Travel Route Planner

Name: Jiamin Yao, Yan Pang, HangYu Yao
Professor: Dr.Rubi

CONTENTS

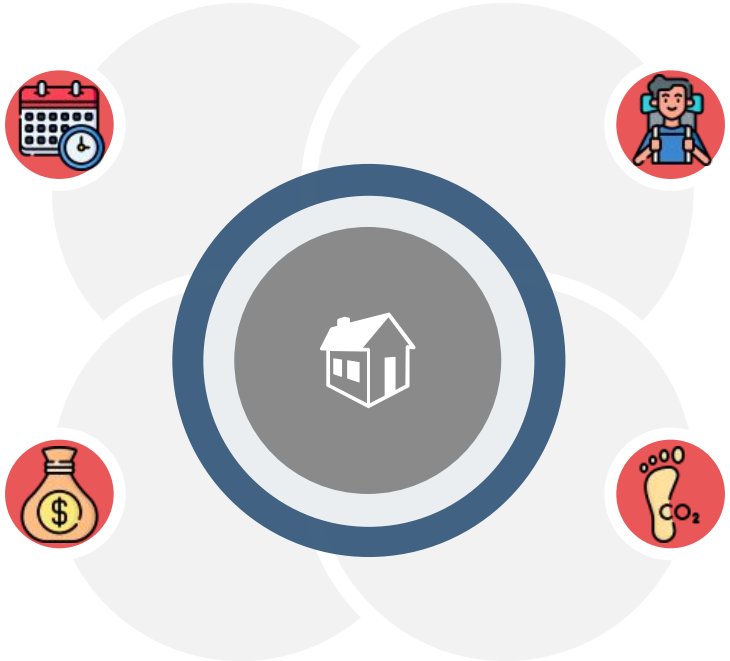
/01

Big Problem

Big Problem

Efficient Time Management

Optimizing the travel route can minimize delays and maximize the number of destinations visited.



Comfort and Experience

With better planning, travelers can focus more on enjoying their destinations rather than logistical hassles.

Cost Optimization

Carefully planned trips can help in avoiding expensive tolls, flights, or unnecessary bookings.

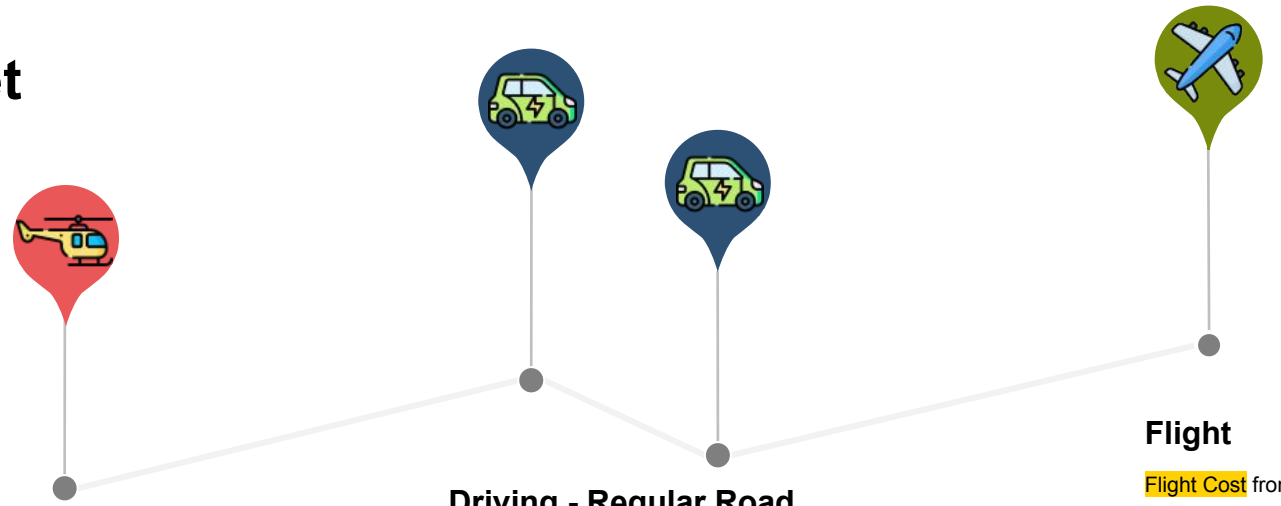
Sustainability

Efficient planning reduces unnecessary travel, lowering the carbon footprint

/02

Dataset

Dataset



Helicopter - Cities Coordinates

Total 18 cities selected in our project, they are represented by (x,y) coordinates. The Coordinates generated through user interaction with a map, where users click to select city points. The distance is calculated using the Euclidean distance formula:

$d(i,j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$

Driving - Regular Road

D_R = Regular Road Distance

S_R = Regular Road Speed Limits in Different Countries

F = Fuel Price Per Liter in Different Countries

RegularRoad Cost = $(D_R / S_R) * 6 * F$

RegularRoad Time = D_R / S_R

Driving - Highway

D_H = Highway Distance

S_H = Highway Speed Limits in Different Countries

T = Highway Tolls between Different Countries

Highway Cost = $(D_H / S_H) * 6 * F + T$

Highway Time = D_H / S_H

Flight

Flight Cost from Google Flights

Flight Duration from Google Flights

/03

Methodology

Block Diagram

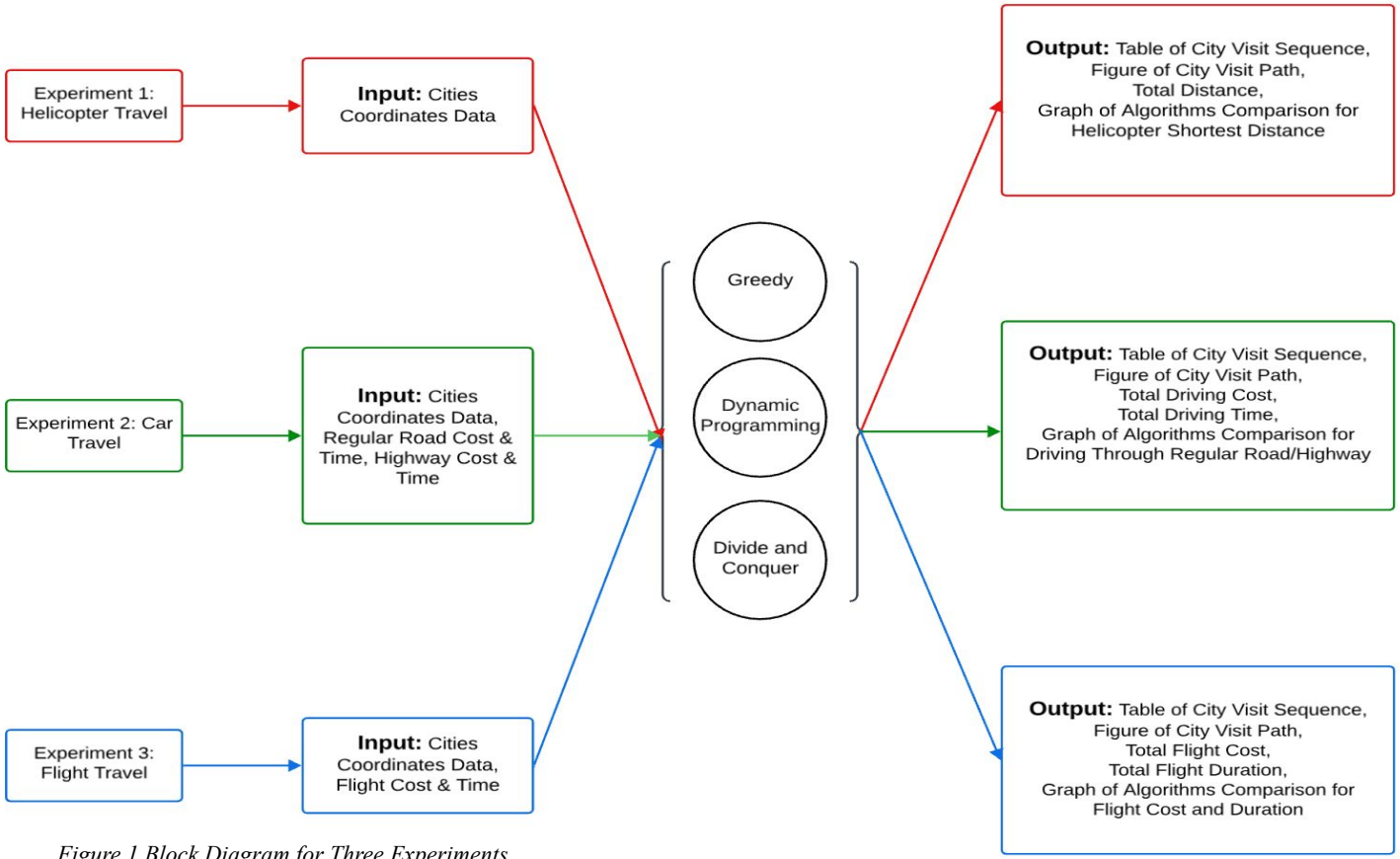
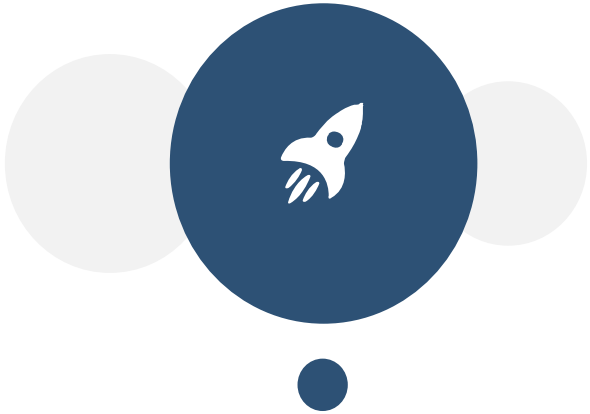


Figure.1 Block Diagram for Three Experiments

Methodology



Greedy

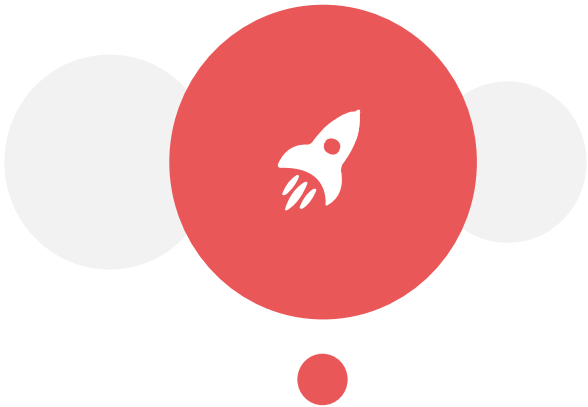
Pseudocode for Greedy Algorithm:
1. Start at the first city
2. Mark the starting city as visited
3. While there are unvisited cities:
4. Find the nearest/lowest price/shortest time unvisited city
5. Travel to that city
6. Mark the city as visited
7. Return the minimum distance/cost/time found

Table.1 Pseudocode for Greedy Algorithm

Time complexity: $O(n^2)$

Space Complexity is O(n)

Methodology



Divide and Conquer

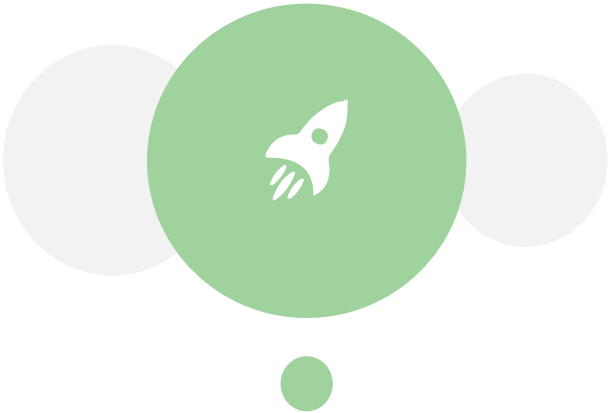
Pseudocode for Divide and Conquer Algorithm:
1. Divide:
2. Sort cities by y-coordinate
3. Split cities into two halves
4. Recursively divide until each subset has 2 or 3 cities
5. Conquer:
6. Use the brute force to find shortest/lowest price/shortest time path for small subsets
7. Combine:
8. Check for closer paths between the two halves
9. Return the minimum distance/cost/time found

Table.2 Pseudocode for Divide and Conquer Algorithm

Time complexity : O(nlogn)

Space complexity : O(n)

Methodology



Dynamic Programming

Pseudocode for Dynamic Programming Algorithm:
1. Initialize dp[mask][i], where the mask is the set of visited cities and i is the current city
2. Set base case: dp[1][0] = 0 (starting at city 0)
3. For each subset of cities:
4. For each city u:
5. For each unvisited city v:
6. Update dp[mask | (1 << v)][v]
7. Once all cities are visited, return to the starting city
8. Reconstruct the path by backtracking

Table.3 Pseudocode for Dynamic Programming Algorithm

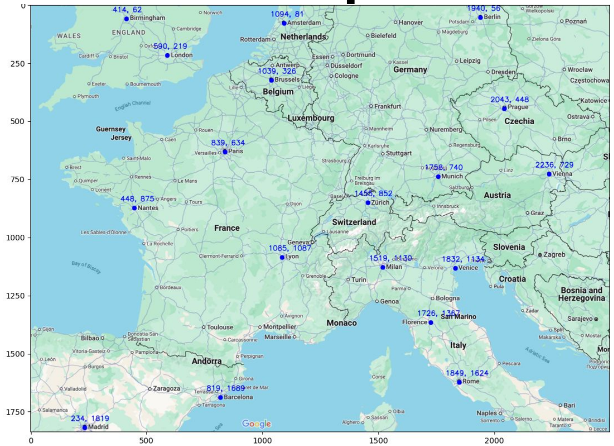
Time complexity: $O(n^2 \cdot 2^n)$

The space complexity: $O(n \cdot 2^n)$

/04

Experiments Design

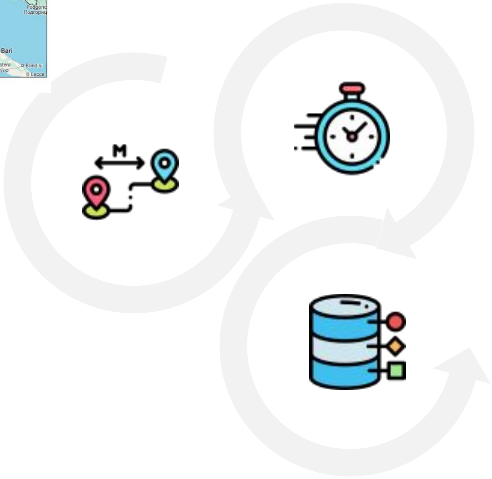
Ex1: Helicopter Shortest Distance Based on Coordinates



- ➔ Helicopter travel use straight-line distance between cities, so we can calculate the distance between cities using their coordinates. Distances -> Edges; Cities -> Nodes.
- ➔ Select a start city and find the shortest distance helicopter route for 18 cities that visits each city once using three algorithms.
- ➔ Vary the number of cities (from 6 to 18, increasing by 3) to compare performance for three algorithms and focuses on three key metrics:

Total Distance

Minimum km covered by each algorithm.



Time Complexity

The computational efficiency of each algorithm as the input size (number of cities) changes.

Space Complexity

The memory usage required by each algorithm to store and process city coordinates and distances.

Ex 2: Minimum Cost and Shortest Time for Driving Through Regular and Highway Roads

		Regular Road		Highway	
From	To	Cost ($)	Time (mins)	Cost ($)	Time (mins)
London	Paris	60.80	378	74.2	236
Milan	Vienna	38.44	204	53.82	133
Barcelona	Munich	178.25	1193	212.10	632
Berlin	Prague	48.91	291	49.54	202
Rome	Zurich	135.41	757	153.59	469
Amsterdam	Florence	228.70	1252	207.15	728

Table.5 Data Sample for Driving Cost and Time for Regular Road and Highway

- ➔ Select a start city and find the most cost-effective and time-efficient driving routes for 18 cities through regular road or highway that visits each city once using three algorithms.
- ➔ Vary the number of cities (from 6 to 18, increasing by 3) to compare performance for three algorithms and focuses on four key metrics:

Total Driving Cost

The sum of driving costs in dollars for the selected route.

Total Driving Time

The time in mins required to complete the route based on road speeds.

Time Complexity

The computational efficiency of each algorithm as the input size (number of cities) changes.

Space Complexity

The memory usage required by each algorithm to store and process driving cost and time.

Ex3: Minimum Cost and Shortest Time for Flights Between Cities

From	To	Cost ($)	Time (mins)
London	Paris	83	85
Milan	Vienna	129	210
Barcelona	Munich	157	125
Berlin	Prague	108	210
Rome	Zurich	114	95
Amsterdam	Florence	315	120

Table.6 Data Sample for Flight Cost and Duration

- Select a start city and find the most cost-effective and time-efficient flight routes for 18 cites that visits each city once using three algorithms.
- Vary the number of cities (from 6 to 18, increasing by 3) to compare performance for three algorithms and focuses on four key metrics:

Total Flight Cost

The sum of flight costs in dollars for the selected route.

Total Flight Duration

The time in mins required to complete the selected flight routes.

Time Complexity

The computational efficiency of each algorithm as the input size (number of cities) changes.

Space Complexity

The memory usage required for storing flight cost and duration matrices

/05

Experiments Results

Ex1: Helicopter Shortest Distance Based on Coordinates

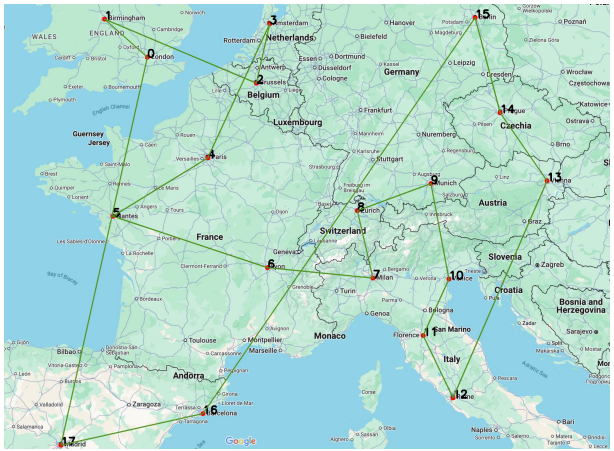


Figure.2 City Visit Path for Helicopter Shortest Distance using Greedy Algorithm

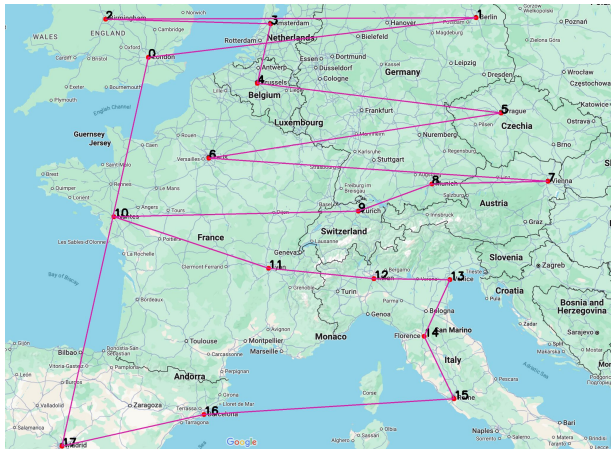


Figure.3 City Visit Path for Helicopter Shortest Distance using Divide and Conquer Algorithm

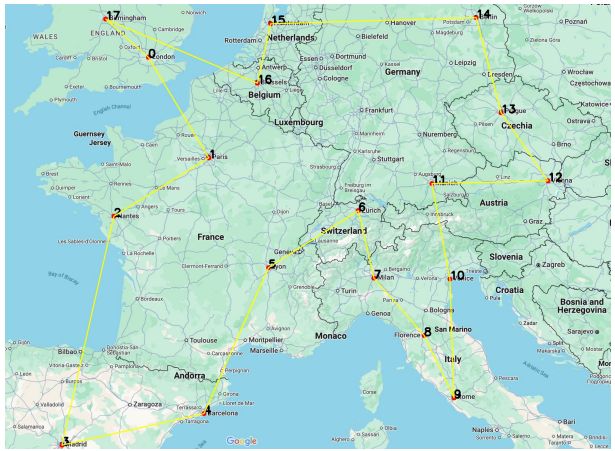


Figure.4 City Visit Path for Helicopter Shortest Distance using Dynamic Programming Algorithm

Ex1: Helicopter Shortest Distance Based on Coordinates

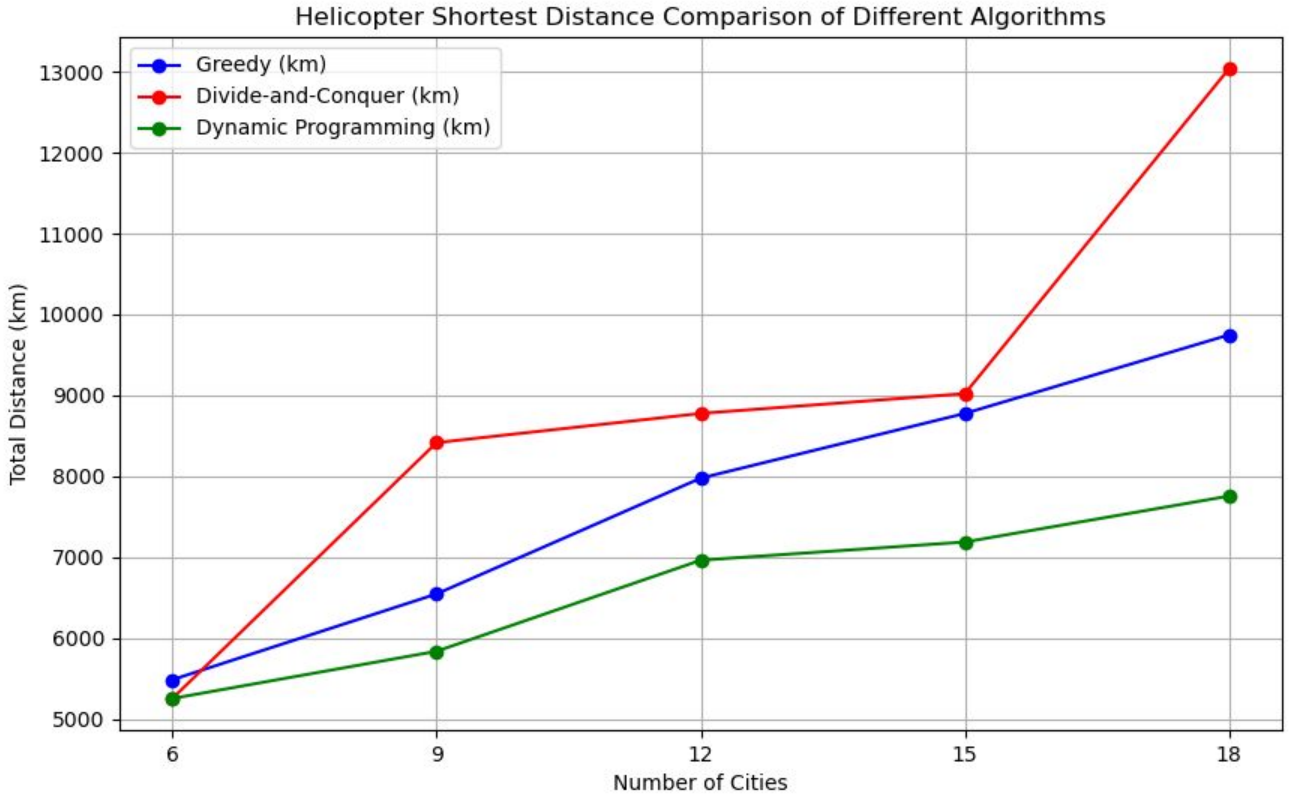


Figure.5 Helicopter Shortest Distance (km) Comparison of Three Algorithms

Ex 2: Minimum Cost and Shortest Time Paths for Driving Through Regular Roads

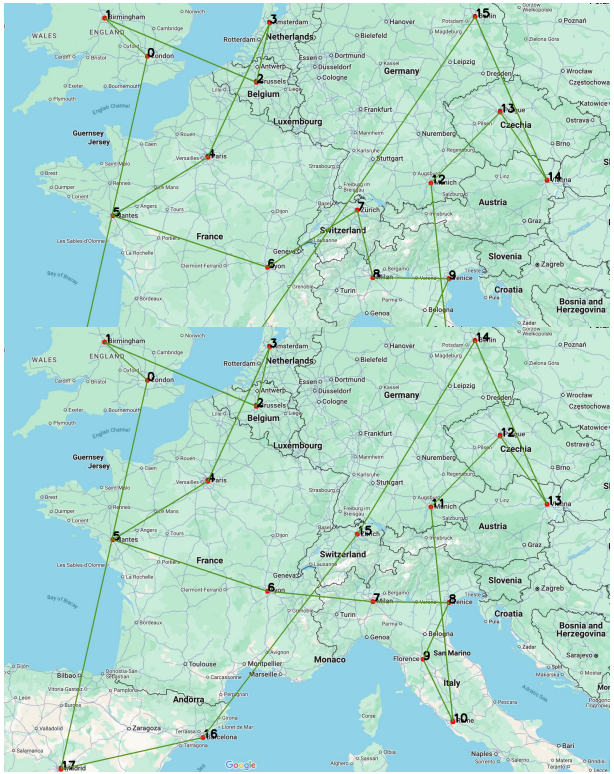


Figure.9 City Visit Path for Driving Min Time in Regular Road using Greedy Algorithm

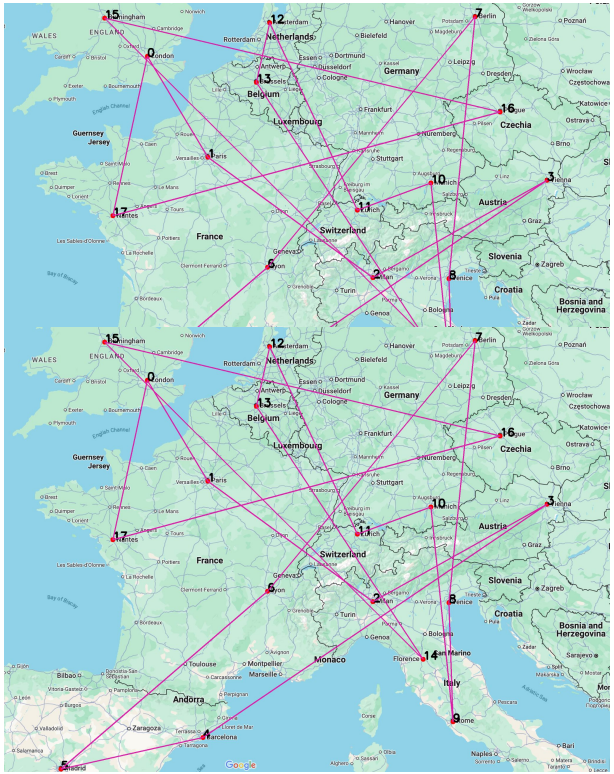


Figure.10 City Visit Path for Driving Min Time in Regular Road using Divide and Conquer Algorithm

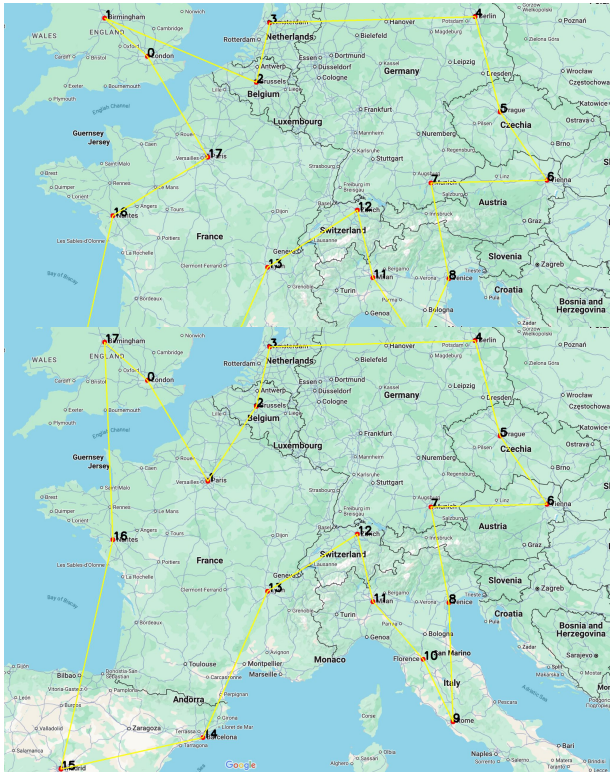


Figure.11 City Visit Path for Driving Min Time in Regular Road using Dynamic Programming Algorithm

Ex 2: Minimum Cost and Shortest Time Paths for Driving Through Highway Roads

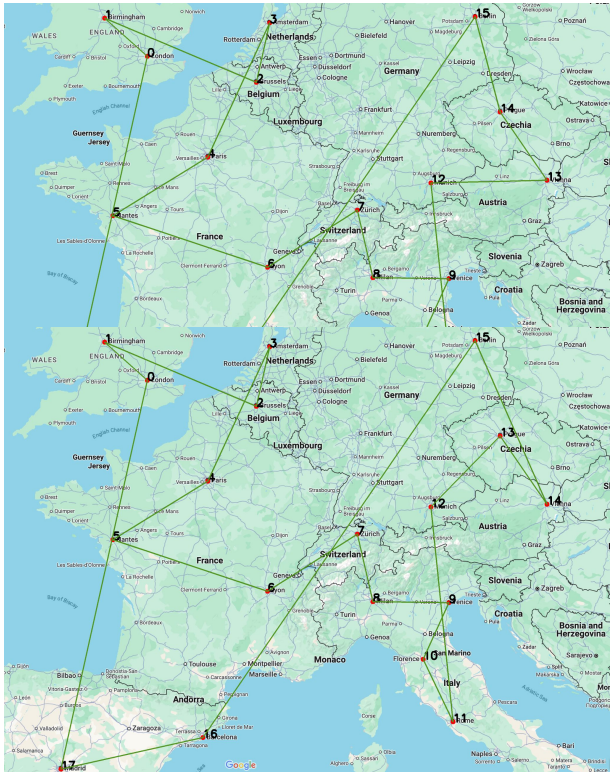


Figure.15 City Visit Path for Driving Min Time in Highway using Greedy Algorithm

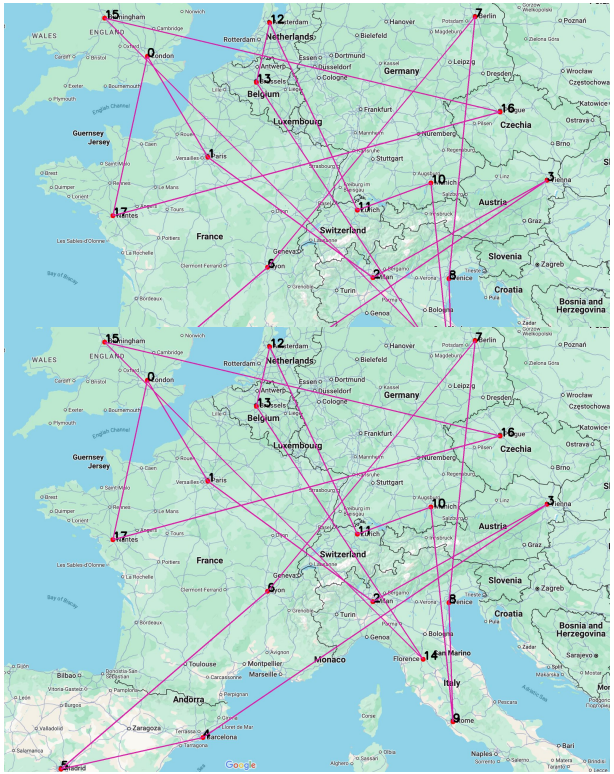


Figure.16 City Visit Path for Driving Min Time in Highway using Divide and Conquer Algorithm

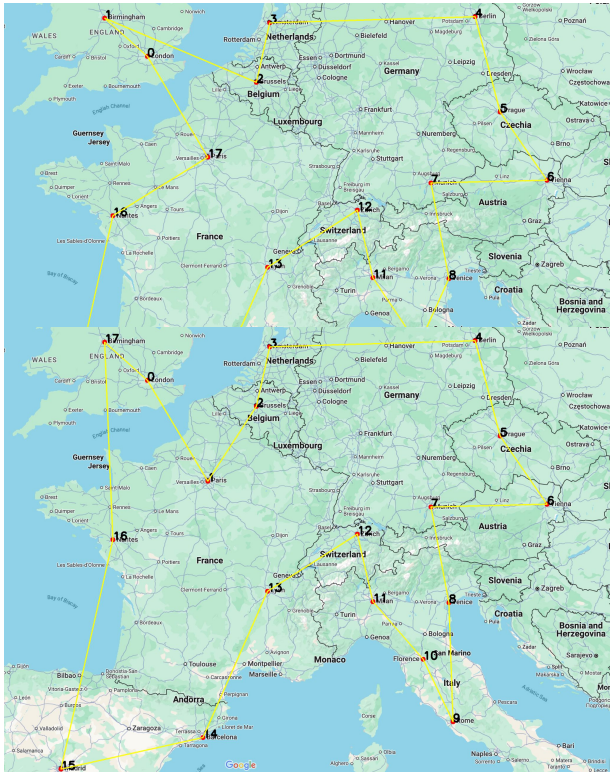


Figure.17 City Visit Path for Driving Min Time in Highway using Dynamic Programming Algorithm

Ex 2: Minimum Cost and Shortest Time for Driving Through Regular and Highway Roads

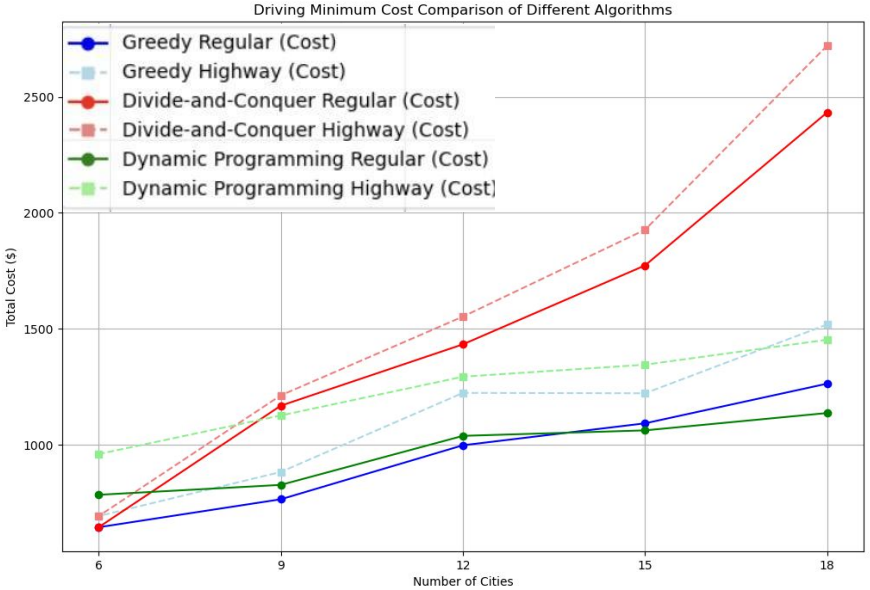


Figure.18 Driving Minimum Cost Comparison of Three Algorithms

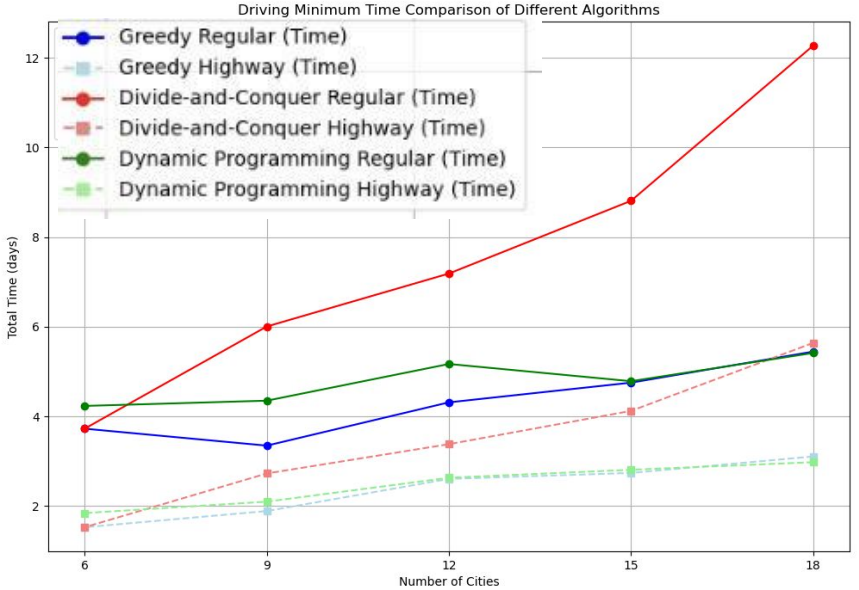


Figure.19 Driving Minimum Time Comparison of Three Algorithms

Ex3: Minimum Cost and Shortest Time for Flights Between Cities

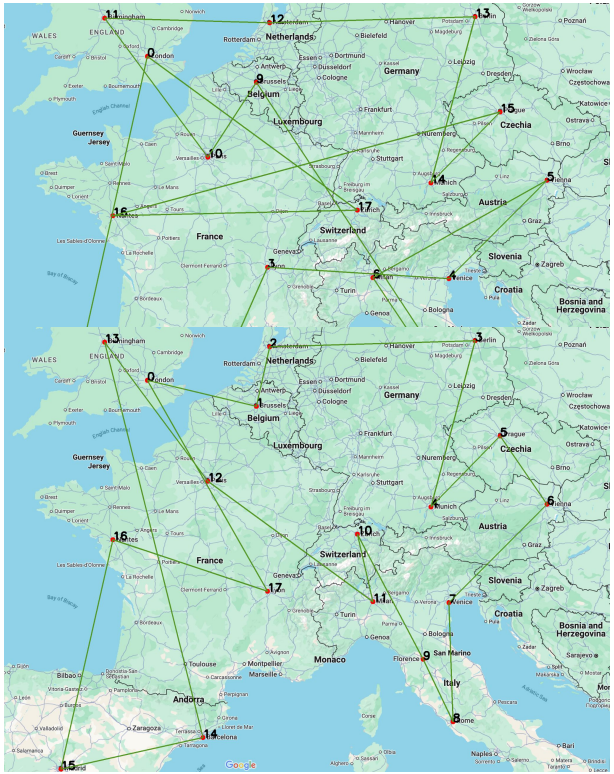


Figure.23 City Visit Path for Driving Min Time in Highway using Greedy Algorithm

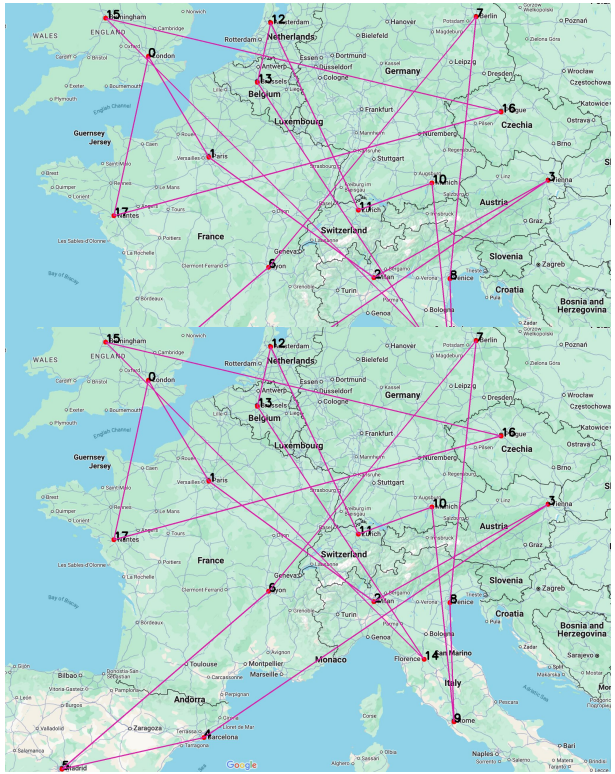


Figure.24 City Visit Path for Driving Min Time in Highway using Divide and Conquer Algorithm

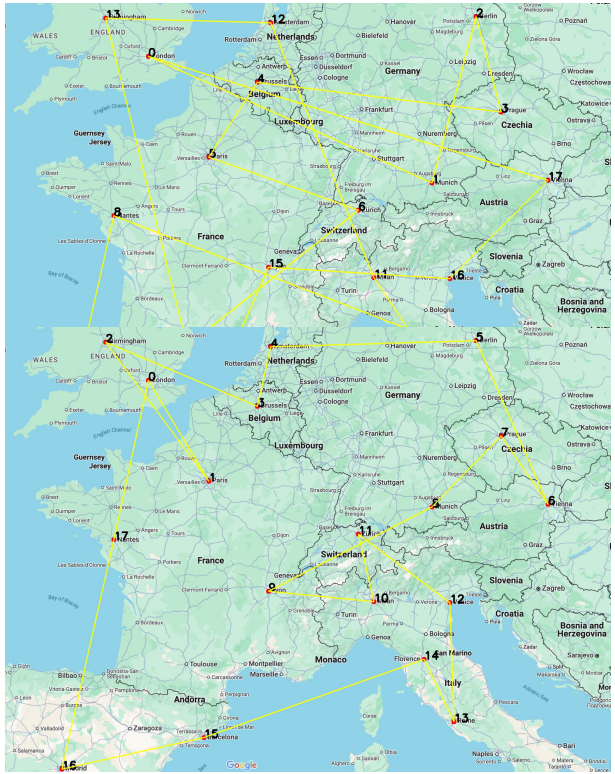


Figure.25 City Visit Path for Driving Min Time in Highway using Dynamic Programming Algorithm

Ex3: Minimum Cost and Shortest Time for Flights Between Cities

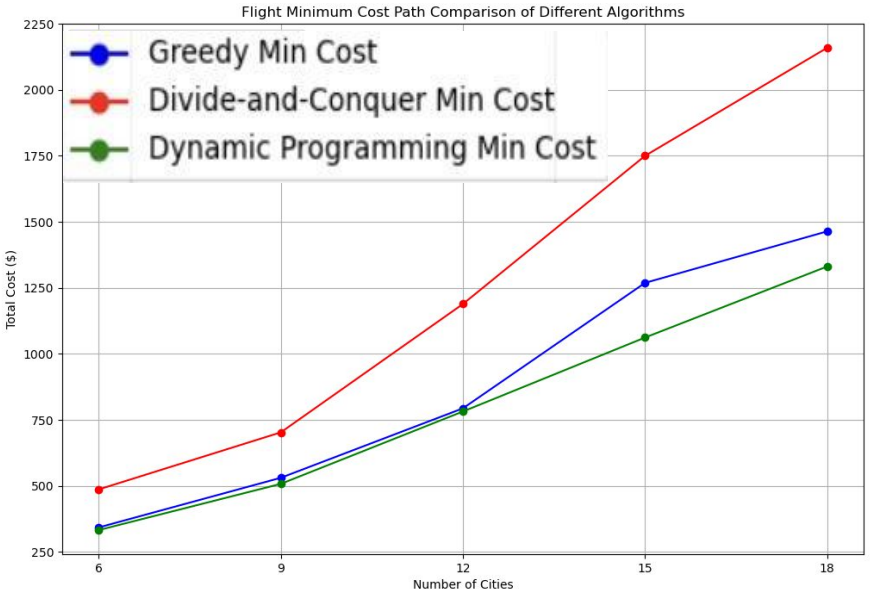


Figure.26 Flight Minimum Cost Comparison of Three Algorithms

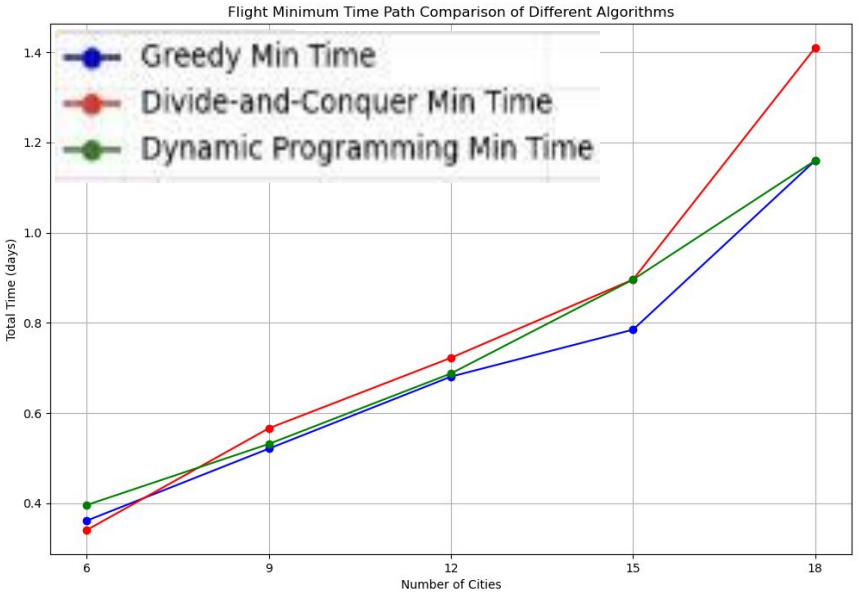
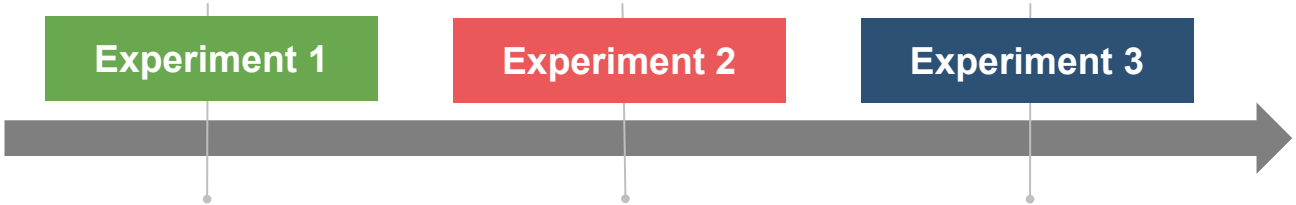


Figure.27 Flight Minimum Time Comparison of Three Algorithms

/06

Conclusion and Future Work

Conclusion



DP - Guarantees the optimal solution with the shortest distances but is computationally expensive $O(n^2 \times 2^n)$.

Greedy - Efficient and faster but produces suboptimal solutions.

DC - Performs the worst, with longer routes and less effective optimization.

DP - Provides globally optimized solutions, especially for larger datasets, but at a high computational cost.

Greedy - Efficient for smaller datasets but struggles to balance cost and time as the dataset grows.

DC - Despite its theoretical advantage in time complexity, it underperforms across all dataset sizes.

DP - Delivers the lowest costs and shortest times, consistently achieving globally optimal solutions.

Greedy - Fast and efficient for smaller datasets but becomes less competitive for larger datasets.

DC - Consistently underperforms, with higher costs and longer times compared to the other algorithms.

Dynamic Programming is the most robust algorithm, but it is computationally expensive.

Greedy offers a fast and simple/suboptimal solution.

Divide and Conquer is the least viable choice overall due to its inability to effectively optimize path selection.

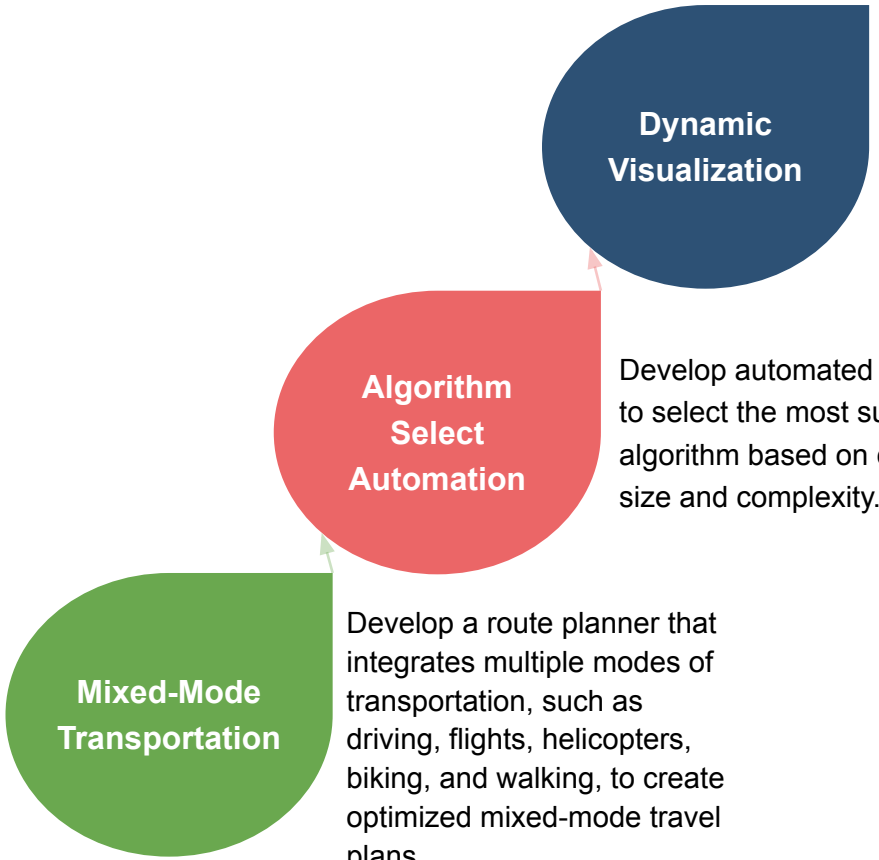
Challenge

Collecting real-world data is essential for meaningful comparisons and analysis. This takes time and requires access to accurate and diverse datasets.

Designing clear visualizations to represent city visit sequences and routes is challenging, especially for larger datasets and multiple optimization criteria.

Each scenario has unique optimization criteria and constraints, such as cost, time, and mode-specific factors, that influence algorithm performance differently.

Future Work


Dynamic Visualization
Implement dynamic visualization tools for route planning and real-time path adjustments.
Algorithm Select Automation
Develop automated systems to select the most suitable algorithm based on dataset size and complexity.
Mixed-Mode Transportation
Develop a route planner that integrates multiple modes of transportation, such as driving, flights, helicopters, biking, and walking, to create optimized mixed-mode travel plans.

Thanks

Any question?