Maximum Flow Algorithms: Ford-Fulkerson vs Edmonds-Karp

A Comparative Study of Classical Flow Network Algorithms

Advanced Algorithms and Data Structures Project

August 08, 2025

Contents

1. Introduction and Objectives

The Maximum Flow problem stands as one of the fundamental challenges in graph theory and network optimization. Given a flow network—a directed graph where each edge has a capacity constraint—the problem seeks to determine the maximum amount of flow that can be sent from a designated source vertex to a sink vertex while respecting the capacity limitations and flow conservation constraints.

This project presents a comprehensive analysis and implementation of two seminal algorithms that solve the Maximum Flow problem: the Ford-Fulkerson algorithm (1956) and its refined variant, the Edmonds-Karp algorithm (1972). Through theoretical analysis, empirical evaluation, and visual demonstration, we explore the fundamental differences, performance characteristics, and practical implications of these algorithmic approaches.

1.1. Project Objectives

The primary goals of this investigation are:

1. **Implementation and Verification**: Develop robust, well-documented implementations of both algorithms with comprehensive test suites ensuring correctness across diverse graph topologies.
2. **Theoretical Analysis**: Provide detailed mathematical analysis of algorithmic complexity, convergence properties, and theoretical performance bounds.
3. **Empirical Evaluation**: Conduct extensive benchmarking across various graph types, sizes, and structural characteristics to understand practical performance differences.
4. **Visualization and Education**: Create intuitive visualizations that demonstrate algorithm execution, flow assignments, and comparative behavior to enhance understanding of the underlying mechanisms.
5. **Practical Applications**: Explore real-world applications where maximum flow algorithms provide optimal solutions to network optimization problems.

1.2. Significance and Applications

Maximum flow algorithms find applications across numerous domains:

- **Network Infrastructure**: Bandwidth allocation, routing optimization, and network reliability analysis
- **Transportation Systems**: Traffic flow optimization, supply chain management, and logistics planning
- **Computer Vision**: Image segmentation, stereo matching, and object recognition
- **Operations Research**: Project scheduling, resource allocation, and capacity planning
- **Bioinformatics**: Sequence alignment, protein folding analysis, and phylogenetic reconstruction

The ubiquity of these applications underscores the fundamental importance of understanding and optimizing maximum flow algorithms.

2. Problem Definition and Mathematical Foundations

2.1. Flow Network Formal Definition

A **flow network** is formally defined as a directed graph $G = (V, E)$ equipped with the following components:

$$G = (V, E, c, s, t)$$

where:

- V is the set of vertices (nodes)
- $E \subseteq V \times V$ is the set of directed edges
- $c : E \to \mathbb{R}^+$ is the **capacity function** assigning positive real capacities to edges
- $s \in V$ is the designated **source** vertex
- $t \in V$ is the designated **sink** vertex, with $s \neq t$

2.2. Flow Function Properties

A **flow** in the network is a function $f : E \to \mathbb{R}^+$ that must satisfy two fundamental constraints:

2.2.1. Capacity Constraint

For every edge $(u, v) \in E$:

$$f(u, v) \leq c(u, v)$$

This ensures that the flow through any edge never exceeds its capacity limit.

2.2.2. Flow Conservation Constraint

For every vertex $v \in V \setminus \{s, t\}$ (all vertices except source and sink):

$$\sum_{\{u:(u,v)\in E\}} f(u, v) = \sum_{\{w:(v,w)\in E\}} f(v, w)$$

This constraint ensures that flow is neither created nor destroyed at intermediate vertices—the total flow entering a vertex must equal the total flow leaving it.

2.3. Maximum Flow Problem Statement

The **Maximum Flow Problem** seeks to find a flow f that maximizes the **value of the flow**, defined as:

$$|f| = \sum_{\{v:(s,v)\in E\}} f(s, v) - \sum_{\{u:(u,s)\in E\}} f(u, s)$$

Equivalently, by flow conservation, this equals:

$$|f| = \sum_{\{u:(u,t)\in E\}} f(u, t) - \sum_{\{v:(t,v)\in E\}} f(t, v)$$

2.4. Residual Network and Augmenting Paths

Central to understanding maximum flow algorithms is the concept of the **residual network**.

2.4.1. Residual Capacity

For any edge (u, v), the **residual capacity** is defined as:

$$c_{f(u,v)} = \begin{cases} c(u,v) - f(u,v) & \text{if } \ (u,v) \in E \\ f(v,u) & \text{if } \ (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

The first case represents remaining forward capacity, while the second represents the possibility of reducing existing flow (creating backward capacity).

2.4.2. Residual Network

The **residual network** $G_f = \left(V, E_f\right)$ with respect to flow f contains only edges with positive residual capacity:

$$E_f = \left\{(u,v) \in V \times V : c_{f(u,v)} > 0\right\}$$

2.4.3. Augmenting Path

An **augmenting path** is a simple path from s to t in the residual network G_f. The **residual capacity** of such a path P is:

$$c_{f(P)} = \min_{\{(u,v) \in P\}} c_{f(u,v)}$$

2.5. Max-Flow Min-Cut Theorem

The theoretical foundation for maximum flow algorithms rests on the celebrated Max-Flow Min-Cut Theorem:

Theorem (Max-Flow Min-Cut): In any flow network, the value of the maximum flow equals the capacity of the minimum cut.

Formally, if f^* is a maximum flow and (S, T) is a minimum cut where $S \cup T = V$, $S \cap T = \emptyset$, $s \in S$, and $t \in T$, then:

$$|f^*| = c(S,T) = \sum_{\{u \in S, v \in T, (u,v) \in E\}} c(u,v)$$

This theorem provides both an optimality condition and a certificate for maximum flow solutions.

3. Algorithm Analysis and Implementation

3.1. Ford-Fulkerson Algorithm

The Ford-Fulkerson method, introduced by L.R. Ford Jr. and D.R. Fulkerson in 1956, establishes the foundational approach for solving maximum flow problems through the iterative augmentation of flow along source-to-sink paths.

3.1.1. Algorithmic Framework

The Ford-Fulkerson method follows a generic framework that can be instantiated with different path-finding strategies:

function FORD-FULKERSON(G, s, t)
    $f(u,v) \leftarrow 0$ for all $(u,v) \in E$
    while there exists an augmenting path `$P$` in `$G_f$` **do**
        $c_{f(P)} \leftarrow$ minimum residual capacity along P

```
        for edges $(u,v)$ in $P$ do
            if $(u,v) ∈ E$ then
                f(u, v) ← f(u, v) + c_{f(P)}
            else
                f(v, u) ← f(v, u) − c_{f(P)}
    return f
```

3.1.2. Path Selection Strategy

The generic Ford-Fulkerson framework does not specify how augmenting paths should be discovered. In our implementation, we employ **depth-first search** (DFS) to locate augmenting paths, which provides a straightforward recursive approach:

```
function DFS-Find-Path(u, t, visited, path)
    if $u = t$ then
        return path ∪ {t}
    visited ← "visited" ∪ {u}
    for $v$ such that $c_f(u,v) > 0$ and $v ∉$ visited do
        "result" ← "DFS-Find-Path(v, t, visited, path ∪ {u})"
        if result $≠$ null then
            return "result"
    return "null"
```

3.1.3. Complexity Analysis

The time complexity of Ford-Fulkerson depends critically on the path selection method and the nature of the input:

Time Complexity: $O(E \cdot |f^*|)$ where $|f^*|$ is the value of the maximum flow.

This bound arises because:
- Each augmenting path increases the flow by at least 1 unit (assuming integer capacities)
- Finding each path requires $O(E)$ time using DFS
- At most $|f^*|$ iterations are needed

Space Complexity: $O(V + E)$ for storing the residual graph and maintaining the DFS recursion stack.

3.1.4. Limitations

The Ford-Fulkerson algorithm exhibits several limitations:

1. **Non-polynomial Runtime:** With irrational capacities, the algorithm may not terminate
2. **Inefficient Path Selection:** DFS may select long paths when shorter alternatives exist
3. **Poor Performance on Dense Graphs:** The $O(E \cdot |f^*|)$ bound becomes problematic when $|f^*|$ is large

3.2. Edmonds-Karp Algorithm

Jack Edmonds and Richard Karp addressed the efficiency limitations of Ford-Fulkerson in 1972 by proposing a specific path selection strategy that guarantees polynomial-time performance.

3.2.1. Key Innovation: Breadth-First Search

The Edmonds-Karp algorithm modifies Ford-Fulkerson by using **breadth-first search** (BFS) to find the **shortest** augmenting paths (in terms of number of edges):

```
function BFS-FIND-PATH(s, t)
    "queue" ← "Queue(s)"
    "parent" ← "empty map"
    "visited" ← {s}
    while queue is not empty do
        "u" ← "queue.dequeue()"
        for $v$ such that $c_f(u,v) > 0$ and $v ∉$ visited do
            "queue.enqueue(v)"
            "visited" ← "visited" ∪ {v}
            "parent[v]" ← u
            if $v = t$ then
                return "reconstruct path using parent map"
    return "null"
```

3.2.2. Theoretical Advantages

The BFS-based path selection provides several theoretical guarantees:

Theorem: The Edmonds-Karp algorithm runs in $O(VE^2)$ time.

Proof Sketch:
1. Each BFS operation requires $O(E)$ time
2. The distance from s to t (in terms of edges) can increase at most V times
3. Between distance increases, at most E edges can become saturated
4. Therefore, at most $O(VE)$ iterations are needed
5. Total complexity: $O(VE) \cdot O(E) = O(VE^2)$

Key Insight: By always choosing shortest paths, the algorithm ensures that the distance to the sink in the residual network is non-decreasing, leading to the polynomial bound.

3.2.3. Implementation Details

Our Edmonds-Karp implementation incorporates several optimizations:

1. **Efficient Residual Graph Representation:** Using adjacency lists with both forward and backward edges
2. **Path Reconstruction:** Parent pointer technique for efficient path recovery
3. **Early Termination:** BFS terminates immediately upon reaching the sink

3.2.4. Comparative Analysis

Aspect	Ford-Fulkerson	Edmonds-Karp
Path Strategy	Any augmenting path (DFS)	Shortest augmenting path (BFS)
Time Complexity	$O(Ec \cdot \lvert f^*\rvert)$	$O(VE^2)$
Termination	Guaranteed only for rational capacities	Always guaranteed
Practical Performance	Variable, can be poor	Consistently polynomial

3.3. Algorithm Correctness

Both algorithms rely on the fundamental correctness of the augmenting path method:

Theorem (Augmenting Path Correctness): A flow f is maximum if and only if there are no augmenting paths in the residual network G_f.

Proof:

- **Necessity:** If an augmenting path exists, the flow can be increased, contradicting maximality
- **Sufficiency:** If no augmenting path exists, then by the Max-Flow Min-Cut theorem, the current flow is maximum

4. Results and Analysis

...

5. Conclusions and Future Work

...