Max flow (cont'd)

Given G, with capacities $c(u,v)$, compute max s-t flow.

Push-Relabel maintains a preflow f:

$$\forall u \neq s,t, \quad \sum_v f(v,u) \geq \sum_v f(u,v)$$

$$\forall (u,v), 0 \leq f(u,v) \leq c(u,v),$$

(def $e_f(u) := \sum_v f(v,u) - \sum_v f(u,v)$)

and a height function h:

$h(s) = n$, $h(t) = 0$, and

if $(u,v) \in E_f$, then $h(u) \leq h(v)+1$.

↑ residual network

Initialize:

set $h(s) \leftarrow n$, $h(u) \leftarrow 0$ for all $u \neq s$

set $f(s,u) \leftarrow c(s,u)$ for all $(s,u) \in E$

$f(u,v) \leftarrow 0$ o.w.

Push: let u be a vertex with $e_f(u) > 0$,
$(u,v) \in E_f$ and $h(u) = h(v)+1$,

"push" $\delta = \min\{c_f(u,v), e_f(u)\}$ amount of flow through (u,v):

$f(u,v) \leftarrow f(u,v) + \delta$.

Relabel: let u be a vertex with $e_f(u) > 0$
$\forall v$ s.t. $(u,v) \in E_f$, $h(u) \leq h(v)$.

increase $h(u)$:

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

General algo:

Initialize
while $\exists u$ s.t. $e_f(u) > 0$, $u \neq s,t$.
if $\exists (u,v) \in E_f$, $h(u) = h(v) + 1$
push(u,v)
else
relabel(u).

last lecture: when the algorithm terminates, it computes a max flow

Lemma: The max height of any vertex is at most $2n-1$. # relabels is $\leq 2n^2$.

To bound the running time, it suffices to bound # push.

Def: A push is saturated if $e_f(u) \geq c_f(u,v)$.
unsaturated o.w.

Lemma: # saturated push $\leq O(n \cdot m)$.

Proof: Consider each edge (u, v).

A saturated push removes (u, v) from G_f.

At the time of the push, $h(u) = h(v) + 1$.

v must be relabeled before flow can be pushed back along (v, u), and restore (u, v) in G_f.

Relabeling v only happens $O(n)$ times.

saturated push on $(u, v) \leq O(n)$. □

Lemma: # unsaturated push $\leq O(n^2 m)$.

Proof. An unsaturated push "moves" excess from u to v with $h(u) = h(v) + 1$.

Consider $\Phi(f) = \sum_{u : e_f(u) > 0} h(u)$. Each unsat push decreases Φ by ≥ 1.

Each relabel increases Φ by $\leq O(n)$

Each saturated push increases Φ by $\leq O(n)$.

Φ starts at $\leq O(n^2)$, total increase

$\leq O(n^2 m)$.

$\Phi \geq 0$. Thus, # unsat push $\leq O(n^2 m)$. □

Storing $(u,v) \in E_f$ with $h(u) = h(v)+1$ using adjacency lists implements each push in $O(1)$ time and each relabel in $O(n)$ time.

Total time: $O(n^2 m)$.

Relabel-to-front

reduce # unsaturated push.

```
Discharge(u):
    while e_f(u) > 0
        if ∃(u,v) ∈ E_f and h(u) = h(v)+1
            push(u,v)
        else relabel(u).
```

Relabel-to-front algo

Initialize

L = linked list of all vertices $\setminus \{s,t\}$ in any order

$u \leftarrow$ head of L

while $u \neq$ tail

discharge(u)

if u was relabelled during the discharge

move u to the front of L

$u \leftarrow$ vertex after u in L

Consider the network formed by all edges (u,v):

- $(u,v) \in E_f$ and $h(u) = h(v)+1$.

(Admissible network)

Lemma: Admissible network is a DAG.

(edges only point to low vertices)

Lemma: Push doesn't add edges to the admissible network.

Relabeling u eliminates all incoming edges to u, may introduce one or more outgoing edges.

(easy to verify, omit)

Lemma: L maintains a topological order w.r.t. the admissible network.

Proof: Prove by induction.

- After initialization, the admissible network is empty.
- Each "discharge" doesn't break the order.
  - if u is relabeled,
    u has no incoming edge,
    u is moved to the front
  - if u is not relabeled
    no new edges are added to the admissible network.

□

Lemma: There are at most $O(n^3)$ unsaturated push.

Proof: Consider # unsaturated pushes between two relabels. Each discharge has only one unsaturated push. L remains unchanged without relabels.
Hence, there are at most n unsaturated pushes between relabels, at most $O(n^3)$ in total. □

Thm: There are no overflowing vertices after the algorithm terminates.

Proof: After the last relabel, the algo makes a pass on the list L. Since only admissible edges are pointing to later vertices in L, no push makes a processed vertex overflow. All overflowing vertices are processed. □

The bound on #relabel, #saturated push remains.

$\leq O(n^2)$ $\leq O(n \cdot m)$

A careful implementation gives $O(n^3)$ time bound.