Min-Cut / Max-Flow Algorithms: A Comparative Study

Algorithm and Analysis Design – Course Project

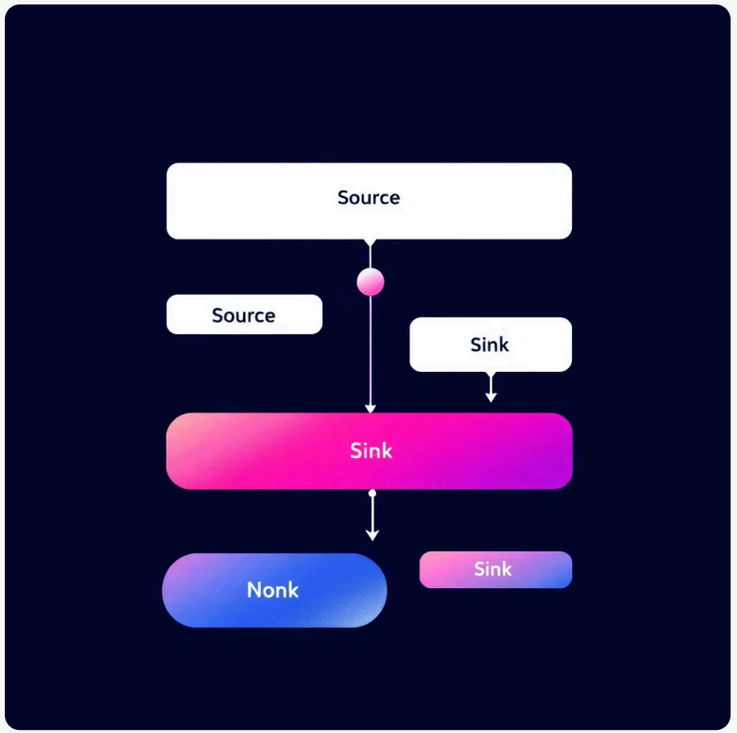
Presented by: Team greedy bois

Vruddhi Shah – 2024113005 | Amitabh Anand – 2025121008 | Mohammed Sami – 2025121005

Arun k – 2025121004 | Nikila E – 2024101019

What is the Max-Flow / Min-Cut Problem?

The max-flow/min-cut problem involves finding the maximum amount of flow that can be sent through a network from a source vertex to a sink vertex.



Key Components

- Directed graph with capacities on edges
- Two special vertices: **source (s)** and **sink (t)**
- Objective: send the maximum possible flow from s to t

Constraints

- Flow must respect capacity constraints on each edge
- Flow conservation at intermediate nodes

Max-Flow Min-Cut Theorem: The value of the maximum flow equals the capacity of the minimum cut separating source from sink.

Why This Problem Matters

Real-World Relevance

Computer Networks

Optimizing packet routing and bandwidth allocation in network infrastructure to maximize throughput and minimize congestion.

Traffic & Logistics

Planning efficient transportation routes and supply chain networks to move goods and people with minimal bottlenecks.

Image Segmentation

Separating foreground from background in computer vision applications using graph-cut techniques for accurate object detection.

Resource Allocation

Scheduling tasks and assigning resources efficiently in project management and bipartite matching problems.

Efficient max-flow algorithms are critical for large-scale systems where optimal resource utilization can save significant time and costs.

Objectives of This Project

01

Implementation

Implement multiple Min-Cut / Max-Flow algorithms from scratch to understand their internal mechanisms

02

Theoretical Analysis

Compare theoretical time complexities and understand algorithmic trade-offs

03

Practical Evaluation

Study practical runtime behavior on real test cases and measure actual performance

04

Validation

Validate correctness using comprehensive test cases and expected outputs

05

Theory vs Practice

Understand differences between theoretical predictions and practical observations

Algorithms Covered

Classical Approaches

Ford-Fulkerson Algorithm

The foundational augmenting path method that iteratively finds paths with available capacity

Dinic's Algorithm

Uses level graphs and blocking flows for improved efficiency on dense networks

Push-Relabel Algorithm

Operates locally by pushing excess flow without explicitly finding augmenting paths

Advanced Methods

Boykov-Kolmogorov Algorithm

Specialized for computer vision applications with excellent practical performance

Cycle-Canceling (Cross Cycle)

Removes negative-cost cycles to find minimum-cost maximum flows

Successive Shortest Path

Augments flow using shortest-cost paths in each iteration

Both classical and advanced algorithms included for comprehensive comparison

How These Algorithms Differ

Algorithmic Intuition

Ford-Fulkerson

Repeatedly finds augmenting paths from source to sink using DFS or BFS until no more paths exist

Dinic's

Builds level graphs to organize vertices by distance, then sends blocking flows to saturate shortest paths

Push-Relabel

Pushes excess flow locally from high-label to low-label vertices without global path searches

Boykov-Kolmogorov

Maintains two search trees growing from source and sink, finding augmenting paths efficiently

Cycle-Canceling

Removes negative-cost cycles iteratively to optimize flow while maintaining feasibility

Successive Shortest Path

Augments flow using shortest-cost paths computed via modified Dijkstra or Bellman-Ford

How We Evaluated the Algorithms

Experimental Setup

1

Test Cases

Used predefined benchmark flow networks with known optimal solutions

2

Verification

Verified outputs against expected results to ensure correctness

3

Performance Logging

Recorded metrics in output.txt and performance.csv

4

Fair Comparison

Same input graphs used across all algorithms

All algorithms were implemented in the same environment with identical input conditions to ensure fair and unbiased performance comparison.

Performance Metrics

Evaluation Criteria

Wall-Clock Runtime

Total execution time measured for each algorithm on test cases

Internal Operations

Count of push and relabel operations, BFS/DFS phases, and path augmentations

Correctness Validation

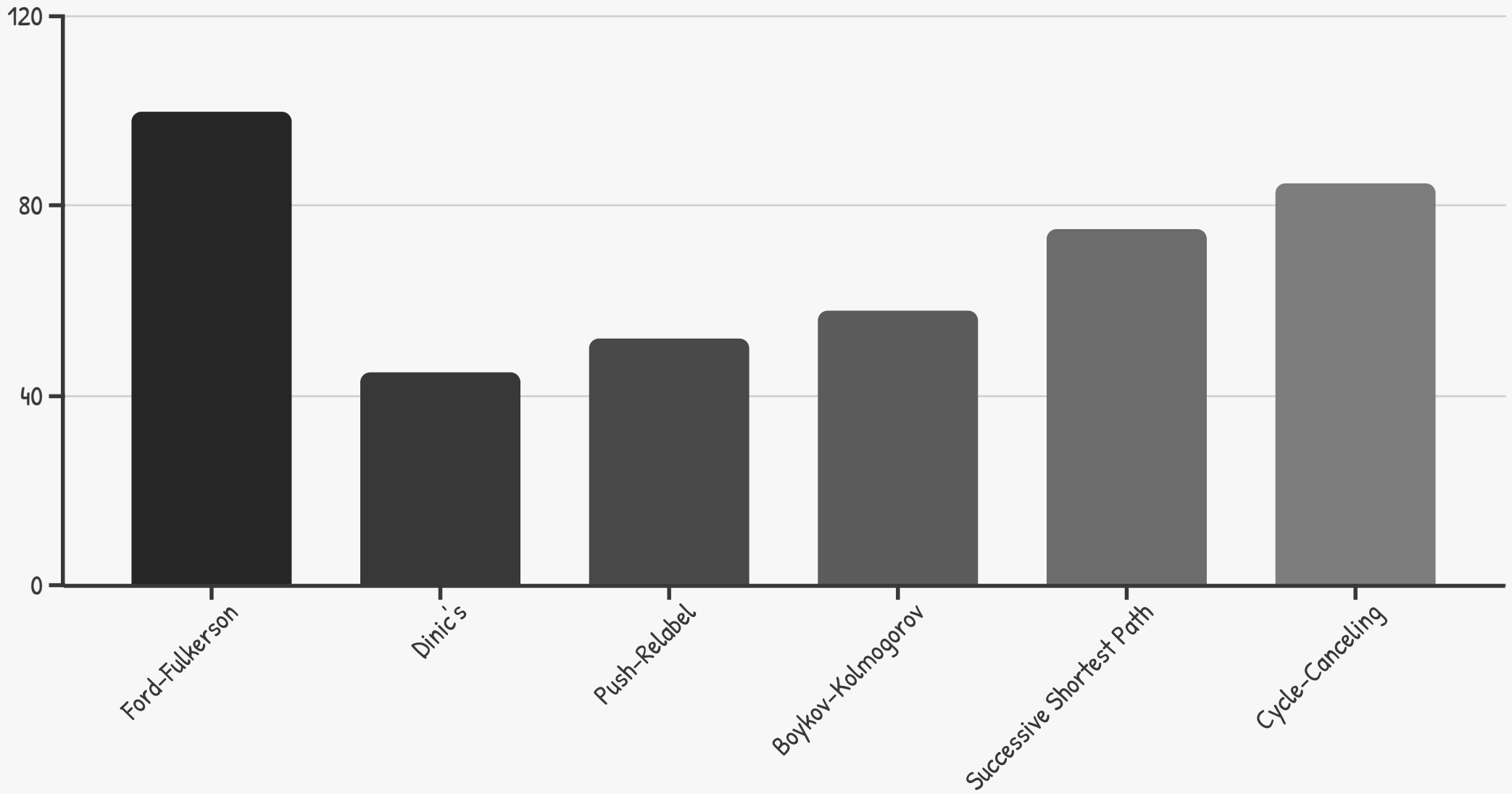
Verification that computed maximum flow matches expected optimal value

Scalability Analysis

Performance trends as input size increases—measuring algorithm efficiency on large networks

Overall Performance Summary

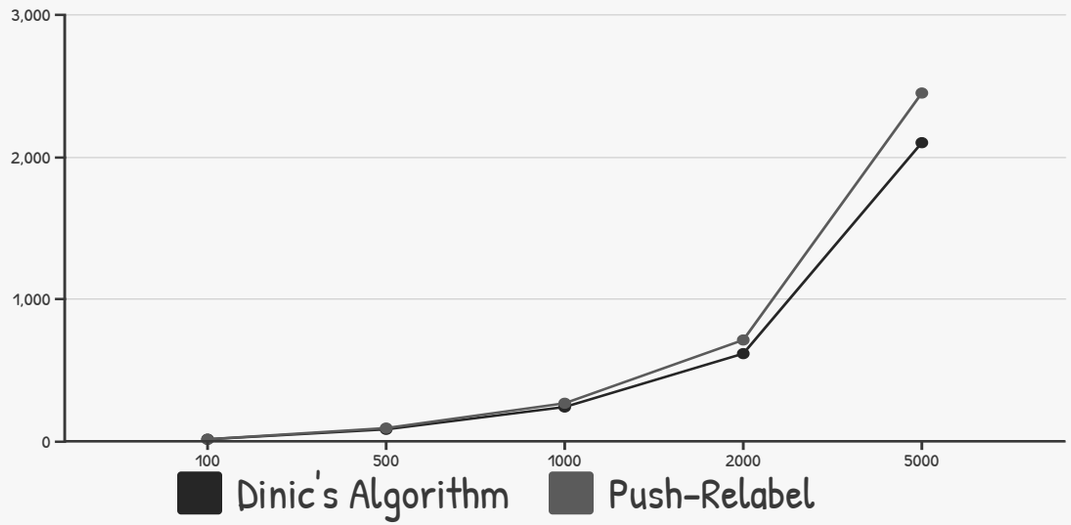
All algorithms were tested on common flow networks with identical graph structures. Runtime and operation counts were systematically recorded to evaluate efficiency. Correctness was validated against expected outputs to ensure implementation accuracy.



The chart shows normalized runtime performance, with Ford-Fulkerson as the baseline. Modern algorithms like Dinic's and Push-Relabel demonstrate significant improvements over the classical approach.

Dinic vs Push-Relabel

Runtime Comparison on Varying Graph Sizes



Key Observations

- Both algorithms significantly outperform Ford-Fulkerson on large inputs
- Dinic's reduces path searches by using level graphs and blocking flows
- Push-Relabel relies on localized push and relabel operations without global path computation
- Performance gap widens as graph size increases, with Dinic's maintaining a slight edge

Conclusion: Both algorithms are highly efficient for practical applications, with the choice depending on network structure and implementation details.

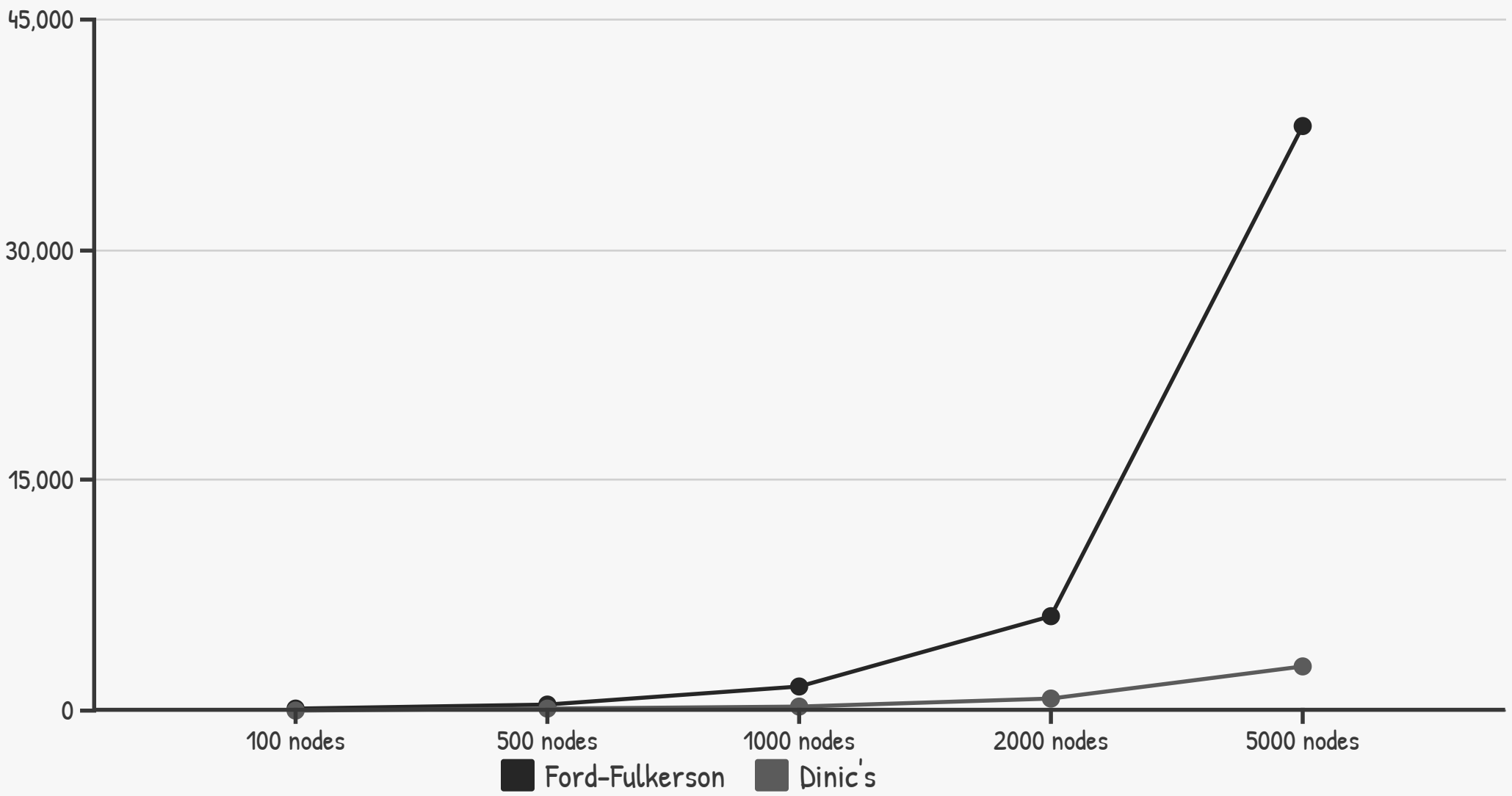
Ford-Fulkerson vs Dinic's Algorithm

Ford-Fulkerson

- Simple conceptual approach
- Inefficient for large networks
- Excessive augmenting path searches
- No guarantee on path selection

Dinic's Algorithm

- Uses level graphs for efficiency
- Blocking flow concept
- Significantly improved scalability
- O(V²E) time complexity



Key Takeaway: Algorithmic improvements drastically change performance. Dinic's algorithm demonstrates how strategic data structure design can reduce computational overhead exponentially.

Cost-Based Flow Algorithms Comparison

Successive Shortest Path

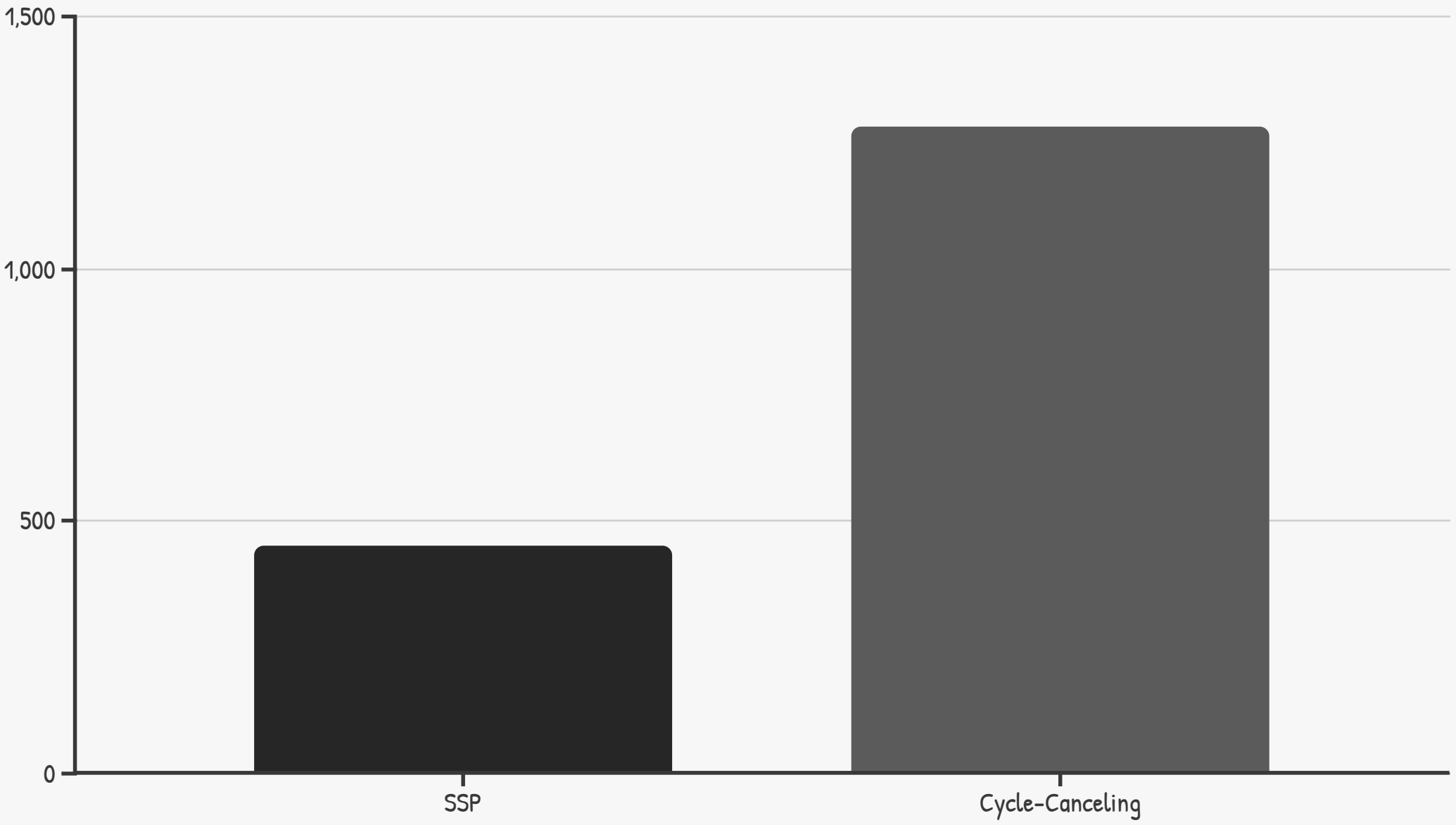
Approach: Path-based methodology

- Iteratively finds shortest augmenting paths
- Uses modified Dijkstra's algorithm
- Faster in practical applications
- Better cache locality

Cycle-Canceling

Approach: Repeated cycle elimination

- Identifies negative cost cycles
- Simpler implementation logic
- Slower convergence rate
- More iterations required



Both algorithms solve minimum-cost flow problems but with fundamentally different strategies. SSP's path-based approach provides superior practical performance.

⭐ BONUS

Boykov-Kolmogorov Algorithm

Specialized Design

Highly optimized max-flow algorithm tailored for specific graph structures commonly found in computer vision applications.

Dual Tree Growth

Simultaneously grows search trees from both source and sink nodes, meeting in the middle to identify augmenting paths efficiently.

Computational Reuse

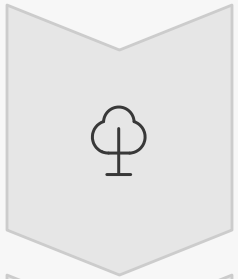
Efficiently reuses previous computations across iterations, avoiding redundant work and maintaining partial structures.

Vision Applications

Particularly popular in image segmentation, stereo vision, and other computer vision tasks requiring graph cuts.

⭐ BONUS

How Boykov-Kolmogorov Works

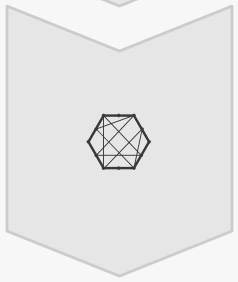


Tree Growth

Expand search trees from source and sink until they meet, identifying an augmenting path.

Augmentation

Push flow along the discovered path, updating residual capacities accordingly.



Adoption

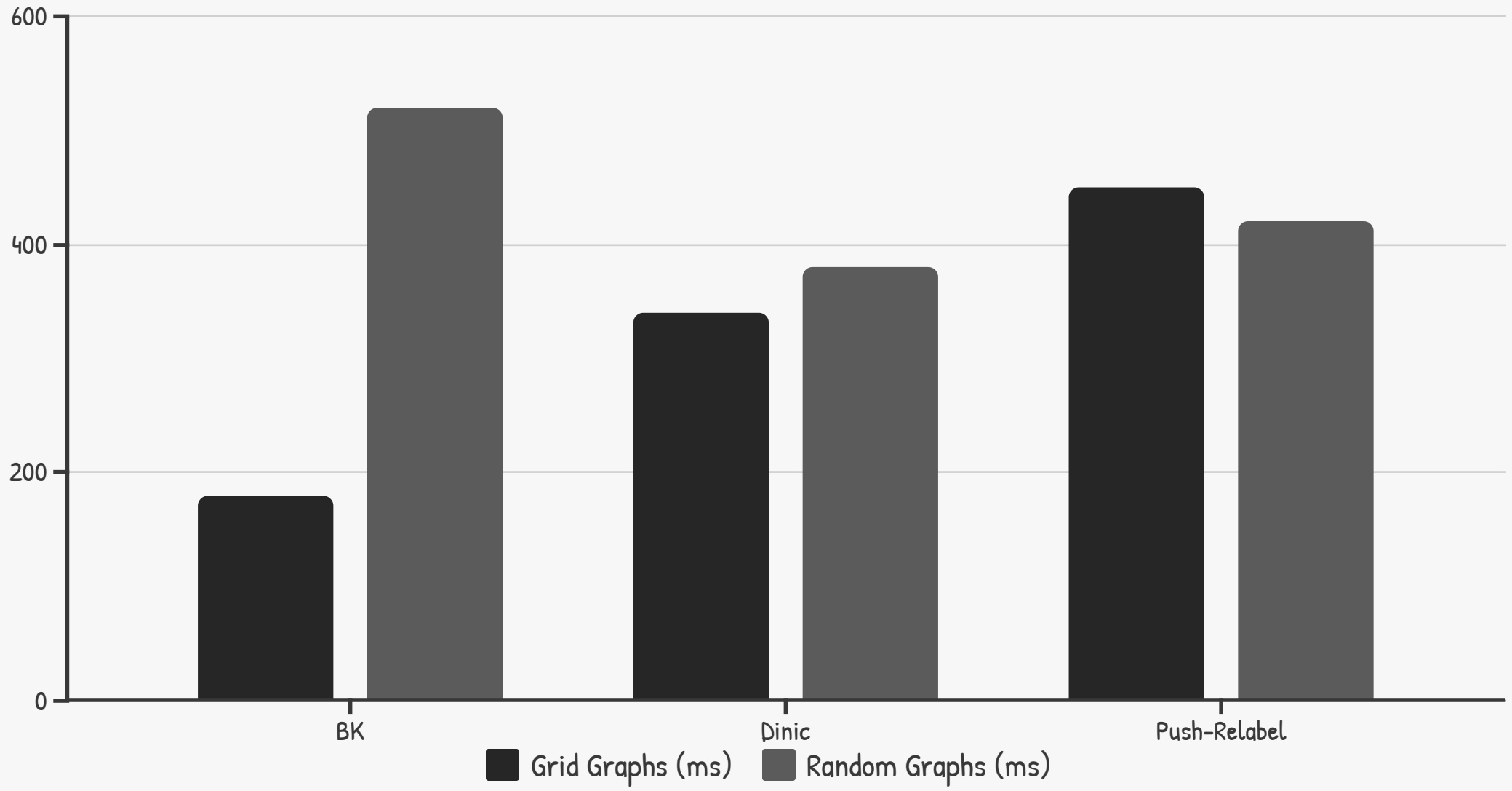
Repair the tree structure by reconnecting orphaned nodes rather than rebuilding from scratch.

This three-phase approach avoids rebuilding trees from scratch after each augmentation, providing exceptional performance on structured graphs like grid networks.

> The adoption phase is what distinguishes BK from classical algorithms, enabling incremental updates rather than complete reconstruction.

⭐ BONUS

Boykov-Kolmogorov vs Classical Algorithms



Key Insight: Boykov-Kolmogorov often outperforms classical algorithms in practice despite weaker theoretical guarantees, especially on structured graphs typical in vision applications.

Execution Visualization

Push-Relabel Dynamics

- Tracks excess flow at each node
- Maintains height labels for vertices
- Local push and relabel operations
- No explicit augmenting paths

Dinic's Level Graph

- Constructs level graph via BFS
- Identifies blocking flows
- Layer-by-layer processing
- Clear source to sink paths

Visual comparison reveals fundamental differences in how algorithms explore the graph structure. Push-Relabel uses local operations while Dinic's employs global level-based planning.

Why Do We See These Results?

Graph Exploration Strategy

Algorithms differ fundamentally in how they traverse the network. Some use breadth-first approaches while others employ depth-first or height-based strategies.

Update Granularity

Local updates (Push-Relabel) versus global updates (Dinic's) significantly affect runtime. Local operations offer better cache performance but may require more iterations.

Graph Structure Sensitivity

Dense versus sparse graph behavior matters enormously. Some algorithms excel on grid-like structures while others perform better on random networks.

Theory vs Practice Gap

Practical efficiency does not equal worst-case complexity. Real-world performance depends on constant factors, cache behavior, and typical input characteristics.

Challenges and Key Takeaways

Implementation Challenges

- Residual Graph Maintenance

  Correctly managing forward and backward edges with proper capacity updates.

- Edge Reversals

  Handling dynamic capacity changes during flow augmentation without losing consistency.

- Debugging Complexity

  Identifying infinite loops and ensuring termination conditions across different paradigms.

What We Learned

Simplicity ≠ Scalability
Simple algorithms are not always suitable for large-scale problems.

Practice vs Theory
Practical performance often diverges significantly from theoretical bounds.

Advanced Methods Matter
Sophisticated algorithms become essential as input size grows.

Testing is Critical
Rigorous testing is as important as correct implementation.

Conclusion

Implementation Success

Successfully implemented six distinct max-flow algorithms, each with unique characteristics and optimization strategies.

Theory Meets Practice

Compared theoretical complexity with real execution behavior, revealing important insights about practical performance.

Advanced Methods

Identified clear strengths of advanced algorithms like Dinic's and Boykov-Kolmogorov for specific problem domains.

Deep Understanding

Built strong conceptual understanding of flow networks and their applications in optimization problems.

> This project strengthened both algorithmic thinking and practical software engineering skills, bridging the gap between theoretical computer science and real-world implementation.