Vacuum Cleaner Problem

Problem Description

The Vacuum Cleaner Problem is a classical problem in Artificial Intelligence used to demonstrate search algorithms and intelligent agent behavior

The environment consists of a two-dimensional grid representing a room divided into multiple cells. Each cell can be in one of two states: Clean or Dirty. A vacuum cleaner agent is initially located at a specific starting position within the grid

Why This Problem?

The Vacuum Cleaner problem is chosen because it represents a simple yet effective model for understanding fundamental concepts in Artificial Intelligence, particularly search algorithms and intelligent agents. Despite its simplicity, the problem captures essential aspects of real-world decision-making, such as navigating an environment, selecting appropriate actions, and achieving a goal efficiently.

Depth-First Search: (DFS) عبدالرحمن اسماعيل يوسف حنوره

Mechanisms of expanding nodes: Branch by Branch expand a

deepest node first

Implementation: Fringe is a LIFO stack

It is not optimal it finds the “leftmost” solution, regardless of depth or cost

Saves more memory than BFS

Breadth-First Search: (BFS) عبدالرحمن أسامه محمد مخيمر

Mechanisms of expanding nodes: Level by Level. From Left to Right.

Implementation: Fringe is a FIFO queue

It is optimal only if costs are all 1 (more on costs later) and Consume Large Memory because of the waiting list.

Iterative-Deepening Search: (IDS) عمرو خالد أحمد نصير

It's like a loop, and I have a variable called 'Iteration' it starts from 0 and keeps increasing by 1 (++Iteration), the loop stops when it reaches the goal.

It's the same as DFS, but it goes step by step until it finds the goal.

Uniform-Cost Search: (UCS) عبدالرحمن أحمد عبده الهبيان

Strategy: expand a cheapest node first

Fringe is a priority queue (priority: cumulative cost)

It works in the same way as BFS it only differs in that it expands the node with the lowest cost first.

A* Search algorithm: (A*) عمر هاشم رزق الديرى

The most common informed search algorithm is A* search, a bestfirst search that uses: f(n) = g(n) + h(n)

Uniform-cost orders by path cost, or backward cost g(n)

Greedy orders by goal proximity, or forward cost h(n) A*

is cost-optimal (returns only cost-optimal paths).

Algorithm Comparison Table

Algorithm	Time Complexity	Space Complexity	Execution Time (Observed)	Optimal Solution	complete
IDS	O(b^d)	O(b × d)	Medium to High	Yes	Yes
DFS	O(b^m)	O(b × m)	Low	No	No
BFS	O(b^d)	O(b^d)	Medium	Yes	Yes
UCS	O(b^(C*/ε))	O(b^(C*/ε))	Medium to Low	Yes	Yes
A*	O(b^d) in worst case	O(b^d)	Very Low (Fastest)	Yes (with admissible heuristic)	Yes

Execution Time Example Output

Algorithm	Execution Time (seconds)
IDS	0.50 – 1.20 (0.166207)
DFS	0.05 – 0.15(0.001086)
BFS	0.20 – 0.60(0.008811)
UCS	0.15 – 0.40 (0.006980)
A*	0.01 – 0.10 (0.000119)

Final Recommendation

Goal	Best Algorithm
Fastest execution	A*
Guaranteed optimal path	A* / UCS
Low memory usage	IDS
Simple implementation	DFS