More Max Flow: The Push-Relabel Algorithm

Today, we will discuss a class of max-flow algorithms that do not use augmenting paths. The new approach is called push-relabel. We will describe the intuition and the basic operations of the push-relabel algorithm; and we will analyze the generic implementation of this algorithm.

1 Intuition Behind Push-Relabel

In algorithms based on the Ford-Fulkerson approach, we keep at every step a feasible flow, and we stop when we reach a step in which there is no $s \rightarrow t$ path in the residual network. The push-relabel algorithm takes a somewhat complementary approach: at every step we have an assignment of flows to edges which is not a feasible flow (it violates the conservation conditions), but for which we can still define the notion of a residual network (via the same definition in lecture). The algorithm maintains the condition that, at every step, $t$ is not reachable from $s$ in the residual network. The algorithm stops when the preflow becomes a feasible flow. Then by the max-flow-min-cut theorem, the resulting flow must be a max-flow.

Let’s make the notion of preflow more precise:

**Definition 1** A preflow is a function $f : V \times V \mapsto \mathbb{R}$ satisfying:

- $f(u, v) = -f(v, u),$
- $f(u, v) \leq c(u, v),$
- $\sum_{u \in V} f(u, v) \geq 0,$ for all $v \in V - \{s, t\}.$

Note that the third condition in the definition above implies that for every vertex, the flow into it can exceed the flow out of it. This naturally leads to a notion of excess flow:

**Definition 2** For all $v \in V - \{s, t\},$ the excess flow at $v,$ $e_f(v),$ is

$$e_f(v) = \sum_{u \in V} f(u, v).$$

We call a vertex $v$ overflowing if $e_f(v) > 0.$ Therefore, a preflow becomes a feasible flow when there are no more overflowing vertices in the network.

Given a preflow $f,$ we compute the capacities of its residual network, $G_f = (V, E_f),$ as we did with a feasible flow: $c_f(u, v) = c(u, v) - f(u, v).$ The main objective of the push-relabel algorithm is simply to push (to be defined later) the excess flow at all the overflowing vertices towards the sink, along the residual edges. If the excess flow cannot reach the sink, the algorithm pushes it
Recitation 8: More Max Flow: The Push-Relabel Algorithm

backwards to the source. The algorithm terminates when no more excess flow exists. A potential problem of this approach is that an excess flow can be pushed along circles indefinitely. To prevent this, we associate to each vertex \( v \) a height, \( h(v) \), and we stipulate that we can push excess flow along edge \((u, v)\) if and only if \( h(u) > h(v) \). We will see later that the heights are updated to allow more push operations; this updating step is called relabel.

## 2 Basic Operations

The following are the three basic operations that constitute the push-relabel algorithm.

### 2.1 INITIALIZE\((G, s)\)

This is carried out once during the beginning of the algorithm. There are two steps:

1. The height of the source is set to \(|V|\) and the heights of all non-source vertices are set to 0:

   \[
   h(v) = \begin{cases} 
   |V| & \text{if } v = s \\
   0 & \text{otherwise}
   \end{cases}
   \]

2. A preflow is generated by filling the capacity of each edge leaving \( s \) and setting all other edges empty:

   \[
   f(u, v) = \begin{cases} 
   c(u, v) & \text{if } u = s \\
   -c(v, u) & \text{if } v = s \\
   0 & \text{otherwise}
   \end{cases}
   \]

3. Given the preflow from step 2, all the excess flow, \( e_f(v) \), is computed according to the definition.

### 2.2 PUSH\((u, v)\)

If vertex \( e_f(u) > 0, e_f(u, v) > 0 \) and \( h(u) > h(v) \), then the excess flow can be pushed onto the residual edge \((u, v)\). The amount of the flow pushed is given by \( \Delta_f(u, v) = \min\{e_f(u), c_f(u, v)\} \).

The effect of this push is

1. \( f(u, v) \leftarrow f(u, v) + \Delta_f(u, v) \) and \( f(v, u) \leftarrow -f(u, v) \).
2. \( e_f(u) \leftarrow e_f(u) - \Delta_f(u, v) \).
3. \( e_f(v) \leftarrow e_f(v) + \Delta_f(u, v) \).
A push operation is **saturating** if edge \((u, v)\) in the residual network becomes saturated \((e_f(u, v) = 0)\) afterwards; otherwise, it is **nonsaturating**. A simple lemma characterizes one result of a nonsaturating push.

**Lemma 1** After a nonsaturating push from \(u\) to \(v\), the vertex \(u\) is no longer overflowing.

**Proof.** Since the push was nonsaturating, the amount of flow \(\Delta f(u, v)\) actually pushed must equal \(e_f(u)\) prior to the push. Since \(e_f(u)\) is reduced by this amount, it becomes \(0\) after the push.

Intuitively, saturating pushes make progress by removing residual edges and nonsaturating pushes make progress by removing overflowing vertices.

### 2.3 RELABEL\((u)\)

This operation applies when \(u\) is overflowing and \(h(u) \leq h(v)\) for all residual edges \((u, v)\). In other words, \(u\) is not “high” enough to push its excess flow along one of its outgoing residual edges. So we increase the height of \(u\):

\[
h(u) \leftarrow 1 + \min\{h(v) : (u, v) \in E_f\}.
\]

### 3 Analysis of The GENERIC-PUSH-RELABEL-ALGORITHM

Initialization, followed by a sequence of push and relabel operations, executed in no particular order, yields the GENERIC-PUSH-RELABEL-ALGORITHM:

**GENERIC-PUSH-RELABEL\((G, s)\)**

1. **INITIALIZE\((G, s)\)**
2. **while** there exists an applicable push or relabel operation:
   3. select an applicable push or relabel operation and perform it

### 3.1 Correctness of Push-Relabel Assuming Termination

We first prove that if the generic push-relabel algorithm terminates, the outputted preflow \(f\) is a max flow. As mentioned in Section 1, we will show that the algorithm maintains the condition that, at every step, \(t\) is not reachable from \(s\) in the residual network.

**Lemma 2** At all times during and after the execution of the algorithm, \(\forall\) residual edge \((u, v) \in E_f\), \(h(u) \leq h(v) + 1\).
Proof. We proceed by induction on the number of steps in the algorithm. The base case is after INITIALIZE(G, s). Here, the only residual edges either connect two vertices of heights 0 or connect vertices of height 0 to the source, which has height |V|. So in both cases, the lemma is true.

Now assume that the algorithm just invokes RELABEL(u). Let v be the vertex of minimum height to which u has a residual edge. Since the height of u is increased, we need to check that the lemma still holds for all residual edges that have u as an end point. By definition of RELABEL, h(u) = h(v) + 1, so this is true for the residual edge (u, v). For any other edge (u, w), we know h(v) ≤ h(w) by our choice of v. Therefore the lemma holds for all these edges too. Finally, for any residual edge (w, u), since u’s height only goes up, the lemma will continue to hold as well.

Finally, assume that the algorithm just invokes PUSH(u, v). After this operation, the edge (v, u) must be present (if it was not before) in the residual network. So we need to check that the lemma holds for (v, u). We know that h(u) > h(v) (precondition of PUSH) and h(u) ≤ h(v) + 1 (inductive hypothesis), therefore h(u) = h(v) + 1 ⇒ h(v) = h(u) - 1 ⇒ h(v) ≤ h(u) + 1.

Lemma 3 At all times during and after the execution of the algorithm, there is no path from source s to sink t in the residual network.

Proof. Assume the contrary that there is a simple path \{v_0, v_1, ..., v_{k-1}, v_k\} from v_0 = s to v_k = t. Then, by Lemma 2, we must have h(v_i) ≤ h(v_{i+1}) + 1 for all i ∈ {0, 1, ..., k - 1}. This implies that h(s) ≤ h(t) + k, or |V| ≤ k. Therefore, the number of vertices in this simple path, k + 1, must be strictly great than |V|. This is a contradiction.

As a direct consequence of Lemma 3, there is no augmenting path from s to t when the algorithm terminates. Since the algorithm terminates with a feasible flow f (no more overflowing vertices), f must be a max flow.

3.2 Proving Termination and Analyzing Run-Time

To show that the generic push-relabel algorithm indeed terminates, we shall bound the number of operations it performs first. Motivated by our discussion above, we classify these operations as relabels, saturating pushes and nonsaturating pushes.

Lemma 4 The total number of relabel operations is at most 2|V|^2.

Lemma 5 The total number of saturating pushes is at most 2|V||E|.

Lemma 6 The total number of nonsaturating pushes is at most 4|V|^2(|V| + |E|).

You can consult section 26.4 of CLRS for proofs of the three lemmas above. As an immediate consequence of these lemmas, we have the following:

Theorem 7 The number of basic operations in the GENERIC-PUSH-RELABEL-ALGORITHM is O(V^2E).
However, we are not done yet... We care about the actual run-time of the algorithm instead of just the number of operations. It turns out (Exercise 26.4-2 in CLRS) that we can design a priority-queue data structure on the overflowing vertices. That is, at each step, we always choose the most-overflowing vertex in the queue in $O(1)$ time; we then apply the appropriate operation (push or relabel) to this vertex. Under this “most-overflowing-vertex-first” selection rule, we can implement the generic push-relabel algorithm so that there is only an overhead of $O(V)$ per relabel operation and $O(1)$ per push. This implies that the run-time of the algorithm is asymptotically the same as the number of basic operations:

**Corollary 8** There is an implementation of the generic push-relabel algorithm that runs in $O(V^2E)$ time on any flow network $G = (V, E)$.

### 4 Improve Running Time

The generic push-relabel algorithm is underspecified in that there could be more than one vertex out of which a push/relabel operation is allowed. Our above run-time analysis is based on the “most-overflowing-vertex-first” selection rule. Better run-times can be achieved if we use more sophisticated selection rule. For example, the relabel-to-front rule (section 26.5 in CLRS) has $O(V^3)$ run-time. Asymptotically, all these two push-relabel algorithms are more efficient than the Edmonds-Karp algorithm, which runs in $O(VE^2)$ time.