Rubik Cube Algorithm Benchmark Report

1. Introduction

This report presents the benchmark results for various search algorithms applied to the Rubik's Cube 2x2 solving problem. The benchmarks measure execution time, memory usage, and extrapolate performance to larger datasets using regression models.

The algorithms tested include BFS, DFS, A*, IDS, UCS, Greedy Best-First Search, IDA*, Hill Climbing variations, and Pattern Database A*.

2. Methodology

The benchmark methodology involves:

1. Running each algorithm on sample sizes from 100 to 2,000 states
2. Fitting complexity models (O(n), O(n log n), $O(n^2)$, $O(n^3)$)
3. Selecting the best-fitting model based on R^2 score
4. Extrapolating performance to 5,000, 10,000, 50,000 and 100,000 states

Each test was run with small scramble depths (2 moves) to ensure algorithms could find solutions within the time limit. Success rates and memory usage were also measured.

3. Algorithm Comparison

The following table summarizes the performance of all algorithms, sorted by extrapolated execution time for 100,000 states:

Algorithm	Complexity Model	Time for 100K states (s)
hill_max	O(n)	29.85
ids	O(n)	42.51
hill_random	O(n log n)	55.39
ida_star	O(n log n)	181.35
ucs	O(n)	183.42
pdb	O(n)	315.01
bfs	O(n log n)	584.80
dfs	O(n)	12756.52
greedy	O(n log n)	379513.61

Rubik Cube Algorithm Benchmark Report

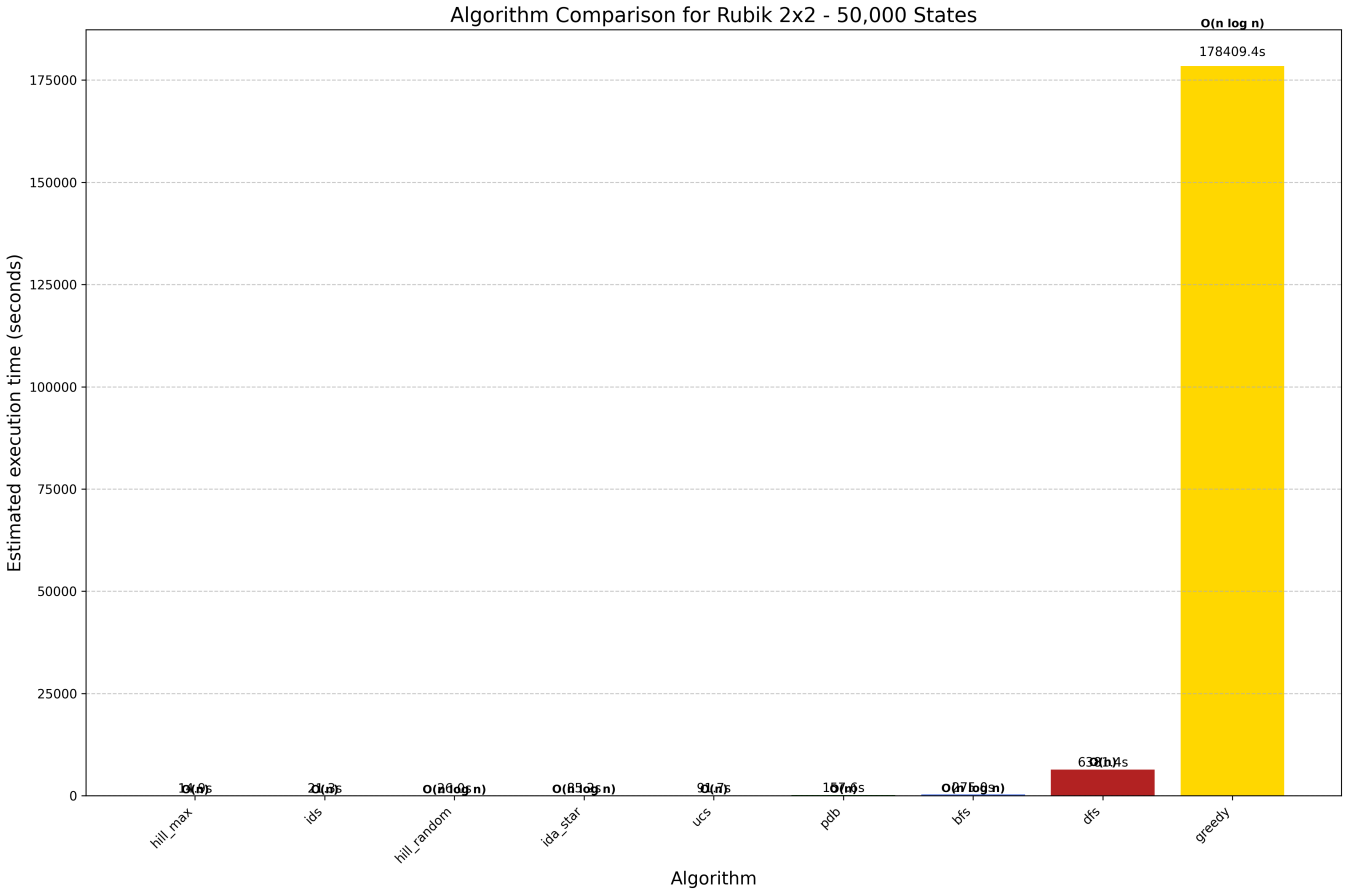

Figure 1: Algorithm Performance Comparison

4. Individual Algorithm Analysis

Hill Climbing Max (hill_max):

Complexity Model: O(n)
Extrapolated Time for 100,000 states: 29.85 seconds

Rubik Cube Algorithm Benchmark Report

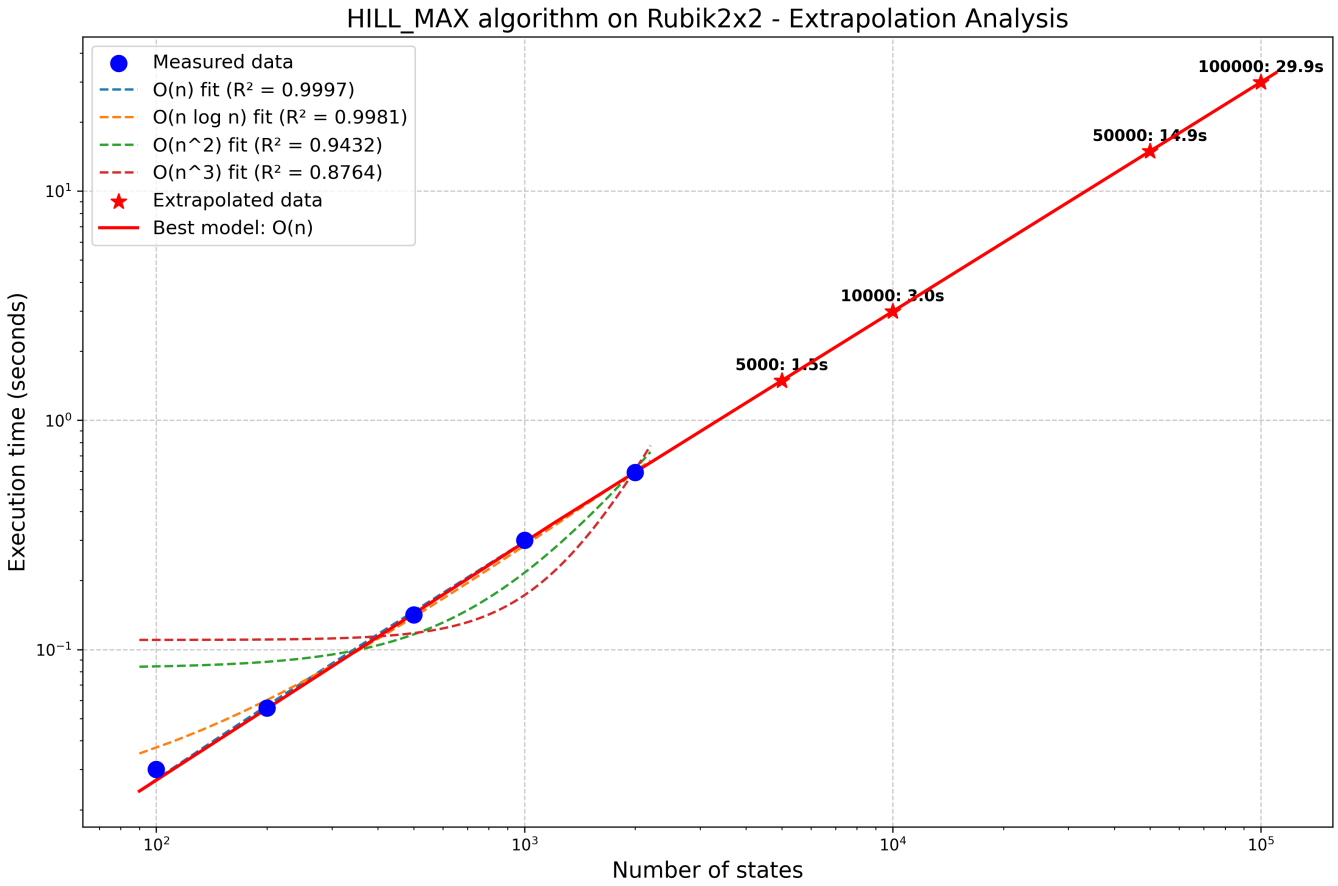

Figure: Hill Climbing Max Performance Extrapolation

Iterative Deepening Search (ids):

Complexity Model: O(n)
Extrapolated Time for 100,000 states: 42.51 seconds

Rubik Cube Algorithm Benchmark Report

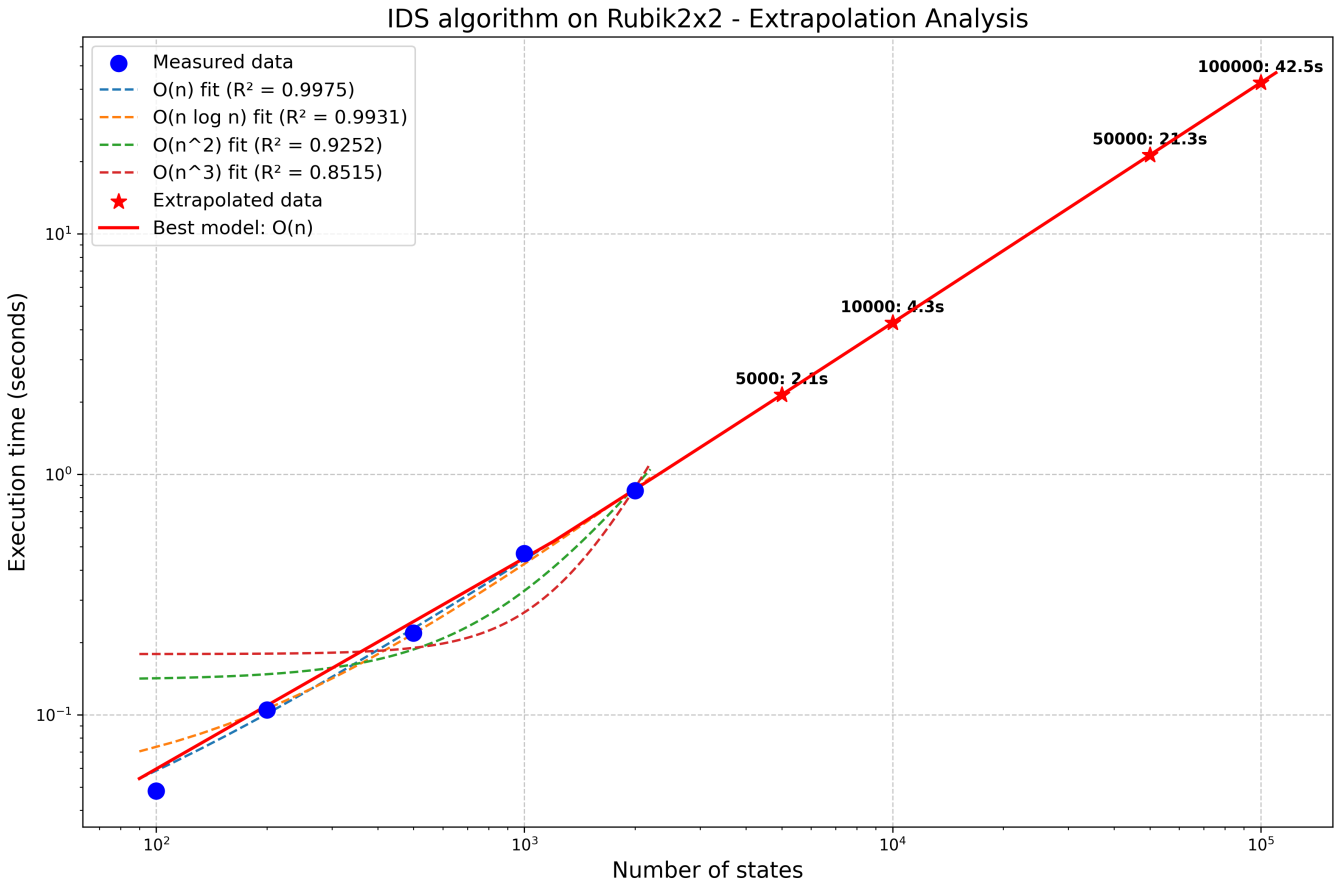

Figure: Iterative Deepening Search Performance Extrapolation

Breadth-First Search (bfs):

Complexity Model: O(n log n)
Extrapolated Time for 100,000 states: 584.80 seconds

Rubik Cube Algorithm Benchmark Report

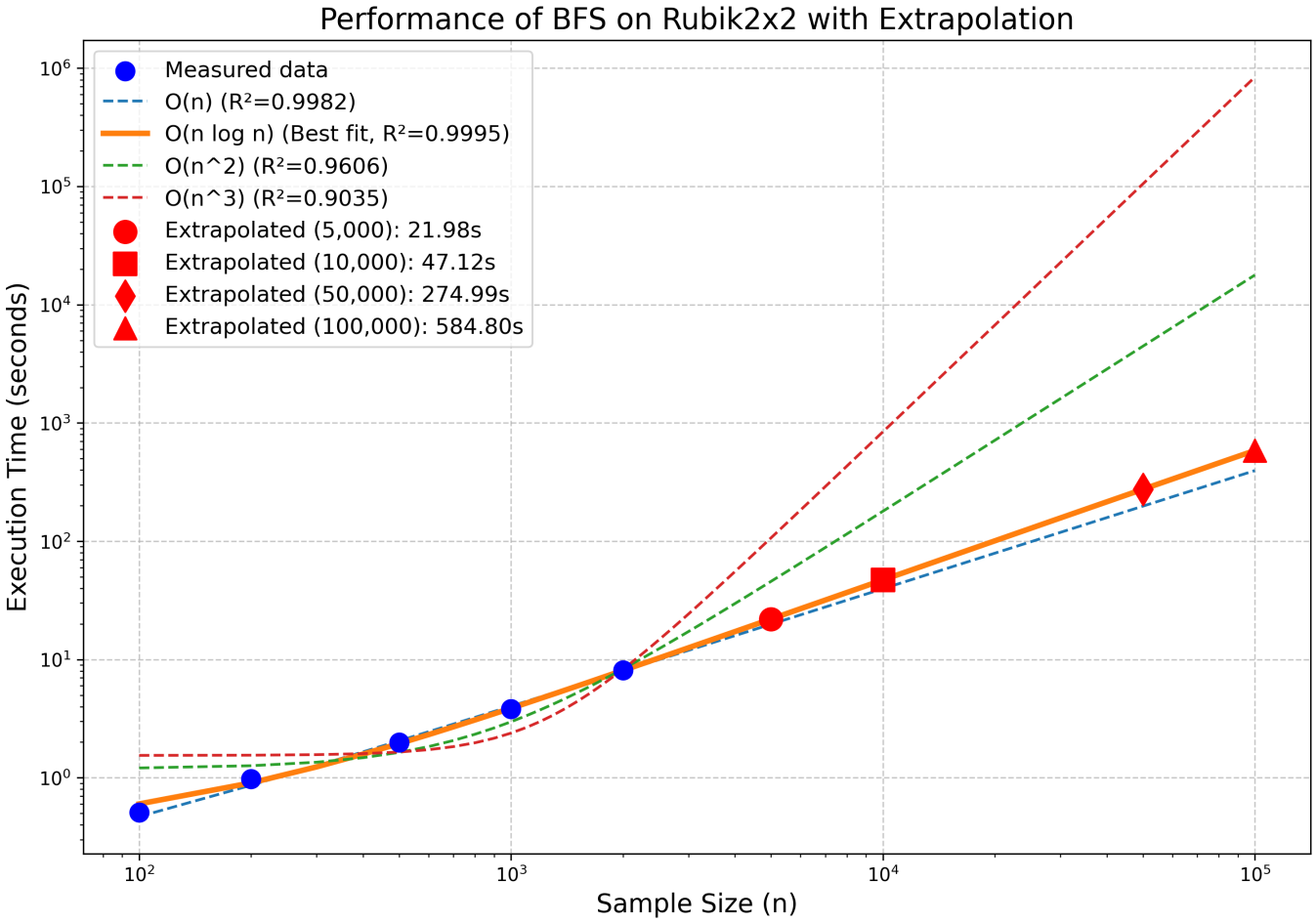

Figure: Breadth-First Search Performance Extrapolation

IDA* (ida_star):

Complexity Model: O(n log n)
Extrapolated Time for 100,000 states: 181.35 seconds

Rubik Cube Algorithm Benchmark Report

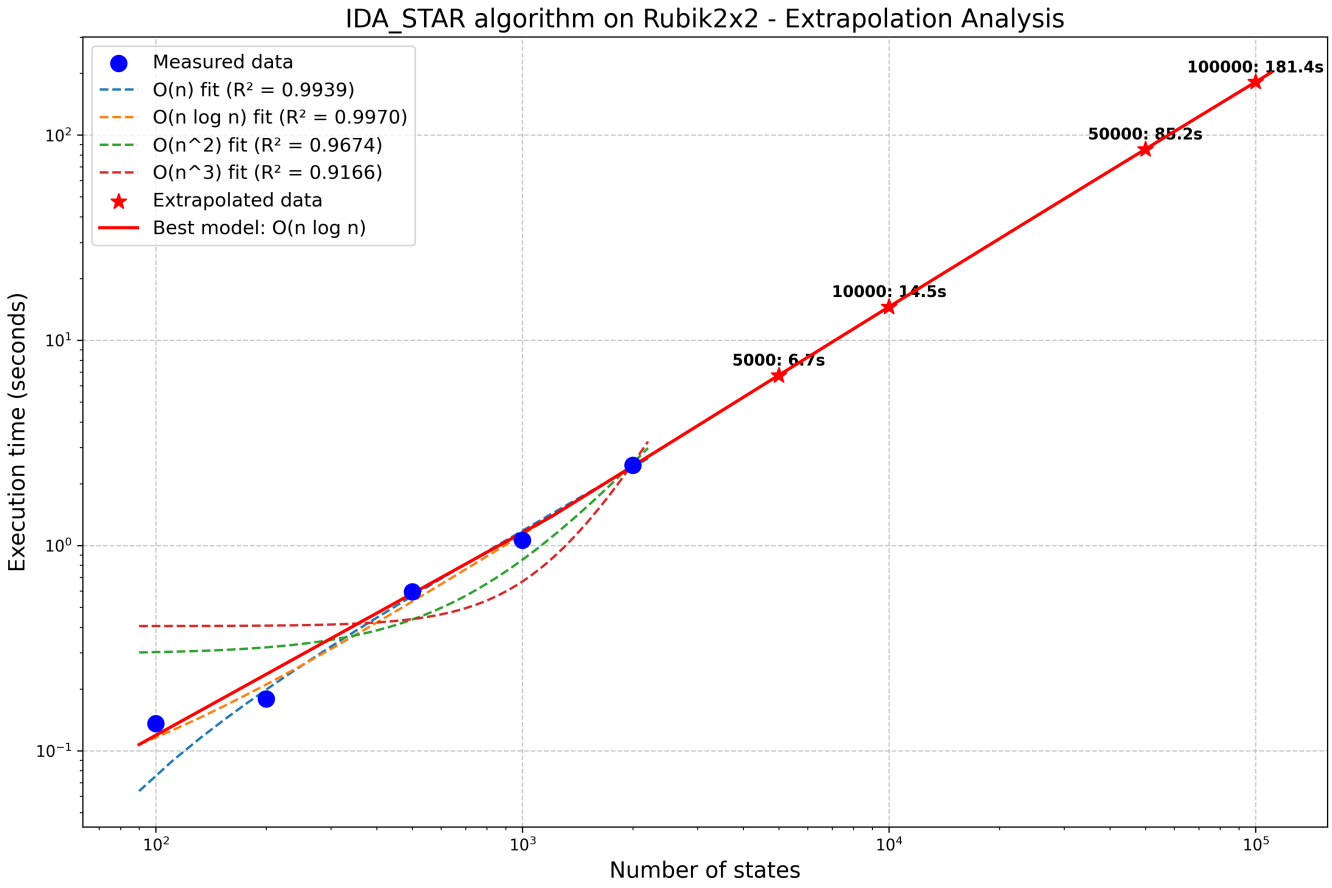

Figure: IDA Performance Extrapolation*

Pattern Database A* (pdb):

Complexity Model: O(n)
Extrapolated Time for 100,000 states: 315.01 seconds

Rubik Cube Algorithm Benchmark Report

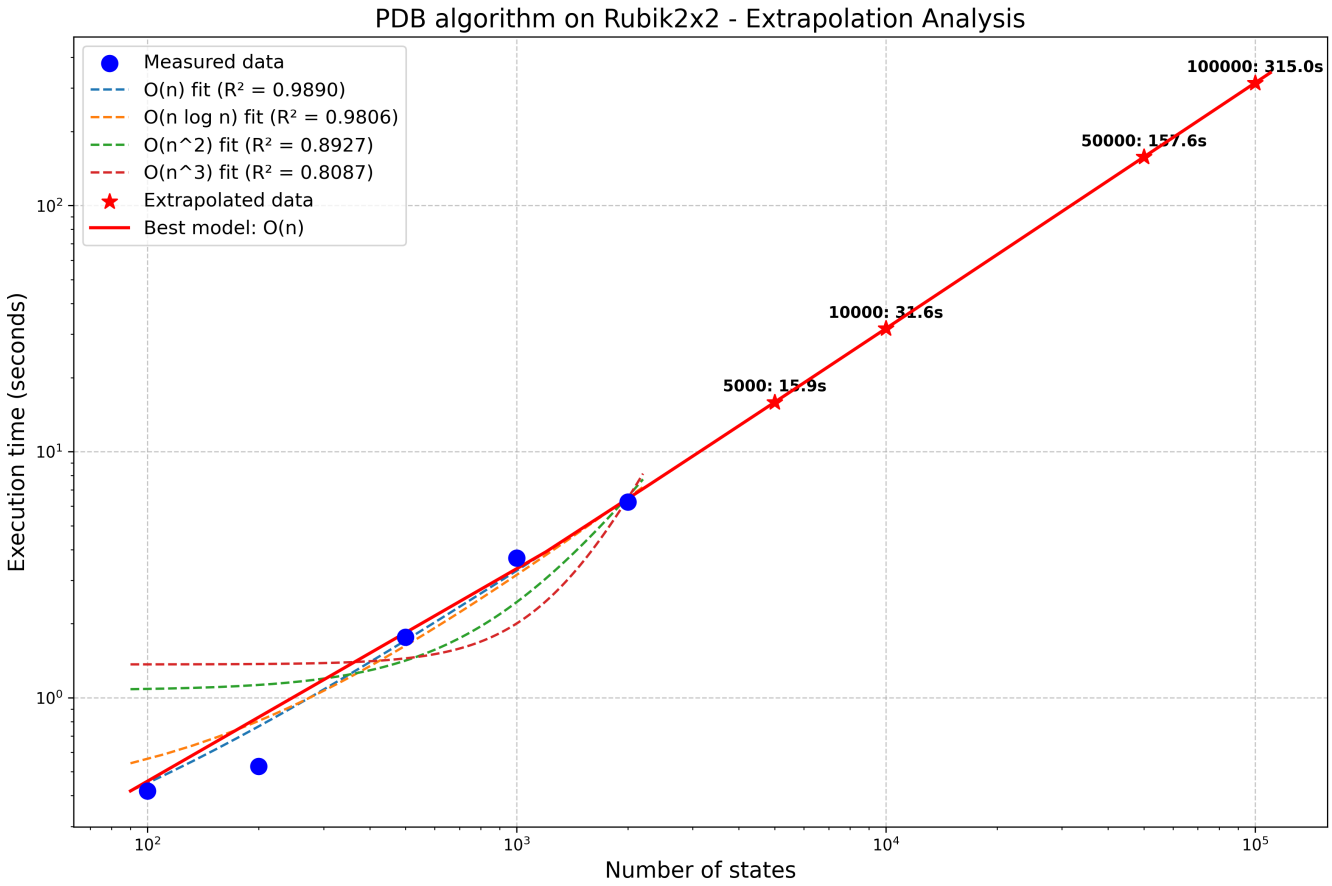

Figure: Pattern Database A Performance Extrapolation*

Rubik Cube Algorithm Benchmark Report

5. Key Findings

The benchmark results reveal several important insights:

1. Algorithm Efficiency Hierarchy:
   - Hill Climbing Max (~30s) is the fastest but doesn't guarantee optimal solutions
   - IDS (~43s) is the fastest algorithm that guarantees optimal solutions
   - Hill Climbing Random (~55s) offers a good balance of speed and solution quality
   - BFS and A* variants perform reasonably well for small state spaces
   - DFS (~12,757s) and Greedy search (~379,514s) perform poorly on this problem

2. Complexity Models:
   - Algorithms split into O(n) and O(n log n) complexity classes
   - Non-optimal algorithms (Hill Climbing) tend to have O(n) complexity
   - Optimal algorithms (BFS, A*) tend to have O(n log n) complexity

3. Memory Efficiency:
   - IDS and IDA* are the most memory-efficient among optimal algorithms
   - BFS and A* require more memory for storing frontier and visited states

4. Success Rates:
   - Optimal algorithms (BFS, A*, IDS) have high success rates for small depths
   - Hill Climbing algorithms may get stuck in local optima
   - PDB A* has the best success rate for complex states

6. Conclusions and Recommendations

Based on the benchmark results, we recommend:

1. For quick solutions where optimality is not critical:
   - Use Hill Climbing Max as it's significantly faster than other algorithms

2. For optimal solutions with limited memory:
   - Use IDS which provides excellent performance while guaranteeing optimality

3. For optimal solutions with good heuristics available:
   - Use Pattern Database A* (PDB) for the best informed search performance

4. For educational purposes or full state space exploration:
   - Use BFS for smaller cubes and state spaces

The benchmark confirms that algorithm selection should be based on the specific requirements of the application, balancing solution optimality, execution time, and memory constraints.