A COMPREHENSIVE ANALYSIS AND ENHANCEMENT OF NETWORK FLOW ALGORITHMS.

A FLOW NETWORK IS A DIRECTED GRAPH WHERE EACH EDGE HAS A CAPACITY AND RECEIVES A FLOW. THE FLOW ON AN EDGE CANNOT EXCEED ITS CAPACITY, AND FOR ALL NON-TERMINAL NODES, THE INCOMING FLOW MUST EQUAL THE OUTGOING FLOW. THE STUDY OF NETWORK FLOW ALGORITHMS HAS EVOLVED SIGNIFICANTLY SINCE THE 1950S, WITH VARIOUS APPROACHES DEVELOPED TO ADDRESS INCREASINGLY COMPLEX SCENARIOS.

DESCRIPTION

Formal Definition
A network N consists of:

- A directed graph G(V, E)
- A source node s ∈ V with only outgoing edges
- A sink node t ∈ V with only incoming edges
- A positive capacity function c: E → R+

A flow f on network N is a function f: E → R+ that satisfies:

1. Edge capacity constraint: ∀e ∈ E, 0 ⩽ f(e) ⩽ c(e)
2. Flow conservation: ∀v ∈ V{s,t}, Σ(flow entering v) = Σ(flow leaving v)

The value of a flow f is defined as v(f) = Σ(flow leaving s) = Σ(flow entering t)

OBJECTIVE

The maximum flow problem aims to find a feasible flow with the maximum possible value from source s to sink. This represents the maximum amount of resources transported through the network under given capacity constraints.

STPES INVOLVED IN SOLVING

The general approach to solving network flow problems involves:

1. Model Construction: Represent the problem as a flow network with source, sink, nodes, and capacities.
2. Algorithm Selection: Choose an appropriate algorithm based on network characteristics and problem requirements.
3. Initialization: Set initial flow values (typically zero) for all edges.
4. Iteration: Apply the chosen algorithm's specific approach:
   - For augmenting path methods: Find paths and increase flow
   - For preflow methods: Push excess flow and relabel vertices
5. Termination: Stop when the algorithm's termination condition is met (e.g., no more augmenting paths).
6. Solution Extraction: Extract the maximum flow value and the flow assignment for each edge.

ALGORITHM ANALYSIS

Ford-Fulkerson Algorithm: Fulkerson computes the maximum flow in a flow network there exists a path from the source (start node) to the sink (end node) with available capacity on all edges in the path, we send flow along that path. Then we find another path and continue this process until no more augmenting paths can be found.
Edmonds-Karp algorithm: is a specific implementation of the Ford-Fulkerson method for solving the maximum flow problem in flow networks.this algorithm improves upon the general Ford-Fulkerson approach by specifying that breadth-first search (BFS) should be used to find augmenting paths.

MATHEMATICAL FORMULATION OF T(N)

For Ford-Fulkerson: $T(n) = O(max\_flow * E)$
For Edmonds-Karp: $T(n) = O(V * E^2)$

PROPOSED ALGORITHM

The "almost-linear-time" network flow algorithm. This algorithm achieves what was long considered theoretically impossible: computing maximum flows in time nearly proportional to the network's size.
The algorithm's innovation lies in its energy optimization approach, which defines a function balancing capacity constraints and flow distribution. It strategically identifies energy-reducing cycles while using low-stretch spanning trees to dramatically narrow the search space. By merging continuous optimization with combinatorial methods, it achieves a time complexity of approximately $O(m \cdot polylog(n))$, where m represents edges and n represents vertices.

Algorithm	Time Complexity	Notes
Ford-Fulkerson	$O(E \cdot f)$	Where E is the number of edges and f is the maximum flow value. Works with integer capacities but may not terminate with irrational capacities [5] [9].
Edmonds-Karp	$O(V \cdot E^2)$	A specific implementation of Ford-Fulkerson that uses BFS to find shortest augmenting paths. V is the number of vertices [2] [6].
Almost-Linear-Time Algorithm (Kyng et al.)	$O(m \cdot polylog(n))$	Where m is the number of edges and n is the number of vertices. Essentially optimal as it approaches linear time [3] [4].

MASTER'S THEOREM

Applying a modified version of the Master Theorem analysis:

1. The reduction factor is not constant (unlike traditional divide-and-conquer where problems are divided into fixed proportions)
2. The work at each level is approximately O(m)
3. The depth of recursion is approximately O(log(m))

Multiplying the work per level by the number of levels:
$O(m) \times O(\log(m)) = O(m \cdot \log(m))$

CONCLUSION & FUTURE WORK

Network flow algorithms have evolved significantly since their inception, substantially improving efficiency. The recent development of nearly linear time algorithms represents a theoretical breakthrough, though practical implementations may still require refinement.

1. Developing more efficient algorithms for large-scale and dynamic network flow problems.
2. Applying machine learning methods to predict and optimize network flow distribution.
3. Building integrated network models that can simultaneously handle multiple types of flows (data, material, financial, etc.)
4. Exploring applications in emerging fields such as the Internet of Things and smart cities

RESULT

The performance of network flow algorithms varies significantly based on network characteristics:

- For small to medium networks, Edmonds-Karp and Dinic's algorithms provide practical efficiency.
- For large networks, Push-Relabel algorithms often perform better.
- The new nearly-linear time algorithm shows promise for extremely large networks, though it may not be practical for smaller instances

PRESENTED BY:
RA2311030010104- TAPPATLA SAIROHITH
RA2311030010117- SAHASRA PERAM

UNDER GUIDANCE OF:
Dr.A.PRABHU CHAKRAVARTHY