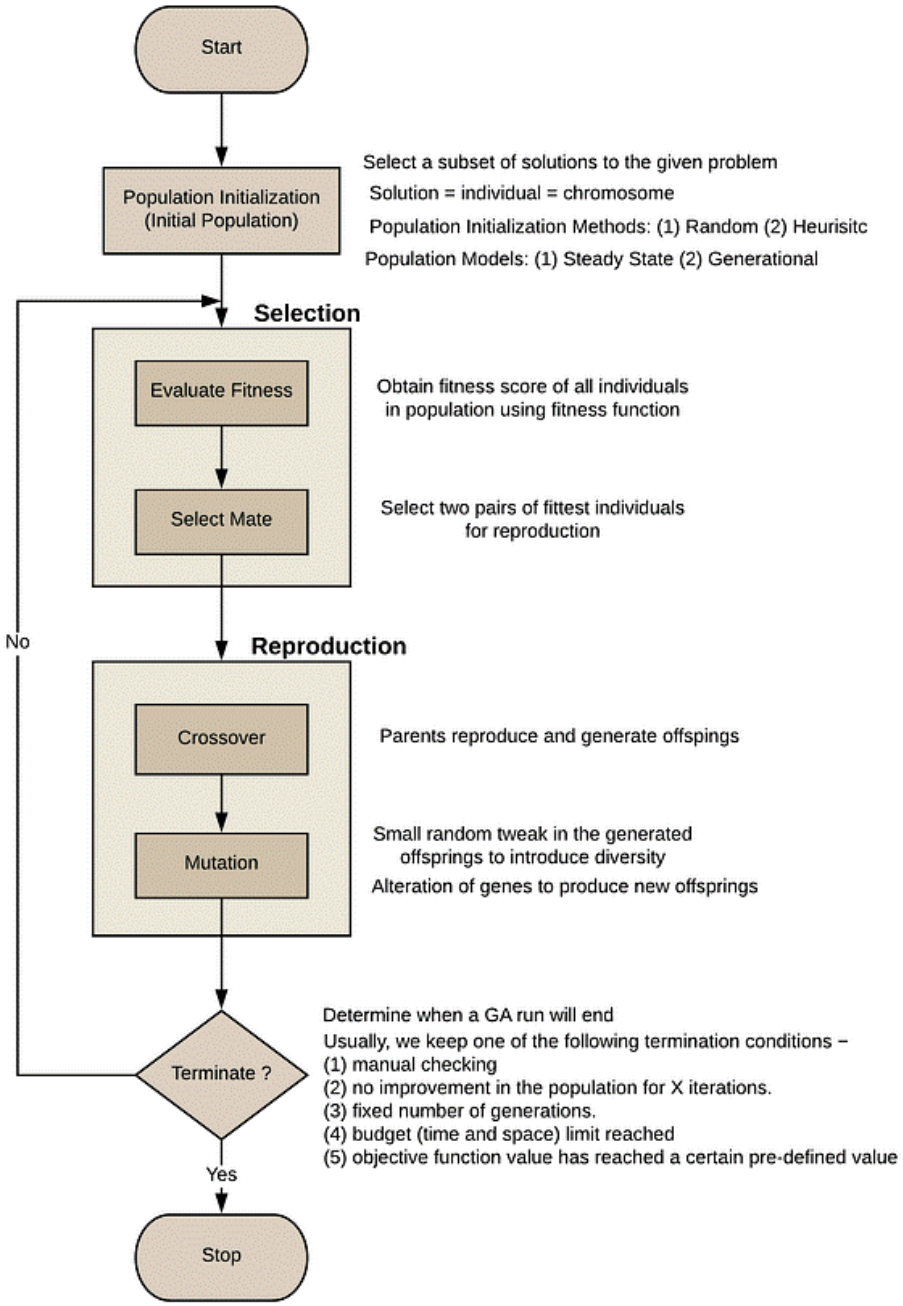
Lab 11

Genetic Algorithm:

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Structure of Genetic algorithm:



Question 1: Code Genetic algorithm in python

```
"""
import random
import matplotlib.pyplot as plt

# Define the target function to be maximized
def target_function(x):
    if x is None:
        return float('-inf')  # Handle None case by returning a very low fitness
    return -(x ** 2) + 8.8

# Genetic algorithm parameters
POPULATION_SIZE = 100
NUM_GENERATIONS = 100
MUTATION_RATE = 0.1

# Function to initialize a random individual
def initialize_individual():
    return random.uniform(-5, 5)

# Function to evaluate the fitness of an individual based on the target function
def evaluate_fitness(individual):
    return target_function(individual)


# Function to perform selection based on roulette wheel selection
def roulette_wheel_selection(population, fitness_values):
    total_fitness = sum(fitness_values)
    selected_value = random.uniform(0, total_fitness)

    cumulative_fitness = 0
    for i, fitness in enumerate(fitness_values):
        cumulative_fitness += fitness
        if cumulative_fitness >= selected_value:
            return population[i]

# Function to perform crossover (single-point crossover)
def crossover(parent1, parent2):
    if parent1 is None or parent2 is None:
        return initialize_individual()  # Handle None case by reinitializing
    crossover_point = random.uniform(-5, 5)
    child = (parent1 + parent2) / 2  # Simple average crossover
    return child

# Function to perform mutation
def mutate(individual):
    mutation_value = random.uniform(-0.5, 0.5)
    return individual + mutation_value
```

```python
# Main genetic algorithm function
def genetic_algorithm(population_size, num_generations, mutation_rate):
    population = [initialize_individual() for _ in range(population_size)]
    best_individual = None
    best_fitness = float('-inf')
    all_best_fitness = []

    for generation in range(num_generations):
        # Evaluate fitness for each individual in the population
        fitness_values = [evaluate_fitness(individual) for individual in population]

        # Select parents for reproduction
        parents = [roulette_wheel_selection(population, fitness_values) for _ in range(population_size)]

        # Perform crossover to generate offspring
        offspring = [crossover(parents[i % population_size], parents[(i + 1) % population_size]) for i in range(population_size)]

        # Perform mutation on offspring
        offspring = [mutate(individual) if random.random() < mutation_rate else individual for individual in offspring]

        # Replace the old population with the new population (parents + offspring)
        population = parents + offspring

        # Update the best individual and fitness
        current_best_index = fitness_values.index(max(fitness_values))
        current_best_fitness = fitness_values[current_best_index]

        if current_best_fitness > best_fitness:
            best_fitness = current_best_fitness
            best_individual = population[current_best_index]

        all_best_fitness.append(best_fitness)

        print(f"Generation {generation + 1}: Best Fitness = {best_fitness:.4f}")

    return best_individual

# Run the genetic algorithm
result = genetic_algorithm(POPULATION_SIZE, NUM_GENERATIONS, MUTATION_RATE)
print(f"\nOptimal Solution: {result}")
print(f"Optimal Fitness: {evaluate_fitness(result):.4f}")
```