Push-Relabel Algorithm

November 28, 2025

1 The Push-Relabel Algorithm

1.1 Introduction

Unlike augmenting-path algorithms (such as Ford-Fulkerson, Edmonds-Karp, and Capacity Scaling) or blocking-flow algorithms (such as Dinic's), Push-Relabel is a preflow-based algorithm. By relaxing the flow conservation constraint during execution, it maintains a **preflow** rather than a valid flow. This localized approach allows for greater parallelism, as operations on nodes occur independently without global synchronization. Furthermore, it offers superior theoretical time complexity; specifically, the FIFO vertex selection variant achieves $O(V^3)$, outperforming $O(VE^2)$ on dense graphs.

1.2 Theoretical Formulation

1.2.1 Preflow and Excess

A preflow is a function $f : V \times V \to \mathbb{R}$ that satisfies two fundamental properties:

$$\begin{aligned} &1. \quad f(u,v) \leq c(u,v) \qquad \forall (u,v) \in V \times V \quad \text{(Capacity Constraint)} \\ &2. \quad f(u,v) = -f(v,u) \qquad \forall (u,v) \in V \times V \quad \text{(Skew Symmetry)} \end{aligned} \tag{1}$$

Unlike a standard flow, a preflow relaxes the conservation constraint. For any vertex $u \in V \setminus \{s\}$, the total flow entering u is allowed to exceed the flow leaving it. This difference is defined as the **excess** flow, $e(u)$:

$$e(u) = \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v) \geq 0 \tag{2}$$

A vertex u is considered **active** if $e(u) > 0$ and $u \notin \{s,t\}$.

In simple words
Imagine the network as a system of water pipes (edges) and junctions (vertices).

- **Preflow:** A state where water flows into a junction faster than it flows out, creating a temporary buildup. (In a standard flow, inflow must exactly equal outflow).
- **Excess:** The net amount of water currently accumulated at a junction, waiting to be pushed into the next pipe.

1.2.2 Height Function

To direct the flow from the source toward the sink, the algorithm maintains a height function $h : V \to \mathbb{N}$. A height function is valid if, for all residual edges $(u,v) \in E_f$:

$$h(u) \leq h(v) + 1 \tag{3}$$

Intuitively, flow can only be pushed "downhill" from node u to neighbor v if $h(u) = h(v) + 1$. The source is fixed at $h(s) = |V|$ and the sink at $h(t) = 0$.

1.3 Algorithmic Operations

The algorithm proceeds by iteratively performing two basic operations (from which it's name is also derived!) on active nodes (that is, nodes with a non-zero excess flow):

1. **Push(u, v):** Applies when $e(u) > 0$, residual capacity $c_f(u, v) > 0$, and $h(u) = h(v) + 1$.

$$\Delta = \min(e(u), c_f(u, v))$$

   We move Δ units of flow from u to v. If the edge becomes saturated, it is a *saturating push*; otherwise, it is *non-saturating*.

2. **Relabel(u):** Applies when $e(u) > 0$ but no admissible edges exist (i.e., for all neighbors v with residual capacity, $h(u) \leq h(v)$). We increase the height of u:

$$h(u) = 1 + \min\{h(v) \mid (u, v) \in E_f\} \tag{4}$$

1.4 Implementation: FIFO Vertex Selection

To optimize performance and ensure polynomial time termination, we implemented the **FIFO Vertex Selection** rule. Active nodes are maintained in a First-In-First-Out queue. When a node u is removed from the queue, it is "discharged" (Push/Relabel operations are applied repeatedly) until it no longer has excess flow or its height increases.

Algorithm 1 Push-Relabel (FIFO Implementation)

```
Initialize: h(s) ← |V|, h(v) ← 0 for others.
Saturate all edges (s, v) leaving source. Update e(v) and e(s).
Enqueue all active nodes into Queue Q.
while Q is not empty do
    u ← Q.dequeue()
    while e(u) > 0 do
        if ∃v such that c_f(u, v) > 0 and h(u) == h(v) + 1 then
            Push min(e(u), c_f(u, v)) from u to v
            if v ≠ s, t and v is not in Q then
                Q.enqueue(v)
            end if
        else
            Relabel u: h(u) ← 1 + min{h(v)}
        end if
    end while
end while
```

1.5 Complexity Analysis

The FIFO implementation of Push-Relabel provides a strong theoretical bound suitable for dense graphs:

- **Relabel Operations:** Each vertex can be relabeled at most $2|V|$ times. Total cost: $O(V^2)$.
- **Saturating Pushes:** At most $O(VE)$ operations.
- **Non-Saturating Pushes:** This is the bottleneck. In the FIFO variant, the number of non-saturating pushes is bounded by $O(V^3)$.

Thus, the overall time complexity is $O(V^3)$. This compares favorably to Edmonds-Karp ($O(VE^2)$) on dense networks where $E \approx V^2$.

2 Code and Data Availability

The full C++ implementation of the Push-Relabel algorithm, along with the scripts used to generate the 1,000 synthetic flow networks for validation, is open-source and available online:

`https://github.com/ashdane/push-relabel-implementation`

3 Empirical Validation: Max-Flow Min-Cut Theorem

A core objective of this project is to validate the Max-Flow Min-Cut theorem empirically. The theorem states that the maximum amount of flow possible from source to sink is equal to the capacity of a minimum cut that separates them.

3.1 Min-Cut Extraction Methodology

Upon termination of the Push-Relabel algorithm, we extract the Minimum Cut using the residual graph G_f:

1. Perform a Breadth-First Search (BFS) starting from the source s using only edges with strictly positive residual capacity ($c(u, v) - f(u, v) > 0$).
2. Let S be the set of all vertices reachable from s in G_f.
3. Let T be the set of all vertices not reachable ($V \setminus S$).
4. The Minimum Cut consists of all edges (u, v) in the original graph such that $u \in S$ and $v \in T$.

3.2 Verification Results

We tested the implementation on $N = 1000$ synthetic flow networks. In every instance, the computed Maximum Flow value equaled the sum of the capacities of the edges in the extracted Minimum Cut.

Table 1: Max-Flow vs. Min-Cut Capacity Verification (Sample Data)

Graph Size ($\|V\|$)	**Density**	**Max Flow (Computed)**	**Min-Cut Capacity**	**Discrepancy**
50	Sparse	450	450	0
50	Dense	12,400	12,400	0
500	Sparse	3,215	3,215	0
500	Dense	89,550	89,550	0

The consistency of these results ($Discrepancy = 0$) confirms the correctness of our Push-Relabel implementation and empirically validates the theorem.