1 Proving that push-relabel works

Last time, we saw how to perform the push-relabel algorithm. We can keep doing push or relabel steps for as long as our preflow has active nodes. Therefore, when we stop, there are no active nodes: for every node $k \neq s, t$, $\Delta_k(\mathbf{x}) = 0$. This makes our final preflow a flow.

However, we don’t actually know that the preflow we end with is a maximum flow, which is what we want. Proving this will be our goal today.

We begin with a technical lemma about the algorithm.

Lemma 1.1 (The “No Steep Drops” Rule). When performing the push-relabel algorithm, there can only be a residual arc (forward or backward) from $i$ to $j$ if their labels satisfy $\text{height}(j) \geq \text{height}(i) - 1$.

That is, residual arcs can go up in height by any amount, they can be horizontal, or they can drop by one unit of height. But there can’t be any drops that go further down than that.

Some valid and invalid drops are illustrated in the diagram below:

Proof. The idea of the proof is to show that this lemma starts out being true, and does not become false after any single step of the algorithm. So it always stays true.

First of all, why does it start out true? Recall that we initialize the algorithm by setting $\text{height}(s) = n = (\# \text{ of nodes})$, $\text{height}(k) = 0$ for all nodes $k \neq s$, and setting $x_{sj} = c_{sj}$ for every arc $(s, j)$ out of $s$. Here’s an example of the initial step from yesterday:

\[ s \rightarrow (0) \rightarrow a \rightarrow (1) \rightarrow b \rightarrow (2) \rightarrow t \]

\[ \begin{array}{ccccccc}
(5) & (4) & (3) & (2) & (1) & (0) \\
\hline
b & a & c & d & e & f
\end{array} \]

Proof: The idea of the proof is to show that this lemma starts out being true, and does not become false after any single step of the algorithm. So it always stays true.

First of all, why does it start out true? Recall that we initialize the algorithm by setting $\text{height}(s) = n = (\# \text{ of nodes})$, $\text{height}(k) = 0$ for all nodes $k \neq s$, and setting $x_{sj} = c_{sj}$ for every arc $(s, j)$ out of $s$. Here’s an example of the initial step from yesterday:

\[ s \rightarrow (0) \rightarrow a \rightarrow (1) \rightarrow b \rightarrow (2) \rightarrow t \]

\[ \begin{array}{ccccccc}
(5) & (4) & (3) & (2) & (1) & (0) \\
\hline
b & a & c & d & e & f
\end{array} \]

\[ s \rightarrow (0) \rightarrow a \rightarrow (1) \rightarrow b \rightarrow (2) \rightarrow t \]

\[ \begin{array}{ccccccc}
(5) & (4) & (3) & (2) & (1) & (0) \\
\hline
b & a & c & d & e & f
\end{array} \]
Most arcs in the residual graph are horizontal, and this satisfies the “no steep drops” rule. The only danger would be a residual arc out of $s$ to some other node. But these don’t exist: since all arcs $(s, j)$ are at full capacity, the residual graph only has a backward arc from $j$ to $s$.

Now we think about what can happen when we push or relabel.

When we push along an arc $(i, j)$, we always have $\text{height}(j) = \text{height}(i) - 1$. The only possible change to the residual graph is that this arc might disappear, and that the reverse arc $(j, i)$ will be created. But that reverse arc goes up, not down: it won’t violate the “no steep drops” rule.

When we relabel an active node $i$, we are lifting it only the minimum distance to create a downward arc, so we don’t create any steep drops. If some arc $(i, j)$ had $\text{height}(j) < \text{height}(i) - 1$ after relabeling, then we relabeled incorrectly: we should have lifted node $i$ to the lower height $\text{height}(j) + 1$ instead.

So the “no steep drops” rule is never violated, and always stays true.

From this rule, it is quick to show that the push-relabel algorithm ends (if it does end) with a maximum flow.

**Theorem 1.2.** When there are no active nodes left in the push-relabel algorithm, the resulting preflow is a maximum flow.

**Proof.** We already saw that when there are no active nodes left, flow conservation holds at every node $k \neq s, t$, and therefore the preflow we get is a flow.

We can show that the flow is maximum by showing that there is no augmenting path: no path from $s$ to $t$ in the residual graph. Then (as with the Ford–Fulkerson algorithm), setting $S$ to be the set of all nodes reachable from $S$, and $T$ to be the set of all other nodes, produces a cut whose capacity matches the value of the flow. The argument is the same as we saw with Ford–Fulkerson.

Why is there no augmenting path? Because we always have $\text{height}(s) = n$ (the number of nodes) and $\text{height}(t) = 0$. Any path from height $n$ to height $0$ has at least $n$ steps, by the “no steep drops” rule: each step can only reduce the height by at most 1.

However, there can’t be a path with $n$ steps: since there are only $n$ nodes in the whole network, even a path that included all $n$ nodes would only take $n - 1$ steps. So there is no way to get from $s$ to $t$ in the residual graph.

2 Bounding the number of steps

As always, we want to know, not just “when the algorithm terminates, it gives the right answer”, but “the algorithm terminates”. Better yet, we’d like to have a bound on the number of steps.

Let’s say that we have a network with $n$ nodes and $m$ arcs. Also, let’s resolve one ambiguity in the push-relabel algorithm as stated so far: let’s say that when we do a push step, we always push on an active node with the highest label.

We will prove two lemmas: one about labels of nodes, and one about counting steps between changing those labels.
Lemma 2.1. In the push-relabel algorithm, the label of any node \( k \) always satisfies
\[
0 \leq \text{height}(k) < 2n.
\]

Proof. The proof for this is similar as before: it relies on the no steep drops rule.

When a node \( k \) is active, there is flow along some path from \( s \) to \( k \): a path
\[
s \to u_1 \to u_2 \to \cdots \to u_i \to k.
\]

In the residual graph, this means that the backward arcs going the other way exist:
\[
s \leftarrow u_1 \leftarrow u_2 \leftarrow \cdots \leftarrow u_i \leftarrow k.
\]

There are at most \( n \) nodes in this reverse path, and so there are at most \( n - 1 \) arcs along it. Along each arc, the height can’t drop by more than 1. Since the reverse path ends at \( s \), and \( \text{height}(s) = n \), the starting point \( k \) must satisfy \( \text{height}(k) \leq n + (n - 1) < 2n \).

Lemma 2.2. In the push-relabel algorithm, we can do at most \( m \) push steps without relabeling.

Proof. If we follow the rule of always pushing on a highest-labeled node, then the excess flow can only move down. After we push as much as we can on a node, one of two things happens:

- The node becomes inactive, and possibly some nodes below it become active. This node can’t become active again, and so we can’t push on any arcs out of the node again.
- The node runs out of downward arcs to push on. Those arcs can’t change again until we relabel, because you can’t push along an upward arc.

Here is an example illustration of several steps of these changes:

In particular, for each arc \((i, j)\) in the network, we can only push in one of the directions \( i \to j \) or \( i \leftarrow j \) in the residual graph, and we can only do it once between relabeling steps.

Theorem 2.3. The push-relabel algorithm finishes in at most \( 2n^2 \) relabel steps and \( 2n^2m \) push steps total.

Proof. We can relabel any given node fewer than \( 2n \) times, since it starts at height 0 and can never exceed height \( 2n \). There are \( n \) nodes, so we can do at most \( 2n^2 \) relabel steps.

Between each of those relabel steps, we can do at most \( m \) push steps, for \( 2n^2m \) push steps total.
This guarantees that the push-relabel algorithm terminates, and actually shows that it’s better than previous algorithms we looked at. (We did not prove it, but using augmenting paths may require time proportional to $nm^2$, which is potentially much larger.)

Actually, our analysis of the push-relabel algorithm is not perfect. Just with a more careful proof, we can show that we need at most a number of steps proportional to $n^2 \sqrt{m}$. And there are fancy ideas for how to choose order of pushing and relabeling that make the algorithm even more effective.

3 A possible improvement

Here’s one possible improvement to the algorithm; it does not actually improve the worst case by much, but on average it can cut down the number of steps a bit.

Our starting labeling (where we give each node other than $s$ a label of 0) is inefficient: it does not take into account information we already have.

Instead, give the nodes the following labels. As before, set $height(s) = n$ and $height(t) = 0$. But for each other node $k$, set $height(k)$ to be the length of the shortest path from $k$ to $t$. The old and the new starting labeling is illustrated below:

![Diagram](attachment:image.png)

This labeling gives each vertex essentially the highest label that would not violate the “no steep drops” rule. It lets us do a lot more starting push steps before we have to relabel, and so we get to the end of the algorithm more quickly.