ARTIFICIAL INTELLIGENCE

ASSIGNMENT # 02

SUBMITTED BY : ZAINAB NAVEED
ROLL NO. BSDSF22M013
SUBMITTED TO : DR. SYED MUHAMMAD ALI

QUESTION # 01

GENETIC ALGORITHM :

The genetic algorithm (GA) is a heuristic optimization technique inspired by the principles of natural selection and genetics. It aims to explore a wide solution space efficiently while ensuring that the fittest solutions evolve over generations. Genetic algorithm does so using the following genetic operators;

- Mutation
- Crossover
- Inversion
- Exchange

MUTATION : Mutation involves the random selection and changes to individual genes in a parent chromosome from a population of solutions. Mutation is important because;

- GENETIC DIVERSITY : It ensures diversity in the solutions i.e if mutation does not occur the solutions can stop evolving after sometime and get stuck in some local minima. Mutation ensures constant evolution of the solution in further generations.
- PRESERVATION OF SKILLS : It involves toggling a few bits only ensuring that certain skills of the parent, making them the best fit ones in their generation, are preserved in the future generations.

For example, consider the following parent and resultant child bit strings after mutating the second bit of the parent;
Parent = 11111111
Child = 10111111

CROSSOVER : Crossover involves the recombination of genetic material from two parents in order to create a child in the next generation having certain traits of both the parents.

- GENETIC DIVERSITY : It ensures genetic diversity by selecting random cut points from two parents in the solution population and then performing crossover so that different traits come from different parents in the child solution which in the later stage is again involved in recombination in the next generations.
- PRESERVATION OF SKILLS : Since some specific portion of genes is taken from both the parents, some of the traits of either parents are preserved in the child to be transferred to the next generation.

For example consider two parents i.e.

- Parent 1 = 10101010
- Parent 2 = 11100010

If we take the crossover point after bit number 4, we will get the following two solutions in case of a single point crossover;

- Child 1 = 10101110
- Child 2 = 10100010

INVERSION : Inversion involves the backward rearrangement of the solution i.e the last bit of the parent becomes the first bit in the child solution.

- GENETIC DIVERSITY : Searches for new patterns by reordering the parent solution completely and enhancing diversity, however, avoiding completely random changes.
- PRESERVATION OF SKILLS :Useful patterns are retained yet rearranged in search of better and more promising solutions.

For example consider the following parent and resultant child solutions after inversion;

- Parent = 11001100
- Child = 00110011

EXCHANGE : The interchange of genetic material of a parent chromosome in order to produce a child chromosome is called exchange. Some certain bits of parent are selected and exchanged to get the child solution.

- GENETIC DIVERSITY : By rearranging the genes, the algorithm can explore different solutions.
- PRESERVATION OF SKILLS : The traits of the parents are preserved. Only rearrangement of a couple or more genes of the parent takes place.

For example the parent and consequent child solutions after exchange of bit 2 and 7 are given below;

- Parent = 11110000
- Child = 10110010

QUESTION # O2

Challenges in Representation for TSP :

In the Traveling Salesman Problem (TSP), cities must be represented as a valid permutation of numbers indicating their sequence. An inappropriate representation can cause:

- Duplicate cities in the route.
- Omission of cities in the solution.

To address this, permutation-based representations are preferred. However, standard genetic operators like crossover and mutation can disrupt the validity of solutions, requiring specialized operators.

Alternative Genetic Operators

1. **Inversion Mutation**:
   This mutation selects a subsequence of cities, reverses it, and preserves all cities' uniqueness.
   Example:
   Original: 1 2 3 4 5 6 7 8
   Invert cities 2 to 4 = 1 4 3 2 5 6 7 8
2. **Cycle Crossover (CX)**:
   Cycle crossover prevents city duplication by identifying cycles of positions

between two parents and swapping them to form valid children.
Example:
Parent 1: 1 2 3 4 5
Parent 2: 2 3 1 5 4
Child: 1 3 2 4 5

3. **Modified Fitness Function :**

Fitness in TSP is calculated using the total route distance. A normalized score Invalid solutions (not permutations) receive a high penalty to eliminate them quickly.

Normalized Fitness Score = 1/ Total route distance

This ensures that shorter routes have higher fitness values.

QUESTION # 04

0-1 Knapsack Problem Using Genetic Algorithm

The 0-1 Knapsack Problem focuses on selecting items with specific weights and values to maximize the total value while staying within a weight capacity. A genetic algorithm (GA) effectively solves this optimization problem.

REPRESENTATION :

A solution is represented as a binary string:

- 1 → Item is selected.
- 0 → Item is not selected.

Example:
For 5 items, the chromosome 1 0 1 0 1 means items 1, 3, and 5 are selected.

Fitness Function

The fitness evaluates a solution based on total value while penalizing overweight solutions:

Fitness = Total Value - P * (Excess Weight)

Where:

- **Total Value** = Sum of selected items' values.
- **Excess Weight** = Total weight of selected items − Knapsack capacity.
- **P** = Penalty factor.

Example:

- Items' values: `[10, 20, 15, 25]`
- Weights: `[5, 10, 20, 15]`
- Capacity: `30`
- Chromosome: `1 0 1 1`
- Total Value = `10 + 15 + 25 = 50`
- Total Weight `40`
- Excess Weight = `10` → Penalty = `P × 10`.

Selection

- **Tournament Selection**: Randomly pick a group of solutions and select the best one based on fitness.
- **Roulette Wheel Selection**: Solutions are chosen probabilistically based on fitness.

Crossover

- **Single-Point Crossover**: A point is randomly chosen to swap binary segments between parents, creating offspring.
  Example:
  Parent 1: `1 0 | 1 1 0`
  Parent 2: `0 1 | 0 1 1`
  Children: `1 0 0 1 1` and `0 1 1 1 0`

Mutation

Mutation flips a random bit in the chromosome (`0 → 1` or `1 → 0`) to maintain diversity and avoid local optima.
Example:
Before Mutation: `1 0 1 0 1` → Flip bit 3 → `1 0 0 0 1`

Replacement

- **Elitism**: Retains the best solutions for the next generation.
- **Fitness-Based Replacement**: Low-fitness solutions are replaced with better offspring

QUESTION # 04

```
def read_input_file(file_path):
    with open(file_path, 'r') as file:
      lines = [line.strip() for line in file if line.strip()]
      n = int(lines[0])  # Number of items
      W = int(lines[1])  # Knapsack capacity
      items = []
      for i in range(2, 2 + n):
          weight, value = map(int, lines[i].split())
          items.append((weight, value))
      return n, W, items

def knapsack_01(n, W, items):
    # Initialize DP table
    dp = [[0 for _ in range(W + 1)] for _ in range(n + 1)]

    # Fill the DP table
    for i in range(1, n + 1):
        weight, value = items[i - 1]
        for w in range(W + 1):
            if weight <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weight] + value)
            else:
                dp[i][w] = dp[i - 1][w]

    # Reconstruct the selected items (optional)
    selected_items = []
    w = W
    for i in range(n, 0, -1):
        if dp[i][w] != dp[i - 1][w]:
            selected_items.append(items[i - 1])
```

```
            w -= items[i - 1][0]

    # Return the maximum value and selected items
    return dp[n][W], selected_items



def process_problem(file_path):
        try:
         # Read input
         n, W, items = read_input_file(file_path)

         # Solve the 0-1 Knapsack problem
         max_value, selected_items = knapsack_01(n, W, items)

         # Format output
         result = f"Maximum Value: {max_value}\nSelected Items (weight, value):
{selected_items}"
         return result
    except Exception as e:
         return f"Error: {e}"

def main():
     # Input file path
    input_file_path = input("Enter the path to the input text file: ")

    try:
         # Process the knapsack problem
         solution = process_problem(input_file_path)
         print("\nSolution:")
         print(solution)
    except FileNotFoundError:
         print("Error: The specified file was not found.")

if __name__ == "__main__":
    main()
```