Dynamic Management of Intelligent Transportation System

Raja Sodani (21ucs164) Jeevesh Lakhmani (21ucs099) Romit Chourdiya (21ucc084)

Supervisor- Dr. Saurabh Kumar

April 18, 2024

LNMIIT
The LNM Institute of Information Technology

Table of Contents

Problem Statement

- The project focuses on dynamic management of ITS (Intelligent Transportation System) resources during emergency situations including medical attention, fire sites, disaster-affected locations, etc.
- ITS can efficiently aid in the process of the allocation of ambulances in case of a medical emergency considering different constraints such as congestion, route, distance, etc. in a timely manner.
- Proposed solutions for the above problem will be achieved by implementing different algorithms such as Dijkstra's and constraint optimization algorithms on an existing framework based on the transportation system.

Motivation

- **Critical Need for Efficient Emergency Response:** Traditional transportation systems struggle to cope with dynamic emergencies like accidents or fires, necessitating a solution for swift and coordinated responses.
- **Dynamic Management Framework:** Our project introduces a dynamic management framework tailored for Intelligent Transportation Systems (ITS), leveraging advanced algorithms and real-time data analysis to optimize resource routing and scheduling during emergencies.
- **Impact and Benefits:** By reducing response times, enhancing coordination, and optimizing resource utilization, our system aims to save lives, minimize damage, and improve overall resilience in the face of unpredictable events.

Assumptions

- **Random Points Generation for Graph Modelling:** The accident site and resource warehouse locations have been randomly generated due to unavailability and sensitivity of the actual database.
- **Constraints assumption:** Different constraints such as road type, neighbourhood of the route, traffic lights, and road congestion have also been assumed to be assigned random values for the purpose of map modelling and algorithm testing.

Algorithm 1: Initialization of Graph from Map Data

- The first algorithm involves initializing a graph from the given map data. The graph is constructed to represent the network of resources and their connections. Resources including hospitals, fire stations, and hazmat vehicle stations are added as nodes to the graph.
- Routes between these resources and the disaster location are established as edges, considering constraints such as distance and accessibility. By creating this graph, we provide a structured representation of the available resources and their connectivity.

Algorithm 1: Initialization of Graph from Map Data

Algorithm 1 Random Adjacency Matrix with Coordinates

```
function GENERATE_COORDINATES(minX, maxX, minY, maxY, srcX, srcY)
    x ← random.uniform(minX, maxX)
    y ← random.uniform(minY, maxY)
    a ← (srcX − x)^2
    b ← (srcY − y)^2
    distance ← sqrt(a+b)
    return (x, y, distance)
end function
function CREATE_RANDOM_ADJACENCY_MATRIX_WITH_COORDINATES(n, edge_probability = 0.25)
    Initialize adjacency matrix with zeros
    adjacency_matrix ← zeros((n + 1, n + 1), int)
    resource ← []
    for i in range(n) do
        coordinate ← GENERATE_COORDINATES(minX,maxX,minY,maxY,srcX,srcY)
        resource.append(coordinate)
    end for
    for i in range(n + 1) do
        for j in range(i + 1, n + 1) do
            if random.random() < edge_probability then
                adjacency_matrix[i, j] ← resource[i][2]
                adjacency_matrix[j, i] ← resource[i][2]
            end if
        end for
    end for
    return adjacency_matrix, resource
end function
```

Algorithm 2: Updating Edge Weights Considering Constraints

- The second algorithm focuses on updating edge weights based on various constraints such as road type, traffic lights, congestion, locality, and time.
- By assigning weights to edges based on these constraints, the algorithm aims to provide a more accurate representation of the road network's characteristics.
- This information is crucial for optimizing resource allocation during emergency situations, as it considers factors that may affect travel time and route efficiency.

Algorithm 2: Updating Edge Weights Considering Constraints

Algorithm 2 Assigning Edge Weight

```
Globals: class Time:
slot = [0, 1, 2, 3]
procedure GET_TIME
    currTime ← int(datetime.now().time().strftime("%H"))
    if (8 ≤ currTime ≤ 10) then
        return 1                                          ▷ Morning
    else if (10 < currTime < 17) then
        return 2                                          ▷ Afternoon
    else if (17 ≤ currTime < 22) then
        return 3                                          ▷ Evening
    else
        return 0                                          ▷ Night
    end if
end procedure
Globals: class Edge:
                ▷ Shows the congestion in different localities during whole day
Locality = {    'institute': ['low', 'high', 'high', 'low'],
  'industrial': ['high', 'moderate', 'low', 'moderate'],
  'residential': ['low', 'high', 'moderate', 'high'],
  'market': ['low', 'high', 'high', 'high']
}
procedure _INIT_(self, length, road_type, traffic_lights, congestion, locality)
                                  ▷ Constructor to initialize edge attributes
    self.road_type← random.choice['highway','4−laneroad','2−laneroad'])
    self.traffic_lights← random.choice([True, False])
    self.congestion← random.choice(['low','medium','high'])
    self.locality← random.choice(['institute','industrial','residential','market'])
```

Algorithm 2: Updating Edge Weights Considering Constraints

```
    if (road_type) then
        self.road_type ← road_type
    end if
    if (traffic_light) then
        self.traffic_lights ← traffic_lights
    end if
    if (congestion) then
        self.congestion ← congestion
    end if
    if (locality) then
        self.locality ← locality
    end if
end procedure
function CALCULATE_CONSTRAINTS_EDGE_WEIGHT(self)
    LENGTH_WEIGHT = 0.5
    ROAD_TYPE_WEIGHT = {'highway' : 0.3,' 4 − laneroad' : 0.5,' 2 − laneroad' : 1.0}
    TRAFFIC_LIGHT_WEIGHT = {True : 1.5, False : 1.0}
    LOCALITY_WEIGHT = {'low' : 1.0,' moderate' : 1.3,' high' : 1.5}
    CONGESTION_WEIGHT = {'low' : 1.0,' medium' : 1.3,' high' : 1.5}
    weight ← 0
    if self.length ≠ 0 then
        weight ← (    LENGTH_WEIGHT × self.length+
          ROAD_TYPE_WEIGHT[self.road_type]+
          TRAFFIC_LIGHT_WEIGHT[self.traffic_lights]+
          CONGESTION_WEIGHT[self.congestion]+
          LOCALITY_WEIGHT[Edge.area[self.locality][Time.get_time()]]
        )
    end if
    return weight
end function
```

Algorithm 3: Dijkstra's Algorithm for Shortest Route Calculation

- We use Dijkstra's algorithm to find the quickest path from the accident to every resource warehouse. By adjusting the graph and edge weights, it picks the best route for sending resources fast.
- This makes sure emergency vehicles get where they need to go as quickly as possible, cutting response times and avoiding hold-ups.

Algorithm 3: Dijkstra's Algorithm for Shortest Route Calculation

Algorithm 3 Dijkstra's Algorithm

```
1: function DIJKSTRA(graph, src)
2:     vertices ← len(graph)
3:     Initialize :
4:     dist ← [sys.maxsize] ∗ vertices
5:     dist[src] ← 0
6:     visited ← {}
7:     shortest_paths ← defaultdict(list)   ▷ Dictionary to store shortest paths
8:     for i in range(vertices)
9:         shortest_paths[i].append(src)
10:    end for
11:    for each vertex in range(vertices) do
12:
13:        ▷ Find vertex with the minimum distance among unvisited vertices
14:        min_dist ← sys.maxsize
15:        min_vertex ← −1
16:
17:        for v in range(vertices) do
18:            if v ∉ visited and dist[v] < min_dist then
19:                min_dist ← dist[v]
20:                min_vertex ← v
21:            end if
22:        end for
```

Algorithm 3: Dijkstra's Algorithm for Shortest Route Calculation

```
        visited.add(min_vertex)                    ▷ Mark vertex as visited

        for v in range(vertices) do ▷ Update distances for adjacent vertices
            if v ∉ visited and graph[min_vertex][v] ≠ 0 and then
                and dist[min_vertex] ≠ sys.maxsize and
                and dist[min_vertex] + graph[min_vertex][v] < dist[v] do
                dist[v] ← dist[min_vertex] + graph[min_vertex][v]
                                          ▷ Update shortest path to vertex v
                shortest_paths[v] ← shortest_paths[min_vertex] + [v]
            end if
        end for

    end for
    return dist, shortest_paths
end function
```

Results and Discussion- Dijkstra's Algorithm

- The results obtained from implementing Dijkstra's algorithm accurately computes the shortest routes to resource warehouses, where we can optimize emergency response efforts and enhance the overall efficiency of the transportation system during critical situations.
- This provides valuable insights into the optimal paths emergency vehicles can take to reach medical facilities quickly.

Results and Discussion- Dijkstra's Algorithm

```
[[ 0  0  0 18  0  0  0 17 19  0 18]
 [ 0  0  0  0 10 10  0  9 10  0  0]
 [ 0  0  0 17  0 16  0  0 16  0 16]
 [18  0 17  0  0  0  0 17 17  0  0]
 [ 0 10  0  0  0  0  0  0  0  0  0]
 [ 0 10 16  0  0  0  0  0  6  6  7]
 [ 0  0  0  0  0  0  0 17  0  0 17]
 [17  9  0 17  0  0 17  0  0  0  0]
 [19 10 16 17  0  6  0  0  0 12 12]
 [ 0  0  0  0  0  6  0  0 12  0  0]
 [18  0 16  0  0  7 17  0 12  0  0]]
```

Adjacency Matrix for Map Modelling

```
Shortest paths from accident spot to every other resource:
Resource 0: Distance = 0, Shortest path = [0]
Resource 1: Distance = 26, Shortest path = [0, 7, 1]
Resource 2: Distance = 34, Shortest path = [0, 10, 2]
Resource 3: Distance = 18, Shortest path = [0, 3]
Resource 4: Distance = 36, Shortest path = [0, 7, 1, 4]
Resource 5: Distance = 25, Shortest path = [0, 10, 5]
Resource 6: Distance = 34, Shortest path = [0, 7, 6]
Resource 7: Distance = 17, Shortest path = [0, 7]
Resource 8: Distance = 19, Shortest path = [0, 8]
Resource 9: Distance = 31, Shortest path = [0, 8, 9]
Resource 10: Distance = 18, Shortest path = [0, 10]
```

All Possible Shortest routes

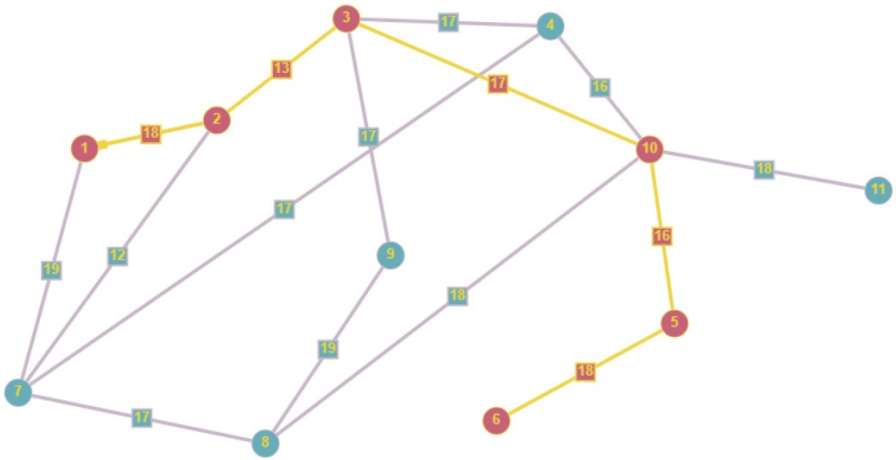
Dijkstra's Algorithm- Visual Representation

Figure: Example for shortest route between accident site and resource location 5

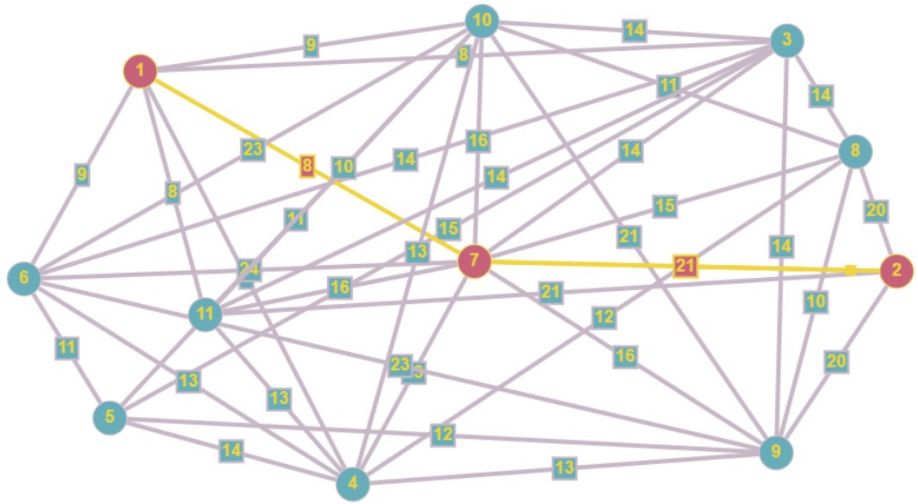
Results and Discussion- Dijkstra's Algorithm vs Bellman Ford Algorithm

- **Efficiency and Suitability of Dijkstra's Algorithm:** Its time complexity of O((V+E)logV) enables quicker calculations compared to Bellman-Ford's O(VE), making it faster for dense or heavily interconnected graphs.
- **Avoidance of Negative Edge Weights:** It avoids negative edge weights, unlike Bellman Ford preventing inaccuracies from corrupt map data. This focuses on non-negative weights which ensures the integrity of our graph model.

Results and Discussion- Variation of p, edge probability



Sparse graph, limited routes between nodes, p = 0.25



Dense graph, multiple route options, p = 0.75

Conclusion

- In summary, our project on "Dynamic Management of Intelligent Transportation System" tackles the urgent need for quick emergency response by introducing a system that adapts to real-time situations in transportation.
- Using smart algorithms like Dijkstra's, we optimize how resources are directed during emergencies, aiming to save lives and minimize harm.
- We assumed accident and resource locations are randomly chosen for testing, along with assigning random constraints. Dijkstra's algorithm performed well in finding the shortest routes, crucial for speedy help in emergencies.
- By avoiding negative weights, we ensure our system's reliability, making it more resilient in unpredictable events

Future Work

- **Multiple objective and constraints implementation:** The problem statement can be modified to include multiple resources and accident sites. Other constraints such as fuel availability, communication and network issues can be introduced in the future.
- **Multi-agent based implementation:** The problem can be modelled such that agent-based solutions like negotiations, and constraint optimization algorithm can be implemented for efficiency and optimality.
- **Scope:** The proposed solution could be further extended to optimize resource allocation for taxi service and product delivery platforms.

References

- A Survey of Intelligent Transportation Systems: S. -h. An, B. -H. Lee and D. -R. Shin, "A Survey of Intelligent Transportation Systems," 2011 Third International Conference on Computational Intelligence, Communication Systems and Networks, Bali, Indonesia, 2011, pp. 332-337, doi: 10.1109/CICSyN.2011.76.
- Choksi, Meghavi Zaveri, Mukesh. (2019). Multiobjective Based Resource Allocation and Scheduling for Postdisaster Management Using IoT. Wireless Communications and Mobile Computing. 2019. 1-16. 10.1155/2019/6185806.
- J Sathish Kumar, Mukesh A Zaveri, Meghavi Choksi, Task Based Resource Scheduling in IoT Environment for Disaster Management, Procedia Computer Science, Volume 115, 2017, Pages 846-852,