Artificial Intelligence

Assignment 1

(Question 7)

Prepared By:

Ajmal Razaq (23F-0524)
Rania Shoaib (23F-0650)

GitHub Repository:

https://github.com/theajmalrazaq/searchalgo

1 Introduction

This report documents the implementation and analysis of six fundamental uninformed search algorithms in a grid environment. The task involves navigating a grid from a start position (S) to a target position (T) while avoiding static walls. The grid supports 6-directional movement: Up, Right, Bottom, Left, and the main diagonals (Bottom-Right and Top-Left).

2 Search Strategies Implementation

2.1 Movement Order

To ensure consistency across algorithms, neighbors are processed in a strict clockwise order:

1. Up $(-1, 0)$
2. Right $(0, 1)$
3. Bottom $(1, 0)$
4. Bottom-Right (Diagonal) $(1, 1)$
5. Left $(0, -1)$
6. Top-Left (Diagonal) $(-1, -1)$

2.2 Breadth-First Search (BFS)

BFS explores the grid level by level using a queue (FIFO). It guarantees the shortest path in terms of steps in an unweighted grid.

- **Pros:** Complete and optimal (finds the shortest path).
- **Cons:** High memory usage as it stores all nodes in the current frontier.

2.3 Depth-First Search (DFS)

DFS explores as deep as possible before backtracking, using a stack (LIFO).

- **Pros:** Low memory consumption compared to BFS.
- **Cons:** Not optimal (may find a very long path) and can get stuck in deep branches if not limited.

2.4 Uniform-Cost Search (UCS)

UCS expands nodes based on the cumulative path cost using a priority queue. In this implementation, diagonal moves cost $2 \approx 1.414$ while straight moves cost 1.0.

- **Pros:** Optimal for weighted grids (finds the path with the lowest cost).
- **Cons:** Can be slow as it explores many nodes with similar costs.

2.5 Depth-Limited Search (DLS)

DLS is a version of DFS that stops exploring once a specific depth limit is reached.

- **Pros:** Prevents infinite depth exploration.
- **Cons:** Not complete if the target is beyond the limit; not optimal.

2.6 Iterative Deepening DFS (IDDFS)

IDDFS repeatedly applies DLS with increasing limits (0, 1, 2, ...).

- **Pros:** Combines the memory efficiency of DFS with the completeness and optimality of BFS.
- **Cons:** Re-explores nodes in each iteration, which adds some computational over- head.

2.7 Bidirectional Search

Bidirectional search runs two simultaneous BFS searches: one from the start and one from the target. The search stops when the two frontiers intersect.

- **Pros:** Drastically reduces the search space ($2 \times b^{d/2}$ vs b^d).
- **Cons:** Requires keeping track of two frontiers and intersection points.

3 Test Cases (Visual Proof)

3.1 Breadth-First Search (BFS)

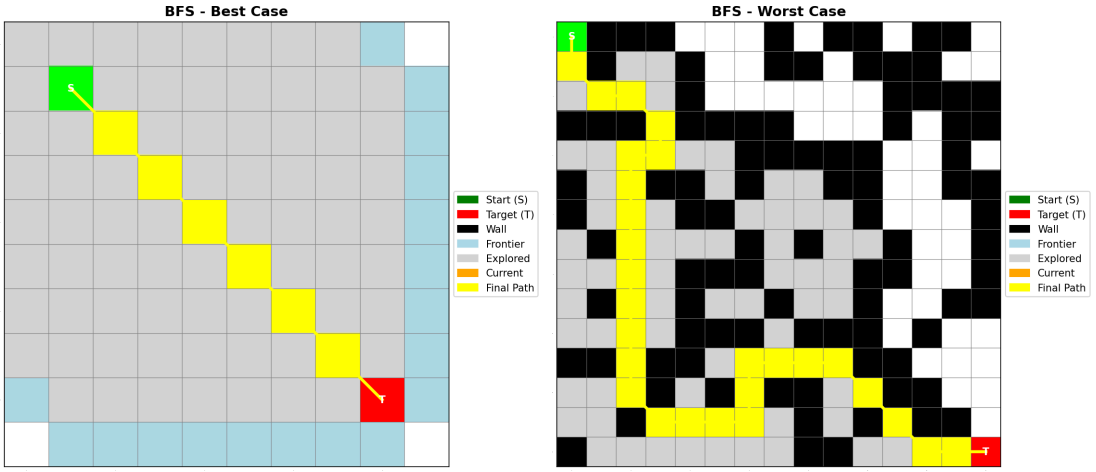

Figure 1: BFS Best Case (Left) and Worst Case (Right)

3.2 Depth-First Search (DFS)

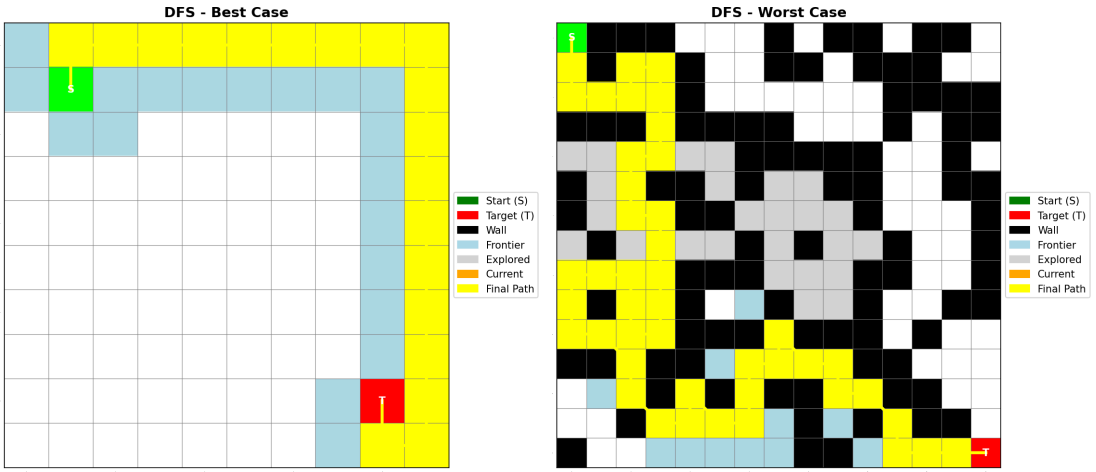

Figure 2: DFS Best Case (Left) and Worst Case (Right)

3.3 Uniform-Cost Search (UCS)

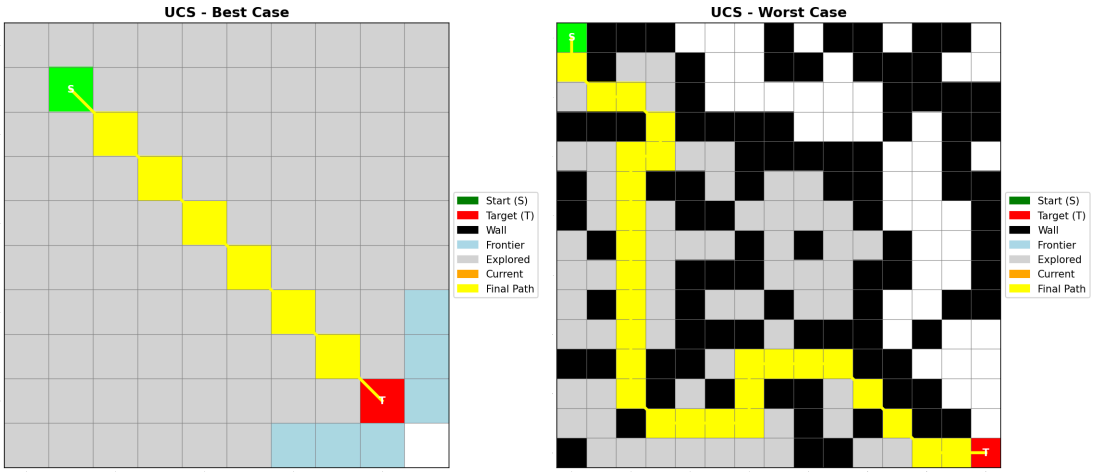

Figure 3: UCS Best Case (Left) and Worst Case (Right)

3.4 Depth-Limited Search (DLS)

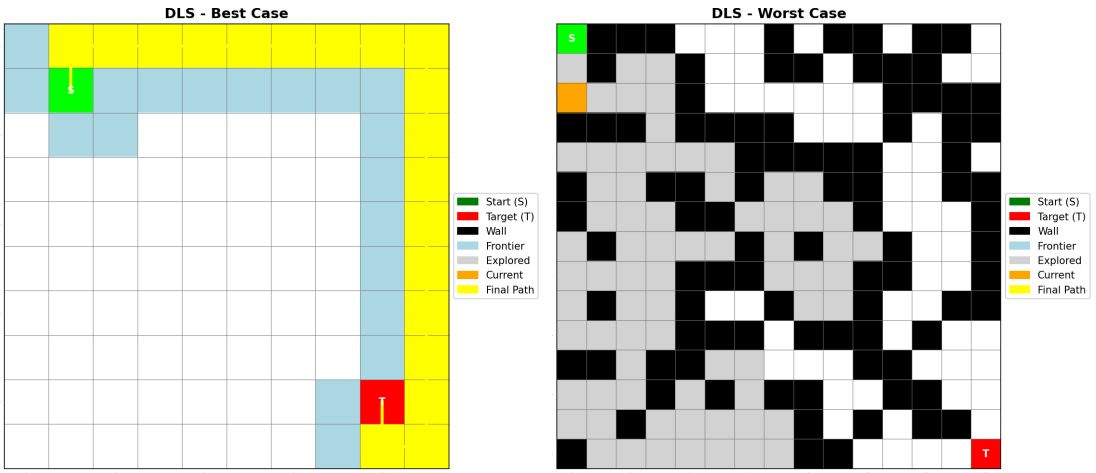

Figure 4: DLS Best Case (Left) and Worst Case (Right)

3.5 Iterative Deepening DFS (IDDFS)

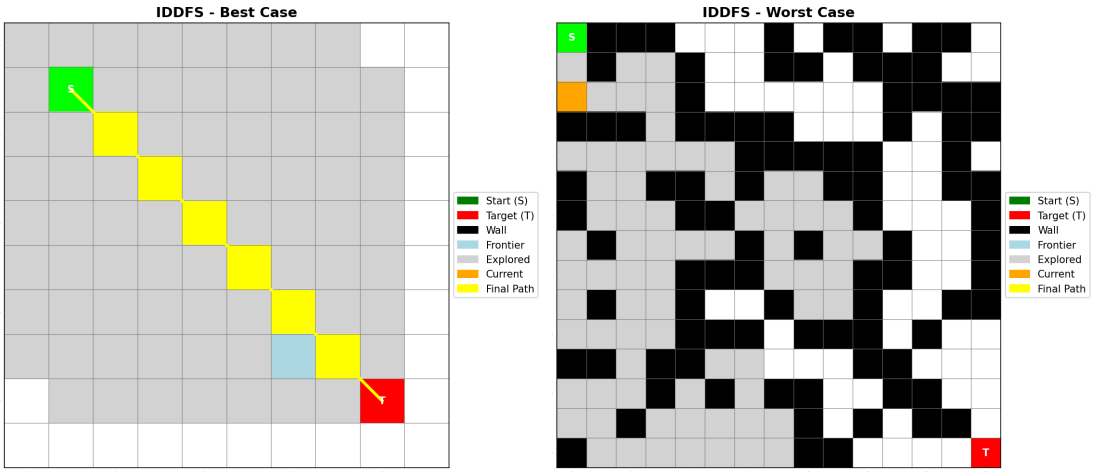

Figure 5: IDDFS Best Case (Left) and Worst Case (Right)

3.6 Bidirectional Search

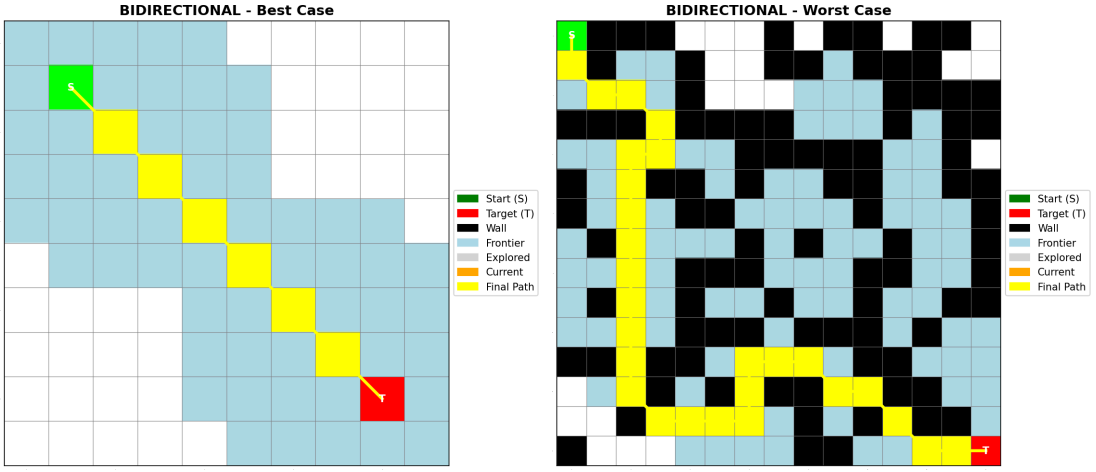

Figure 6: Bidirectional Search Best Case (Left) and Worst Case (Right)

4 Conclusion

The implementation successfully visualizes the differences between uninformed search strategies. BFS and Bidirectional search were found to be most effective for finding direct paths, while DFS and UCS showed their unique exploration patterns based on stack depth and cumulative cost respectively.