FINAL REPORT

TITLE:

Optimizing Pathfinding Algorithms for Real-World Applications: A Comparative Study

INTRODUCTION:

Pathfinding is a core challenge in artificial intelligence (AI), with wide-ranging applications in robotics, video game development, logistics, and autonomous navigation systems. An AI agent's ability to efficiently determine a viable route from a starting point to a goal is critical for real-time decision-making in dynamic environments. Although classical path-finding algorithms are well-established, their performance can vary considerably depending on the structure of the environment and the computational constraints involved.

In this project, I compare and evaluate the performance of five pathfinding algorithms: A*, Dijkstra's Algorithm, Iterative Deepening Depth-First Search (IDDFS), Beam Search, and Hill Climbing. These algorithms were implemented in Python and tested on randomly generated 2D grid environments with obstacles. My objective is to examine each algorithm's efficiency, path optimality, and applicability to different real-world scenarios

BACKGROUND:

Pathfinding algorithms are fundamental to both classical AI and modern decision-making systems. Among the most widely used is **A***, which balances the actual cost of reaching a node with a heuristic estimate of the remaining cost to the goal. When paired with an admissible heuristic, A* guarantees an optimal solution and is favored in robotics and game design for its efficiency.

Dijkstra's Algorithm is effectively a variant of A* with a zero-valued heuristic. It ensures optimal solutions in graphs with non-negative edge weights but can be slower due to its uninformed nature.

Iterative Deepening Depth-First Search (IDDFS) combines the memory efficiency of depth-first search with the completeness of breadth-first search by gradually increasing the depth limit. While useful in memory-constrained settings, it tends to be slower due to repeated exploration of nodes.

Beam Search introduces a memory-bound, greedy variant of best-first search by retaining only a limited number of nodes at each level. Though it is not complete or optimal, it can perform well in constrained environments.

Hill Climbing, a simple greedy algorithm, selects the neighbor closest to the goal at each step. It is computationally inexpensive but highly susceptible to getting stuck in local minima, especially in environments with complex obstacle layouts.

As noted in prior literature (e.g., Russell and Norvig, 2020), the performance of these algorithms is highly task dependent. For this project, I tested each algorithm under identical conditions on randomized 2D grid maps to provide a clear and consistent comparison.

METHODOLOGY:

To evaluate the selected pathfinding algorithms, I implemented each one in Python and tested them on randomly generated 2D grid environments. Each grid consisted of 10×10 cells, with a fixed probability of any given cell being an obstacle. Grid generation ensured that the start node at the top-left and the goal node at the bottom-right were not obstructed. The search graph was constructed using the NetworkX library, with valid neighbors connected in the cardinal directions (up, down, left, right).

Each algorithm was evaluated across a dataset of multiple grid maps, using consistent start and goal positions. The core logic of each algorithm was kept modular and parameterized to allow flexible tuning (e.g., beam width for Beam Search). The Manhattan distance heuristic was used in A*, Beam Search, and Hill Climbing, as it is admissible and efficient in grid environments.

I recorded three key performance metrics:

- **Path length**: the number of steps in the final path
- **Execution time**: runtime measured in seconds using time.perf_counter()
- **Success rate**: whether a valid path was found (relevant for non-optimal or greedy methods)

All experiments were performed using Google Colab with Python 3, ensuring consistent runtime and reproducibility.

DESIGN AND RESULTS:

I conducted experiments by running each of the five algorithms on a dataset of randomly generated 10×10 grid maps with 20% obstacle probability. Each algorithm was tested on the same set of grids, using consistent start and goal locations. The evaluation recorded path length, and execution time.

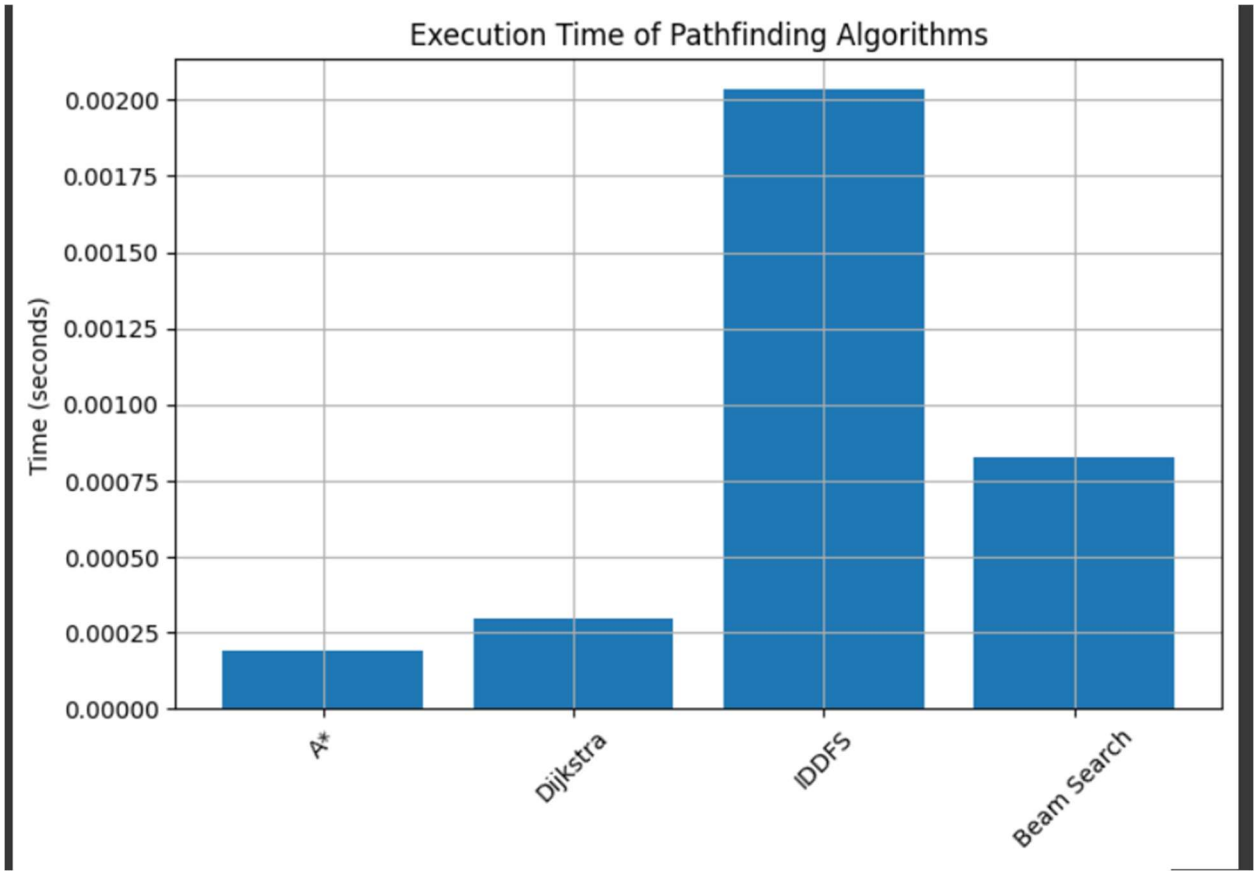
The results are summarized in the following table:

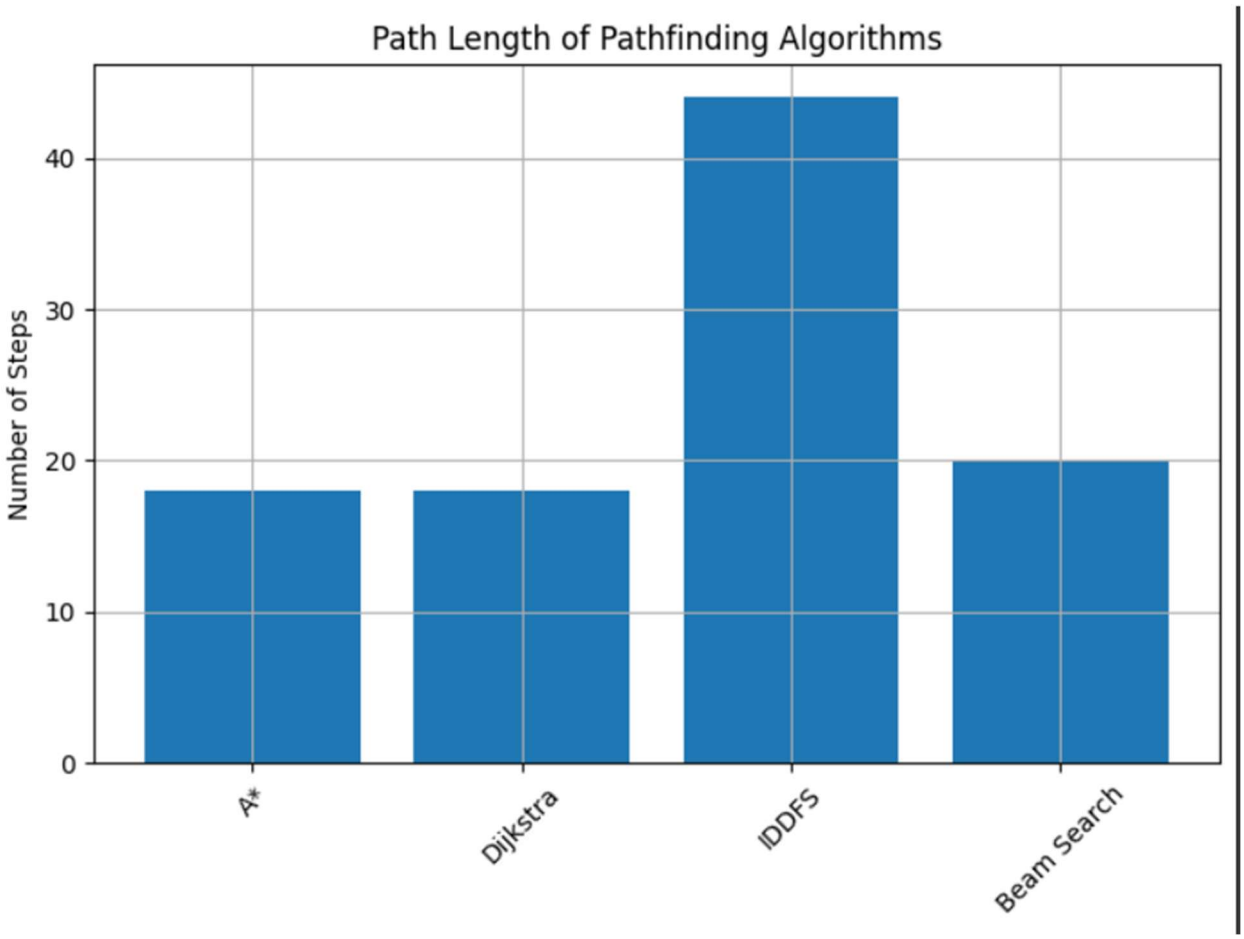
	Path Length	Execution Time (s)
A*	18.0	0.000188
Dijkstra	18.0	0.000295
IDDFS	44.0	0.002033
Beam Search	20.0	0.000826
Hill Climbing	NaN	NaN

Observations:

- **A*** yielded optimal paths quickly, making it the best overall performer.
- **Dijkstra's Algorithm** produced the same path quality but was slower due to lack of heuristic guidance.
- **IDDFS** was slower and produced significantly longer paths due to its depth-limited nature.
- **Beam Search** was fast but returned slightly suboptimal paths.
- **Hill Climbing** frequently failed to return any path, making it unreliable in obstacle-filled maps.

The following plots provide visual comparisons:





SUMMARY:

This project explored and evaluated five classical pathfinding algorithms—A*, Dijkstra's Algorithm, Iterative Deepening Depth-First Search (IDDFS), Beam Search, and Hill Climbing—on randomly generated 2D grid environments with obstacles. The comparison was based on path length, and execution time.

Among the evaluated algorithms, A* demonstrated the best overall performance, providing both optimal paths and efficient execution. Dijkstra's algorithm, though optimal, was slower due to its exhaustive exploration. IDDFS was significantly less efficient and returned suboptimal paths. Beam Search showed potential as a fast and reasonably accurate method when memory constraints are a concern. Hill Climbing proved to be unreliable, frequently failing to find valid paths due to its susceptibility to local minima and lack of backtracking.

Future directions for this work include:

- Testing on **larger and weighted graphs**, such as real-world road networks.
- Comparing performance on **dynamic environments** with moving obstacles.
- Implementing **reinforcement learning agents** for learned pathfinding strategies.
- Using **neural network models** to predict next steps based on map inputs.

These enhancements would further evaluate the scalability, robustness, and adaptability of different pathfinding strategies in real-world scenarios.

REFERENCES:

Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

NetworkX developers. (n.d.). *NetworkX Documentation*. Retrieved from https://networkx.org

Matplotlib developers. (n.d.). *Matplotlib: Visualization with Python*. https://matplotlib.org

OpenAI. (2024). *ChatGPT*. https://chat.openai.com