BFS, DFS, and UCS Analysis Report

Project 1 – Grid Search Algorithms

1. Problem Formulation

State Space

Each state is a grid cell represented as a tuple **(row, column)**. The grid has dimensions **m × n**, and all states lie within the bounds:

$0 \leq row < m$

$0 \leq column < n$

Initial State

The start state is always:

(0, 0)

Actions

From any state, the agent may attempt one of four actions:

- **U** (up)
- **D** (down)
- **L** (left)
- **R** (right)

An action is only valid if it results in a state that remains inside the grid boundaries.

Transition Model

The transition function applies an action to the current state using a deterministic move(state, action) function. Each legal action produces exactly one successor state.

Goal Test

A state is a goal state if:

state == goal

The goal is assigned by the experiment runner and corresponds to the **top-right cell**:

(m − 1, n − 1)

Cost Model

Costs are associated with **actions (edges)** rather than states.
For each (state, action) pair, a random cost is generated using:

buildCosts(m, n, min_cost, max_cost, seed)

- min_cost = 1
- max_cost = 20

For a given seed, the cost map is fixed and shared across BFS, DFS, and UCS, ensuring fair comparisons.
The total path cost is the sum of all action costs along the solution path.

---

2. Algorithm Implementations

All algorithms strictly follow the pseudocode provided in the course slides and operate as **graph searches** using an explored set to avoid revisiting states.

Breadth-First Search (BFS)

BFS uses a **FIFO queue** as its frontier and expands nodes in order of increasing depth. Each action has equal step cost, so BFS guarantees the shortest path in terms of **number of steps**.

- Frontier: Queue
- Explored: Set of visited states
- Optimality: Yes (for unit step cost)

Depth-First Search (DFS)

DFS uses a **stack** as its frontier and explores one branch as deeply as possible before backtracking. It does not consider step count or cost when selecting nodes.

- Frontier: Stack
- Explored: Set of visited states
- Optimality: No

DFS's behavior depends heavily on successor ordering and can result in very long or inefficient paths.

Uniform-Cost Search (UCS)

UCS uses a **priority queue ordered by cumulative path cost g(n)**. Nodes with the lowest total cost are expanded first. A best-cost map is maintained so that a state is only re-expanded if a cheaper path is found.

- Frontier: Priority Queue (min-heap)
- Key: cumulative cost g(n)

- Optimality: Yes (for non-negative costs)

UCS may take more steps than BFS, but it guarantees the minimum-cost solution.

---

3. Testing Summary

Testing was implemented using **pytest** and executed in **WSL**.
All three algorithms were tested using structurally similar test suites.

The tests verified:

- Returned paths start at the initial state and end at the goal
- All states in the path remain within grid bounds
- Each transition corresponds to a legal action (U/D/L/R)
- Paths contain no repeated states (no cycles)
- Edge cases:
    - Start equals goal
    - Unreachable or out-of-bounds goal returns failure
- Metrics correctness and sanity:
    - Required fields are present
    - All metric values are non-negative
- **UCS only**:
    - Reported total cost matches a recomputed cost using the cost map

All tests passed for BFS, DFS, and UCS.

---

4. Experimental Results

Experiments were run on **10×10, 25×25, and 50×50 grids**, using seeds **1–10**.
Each algorithm ran **10 times per grid size**, for **90 total runs**.

Aggregated Metrics (Means)

m	n	algorithm	steps	cost	expanded	max frontier	runtime (ms)
10	10	BFS	18.0	193.9	99.0	37.0	0.133
10	10	DFS	90.0	952.4	90.0	196.0	0.131

m	n	algorithm	steps	cost	expanded	max frontier	runtime (ms)
10	10	UCS	18.2	115.4	98.7	71.7	0.274
25	25	BFS	48.0	499.8	624.0	97.0	0.868
25	25	DFS	624.0	6523.0	624.0	1452.0	0.993
25	25	UCS	49.2	261.9	623.2	214.3	1.688
50	50	BFS	98.0	1054.2	2499.0	197.0	3.493
50	50	DFS	2450.0	25670.1	2450.0	5956.0	4.391
50	50	UCS	100.6	538.0	2497.7	435.7	8.145

---

5. Discussion

Why BFS Finds Short Paths but Uses More Memory

BFS expands the grid level by level, guaranteeing the shortest path in terms of steps.
In this grid setup, the shortest path length closely matches the Manhattan distance from (0,0) to (m−1,n−1), which explains the nearly fixed step counts.

However, BFS must store all frontier nodes at the current depth, causing memory usage to grow rapidly with grid size.

Why DFS Is Unpredictable

DFS does not optimize steps or cost. It may reach the goal quickly or take a long detour depending on exploration order.
This behavior is reflected in the extremely large step counts and total costs, especially on larger grids.

Why UCS Finds Lower-Cost Paths

UCS prioritizes cumulative cost rather than step count. When action costs vary, UCS avoids expensive transitions even if this requires more steps.
Although UCS generally runs slower due to priority queue operations, it consistently finds paths with significantly lower total cost than BFS.

---

6. Worked Example (Required Comparison – Section 7.2)

50×50 Grid, Seed = 10

- **BFS**
    - Steps: 98
    - Total cost: 1028
- **UCS**
    - Steps: 102
    - Total cost: 557

BFS reaches the goal using fewer steps but incurs a much higher total cost because it ignores action costs. UCS takes a slightly longer path but minimizes cost by avoiding expensive actions. This example clearly demonstrates the tradeoff between step optimality and cost optimality.

---

7. Summary Comparison

Algorithm	Optimizes	Strength	Weakness
BFS	Steps	Shortest path in steps	High memory usage
DFS	None	Simple, low overhead	Unpredictable, inefficient paths
UCS	Cost	Lowest-cost solution	Slower, larger frontier

---

8. AI Disclosure Appendix

AI tools were used to assist with **test design**, **analysis wording**, **experiment configuration**, and **metric reporting**.

Example prompts included:

- Generating pytest test cases for BFS, DFS, and UCS
- Producing an analysis from aggregated metrics
- Creating an experiment configuration file

All algorithm implementations were written by the author following course pseudocode. AI-generated content was reviewed line-by-line, simplified when necessary, and validated by confirming that reported metrics matched the generated CSV results. More details can be found in the README.md and Analysis_Report.md.