Project Report: Philadelphia Weather Crisis Evacuation Assistant (WCEA)

Course: CMPSC 463 – Algorithms
Authors: Tommy Lu & Matthew Kaplan

---

1. Project Description

The *Philadelphia Weather Crisis Evacuation Assistant (WCEA)* is a web-based navigation platform designed to help individuals evacuate safely during severe weather events. The system uses real geographical coordinates and graph algorithms to guide users to the safest or fastest path through Philadelphia while avoiding common danger zones.

Users can enter any start and destination address, and the system computes a route that:

- Avoids danger zones (flood or storm areas)
- Offers a safe route for avoiding danger
- Provides a button for an automatic evacuation to the nearest Safe Zone

The project showcases the real-world application of algorithmic design and graph data structures.

---

2. Project Significance

Weather disasters such as floods, hurricanes, and extreme storms pose life-threatening risks, especially in dense urban environments like Philadelphia. Many evacuation apps are generic and do not incorporate danger zones directly into their algorithms.

This project is significant because it:

Integrates hazard-awareness into routing

Unlike typical navigation tools, WCEA treats danger zones as regions with extremely high traversal cost, forcing algorithms to avoid hazards.

Demonstrates real-world use of graph algorithms

Using A* and Dijkstra’s algorithms to plan safe and quick routes for avoiding dangers.

Supports emergency resilience

Safe evacuation routes can save lives, especially in low-visibility conditions or unexpected floods.

Uses real-world geodata

Addresses are converted to coordinates via geocoding, and a graph allows for some routing to be done.

Offers multiple decision-making modes

- Safest route (Dijkstra’s avoids hazards)
- Fastest route (A*)
- Nearest shelter evacuation

Overall, the system provides a modern, algorithmically optimized solution to a relevant public safety problem.

---

3. Code Structure

The codebase is modular and organized for clarity and scalability.

```
WCEA/
 ├── index.html        (HTML interface + UI buttons)
 ├── style.css         (User interface styling)
 ├── app.js             (Main logic, routing modes, hazard avoidance)
 ├── /src
 │      ├── graph.js    (Graph data structure + nearest-node function)
 │      ├── dijkstra.js (Safest route algorithm)
 │     ├── astar.js    (Fastest route algorithm)
 │     ├── priorityqueue.js (Min-heap priority queue)
 ├── /data
 │     └── philly_graph.json (Philadelphia street network)
 ├── README.md
```

Diagram Overview

```
User Input -> Geocoder -> (Start/End Coordinates)
                   |
                   v
           Graph Loader -> Graph Object -> Danger Penalty Modifier
                   |
                   v
     +----------------------------+
     |   Routing Algorithms       |
     |                            |
     |  Dijkstra (Safest)         |
     |  A* (Fastest)              |
     |  Nearest Safe Zone Logic   |
     +----------------------------+
                   |
                   v
           Map Rendering (Leaflet)
```

Each module has a single responsibility, supporting clean, readable, and maintainable code.

---

4. Description of Algorithms

4.1 Dijkstra's Algorithm (Safest Route)

Dijkstra's algorithm computes the lowest-cost path through the graph. It does so by comparing costs overall to get to the next destination and picking the lowest. This always leads to the selection of the lowest cost path to get to the destination, making it perfect for evacuation scenarios since it will avoid danger zones. In our project, nodes inside danger zones receive a huge cost penalty, making it so that the algorithm consistently chooses the best paths that avoid danger zones.

Cost function:

```
cost(u → v) = distance_weight + danger_penalty
```

4.2 A* Search Algorithm (Fastest Route)

A* optimizes pathfinding by estimating distance to the goal using a heuristic. By doing this, we get a faster route to the destination, but it completely ignores the danger zones. Despite ignoring the danger zones, it is still useful in situations where that particular danger zone ended up ok to pass through or in cases where the user is pre-emptively leaving their home prior to a disaster to get to a safer place.

Heuristic:

```
h(n) = Euclidean distance between n and destination
```

Cost function:

```
f(n) = g(n) + h(n)
```

Where:

- g(n) = cost from start
- h(n) = estimate to end

---

5. Verification of Algorithms with Toy Examples (5%)

Toy Graph Example

```
A ----2---- B ----2---- C
 \                       \1
  \-----1+danger--------- D
```

Case 1 – Safest Route (Dijkstra)

Danger node D is penalized heavily:

Path chosen:

```
A → B → C
Cost = 2 + 2 = 4
```

Case 2 – Fastest Route (A* without hazard penalties)

A* will consider the path to D because heuristic is smaller:

```
A → D → C
Cost = 2
```

Verification Outcome:

- Dijkstra avoids hazards → working correctly
- A* prioritizes distance → working correctly

---

6. Functionalities

Address Search

Any Philadelphia address typed into the program will be converted to coordinates which allows the application to use the nearest node for navigation.

Safest Route Mode (Dijkstra)

Avoids all danger zones and finds hazard-free paths, allowing users to reach their destinations safely.

Fastest Route Mode (A*)

Finds shortest path ignoring hazard penalties, allowing users to reach their destinations quickly.

Hazard Avoidance

Nodes inside red circles are automatically penalized for Dijkstra’s algorithm

Evacuate to Nearest Safe Zone

Computes the distance to the nearest shelters and automatically plans a route from the starting location.

Map Visualization

Safe zones are green circles, danger zones are red circles, and Dijkstra’s and A* are represented as paths between a starting and ending point.

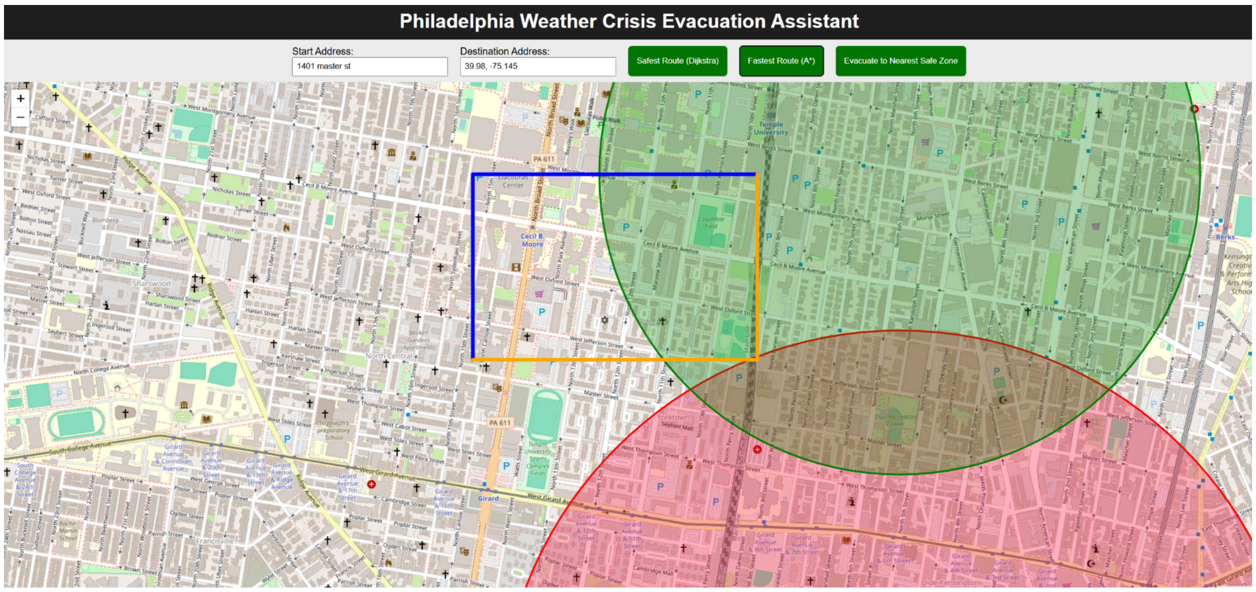
Geospatial Accuracy

All routing follows the streets defined in `philly_graph.json`.

---

7. Execution Results & Analysis

Test


Philadelphia Weather Crisis Evacuation Assistant

Start Address: 1401 master st

Destination Address: 39.98, -75.145

- Safest routes avoid the red area

- The fastest route passes slightly through red danger areas.

Conclusion

Dijkstra’s algorithm prioritizes safety and A* provides speed, showing that the hazard penalty strategy works.

---

8. Conclusions

This project demonstrates how classical algorithms like Dijkstra and A* can be applied to real-world safety-critical scenarios such as emergency evacuation routing. We successfully integrated:

- Graph Algorithms
- Heuristic search
- Hazard modeling
- Web development
- Data structures

We found that our Philadelphia Weather Crisis Evacuation Assistant is able to use Dijkstra’s algorithm and A* to successfully map paths from starting locations to safe destinations. Given the ability to either prioritize a quick or safe evacuation route, this could improve safety in weather crises by giving users access to routing that consistently avoids disaster zones. We found that by assigning very high costs to paths inside danger zones in our Dijkstra’s algorithm rather than deleting the nodes in these zones, we allow users to traverse through the zone using our A* algorithm or to have a starting point inside the zones for Dijkstra’s algorithm.

One issue with our project is that the small number of nodes we used are not nearly enough to cover a complex city like Philadelphia. The project would greatly benefit from an increased number of nodes on the graph, allowing for streets to be taken and faster or safer routes to be selected. Another issue is our system for geocoding does require internet access. It’s not guaranteed that a user will have access to the internet during a crisis, so finding a way to make the application work offline for at least routing to shelters would allow for more users to find safe routing during disasters, even though offline would not allow for up to date disaster data.

Overall, the project successfully demonstrates knowledge of Dijkstra’s algorithm which we studied in the course. It shows that pathing algorithms can be applied to real world scenarios to create useful applications. Dijkstra’s is an incredibly useful algorithm and the application we created would not function properly without it.