Optimizing Autonomous Agents in

# Vacuum World

Group 1

**2026**
Artificial Intelligence

# Table of Contents


#1 Introduction & Problem Specification

#2 Search Problem Formalization

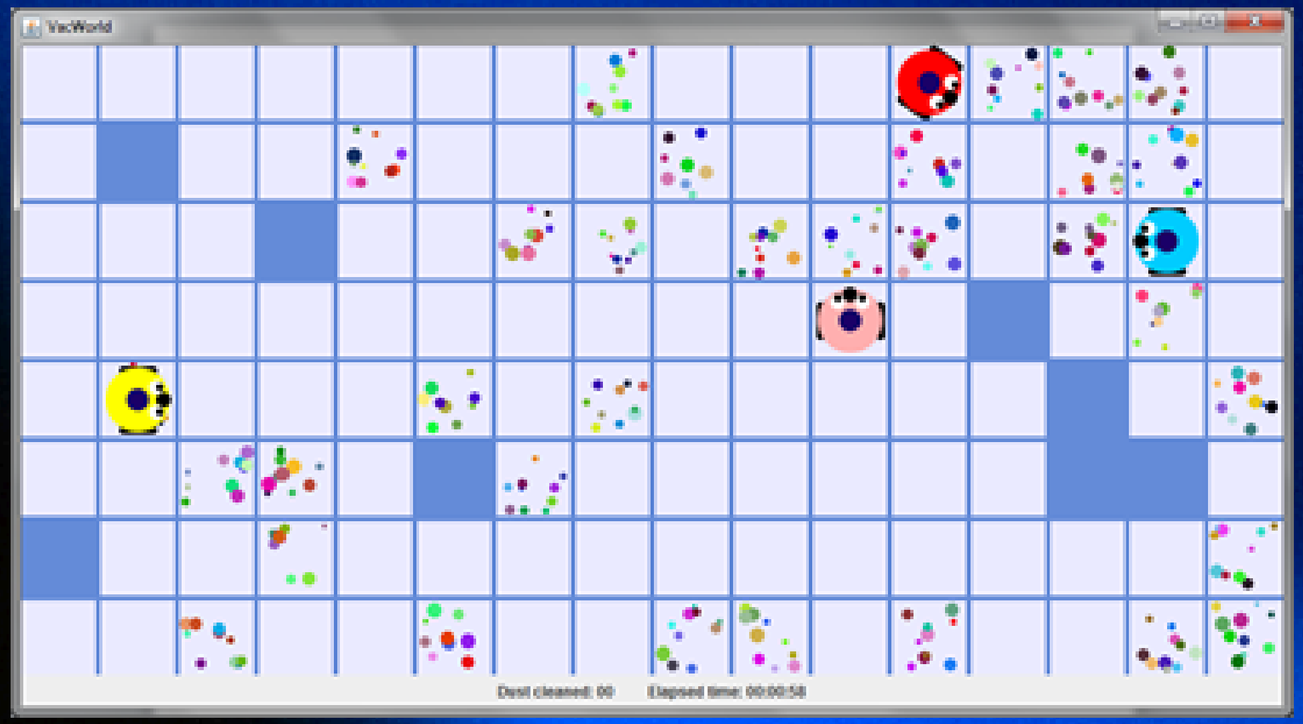
#3 Search Tree & Uninformed Strategies

#4 Informed Search & Heuristics

#5 Experimental Results & Comparative Analysis

#6 Conclusion & Future Work

# Why Vacuum World




A classic AI benchmark for rational agents.

Demonstrates core concepts: State space, Search trees, and Heuristics.

Goal

- Autonomous cleaning with efficiency (path length vs. time).

Challenges

- Exponential state growth as grid size *(n x n)* increases.

# Agent Specification:
# PEAS Framework

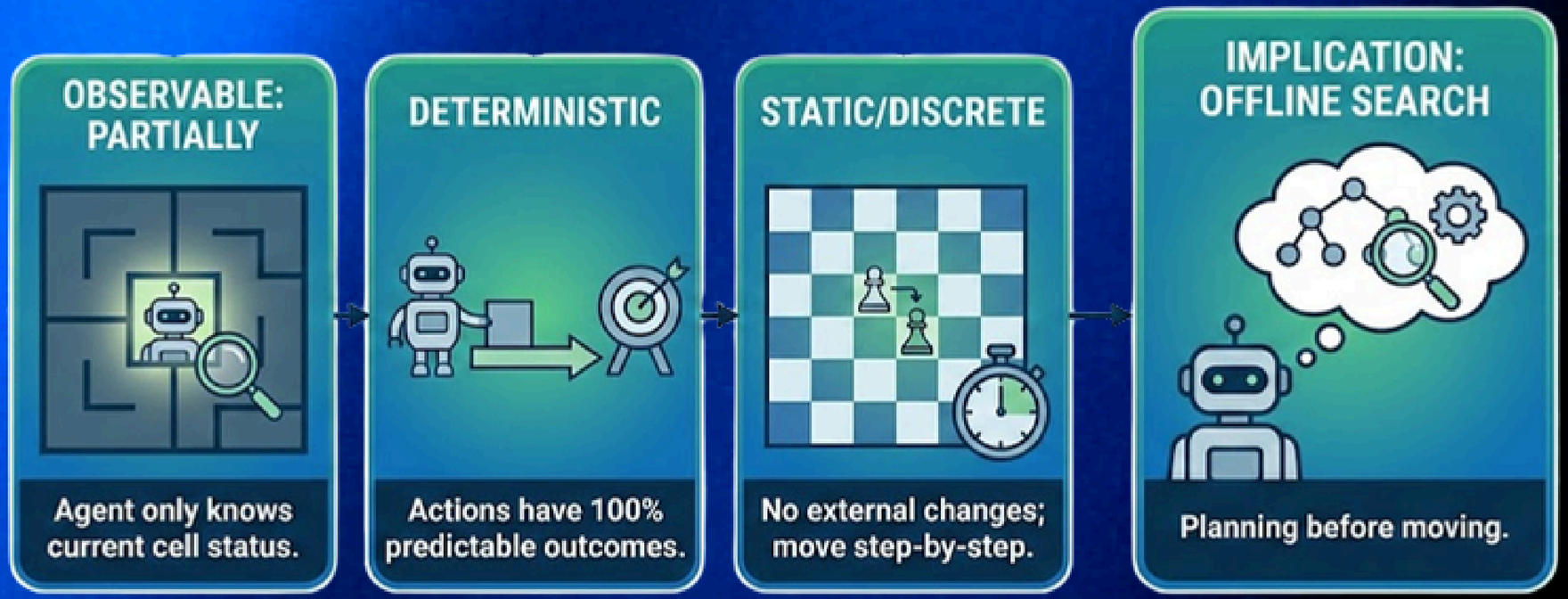
**Performance:**
+10 per clean cell, -1 per move, -1000 for power out (optional).

**Environment:**
***n x n*** grid, walls, varying dirt density.

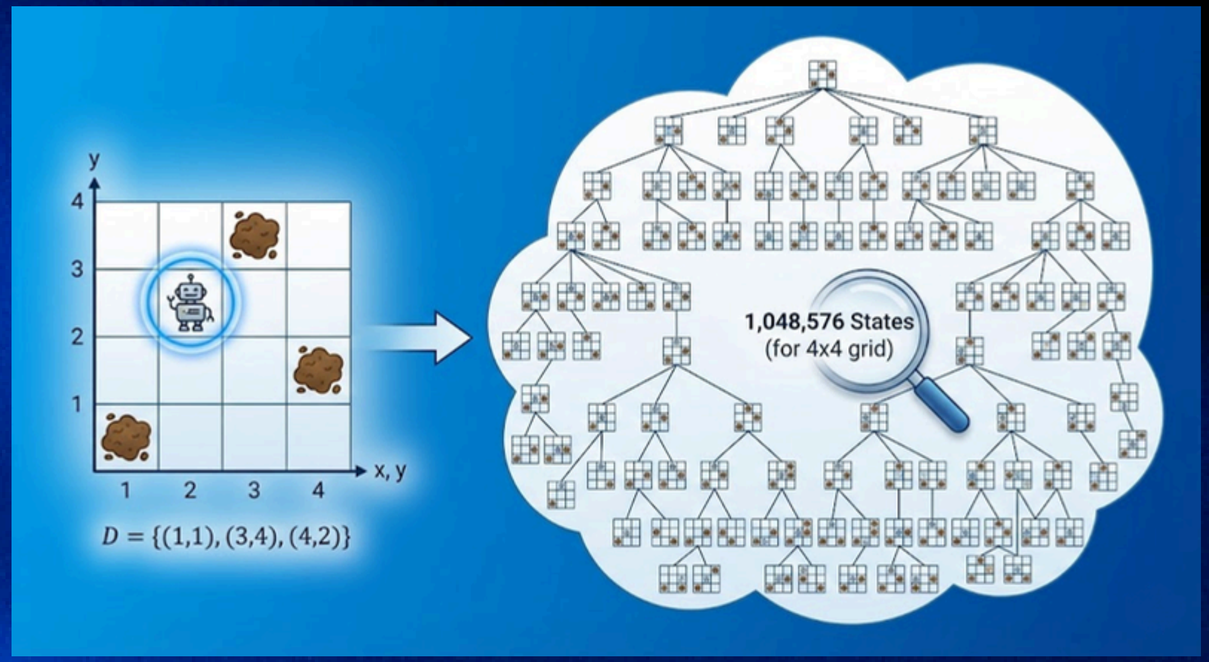
**Actuators:**
Move{Up, Down, Left, Right}, Suck.

**Sensors:**
Location sensor *(x, y)*, Dirt sensor *IsDirty(Current)*.

# Environment Properties



# Formal State Definition




**Formula:** *State = (x, y, D)*

- ***(x, y)***: Robot coordinates.
- ***D***: A set of coordinates {(x_1, y_1), (x_2, y_2), ...} representing dirty cells.

Note: The state space size is $n^2 \times 2^{n^2}$. For a 4 x 4 grid, this is $16 \times 2^{16} = 1,048,576$ states!

# Action Set & Goal Test

## Action Set

To interact with the n x n grid, the agent is equipped with 5 discrete actions:

- Movement Actions: {Up, Down, Left, Right}
  - **Logic:** Modifies the agent's (x, y) coordinates by +-1 unit.
  - **Boundary Constraint:** If an action leads to a position outside the grid (e.g., x < 0 or x > n-1), the agent stays put in its current cell.
  - **Cost:** Each movement attempt costs 1 unit, regardless of success.
- Cleaning Action: {Suck}
  - **Logic:** Checks if the current $(x, y)$ location exists in the dirt set $D$.
  - **Effect:** If dirty, the cell is removed from $D$ (becomes clean). If already clean, the state remains unchanged.
  - **Cost:** Costs 1 unit.

# Action Set & Goal Test

## Goal Test

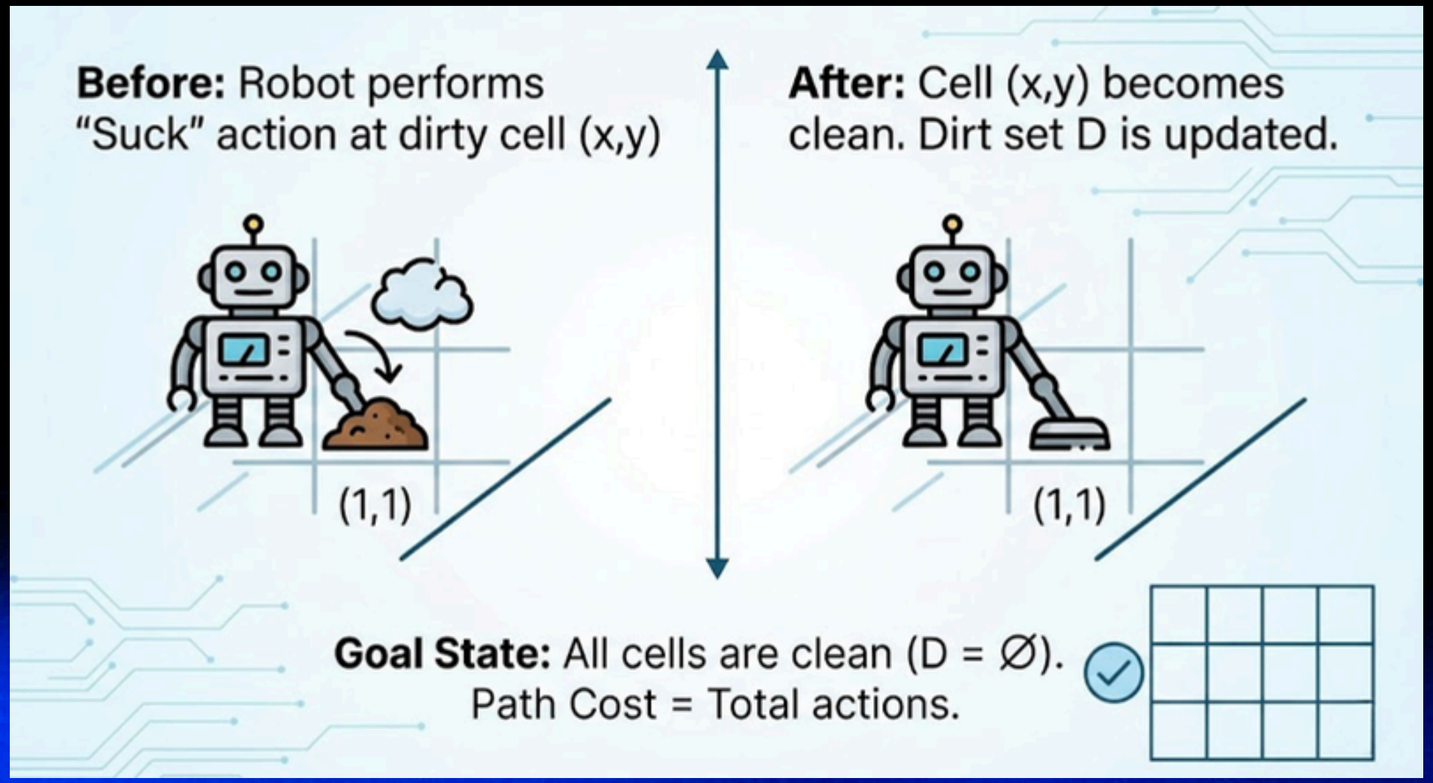
The search algorithm constantly monitors the state to identify the completion of the mission:

- **Condition:** A state s = (x, y, D) is identified as a Goal State if and only if:

$$D = \varnothing$$

- **Description:** The set of dirty cells must be empty, meaning every cell in the grid is clean.
- **Termination:** Once this condition is met, the agent stops and returns the optimal sequence of actions.

# Action Set & Goal Test




## Path Cost Function

**Step Cost:** ***c(s, a, s') = 1.***

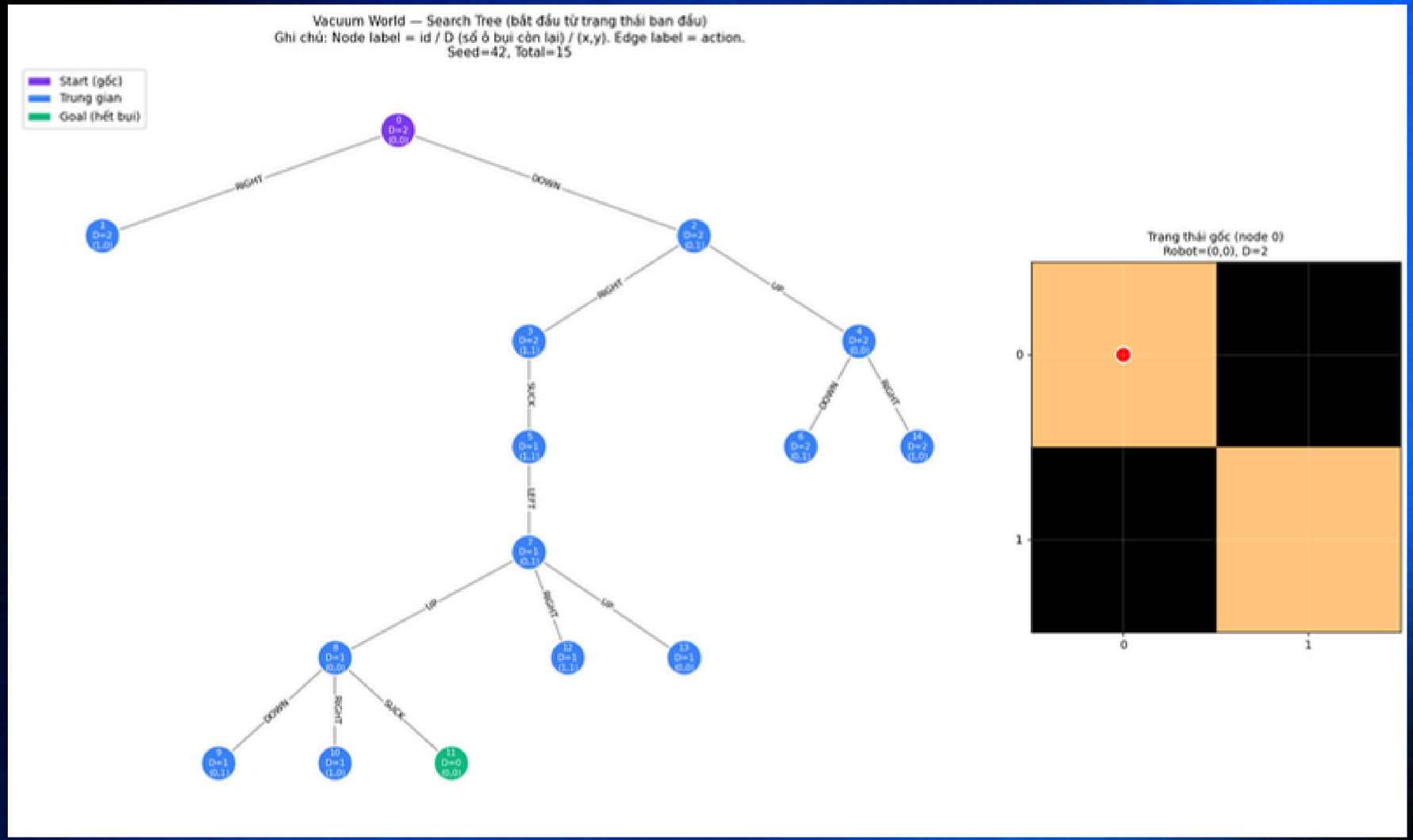
**Total Path Cost:** The sum of all individual step costs from the initial state to the goal.

**Objective:** The agent's primary directive is to find a path where the total cost is minimized.

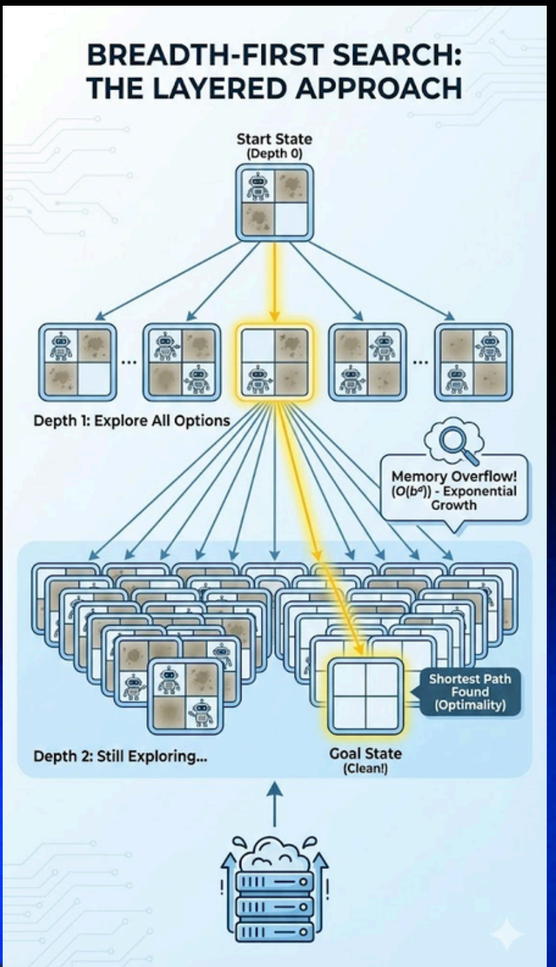
# State Transitions Function

| No. | Current state (s)<br>(Robot position; Dirt list) | Action (a) | Next state(s')<br>(Robot position; Dirt list) | Explanation (Execution logic) |
|---|---|---|---|---|
| 1 | Robot: (2,2); Dirt: {(1,1),(3,3)} | Up | Robot: (1,2); Dirt: {(1,1),(3,3)} | MOVEMENT: Successfully moved onto the empty space. |
| 2 | Robot: (1,2); Dirt: {(1,1),(3,3)} | Left | Robot: (1,1); Dirt: {(1,1),(3,3)} | MOVEMENT: Into the dirty square, not yet sucked up dirt. |
| 3 | Robot: (1,1); Dirt: {(1,1),(3,3)} | Suck | Robot: (1,1); Dirt: {(3,3)} | Dirt SUCTION: At (1,1) there is Dirt -> Cleaned. |
| 4 | Robot: (1,1); Dirt: {(3,3)} | Suck | Robot: (1,1); Dirt: {(3,3)} | Dirt SUCTION: Cell (1,1) is clean -> No change. |
| 5 | Robot: (1,1); Dirt: {(3,3)} | Up | Robot: (1,1); Dirt: {(3,3)} | WALL: Top Border -> Stay still. |
| 6 | Robot: (1,1); Dirt: {(3,3)} | Left | Robot: (1,1); Dirt: {(3,3)} | WALL: Left Border -> Stay still. |

# Search Tree Visualization



# Search Strategies




## Concept:

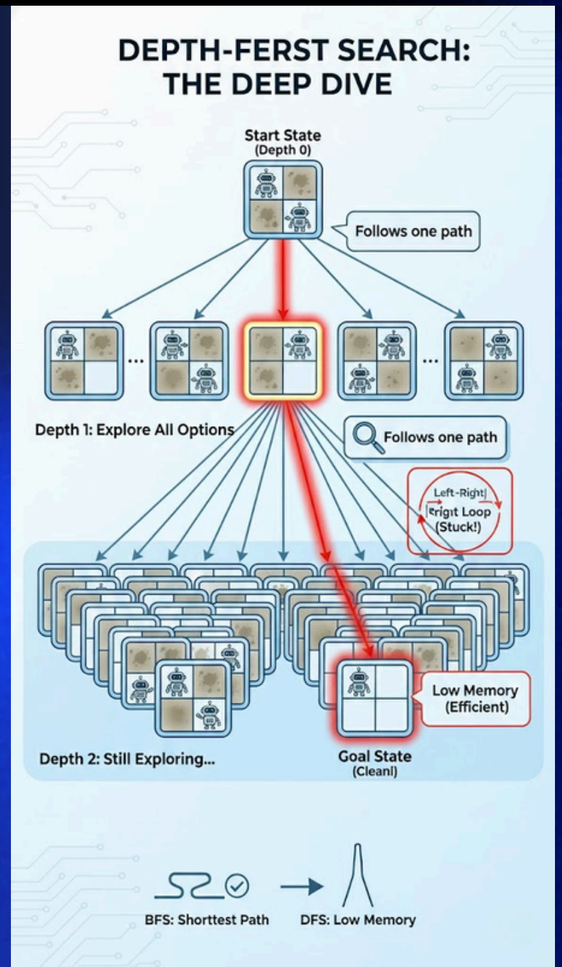
Explores all nodes at depth d before d+1.

## Pros:

Guaranteed to find the shortest path (Optimality).

## Cons:

Exponential memory growth $(O\left(b^d\right))$. In Vacuum World, "Depth" is the number of moves to clean everything.




## Concept:

Follows one path until a leaf or goal is reached.

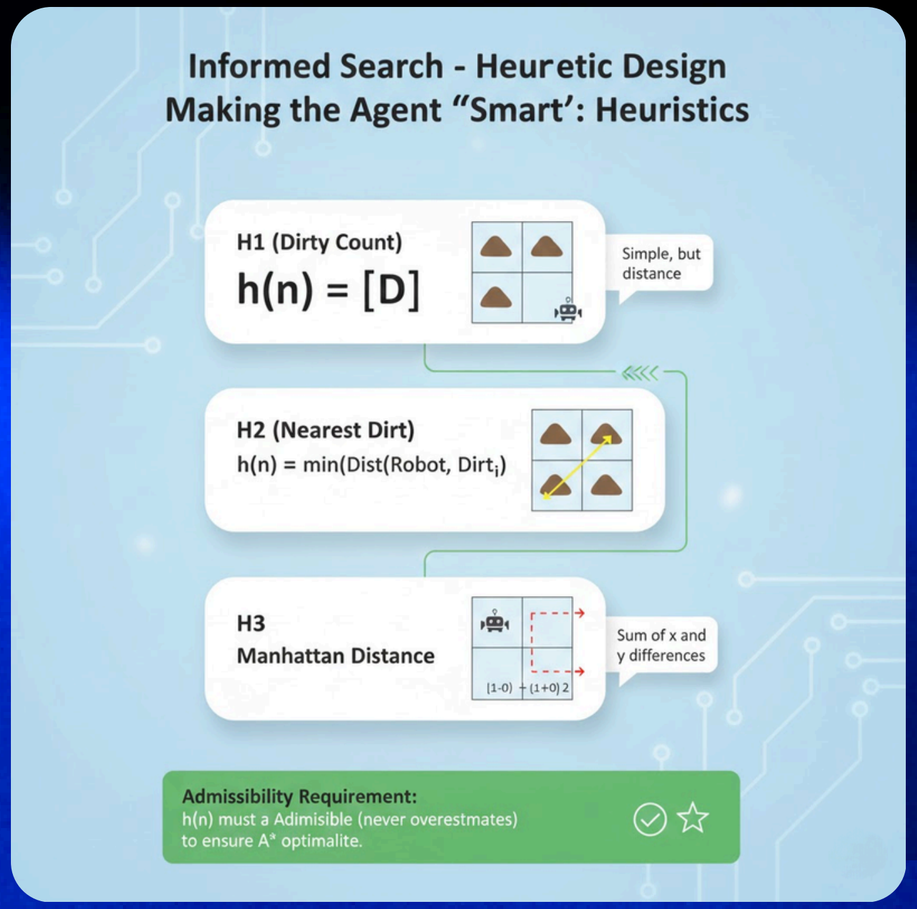
## Pros:

Can get stuck in "Movement Loops" (Left-Right-Left) without an Explored Set.

## Cons:

BFS is better for "Shortest Path", DFS is better for "Low Memory".

# Making the Agent "Smart": Heuristics




## Concept:

Follows one path until a leaf or goal is reached.

## Pros:

Can get stuck in "Movement Loops" (Left-Right-Left) without an Explored Set.

## Cons:

BFS is better for "Shortest Path", DFS is better for "Low Memory".

# Heuristic Design

What is a Heuristic (h(n))?

- **Definition:** A function that estimates the cheapest cost from the current state *n* to a Goal State.
- **Role:** It acts as a "compass," guiding the search algorithm toward promising states and pruning irrelevant paths.
- **Admissibility Condition:** For A* to guarantee an optimal solution, the heuristic must never overestimate the true cost (h(n) =< h^*(n)).

# Algorithm Performance

**Strategy 1: Dirt-Count Heuristic ( $h_1$ )**

**Formula:**

$$h_1(n) = |D|$$

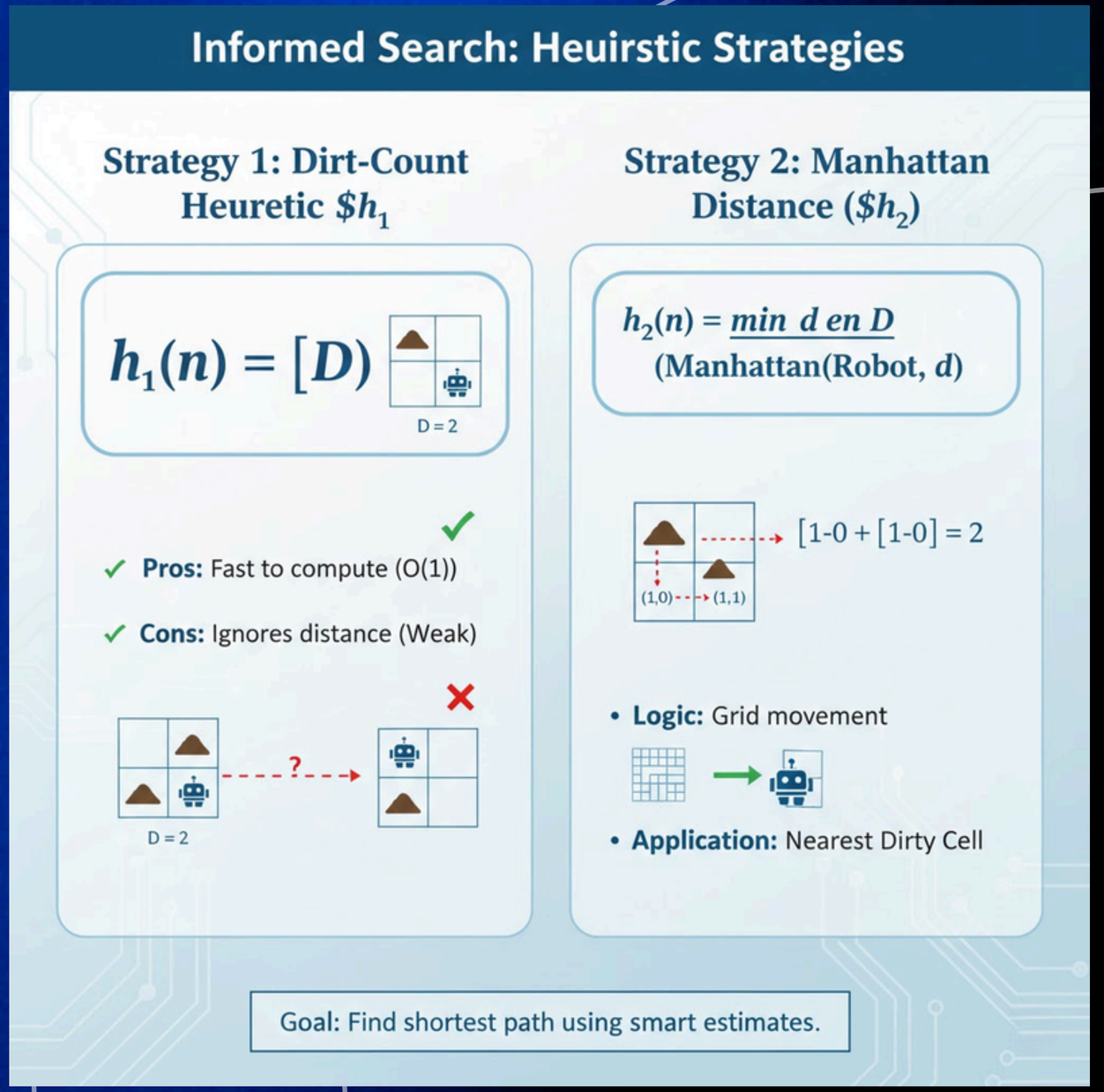
**Strategy 2: Manhattan Distance ( $h_2$ )**

**Formula:**

$$h_{Manhattan}(A, B) = |x_A - x_B| + |y_A - y_B|$$

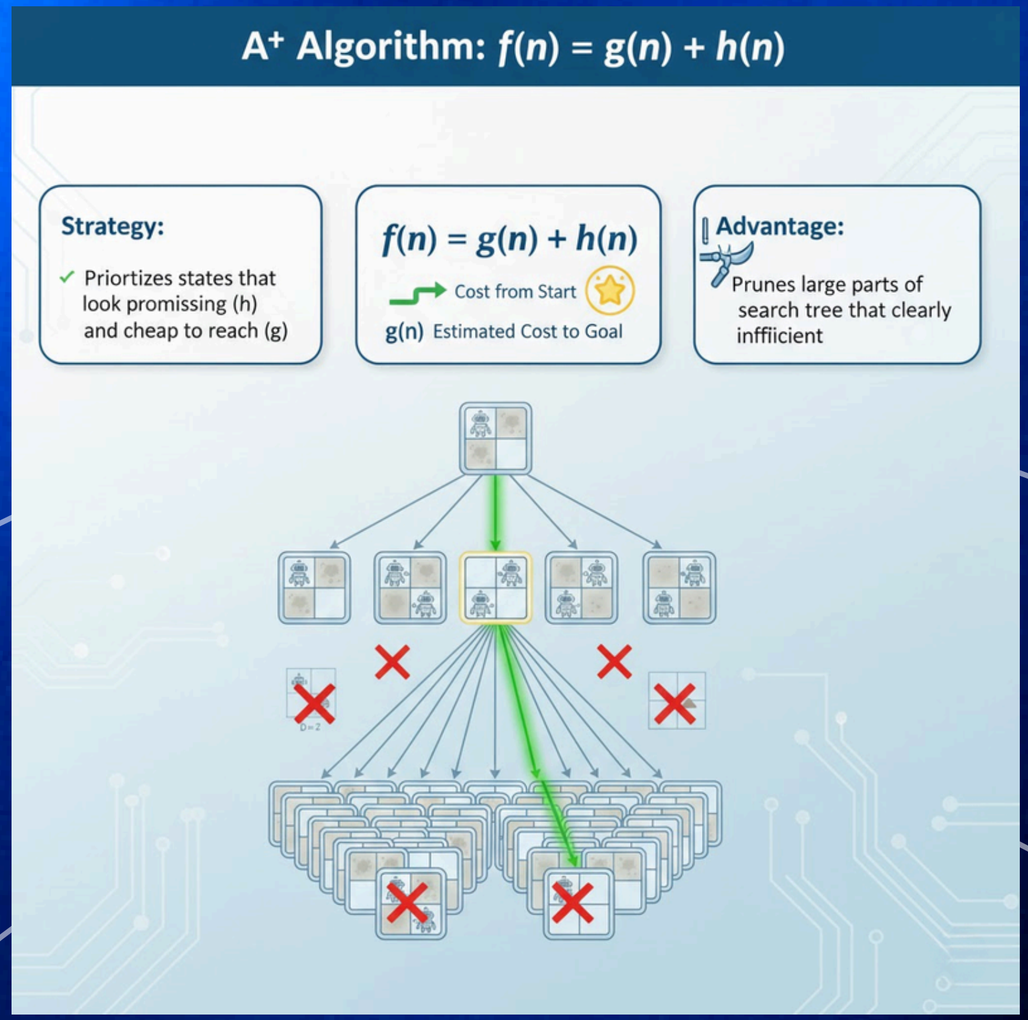
**Application in Vacuum World:**

$$h_2(n) = min(Manhattan(Robot, d))$$



# A Search

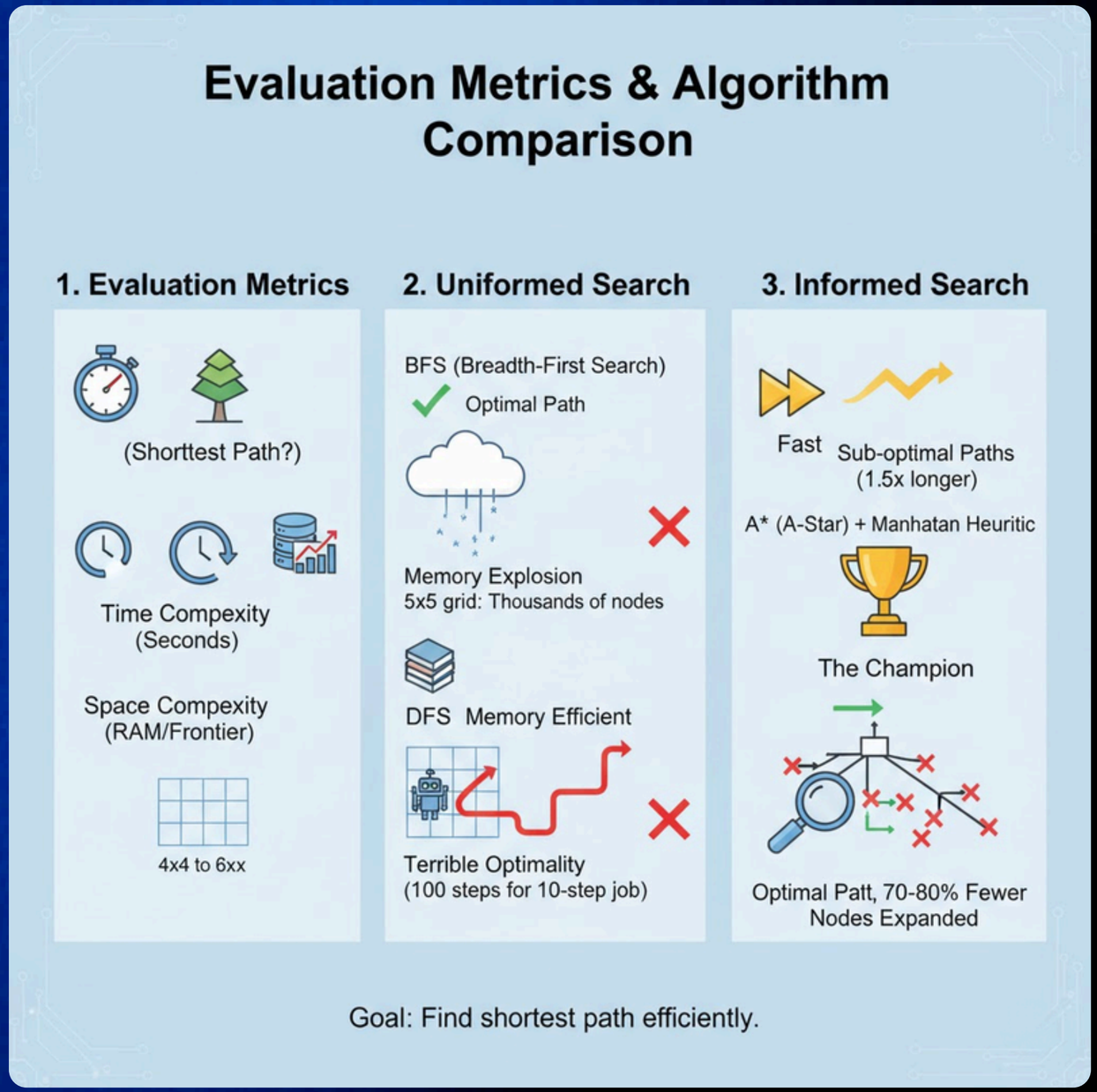
*Efficiency Meets Optimality**



# Algorithm Performance

To judge our agents, we measured three key metrics on varying grid sizes (4 x 4 to 6 x 6):

- **Optimality:** Does the agent find the path with the minimum number of steps?
- **Time Complexity:** How long does it take to run? (Measured in seconds/nodes expanded).
- **Space Complexity:** How much memory (RAM) does the "Frontier" queue consume?



# "Graph Search" VS "Constructive Heuristics"

## Scaling Up: Tacking the State Space Explesion

### 1. The "State Space Explesion

$n^2 \times 2\ n^2$

✓ Manageable

10x4

Exceds Memory

Computational Wall
(BFS, DFS, A* fail)

### 2. Two Approaches to Problem Solving

**Approach A:** Graph Search (A*, BFS)

Verdidt: ✓ Perfect for small tasks

**Approach B:** Constructive Heuritic (Gredy NN) (Gredy NN)

Recting ✓ Perfect for small tasks

Essential for large tasks

### 3. The Solution: Gredy Neighbor (Gredy NN)

$O(k^2 \cdot n^2)$

Millessionts for 20x20

Sub-optimal path

✓ Gets job done instanty

Goal: Solve large-scale problems quickly

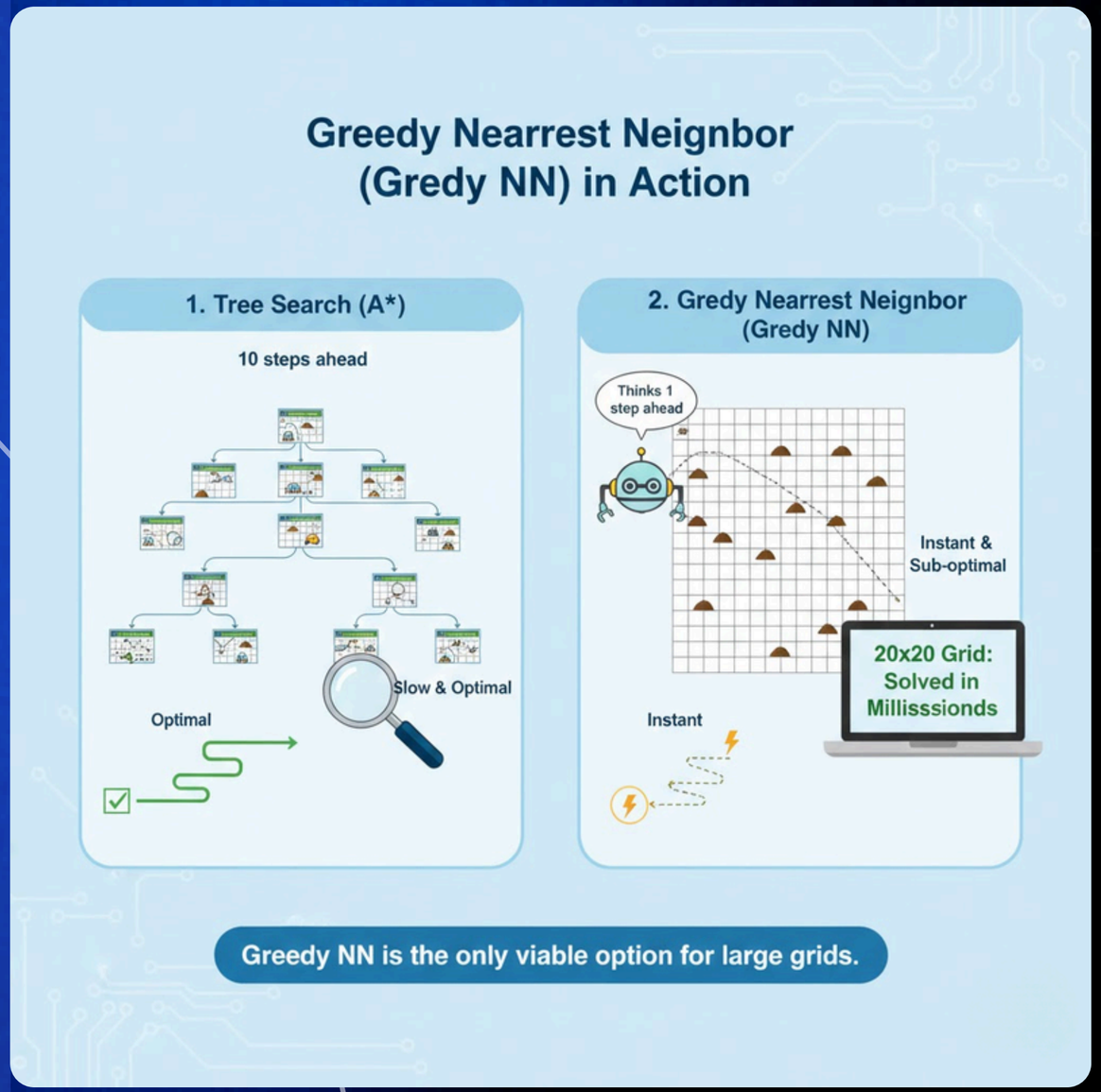
# Constructive Heuristic vs. Search

- **Greedy NN**: Don't search a tree; just move to the closest dirt.
- **Comparison:**

**Search:** Thinks 10 steps ahead (Slow, Optimal).

**Greedy NN:** Thinks 1 step ahead (Instant, Sub-optimal).

- **Result:** Greedy NN is the only viable option for 20 x 20 grids.



# DEMO

# Experimental Results & Algorithm Comparison

## Completness

Guarnteed to find find a solution if one exists?

## Optimitally

Finds the Shortest Shortest Path (minimum moves?)

## Time Complexity

$O(b^{-d} vs. $O(n.^{k}

How fast as problem grows?

## Space Complexity


How much RAM consumed?

PERFORMANCE Mectrics

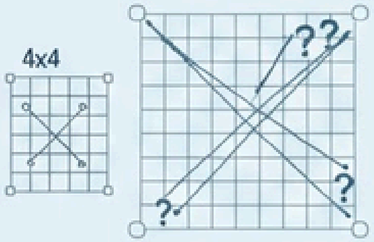
# Comprehensive Comparison Table

| Search Category | Algorithm | Optimality | Completeness | Time/Space | Scalability |
|---|---|---|---|---|---|
| Uninformed | BFS | Yes | Yes | $O\left(b^d\right)$ | Very Low (max 5x5) |
| Uninformed | DFS | No | Yes* | $O\left(b^m\right)$ | Moderate |
| Informed | A* | Yes | Yes | $O\left(b^d\right)$ | Low (max 7x7) |
| Informed | Greedy Best-First | No | No (Loops) | $O\left(b^d\right)$ | Low |
| Constructive | Greedy NN | No | Yes | $Polynomial$ | High (100x100+) |

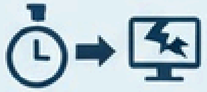
# Conclusion & Future Work



## 1. Graph Search (BFS, A*)

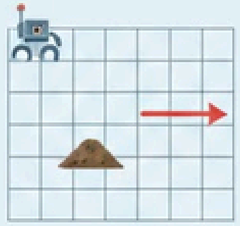






Algorithms Crash

## 2. Constructive Heueritic (Gredy NN)

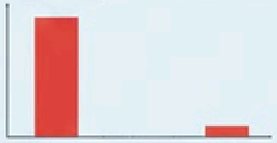


**Acts:** Moves to nearest goal immediately

**Result:** Extremely fast (<100ms) on 20x20, path 50-100% longer

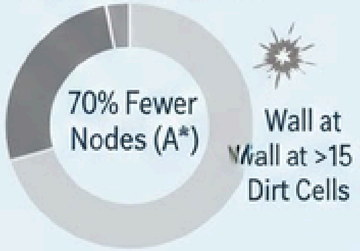
## 3. Key Findings from Our Experiments

Memory Bottlonceck



Frontier Queue (BFS)

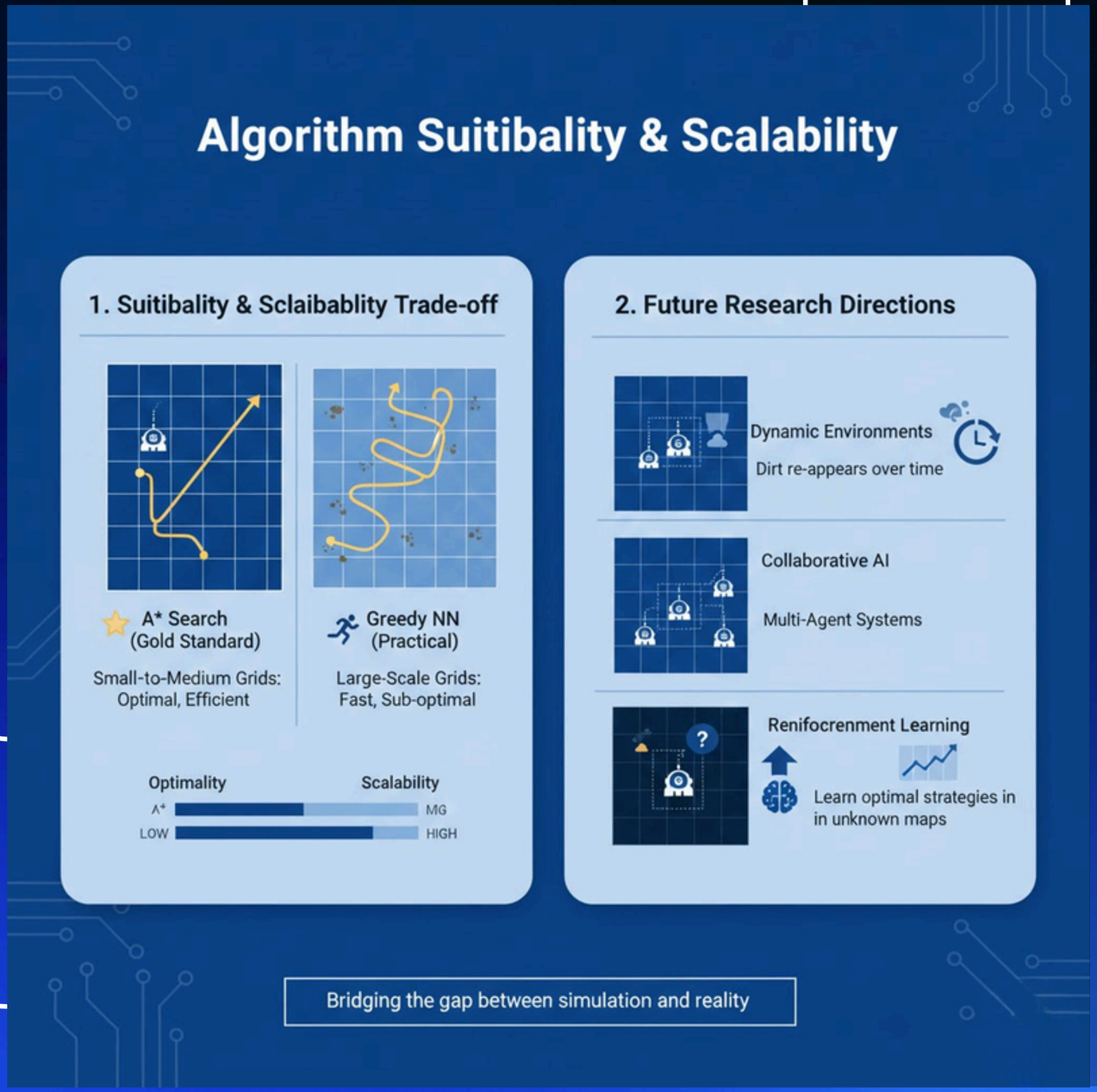
### Heueitic Impact




### The Trade-off

A* (Min Energy

(Real-time)




## Algorithm Suitability & Scalability

The Optimality-Scalability Trade-off: Our analysis confirms that A Search* is the "Gold Standard" for small-to-medium grids, offering the perfect balance of path precision and memory efficiency—provided the **Heuristic Quality** (e.g., Manhattan distance) is high. However, for large-scale practical applications, we observed that **Constructive Heuristics** (Greedy NN) are strictly necessary, as they trade theoretical optimality for the execution speed required to handle the exponential growth of state spaces.

## Future Research Directions

- **From Static to Dynamic Intelligence:** The next phase of development aims to bridge the gap between simulation and reality by introducing:
- **Dynamic Environments:** Handling stochastic elements where dirt re-appears over time.
- **Collaborative AI:** Implementing Multi-Agent Systems to clean the grid in parallel.
- **Reinforcement Learning:** allowing the agent to learn optimal strategies in unknown map layouts without prior knowledge.

# THANK YOU
## FOR LISTENING!