Experiment-2.3

Aim of the Experiment:

Implementation of Genetic Application- Travelling Salesman Problem.

Theory:

A genetic algorithm (GA) is a computational technique inspired by the process of natural selection and evolution. It is used to solve optimization and search problems by mimicking the process of natural selection and evolution in a population of candidate solutions. A genetic algorithm is an adaptive heuristic search algorithm inspired by "Darwin's theory of evolution in Nature."

Genetic Algorithm involves five phases to solve the complex optimization problems:

1) Initialization: Process of Genetic algorithm starts by generating the set of individuals, called population. Each individual is the solution for the given problem. An individual contains a set of parameters called Genes. Genes are combined into a string and generate chromosomes, which is the solution to the problem.

2) Fitness Assignment: Fitness function is used to determine how fit an individual is? In every iteration, individuals are evaluated based on their fitness function. The fitness function provides a fitness score to each individual. The high the fitness score, the more chances of getting selected for reproduction.

3) Selection: The selection phase involves the selection of individuals for the reproduction of offspring. All the selected individuals are then arranged in a pair of two to increase reproduction. Then these individuals transfer their genes to the next generation. (Roulette wheel selection, Tournament selection, Rank-based selection).

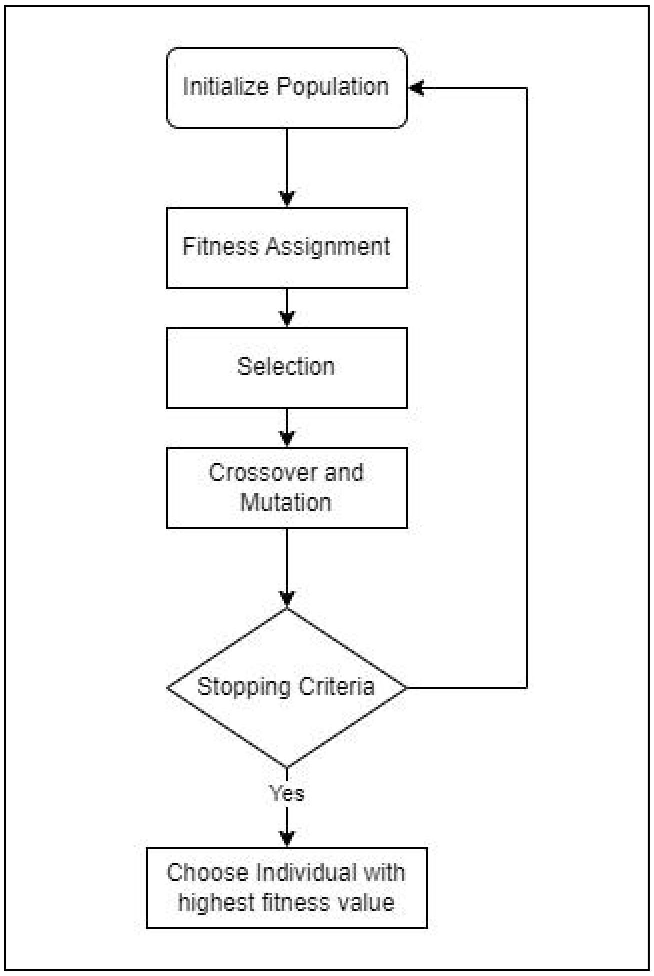
4) Reproduction: After the selection process, the creation of a child occurs in the reproduction step. In this step, the genetic algorithm uses two variation operators that are applied to the parent population.

a) Crossover: A crossover point is selected at random within the genes. Then the crossover operator swaps genetic information of two parents from the current generation to produce a new individual representing the offspring. The genes of parents are exchanged among themselves until the crossover point is met. (One point crossover, Two-point crossover).

b) Mutation: The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes. Mutation helps in solving the issue of premature convergence and enhances diversification. (Flip bit mutation, Gaussian mutation, Swap mutation).

5) Termination: After the reproduction phase, a stopping criterion is applied as a base for termination. The algorithm terminates after the threshold fitness solution is reached. It will identify the final solution as the best solution in the population.

<u>FlowChart for Genetic Algorithm:</u>

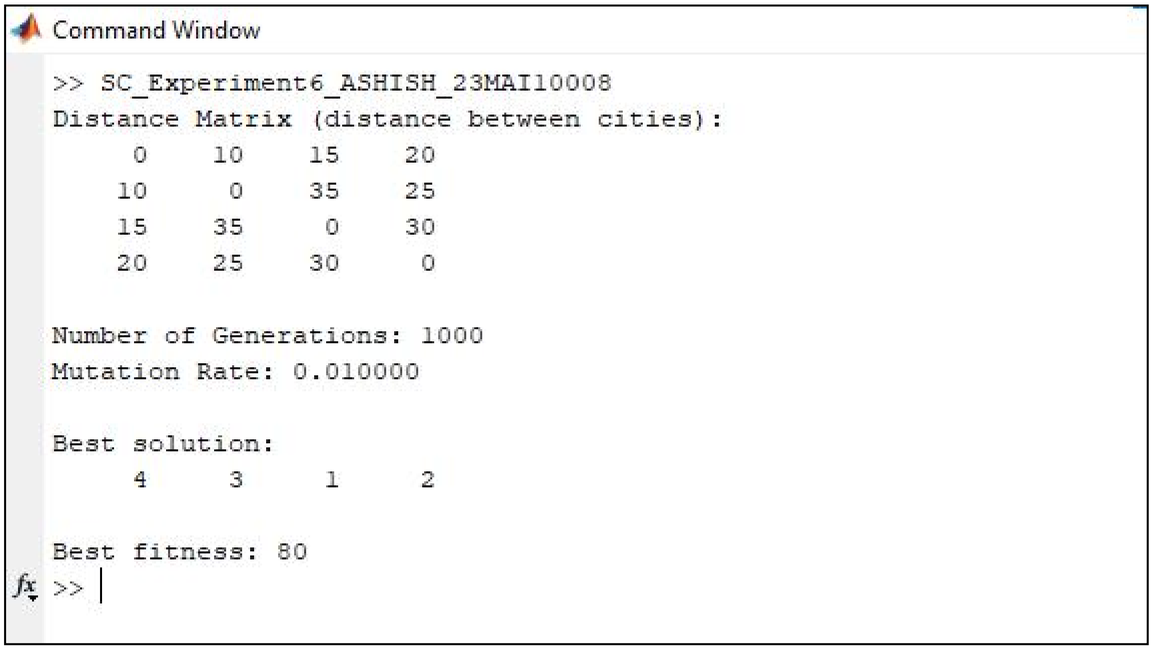


Code for Experiment :

```
% Define the distance matrix (distance between cities)
% d(i, j) represents the distance from city i to city j
d = [0, 10, 15, 20;
     10, 0, 35, 25;
     15, 35, 0, 30;
     20, 25, 30, 0];
fprintf("Distance Matrix (distance between cities): \n")
disp(d)

% Number of cities
num_cities = size(d, 1);

% Genetic Algorithm Parameters
population_size = 50;
num_generations = 1000;
mutation_rate = 0.01;

% Generate initial population
population = zeros(population_size, num_cities);
for i = 1:population_size
      population(i, :) = randperm(num_cities);
end

fprintf("Number of Generations: %d\n",num_generations)
fprintf("Mutation Rate: %f\n\n",mutation_rate)

% Evaluate fitness for each individual in the population
fitness = zeros(population_size, 1);
for i = 1:population_size
      fitness(i) = evaluate_fitness(population(i, :), d);
end

% Main loop
for generation = 1:num_generations
      % Selection: Roulette Wheel Selection
      probabilities = 1 ./ fitness;
      probabilities = probabilities / sum(probabilities);
      selected_indices = randsample(1:population_size, population_size, true,
       probabilities);

      % Crossover: Ordered Crossover (OX)
      new_population = zeros(population_size, num_cities);
      for i = 1:2:population_size
            parent1 = population(selected_indices(i), :);
            parent2 = population(selected_indices(i+1), :);
            [offspring1, offspring2] = crossover(parent1, parent2);
            new_population(i, :) = offspring1;
            new_population(i+1, :) = offspring2;
      end
```

```
        % Mutation: Swap Mutation
        for i = 1:population_size
            if rand < mutation_rate
                idx1 = randi(num_cities);
                idx2 = randi(num_cities);
                new_population(i, [idx1, idx2]) = new_population(i, [idx2, idx1]);
            end
        end

        population = new_population;

        % Update fitness
        for i = 1:population_size
            fitness(i) = evaluate_fitness(population(i, :), d);
        end
    end


    % Find the best solution
    [best_fitness, idx] = min(fitness);
    best_individual = population(idx, :);

    disp('Best solution:');
    disp(best_individual);
    disp(['Best fitness: ', num2str(best_fitness)]);


    % Function to evaluate fitness (total distance)
    function total_distance = evaluate_fitness(individual, distance_matrix)
        total_distance = 0;
        n = length(individual);

        for i = 1:n-1
            city1 = individual(i);
            city2 = individual(i+1);

            % Check if indices are within bounds
            if city1 > 0 && city1 <= size(distance_matrix, 1) && ...
               city2 > 0 && city2 <= size(distance_matrix, 2)
                total_distance = total_distance + distance_matrix(city1, city2);
            else
                % Assign a high penalty for invalid indices
                total_distance = total_distance + 1e6; % Adjust the penalty as needed
            end

        end

        % Add distance from the last city back to the starting city
        city1 = individual(end);
        city2 = individual(1);
        % Check if indices are within bounds
        if city1 > 0 && city1 <= size(distance_matrix, 1) && ...
           city2 > 0 && city2 <= size(distance_matrix, 2)
              total_distance = total_distance + distance_matrix(city1, city2);
        else
              % Assign a high penalty for invalid indices
              total_distance = total_distance + 1e6; % Adjust the penalty as needed
        end
end


% Function for crossover: Ordered Crossover (OX)
function [offspring1, offspring2] = crossover(parent1, parent2)
        n = length(parent1);
        start = randi(n);
        stop = randi(n);
        if start > stop
              temp = start;
              start = stop;
              stop = temp;
        End

        mask = zeros(1, n);
        mask(start:stop) = 1;
        offspring1 = parent1 .* mask;
        offspring2 = parent2 .* mask;

        idx1 = find(~offspring1);
        idx2 = find(~offspring2);
        idx11 = 1;
        idx22 = 1;

        for i = 1:n
              if offspring1(i) == 0
                    if ~ismember(parent2(i), offspring1)
                          offspring1(idx1(idx11)) = parent2(i);
                          idx11 = idx11 + 1;
                    end
              End

              if offspring2(i) == 0
                    if ~ismember(parent1(i), offspring2)
                          offspring2(idx2(idx22)) = parent1(i);
                          idx22 = idx22 + 1;
                    end
              end
        end
end
```

Result/Output :



```
>> SC_Experiment6_ASHISH_23MAI10008
Distance Matrix (distance between cities):
     0    10    15    20
    10     0    35    25
    15    35     0    30
    20    25    30     0

Number of Generations: 1000
Mutation Rate: 0.010000

Best solution:
     4     3     1     2

Best fitness: 80
>> 
```

Learning outcomes :

1. Learnt about the concept of Genetic Algorithm.
2. Learnt about different phases of Genetic Algorithm.
3. Learnt about Crossover and Mutation methods in Genetic Algorithm.
4. Learnt about how to solve the Travelling Salesman problem.
5. Learnt about different methods of Selection in Genetic Algorithm.