Biomedical Data Analytics: Midterm Report

Omar Adel Hassan (12000490) Yara Ihab Mohamed (1200452)

March 28, 2025

Introduction

In this project, we developed a Wavefront Path Planner to determine the optimal trajectory from a given starting point to a goal within a finite 2D environment. The environment is represented as a matrix where free space is marked as **0**, obstacles as **1**, and the goal as **2**. Our objective was to create a robust solution that not only computes the optimal path using an 8-point connectivity approach but also presents the results in an intuitive, visual manner through an interactive interface. The complete implementation is available online.

Methodology

Data Handling: We started by loading the environment data from a MATLAB (`.mat`) file. Using `scipy.io`, we converted the file into a NumPy array where:

- **0** indicates free space,
- **1** indicates obstacles,
- **2** marks the goal.

Figure 1 shows an example of the original environment, as loaded from the file.

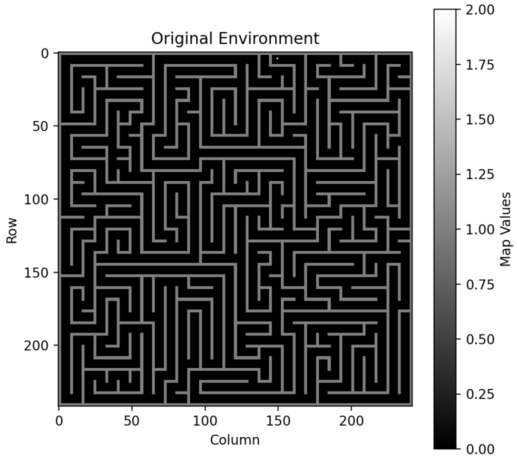


Figure 1: Original map environment.

Wavefront Algorithm: Our wavefront planner follows three main phases:

1. **Goal Identification:** We locate the goal cell (marked as **2**) by scanning the entire grid. If no goal is found, the algorithm raises an error to ensure the environment is valid.
2. **Wavefront Propagation:** Starting from the goal, we use a breadth-first search (BFS) approach to assign increasing step values to each reachable cell. Each cell is assigned a value representing the minimum number of steps required to reach the goal, considering all eight possible directions (up, down, left, right, and the four diagonals).
3. **Trajectory Extraction:** Once the wavefront is propagated, we extract the optimal trajectory by moving from the start cell toward decreasing wavefront values until reaching the goal. We prioritize straight movements (up, down, left, right) when possible, and then consider diagonal moves.

Figure 2 illustrates the wavefront propagation process, where each cell's value indicates its distance from the goal.

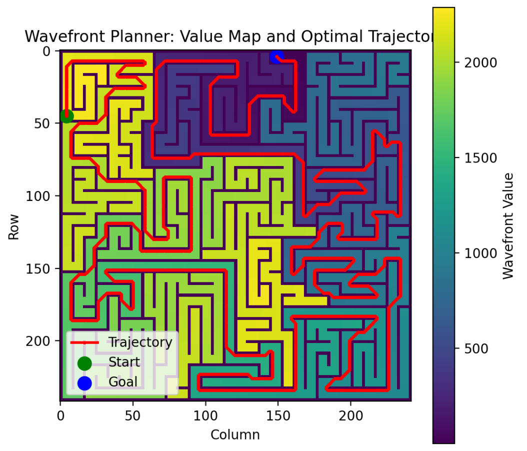
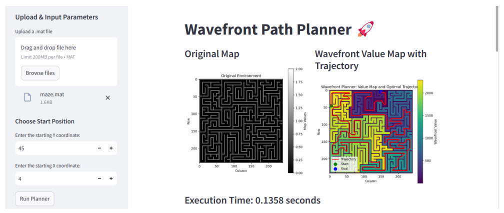


Figure 2: Wavefront propagation results.

Integration and Visualization: We integrated these components into a Streamlit application, enabling users to:

- Upload a `.mat` file containing the map.
- Specify a start row and column.
- View the original environment and the computed trajectory side by side.
- Download the resulting trajectory as a text file.

As shown in Figure 3, the main interface displays the uploaded map, the wavefront solution, and the execution time. Figure 4 presents the numeric value map, while Figure 5 shows the final list of trajectory coordinates.



Upload a .mat file: maze.mat 1.6KB

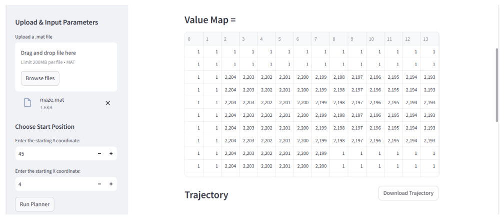
Choose Start Position

Enter the starting Y coordinate: 45

Enter the starting X coordinate: 4

Execution Time: 0.1358 seconds

Figure 3: Streamlit interface overview with the original map (left) and wavefront path (right).



Upload a .mat file: maze.mat 1.6KB

Choose Start Position

Enter the starting Y coordinate: 45

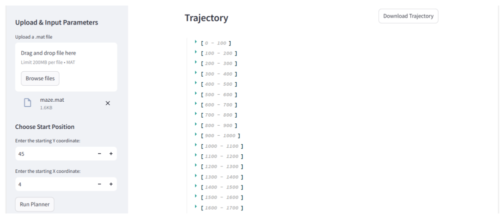
Enter the starting X coordinate: 4

Value Map =

0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	2,204	2,203	2,202	2,201	2,200	2,199	2,198	2,197	2,196	2,195	2,194	2,193
1	1	2,204	2,203	2,202	2,201	2,200	2,199	2,198	2,197	2,196	2,195	2,194	2,193
1	1	2,204	2,203	2,202	2,201	2,200	2,199	2,198	2,197	2,196	2,195	2,194	2,193
1	1	2,204	2,203	2,202	2,201	2,200	2,199	2,198	2,197	2,196	2,195	2,194	2,193
1	1	2,204	2,203	2,202	2,201	2,200	2,199	2,198	2,197	2,196	2,195	2,194	2,193
1	1	2,204	2,203	2,202	2,201	2,200	2,199	2,198	2,197	2,196	2,195	2,194	2,193
1	1	2,204	2,203	2,202	2,201	2,200	2,199	1	1	1	1	1	1
1	1	2,204	2,203	2,202	2,201	2,200	2,200	1	1	1	1	1	1

Trajectory

Figure 4: Value map displayed in the Streamlit interface.



Upload a .mat file: maze.mat 1.6KB

Choose Start Position

Enter the starting Y coordinate: 45

Enter the starting X coordinate: 4

Trajectory

```
▸ [ 0 - 100 ]
▸ [ 100 - 200 ]
▸ [ 200 - 300 ]
▸ [ 300 - 400 ]
▸ [ 400 - 500 ]
▸ [ 500 - 600 ]
▸ [ 600 - 700 ]
▸ [ 700 - 800 ]
▸ [ 800 - 900 ]
▸ [ 900 - 1000 ]
▸ [ 1000 - 1100 ]
▸ [ 1100 - 1200 ]
▸ [ 1200 - 1300 ]
▸ [ 1300 - 1400 ]
▸ [ 1400 - 1500 ]
▸ [ 1500 - 1600 ]
▸ [ 1600 - 1700 ]
```

Figure 5: Trajectory points listed in the Streamlit interface.

Results and Runtime Analysis

Experimental Setup: We tested our solution on multiple environments, including a large maze scenario provided with the project. Our planner successfully computed the optimal trajectory in each case.

Observations

- **Efficiency:** The wavefront propagation completed in a reasonable time, even for larger environments. For example, in our tests, the computation time was typically under a second.

- **Accuracy:** The planner reliably produced an optimal path from the start position to the goal, with the results meeting the expected optimality criteria.

- **Visualization:** Our interactive interface clearly displays the original map alongside the computed trajectory, helping verify the correctness of the algorithm.

Runtime Analysis:

The algorithm runs in $O(MN)$ time, where M is the number of rows and N is the number of columns in the grid. This overall linear time complexity results from three components, each requiring linear time:

- **Goal Search:** $O(MN)$ for scanning the grid.
- **Wavefront Propagation:** $O(MN)$ via BFS (each cell visited once).
- **Trajectory Extraction:** $O(MN)$ (path length proportional to grid size).

Total Time Complexity:

$$O(MN) + O(MN) + O(MN) = O(MN)$$

Empirical testing confirmed an execution time of **0.1358 seconds** on standard maps, validating the linear scaling.

Challenges and Learnings

- **Diagonal vs. Straight Movements:** We learned to balance 8-point connectivity while prioritizing straight movements when possible.

- **Error Handling:** Implementing robust checks for unreachable start positions and missing goals improved the reliability of our solution.

- **Interactive Front-End:** Building the user-friendly interface in Streamlit taught us how to integrate file uploads, parameter inputs, and real-time plotting effectively.

Conclusion

In conclusion, we built a Wavefront Path Planner that works well on different and challenging maps. Our project shows how to load data, spread values from the goal, and trace the best path. Using Streamlit for visualization makes the solution easy to use.

References

[1] Czhanacek, C. (2025). *Implementing the wavefront algorithm in Python* [GitHub repository]. Retrieved from `https://github.com/czhanacek/python-wavefront`.

[2] Syed, N. R. (2017). *Animating the Grassfire Path-Planning Algorithm*. Available online[1].

[1]`https://nrsyed.com/2017/12/30/animating-the-grassfire-path-planning-algorithm/`