Pathfinding Optimization

Comparative Study of Jump-Point Search and Classical Search Algorithms

Binger Yu, Vibhor Malik, Sepehr Mansouri, Yansong Jia

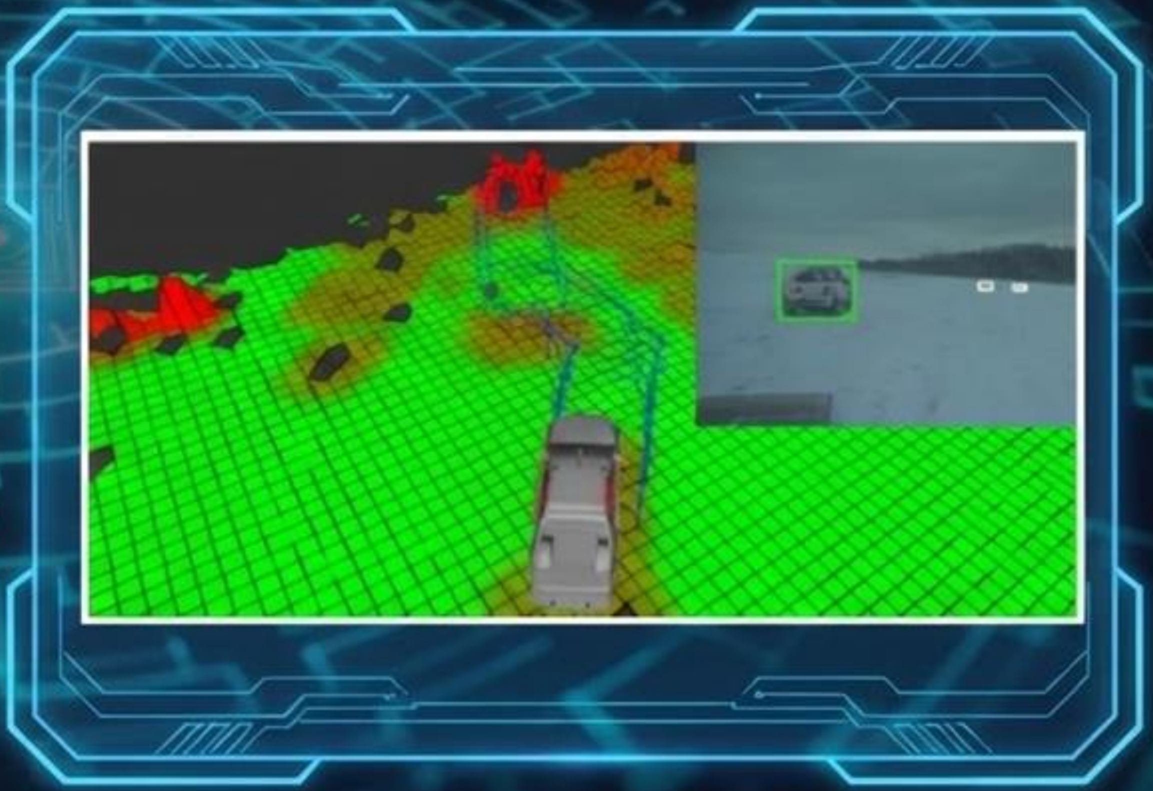
School of Computing and Academic Studies, BCIT
COMP 9060 Applied Algorithm Analysis

Poster design and layout assistance provided by Gemini (a large language model, Google) using a structured template.

Introduction: The Navigation Challenge

Why It Matters

Pathfinding is the computational backbone of modern autonomy. From robotics navigating warehouses to NPCs in video games, the speed of search directly impacts system responsiveness.



The Objective

- Compare **Modern** (JPS) vs. **Classical** (A*, Dijkstra, DFS) algorithms.
- Test in **highly structured environments** (DFS Mazes).
- Measure **Runtime**, **Optimality**, and **Node Expansions**.

Background & Related Work

Base | Modern

Dijkstra & DFS

Dijkstra: Guarantees optimal paths but explores exhaustively (high cost).

DFS: Fast and memory-efficient but produces suboptimal, winding paths.

A* Algorithm

The industry standard. Combines path cost with a heuristic (Manhattan distance) to guide the search, significantly reducing exploration.

Jump Point Search

An optimization for A* on uniform-cost grids. Skips "symmetric" path segments to reduce node expansion. Typically excels in open fields.

Background & Related Work

Depth First Search (DFS)

Core Mechanism:
- Stack-based traversal algorithm
- "visited" matrix to track nodes

Optimality:
- Not optimal for path finding
- Long, suboptimal detours

Time & Space Complexity:
- Time: O(V+E)
- Space: O(V)

Advantages:
- Simple to implement
- Low memory overhead

Limitations:
- No guarantee of shortest path
- Stuck in deep branches
- Unnecessary backtracking

Background & Related Work

Dijkstra's Algorithm

Core Mechanism:
- Priority queue
- Select the node with the smallest known cost
- Update the cost of neighboring nodes

Advantages:
- Finds the shortest path
- Weighted and unweighted grids

Optimality:
- Guaranteed optimal for non negative edge graphs
- Explore all reachable nodes

Time & Space Complexity:
- Time: O((V+E)logV)
- Space: O(V)

Heuristics:
- No dependency on heuristics

Limitations:
- Computationally expensive
- Explores irrelevant nodes
- Longer run-time

Background & Related Work

A* Algorithm

Core Mechanism:

- Heuristic-guided
- A* score = Actual cost + Admissible heuristic

Advantages:

- Balances efficiency and optimality
- Faster than Dijkstra’s in most grid-based pathfinding

Optimality:

- Optimal if heuristic is admissible

Time & Space Complexity:

- Time: From O((V+E)logV) if weak heuristic, to O(V) if strong heuristic
- Space: O(V)

Limitations:

- Performance relies on heuristic quality
- Requires prior knowledge of the target

Jump Point Search (JPS)

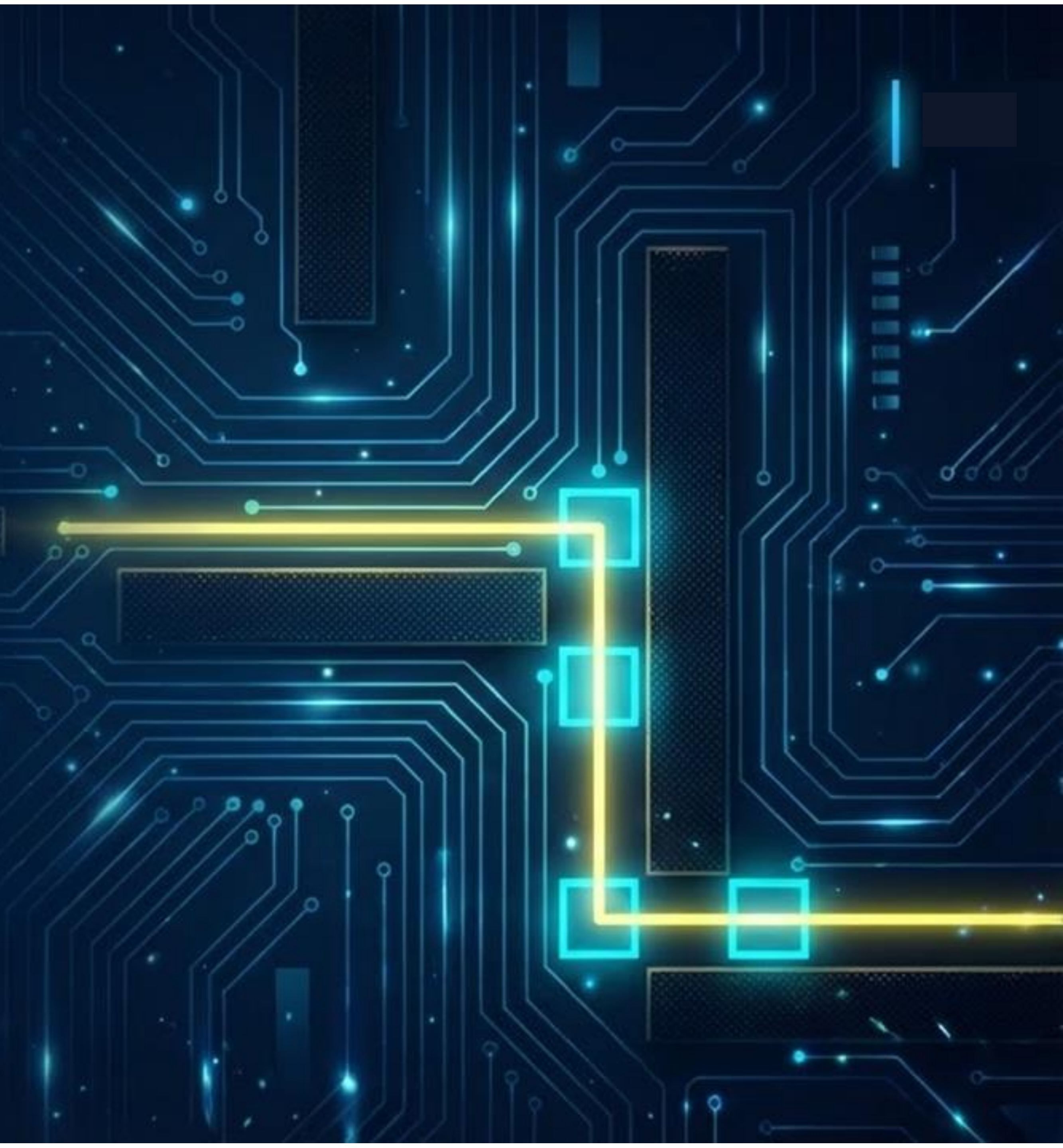
What's JPS?

- Optimization of A* algorithm
- Addresses a fundamental inefficiency in traditional grid-based pathfinding
- Maintains A*'s optimality guarantees while reducing node expansions
- Developed by Daniel Harabor & Alban Grastien

Worst case: O(b^d) **Best Case: O(d log (d))**

Core Problem

- Traditional pathfinding explores multiple equivalent paths, leading to redundant & expensive computations



Jump Point Search (JPS)

Pruning

Eliminates nodes reachable optimally without traversing current node

Jumping

Skips consecutive nodes along a straight line until it reaches a “Jump Point”

Jump Point Search (JPS)

Performance Highlights

Key Metrics

↓70% About 70% reduction in explorations

Same complexity bounds as A*

No additional memory loss

Environmental Impact

JPS performs its' best in areas/grids with high symmetry

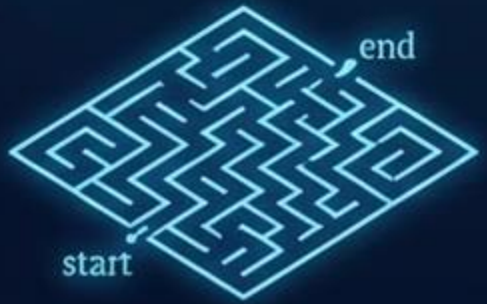
In highly constrained environments with fewer symmetric paths, benefits are reduced but still present

Methodology & Framework

Test Environment & Maze Generation

Test Environment Construction:
- **Environment**: Windows 11 (Python 3.11 + NumPy)
- **Benchmarking System:** automated logging; visualization; consistent interfaces

Maze Generation:
- Recursive DFS
- Fixed random seeds
- Variables:
  - Grid sizes
  - Map types (obstacle probabilities)
- Fixed Settings:
  - Start and end position



Algorithm Implementation

Core Implementations:
- **DFS**: Stack + visited matrix; prioritizes depth over cost
- **Dijkstra:** Binary min-heap priority queue; expands nodes by g(n) order
- **A***: Binary min-heap priority queue; Manhattan distance; prioritize f(n)=g(n)+h(n) (total cost)
- **JPS**: 8-neighborhood model; 4-connected A* recalculates final path

Experimental Design

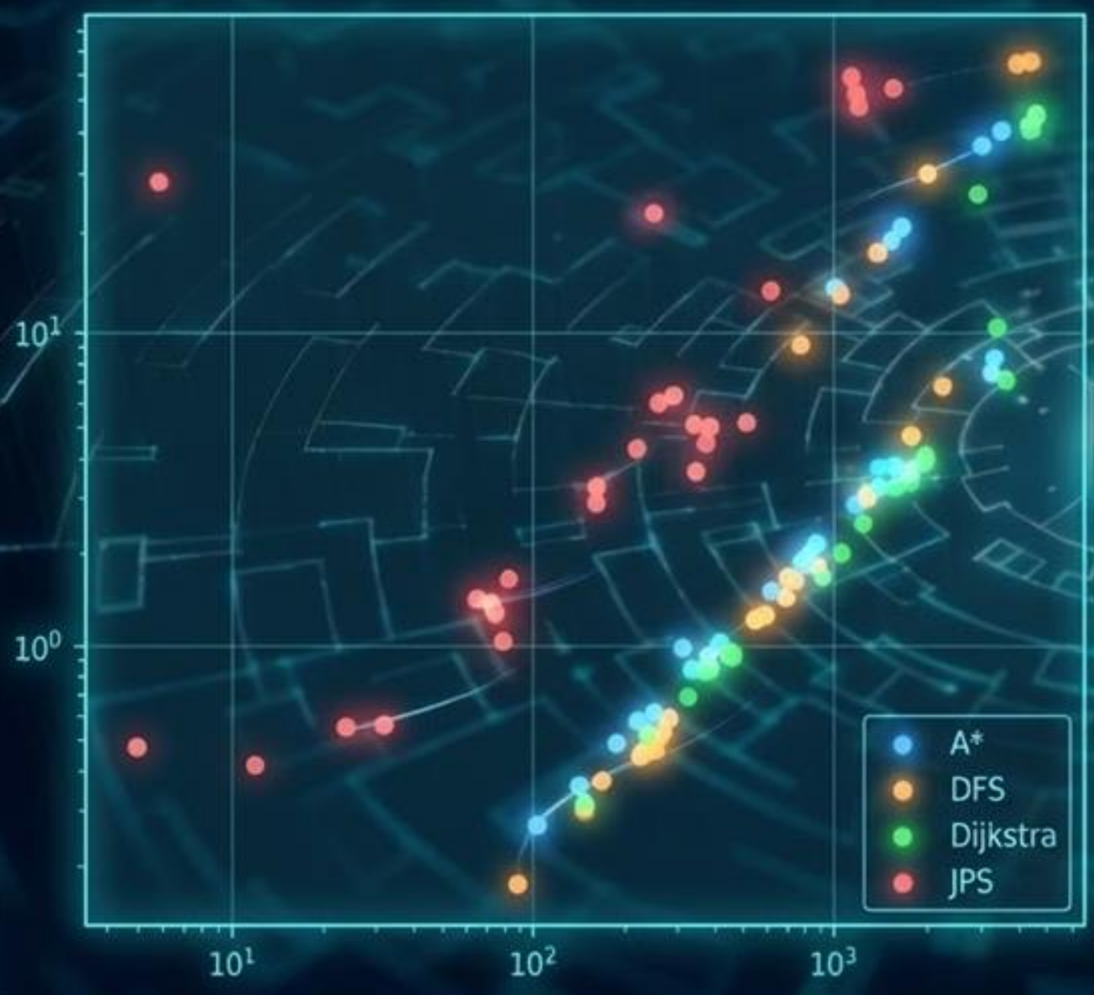
Parameters

- **Grid Sizes:** 31×31, 61×61, 91×91
- **Map Types:** Dense (Perfect Maze), Sparse 0.05, Sparse 0.10
- **Metrics:** Runtime (ms), Nodes Expanded, Path Optimality

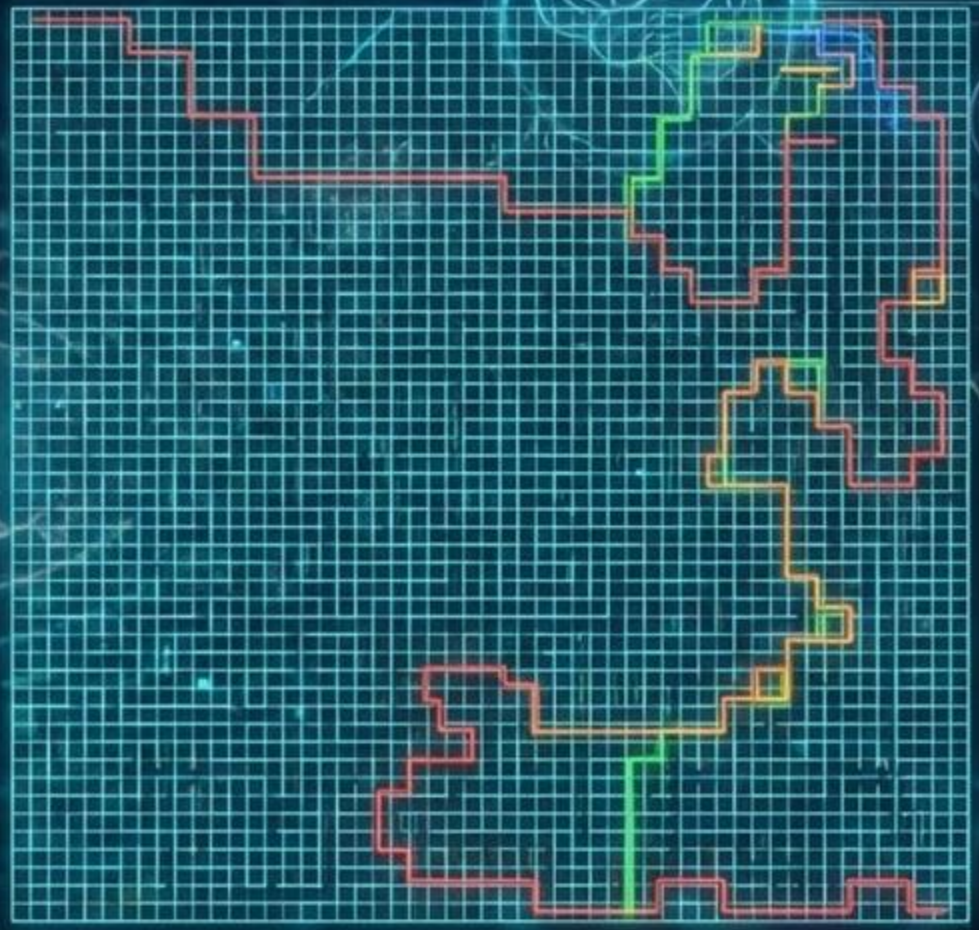
Setup Note:
Same mazes, same metrics — only the search strategy changes.

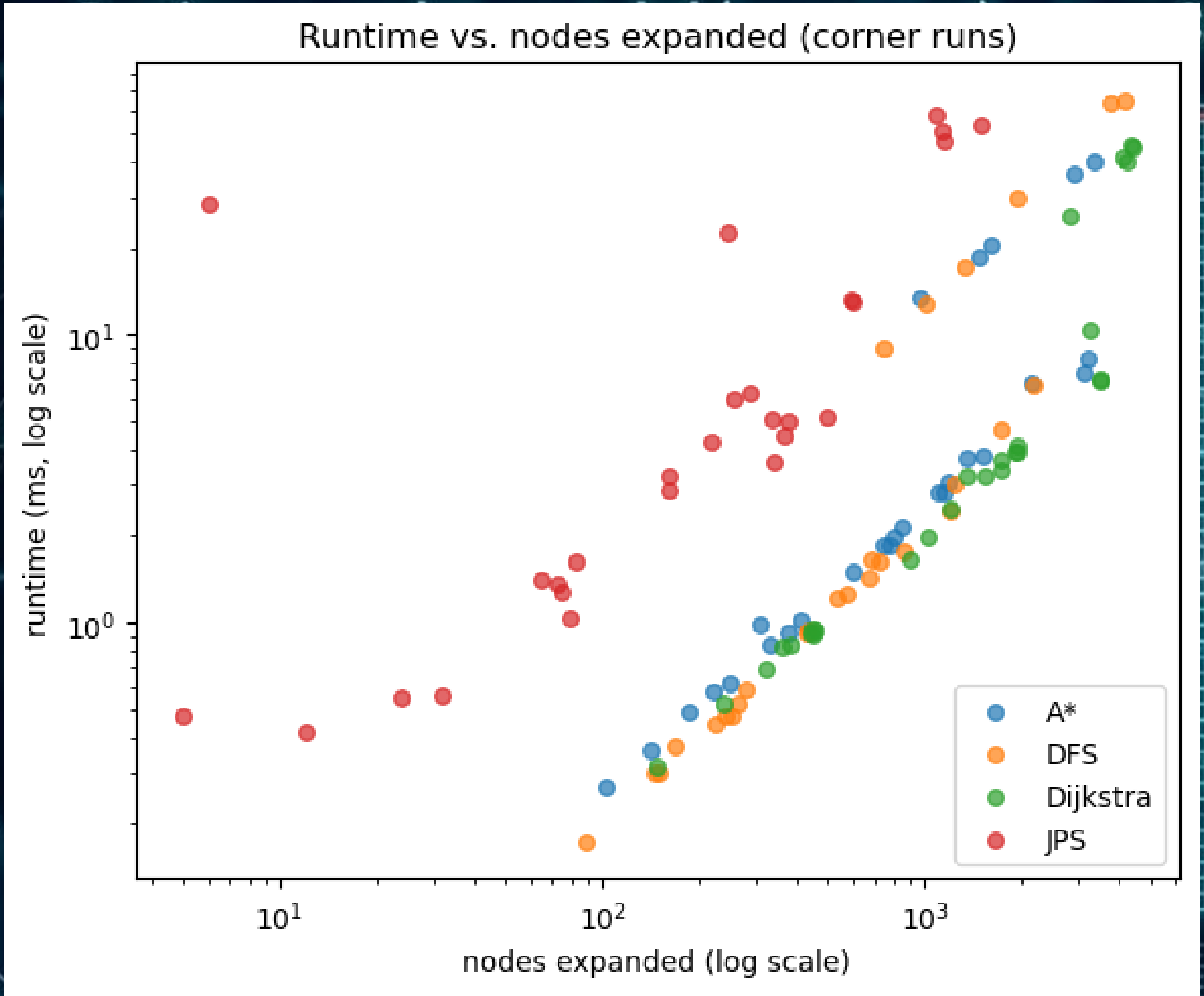
Canonical Overlays (61x61, sparse0.05)

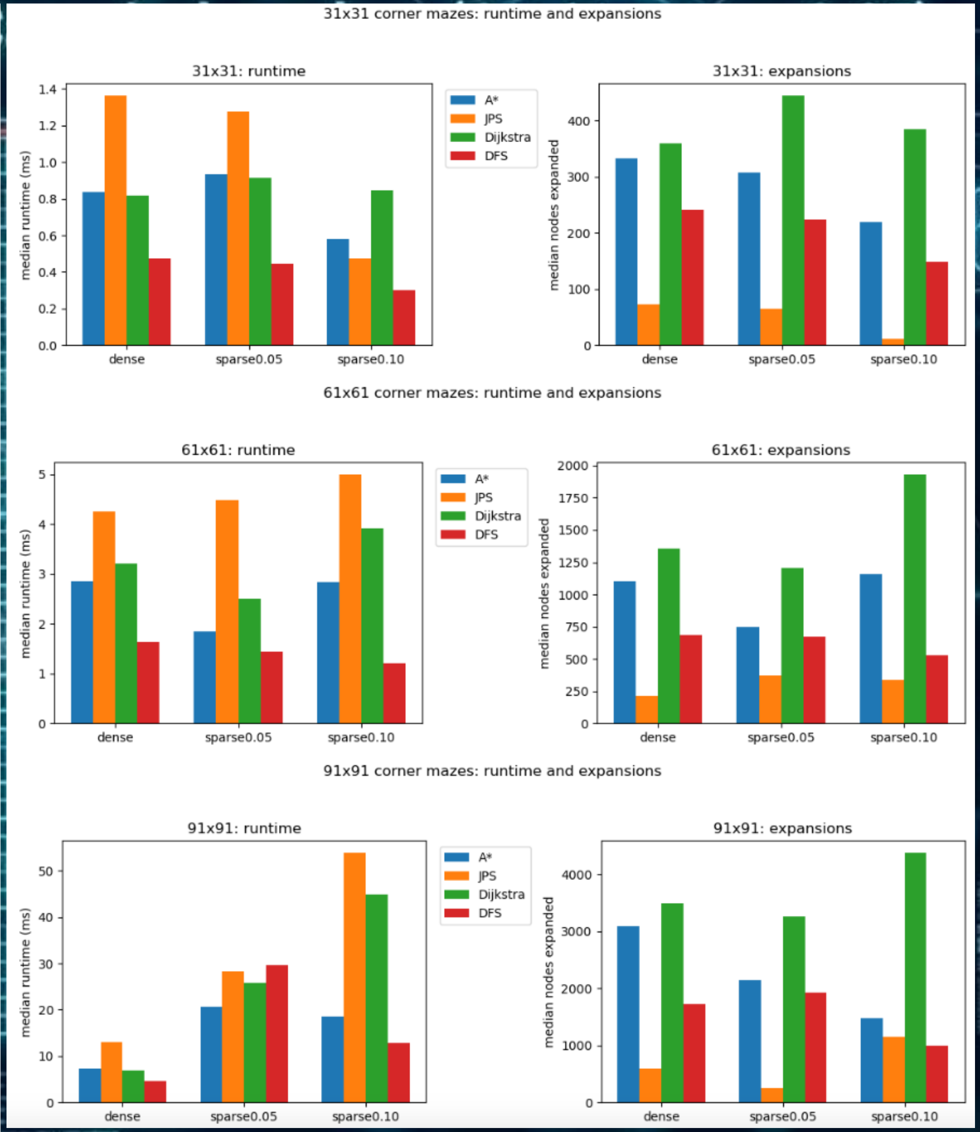
Runtime vs. nodes expanded (corner runs)



Canonical overlays (61x61, sparse0.05)




Runtime vs. nodes expanded (corner runs)
runtime (ms, log scale)
nodes expanded (log scale)
10^0
10^1
10^1
10^2
10^3
A*
DFS
Dijkstra
JPS


31x31 corner mazes: runtime and expansions
31x31: runtime
31x31: expansions
median runtime (ms)
median nodes expanded
A*
JPS
Dijkstra
DFS
dense
sparse0.05
sparse0.10
61x61 corner mazes: runtime and expansions
61x61: runtime
61x61: expansions
median runtime (ms)
median nodes expanded
A*
JPS
Dijkstra
DFS
dense
sparse0.05
sparse0.10
91x91 corner mazes: runtime and expansions
91x91: runtime
91x91: expansions
median runtime (ms)
median nodes expanded
A*
JPS
Dijkstra
DFS
dense
sparse0.05
sparse0.10

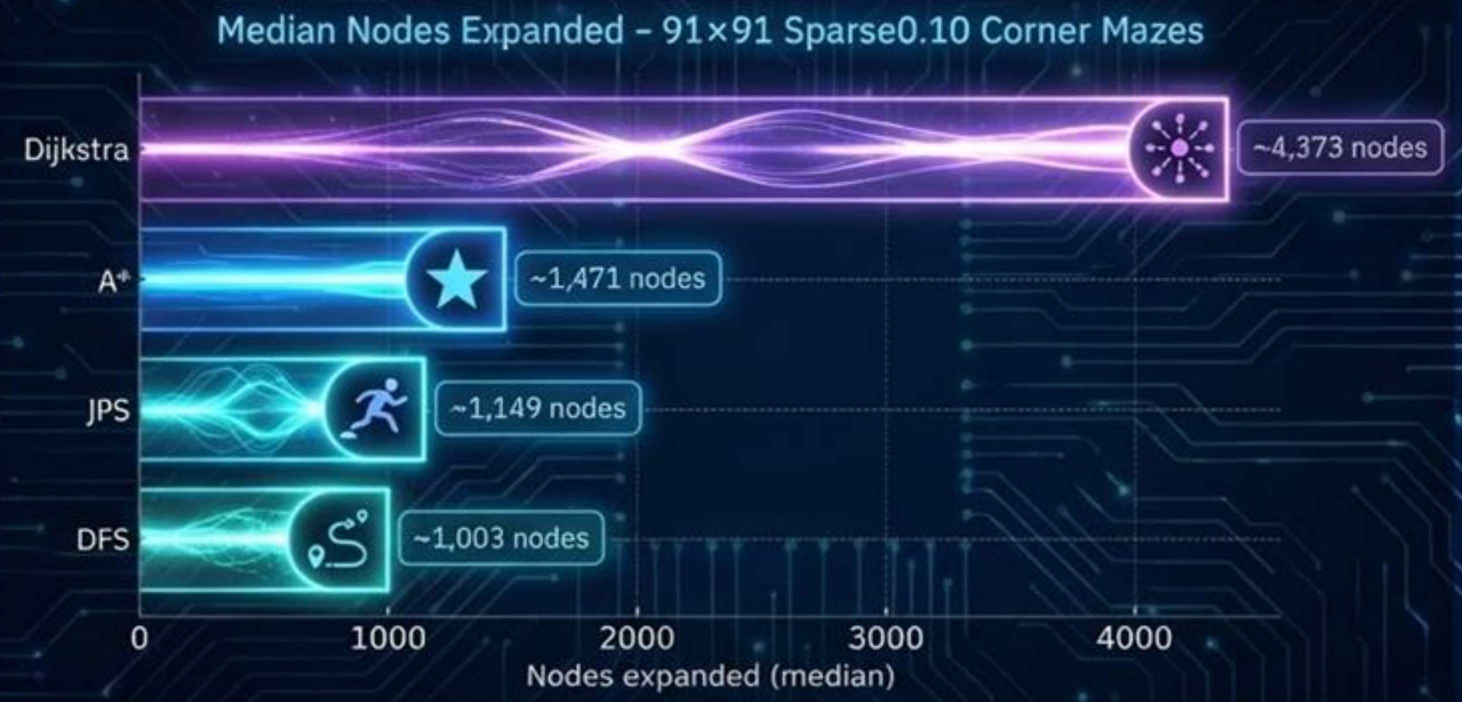
Results: Runtime Performance

Median runtime on large (91x91) sparse mazes. A* proved most consistent.

DFS is fastest but noisy; A* gives the best speed-stability trade-off, while JPS and Dijkstra lag behind.

Results: Search Efficiency

JPS is most expansion-efficient, DFS is fastest but approximate, and A* is the safest balanced choice.



Key Insight:

JPS is strictly dominated by overhead in this specific maze type but remains the most "intelligent" in terms of exploration.

Path Quality:

A*, Dijkstra, and JPS all produced optimal paths. DFS paths were ~2x longer (suboptimal).

Discussion:
The Influence of Structure

Maze Topology

Narrow corridors in DFS mazes limit JPS's ability to "jump" long distances, neutralizing its primary advantage seen in open maps.

A* Consistency

A* demonstrated the best balance. The heuristic provided sufficient guidance without the computational overhead of jump-point calculations.

DFS Limitations

While fast to implement, DFS is ill-suited for pathfinding due to extreme suboptimality (detours) in maze environments.

Conclusion & Future Work

Summary

- No single algorithm is universally superior.
- Performance is dictated by grid layout and obstacle distribution.
- A* is the robust choice for structured mazes.
- JPS is powerful but situational (requires open spaces).

Future Directions

- Test on weighted terrains and randomized obstacle fields.
- Implement JPS+ (Pre-processed jumps).
- Evaluate Bi-directional search variants.

References

[1] S. M. LaValle, Planning Algorithms. Cambridge University Press, 2006.

[2] G. Duchanois et al., “Real-time path planning in unknown environments for uavs,” in IEEE Aerospace Conference, 2012.

[3] J. Togelius et al., “Search-based procedural content generation,” in Evolutionary Computation and Games, 2016.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE Transactions on Systems Science and Cybernetics, 1968.

[5] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, 1959.

[6] D. Harabor and A. Grastien, “Online graph pruning for pathfinding on grid maps,” in AAAI, 2011.

[7] N. R. Sturtevant, “Benchmarks for grid-based pathfinding,” Transactions on Computational Intelligence and AI in Games, 2012.

[8] J. Wilson, “Maze generation using randomized algorithms,” Journal of Game Development, 2011.

Project Repository on GitHub:
https://github.com/bing-er/pathfinding-optimization