natural Selection

1) Each species shows variation:

2) There is competition within each species for food, living space, water, mates etc.

3) The "better adapted" members of these species are more likely to survive – "Survival of the Fittest"

4) These survivors will pass on their better genes to their offspring who will also show this beneficial variation.



	Selection error	Rank	Fitness
Individual 1	0.9	1	1.5
Individual 2	0.6	3	4.5
Individual 3	0.7	2	3.0
Individual 4	0.5	4	6.0

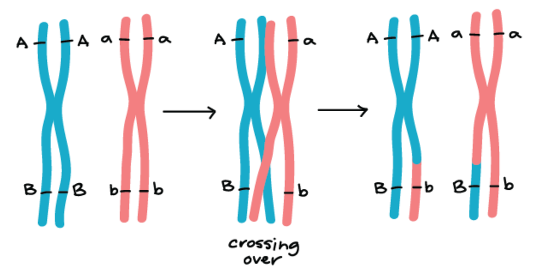
3.6 Mating / Crossover / Generating Generations

Biology

- Mating or Reproducing

Algorithm

- Interchanging values between selected states.



Individual 3	1	1	0	0	0	1
Individual 4	0	1	0	1	0	0
Offspring 1	0	1	0	1	0	1
Offspring 2	1	1	0	1	0	1
Offspring 3	0	1	0	1	0	1
Offspring 4	1	1	0	0	0	0

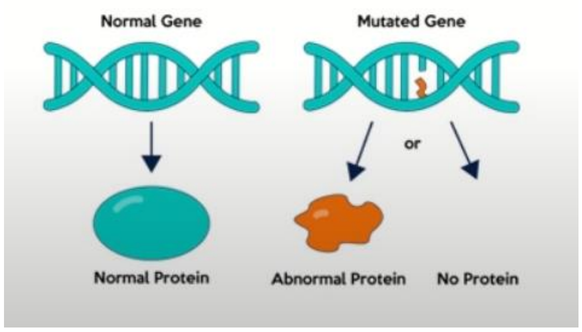
3.7 Mutation

Biology

- Change or variation.

Algorithm

- Alteration.

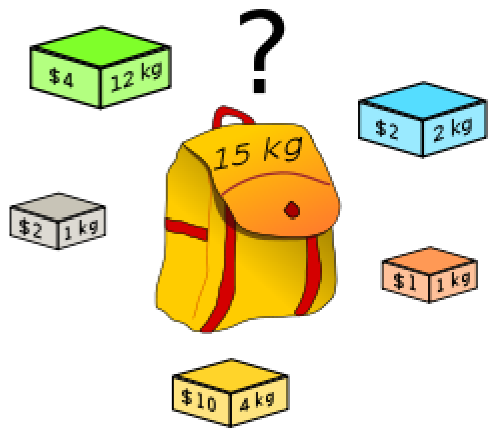


Offspring1: Original	0	1	0	1	0	1
Offspring1: Mutated	0	1	0	0	0	0

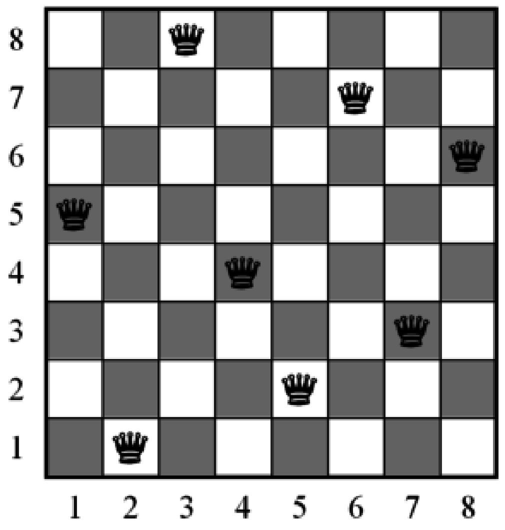
4 Problem Solving using Genetic Algorithm

Famous problems which use Genetic Algorithms:

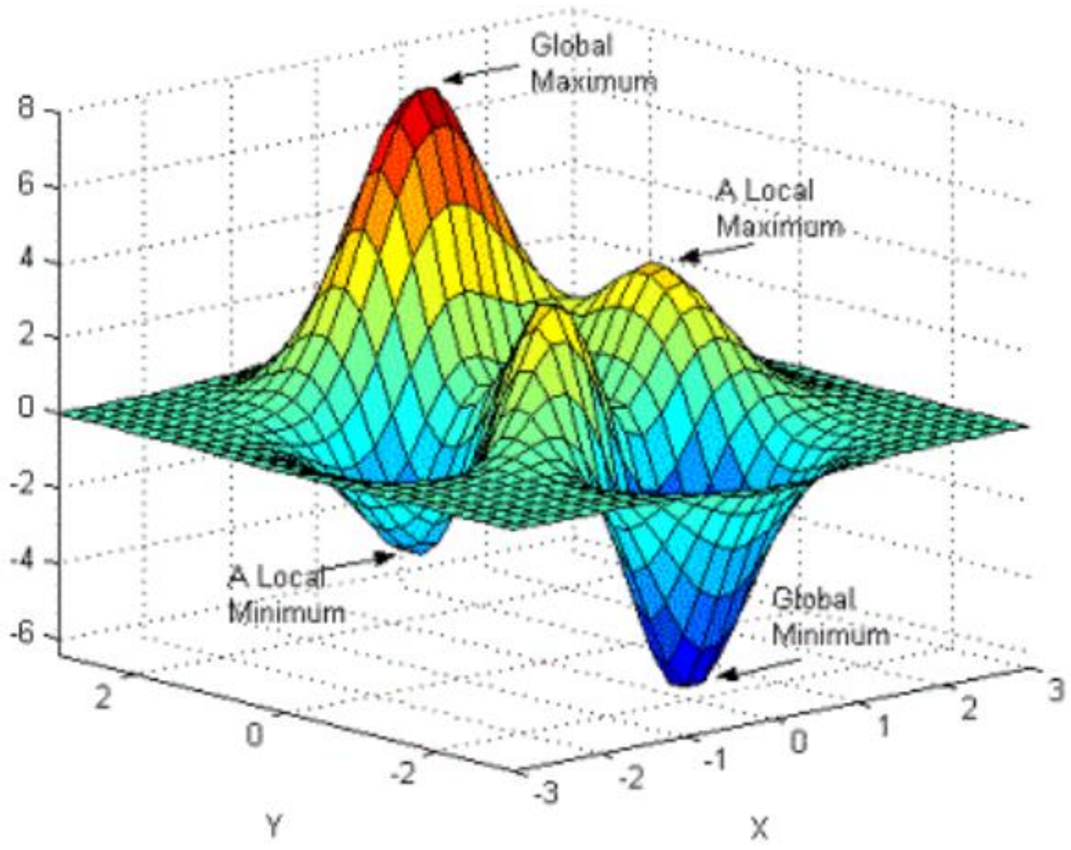
<u>KnapSack Problem</u>



<u>8 Queen Problem</u>



Optimization Problems/ Finding Global Maxima of a Function



Let's try to find optimized solution of a function:
Execute the attached notebook for this problem and follow the step by step details below for understanding.
Let's have random expression which evaluates to 25:
$6x^3 + 9y^2 + 90z^1 = 25$

Problem: What could be the best possible values of x, y and z that could satisfy the above expression.
For this purpose, we make function in python such that:

```
def evaluateExpression(x, y, z):
  return 6 * x ** 3 + 9 * y ** 2 + 90 * z - 25
```

This function that evaluates the expression to 0 if the answer of expression " 6 * x ** 3 + 9 * y ** 2 + 90 * z " is 25. So that means we need the most suitable values of x, y and z so that we could achieve our target value which is 25 in this case.

Now we need to apply Genetic Algorithm concepts to solve this problem.

Step1: Population of Solutions.
Population is generated entirely from random numbers let say up to 1000 individuals.

```
# generate solutions population
import random

solutions = []
for counter in range(1000):
  solutions.append((random.uniform(0, 1000), random.uniform(0, 1000), rand
om.uniform(0, 1000)))
```

Step2: Fitness function: So the fittest solution will be the one, which evaluates the expression to "0". Otherwise, the best solution will be closest to zero. Therefore, the fitness in this case is the highest if the solution is closest to zero. Hence, we can return highest fitness value to those solutions, which are closest to 0.

```
def fitness(x, y, z):
  ans = evaluateExpression(x, y, z)

  if ans == 0:
    return 99999
  else:
    return abs(1 / ans)
```

Step3: Mating, Crossover or Generating the Generations:
During each generations, further sub steps performed such as:
Step 3.1: Selection of top ranked solutions
Step 3.2: Mutation or slight changes or variation in values of solution.
P.S. Here for the sake of analogy if solution can be considered as chromosome then variable values can be considered as genes)

```
for generation_count in range(10000):
  rankedSolutions = []
  # fitness step
  for solution in solutions:
    rankedSolutions.append((fitness(solution[0], solution[1], solution[2])
, solution))
  rankedSolutions.sort()
  rankedSolutions.reverse()
  print(f"=== Generation {generation_count} best solutions ====")
  print(rankedSolutions[0])

  if rankedSolutions[0][0] > 999:
    break

  bestSolution = rankedSolutions[:100]
  # print(bestSolution)

  # selection step
  variables = []
  for solution in bestSolution:
    variables.append(solution[1][0])  # variable x
    variables.append(solution[1][1])  # variable y
    variables.append(solution[1][2])  # variable z

  newGeneration = []
  # mutation step
  for counter in range(1000):
    x = random.choice(variables) * random.uniform(0.99, 1.01)
    y = random.choice(variables) * random.uniform(0.99, 1.01)
    z = random.choice(variables) * random.uniform(0.99, 1.01)

    newGeneration.append((x, y, z))

  solutions = newGeneration
```

Note: After running this code given in Jupyter Notebook, confirm the values from best optimized solution into given expression to verify.

4.1 Exercise:

A thief enters a shop carrying bag which can carry 35 kgs of weight. The shop has 10 items, each with a specific weight and price. Now, the thief's dilemma is to make such a selection of items that it maximizes the value (i.e., total price) without exceeding the knapsack weight. We have to help the thief to make the selection.
Available Items are:

Item No.	Weight	Value
1	3	266
2	13	442
3	10	671
4	9	526
5	7	388
6	1	245
7	8	210
8	8	145
9	2	126
10	9	322

Initial population:

```
[[0 1 0 1 1 0 0 1 1 1]
 [1 1 1 1 0 1 1 1 0 0]
 [0 1 0 0 0 0 1 1 0 1]
 [0 0 1 0 1 1 0 0 0 0]
 [0 0 1 1 0 0 0 0 0 1]
 [0 1 0 1 1 0 1 0 0 0]
 [1 1 1 0 0 0 1 0 1 0]
 [0 0 0 0 1 1 1 0 0 0]
```

Fitness Function:

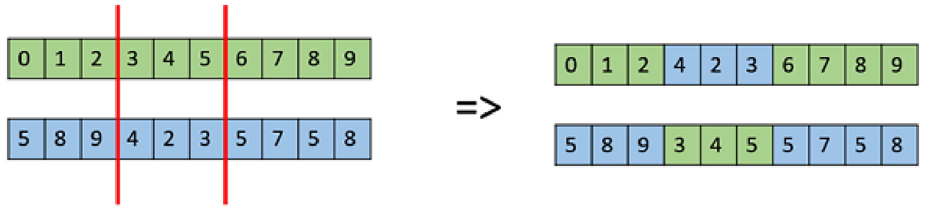
$$fitness = \sum_{i=1}^{n} c_i v_i; if \sum_{i=1}^{n} c_i w_i \leq kw$$

$$fitness = 0; otherwise$$

where,
n = chromosome length
Ci = ith gene
Vi = ith value
Wi = ith weight
kw = allowed weight

To generate new population:

1- Two-point crossover
   Example of two-point cross-over. Choose the positions as in the example.



2- Mutate offspring got from cross over at 2 random positions. Select 2 random positions: if you find 1 change it to 0 and vice versa.

Always keep 8 best chromosomes, 50% from initial population and 50% from new generation. If your program taking time you can iterate for fixed number of iterations.

5 Submission Instructions

Always read the submission instructions carefully.

- Rename your Jupyter notebook to your roll number and download the notebook as **.ipynb** extension.
- To download the required file, go to **File->Download .ipynb**
- Only submit the **.ipynb** file. DO NOT **zip** or **rar** your submission file.
- Submit this file on Google Classroom under the relevant assignment.
- Late submissions will not be accepted.