DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
Course Code: CS-323
Course Title: Artificial Intelligence
Open Ended Lab
TE Batch 2022, Fall Semester 2024Grading Rubric
Group Members:

Student No.	Name	Roll No.
S1	Muhammad Ibrahim	CS-22119
S2	Muhammad	CS-22123
S3		

CRITERIA AND SCALES				Marks Obtained		
				S1	S2	S3
Criterion 1: Has the student appropriately simulated the working of the genetic algorithm?						
0	1	2	-			
The explanation is too basic.	The algorithm is explained well with an example.	The explanation is much more comprehensive.				
Criterion 2: How well is the student's understanding of the genetic algorithm?						
0	1	2	3			
The student has no understanding.	The student has a basic understanding.	The student has a good understanding.	The student has an excellent understanding.			
Criterion 3: How good is the programming implementation?						
0	1	2	3			
The project could not be implemented.	The project has been implemented partially.	The project has been implemented completely but can be improved.	The project has been implemented completely and impressively.			
Criterion 4: How good is the selected application?						
0	1	2	-			
The chosen application is too simple.	The application is fit to be chosen.	The application is different and impressive.				
Criterion 5: How well-written is the report?						
0	1	2	-			
The submitted report is unfit to be graded.	The report is partially acceptable.	The report is complete and concise.				
			Total Marks:			

Complete Working of a Genetic Algorithm (GA)

A Genetic Algorithm (GA) is a search heuristic inspired by the process of natural evolution. It uses concepts like selection, crossover, and mutation to find optimal or near-optimal solutions to problems. Here's a step-by-step breakdown with an example.

Steps in a Genetic Algorithm

1. Problem Representation (Chromosome Encoding)
Represent potential solutions (individuals) as chromosomes. The representation can vary:

Binary strings (e.g., 10101)

Real numbers (e.g., [3.5, 2.1])

Permutations (e.g., [A, B, C, D])

2. Define Parameters

Population size N: Number of individuals in each generation.

Crossover probability p_c: Likelihood of crossover occurring.

Mutation probability p_m: Likelihood of mutation occurring.

Stopping criteria: Number of generations or a fitness threshold.

3. Define the Fitness Function
The fitness function evaluates how "good" a solution is. For example, if maximizing $f(x)=x^2$, the fitness of a chromosome x=5 is f(5) =25.

4. Generate Initial Population
Randomly generate N chromosomes within the problem domain.

5. Calculate Fitness
Evaluate each chromosome using the fitness function.

6. Selection
Select pairs of chromosomes (parents) for reproduction, based on fitness:

Roulette wheel selection: Higher fitness → higher chance of selection.

Tournament selection: Compete subsets of individuals; best ones win.

7. Crossover (Recombination)
Combine two parents to produce offspring by swapping segments of their genes.

8. Mutation
Randomly alter genes to maintain diversity. This prevents the population from getting stuck in local optima.

9. Replace Old Population
Replace the current population with the new offspring.

10. Repeat
Continue until the stopping criteria are met (e.g., maximum generations).

Example: Maximize $f(x)= x^2$ for $x\in [0,31]$

Step-by-Step Execution

1. Chromosome Representation:

Each chromosome represents an integer x encoded as a 5-bit binary string.
Example: $x = 19 \rightarrow 10011$

2. Population Initialization:
Randomly generate a population of size N=4:
[01101,10010,10101,11001]
Decode to [13,18,21,25]

3. Fitness Calculation:
Use $\mathbf{f(x) = x^2}$:
[f(13) = 169, f(18) = 324, f(21) = 441, f(25) = 625]

4. Selection:
Use roulette wheel selection:

Total fitness = 169 + 324 + 441 + 625 = 1559

Probabilities:
P (13) = 169/1559, P (18) = 324/1559, P (21) = 441/1559, P (25) = 625/1559
Randomly select pairs based on these probabilities.

5. Crossover:
For parents 10010(18) and 11001(25):

Perform single-point crossover at position 3:
100| 10 and 110| 01 → 10001(17) and 11010(26)

6. Mutation:
If mutation occurs with $p_m = 0.1$, flip one random bit:
10001 → 10011(19)

7. New Population:
After crossover and mutation:
[19, 26, ...]

8. Repeat:
Evaluate fitness for the new population and repeat until a stopping condition is met.

Iterative Process

Over successive generations, the population evolves toward the optimal solution. For $f(x) = x^2$, the algorithm will converge to $x = 31$ (the maximum possible x) with $f(31) = 961$.

Problem Description

Traveling Salesman Problem (TSP)

The **Traveling Salesman Problem (TSP)** is one of the most famous optimization problems in computer science and operations research. It is a **combinatorial optimization problem** where the objective is to find the shortest possible route that allows a salesman to:

1. Visit each city exactly once.
2. Return to the starting city.

Problem Statement

Input:

A list of cities and the distances (or costs) between every pair of cities.

Output:

The shortest route (a permutation of cities) that satisfies the problem's constraints.

Example

Suppose there are 4 cities: A, B, C, D

The distance matrix is:

	A	B	C	D
A	0	10	15	20
B	10	0	35	25
C	15	35	0	30
D	20	25	30	0

- One possible tour: A→B→C→D→A
- Total cost: 10+35+30+20=95
- The goal is to find the tour with the **minimum cost**