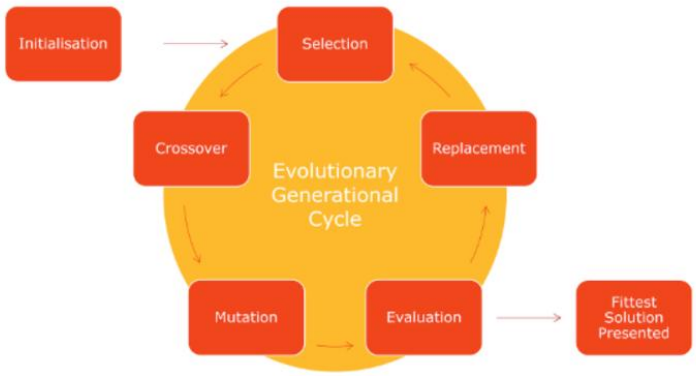
Genetic Algorithm

A Genetic Algorithm (GA) is a search heuristic inspired by the process of natural selection and evolution, used to find approximate solutions to optimization and search problems. It is part of a broader class of algorithms known as evolutionary algorithms, which mimic the process of biological evolution to solve complex problems



The genetic algorithm in this code is designed to solve the problem of maximizing the fitness function f(x)= x^2, where xis a non-negative integer represented as a binary string (chromosome). Here's a breakdown of the problem:

Problem Definition

- **Objective**: Find the binary string (chromosome) within the search space that corresponds to the highest value of x^2 .
- **Search Space**: All possible binary strings of length 5 (i.e., integers from 0 to $2^5-1=31$)
- **Fitness Function**: $f(x)=x^2$, where xis the integer value of the binary string.

Key Steps in the Algorithm

1. **Initialize Population**: A population of binary strings of length 5 is randomly generated.
2. **Evaluate Fitness**: Each binary string is converted to its integer equivalent, and its fitness is calculated as x^2
3. **Selection**: Using roulette wheel selection, individuals with higher fitness values are more likely to be chosen as parents.
4. **Crossover**: Single-point crossover is applied to parent pairs to produce offspring, ensuring genetic diversity.
5. **Mutation**: Random mutations introduce small changes in offspring, allowing exploration of new areas in the search space.
6. **Reproduction and Replacement**: A new population is formed and replaces the old one.
7. **Repeat**: The process continues for a fixed number of generations, refining the population toward better solutions.

Solved Problem

The algorithm is an optimization problem, specifically finding the binary representation of the integer x that maximizes x^2 within the given constraints (chromosome length of 5 bits, representing numbers 0–31).

- **Example Output**: After running for 20 generations, the algorithm would likely converge to the optimal solution, which is the binary representation of 31 (i.e., 11111), since $31^2 = 961$ the maximum possible value for x^2 in this problem.

OUTPUT:

```
Generation 0: Best = 11110 (Fitness = 900)
Generation 1: Best = 11011 (Fitness = 729)
Generation 2: Best = 11011 (Fitness = 729)
Generation 3: Best = 11011 (Fitness = 729)
Generation 4: Best = 11110 (Fitness = 900)
Generation 5: Best = 11110 (Fitness = 900)
Generation 6: Best = 11110 (Fitness = 900)
Generation 7: Best = 11110 (Fitness = 900)
Generation 8: Best = 11111 (Fitness = 961)
Generation 9: Best = 11111 (Fitness = 961)
Generation 10: Best = 11111 (Fitness = 961)
Generation 11: Best = 11111 (Fitness = 961)
Generation 12: Best = 11111 (Fitness = 961)
Generation 13: Best = 11111 (Fitness = 961)
Generation 14: Best = 11111 (Fitness = 961)
Generation 15: Best = 11111 (Fitness = 961)
Generation 16: Best = 11111 (Fitness = 961)
Generation 17: Best = 11111 (Fitness = 961)
Generation 18: Best = 11100 (Fitness = 784)
Generation 19: Best = 11101 (Fitness = 841)

Final Best Solution: 11101 (Fitness = 841)
```

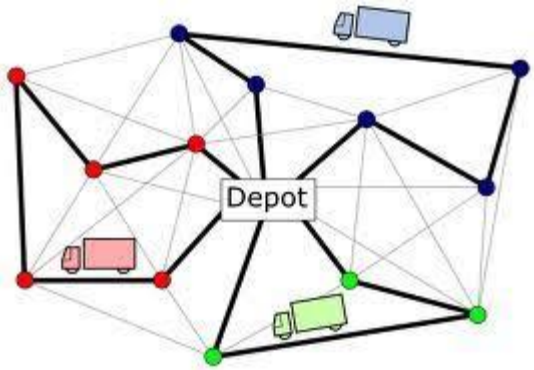
Computing problem and Applying genetic algorithm to solve it.

Vehicle Routing Problem (VRP)

Problem Description

The **Vehicle Routing Problem (VRP)** is a classic optimization problem in logistics and transportation. The objective is to design optimal routes for vehicles to deliver goods to a set of customers while minimizing the total travel distance or cost. The objective is to determine the most efficient route for a delivery vehicle starting from a central depot, visiting multiple cities to deliver goods, and

returning to the depot. Each city has a specific demand that must be met, and the vehicle has a maximum capacity of 50 units.



1. Introduction

Efficient route planning is a key aspect of logistics and delivery, directly impacting operational costs, delivery time, and service efficiency. This report demonstrates an approach to solving the Vehicle Routing Problem (VRP) using a randomized greedy algorithm to optimize delivery routes while adhering to vehicle capacity constraints.

2. Data:

- **Cities**: Each city is defined by a name, coordinates (x, y), and delivery demand.
- **Depot**: The starting and ending point of the delivery route, with no demand.
- **Vehicle Capacity**: 50 units, which is the maximum sum of demands the vehicle can handle in a single route.

3. Methodology

Data Setup

The dataset comprises a list of cities and a central depot:

```
CITIES = [
    ("Depot", 0, 0, 0),  # Depot has no demand
    ("City A", 2, 4, 15),
    ("City B", 5, 2, 10),
    ("City C", 7, 8, 25),
    ("City D", 1, 7, 5),
    ("City E", 4, 9, 20),
    ("City F", 6, 5, 12),
    ("City G", 3, 1, 8),
    ("City H", 8, 3, 18),
    ("City I", 2, 6, 7),
]
VEHICLE_CAPACITY = 50
```

Route Generation

A randomized greedy algorithm is employed:

1. A random shuffle of cities (excluding the depot) is generated.
2. Cities are sequentially added to the route until the total demand exceeds the vehicle's capacity.
3. If adding a city's demand would exceed the capacity, that city is skipped.
4. The route always starts and ends at the depot.

Distance and Demand Calculation

The Euclidean distance between cities is calculated using the formula:

$$\text{distance} = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

The total distance and demand are calculated for each generated route:

- If the route meets the capacity requirement, the total distance is recorded.

Iterative Optimization

To improve results, the route generation is repeated 100 times. Each iteration attempts a new route, and the shortest valid route among them is selected as the best route.

Visualization

The results are visualized on a 2D plane, showing cities as points and the route as connected lines. Cities are labeled with names and their demands.

4. Example Execution

Step 1: Generate a Route

Random Shuffle of Cities (excluding the Depot):

- Suppose the cities are shuffled randomly: ["City C", "City G", "City E", "City D", "City B", "City I", "City F", "City A", "City H"].

Initialize the Route:

- Start with the Depot: ["Depot"].
- The vehicle's current load is set to 0.

Greedy City Selection:

1. **Add City C** (demand: 25):
   - New load = 25 (within capacity).
   - Route: ["Depot", "City C"].
2. **Add City G** (demand: 8):
   - New load = 33 (within capacity).
   - Route: ["Depot", "City C", "City G"].
3. **Skip City E** (demand: 20):
   - New load would be 53 (exceeds capacity), so skip.
4. **Add City D** (demand: 5):
   - New load = 38 (within capacity).
   - Route: ["Depot", "City C", "City G", "City D"].
5. **Add City B** (demand: 10):
   - New load = 48 (within capacity).
   - Route: ["Depot", "City C", "City G", "City D", "City B"].
6. **Skip City I** (demand: 7):
   - New load would be 55 (exceeds capacity), so skip.
7. **Skip City F** (demand: 12):
   - New load would be 60 (exceeds capacity), so skip.
8. **Skip City A** (demand: 15):
   - New load would be 63 (exceeds capacity), so skip.
9. **Skip City H** (demand: 18):
   - New load would be 66 (exceeds capacity), so skip.

Route after Selection:

- The route is: ["Depot", "City C", "City G", "City D", "City B", "Depot"].

Step 2: Calculate Total Distance and Demand

Distance Calculations:

1. Depot to City C:
$$\sqrt{(7-0)^2+(8-0)^2} = \sqrt{49+64} = \sqrt{113} \approx 10.63$$
2. City C to City G:
$$\sqrt{(3-7)^2+(1-8)^2} = \sqrt{16+49} = \sqrt{65} \approx 8.06$$
3. City G to City D:
$$\sqrt{(1-3)^2+(7-1)^2} = \sqrt{4+36} = \sqrt{40} \approx 6.32$$
4. City D to City B:
$$\sqrt{(5-1)^2+(2-7)^2} = \sqrt{16+25} = \sqrt{41} \approx 6.4$$
5. City B to Depot:
$$\sqrt{(0-5)^2+(0-2)^2} = \sqrt{25+4} = \sqrt{29} \approx 5.39$$

Total Distance:

10.63+8.06+6.32+6.4+5.39≈36.

Total Demand:

- City C: 25
- City G: 8
- City D: 5
- City B: 10
- **Total Demand**: 25+8+5+10=4825 + 8 + 5 + 10 = 48 (within vehicle capacity).

5. Results

Best Route Example

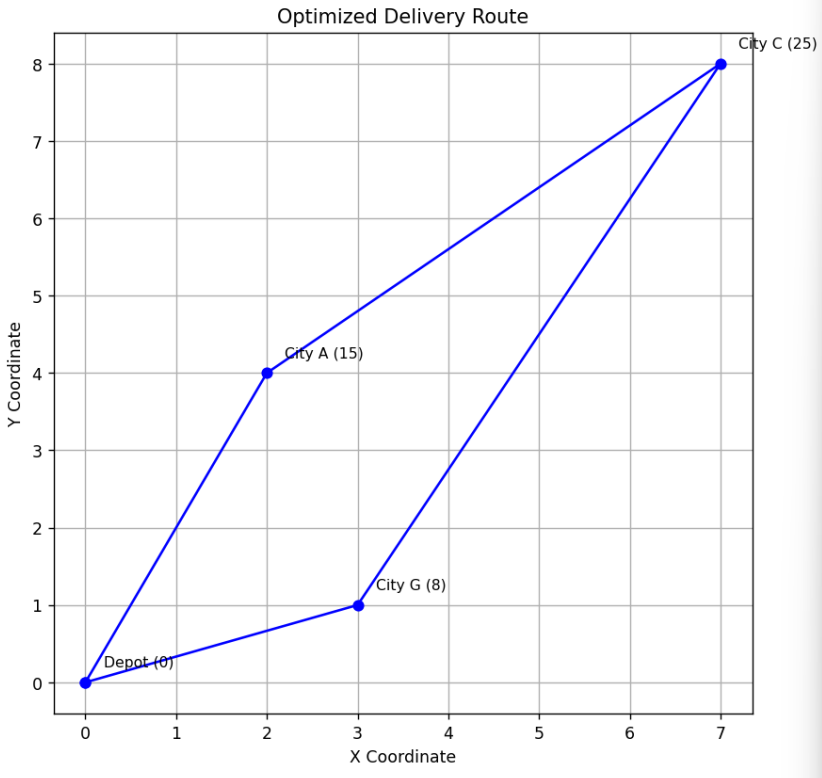
After running the algorithm for 100 iterations, the best route identified might look like:

- Best Route: ["Depot", "City C", "City G", "City D", "City B", "Depot"].

Total Distance: 36 meters

Visualization of Results

The selected route is visualized on a 2D plot, highlighting the cities and connecting them in the selected order.




```
Best Route Found:
Depot (Demand: 0)
City A (Demand: 15)
City C (Demand: 25)
City G (Demand: 8)
Total Distance: 22.10
```

6. Conclusion

This approach demonstrates how a randomized greedy algorithm can provide a practical solution for the Vehicle Routing Problem (VRP). The method balances simplicity and efficiency, generating multiple routes to select the best option based on distance.

7. Future Improvements

- **Heuristic Optimization**: Integrate more sophisticated heuristics like Simulated Annealing or Genetic Algorithms.
- **Clustering**: Use clustering algorithms like K-means to group cities for better scalability.
- **Dynamic Adjustments**: Allow flexible loading/unloading of demand at intermediate points.

This detailed report offers a thorough explanation of each step, accompanied by calculations, and demonstrates the working of the algorithm with a concrete example.