Fusing Trajectories of Exploring Agents in a Crisis Environment using DeepSeek

Rothkrantz, Leon; Fitrianie, Siska

DOI
10.1109/InfoTech67177.2025.11175940

Publication date
2025

Document Version
Final published version

Published in
2025 39th International Conference on Information Technologies, InfoTech 2025 - Proceedings

2025 International Conference on Information Technologies (InfoTech) | 979-8-3315-9762-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/InfoTech67177.2025.11175940

Fusing Trajectories of Exploring Agents in a Crisis Environment using DeepSeek

Leon Rothkrantz and Siska Fitrianie
Department of Man Machine Interaction, Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
L.J.M.Rothkrantz@tudelft.nl

Abstract – **During a natural disaster, when roads are damaged or blocked, rescue agents search the area to find new routes from start to destination. Their trajectories are sent to a crisis center and merged into a new map. The DeepSeek and ChatGPT algorithms help build this map by combining the agents' explored routes. This paper presents the algorithm and its application.**

Keywords – **ChatGPT, DeepSeek, exploring agents, fusing trajectories.**

I. Introduction

In 2002, Prague experienced severe flooding when the Vltava River overflowed, spreading water across the city. The sewer system, along with underground canals and tunnels, allowed the water to reach areas far from the river. The city's uneven terrain, with peaks, valleys, and tunnels, meant that low-lying areas, both near and far from the river, were eventually flooded (see Fig. 1 and Fig. 2). Authorities warned people in time to evacuate, but some delayed until the last moment, while others, such as disabled individuals, required vehicle assistance. A major challenge was that mobile navigation systems failed as water levels rose because street networks were no longer usable. Updated maps were needed to guide evacuations.

Fig. 1: Flooded area next to the river and the valley of the city

During floods, safe zones are established, and victims must be moved from flooded areas to these zones. To create an updated map of passable streets, rescue workers leave safe areas and explore the city from different directions. Using handheld devices, they could record which streets remain open, marking each crossing they pass. This generates a sequence of connected crossings, forming straight-line segments. Rescue workers periodically send their recorded routes to a crisis center, where they are plotted on a map. This map is represented as a graph, with street crossings as nodes and connecting streets as edges. Since water levels change dynamically, the map must be updated continuously. Multiple rescue teams contribute to this process, sending updates from different locations and times.

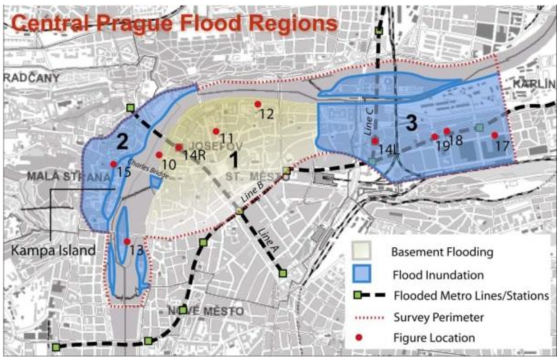

Fig. 2: Map of the inner city of Prague with flooded areas

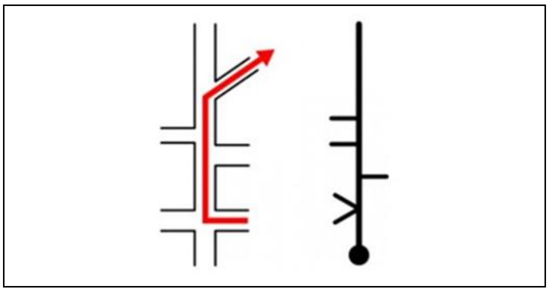

Fig. 3: Part of a street network with examples of fishbone

Geometric features such as distances and angles are not considered during the process. At fixed intervals, the explored routes are transmitted to the crisis center, where they are plotted on a map. The map is structured as a graph, with street intersections as nodes and connecting streets as edges. Map updates occur dynamically, particularly when water levels are still rising or falling. Typically, multiple rescue workers contribute to updating the map. Updates from different locations and times are sent to the crisis center and must be merged.

One notable routing method is the Fishbone algorithm (Fig. 3), a complex routing method used in exploration tasks. A first responder moves through a city, always heading North on the map. At each intersection, he notes how many roads they pass on the left or right. The resulting map is not a precise Euclidean map with exact angles and distances, but it should be simple to create, even using voice commands. This paper addresses the challenge of combining multiple Fishbone routes into a single graph. We present a solution to this problem.

A similar issue arises in buildings, such as hotels, where corridors form a network leading from the entrance to the rooms. In emergencies like fires or explosions, some corridors may become blocked, disrupting the original navigation trajectories. Rescue workers must explore the building to determine which corridors are still safe and which rooms remain accessible.

A single rescuer can update the map, but using multiple workers speeds up exploration. Each rescuer records their trajectories on a handheld device. At certain intervals, these routes are shared with a central system, where they are combined into a unified world model.

The problem to solve:

1. Generate exploration trajectories: Agents explore a graph-based environment and record their trajectories.
2. Merge trajectories into a new graph: Using DeepSeek technology, combine individual routes into a complete graph.
3. Visualize the graph: Display the final graph using Graph Online for analysis and decision-making.

II. LITERATURE SURVEY

A Multi-Agent System (MAS) can be used to explore and map a maze. Tjiharjadi et al. (2022) [1] reviewed different approaches for maze exploration using agents. While a single agent can find a path from start to goal, using multiple agents speeds up the search and helps locate multiple goals. The study covers applications like warehouse robots, self-driving cars, video games, and traffic control. The paper discusses various pathfinding algorithms, including A*, Ant Colony Optimization, Genetic Algorithms, Depth-First Search, Breadth-First Search, and Flood Hill. However, these methods focus on static environments. In contrast, our research deals with dynamic environments affected by disasters like floods, earthquakes, fires, or hurricanes. Another key concern is whether wireless networks remain functional in such scenarios. The next section explores mobile ad-hoc networks as a potential solution.

Linardakis et al. (2024) [2] studied robot exploration in complex environments. They highlight challenges like limited communication range, high communication costs, and overlapping coverage areas. Their solution, CU-LVP, uses Voronoi diagrams to improve area division among agents. The method was tested in different maze layouts. Marjovi et al. (2009) [3] later applied this approach in fire-detection scenarios, where agents move in four directions (up, down, left, right) while avoiding occupied cells to prevent collisions. We will refer to this example later in the paper.

Kivelvitch et al. (2010) [4] investigated maze exploration using multiple robots. They implemented Tarry's algorithm to coordinate group behavior and found that more robots increase system robustness. Their method worked well in mazes of any size, supporting the idea that "more robots lead to better results."

Our research incorporates ChatGPT and DeepSeek. As shown in Song et al., (2024) [5], Large Language Model (LLM) agents improve by learning from their mistakes in real-world interactions. This approach outperforms traditional expert systems in optimizing movement trajectories.

Das et al. (2007) [6] studied how multiple agents can explore an unknown graph and build a labeled map together. Key challenges include navigation and coordination, especially when agents know the graph size or agent count in advance. We will discuss a similar problem in this paper. On the other hand, Dudek et al., (1992) [7] explored graph-like environments where robots lack distance or orientation data. They found that exploration is impossible without special markers. Their solution involves algorithms where robots place and detect markers to navigate effectively.

Crnković et al. (2023) [8] focused on using robots in disasters like earthquakes, fires, and floods. Their goal was to locate victims in collapsed buildings, mines, or cellars. While single-agent algorithms exist, multi-agent solutions are still rare. The team developed a system where mobile agents explore unknown mazes while avoiding collisions and coordinating coverage. Their method, inspired by the Heat Equation Driven Area Coverage, performed better than existing systems.

Ruan et al. (2020) [9] proposed a new way to update urban road networks. Traditional surveys are time-consuming, so they used GPS data from mobile devices to reconstruct road layouts. Their deep-learning framework outperformed older methods by uncovering hidden patterns in traffic data.

Zhang et al. (2021) [10] analyzed wireless networks inside buildings—a critical need for first responders. In the Netherlands, the C2000 network was designed for emergencies but struggled indoors without signal boosters. Since disasters can damage infrastructure, the team developed tools to optimize building designs for better wireless performance.

Kim et al. (2017) [11] studied city traffic by visualizing vehicle movements using graphs. They collected data from sensors, mobile devices, and IoT systems, then applied mining algorithms to detect traffic pattern changes over time.

III FUSION OF LOCAL MAPS IN MOBILE AD-HOC NETWORKS

In 2002, our research team became involved in a project focused on dynamic routing for emergency response during disasters [12, 13, 14, 16]. The study was prompted by a real-world crisis: the severe flooding of the Vltava River in and around Prague. Large sections of the city were submerged, making it difficult for rescue teams to navigate. To assist first responders, an updated street map was urgently needed, but traditional mapping methods were insufficient due to rapidly changing conditions.

To address this, emergency teams were dispatched from the crisis center to survey the city and determine which streets remained accessible. A critical tool in this effort was the Sharp Zaurus, a handheld digital assistant (Fig. 4). Equipped with custom navigation software, the device guided responders through the flooded areas. At each intersection, they received real-time instructions on which direction to take. Their chosen paths were periodically transmitted back to the crisis center, allowing for continuous updates to the city's accessibility map.

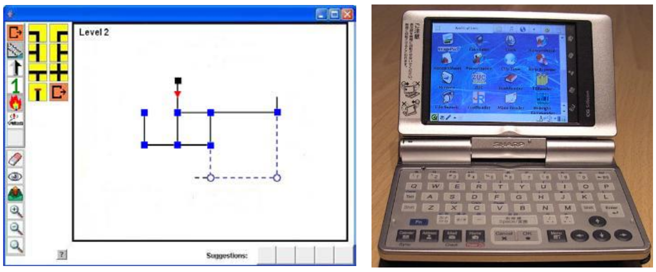

Fig. 4: Zaurus palmtop and iconic interface

At the crisis center, the challenge was to combine multiple incoming trajectories into a single, coherent map. In 2003, we developed specialized software to merge these paths, assuming that streets intersected at right angles—a common feature in urban grids. Fig. 5 illustrates how the software fused trajectories when they shared a common square intersection. To ensure robustness, we later rewrote the software in Python and conducted additional testing. A more in-depth discussion of this early work, including algorithmic details, can be found in Waasdorp (2010) [15]. However, the approach presented in this paper diverges from that earlier methodology.

```python
def get_edge(edges, vertex, direction):
    return edges.get((vertex, direction))

def get_other_vertex(vertices, vertex, edge):
    return edge[1] if edge[0] == vertex else edge[0] if edge[1] == vertex else None

def compare_edges(edge1, edge2):
    return edge1 == edge2  # Modify this comparison logic as needed

def build_hypotheses_graph(VH, E1, E2, V1, V2):
    directions = ['north', 'east', 'south', 'west']
    EH = []

    for u1, u2 in VH:
        for d in directions:
            e1 = get_edge(E1, u1, d)
            e2 = get_edge(E2, u2, d)

            if e1 is not None and e2 is not None:
                v1 = get_other_vertex(V1, u1, e1)
                v2 = get_other_vertex(V2, u2, e2)

                if v1 is not None and v2 is not None:
                    if (v1, v2) in VH:
                        if compare_edges(e1, e2):
                            EH.append(((u1, u2), (v1, v2)))

    return EH

```

Fig. 5: Python code for the graph construction algorithm

Waasdorp (2010)'s research focused on merging topological maps—simplified representations of environments where only connectivity matters, not exact distances. He explored several existing fusion algorithms before developing his own solution. To validate his approach, an experiment was conducted at the EWI-Faculty building of TU Delft, where student volunteers, equipped with digital assistants, explored the structure. Their movement data was transmitted to a central server, where it was processed and merged into a unified map. The results were particularly promising in a simulated hotel fire scenario, demonstrating the system's potential for real-world emergencies.

Waasdorp (2010) used topological maps to solve the map problem. Later approaches focused on graph-based and maze-based solutions. In this paper, we use unstructured graphs to address the challenge.

Meanwhile, technological advancements have made the Zaurus handheld obsolete, as smartphones, with their widespread infrastructure, are now the standard. In the Netherlands, first responders operate on a separate, secure network with a custom broadcast system. However, this network still has gaps in coverage, particularly indoors and in certain rural areas.

III. DeepSeek, ChatGPT, Chatbots

In 2024, OpenAI launched ChatGPT, an advanced chatbot designed for interactive conversations with users. This chatbot is powered by a large language model that uses supervised and reinforcement learning techniques. Studies show that ChatGPT can solve complex problems, including mathematical ones. However, the latest version is no longer free. Users must pay for full access, with only limited testing available without charge.

A year later, in 2025, a Chinese competitor called DeepSeek entered the market. Founded by Liang Wenfeng and backed by the hedge fund High-Flyer, DeepSeek remains free to use, which is why we chose it for this study.

Many research papers have already explored using chatbots to solve problems. However, comparing ChatGPT and DeepSeek is challenging because their algorithms and technical details are not publicly available. Some experts suggest that DeepSeek performs better at mathematical tasks, while ChatGPT excels in language-based problems. Still, no definitive study has confirmed these claims. For our research, we selected DeepSeek due to its free access.

Chatbots have become increasingly popular among students, even at the secondary school level. Many learners use them regularly for assistance. To counter potential misuse, specialized software has been developed to detect whether students' work was generated by chatbots. Beyond this, tools like ChatGPT can also serve as intelligent tutors, helping students prepare for exams by providing step-by-step solutions [19].

IV. Visualisation of a Graph

Many graph visualization tools are available today to represent networks from adjacency matrices, including popular libraries like Graphviz and NetworkX. These tools enable users to generate, analyze, and customize graph structures efficiently. However, while DeepSeek can compute graph-related data, its current version lacks built-in graphical input or output capabilities. Instead, it produces only the adjacency matrix, which must then be visualized using external software.

To address this limitation, we used Graph Online (https://graphonline.top), a web-based platform that allows interactive graph plotting and analysis. This tool not only visualizes graphs but also provides useful features such as computing the shortest path, applying different layout algorithms, and running graph traversals.

Additionally, it supports various fundamental algorithms, including Depth-First Search (DFS), Breadth-First Search (BFS), and Dijkstra's algorithm, making it a versatile solution for both visualization and graph-theoretic computations.

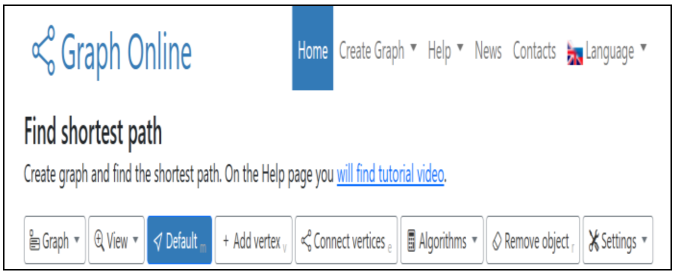

Fig. 6: Interface of graph generation tool Graph Online

IV. METHODOLOGY

In the introduction, we presented the problem we aim to solve. A city's street network can be represented as a graph, where streets are edges and intersections are nodes. If flooding damages this network, or if an explosion or fire damages a hotel's corridor network, multiple agents, such as rescue robots or firemen, can explore the area. These agents move randomly through the network, recording and sending their trajectories to a central point. The key question is: How can we reconstruct the full network from these saved trajectories?

We study two scenarios. In the first, the network is a labeled graph, and each trajectory is a sequence of labeled nodes. In the second, the graph's nodes are unlabeled, and trajectories consist of connected but unnamed nodes.

We begin with a simple network. First, we define exploring trajectories and the problem of composing them into a new network. We explicitly state the problem and ask DeepSeek to find a solution. Next, we task DeepSeek with solving the more complex problem of trajectories in an unknown graph. We then adapt its solution into our own version. Finally, we ask ChatGPT to solve the same problems. Although the phrasing differs, ChatGPT and DeepSeek produce similar solutions. In a later section, we present and annotate some of these solutions.

V EXPLORATION OF A LABELED GRAPH

Consider a burning building has an RFID-based navigation system. Robots or first responders equipped with this tag-reading system explore the building randomly. If the corridor crossings are numbered, each exploration route can be represented as a sequence of numbers. These routes allow us to build an adjacency matrix for a graph, which can then be visualized using our tool.

To explain our problem and solution, we use a step-by-step example. Suppose the ground floor of a hotel is divided into a 4x4 grid. We convert this floor plan into a graph with 16 nodes (representing intersections) and 24 edges (representing corridors). At some point, an explosion or fire occurs, blocking some edges due to damage or flames. To assess the damage, three firefighters take different paths through the hotel:

- Route 1: Travels around the outer boundary.
- Route 2: Goes from the top-left to the bottom-right corner.
- Route 3: Goes from the bottom-left to the top-right corner.

Based on this scenario, we define the following problem for DeepSeek and request a solution:

Problem: *Given a 2D grid graph with nodes labeled from 1 to n and edges defined by pairs of connected nodes (i, j), we consider three trajectories: 3-8-1-15-6-13-2, 9-5-10-1-15-16-7, and 9-4-14-2-13-11-7-16-8-3-12-5. Construct the minimal graph that contains all the edges required to represent these three paths, please!*

We analyzed the solution generated by DeepSeek and annotated it as follows:

Step 1. The nodes of the graph are numbered randomly or systematically.

Step 2. Define the trajectories using the numbered nodes

Trajectories:

1. **Trajectory 1**: 3-8-1-15-6-13-2
2. **Trajectory 2**: 9-5-10-1-15-16-7
3. **Trajectory 3**: 9-4-14-2-13-11-7-16-8-3-12-5

Step 3: Identify all unique nodes (points):

From the trajectories, we can extract all the unique nodes, such as:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

Step 4: Extract edges from the trajectories:

- **Trajectory 1**: (3,8), (8,1), (1,15), (15,6), (6,13), (13,2)
- **Trajectory 2**: (9,5), (5,10), (10,1), (1,15), (15,16), (16,7)
- **Trajectory 3**: (9,4), (4,14), (14,2), (2,13), (13,11), (11,7), (7,16), (16,8), (8,3), (3,12), (12,5)

Step 5: Combine all edges and remove duplicates:

The combined list contains undirected edges, meaning (i,j) and (j,i) represent the same connection:

[(3,8), (8,1), (1,15), (15,6), (6,13), (13,2), (9,5), (5,10), (10,1), (15,16), (16,7), (9,4), (4,14), (14,2), (13,11), (11,7), (16,8), (3,12), (12,5)]

Step 6: Represent the graph:

The graph can be represented in different ways, depending on the needs of the analysis. One common method is an adjacency list, which explicitly lists the connections (edges) between nodes (vertices). Alternatively, the graph can be depicted visually to provide an intuitive understanding of its structure. For example, Fig. 7 illustrates a graphical representation generated using a visualization tool. Below, we provide the adjacency list for further reference:

- **1**: 8, 10, 15
- **2**: 13, 14
- **3**: 8, 12
- **4**: 9, 14

- **5**: 9, 10, 12
- **6**: 15, 13
- **7**: 11, 16
- **8**: 1, 3, 16
- **9**: 4, 5
- **10**: 1, 5
- **11**: 13, 7
- **12**: 3, 5
- **13**: 2, 6, 11
- **14**: 4, 2
- **15**: 1, 6, 16
- **16**: 7, 8, 15

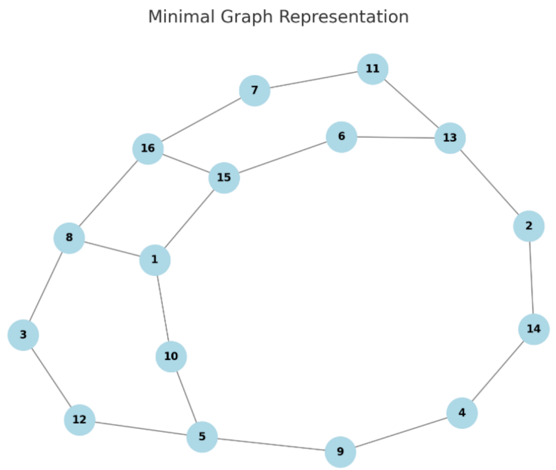

Fig. 7: Graph visualization defined by the adjacency matrix

VI Exploration of an unknown graph

We now study an environment where the street network is modeled as a graph, but without any labels. The only known information is that somewhere in this network, there is a dangerous area, such as an explosion, fire, or flooding. This disaster causes continuous changes in the network, making some nodes unreachable.

To explore this changing environment, multiple agents (such as robots or first responders) move through the network. At each intersection, an agent can choose to go left, right, or straight. They also detect whether a path is safe (S) or dangerous (D). Using symbols like E, W, N (for directions) and D or S (for danger/safety), each exploration path can be represented as a string of these symbols. The goal is to combine these exploration paths into a single graph. Initially, the nodes have no labels, but over time, we can assign coordinates to them step by step. This process creates a labeled graph. The following pseudo-code provides more details. For simplicity, we assume the network is structured as a 4x4 grid.

Based on this scenario, we analyzed the solution generated by DeepSeek and annotated it as follows:

Let's tackle this labyrinth problem step by step. We have four agents moving through a grid-like labyrinth where edges are either horizontal or vertical, and nodes are labeled as safe or danger. Each agent starts at the same initial node and records their movements based on their current orientation, choosing to go straight, turn left, or turn right relative to their current facing direction.

Step 1: Understanding the Problem:

First, let's summarize the movement sequences of the four agents. Each agent's movement is a sequence of actions (straight, left, right) and the resulting node's status (safe or danger). Our goal is to reconstruct the labyrinth's structure based on these paths.

Agent 1:
1. Starting Point
2. Straight-forward, safe
3. Turn-left, safe
4. Turn-right, safe
5. Turn-left, danger
6. Straight-forward, danger

Agent 2:
1. Starting Point
2. Straight-forward, safe
3. Straight-forward, safe
4. Straight-forward, safe
5. Turn-left, safe
6. Straight-forward, danger

Agent 3:
1. Starting Point
2. Turn-left, safe
3. Turn-right, safe
4. Straight-forward, safe
5. Turn-left, danger

Agent 4:
1. Starting Point
2. Turn-left, safe
3. Straight-forward, safe
4. Straight-forward, safe
5. Turn-right, safe
6. Straight-forward, danger

Step 2: Initial Orientation:

All agents start at the same initial node. We need to assume an initial orientation for the agents. Typically, we can assume they start facing a certain direction, say North (N), unless specified otherwise. The initial orientation is crucial because "turn-left" and "turn-right" are relative to the current facing direction. Let's assume all agents start facing North.

Step 3: Defining Directions:

Let's define the directions with their left and right turns:

- North (N):
  - Left turn: West (W)
  - Right turn: East (E)
- East (E):
  - Left turn: North (N)
  - Right turn: South (S)
- South (S):
  - Left turn: East (E)
  - Right turn: West (W)
- West (W):

- Left turn: South (S)
- Right turn: North (N)

Step 4: Tracing Agent Paths:

Now, let's trace each agent's path step by step, keeping track of their current position and orientation.

Agent 1:

1. Start at origin (0,0), facing N.
2. Straight-forward: move N to (0,1), safe.
   - Now at (0,1), facing N.
3. Turn-left: from N, left is W. Now facing W, move W to (-1,1), safe.
   - Now at (-1,1), facing W.
4. Turn-right: from W, right is N. Now facing N, move N to (-1,2), safe.
   - Now at (-1,2), facing N.
5. Turn-left: from N, left is W. Now facing W, move W to (-2,2), danger.
   - Now at (-2,2), facing W.
6. Straight-forward: move W to (-3,2), danger.
   - Now at (-3,2), facing W.

Agent 2:

1. Start at origin (0,0), facing N.
2. Straight-forward: move N to (0,1), safe.
   - Now at (0,1), facing N.
3. Straight-forward: move N to (0,2), safe.
   - Now at (0,2), facing N.
4. Straight-forward: move N to (0,3), safe.
   - Now at (0,3), facing N.
5. Turn-left: from N, left is W. Now facing W, move W to (-1,3), safe.
   - Now at (-1,3), facing W.
6. Straight-forward: move W to (-2,3), danger.
   - Now at (-2,3), facing W.

Agent 3:

1. Start at origin (0,0), facing N.
2. Turn-left: from N, left is W. Now facing W, move W to (-1,0), safe.
   - Now at (-1,0), facing W.
3. Turn-right: from W, right is N. Now facing N, move N to (-1,1), safe.
   - Now at (-1,1), facing N.
4. Straight-forward: move N to (-1,2), safe.
   - Now at (-1,2), facing N.
5. Turn-left: from N, left is W. Now facing W, move W to (-2,2), danger.
   - Now at (-2,2), facing W.

Agent 4:

1. Start at origin (0,0), facing N.
2. Turn-left: from N, left is W. Now facing W, move W to (-1,0), safe.
   - Now at (-1,0), facing W.
3. Straight-forward: move W to (-2,0), safe.
   - Now at (-2,0), facing W.
4. Straight-forward: move W to (-3,0), safe.
   - Now at (-3,0), facing W.
5. Turn-right: from W, right is N. Now facing N, move N to (-3,1), safe.
   - Now at (-3,1), facing N.
6. Straight-forward: move N to (-3,2), danger.
   - Now at (-3,2), facing N.

Step 5: Mapping the Nodes and Their Statuses:

Now, let's list all the nodes visited by the agents and their statuses:

- (0,0): starting point, status not given (assuming safe)
- (0,1): safe (Agent 1, 2)
- (-1,1): safe (Agent 1, 3)
- (-1,2): safe (Agent 1, 3)
- (-2,2): danger (Agent 1, 3)
- (-3,2): danger (Agent 1, 4)
- (0,2): safe (Agent 2)
- (0,3): safe (Agent 2)
- (-1,3): safe (Agent 2)
- (-2,3): danger (Agent 2)
- (-1,0): safe (Agent 3, 4)
- (-2,0): safe (Agent 4)
- (-3,0): safe (Agent 4)
- (-3,1): safe (Agent 4)

Step 6: Identifying Safe and Danger Nodes:

From the above, the danger nodes are:

- (-2,2)
- (-3,2)
- (-2,3)

All other visited nodes are safe.

Step 7: Verifying Paths for Consistency:

Let's ensure that all agents' paths make sense with these node statuses.

Agent 1:

1. (0,0)
2. N to (0,1) - safe
3. Left from N is W, move to (-1,1) - safe
4. Right from W is N, move to (-1,2) - safe
5. Left from N is W, move to (-2,2) - danger
6. Straight W to (-3,2) - danger

Matches given sequence.

Agent 2:

1. (0,0)
2. N to (0,1) - safe
3. N to (0,2) - safe
4. N to (0,3) - safe
5. Left from N is W, move to (-1,3) - safe
6. W to (-2,3) - danger

Matches given sequence.

Agent 3:

1. (0,0)
2. Left from N is W, move to (-1,0) - safe
3. Right from W is N, move to (-1,1) - safe
4. N to (-1,2) - safe
5. Left from N is W, move to (-2,2) - danger

Matches given sequence.

Agent 4:
1. (0,0)
2. Left from N is W, move to (-1,0) - safe
3. W to (-2,0) - safe
4. W to (-3,0) - safe
5. Right from W is N, move to (-3,1) - safe
6. N to (-3,2) - danger

Matches given sequence.

Step 8: Drawing the Labyrinth:

Let's represent the nodes in a grid based on their coordinates:

```
y=3: ...      (-1,3)S (-2,3)D ...
y=2: ...      (-1,2)S (-2,2)D (-3,2)D ...
y=1: (0,1)S (-1,1)S ...         (-3,1)S
y=0: (0,0)  (-1,0)S (-2,0)S (-3,0)S ...

```

From this, we can see the connections based on the agents' movements. All edges are either horizontal or vertical, and each move is to an adjacent node in the grid.

Step 9: Identifying the Labyrinth Structure:

Based on the paths:
- From (0,0), you can go N to (0,1) or W to (-1,0).
- From (0,1), you can go N to (0,2), or W to (-1,1) (Agent 1 turned left from N to W).
- From (-1,1), you can go N to (-1,2) (Agent 1 turned right from W to N).
- From (-1,2), you can go W to (-2,2).
- From (-2,2), you can go W to (-3,2).
- From (0,2), you can go N to (0,3).
- From (0,3), you can go W to (-1,3).
- From (-1,3), you can go W to (-2,3).
- From (-1,0), you can go W to (-2,0).
- From (-2,0), you can go W to (-3,0).
- From (-3,0), you can go N to (-3,1).
- From (-3,1), you can go N to (-3,2).

This suggests a grid where:
- The x-axis ranges from -3 to 0.
- The y-axis ranges from 0 to 3.

Connections are as per the movements above.

Step 10: Final Labyrinth Structure:

Here's a possible layout of the labyrinth based on the agents' paths:
- Nodes and Connections:
  - (0,0) connected to (0,1) and (-1,0).
  - (0,1) connected to (0,0), (0,2), (-1,1).
  - (0,2) connected to (0,1), (0,3).
  - (0,3) connected to (0,2), (-1,3).
  - (-1,0) connected to (0,0), (-2,0).
  - (-1,1) connected to (0,1), (-1,2).
  - (-1,2) connected to (-1,1), (-2,2).
  - (-1,3) connected to (0,3), (-2,3).
  - (-2,0) connected to (-1,0), (-3,0).
  - (-2,2) connected to (-1,2), (-3,2).
  - (-2,3) connected to (-1,3).
  - (-3,0) connected to (-2,0), (-3,1).
  - (-3,1) connected to (-3,0), (-3,2).
  - (-3,2) connected to (-2,2), (-3,1).
- Danger Nodes:
  - (-2,2)
  - (-3,2)
  - (-2,3)

All other nodes are safe.

Step 11: Visual Representation:

For clarity, here's a simple grid representation (S = safe, D = danger):

```
y=3: ...      (-1,3)S ── (-2,3)D ...
     ...                |
y=2: ...      (-1,2)S ── (-2,2)D ── (-3,2)D ...
     ...                |          |
y=1: (0,1)S ── (-1,1)S ...         ── (-3,1)S
     ...     |          |          |
y=0: (0,0)  ── (-1,0)S ── (-2,0)S ── (-3,0)S ...

```

(Note: Not all connections are shown in this simple representation, but it gives an idea of the structure.)

Conclusion:

By carefully tracing each agent's path, keeping track of their positions and orientations after each move, and noting the status of each visited node, we've reconstructed the labyrinth's structure. The key was to maintain consistent orientations and ensure that all agents' paths align with the same node statuses and connections.

Final Answer:

The labyrinth consists of nodes connected in a grid-like structure with the following properties:
- Nodes:
  - Coordinates range from x = -3 to 0 and y = 0 to 3.
  - Safe nodes: All visited nodes except (-2,2), (-3,2), (-2,3).
  - Danger nodes: (-2,2), (-3,2), (-2,3).

- Connections:
  - Each node is connected to adjacent nodes horizontally or vertically based on the agents' movements.
  - Specific connections can be derived from the agents' paths as detailed above.

This structure satisfies all given movement sequences for the four agents.

VII. CONCLUSIONS

In this paper, we study street networks and building corridor systems, focusing on how they can be affected by disasters such as floods, earthquakes, fires, or hurricanes. When such events damage these networks, it becomes necessary to design new routing systems to ensure safe navigation. To address this, we explore the use of mobile agents, such as robots or first responders, that move randomly through the network. These agents record their trajectories and send the data to a central crisis management center. The key question we investigate is whether these recorded trajectories can be fused together to reconstruct a functional network for emergency routing.

As a first step, we model the street network as a graph, aiming to generate a new graph that meets the necessary requirements for disaster response. We examine two extreme scenarios: a labeled graph, where the network structure is fully known, and an unlabeled graph, where no prior information is available. For the labeled graph, our solution is straightforward, but the unlabeled case presents a more complex challenge, requiring methods to infer the graph's structure. We tested and compared our approach with DeepSeek, an advanced AI model. By framing the problem correctly, DeepSeek produced a solution similar to ours. Additionally, ChatGPT also generated a comparable solution, confirming the robustness of our method.

To demonstrate the feasibility of our approach, we applied it to simple networks, which are easy to generalize to more complex scenarios. The solutions obtained in these test cases are designed in a way that allows for straightforward generalization, ensuring scalability for real-world applications.

REFERENCES

[1] S. Tjiharjadi, S. Razali, H. Asyrani Sulaiman. A systematic Literature Review of Multi-Agent pathfinding for maze Research. Journal of Advances in Information Technology Vol.13, No. 4, 2022

[2] M. Linardakis, I. Varlamis, G. Papadopoulos. Distributed Maze Exploration Using Multiple Agents and Optimal Goal Assignment. IEEE Access. Vol 12, 2024.

[3] A. Marjovi, J. G. Nunes, L. Marques, and A. de Almeida, ''Multi-robot exploration and fire searching,'' in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., Oct. 2009, pp. 1929–1934.E.

[4] E. Kivelevitch, K. Cohen. Multi-Agent Maze Exploration. Journal of Aerospace Computing, Information and Communication Vol. 7, December 2010.

[5] Y. Song. D. Yin Xiang Yue, J. Huang, S. Li, B., Yuchen Lin. Trial and Error: Exploration-Based Trajectory Optimization for LLMAgents. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7584–7600 August 11-16, 2024.

[6] S. Das, P. Flochini, S. Kutten, A. Nayak. Map construction of unknown graphs by multiple agents. Theoretical Computer Science, Volume 385, Issues 1–3, 15 October 2007, Pages 34-48, 2007

[7] G. Dudek, M. Jenkin, E.Milos, D. Wilkes. Robotic exploration as a graph construction, IEEE Transactions on Robotics and Automation · January 1992.

[8] B. Crnković, S. Ivić, M. Zovko. (2023). Fast algorithm for centralized multi-agent maze exploration. arXiv. 2023-10-03. arXiv ID: 2310.02121.

[9] S. Ruan, C. Long, J. Bao, C. Li, Z. Yu, R. Li, Y. Liang, T. He, & Y. Zheng. (2020). Learning to Generate Maps from Trajectories. Proceedings of the AAAI Conference on Artificial Intelligence, 34(01), 890-897. https://doi.org/10.1609/aaai.v34i01.5435

[10] Zhang, J., Alayón Glazunov, A., Yang, W., & Zhang, J. (2021). Fundamental Wireless Performance of a Building. IEEE wireless communications magazine, 1-8. https://doi.org/10.1109/MWC.121.2100244

[11] J. Kim, K. Zheng, J. Corcoran, S. Ahn, M. Papamanolis, Trajectory Flow Map: Graph-based Approach to Analysing Temporal Evolution of Aggregated Traffic Flows in Large-scale Urban Networks, Tech. rep. (2017). Presented at the 96th Annual Meeting of the Transportation Research Board, January 2017. arXiv:2212.02927

[12] L. Rothkrantz, M. van Velden, D. Datcu. Fusion of local maps in mobile ad-hoc networks. Neural Network World 19(1): 67-79

[13] B. Tatomir, L. Rothkrantz, M. Popa. Intelligent system for exploring dynamic crisis environments. Proceedings of the 3rd International ISCRAM Conference, Newark, NJ (USA), May 2006.

[14] L. Rothkrantz, M. Van Velden, and D. Datcu, (2005) A Location based Approach for Distributed World-Knowledge in Mobile Ad-Hoc Networks, Proceedings of the International Conference on Computer Systems and Technologies ComSysTech'05, Varna, Bulgaria.

[15] P. Waasdorp. Distributed Map Construction. Thesis TUDelft, 2010.

[16] L. Rothkrantz. Emergence routing using fishbone technology. European Simulation and Modelling Conference, ESM 2022, pp. 224-228

[17] DeepSeek, http://deepseek.com

[18] ChatGPT, http://chatGPT.com

[19] C. Rothkrantz, S. Fitrianie, L. Rothkrantz. A MOOC for exam training mathematics using intelligent tutoring. Proceedings of the International Conference on Computer Systems and Technologies. 2024.