Faculty of Computing and Information Technology

University of the Punjab,

Lahore

Artificial Intelligence Lab 9

Instructor: Qamar U Zaman

Maximum Ones Problem Using Genetic Algorithm

Objective:

Optimize a binary string of length nnn to maximize the number of ones using Genetic Algorithms.

Problem Description:

The Maximum Ones Problem is a foundational optimization problem. The aim is to evolve a population of binary strings to maximize the number of `1`s in each string.

For example:

- **Initial String:** `1010010110` (4 ones)
- **Optimal String:** `1111111111` (10 ones for n=10n = 10n=10)

Genetic Algorithms provide a robust framework to solve this problem through evolutionary processes.

Key Concepts:

1. **Population:** A group of binary strings (potential solutions).
2. **Chromosome:** A single binary string representing a candidate solution.
3. **Gene:** A single bit (0 or 1) in the binary string.
4. **Fitness Function:** Counts the number of ones in a string. Strings with more ones have higher fitness.
5. **Selection:** Picks the most "fit" strings as parents for reproduction.
6. **Crossover:** Combines two parent strings to create offspring.
7. **Mutation:** Randomly flips a bit in the offspring to maintain diversity.

Steps:

1. **Initialize Population:** Generate a set of random binary strings.
2. **Evaluate Fitness:** Count the number of ones in each string.
3. **Select Parents:** Choose two strings based on fitness scores.
4. **Perform Crossover:** Combine parts of the two parents to create offspring.
5. **Mutate Offspring:** Randomly flip bits in the offspring with a small probability.
6. **Repeat:** Evolve the population over several generations until an optimal solution is found or a maximum number of generations is reached.

Code Template:

```
# Initialize the population with random binary strings
def initialize_population(pop_size, string_length):
    # Create a list of random binary strings of given length
    pass

# Calculate fitness for an individual
def calculate_fitness(individual):
    # Count the number of ones in the binary string
    pass

# Select parents based on fitness
def select_parents(population, fitness_scores):
    # Choose two parents using methods like roulette wheel or tournament
selection
    pass

# Perform crossover to generate offspring
def crossover(parent1, parent2):
    # Combine parts of two parents to create a new offspring
    pass

# Apply mutation to introduce diversity
def mutate(individual, mutation_rate):
    # Randomly flip bits in the binary string
    pass

# Genetic Algorithm implementation
def genetic_algorithm(string_length, pop_size, num_generations,
mutation_rate):
    # Evolve the population over generations to maximize the number of ones
    pass
```

Task Description:

Students are expected to:

1. Implement the logic for all functions in the template.
2. Experiment with:
    - Different string lengths (e.g., 10, 20, 50).
    - Population sizes (e.g., 20, 50, 100).
    - Mutation rates (e.g., 0.01, 0.05, 0.1).
3. Analyze how the population evolves over generations.