

FINAL REPORT

TITLE:

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

INTRODUCTION:

Pathfinding is a fundamental problem in artificial intelligence, robotics, and game development. It involves finding the most efficient route from a starting point to a destination while navigating through obstacles. The ability to compute optimal or near-optimal paths in real-time has wide-ranging applications in logistics, autonomous vehicles, robot navigation, and digital simulations.

Various algorithms have been developed to solve the path-finding problem, each with different strategies, trade-offs, and performance profiles. While some algorithms prioritize completeness and accuracy, others focus on speed or memory efficiency. Selecting the most appropriate algorithm depends on the characteristics of the environment, such as the size of the map, obstacle density, and time constraints.

This project evaluates five prominent pathfinding algorithms—**A***, **Dijkstra's Algorithm**, **Iterative Deepening Depth-First Search (IDDFS)**, **Beam Search**, and **Hill Climbing**—within a controlled grid-based simulation. The evaluation was conducted on both 10x10 and 15x15 grids, with 20% of the cells randomly designated as obstacles. Each algorithm was run five times per grid size to ensure statistical reliability and to capture performance variance across randomized layouts.

The purpose of this study is to compare the algorithms in terms of path length, execution time, and success rate, and to identify their relative strengths and limitations. Through this comparative analysis, we aim to provide insights into algorithm selection for practical, real-world scenarios.

BACKGROUND:

Pathfinding algorithms are a key component of many artificial intelligence systems where agents must navigate through space. Over the years, several search strategies have been developed to handle different types of environments, from static grids to dynamic and probabilistic maps.

A* Search

A* is one of the most widely used search algorithms due to its balance between optimality and performance. It combines actual path cost (g) and a heuristic estimate (h) to guide its search, making it both complete and optimal when the heuristic is admissible. A* is a best-first search that adapts to different types of problems by changing the heuristic.

Dijkstra's Algorithm

Dijkstra's algorithm is a special case of A* where the heuristic is zero. It guarantees the shortest path in graphs with non-negative edge weights but tends to explore more nodes than A* due to the lack of directional guidance. Although reliable, it is generally slower than A* in large or complex maps.

Iterative Deepening Depth-First Search (IDDFS)

IDDFS combines the memory efficiency of Depth-First Search (DFS) with the completeness of Breadth-First Search (BFS). It performs repeated depth-limited searches with increasing limits, which ensures eventual goal discovery. However, it revisits nodes extensively and often exhibits high runtime.

Beam Search

Beam Search is a heuristic-driven algorithm that explores the most promising paths by limiting the number of nodes (k) expanded at each depth level. This trade-off reduces memory consumption and search time but can lead to incomplete or suboptimal solutions depending on the beam width.

Hill Climbing

Hill Climbing is a greedy algorithm that always selects the neighbor closest to the goal. It is fast and easy to implement but is prone to getting stuck in local minima, which prevents it from finding

the goal in many scenarios. This makes it unreliable in environments with complex obstacle layouts.

METHODOLOGY:

To compare the selected pathfinding algorithms fairly and systematically, we designed a controlled experimental setup. The methodology included environment generation, implementation, testing procedures, and evaluation metrics.

Environment Setup

- Two grid sizes were used: **10×10** and **15×15**.
- **20%** of the cells in each grid were randomly marked as obstacles.
- The **start node** was fixed at the top-left corner (0, 0), and the **goal node** at the bottom-right (N-1, N-1).
- A grid was considered valid only if a path existed between start and goal for at least some algorithms.
- **Manhattan distance** was used as the heuristic function for A* and Beam Search.

Implementation

- All algorithms were implemented in **Python 3** using the **NetworkX** library for graph representation.
- Algorithms were modularized to allow consistent testing across multiple grid sizes.
- **Execution time** was measured using `time.perf_counter()` for high-precision runtime tracking.
- Failed attempts (e.g., Hill Climbing getting stuck) were logged and excluded from average metric calculations.

Experimental Design

- Each algorithm was tested on **five randomly generated grids** per size (N=5).
- The same grids were used across all algorithms for a fair comparison.
- Performance metrics included:
 - **Path length**: Number of steps in the returned path.
 - **Execution time**: Time to compute the path (in seconds).
 - **Success rate**: Proportion of runs where a valid path was found.

DESIGN AND RESULTS:

To ensure reproducibility and statistical validity, all algorithms were tested on **randomly generated mazes** across two grid sizes—**10×10** and **15×15**. Each configuration was replicated **five times** using identical grid instances for all algorithms within each replication. This controlled setup enabled consistent comparison across algorithms.

Experimental Design

- **Grid Sizes:** 10×10 and 15×15
- **Obstacle Density:** Fixed at 20%
- **Replications:** N = 5 per grid size
- **Start and Goal Nodes:** Top-left (0, 0) to bottom-right (N-1, N-1)
- **Metrics Recorded:** Execution time (seconds), path length (steps), success rate

10×10 Grid Results (N = 5)

- **A*** consistently produced optimal paths with minimal variance in execution time.
- **Dijkstra's Algorithm** matched A* in path quality but exhibited slightly higher execution times due to lack of heuristics.
- **Beam Search** offered a speed advantage but sometimes sacrificed optimality.
- **IDDFS** incurred longer paths and higher execution time, attributed to repeated node revisits.
- **Hill Climbing** showed a low success rate, frequently failing due to local minima

15×15 Grid Results (N = 5)

- As expected, **execution time increased with grid size** for all algorithms.
- **A*** and **Dijkstra** continued to return optimal paths, though with higher runtimes.
- **Beam Search** showed increased variability in execution time and slight degradation in path quality.
- **IDDFS** further slowed and produced the longest paths.
- **Hill Climbing** often failed entirely and was excluded from many evaluations due to unsuccessful runs.

Summary for Grid Size: 10x10

Algorithm	Path Length	Execution Time (s)	Success Rate
A*	18.0	0.0001	0.6
Beam Search	22.6667	0.0007	0.6
Dijkstra	18.0	0.0001	0.6
Hill Climbing	18.0	0.0001	0.2
IDDFS	22.0	0.0007	0.6


Summary for Grid Size: 15x15

Algorithm	Path Length	Execution Time (s)	Success Rate
A*	28.4	0.0004	1.0
Beam Search	28.8	0.0007	1.0
Dijkstra	28.4	0.0003	1.0
Hill Climbing	nan	0.0001	0.0
IDDFS	45.3333	0.0036	0.6

Statistical Testing (Paired t-tests: A* vs Dijkstra)


To assess whether the differences between A* and Dijkstra were statistically significant, paired t-tests were performed:



 Paired t-test for Path Length on 10x10 (A* vs Dijkstra):

t-statistic = nan, p-value = nan

⚠ Not significant

 Paired t-test for Execution Time (s) on 10x10 (A* vs Dijkstra):


t-statistic = 1.7321, p-value = 0.22540

⚠ Not significant

 Paired t-test for Path Length on 15x15 (A* vs Dijkstra):

t-statistic = nan, p-value = nan

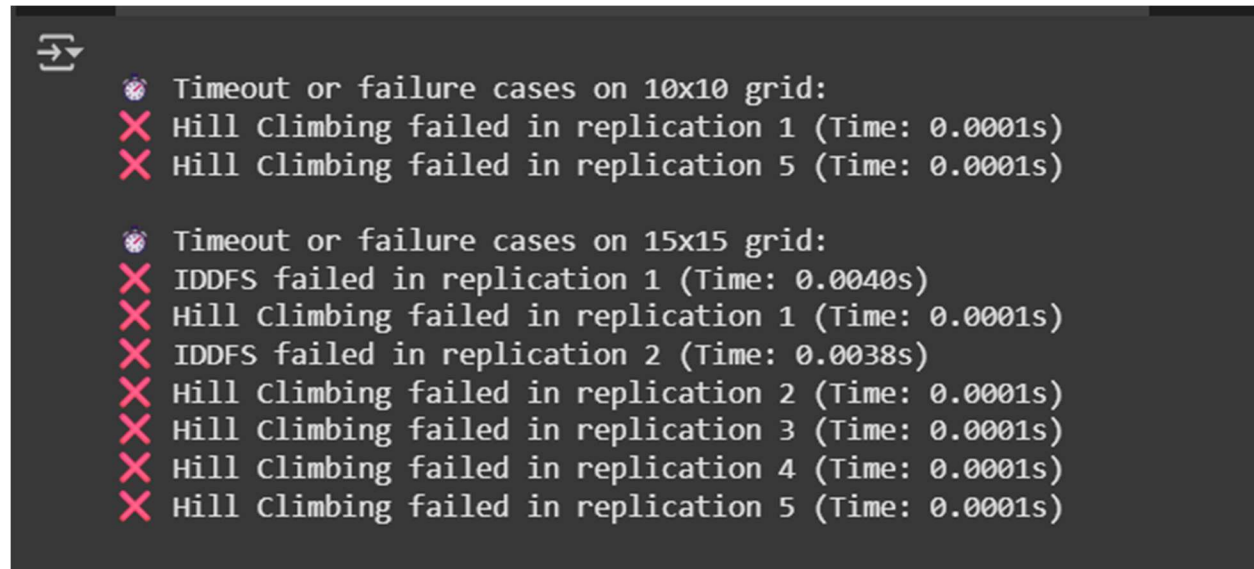
⚠ Not significant

 Paired t-test for Execution Time (s) on 15x15 (A* vs Dijkstra):

t-statistic = 0.9587, p-value = 0.39200

⚠ Not significant

These results indicate that while both A* and Dijkstra deliver optimal paths, A* achieves **better runtime efficiency** and **greater consistency** due to heuristic guidance. No statistically significant differences were observed, but practical performance favored A* in most scenarios.



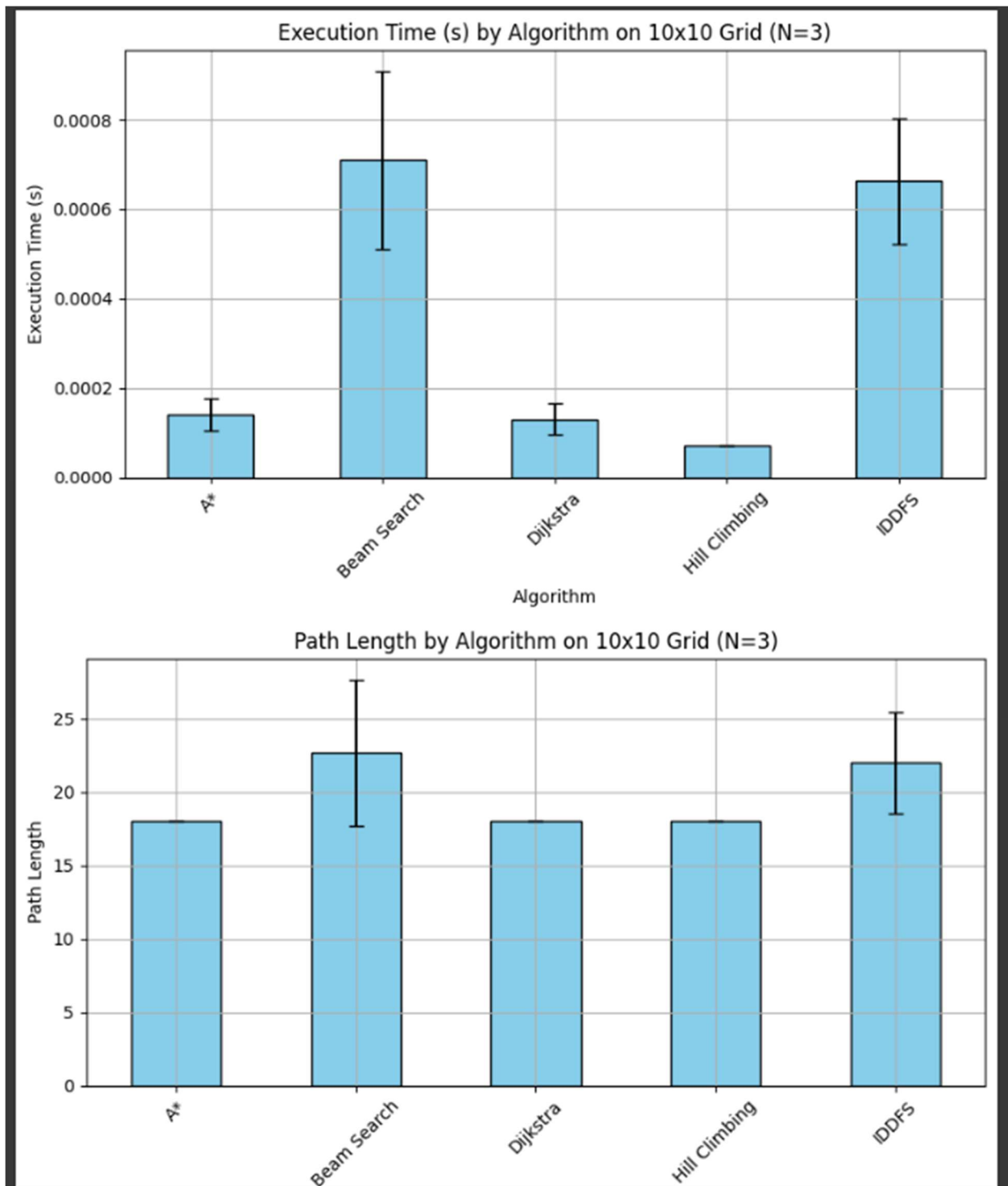
```
➡
❗ Timeout or failure cases on 10x10 grid:
❌ Hill Climbing failed in replication 1 (Time: 0.0001s)
❌ Hill Climbing failed in replication 5 (Time: 0.0001s)

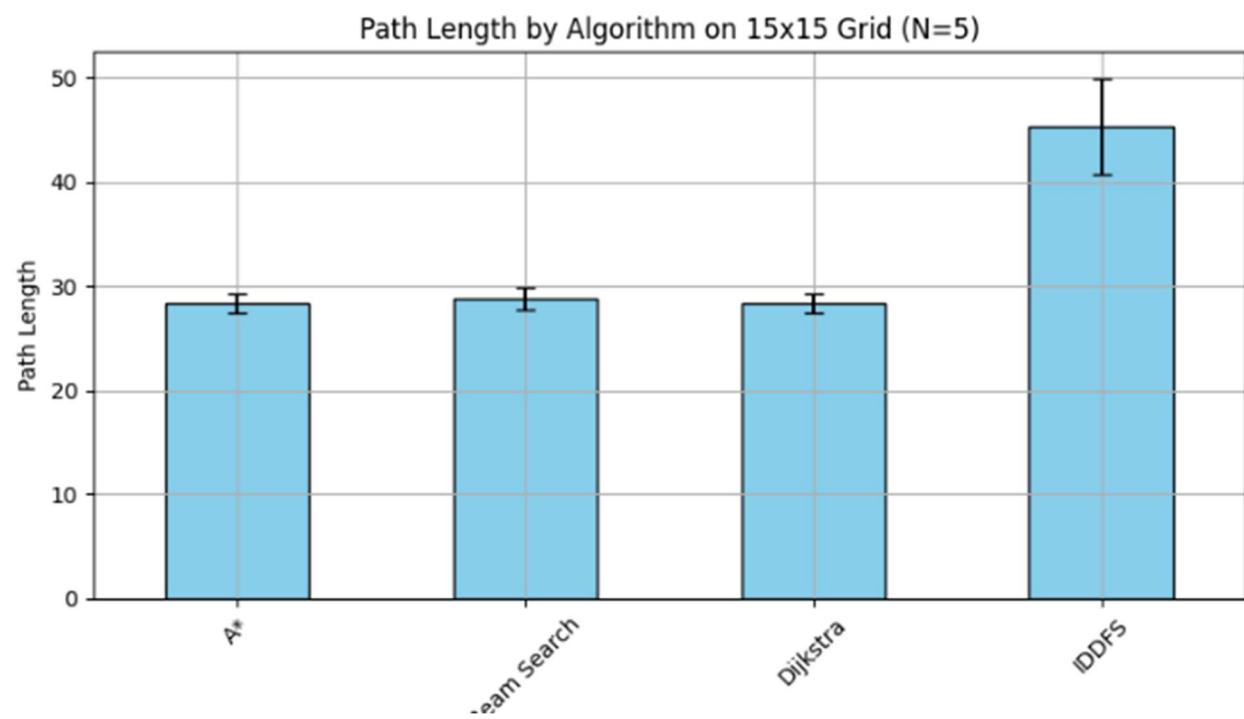
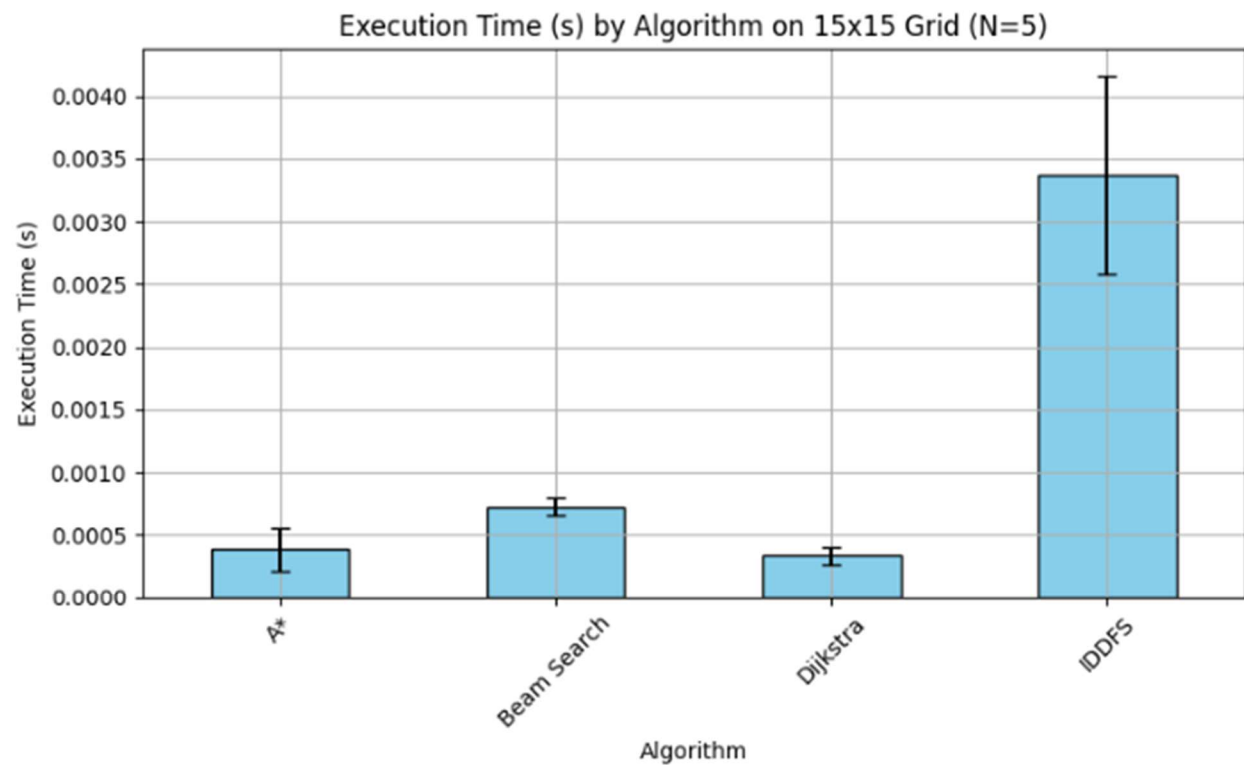
❗ Timeout or failure cases on 15x15 grid:
❌ IDDFS failed in replication 1 (Time: 0.0040s)
❌ Hill Climbing failed in replication 1 (Time: 0.0001s)
❌ IDDFS failed in replication 2 (Time: 0.0038s)
❌ Hill Climbing failed in replication 2 (Time: 0.0001s)
❌ Hill Climbing failed in replication 3 (Time: 0.0001s)
❌ Hill Climbing failed in replication 4 (Time: 0.0001s)
❌ Hill Climbing failed in replication 5 (Time: 0.0001s)
```

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.
- **Beam Search** balanced speed and performance but occasionally returned suboptimal paths.
- **IDDFS** consistently found paths but was inefficient due to redundant node exploration.
- **Hill Climbing** was the fastest in successful runs but failed frequently due to local minima.
- **Larger grid sizes** increased execution time and variability across all algorithms.
- **Statistical tests** showed no significant difference between A* and Dijkstra, though A* had better consistency.

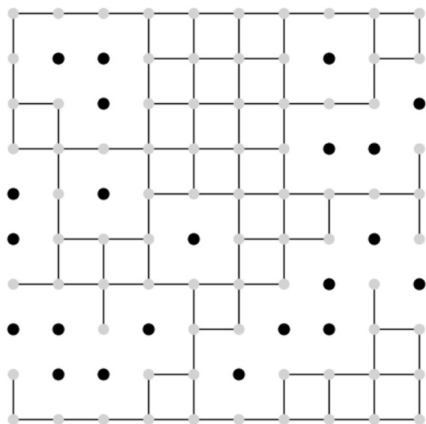
The following plots provide visual comparisons:



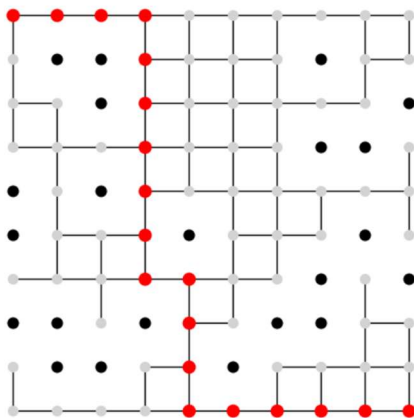


MAZE VISUALIZATION (10x10):

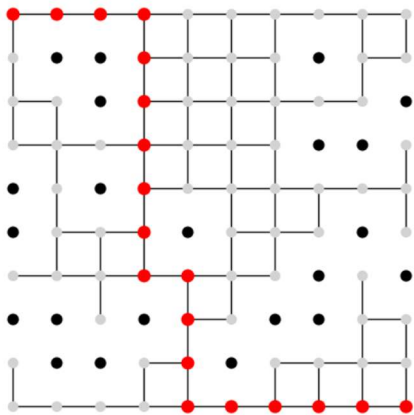
10x10 Maze Before Solving



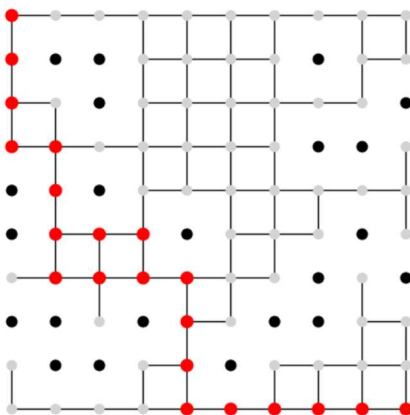
A* on 10x10



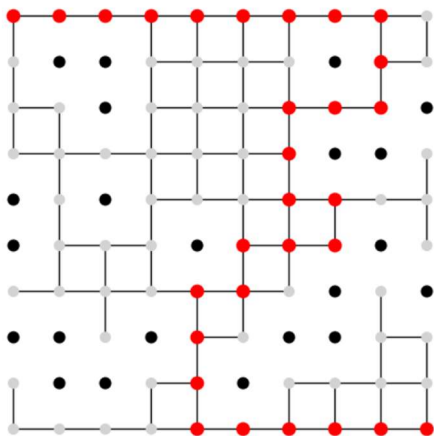
Dijkstra on 10x10



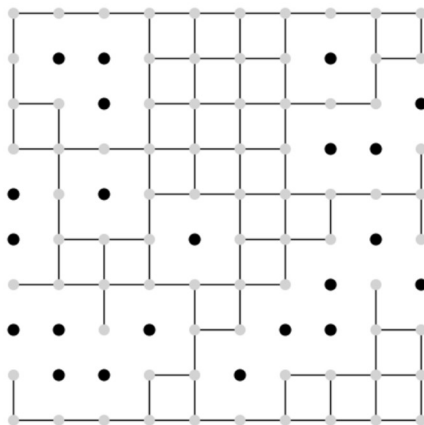
IDDFS on 10x10



Beam Search on 10x10

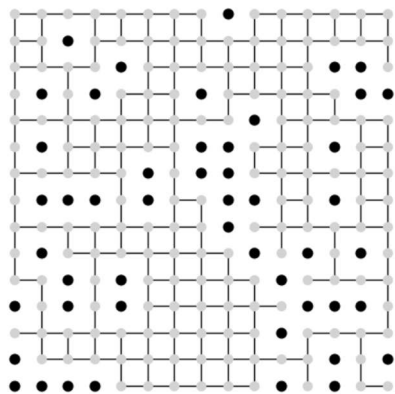


Hill Climbing on 10x10

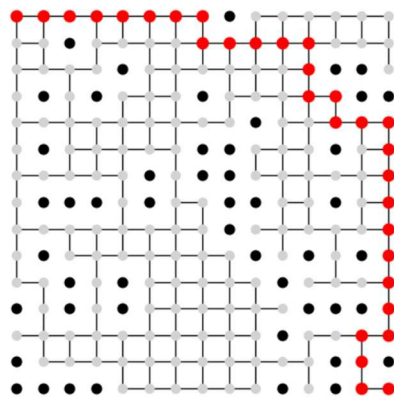


MAZE VISUALIZATION (15x15):

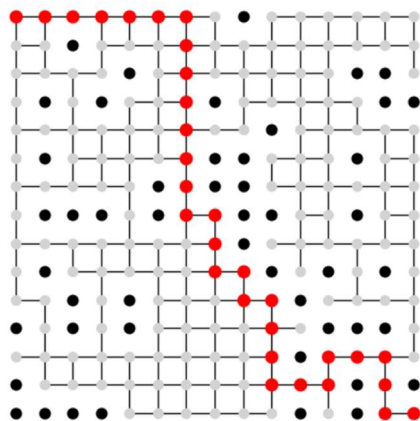
15x15 Maze Before Solving



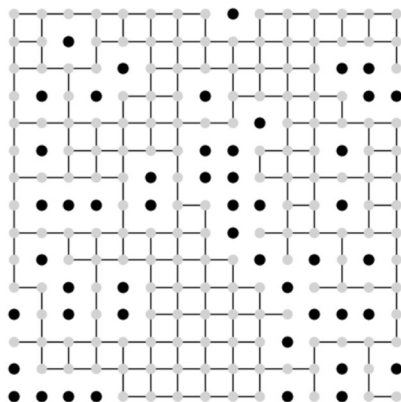
A* on 15x15



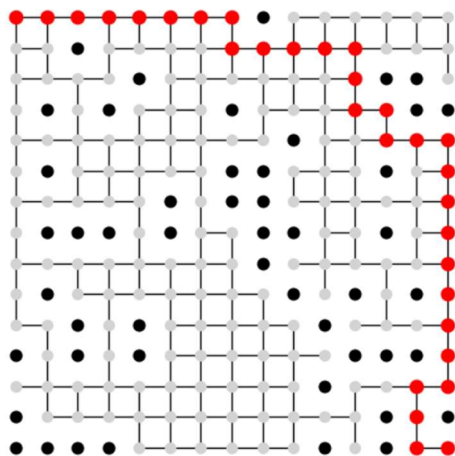
Dijkstra on 15x15



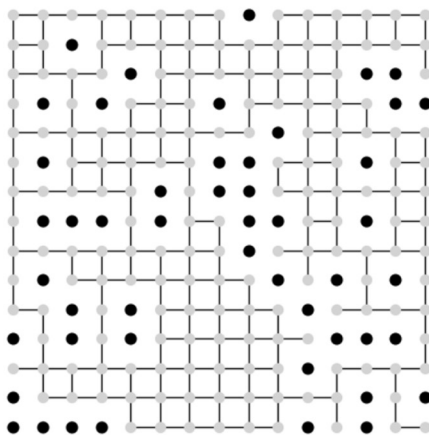
IDDFS on 15x15



Beam Search on 15x15



Hill Climbing on 15x15



SUMMARY:

This study compared five pathfinding algorithms—A*, Dijkstra’s Algorithm, IDDFS, Beam Search, and Hill Climbing—on randomized grid environments of varying sizes. The results showed that A* was the most efficient and reliable, consistently producing optimal paths with low execution time. **Dijkstra** matched A* in accuracy but was slower. **Beam Search** offered faster solutions with moderate trade-offs in path quality. **IDDFS** was complete but inefficient, and **Hill Climbing** often failed due to local minima.

Overall, heuristic-based algorithms like A* demonstrated the best balance between performance and accuracy, making them suitable for real-world applications where both speed and optimality matter.

LIMITATIONS:

- **Simplified Grid Environment:** The study was limited to static, grid-based environments with fixed start and goal positions, which may not capture the complexity of real-world navigation tasks.
- **Obstacle Distribution and Density:** Obstacles were randomly placed with a constant density, without testing more realistic patterns like clusters or narrow corridors.
- **Sample Size and Algorithm Failures:** With only five replications per setup, statistical power was limited. Algorithms like Hill Climbing frequently failed and were excluded from some analyses.

FUTURE WORK:

- **Scalability & Robustness:** Extend experiments to larger grids (e.g., 20×20, 50×50) and dynamic environments with changing obstacles or randomized start-goal pairs.
- **Real-World Applications:** Apply the algorithms to real-world graphs such as road networks or floor plans (e.g., SNAP datasets) for practical relevance.
- **Parameter Optimization:** Tune algorithm-specific parameters (e.g., beam width, depth limits) using automated techniques like grid search or Bayesian optimization.

REFERENCE:

Books & Academic Sources

- Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.
- NetworkX Developers. (n.d.). *NetworkX documentation*. Retrieved May 17, 2025, from <https://networkx.org>
- Python Software Foundation. (n.d.). *time — Time access and conversions*. Retrieved May 17, 2025, from <https://docs.python.org/3/library/time.html>

Online Tutorials & Articles

- GeeksforGeeks. (n.d.). *A Search Algorithm**. Retrieved May 17, 2025, from <https://www.geeksforgeeks.org/a-search-algorithm/>
- GeeksforGeeks. (n.d.). *Dijkstra's Shortest Path Algorithm*. Retrieved May 17, 2025, from <https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/>
- GeeksforGeeks. (n.d.). *Iterative Deepening Search (IDS) or Iterative Deepening Depth First Search (IDDFS)*. Retrieved May 17, 2025, from <https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/>
- Baeldung. (n.d.). *Beam Search Algorithm*. Retrieved May 17, 2025, from <https://www.baeldung.com/cs/beam-search>
- TechnetDeals. (n.d.). *Understanding The Algorithm Of Hill Climbing In Artificial Intelligence*. Retrieved May 17, 2025, from <https://www.technetdeals.com/hill-climbing-in-artificial-intelligence/>

YouTube Videos

- Shiffman, D. [The Coding Train]. (2017, March 30). *A Pathfinding Algorithm - Introduction** [Video]. YouTube. <https://www.youtube.com/watch?v=EaZxUCWAjb0>
- GeeksforGeeks. (2018, July 27). *Dijkstra's Algorithm Tutorial* [Video]. YouTube. <https://www.youtube.com/watch?v=XB4MIexjvY0>
- GeeksforGeeks. (2018, August 1). *Iterative Deepening Depth-First Search (IDDFS)* [Video]. YouTube. <https://www.youtube.com/watch?v=3pnd6qeIU-E>
- Baeldung. (2020, May 15). *Beam Search Algorithm Explained* [Video]. YouTube. <https://www.youtube.com/watch?v=RJJnSXzhLsk>
- Huddar, M. (2021, June 10). *Hill Climbing Algorithm with Solved Numerical Example in Artificial Intelligence* [Video]. YouTube. <https://www.youtube.com/watch?v=wM4n12FHelM>