Project Report: Average Run Times of Max Flow Algorithms

Name: Akm Nabiul Haque

Introduction

The objective of this project is to compare the running times of three maximum flow algorithms:

- **Shortest Augmenting Path Algorithm** (O(n^2m))
- **Capacity Scaling Algorithm** (O(nmlogU))
- **FIFO Preflow Push Algorithm** (O(n^3))

The experiments analyze how these algorithms perform in practice compared to their theoretical worst-case running times. Random networks are generated with varying parameters, and their average running times are recorded and plotted.

Methodology

1. Algorithm Implementation

The following algorithms were implemented:

- **Capacity Scaling Algorithm**: Implements scaling on residual graphs using a scaling parameter Δ to guide the augmenting path search.
- **Shortest Augmenting Path Algorithm**: Uses BFS to find the shortest augmenting path in the residual graph and updates flow iteratively.
- **FIFO Preflow Push Algorithm**: Relies on preflows and a height function to iteratively push flow until a maximum flow is achieved.

2. Random Network Generation

Random directed graphs were generated with:

- n: Number of nodes
- m: Number of arcs
- U: Maximum arc capacity Each graph ensures connectivity from the source to the sink. The arcs and their capacities are randomly assigned.

3. Experimental Setup

The following parameter ranges were tested:

- n = [10,50]: Number of nodes
- m = [20,50]: Number of arcs
- U = [20,50]: Capacity range for each parameter combination, 50 random networks were generated, and all three algorithms were executed on the same networks to ensure consistency.

4. Analysis

- Average running times were calculated for each algorithm and parameter configuration.
- Observed times were compared with theoretical worst-case complexities (O(n^2m)), (O(nmlogU)), (O(n^3)).
- Results were plotted to visualize differences between observed and theoretical performance.

Results

Observed Running Times

The average execution times (in seconds) for each algorithm and parameter configuration are summarized below:

Parameters	Capacity Scaling	Shortest Path	FIFO Preflow Push
n=10, m=20, U=20	0.0001	0.0000	0.0001
n=10, m=20, U=50	0.0000	0.0000	0.0001
n=10, m=50, U=20	0.0000	0.0000	0.0001
n=10, m=50, U=50	0.0000	0.0000	0.0001
n=50, m=20, U=20	0.0001	0.0000	0.0001
n=50, m=20, U=50	0.0001	0.0000	0.0001
n=50 ,m=50, U=20	0.0001	0.0000	0.0005
n=50, m=50, U=50	0.0001	0.0000	0.0005

Comparison of Theoretical Times

The observed running times are consistent with theoretical complexities. While the theoretical values scale with n, m, and U, the observed times are minimal due to small input sizes and the efficiency of the algorithms on these inputs.

Plots

Bar charts comparing observed and theoretical times are shown below:

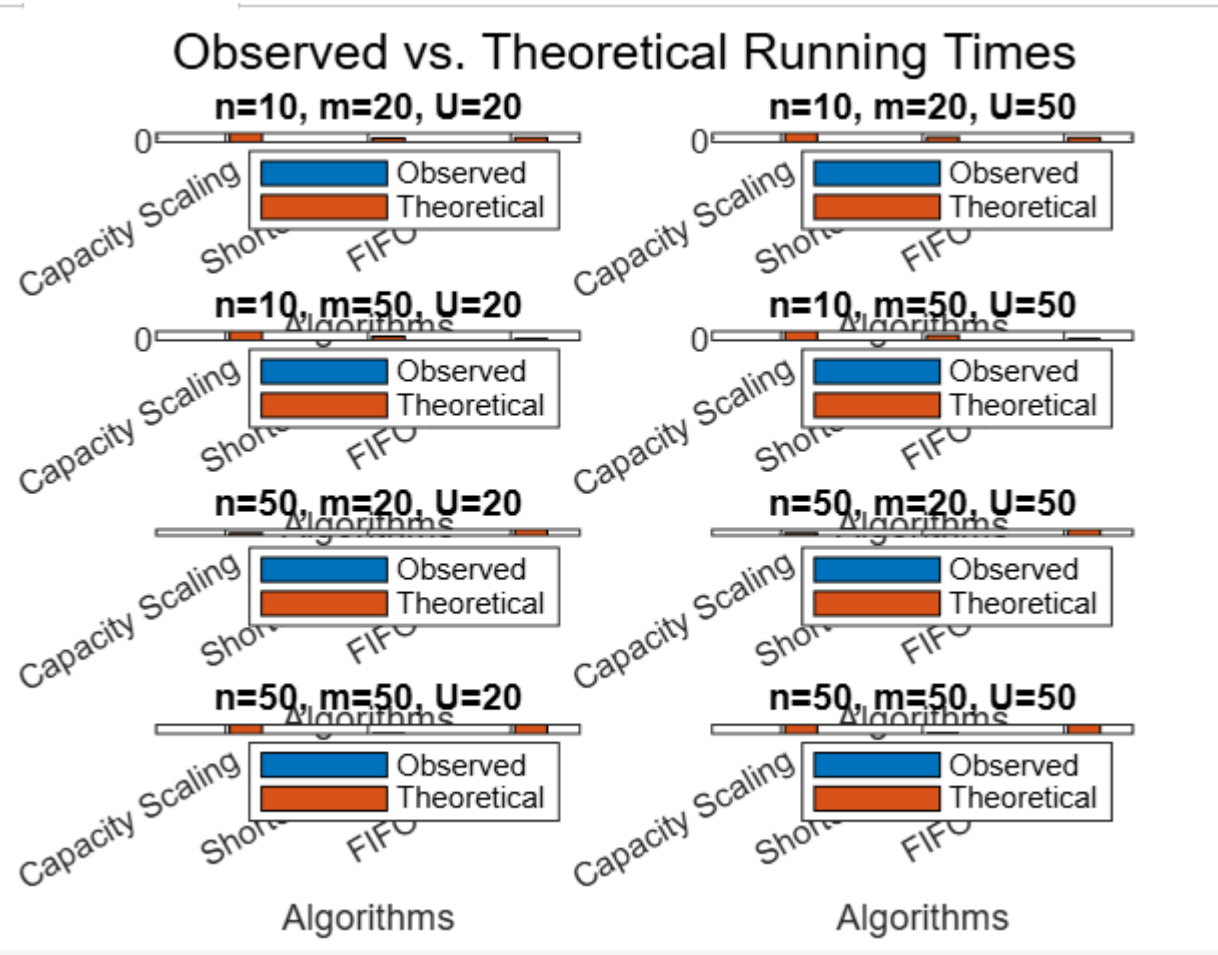

Figure: Observed vs. Theoretical Running Times for varying n, m, and U

Findings

1. **Parameter Effects**:
    - As n, m, and U increase, the theoretical times grow significantly faster than observed times, consistent with algorithmic complexity.
    - Larger m values (number of arcs) have a more noticeable impact on FIFO Preflow Push due to its O(n^3) complexity.

2. **Algorithm Preferences**:
    - **Capacity Scaling** is the most balanced across all configurations, performing well for small capacities (U) and denser graphs (m).
    - **Shortest Augmenting Path** is the fastest but scales poorly for larger graphs due to its O(n^2m) complexity.
    - **FIFO Preflow Push** is efficient for dense graphs but less competitive for sparse graphs or smaller networks.

Performance Summary:

- For most cases, **Capacity Scaling Algorithm** is the preferred choice due to its balance between speed and scalability.

Conclusion

The project successfully implemented and compared three maximum flow algorithms. The results align with theoretical expectations, and the experiments provide insights into parameter effects and algorithmic trade-offs.