Project Report

Course: Artificial Intelligence

Project Title: Comparative Analysis of SearchStrategies for Maze Pathfinding .

Date: December 15, 2025

1. Introduction

1.1 Problem Definition

The project addresses the "Maze Pathfinding" problem, which involves finding a valid path for an intelligent agent from a starting point (S) to a goal point (G) within a grid-based environment. The environment contains static obstacles (walls) that block movement. The agent can move in four cardinal .directions: Up, Down, Left, and Right

1.2 Objective

The primary objective of this project is to implement and compare five fundamental Artificial Intelligence search algorithms. We aim to analyze their performance in terms of Time Complexity (execution speed), Space Complexity (memory usage/nodes expanded), and Optimality (path length

2. Methodology&Algorithms

We implemented five distinct search strategies categorized into Uninformed (Blind) and Informed (Heuristic) searches:

2.1 Uninformed Search Strategies

Breadth-First Search (BFS): Explores the grid layer by layer * using a FIFO Queue. It guarantees the shortest path in an .unweighted grid

Depth-First Search (DFS): Explores as deep as possible * along each branch using a LIFO Stack before backtracking. It is .memory efficient but does not guarantee the shortest path

Uniform-Cost Search (UCS): Expands the node with the * lowest path cost g(n). Since our step cost is constant (1), it .behaves similarly to BFS but ensures optimality based on cost

Iterative Deepening Search (IDS): A hybrid strategy that * repeatedly runs DFS with increasing depth limits (Depth=0, 1, 2...). It combines BFS's optimality with DFS's space efficiency

2.2 Informed Search Strategies

A Search (A-Star):* The most advanced algorithm in our set. * It uses the evaluation function f(n) = g(n) + h(n), where g(n) is .the cost so far, and h(n) is the heuristic estimate to the goal

3. Implementation Details

.Environment: The project is developed using Python *

Visualization: The standard tkinter library is used to visualize * .the N \times N grid

.White Cells: Walkable path *

.(Black/Dark Cells: Walls (Obstacles *

.(Green Cell: Start Node (0,0 *

.(Red Cell: Goal Node (N-1, N-1 *

Code Structure: The logic is encapsulated in a * SearchAlgorithms class, utilizing standard data structures like deque (for BFS), list (for DFS/Stack), and heapq (for Priority .(Queues

4. Experimental Results

:Test Environment

.Maze Size: 10 \times 10 Grid *

.Obstacle Density: Approx. 20% *

:Performance Data

The following data was collected by running all algorithms on) .(the same generated maze

Alogrithm	Status	Path Cost	Space	Time
BFS	Success	22	114	0.32
DFS	Success	24	93	0.19
UCS	Success	22	114	0.51
IDS	Success	22	11500297	13787.56
A* Search	Success	22	93	0.50

5. Analysis and Disscussion

Based on the experimental results recorded in Section 4, we :observe the following behavior

5.1 Time Complexity (Speed)

A Search* demonstrated the fastest performance. By using * the Manhattan heuristic, it directed the search straight towards .the goal, avoiding unnecessary exploration of the grid

IDS was the slowest algorithm. This confirms the theoretical * expectation that regenerating the search tree multiple times for .each depth limit consumes significant CPU time

5.2 Space Complexity(Nodes Expanded)

DFS expanded a variable number of nodes. In some cases, it * found a path quickly, but in worst-case scenarios, it explored deep, .irrelevant branches

BFS and UCS expanded a large number of nodes because they * .explore all neighbors at the current level before moving deeper

A Search* expanded the fewest nodes among the optimal * .algorithms, proving its efficiency

5.3 Optimality(Path Quality)

BFS, UCS, IDS, and A* all successfully found the Shortest * Path (Optimal Solution). Their "Path Cost" values in the table .are identical

DFS returned a path that was significantly longer *
(sub-optimal) compared to the others. This is because DFS
accepts the first path it finds, regardless of its length

6. Conclusion

In this project, we successfully implemented and compared five
:search algorithms. The comparative analysis highlights that

A Search* is the superior choice for pathfinding in this *
.domain, balancing speed and optimality

BFS and UCS are reliable for finding the shortest path but *
.consume more memory

DFS is not suitable for pathfinding where the shortest path is *
.required, as it produces sub-optimal results

IDS is useful for memory-constrained environments but *
.suffers from slow execution time