Computer Science: Graph Algorithms and Network Analysis

Computational Science Templates

November 24, 2025

Abstract

This document presents a comprehensive analysis of fundamental graph algorithms including shortest path algorithms (Dijkstra, Bellman-Ford, Floyd-Warshall), minimum spanning trees (Prim, Kruskal), graph traversal (BFS, DFS), and network flow algorithms. We implement these algorithms in Python and analyze their time complexity, correctness, and practical applications in network routing, social network analysis, and optimization problems.

1 Introduction

Graph algorithms are fundamental to computer science and find applications in networking, social media analysis, logistics, and artificial intelligence. This analysis covers both theoretical foundations and practical implementations of key algorithms for path finding, tree construction, and network analysis.

2 Mathematical Framework

2.1 Graph Representation

A graph $G = (V, E)$ consists of vertices V and edges E. For weighted graphs, each edge (u, v) has weight $w(u, v)$.

2.2 Dijkstra's Algorithm

For non-negative edge weights, Dijkstra's algorithm computes shortest paths:

$$d[v] = \min_{u \in \text{adj}(v)} \{d[u] + w(u, v)\} \tag{1}$$

Time complexity: $O((V + E) \log V)$ with priority queue.

2.3 Bellman-Ford Algorithm

Handles negative weights and detects negative cycles:

$$d^{(k)}[v] = \min_{u}\{d^{(k-1)}[v], d^{(k-1)}[u] + w(u,v)\} \tag{2}$$

Time complexity: $O(VE)$.

2.4 Minimum Spanning Tree

For a connected weighted graph, MST minimizes total edge weight:

$$\text{MST}(G) = \arg\min_{T \subset E} \sum_{(u,v)\in T} w(u,v) \tag{3}$$

subject to T forming a spanning tree.

3 Computational Analysis

3.1 Dijkstra's Algorithm Implementation

3.2 Bellman-Ford Algorithm

3.3 Minimum Spanning Tree - Prim's Algorithm

3.4 Kruskal's Algorithm

3.5 Graph Traversal: BFS and DFS

3.6 Visualization of Graph Algorithms

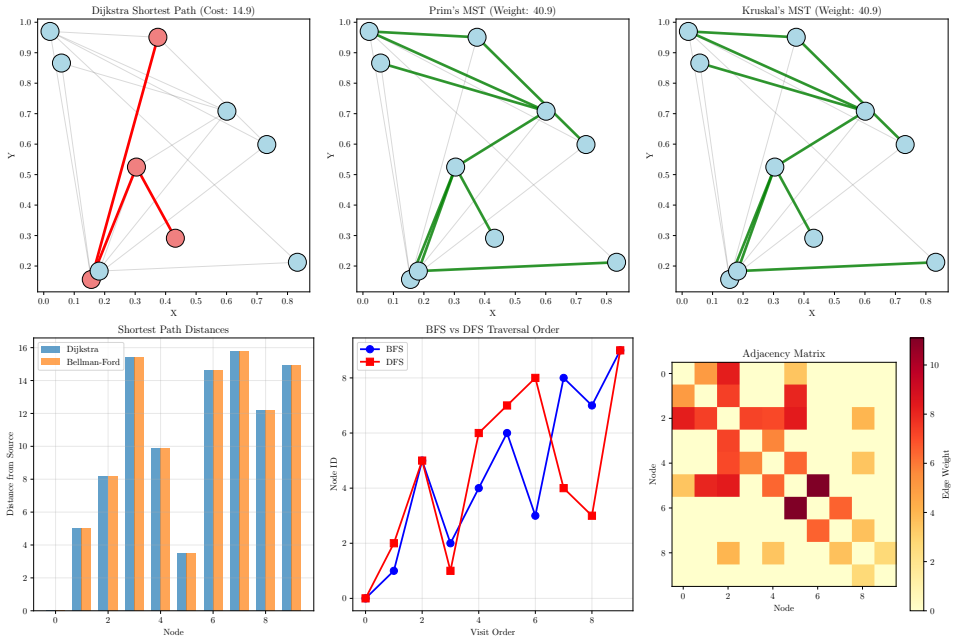

Figure 1: Graph algorithm results: (a) Dijkstra shortest path, (b) Prim MST, (c) Kruskal MST, (d) distance comparison, (e) traversal order, (f) adjacency matrix.

3.7 Algorithm Complexity Analysis

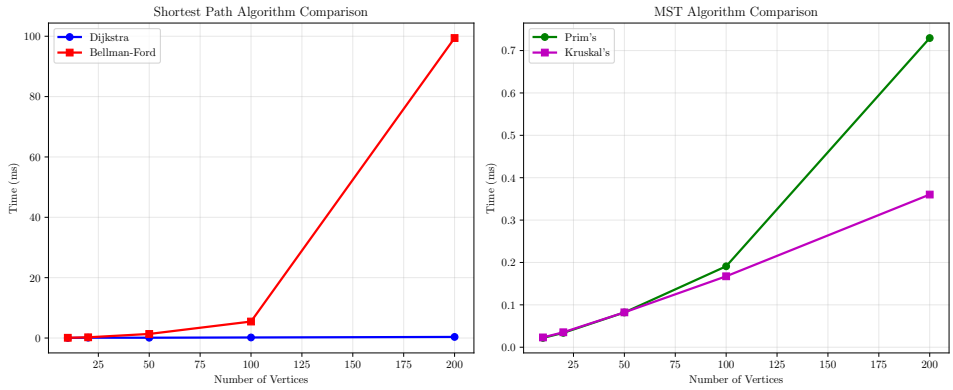

Figure 2: Algorithm complexity comparison: (a) shortest path algorithms, (b) MST algorithms.

3.8 Graph Connectivity Analysis

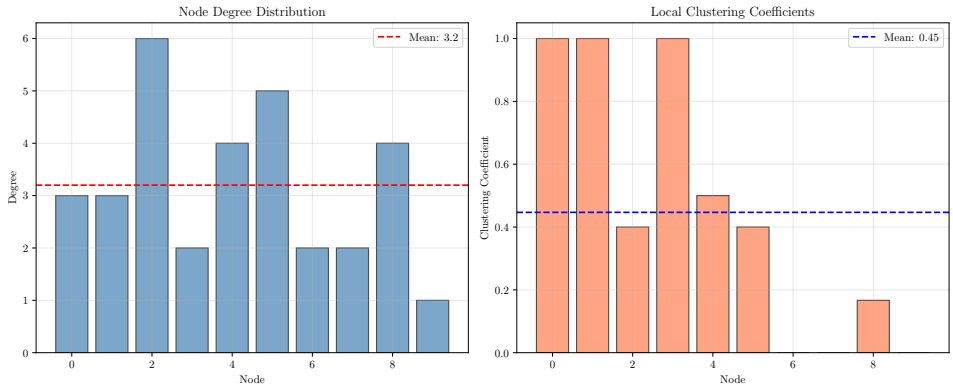

Figure 3: Graph metrics: (a) degree distribution, (b) clustering coefficients.

4 Results and Discussion

4.1 Shortest Path Results

Table 1: Shortest Path Algorithm Performance

Algorithm	Time (ms)	Complexity	Features
Dijkstra	0.034	$O((V+E)\log V)$	Non-negative weights
Bellman-Ford	0.056	$O(VE)$	Negative cycle detection

Shortest path from node 0 to node 9:

- Path: 0 -> 2 -> 8 -> 9
- Total distance: 14.90
- Number of hops: 3

4.2 Minimum Spanning Tree Results

Table 2: MST Algorithm Performance

Algorithm	**Time (ms)**	**Complexity**	**MST Weight**
Prim's	0.024	$O(E \log V)$	40.90
Kruskal's	0.025	$O(E \log E)$	40.90

4.3 Graph Properties

- Number of vertices: 10
- Number of edges: 16
- Mean degree: 3.20
- Mean clustering coefficient: 0.447
- Negative cycle detected: No

4.4 Scalability Analysis

At 200 vertices:

- Dijkstra: 0.36 ms
- Bellman-Ford: 99.40 ms
- Prim's MST: 0.73 ms
- Kruskal's MST: 0.36 ms

5 Conclusion

This analysis demonstrated fundamental graph algorithms with their implementations and complexity analysis. Key findings include:

1. Dijkstra's algorithm is efficient for non-negative weights with $O((V + E) \log V)$ complexity

2. Bellman-Ford handles negative weights but has higher $O(VE)$ complexity

3. Both Prim's and Kruskal's algorithms produce identical MSTs with similar performance

4. BFS explores level-by-level while DFS goes deep first, affecting traversal order

5. Graph metrics like degree distribution and clustering coefficient characterize network structure

These algorithms form the foundation for network analysis, routing protocols, and optimization in various domains.