Ai Project Proposal

Solving Problem Using AI Search Algorithms

- ## The Problem Is :

Block Stacking Problem

- ## Introduction :

This project explores and applies fundamental Artificial Intelligence search algorithms to solve the **Block Stacking Problem**, a classic planning and state-space search problem in AI.

The objective is to transform an initial block configuration into a goal configuration using valid moves while respecting predefined constraints. The project emphasizes algorithm behavior, performance trade-offs, and comparison between uninformed, informed, and optimal search techniques.

- ## Problem Description :

In the **Block Stacking Problem** a set of blocks (e.g., A, B, C) can be stacked on top of each other or placed on the table.

Objective :

Transform the initial configuration into the target goal configuration using the minimum number of valid moves.

Example :

Initial State : A on B, B on Table, C on Table
Goal State : C on B, B on A

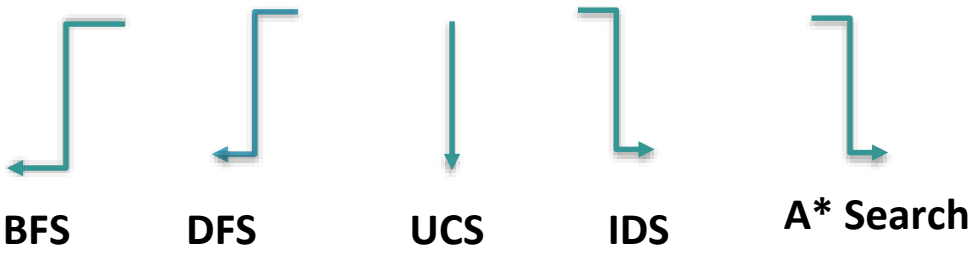
Rules and Constraints :

Only **one block** can be moved at a time
A block can be moved **only if it is clear** (no block on top of it)
A block can be placed:
On the table, or
On top of another clear block
Each move has a uniform cost = 1

• Why Block Stacking?

- It is a perfect benchmark for search algorithms
- It demonstrates the difference between optimal search and heuristic search
- Suitable for theoretical and practical comparison of algorithms

Algorithms Implemented



1 : Breadth-First Search (BFS)

BFS explores the state space **level by level** expanding all states at the current depth before moving deeper.

Why BFS fits Block Stacking ?

- Guarantees the **shortest sequence of moves**
- Simple and systematic exploration

Expected Behavior :

- Always finds the optimal solution
- Very high memory consumption for larger problems

2 : Depth-First Search (DFS)

DFS explores the search space by going **as deep as possible** along one branch before backtracking.

Why DFS fits Block Stacking ?

- Very memory-efficient
- Simple to implement

Expected Behavior :

- May find a solution quickly
- Does **not guarantee optimality**

---

3 : Uniform Cost Search (UCS)

UCS expands nodes based on the **lowest cumulative path cost .**

Why DFS fits Block Stacking ?

- Guarantees optimal solutions
- Works well since all moves have uniform cost
- Serves as a baseline for optimal comparison

Expected Behavior :

- Slower than DFS and IDS
- Extremely accurate (optimal)

4 : Iterative Deepening Search (IDS)

IDS combines DFS and BFS by performing depth-limited DFS, increasing the depth limit gradually until the goal is found.

Why LDS fits Block Stacking ?

- Guarantees optimality like BFS
- Uses memory efficiently like DFS

Expected Behavior :

- Optimal solution
- Lower memory usage than BFS

5 : A* Search Algorithm

A* is an informed search algorithm combining path cost

Heuristic estimate : **f(n) = g(n) + h(n**

Heuristic Used :

- Number of blocks that are not in their goal position
- Admissible and simple

Why A* fits Block Stacking :

- Faster than UCS
- Reduces search space dramatically
- Still capable of reaching optimal or near-optimal solutions

Expected Behavior :

- Faster convergence than BFS/UCS
- Higher memory usage

• Evaluation & Expected Results :

During experimentation we will evaluate each algorithm based on:

Performance Metrics :

- Execution time
- Memory usage
- Path cost (tour length)
- Optimality vs. speed
- Scalability with number of blocks

What we expect ?

- **UCS** : Best optimal result, but slowest
- **A*** : Fast + near-optimal, depending on heuristic strength
- **DFS** : fast but not always optimal

This comparison highlights the trade-offs between uninformed search,heuristic search, and optimal search approaches.

Algorithm	*Expected Solution Quality*	*Expected Speed*	*Key Insight*
BFS	*Optimal*	*Slow*	Baseline for optimal search
DFS	*Not Guaranteed*	*Fast*	Shows depth bias
UCS	*Optimal*	*Slow*	cost based / optimal search
IDS	*Optimal*	Moderate	Best trade off
A Search*	*Optima / Near Optimal*	*Fast*	Power of heuristics

Deliverables :

The final submission will be through a single Public GitHub Repository link, which will contain the following components:

- Source Code for all five algorithms (UCS, A*, BFS,DFS,IDS).
- A README.md file providing clear instructions on how to set up, run, and test the project.
- A detailed PDF Report including implementation specifics, comprehensive results,comparison tables, and visual analysis of the performance metrics across all algorithms

Names of Team :

1. ياسمين حاتم عبدالله سمك
2. محمد حسام عبدالمقصود ابوريه
3. محمد وسيم محمد وجيه سكر
4. يوسف ياسر حافظ طه حافظ
5. فوزي محمد وسيم فوزي عبدالقادر
6. نور احمد رجب محمد
7. محمد حسن سعدالدين محمد ابراهيم