Logistics Optimization

Abstract:

Developed a web application for solving the Capacitated Vehicle Routing Problem (CVRP) using a genetic algorithm (GA).Integrated Flask for the backend API development and React for the frontend interface.Utilized crossover and swap mutation operators to evolve solutions efficiently.Demonstrated the effectiveness of the solution through performance metrics and user interface evaluation.

Introduction:

- CVRP is a significant problem in logistics optimization, involving routing vehicles to serve customer demands while respecting capacity constraints.
- Genetic algorithms offer an effective approach for solving combinatorial optimization problems like CVRP.
- The project aimed to develop a web application that enables users to input CVRP instances, visualize solutions, and analyze results.

Literature Review:

- Reviewed existing research on CVRP solutions, genetic algorithms, and web application development.
- Identified crossover and swap mutation as familiar genetic algorithm operators for CVRP optimization.
- Explored Flask as a lightweight Python web framework and React as a popular JavaScript library for building user interfaces.

Methodology:

- In CVRP, each chromosome represents a potential routing plan for the vehicles. It consists of a sequence of genes, each representing a customer or a depot (starting/ending point).
- The chromosome encodes the order in which the vehicles visit the customers, ensuring that each customer is visited exactly once and the cars return to the depot within their capacity constraints.
- Utilized Flask for backend API development, exposing endpoints for receiving CVRP instances, running the genetic algorithm, and returning optimized solutions.
- Integrated React for the frontend interface, allowing users to interact with the application through an intuitive and responsive web interface.

Mathematical Constraints of CVRP:

- **Capacity Constraint**: Each vehicle's total demand served cannot exceed its capacity.
- **Route Connectivity**: Each customer must be visited exactly once.
- **Flow Conservation**: Ensures that vehicles depart from and return to the depot, and intermediate nodes are visited only once.

```
Capacity Constraint:
Σ(q_j * x_ij) ≤ Q_i for all i (vehicles)

Route Connectivity:
Σx_ij = 1 for all j (customers)

Flow Conservation:
Σx_ij = 2 for all i (vehicles)
Σx_ij = 1 for all j (customers)
```

Use Cases for CVRP:

- **Logistics and Distribution**: Optimizing delivery routes for transportation companies to minimize fuel costs and vehicle usage while meeting customer demands.
- **Supply Chain Management**: Efficiently managing inventory replenishment and distribution across multiple warehouses and retail locations.
- **Waste Collection and Recycling**: Optimizing waste collection routes for garbage trucks to minimize travel time and fuel consumption while servicing all collection points.
- **Healthcare Services**: Scheduling home healthcare visits or medical supply deliveries to patients while ensuring timely and efficient service.

Implementation:

- Set up the development environment using Python for backend development with Flask and JavaScript for frontend development with React.
- Implemented crossover and swap mutation operators to effectively generate diverse solutions and explore the solution space.
- Designed a user-friendly interface using React components to input CVRP instances, visualize routes on a map, and display optimization results.
- Analyzed the impact of front-end design choices on user experience and engagement.
- Explored potential enhancements, such as incorporating additional optimization techniques or integrating real-time data for dynamic routing.

Challenges:

- The code has been modified to include a validation check after generating offspring. If the offspring is found to be invalid, it is rejected.
- Similarly, only valid random solutions are included in the initial population during the generation of initial solutions.

```python
def is_valid_solution(solution):
    # Check if each customer is visited by exactly one vehicle
    for routes in solution:
        all_customers = set(range(len(coords)))
        for route in routes:
            all_customers -= set(route)
    # Check if the total demand of each vehicle does not exceed its capacity
    if all_customers:
        return False
    # Check if all customers are covered
    return True
```

```
11.806172446117568
```

```python
def initial_population(num_vehicles, vehicle_capacities, users):
    population = []
    shuffled_users = list(users)
    random.shuffle(shuffled_users)
    offSpring=[]
    for user in shuffled_users:
        shuffled_vehicle = []
        for i,v in vehicle_capacity:
          shuffled_vehicle.append((i,v))

        random.shuffle(shuffled_vehicle)

        assigned = False

        for i, (ind,capacity) in enumerate(shuffled_vehicle):
            if capacity >= user:
                if i < len(population):
                    if sum(population[i]) + user <= capacity:
                        population[i].append(user)
                        assigned = True
                        break
                else:
                    population.append([user])
                    assigned = True
                    break

        if not assigned:
            # If no vehicle can accommodate the user, skip adding this user
            continue

    return population
```

Results:

- We ran a genetic algorithm for various generation values. Then, we found that it performs best in the range of 30 - 40 generations, which takes an average of 4-5 seconds to display the solution.
- For this experiment, we took all other parameters to standard constants :

  Mutations rate = 0.1

  Initial population = 15

  No of vehicles = 2

  Capacity of vehicles = 2,4

Hyperparameters Initial Population Size - 15

No of users	No of vehicles	Capacities	No of generation	Genetic Algorithm Fitness	Actual Fitness	execTime	Accuracy
4	2	2,4	5	8.10223414242567	10.30859375000001	~1sec	0.786
4	2	2,4	10	9.21415133914153	10.30859375000001	~2sec	0.894
4	2	2,4	100	9.88522147283254	10.30859375000001	~10sec	0.959

10. References:

- https://www.researchgate.net/publication/274718381_A_study_of_performance_on_crossover_and_mutation_operators_for_vehicle_routing_problem
- https://iopscience.iop.org/article/10.1088/1757-899X/1071/1/012025
- https://legacy.reactjs.org/docs/getting-started.htmlhttps://legacy.reactjs.org/docs/getting-started.html

Useful Links:

https://github.com/Devendra137/Delivery-Optimization

Video Link - https://drive.google.com/drive/folders/1dNtl9LxWrs6WcmvRtBwm63CoglAa5LTB?usp=sharing