Lecture 12

In which we prove that the basic implementation of the push-relabel algorithm runs in time $O(|V|^2 \cdot |E|)$.

1 The Push-Relabel Algorithm

In the last lecture we described the *push-relabel* method to solve the maximum flow problem.

In this method, instead of maintaining a feasible flow, and improving it until it becomes optimal, we maintain a *preflow*, which is an assignment of flows to edges that allows vertices to have more incoming flow than outgoing flow (but not vice versa) and that satisfies the capacity constraints.

Definition 1 (Preflow) *An assignment of a non-negative flow $f(u,v)$ to each edge (u,v) of a network $(G=(V,E),s,t,c)$ is a preflow if*

- *for each edge (u,v), $f(u,v) \leq c(u,v)$*
- *for each vertex $v \in V - \{t\}$,*

$$\sum_u f(u,v) - \sum_w f(v,w) \geq 0$$

we define the excess flow at u as

$$e_f(v) := \sum_u f(u,v) - \sum_w f(v,w)$$

Furthermore, we will also assume that if f is a flow or a preflow, then it is never the case that there are two vertices (u,v) such that both $f(u,v) > 0$ and $f(v,u) > 0$. (We can always transform an arbitrary flow or preflow to an equivalent one that satisfies this condition, by subtracting the minimum of the two amounts from both $f(u,v)$ and $f(v,u)$. The algorithm will automatically construct preflows that satisfy the condition.)

The *residual network* with respect to a preflow is defined like the residual network of a flow. The capacity of an edge (u, v) in the residual network is

$$c'(u, v) := c(u, v) - f(u, v) + f(v, u)$$

At the start of the algorithm, s sends flow to its neighbors, completely saturating all its outgoing edges. This means that, at the start of the algorithm, t is not reachable from s in the residual network. The algorithm will maintain this invariant through every step.

Eventually, the algorithm reaches a point at which all the excess flows are zero at all vertices (except s and t), which means that the preflow is actually a feasible flow. Since we maintained the invariant that there is no path from s to t in the residual network, the flow that we have at that point is an optimal flow.

To eventually reach a feasible flow, we want to push flow out of vertices that have positive excess flow, but we want to do so in a carefully arranged way so that the algorithm converges quickly. The algorithm maintains, for every vertex v, a *height* $h(v)$. Through the various phases of the algorithm, the heights will "self-organize" so that the flow will mostly flow downhill, and the heights will roughly correspond to the distance from a vertex to t. (The actual behavior will be somewhat more complicated.)

At each step, we pick a vertex v with positive excess flow, and select an outgoing edge (v, w) with positive residual capacity, and push flow out of v and into w, provided that w is "downhill from v," that is, $h(v) > h(w)$. If there is no vertex for which we can do such a *push* operation, we take any vertex with positive excess flow, and increase its height. (This is called a *relabel* operation.)

This is all that the algorithm is doing, and the following is a complete description of the algorithm:

- Input: network $(G = (V, E), s, t, c)$
- $h[s] := |V|$
- for each $v \in V - \{s\}$ do $h[v] := 0$
- for each $(s, v) \in E$ do $f(s, v) := c(s, v)$
- while f is not a feasible flow
  - let $c'(u, v) = c(u, v) + f(u, v) - f(v, u)$ be the capacities of the residual network
  - if there is a vertex $v \in V - \{s, t\}$ and a vertex $w \in V$ such that $e_f(v) > 0$, $h(v) > h(w)$, and $c'(v, w) > 0$ then
    - push $\min\{c'(v, w), e_f(v)\}$ units of flow on the edge (v, w)
  - else, let v be a vertex such that $e_f(v) > 0$, and set $h[v] := h[v] + 1$
- output f

Note that, by this description, it is not even clear that the algorithm converges in finite time. Our goal today is to show that it does, and to prove that the running time is $O(|E| \cdot |V|^2)$.

2 Analysis of the Push-Relabel Algorithm

We begin by showing that no vertex can reach a height bigger than $2 \cdot |V| - 1$. This automatically puts an upper bound to the number of *relabel* operations that can be executed, and is an important starting point in analyzing the number of push operations.

Lemma 2 *At every step, if a vertex v has positive excess flow, then there is a path from v to s in the residual network.*

PROOF: First, let us why this is "obvious:" in a preflow, vertices are allowed to "destroy" stuff, but not to "create" it, so if a vertex has positive excess flow, then in particular it has positive incoming flow, and the incoming stuff must be coming from s along a path made of edges with positive flow. To each such edge corresponds an edge in the opposite direction with positive capacity in the residual network, and so v is connected to s in the residual network.

The only part of the above argument that is not rigorous is when we say that if a vertex v has positive excess flow then there must be a path from s to v made entirely

of edges with positive flow. Let A be the set of vertices which are reachable from s via such a path. Because of the preflow constraints on the vertices $v \notin A$, we have

$$\sum_{v \notin A} e_f(v) = \sum_{v \notin A} \left(\sum_u f(u,v) - \sum_w f(v,w) \right) \geq 0$$

but, in the second expression, all terms of the form $f(x,y)$ in which $x \notin A$ and $y \notin A$ cancel, because they appear once with a plus sign and once with a minus sign. The result of such cancellations is

$$\sum_{v \notin A} e_f(v) = \sum_{u \in A, v \notin A} f(u,v) - \sum_{v \notin A, w \in A} f(v,w) \leq 0$$

where the last inequality follows from the fact that $f(u,v)$ must be zero when $u \in A$ and $v \notin A$.

So we have that $e_f(v) = 0$ for every $v \notin A$, which means that every vertex v that has $e_f(v) > 0$ must be in A, and so it must be reachable from s via a path made of edges with positive flow.□

The connection between the previous lemma and the task of bounding the heights of vertices comes from the following observation.

Lemma 3 *At every step, if there is an edge (u,v) that has positive capacity $c'(u,v) > 0$ in the residual network, then*

$$h(u) \leq h(v) + 1$$

Before proving the lemma, let us understand what it is getting at. Our intuition for the heights, is that we want the flow to go "downhill," and in fact every time we do a push operation we do so from a higher vertex to a lower one. If the flow goes downhill, the edges with positive residual capacity go "uphill." This is not exactly true at each step, because of the relabeling operations, but the lemma is saying that edges with positive residual capacity cannot go downhill by more than one unit.

PROOF: We show that the property is an invariant preserved by the algorithm at each step.

At the beginning, the residual network contains: (i) the edges of the original networks between vertices other than s, and all such vertices have the same height 0, and (ii) edges from the neighbors of s to s, and such edges go uphill.

Now we show that the property is preserved at each step.

If we do a relabel step on a vertex v, the property remains true for all the edges (u, v) with positive capacity coming into v. About the edges (v, w) with positive capacity coming out of v, if we did a relabel step it was because we had $h(v) \leq h(w)$; after the relabel, we still have $h(v) \leq h(w) + 1$.

If we do a push step along an edge (u, v), we might introduce the reverse edge (v, u) in the residual network. The push step, however, happens only when $h(u) > h(v)$, and so the edge (v, u) satisfies the property. □

Fact 4 *At every step, if there is a path from u to v in the residual network, then*

$$h(u) \leq h(v) + |V| - 1$$

PROOF: If there is a path of length ℓ from u to v in the residual network, then, by applying ℓ times Lemma 2, we have $h(u) \leq h(v) + \ell$, and if there is a path from u to v there must be a path of length at most $|V| - 1$. □

We can now being to draw conclusions that are relevant to our analysis.

Fact 5 *At each step of the algorithm, there is no path from s to t in the residual network.*

Because, if there were such a path, we would have $h(s) \leq h(t) + |V| - 1$, but at the beginning we have $h(s) = |V|$ and $h(t) = 0$, and the heights of s and t never change.

This means that *if the algorithm terminates, then it outputs an optimal flow.* From now on, it remains to estimate the running time of the algorithm, which we do by finding upper bounds to the number of times the various operations can be executed.

Fact 6 *At each step of the algorithm, every vertex has height at most $2|V| - 1$.*

PROOF: Each time the height of a vertex v is increased, it is because it has positive excess flow. If a vertex v has positive excess flow, then there is a path from v to s in the residual network. If there is such a path, then $h(v) \leq h(s) + |V| - 1 \leq 2|V| - 1$. □

Fact 7 *The algorithm executes at most $(|V|-2)\cdot(2\cdot|V|-1) \leq 2|V|^2$ relabel operations.*

PROOF: There are at most $|V| - 2$ vertices on which the relabel operation is admissible, and on each of them the algorithm executes the operation at most $2 \cdot |V| - 1$ times. □

We now estimate the number of *push* operations.

We call a push operation *saturating* if it uses the entire residual capacity of edge, making it disappear from the residual network. Otherwise, the push operation is *nonsaturating*.

Fact 8 *The algorithm executes at most* $2|V| \cdot |E|$ *saturating push operations.*

PROOF: Consider an edge (u, v). The first time there is a saturating push from u to v, it is because $h(u) > h(v)$. After the saturating push, the edge (u, v) disappears from the residual network, and so there cannot be any other saturating push from u to v (and, indeed, no push of any kind), until v sends back some flow to u with a push in the opposite direction. But for this to happen we must first have $h(v) > h(u)$, which requires at least two relabels of v. For the next saturating push from u to v, we must have again $h(u) > h(v)$, which requires two more relabels, at least. So, between two saturating pushes from u to v, at least four relabels must take place on u and v. Overall, u and v can relabeled at most $4|V|$ times, and so there can be at most $|V|$ saturating pushes.

There are $2 \cdot |E|$ edges that can appear in the residual network, and so in total we have at most $2 \cdot |V| \cdot |E|$ saturating pushes. □

The most interesting part of the analysis is how we analyze the number of non-saturating push operations.

At each step of the execution of the algorithm, we define the "energy" of the current preflow f as

$$\Phi(f) := \sum_{v : e_f(v) > 0} h(v)$$

the sum of the heights of all vertices that have positive excess flow. The algorithm starts in a zero-energy state, but the energy becomes one after the first relabel operation. When the energy becomes zero again, it is because there are no nodes with excess flow, and the algorithm stops.

We have the following observations.

Fact 9 *Each relabel step increases the energy by exactly one unit.*

Fact 10 *Each saturating push increases the energy by at most* $2|V|$ *units.*

PROOF: A push step along an edge (u, v) does not change the height of any vertex, but it could give excess flow to vertex v, which possibly had zero excess flow before, so that the energy increases by $h(v) \leq 2|V|$ units. □

Fact 11 *Each nonsaturating push decreases the energy by at least one unit.*

PROOF: If we do a push on an edge (u, v), why would the push be nonsaturating? The only reason why we would not saturate the edge is that the excess flow of u is less than the residual capacity of (u, v), and so we can push the entire excess flow of u along (u, v) with residual capacity to spare. But this means that, after a nonsaturing push along (u, v), the excess flow of u becomes zero, and so $h(u)$ is not counted in the energy any more. It is possible that v had no excess flow before the push and now it does, which means that we need to add $h(v)$ in the energy, but we still have that the new energy is at most the old energy minus $h(u)$ plus $h(v)$ and, recalling that we do a push only if $h(v) < h(u)$, we have that the new energy is at most the old energy minus one. □

Fact 12 *The total number of nonsaturating pushes is at most* $2|V|^2 + 4|V|^2|E|$.

PROOF: If, at some step of the execution of the algorithm, the preflow is not yet a feasible flow, then the energy must by > 0. If, up to that point, the algorithm has executed r relabel operations, sp saturating push operations, and np nonsaturating push operations, then

$$0 < \Phi(f) \leq r + 2|V|sp - np$$

we now that $r \leq 2|V|^2$ and $sp \leq 2|V| \cdot |E|$, so the above expression implies

$$np < 2|V|^2 + 4|V|^2|E|$$

So if, at some point of the execution of the algorithm, we haven't reached the termination condition yet, this implies that we have executed fewer than $2|V|^2 + 4|V|^2|E|$ nonsaturating pushes.

Equivalently, when the algorithm terminates, it has executed at most $2|V|^2+4|V|^2|E|$ nonsaturating pushes. □

Overall, we have a total of at most $O(|V|^2 \cdot |E|)$ operations, and each can be implemented in $O(1)$ time, so the running time of the algorithm is $O(|V|^2 \cdot |E|)$.

3 Improved Running Time

We note that the algorithm is somewhat underspecified, in the sense that there could be more than one vertex out of which a push operation is allowed, and we are not saying how to pick one; if no push operation is possible, any of the vertices with positive excess can be picked for a relabel operation, and we are not specifying how

to pick one. The analysis of the running time applies to any possible way to make such choices.

A reasonable heuristic is that if we have the choice of multiple vertices out of which to push flow, then we choose the vertex of biggest height. It can be proved (but we will not), this implementation of the algorithm executes at most $O(|V|^2\sqrt{|E|})$ nonsaturating pushes, and so the running time is $O(|V|^2\sqrt{|E|})$. A more complicated implementation, in which multiple pushes are done together, and the *dynamic tree* data structure is used to keep information about the current preflow, has running time $O(|V| \cdot |E| \cdot \log |V|)$. In terms of worst-case running time, this is the best known for strongly polynomial algorithms.

An algorithm of Goldberg and Rao has running time

$$O((\min\{|E| \cdot |V|^{2/3}\ ,\ |E|^{1.5}) \cdot (\log |V| \cdot \log opt))$$

In the interesting case in which $|E| = O(|V|)$, this is roughly $|V|^{1.5}$, compared the to the $|V|^2$ running time of the optimized push-relabel algorithm. This year, a new algorithm has been discovered that, in undirected networks, finds a flow of cost $\geq (1-\epsilon) \cdot opt$ in time $O(|E|^{4/3} \cdot \epsilon^{-4} \cdot (\log |V|)^{O(1)})$.

There has been extensive experimental analysis of maximum flow algorithms. The fastest algorithms in practice are carefully tuned push-relabel implementations.