Analysis of Intelligent Path Planning using BFS in Complex Environment

T. Sneha, B. Pavithra, N. Vaishnavi, K. Lohith, N. Suresh Kumar
Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University), Visakhapatnam

ABSTRACT
Breadth-First Search (BFS) is a method to explore or search through graphs step by step. In this task, the BFS is used to find the shortest path from source to destination. In the present paper the workflow of BFS is presented node by node by moving level by level and checking all nearby rooms before going deeper. In the present work the time and memory consumed is analyzed during the graph traversing by BFS. The queue is considered to track the search nodes and find the best path. Along the way, also returned which rooms were checked, which paths were avoided, and how BFS helps in finding the shortest route. Finally, the shortest route is identified and calculated the path cost.

Keywords
Breadth-First Search (BFS), path finding, path cost, graph, artificial intelligence

1. INTRODUCTION
Controlling a robot in complex environments is tedious job, especially when it comes to autonomous environment. There are many challenges the robot must face such as obstacle avoidance and moving goal-directed path. At the same time routing the robot in shortest path is also a challenging task for the developer. In the present scenario, a closed building is considered and analysis done for a particular floor with pre-defines nodes and vertices. The route for the destination is unknown to the robot [1].
In the present work, an autonomous vehicle is guided to reach the destination. The vehicle is free to move on the floor and need to explore best path to reach the destination. The grid layout of the floor is represented in figure 1. The path, obstacles, and other objects on the floor are represented in figure 1. The grid layout is converted and considered as unweighted graph for exploring the best path for the vehicle. The Breadth-First Search (BFS) algorithm is suitable for finding the shortest path in non-weighted graphs. Here the BFS is opted, as the vehicle needs to explore all the paths and ensures the shortest route for destination [1][2][3]. In the present scenario the starting location (start node) for the vehicle is considered at room No. 218 and destination (goal node) is room No. 229. The step-by-step directions of the vehicle with respect to the graph are shown in figure 2 in tree structure.
Breadth-First Search (BFS) is one of the most fundamental algorithms in graph theory. It explores nodes (or rooms, in the present context) in a breadthwise manner, meaning it visits all neighbours of a given node before moving deeper into the tree. This systematic approach ensures that BFS is highly effective for finding the shortest path in a tree [5][6]. In the current scenario, the tree is represented by the layout of rooms, corridors, and steps as shown in the Figure 1. Each room is treated as a node, and the connecting corridors or paths represent the edges. BFS will be used to traverse the tree from room 218 (starting node) to room 229 (target node).

2. METRICS OF BFS
2.1 Key Components of BFS
2.1.1 Time Complexity: BFS explores every node and edge once, resulting in a time complexity of **O(V+E)** [7][8]. Where, rooms are represented as vertices and corridors are represented as edges. The edges are connected between vertices. For the present layout, these values are calculated based on the number of rooms and connections.
2.1.2 Space Complexity: BFS uses a queue to store nodes during traversal, leading to a space complexity proportional to **O(V)**. The maximum queue size depends on the breadth of the graph. [7][8].
2.1.3 Nodes Traversed: BFS explores all possible paths systematically until it reaches the target node [9]. In the present case, it is observed the sequence of rooms traversed to reach room 229.
2.1.4 Total Nodes: The algorithm ensures that every accessible node is visited at least once. In the present layout, the total nodes represent the total rooms.

2.2 Mathematical Representation and Steps
2.2.1. Representation:
The layout can be represented as a graph G=(V,E) where V is the set of rooms, and E includes edges connecting the rooms. For example, there is an edge between room 218 and its adjacent rooms (217 and 219) as shown in Figure1.

2.2.2 Algorithm Steps:
Initialization: Start from room 218. Mark it as visited and add it to the queue as shown in figure 3.

- Traversal: Dequeue a room, check if it is room 229, and enqueue all its unvisited neighbours.
- Path Construction: Record the path traversed to backtrack and identify the shortest path.

2.3 Applications of Breadth-First Search (BFS) [5][8]:
2.3.1 Finding the Shortest Path in Unweighted Graphs:
BFS is great for finding the shortest way from one point to another when all moves have the same cost. Since it explores everything at the same level first, it always finds the shortest route. It is mostly used in navigation apps, board games, and etec.,

2.3.2 Peer-to-Peer (P2P) Networks:
In P2P networks, BFS helps find data by searching connected devices step by step. For instance, it is used in finding files in torrent sharing and resource searching in distributed networks.

2.3.3 Web Crawling

Search engines use BFS to scan web pages efficiently. It starts from one page and moves to all linked pages before going deeper. It is mostly used in indexing in webpages, scraping social media connections, and etc.

2.3.4 Social Network Analysis:

BFS helps study relationships in social networks by checking who's connected to whom. For example, finding how two people are connected (like LinkedIn's "degrees of separation"), and Finding groups or communities in social networks.

2.3.5 Solving Mazes and Puzzles:

BFS explores all possible paths in an orderly way, making it perfect for solving mazes and puzzles. It helps in finding the shortest escape route in a maze and Word ladder puzzles.

2.3.6 Network Flow and Connectivity:

BFS is used in network algorithms to figure out the best way to send stuff like water, electricity, or data. It is mainly used in Optimizing traffic or resource flow in cities, and checking the connection of a data center to a resource.

2.3.7 Broadcasting in Networks:

BFS models how messages or updates spread across networks, making sure everyone gets the info. It is used in sending software updates across a distributed system, flooding protocols in wireless networks and etc.

2.3.8 Game AI:

Games use BFS to explore possible moves and find the best options. For instance, the Chess and checkers for evaluating moves for optimization, and Helping characters find the shortest route in grid-based games.

2.4 Merits of BFS [5]:

- **Always Finds the Shortest Path (in Unweighted Graphs)**
  BFS guarantees the shortest path from start to end in an unweighted graph.
- **Always Finds a Solution (if One Exists)**
  If there's a path, BFS will find it.
- **Searches Level by Level**
  BFS checks all nodes at one level before moving deeper.
- **Used in Many Problems**
  BFS helps with pathfinding, network flow, checking bipartite graphs, and finding connected components.
- **Can Be Done Faster with Parallel Computing**
  BFS can run on multiple systems at once to speed things up.
- **Good for Shallow Graphs**
  If a graph isn't too deep, BFS uses less memory.

2.5 Limitations of BFS

- **Uses a Lot of Memory**
  BFS stores all nodes at the current level and their neighbors, which can take up a lot of space in big graphs.
- **Doesn't Work for Weighted Graphs**
  BFS ignores edge weights, so it can't find the shortest path in weighted graphs. Dijkstra's or A* work better.
- **Not Good for Deep Graphs**
  If a graph is very deep, BFS must explore many nodes, making it slow.
- **No Built-in Backtracking**
  BFS doesn't easily go back and change paths, which can be a problem in some cases.
- **Struggles with Infinite Graphs**
  BFS might run forever or use too much memory in very large or infinite graphs.
- **Needs Extra Storage**
  BFS depends on a queue, which can be a problem if memory is limited.

3. METHODOLOGY

Robot Movement from room 218 to destination 229 is explored in the following sections based on Optimized paths with respect to shortest to longest distance. The goal is to move the robot from room 218 to room 229 by following the most efficient paths. Below is a step-by-step breakdown of the three paths, arranged from shortest to longest, along with possible dead ends. The reason for robot traversing via different paths are analysed in the following sections. In these path the robot move from starting point 218 and considered the best path to reach the destination room 229 which is a goal node.

3.1 Path 1 (Shortest Path - Side Path Route)

The sequence of Path1 on the selected floor is in the following order.

👉 218 → 215 → 208 → 209 → 210 → 211 → 228 → 229

Step-by-Step Movement:

1. Move Left from Room 218 to Room 215.
2. Move Left Again to Room 208.
3. Move Up to Room 209.
4. Continue Up to Room 210.
5. Move Up Again to Room 211.
6. Turn Right at Room 211 and enter the Corridor at Room 228.
7. Move Left into goal node 229.

Why This Path?

- Shortest route to goal node 229.
- Avoids the long corridor and reduces travel time.

Dead Ends in Path 1:

- If the robot does not turn at Room 215, it might go towards Entry Gate (207) and get stuck.

If the robot misses the turn at room 211, it might go into a loop.

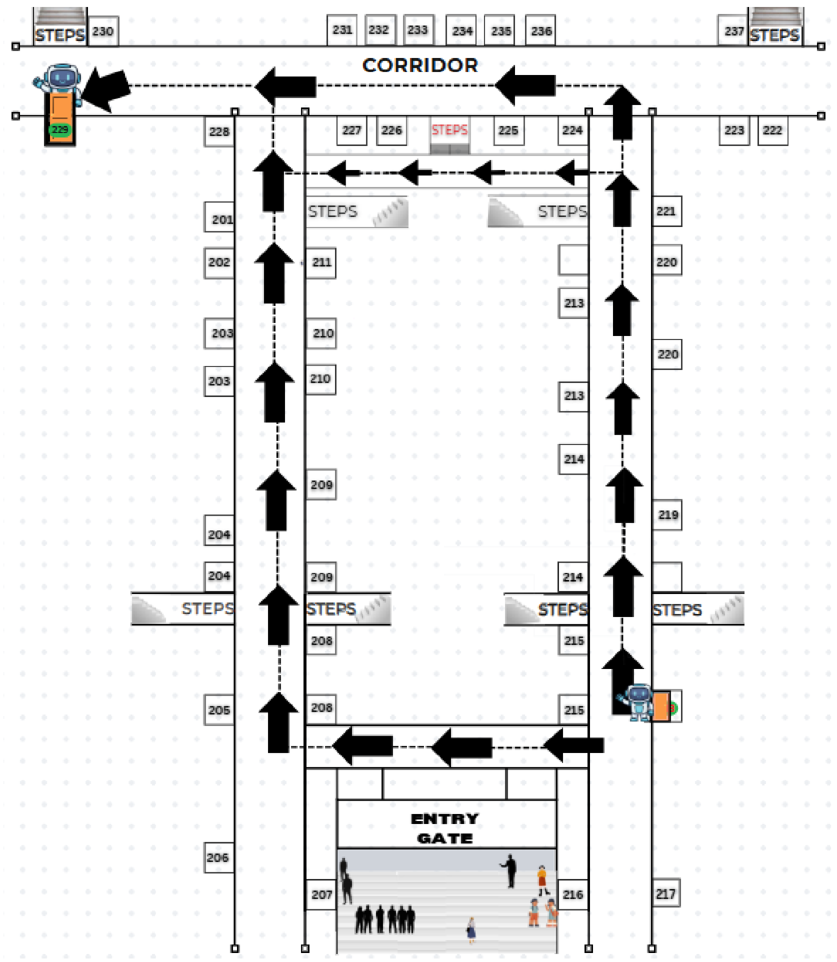

Figure 1 Grid Layout of the floor

3.2 Path 2 (Medium Distance - Steps Shortcut Path)

The sequence of Path2 on the selected floor is in the following order

👉 218 → 219 → 220 → 221 → Steps → 228 → 229

Step-by-Step Movement:

1. Move Up from Room 218 to Room 219.
2. Continue Up to Room 220.
3. Move Up Again to Room 221.
4. Take the Steps near Room 221 (Shortcut to Room 228).
5. Move Left into goal node 229.

Why This Path?

- Faster than Path 3 because it avoids the long corridor.
- Good option if the corridor is blocked or crowded.

Dead Ends in Path 2:

- If the robot misses the steps, it will be forced to follow Path 3, making the journey longer.
-

3.3 Path 3 (Longest Path - Main Corridor Route)

The sequence of Path3 on the selected floor is in the following order

👉 218 → 219 → 220 → 221 → 224 → 225 → 226 → 227 → 228 → 229

Step-by-Step Movement:

1. Move Up from Room 218 to Room 219.
2. Continue Up to Room 220.
3. Move Up Again to Room 221.
4. Turn Left at Room 224 (Enter the Corridor).
5. Move Left passing Room 225, 226, and 227.
6. Turn Left Again at Room 228.
7. Move Left into goal node 229.

Why This Path?

- Most direct but longest due to the corridor path.
- Only use this if the other two paths are blocked.

Dead Ends in Path 3:

- If the robot goes right at Room 224, it will reach a dead end at Room 223.

Table 1: Best Routes & Efficiency

Path	**Distance**	**Efficiency**	**Best Use Case**
Path 1 (Side Path Route via 215, 208, 209, 211)	Shortest	Fastest	Best for quick movement
Path 2 (Steps Shortcut)	Medium	Faster	Use if steps are available
Path 3 (Corridor Route)	Longest	Slowest	Only use if other paths are blocked

Best Path for the Robot is analysed as shown in table 1.

- Path 1 is the best because it is shortest and fastest.
- Path 2 is a good backup if the robot can take the steps shortcut.
- Path 3 is the longest and should only be used if the other two are blocked.

This ensures the most efficient movement while avoiding dead ends.

Key Reasons:

Efficiency: The robot follows the shortest, quickest path, saving time and energy, making it faster and more cost-effective.

Error Prevention: It avoids dead ends and mistakes, ensuring smooth, uninterrupted movement.

Adaptability: The system can easily adjust to new environments or changes, ensuring long-term flexibility.

Backup Routes: In case one path is blocked, the robot can automatically choose another, avoiding delays.

Practical Applications: This project can be used in real-world scenarios like warehouses, hospitals, or delivery systems, making it ideal for industries requiring fast, reliable movement.

4. PATH TRAVERSING

The grid layout of the room is mapped to the directions east, west, north, south, north-east, north-west, south-west, and south-east. So, that it is easy for the robot to select the node in specific direction towards destination. The destination is present at North-west direction. The nodes are considered with respect to these directions. The exploration of these nodes are represented with tree structure for better exploration of robot path as shown in figure 2.

4.1 Navigation Paths of Robot

The navigation of the robot is mapped with respect to the directions such as east, west, north, and south. The best next node in these directions is selected and moves until it reaches to the goal node. The process will be repeated until it analyses all the paths and finds the best path. While moving towards the goal node it considers the directions to select the next best node.

4.1.1 Path: START → NORTH

This path branches into several sub-paths:

- NORTH → NORTH → CONTINUES
- NORTH → NORTH → CONTINUES
- NORTH → NORTH-EAST → DEAD END
- NORTH → NORTH-WEST →DEAD END

NORTH → NORTH → CONTINUES (Further):

- NORTH → WEST → DEAD END
- NORTH → EAST → DEAD END
- NORTH → NORTH-WEST →EAST→ DEAD END
- NORTH → NORTH-WEST →WEST→NORTHWEST→GOAL (successful)
- NORTH → NORTH-WEST →SOUTH-WEST→ DEAD END
- NORTH → NORTH→ WEST →GOAL (successful)
- NORTH → NORTH →WEST→SOUTH→ DEAD END
- NORTH → EAST → DEAD END

4.1.2 Path: START → SOUTH

This path branches into several sub-paths:

1. SOUTH → SOUTH-EAST → DEAD END

4.1.3 Path: START → NORTH-WEST

This path branches into several sub-paths:

1. NORTH-WEST → SOUTH → DEAD END
2. NORTH-WEST → NORTH-EAST → CONTINUES

NORTH-WEST → NORTH-EAST → CONTINUES:

- o NORTH-EAST → NORTH-EAST → CONTINUE
- o NORTH-EAST → EAST → DEAD END
- o NORTH-EAST → EAST → DEAD END
- o NORTH-EAST →WEST → GOAL (successful)

 Dead Ends: 12

 GOAL Nodes Reached: 3

4.2 General Observations and Insights

The robot succeeding movements in different directions are shown in figure 2.

- The diagram figure 2 demonstrates a hierarchical tree structure starting from a single START node.
- The tree splits into multiple branches, each exploring potential directions like NORTH, SOUTH, EAST, WEST, and their combinations.

Key Nodes:

- Dead Ends (X): These are nodes where the exploration terminates without success, represented by red crosses.
- Goal Nodes (Green): There are three successful endpoints where the paths lead to the GOAL, as shown in green.

Branching Characteristics:

- Some paths terminate early (e.g., SOUTH directly leads to a dead end).
- Others extend deeply with multiple sub-paths before reaching a GOAL or a DEAD END.

Total Nodes of Interest:

- Dead Ends: The diagram contains 12 dead ends, where exploration ceases.
- Goals Reached: 3 paths successfully reach the goal.

Optimization Opportunities:

- Paths leading to dead ends could represent unnecessary exploration.

Efficient search strategies can be applied to minimize traversal

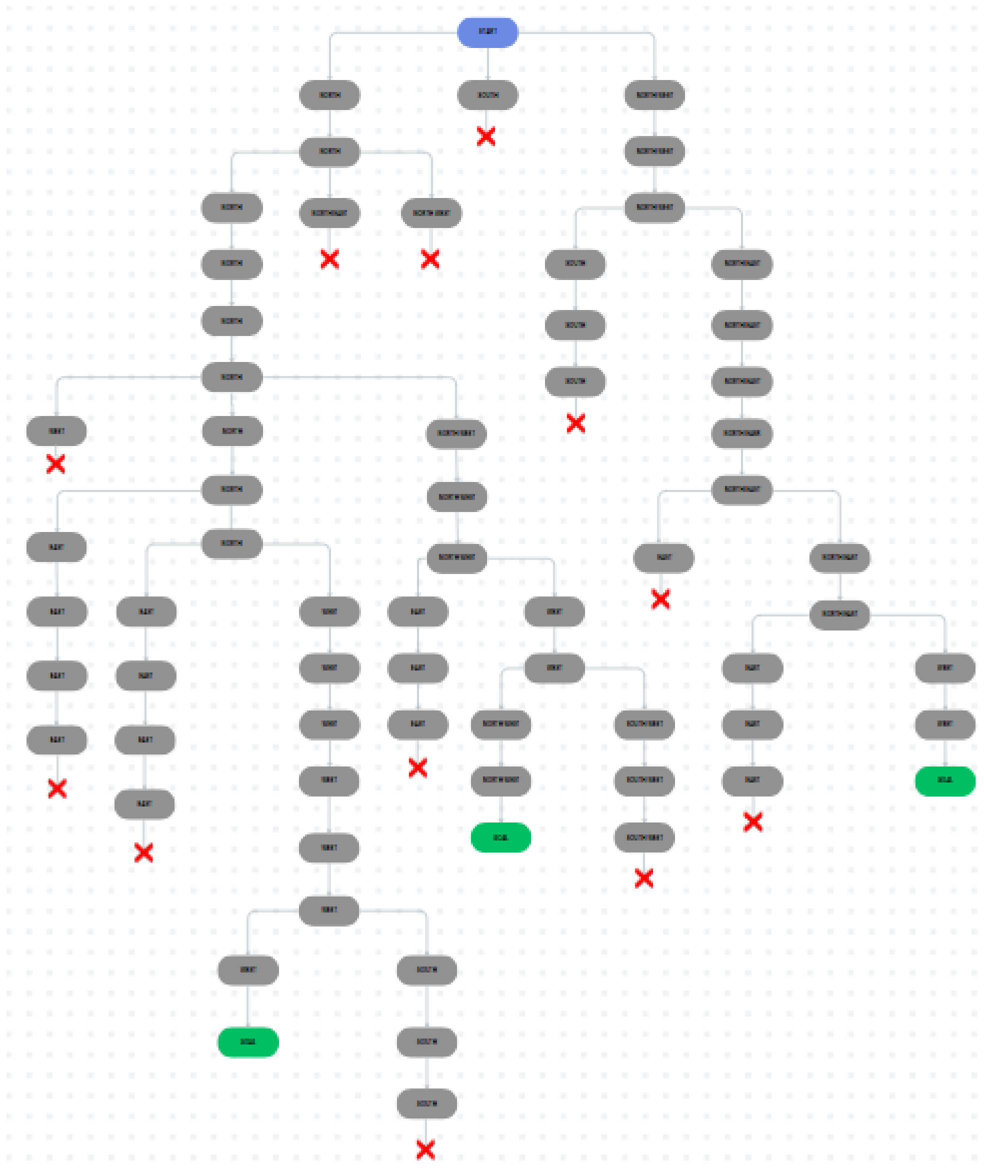

Figure 2. Path traversing in Tree structure

5. QUEUE

The possible node selections are done through queue data structure. The node which existent for long time or entered first is removed from the queue and returned for robot path. The order of inserting and removing nodes from the queue is represented in figure 3. The queue starts with the start node being enqueued. Upon dequeuing start, the possible moves north, south, and northwest are enqueued. Next, north is dequeued, enqueueing of south and northwest. When south is dequeued, northeast is added to the queue. Then, northwest is dequeued, resulting in northeast and south being enqueued. Continuing, south is dequeued, and northwest is added again. The next step involves dequeuing northwest, leading to the enqueueing of northeast. When northeast is dequeued, east and west are enqueued. After that, south is dequeued, adding northwest to the queue. The next dequeued node, northeast, results in west and east being enqueued. Northwest is then dequeued, enqueuing another northwest. When northeast is dequeued, it leads to the enqueueing of east and west. East is then dequeued, adding west to the queue. West is then dequeued, which results in the goal node being enqueued.

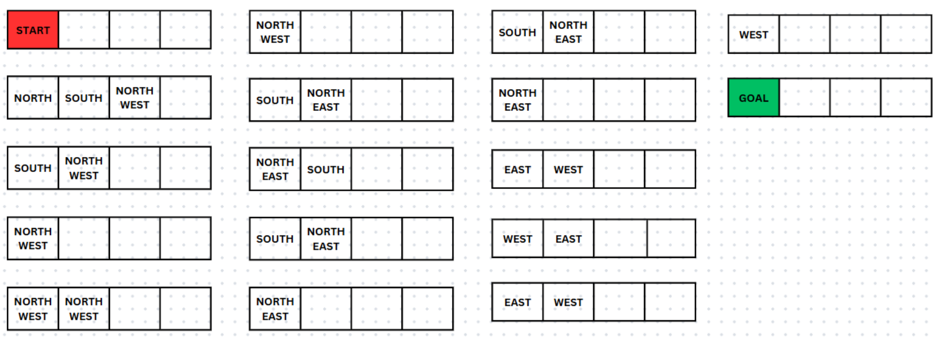

Figure 3. enqueue and Dequeuing the nodes

Heuristic values:

The heuristic values are assigned based on the direction as shown below. The North and west are assigned highest heuristic value, and South and East are assigned lower heuristic value. Based on the pre-assigned heuristic values the heuristic values for individual paths are calculates as follows. For path1 the heuristic value calculated as 70, path2 is calculated as 75, and path3 is calculated as 90. The individual node calculations with respect to path distances are shown below.

Path 1:(North-West): It takes 12 nodes to reach the goal node and calculated the path cost as below.

10+10+10+5+5+5+5+5+5+5+5→Goal (70)

Path 2:(North): It takes 13 nodes to reach the goal node and calculated the path cost as below.

5+5+5+5+5+5+5+5+5+5+5+5+5+5+5→Goal (75)

Path 3:(North): It takes 16 nodes to reach the goal node and calculated the path cost as below.

5+5+5+5+5+5+10+10+10+5+5+10+10→Goal (90)

From the above analysis the robot consideres the path1 as best path with lower patch cost.

5.1 Time Complexity

Time complexity depends on how the robot navigates the path.

Practical Time Complexity for 218 → 229 via 208

1. Number of Steps (Nodes Visited)

The robot follows 8 steps: 218 → 215 → 208 → 209 → 210 → 211 → 228 → 229

Each step (moving between rooms) takes constant time O(1).

Total time = 8 × O(1) = O(N), where N is the number of rooms visited.

2. Algorithm-Based Complexity

BFS (Breadth-First Search): It Checks all possible moves level by level.

Time Complexity: O(N) for a single shortest path, but O(V + E) if searching all paths.

3. Real-World Execution Time

Each step (movement between rooms) might take 0.5 seconds.

Total movement time ≈ 8 × 0.5s = 4 seconds (assuming no delays).

Final Practical Time Complexity

Direct path: O(N) = O(8) (Linear Time)

For a large map search: O(V + E) (Graph Traversal Time)

Real-time execution: ~4 seconds (for 8 steps, assuming 0.5s per step).

5.2 Space Complexity:

Practical Space Complexity Calculation for 218 → 229 via 208

1. Memory Used for Storing the Path

The robot follows the path: 218 → 215 → 208 → 209 → 210 → 211 → 228 → 229 (8 rooms).

Each room (node) can be stored as a fixed-size structure (e.g., room ID, coordinates).

If each room takes 4 bytes, then storing the path needs:

8 rooms × 4 bytes = 32 bytes

2. Memory Used for Algorithm Execution

BFS (Breadth-First Search):

Uses a queue to track visited rooms.

Worst-case memory: O(W) (width of the graph).

If maximum simultaneous storage is 4 rooms, it takes 4 × 4 = 16 bytes.

3. Other Memory Considerations

Visited rooms List (to avoid re-visiting): 8 × 1 byte = 8 bytes

Additional Variables (pointers, counters, etc.): ~8 bytes

Total Practical Space Usage

BFS: ~32 bytes (lower because queue is managed efficiently)

Thus, the practical space complexity remains O(N) (linear) and small (32 bytes for this path).

6. CONCLUSION

Breadth-First Search (BFS) is a simple way to explore complex graphs step by step. It always finds the shortest path in unweighted graphs. In the present work, BFS found three different paths from source (room 218) to goal (room 229) by checking each level one by one. This made sure no path was missed. BFS is useful for finding the best way to move in small spaces. But it has some problems, like using a lot of memory and not working well with weighted paths. Even with these issues, BFS is still a good choice for finding short paths in mazes, networks, and maps. The path optimization can be further optimized with hybrid model.

7. REFERENCES

[1] B. S. Harapan, Pencarian Shortest Path Dinamik dengan Algoritma Bellman Based Flood Fill dan Implementasinya pada Robot Micromouse, Institut Teknologi Bandung, 2009.

[2] I. Elshamarka and B. S. S. Abu, Design and Implementation of a Robot for Maze-Solving using Flood-Fill Algorithm, Universiti Teknologi Petronas. 2012.

[3] R. K. Sreekanth, “Artificial intelligence algorithms,” IOSR Journal of Computer Engineering, vol. 6, no. 3 September-October 2012.

[4] D. Zai, H. Budiati and S.B. Berutu, "Simulation of the Shortest Route to Tourism Locations in Nias Using the Breadth-FirstSearch and Tabu Search Methods", InFact Journal ,Vol. 1 No. 2, Nov. 2016.[2] B. Fu et al., "An improved A* algorithm for the industrial robot path planning with high success rate and short length,"Rob. Auton. Syst., vol. 106, pp. 26–37, 2018.

[5] S. K. Debnath et al., "A review on graph search algorithms for optimal energy efficient path planning for an unmanned airvehicle," Indones. J. Electr. Eng. Comput. Sci., vol. 15, no. 2, pp. 743–750, 2019.

[6] G. L. Andrade and D. H. Thomas, "An Optimized Breadth-First Search Algorithm for Routing in Optical AccessNetworks," IEEE Lat. Am. Trans., vol. 17, no. 7, pp. 1088–1095, 2019.
[7] R. Zhou and E. A. Hansen, "Breadth-first heuristic search," Artif. Intell., vol. 170, no. 4–5, pp. 385–408, 2006.

[8] D. C. Kozen, "Lecture 4. Depth-First and Breadth-First Search," in The Design and Analysis of Algorithms, Springer, NewYork, NY, 2011.

[9] F. Zhang et al., "An adaptive breadth-first search algorithm on integrated architectures," J. Supercomput., vol. 74, no. 11,pp. 6135–6155, 2018.

[10] A. N. Putri, "Search the blind breadth first search algorithm in 3d game engine maze third person shooter android based onintelligent agent". Transformatika Journal, Vol. 14, No. 1, Jul. 2016.