Graph-Theoretic Approach to Accessible Pathfinding Using OpenStreetMap Data

Alex Martinellli

1 Introduction

Navigating urban environments poses unique challenges for individuals with disabilities, particularly those using wheelchairs. To ensure equitable access, cities need optimized infrastructure that accommodates the mobility needs of all users. One way to contribute to this goal is by applying graph theory and algorithms in the context of geospatial data, particularly OpenStreetMap (OSM) data. In this essay, I explore how graph theory can be employed to identify the most accessible path for a wheelchair user, integrating both geospatial analysis and mathematical optimization. I will discuss key graph theory concepts, algorithms like A*, and metrics such as path cost, distance, and accessibility constraints.

2 Problem Formulation: Mapping Urban Spaces

The project uses geospatial data from OpenStreetMap to model urban environments as a graph $G = (V, E)$, where V represents the set of nodes (intersections or specific locations) and E represents the set of edges (paths connecting those intersections). Each edge has an associated cost based on the physical characteristics of the path, such as incline, surface type, and kerb height. The goal is to calculate the shortest and most accessible path from a starting point to a destination for a wheelchair user.

Let V be the set of nodes, and E be the set of edges between nodes. Each edge $e_{ij} \in E$ has a weight w_{ij}, which represents the accessibility cost for that edge. The weights are determined by a combination of geospatial distance and accessibility features:

$$w_{ij} = d_{ij} + a_{ij}$$

where d_{ij} is the geodesic distance between nodes i and j, and a_{ij} is the accessibility cost, which accounts for surface quality, incline, and other obstacles.

3 Graph Construction: Modeling Streets as a Network

Using OSM data, we build a directed weighted graph. Nodes represent locations (intersections, buildings, etc.), and edges represent streets, footpaths, or crossings. Each edge has a weight calculated based on both physical distance and accessibility factors.

The graph is constructed from three main elements in OSM:

- **Nodes**: Represent geolocations, each having coordinates (lat, lon).
- **Ways**: Ordered lists of nodes that form paths.
- **Relations**: Metadata that provide context on the paths, such as whether a path is pedestrian-friendly or accessible.

The first step is retrieving the nodes near a wheelchair user's start and end points. For this, we query MongoDB to find all nodes within a certain radius from the midpoint between the start and end coordinates:

Listing 1: Query to retrieve nearby nodes

```
node_results = list(collection_nodes.find({
    "location": {
        "$nearSphere": {
                "$geometry": {
                    "type": "Point",
                    "coordinates": [middle_point[1], middle_point[0]]
                },
                "$maxDistance": radius
            }
        }
}))
```

Next, we build the graph by creating edges between the nodes based on OSM "ways" that contain these nodes. Each edge is assigned a weight that combines geodesic distance and the accessibility cost of the corresponding path segment.

4 A* Algorithm: Optimizing Pathfinding

Given the graph, the problem of finding the most accessible path is analogous to finding the shortest path in a weighted graph. The A* algorithm, a well-known heuristic-based search algorithm, is used to find the optimal path. The A* algorithm operates by minimizing the total cost $f(n)$ for each node n, which is the sum of two components:

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

Where:

- $g(n)$ is the cost of the path from the start node to n,
- $h(n)$ is a heuristic estimate of the cost from n to the goal.

In our case, $g(n)$ is the sum of the distances and accessibility costs encountered so far, and $h(n)$ is the geodesic distance from node n to the destination.

The heuristic function $h(n)$ is computed using the geodesic distance between nodes, and the total path cost is adjusted according to the accessibility constraints for each path segment:

Listing 2: A* algorithm for pathfinding

```
def heuristic(node_coords, goal_coords):
    return geodesic(node_coords, goal_coords).meters

def a_star(graph, start, goal, nodes_dict, ways_to_nodes, ways_dict):
    open_heap = []
    heapq.heappush(open_heap, (0, start))
    cost_so_far = {start: 0}
    while open_heap:
        current = heapq.heappop(open_heap)[1]
        if current == goal:
            break
        for next_node, details in graph[current].items():
            new_cost = cost_so_far[current] + details['distance']
            accessibility_cost = calculate_accessibility_cost(next_node, nodes_dict, wa
            new_cost += accessibility_cost
            if next_node not in cost_so_far or new_cost < cost_so_far[next_node]:
                cost_so_far[next_node] = new_cost
                heapq.heappush(open_heap, (new_cost + heuristic(nodes_dict[next_node]['c
```

5 Accessibility Score Calculation

The Accessibility Score measures how well the route's available accessibility information aligns with recommended values. An Accessibility Score of 1 indicates that the route is perfectly suitable for the user, while a score of 0 indicates the opposite.

The Accessibility Score A_s for the entire route is computed as the weighted sum of the segment accessibility scores, normalized by the total length of the route. The formula for calculating the accessibility score of the entire route is:

$$A_s = \frac{\sum_{i=1}^{n}(segmentAccessibility_i \times segmentLength_i)}{totalLength}$$

Where:

- $segmentAccessibility_i$ is the accessibility score for segment i,
- $segmentLength_i$ is the length of segment i,
- $totalLength$ is the total length of the route.

To calculate the accessibility score for each segment, we compute the accessibility value for each feature based on how well it aligns with the user's preferences. This accessibility value $accessibilityValue_x$ is determined by:

$$accessibilityValue_x = \begin{cases} 1, & \text{if Accessibility Information } x \text{ aligns with the recommended value} \\ 0, & \text{if Accessibility Information } x \text{ does NOT align with the recommended value} \end{cases}$$

The segment accessibility score is then calculated by summing up the weighted accessibility values for all features:

$$segmentAccessibility_i = \frac{\sum_{j=1}^{m}(userWeight_j \times accessibilityValue_j)}{\sum_{j=1}^{m} userWeight_j}$$

Where:

- $userWeight_j$ is the weight assigned by the user to feature j,
- $accessibilityValue_j$ is the accessibility value for feature j,
- m is the number of available accessibility features for the segment.

6 Conclusion

This project demonstrates how graph theory and geospatial analysis can be applied to real-world problems like accessible pathfinding. By representing urban spaces as a graph, we can efficiently compute paths that meet both distance and accessibility criteria, enhancing mobility for wheelchair users. The combination of algorithms like A*, mathematical formulations of accessibility costs, and graph theory provides a robust framework for solving similar problems in smart city planning and infrastructure development.

References

[1] OpenStreetMap contributors. (n.d.). *OpenStreetMap*. `https://www.openstreetmap.org/`.

[2] Author Name. (2021). *A Graph-Based Approach to Accessible Routing*. School of Computer Science and Statistics, Trinity College Dublin. `www.scss.tcd.ie/2021/TCD-SCSS-DISSERTATION-2021-033`.

[3] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 269–271.

[4] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.