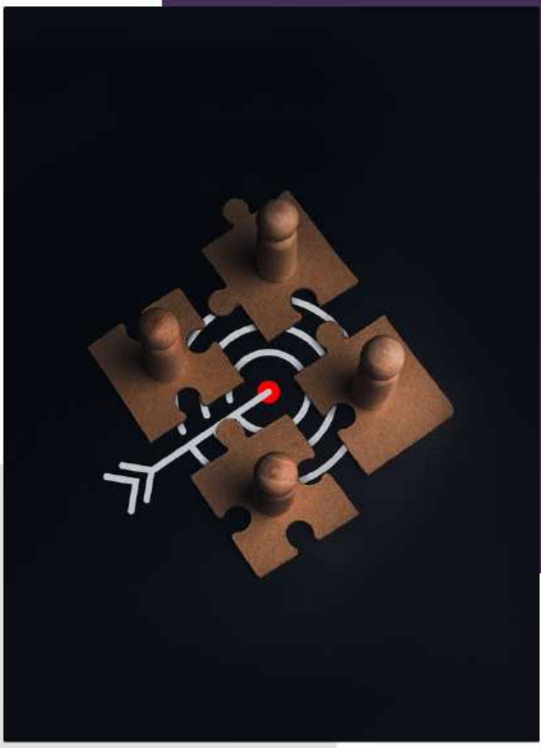


Optimizing Routes with **Real-World** Constraints

28 U.S. cities — November 1 to 30, 2025

Integrates elevation, weather risk, fuel, and daily drive limits to produce safety-aware, reproducible routes.

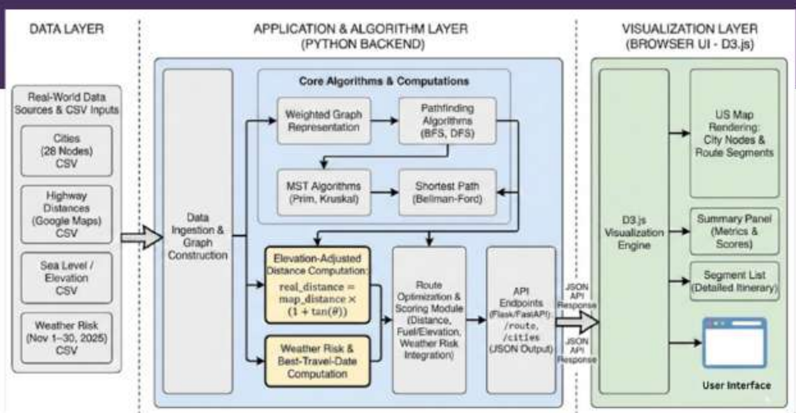
Acknowledging The Team

Project Contributors

- Amartya Mishra - Documentaion
- Mammai Sreeja - Data Collection
- Siddhi Mhasawade - Performance Testing
- Mekala Ruthvik - Python Code
- Patel Raaj - Research Report and Aechitecture Diagram
- Hirak Modi - Project Presentation
- Prasanth Nalluri - Python Routine Logic and Implementation

System Overview Diagram

High-level architecture showing components, data flows, and interactions



Route quality beyond distance

Weighted, elevation-aware routing for 28 cities

Datasets: Google Maps distances, sea-level differences, daily weather risk Nov 1–30, 2025 Distance, elevation and daily risk for 28 cities	**Fuel model: elevation tan(θ) penalty, consumption and daily driving limits** Elevation-adjusted fuel penalty and daily caps	**Algorithms: BFS, DFS, Prim, Kruskal, Bellman-Ford implemented in Python** Graph algorithms coded and tested in Python
Outputs: combined route-quality score, best route and best date selection Scores rank routes by distance, fuel, and risk	**Interfaces: HTTP API and D3.js visualizations** Programmatic access and interactive route visuals	**Key insight: context transforms what is 'best' beyond distance** Elevation and weather shift optimal route and date

Bridging the Gap Between Theory and Operational Routing

When terrain, fuel, and weather change the shortest path

1 **Gap: textbook shortest-paths assume static graphs; real routing must handle terrain, elevation, fuel, and weather**

Classical assumptions break down in operational settings

2 **Augmentation approach: incorporate elevation (sea-level differences) and temporal weather risk to enable multi-day plans and optimal departure date selection**

Model elevation and time-varying weather to extend routing scope

3 **Research question: how do classical algorithms behave when augmented with environmental and operational data?**

Evaluate algorithm performance after environmental augmentation

Data Collection: **Datasets** and Preprocessing

Three CSVs read into objects, adjacency lists, and date-indexed risk lookups

1 cities.csv — city id, name, state, sea-level elevation (meters)

Read with csv.DictReader into city objects

2 edges.csv — Google Maps one-way highway mileages

Built into adjacency lists for routing

3 weather_risk.csv — daily risk codes 1/5/10 for Nov 1–30, 2025

Loaded as date-indexed risk lookup for November 2025

4 Data hygiene: consistent ids, convert meters to miles, handle missing elevations and dates

Unit conversion and missing-data policies reduce errors

5 Why CSV: simple, transparent, reproducible

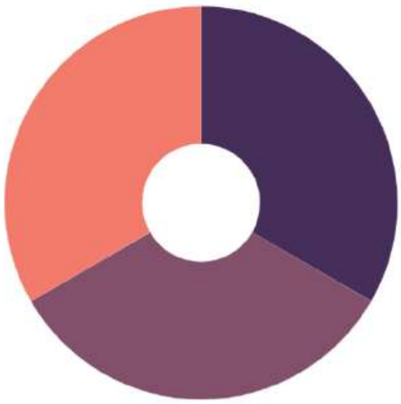
Easy to audit and version-control

System Design: Architecture and Components

Clear separation of backend, API, and D3.js frontend for maintainability

Backend - Python Graph Core

Graph class, pathfinding algorithms, bidirectional asymmetric elevation-adjusted edges; lazy risk evaluation computed on demand.



API Layer - Lightweight HTTP

Endpoints /cities and /route; thin adapter exposing algorithmic core without coupling visualization.

Frontend - D3.js Map UI

TopoJSON projection, interactive map and UI; consumes API and renders routes and risk overlays.

Algorithms Implemented: Roles and Trade-offs

Practical use cases, strengths, limitations, and qualitative complexity

Traversal and exploration

- BFS: level-order discovery; treats edges equally; strength: simple path finding in unweighted graphs; limitation: ignores weights; complexity: linear
- DFS: deep exploration; useful for topology, cycle detection; limitation: ignores weights and may be less predictable; complexity: linear
- Use case summary: topological exploration and reachability

Global optimization and weighted paths

- Prim and Kruskal: build minimum spanning trees for network backbone analysis; strength: global connectivity with minimal total weight; limitation: not optimal for point-to-point shortest paths; complexity: linear with log factors
- Bellman-Ford: computes weighted shortest paths and supports negative weights and dynamic edge weights; chosen as primary algorithm for BEST mode; limitation: slower on large graphs; complexity: higher-order
- Use case summary: MST for infrastructure, Bellman-Ford for weighted and dynamic routing

Travel Computation Models: Distance, Fuel, and Risk

Elevation, weather, daily limits, and a composite score for route decisions

- Map distance
  - Base segment length from map data used as the starting measurement
- Elevation adjustment
  - Effective distance = map_distance multiplied by 1 plus tan(theta); slope increases distance and fuel
- Fuel calculation
  - Use gasoline baseline 45 mpg to convert effective distance to fuel consumption
- Date-indexed weather
  - Weather per date mapped to risk: sunny/cloudy 1, rain 5, snow/ice 10
- Segment risk
  - Risk for a segment is the mean of its endpoint risks
- Accumulated route risk
  - Sum segment risks across route and across days for multi-day trips
- Daily limits
  - Driver limit 8 hours per day, max 75 mph yields about 600 miles per day
- Composite score
  - Score = distance plus 20 times risk to balance safety against distance

Route Study: Chicago to Dallas - Algorithm Comparison

Preserved routes and metrics for decision-ready planning

Algorithms

- Bellman-Ford (BEST)
- BFS
- DFS
- Prim
- Kruskal

Routes

- CHI to STL to SPR to TUL to OKC to DAL
- BFS route preserved
- DFS route preserved
- Prim route preserved
- Kruskal route preserved

Key Metrics

- Bellman-Ford distance 985.98 mi, gas 21.91 gal, risk 5.00, best date 11/11/2025
- DFS distance 2504.98 mi, gas 55.67 gal, risk 12
- BFS totals preserved
- Prim totals preserved
- Kruskal totals preserved

Performance Monitoring Findings for Linux

Key KPIs and scalability interpretation

Cities loaded: 28

Backend initialized 28 city records

Approximate edges: ~39

Graph contains about 39 edges

CPU usage: 3–8% during runs

Low CPU consumption across tests

Python memory: 50–70 MB

Process RSS around 50 to 70 megabytes

Swap activity: 0 observed

No swapping detected by vmstat

Free RAM: ~330+ MB

Several hundred MB free during runs

Disk I/O: minimal after initial CSV load

iostat shows negligible ongoing disk activity

Scalability interpretation: system is compute- and memory-light

Responsive for interactive use at this scale

Scaling caveat: optimization needed for very large graphs or heavy concurrency

Expect need for tuning as graph size or load increases

Reproducibility and Code Accessibility

Exact run command, preserved files, and reproducibility checklist

1. **Exact run command: python3 bestpath.py**
   Run this to reproduce outputs using provided code and data

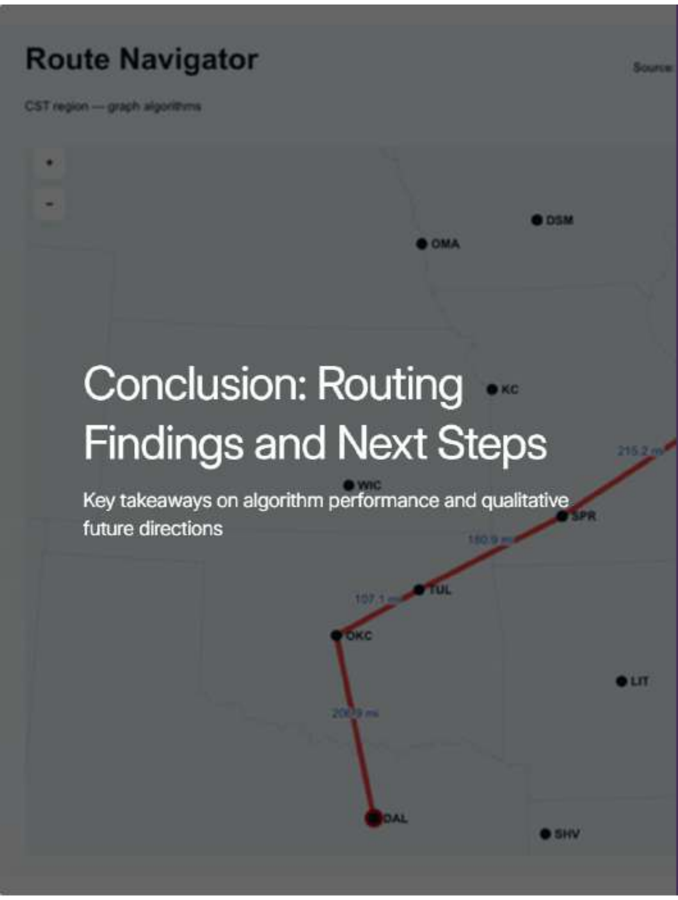
2. **Preserved files: bestpath.py, cities.csv, edges.csv, weather_risk.csv, index.html, script.js, style.css**
   All code, datasets, and frontend files included

3. **README with execution steps**
   Include setup, run steps, and expected outputs

4. **Pin Python version**
   Specify exact Python version used to run tests

5. **Document CSV schema**
   Describe columns, types, and sample rows

6. **Sample server start and API calls**
   Provide commands for starting server and example API request



Conclusion: Routing Findings and Next Steps

Key takeaways on algorithm performance and qualitative future directions

1

Weighted algorithms delivered shorter, safer, and more fuel-efficient routes, with Bellman-Ford highlighted

Preserved conclusion: Bellman-Ford gave best tradeoffs in experiments

2

BFS and DFS were inadequate for realistic routing due to lack of weighted path handling

Unsuited for distance or risk-based routing

3

Prim and Kruskal are useful for network structure analysis but not for point-to-point optimal routes

Good for spanning network insights, not direct routing

4

Expand the city set to increase scenario coverage

Broaden geographic and route diversity for robustness

5

Incorporate live weather APIs to model dynamic environmental risk

Introduce real-time conditions into route selection

6

Consider Dijkstra and A-star for efficiency and add stochastic risk models; evaluate multi-criteria optimization formally

Algorithmic efficiency and probabilistic risk assessment; formal multi-criteria study

Acknowledgements and Reference Summary

Guidance, sources, and where to find project artifacts

Acknowledgement: gratitude to Professor for guidance

Thank you to Professor David for mentorship and technical guidance.

Project artifacts: local project folder holds CSVs and code

Datasets and scripts available in the project folder for reviewer inspection.

References: Weather Underground

Weather Underground site used for historical weather data.

References: Google Maps Platform

Maps and geocoding APIs used for spatial data.

References: Cormen et al.

Algorithm theory and background from Cormen et al.

References: GeeksforGeeks

Practical coding examples and explanations.

References: Python docs

Language reference and standard library guidance.

References: OpenStreetMap

Open geographic data sources used in mapping.

References: D3.js

Visualization toolkit choices informed by D3.js.

References: MDN

Web API and browser behavior references from MDN.

References: Linux docs

System and shell usage from Linux documentation.

References: Rosen

Systems and networking theory from Rosen.

References: USGS

Geospatial and geological reference data from USGS.

References: Stack Overflow

Practical Q and A used during development.

Reviewer note: references supported toolkit choices and theory

Cited sources informed design decisions and background theory.

Thank