# Graph Search Algorithm Performance Report

## Executive Summary

This report presents the results of a benchmarking experiment (located in the ‘output-files’ folder) comparing three fundamental graph search algorithms: **Depth-First Search (DFS)**, **Breadth-First Search (BFS)**, and **Dijkstra’s Algorithm**. The algorithms were tested on directed weighted graphs of varying sizes (100, 1,000, and 10,000 edges) to evaluate their performance across three key metrics: runtime, step count, and path cost.

## Experimental Results

### Performance Metrics by Graph Size

#### 100 Edges Graph

| Algorithm | Runtime (ms) | Steps | Path Cost |
| --- | --- | --- | --- |
| DFS | 3 | 22 | 164 |
| BFS | 1 | 26 | 81 |
| Dijkstra | 11 | 41 | 62 |

#### 1,000 Edges Graph

| Algorithm | Runtime (ms) | Steps | Path Cost |
| --- | --- | --- | --- |
| DFS | 2 | 233 | 2,361 |
| BFS | 0 | 27 | 82 |
| Dijkstra | 0 | 32 | 82 |

#### 10,000 Edges Graph

| Algorithm | Runtime (ms) | Steps | Path Cost |
| --- | --- | --- | --- |
| DFS | 54 | 523 | 5,383 |
| BFS | 10 | 3,193 | 228 |
| Dijkstra | 17 | 1,888 | 191 |

## Analysis and Discussion

### Path Optimality

The results clearly demonstrate that **Dijkstra’s algorithm consistently produces the most optimal paths** in terms of total cost. Across all three graph sizes, Dijkstra achieved the lowest path costs (62, 82, and 191 for the 100, 1k, and 10k edge graphs respectively). This is exactly what we expect from Dijkstra’s algorithm—it is specifically designed to find the shortest path in weighted graphs by considering edge weights during traversal.

BFS performed reasonably well on path cost for smaller graphs, achieving the same optimal cost as Dijkstra on the 1,000-edge graph (82). However, as graph complexity increased to 10,000 edges, BFS’s path cost increased to 228, demonstrating that while BFS finds the shortest path in terms of number of edges, it does not account for edge weights and therefore cannot guarantee the minimum-cost path in weighted graphs.

DFS performed poorly in terms of path optimality, producing the highest costs across all test cases (164, 2,361, and 5,383). This is expected behavior since DFS explores deeply into the graph without regard for path cost or even path length. The exponential growth in DFS’s path cost (from 164 to 5,383) illustrates how its blind exploration can lead to extremely inefficient paths in larger graphs.

### Computational Efficiency

When examining runtime performance, BFS emerged as the fastest algorithm for smaller graphs, completing in just 1ms for 100 edges and 0ms (sub-millisecond) for 1,000 edges. Even on the 10,000-edge graph, BFS completed in only 10ms. This efficiency stems from BFS’s straightforward queue-based implementation and its ability to terminate as soon as the target is found at the current level.

DFS showed competitive runtime performance on smaller graphs (3ms and 2ms for 100 and 1k edges), but its runtime scaled poorly to 54ms on the 10,000-edge graph—more than 5× slower than BFS and 3× slower than Dijkstra. This degradation occurs because DFS can explore deep, unproductive paths before backtracking.

Interestingly, Dijkstra’s algorithm had the highest runtime on the 100-edge graph (11ms) despite finding the optimal path. This overhead comes from the priority queue operations required to continuously select the next minimum-cost node. However, Dijkstra’s runtime scaled remarkably well, taking only 17ms on the 10,000-edge graph—faster than DFS and only slightly slower than BFS while maintaining optimal path quality.

### Step Count and Search Efficiency

The step count metric reveals how thoroughly each algorithm explores the graph. DFS exhibited the lowest step counts for the smallest graphs (22 and 233 steps), suggesting it happened to find relatively direct paths through random exploration. However, this efficiency is misleading—DFS’s low step count doesn’t correlate with finding good paths, as evidenced by its poor path costs.

BFS maintained remarkably consistent step counts on the smaller graphs (26 and 27 steps), reflecting its systematic level-by-level exploration. However, on the 10,000-edge graph, BFS’s step count exploded to 3,193—the highest of all algorithms. This indicates that BFS explored a massive portion of the graph before finding the target, likely because the optimal path by edge count was quite long.

Dijkstra showed moderate step counts that scaled more gracefully: 41, 32, and 1,888 steps. While higher than DFS for smaller graphs, Dijkstra’s intelligent prioritization of lower-cost paths means it explores more judiciously than BFS’s exhaustive approach, resulting in fewer steps than BFS on the largest graph while still guaranteeing optimality.

## Practical Applications and Context

The performance characteristics observed in this experiment have significant implications for real-world applications:

### Network Routing Context

If these graphs represented a network of routers where edge weights indicate transmission latency in milliseconds, Dijkstra’s algorithm would be the clear choice. Network routing protocols like OSPF (Open Shortest Path First) use Dijkstra’s algorithm for exactly this reason—finding the path with minimum total latency is critical for network performance. While Dijkstra takes slightly longer to compute (17ms vs 10ms for BFS on 10k edges), this one-time computational cost is trivial compared to the ongoing savings from routing packets along a path that costs 191ms instead of 228ms (BFS) or 5,383ms (DFS). In a network handling millions of packets, these savings compound dramatically.

### Video Game Context

In a video game scenario where nodes represent player positions and edges represent distances between locations, the choice of algorithm depends on the game’s requirements. For a competitive multiplayer game where players need to find the shortest physical path to reach other players quickly, Dijkstra would again be optimal. The 17ms computation time is well within acceptable frame budgets (typically 16ms for 60fps games), and finding the truly shortest path improves gameplay quality.

However, if the game needed to generate enemy patrol routes or NPC behavior that should appear more natural and unpredictable, DFS’s tendency to explore random deep paths might actually be advantageous for creating varied, less optimal movement patterns. Similarly, BFS might be preferred for implementing game mechanics like “flood fill” area-of-effect abilities or visibility calculations where all reachable positions within a certain range need to be found.

### Scale Considerations

The experiment reveals that algorithm choice becomes increasingly critical as graph size grows. The relatively small performance differences on 100-edge graphs become pronounced at 10,000 edges. For truly massive graphs (millions of nodes, as found in social networks or large-scale logistics systems), these scaling behaviors would amplify further. The data suggests that while BFS remains fast, its step count explosion could become problematic. Dijkstra’s balanced scaling of both runtime and path quality makes it robust for production systems operating at scale.

## Conclusion

This benchmarking experiment confirms the theoretical properties of these fundamental graph algorithms while revealing practical performance trade-offs. Dijkstra’s algorithm proves its value as the gold standard for shortest-path problems in weighted graphs, offering optimal path costs with acceptable computational overhead. BFS excels in runtime efficiency but cannot guarantee cost-optimality in weighted graphs and may explore excessively in large, densely-connected networks. DFS, while fast on small graphs, scales poorly and produces highly suboptimal paths, making it unsuitable for pathfinding applications but potentially useful for exhaustive exploration tasks.

The choice of algorithm must ultimately align with application requirements: use Dijkstra when path optimality is paramount, BFS when runtime is critical and weights don’t matter, and reserve DFS for scenarios requiring deep exploration or where path optimality is explicitly undesired.