AI Search Algorithms Benchmark Report

Executive Summary

This implementation compares five search algorithms on route finding between Kansas cities. Best-First search demonstrated the fastest execution time, while A* found the most optimal paths with near-minimal node exploration.

Algorithms & Performance (Abilene → Towanda)

Algorithm	Success Rate	Avg Time (ms)	Nodes Expanded	Path Cost	Memory
Best-First	100%	0.20 ± 0.03	7	2.39	0.00 KB
A*	100%	0.54 ± 0.21	8	1.96	0.80 KB
BFS	100%	0.43 ± 0.17	19	2.49	0.00 KB
IDDFS	100%	0.81 ± 0.14	65	2.49	0.00 KB
DFS	100%	0.80 ± 0.10	41	2.90	0.00 KB

Key Findings

Best-First Search was the fastest algorithm (0.20ms) and most efficient in node exploration (7 nodes), though it found slightly suboptimal paths.

A* achieved the lowest path cost (1.96) while maintaining excellent efficiency (8 nodes expanded), demonstrating perfect balance between heuristic guidance and optimality.

BFS performed well with reasonable time (0.43ms) and found the same path cost as IDDFS, confirming its optimality guarantee for unweighted graphs.

DFS showed the highest path cost (2.90), illustrating its tendency to find longer, suboptimal routes despite good time performance.

Experimental Setup

- Dataset: Kansas Towns (46 cities)
- Test Case: Abilene → Towanda
- Runs: 5 per algorithm, Euclidean heuristic
- Metrics: Time, nodes expanded, path cost, memory

Algorithm Performance Interpretation

Best-First Search:

Best Performs When: Rapid solutions are prioritized over optimality

Why: Uses only heuristic guidance (Euclidean distance) to greedily move toward the goal

Strengths: Fastest execution (0.20ms), minimal node exploration (7 nodes)

Weaknesses: Suboptimal paths (cost 2.39 vs A*'s 1.96)

Ideal Use: Real-time applications where any reasonable path suffices

A Search:*

Best Performs When: Both optimality and efficiency matter
Why: Balances actual path cost (g(n)) with heuristic estimate (h(n))
Strengths: Lowest path cost (1.96), excellent node efficiency (8 nodes)
Weaknesses: Slightly slower than Best-First due to cost calculations
Ideal Use: Mission-critical routing where optimal paths are essential

BFS:

Best Performs When: Guaranteed optimality in unweighted graphs is required
Why: Explores all nodes at depth k before depth k+1
Strengths: Consistent performance, optimal in unweighted scenarios
Weaknesses: Higher node expansion (19 nodes) due to radial exploration
Ideal Use: Small graphs where completeness is more important than speed

DFS:

Best Performs When: Memory constraints are tight and any solution suffices
Why: Pursues paths deeply before backtracking
Strengths: Low memory usage, good for deep graphs

Weaknesses: Worst path cost (2.90), unpredictable performance
Ideal Use: Memory-constrained environments or when exploring solution spaces

IDDFS:

Best Performs When: You want BFS optimality with DFS memory efficiency
Why: Repeated depth-limited DFS with increasing limits
Strengths: Combines completeness with reasonable memory
Weaknesses: Highest node expansion (65 nodes) due to repeated work
Ideal Use: Unknown depth problems with memory constraints

Performance Trade-offs Summary

Speed vs Optimality Trade-off:

- Fastest: Best-First (0.20ms) but suboptimal
- Most Optimal: A* (cost 1.96) with good speed
- Balanced: BFS reasonable speed with guaranteed optimality

Memory vs Completeness:

- Low Memory: DFS and Best-First
- High Completeness: BFS and IDDFS
- Best Balance: A*

Conclusion

The choice of search algorithm depends critically on the problem constraints. Best-First excels when speed is paramount, A* dominates when optimality matters, and BFS provides reliable baseline performance. For the Kansas cities routing problem, A* emerges as the superior choice, offering near-optimal paths with efficient search characteristics.