15.082J and 6.855J and ESD.78J
October 19, 2010

Max Flows 3

Preflow-Push Algorithms

Review of Augmenting Paths

At each iteration: maintain a flow x.

Let G(x) be the residual network.

At each iteration, find a path from s to t in G(x).

In the shortest augmenting path algorithm, we kept distance labels d( ), and we sent flow along the shortest path in G(x).

This lecture: present and analyze a different type of max flow algorithm called preflow-push. New analysis technique: potential functions.

Flows vs. Preflows

Augmenting Path Algorithm

Flow into i = Flow out of i

Push flow along a path from s to t

d(j) = distance from j to t in the residual network.

Preflow Algorithm

Flow into i ≥ Flow out of i
for i ≠ s.

Push flow in one arc at a time

d(j) ≤ distance from j to t in the residual network

- d(t) = 0
- d(i) ≤ d(j) + 1 for each arc (i, j) ∈ G(x),

Preflows

At each intermediate stages we permit more flow arriving at nodes than leaving (except for s)

A ***preflow*** is a function $x: A \rightarrow R$ s.t. $0 \leq x \leq u$ and such that

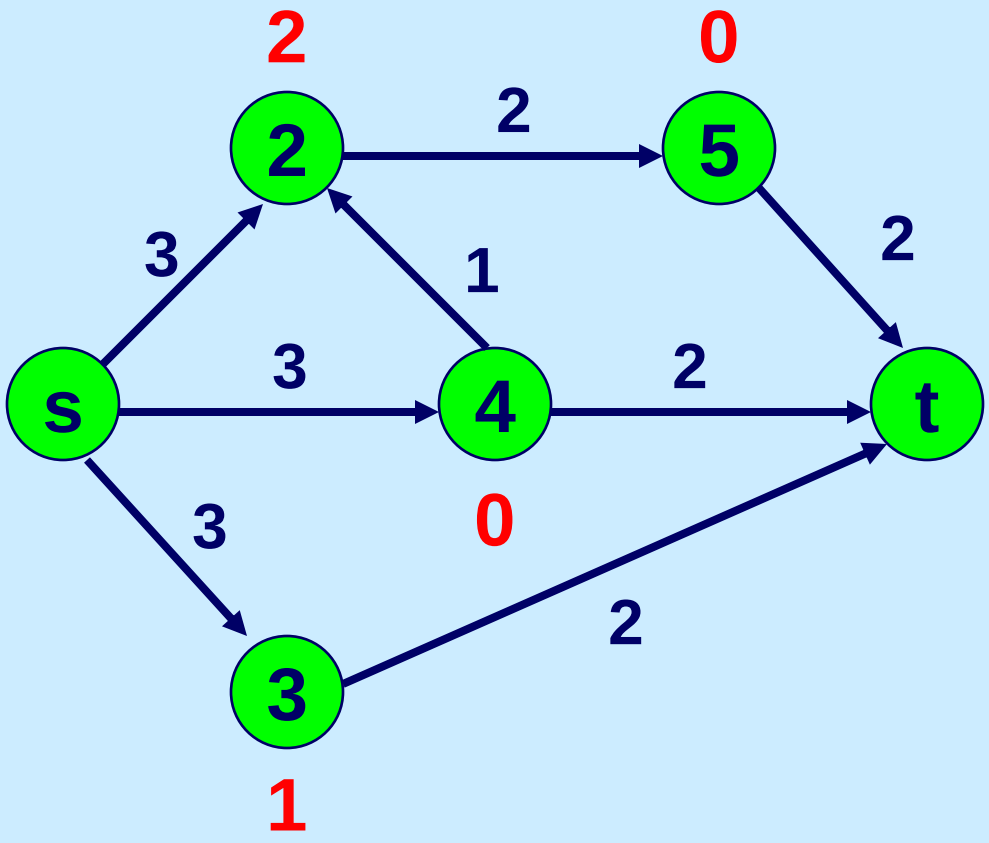
$$e(i) = \sum_{j \in N} x_{ji} - \sum_{j \in N} x_{ij} \geq 0,$$

for all $i \in N - \{s, t\}$.

i.e., e(i) = ***excess*** at i = net excess flow into node i.

The excess is required to be nonnegative.

A Feasible Preflow



The *excess e(j)* at each node j ≠ s, t is the flow in minus the flow out.

Note: total excess = flow out of s minus flow into t.

Distance Labels

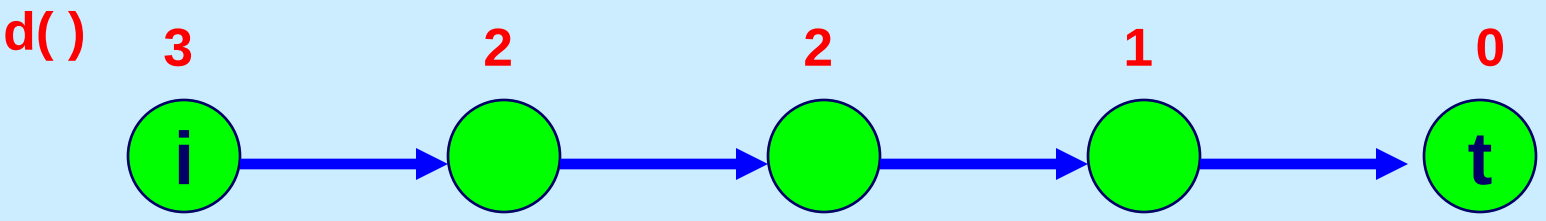
Distance labels **d( ) are** ***valid*** **for G(x) if**

i. $d(t) = 0$

ii. $d(i) \leq d(j) + 1$ for each $(i, j) \in G(x)$

Defn. An arc (i, j) is ***admissible*** **if** $r_{ij} > 0$ **and** $d(i) = d(j) + 1$.

Lemma. ***Let d( ) be a valid distance label. Then d(i) is a lower bound on the distance from i to t in the residual network.***



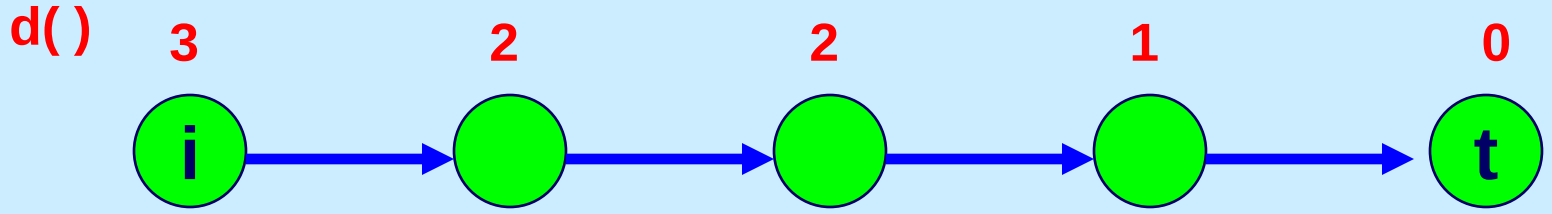
P = the shortest path from i to t in G(x)

Distance labels and gaps

We say that there is a gap at a distance level k (0 < k < n) if there is no node with distance label k.

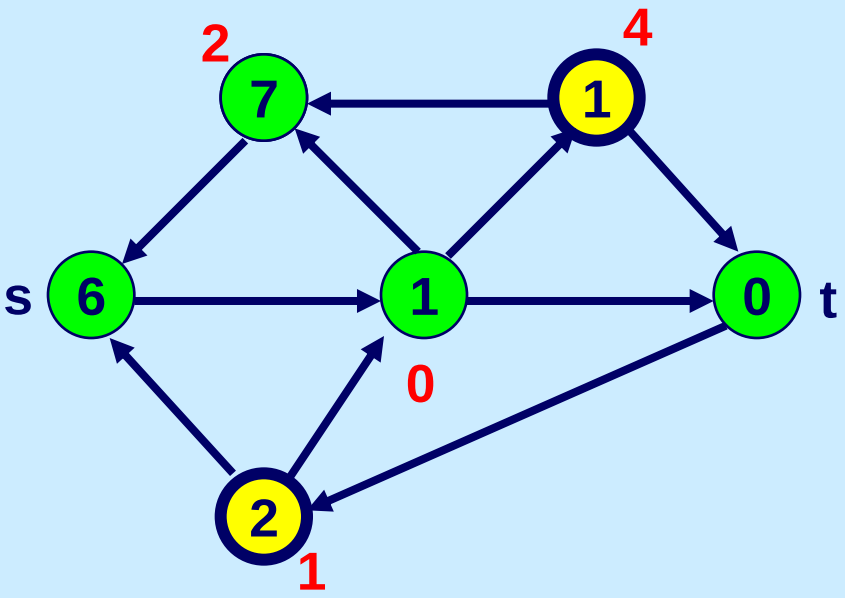
Lemma. Suppose there is a gap at distance level k. Then for any node j with d(j) > k, there is no path from j to t in the residual network.

Proof. The shortest path from j to t would have to pass through a node whose distance level is k.



P = the shortest path from i to t in G(x)

Active nodes in the residual network



k	d( ) = k
w	e( ) = w

A node j in G\{s} is active if:

- **e(j) > 0 and**
- **there is no gap at a distance level less than d(j)**

The preflow push algorithm will push flow from active nodes "towards the sink", relying on d( ).

Push/Relabel, the fundamental subroutine

Suppose we have selected an active node i.

Procedure Push/Relabel(i)

begin

if the network contains an admissible arc (i, j) then
push ™ : = min{ e(i), r_{ij} } units of flow from i to j;

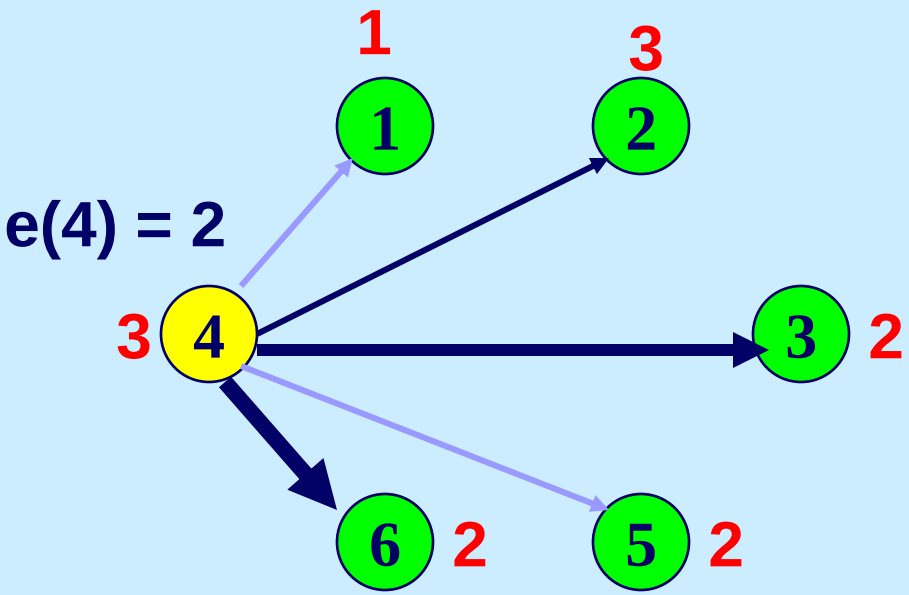
else replace d(i) by min{d(j) + 1 : (i, j) ∈ A(i) and r_{ij} > 0}

end;

Pushing using current arcs

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	4	Yes
4	5	0	No
4	6	2	Yes

Suppose that node 4 is active, and has excess.



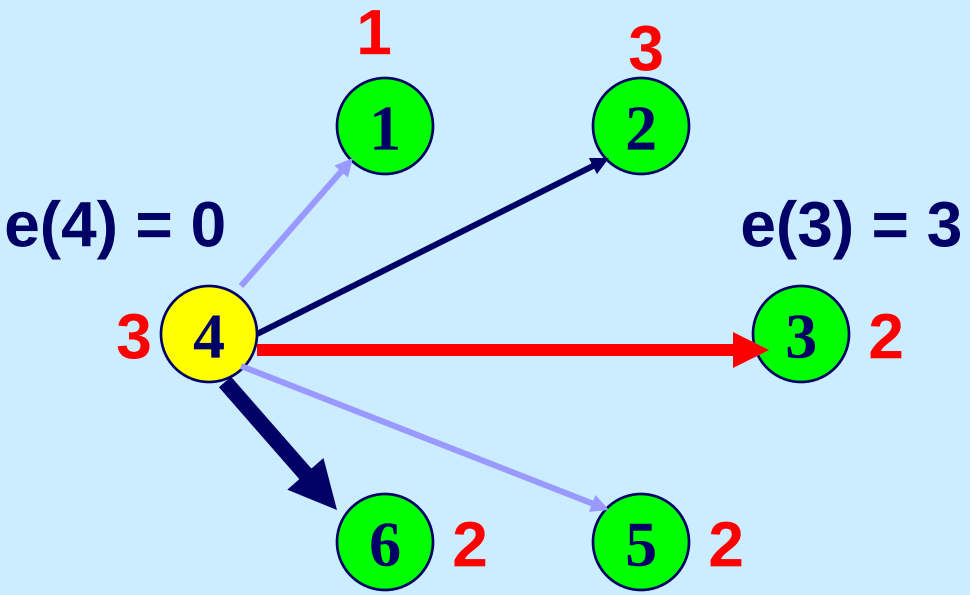
Scan arcs in A(4) one at a time using "Current Arc" till an admissible arc is found.

Push on (4,3)

Pushing on (4,3)

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	2	Yes
4	5	0	No
4	6	2	Yes

Push on (4,3)



Send min (e(4), r_{43}) = 2 units of flow.

Update the residual capacities and excesses.

For the next push from node 4, start with arc (4,3).

Goldberg-Tarjan Preflow Push Algorithm

Procedure Preprocess

x :=0;

compute the exact distance labels d(i) for each node;

x_{sj} := u_{sj} for each arc (s, j) ∈ A(s); d(s) := n;

Algorithm PREFLOW-PUSH;

preprocess;

while there is an active node i do

select an active node i;

push/relabel(i);

convert the max preflow into a max flow

Preflow Push Animation

Note: the "while loop" ends when there are no active nodes; i.e., if e(j) > 0, then d(j) is above a gap.

Preview of Results on Goldberg-Tarjan Preflow Push

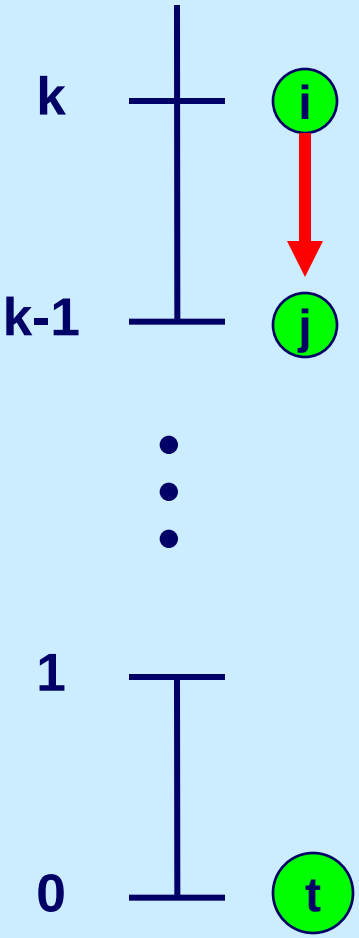
The GT algorithm is superb both in theory and in practice.

Suppose that the flow into t is v* after the "while loop."

We will show the following:

1. **Distance labels stay valid after a push and after a relabel. They never go down in a relabel.**
2. **It is possible to convert the preflow into an s-t flow with a feasible flow out of s (and into t) equal to v*.**
3. **The algorithm determines a cut with**
   - **the algorithm determines a preflow with max flow into t**
   - **the algorithm determines a minimum s-t cut**
4. **The running time of the algorithm is $O(n^2m)$**

Distance labels remain valid



Let x be the flow before the push.

Let x' be the flow after the push.

There is at most one arc in G(x') that is not in G(x), and that arc is (j, i)

Suppose that we push flow in (i, j) when d(i) = k and d(j) = k - 1.

Validity remains satisfied for all arcs of G(x). And it is also satisfied for arc (j, i). That is,

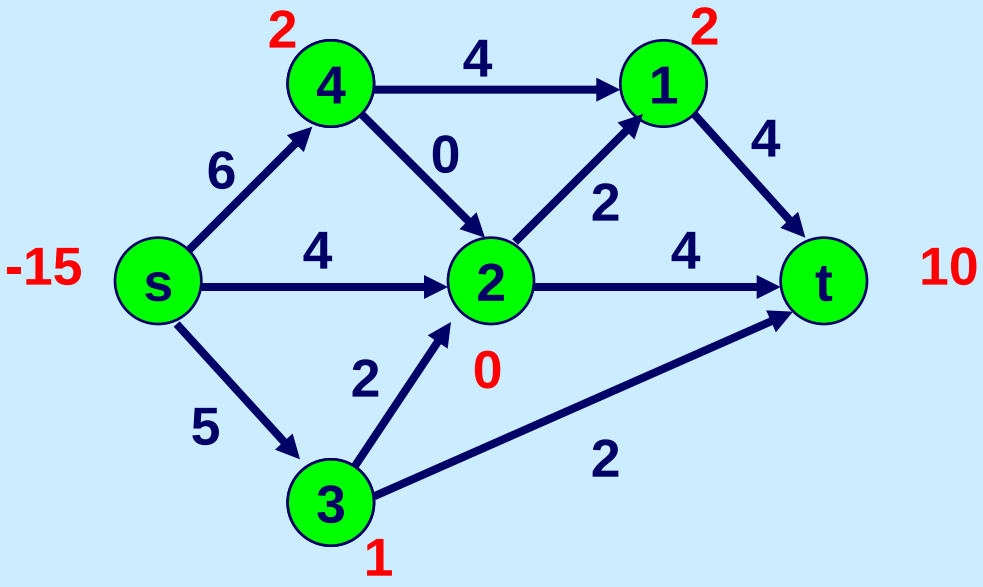
$$d(j) \leq d(i) + 1 = k+1.$$

Before a relabel of node i, the distance labels are valid, but no arc out of i is admissible. So, in a relabel of node i, d(i) must increase.

Converting any preflow into a flow

Let x' be any preflow at some stage of the algorithm.

Express x' using flow decomposition.



Path	Flow
s-4	2
s-3-2-1	2
s-3	1
s-4-1-t	4
s-2-t	4
s-3-t	2

Note: s is the only node with deficit.

The feasible flow is the sum of the flows on paths from s to t.

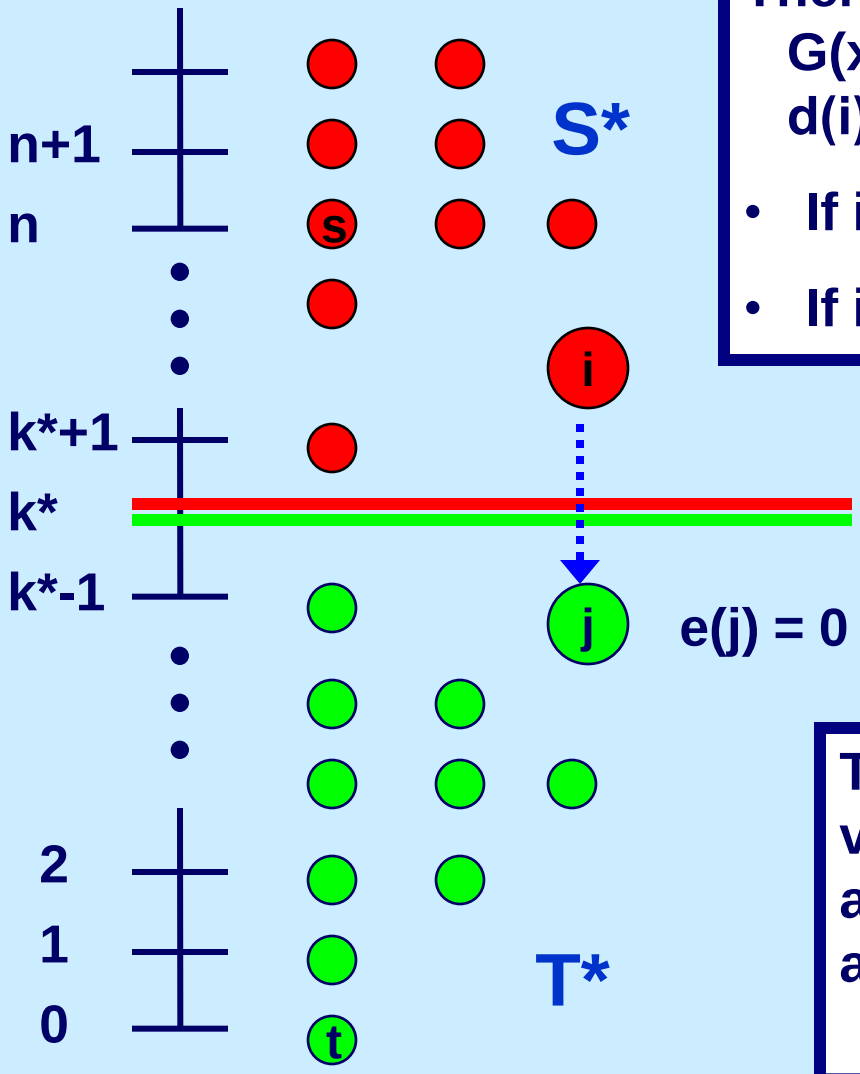
The flow into t is the same for the flow and preflow.

Finding the minimum cut

- Let d*( ) be the distance labels at the end of the algorithm.
- Let k* be the minimum positive value such that there is a gap at level k*.
- Let S* = {j : d*(j) < k*}. Let T* = {j : d(j*) > k*}.

Theorem. (S*, T*) is a minimum capacity cut, and the capacity of the cut is the amount of flow into t.

The minimum cut



There is no arc (i, j) from S* to T* in G(x*) because for each arc (i, j) ∈ G(x*) $d(i) \le d(j) + 1$.

- If $i \in S^*$ and $j \in T^*$, then $x^*_{ij} = u_{ij}$
- If $i \in T^*$ and $j \in S^*$, then $x^*_{ij} = 0$

There is no excess at a node of T* because the algorithm terminated.

The flow v* into t is
v* = CAP(S*, T*) because any flow across the cut must go to t. (Same as in proof of max-flow min-cut).

Mental Break

Something happened to the actors in the first English play in America. What was it?

They were arrested. Acting was considered evil.

What was the first country to give women the vote?

New Zealand, in 1890.

The first phone book ever issued was in New Haven, CT. How many names were in the book?

50. It was published in 1878.

Mental Break

In London, in the 1700s, one could purchase insurance for a particularly catastrophic event. But it was not clear how one could prove the event happened. What was the event?

One could insure against going to Hell.

Bobby Beach survived a barrel ride over Niagara Falls. He died many years later. What was the cause of his death?

He slipped on a banana peal.

Native Americans did not eat turkey at the time of the Pilgrims. Why not?

They thought that killing such a timid bird indicated laziness.

Finiteness and Running Time

Lemma 1. The time to relabel all nodes is $O(nm)$.

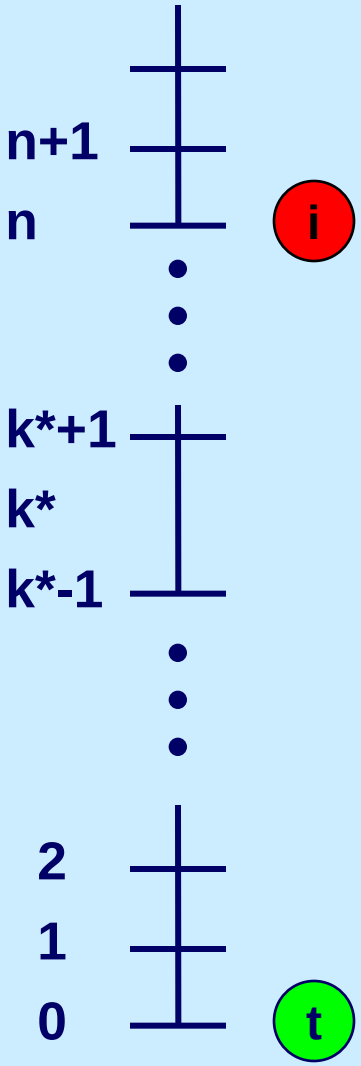
Lemma 2. The number of saturating pushes is $O(nm)$.

Lemma 3. The time spent scanning arcs that are not admissible is $O(nm)$.

Lemma 4. The number of non-saturating pushes is $O(n^2m)$.

Theorem. The preflow push algorithm is finite. Its running time is $O(n^2m)$.

Proof of Lemma 1



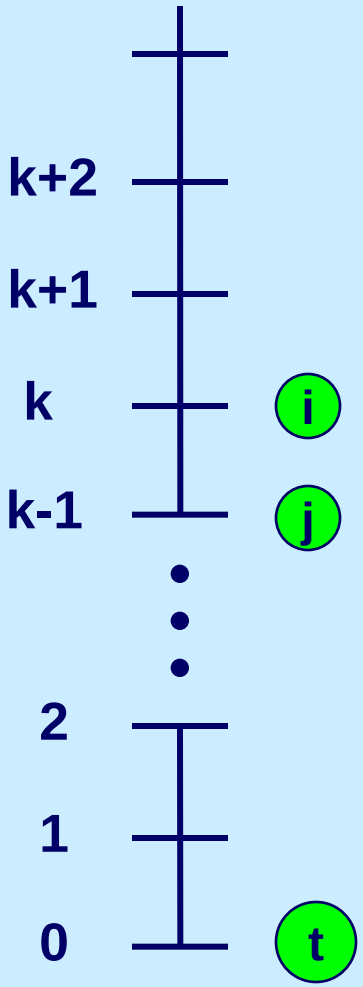
If d(i) ≥ n, then there must be a gap at some distance level less than n because there are at most n-2 remaining nodes.

So, each node is relabeled at most n times.

The time to relabel nodes 1 to n at most once is O(m) since each arc out of node j must be scanned for each j.

Conclusion: the time to relabel each nodes 1 to n at most n times is O(nm)

Proof of Lemma 2



Suppose that (i, j) is saturated when d(i) = k. It cannot be saturated again until d(i) ≥ k+2.

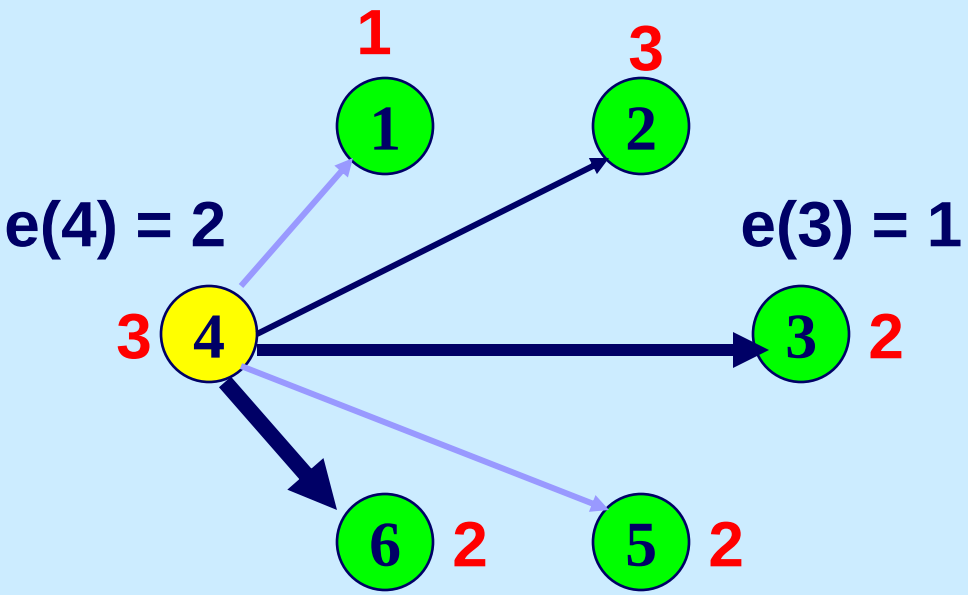
There is no capacity in (i, j) until flow is sent in (j, i). At that point d(j) ≥ k + 1.

But (i, j) cannot be saturated again until it is admissible. Then d(i) = d(j) + 1 ≥ k + 2.

Conclusion: each arc (i, j) is saturated at most n/2 times. The number of saturating pushes is O(nm) and the running time for saturating pushes is O(nm).

Lemma 3 and scanning arcs.

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	4	Yes
4	5	0	No
4	6	2	Yes



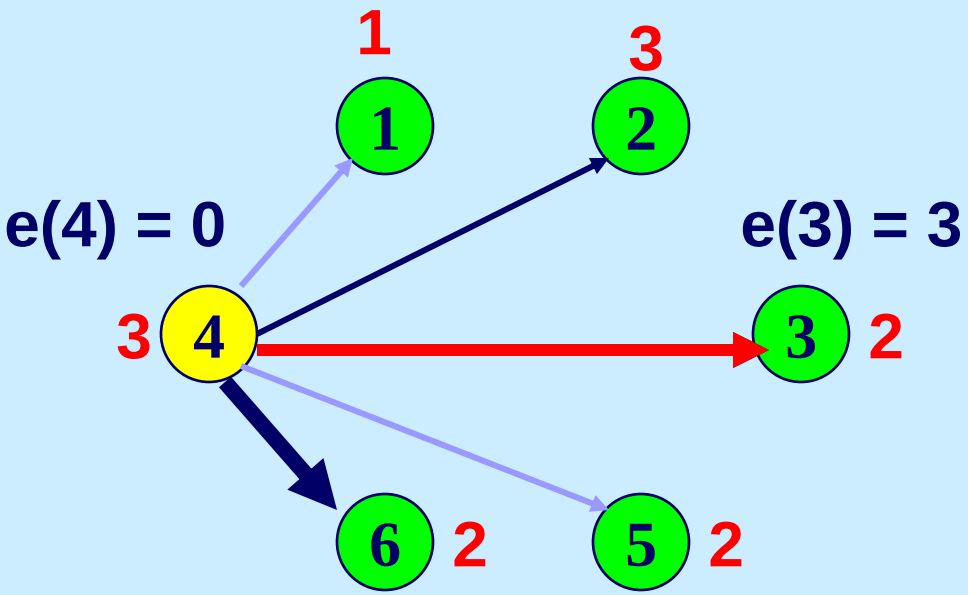
Arcs in A(i) are scanned in order. If arc (i, j) is not admissible, it cannot become admissible until i is relabeled again.

e.g., arc (4, 1) has no capacity. It cannot get capacity until flow is sent in (1, 4) at which point d(1) is at least 4.

e.g., arc (4, 2) has capacity but it is not admissible. It will not become admissible until d(4) increases.

Proof of Lemma 3

Tail	Head	Res. Cap	Admissible ?
4	1	0	No
4	2	1	No
4	3	2	Yes
4	5	0	No
4	6	2	Yes



Each inadmissible arc in A(i) is scanned at most once between relabels.

Each arc (i, j) is scanned at most n/2 times when it is inadmissible.

The running time to scan inadmissible arcs is O(nm).

Running time analysis, so far.

Time for initialization:	O(m)
Time for Sat pushes and updates of r	O(nm)
Time for relabels	O(nm)
Time for scanning inadmissible arcs	O(nm)
Time for nonsat pushes and updates of r	?

We will use a different type of running time analysis using *potential functions* with an analogy to gamblers.

Bounding the number of losing bets

Example. John is a gambler. On each bet, he can lose up to $L dollars and win up to $W. When John loses, he always loses at least $1. Suppose that

- John starts with at most $D
- the number of times that he wins bets is at most n.

What is an upper bound on the number K of times that John loses a bet.

Final amount = Initial Amount + Winnings - Losses

Formula 1: Losses = Initial Amount + Winnings - Final amount

K ≤ Losses = Initial Amount + Winnings - Final Amount.

K ≤ D + nW - 0.

Use Formula 1 to bound the number of bets at which the gambler loses.

Potential Functions for Preflow-Push

Let $\Phi(x)$ be a function that depends on the residual network $G(x)$. We will call Φ a ***potential function***.

For example, $\Phi(x) = \sum_{j \text{ active}} d(j)$.

Let $\Phi_k = \Phi(x_k)$, where x_k is the flow in the residual network immediately prior to the k-th step of the algorithm.

Let $\delta_k = \Phi_{k+1} - \Phi_k$. $d_k( )$ = distance labels prior to the k-th step.

View that Φ_k is the amount that gambler John has prior to the k-th bet. Then δ_k is the win or loss for John at the k-th bet.

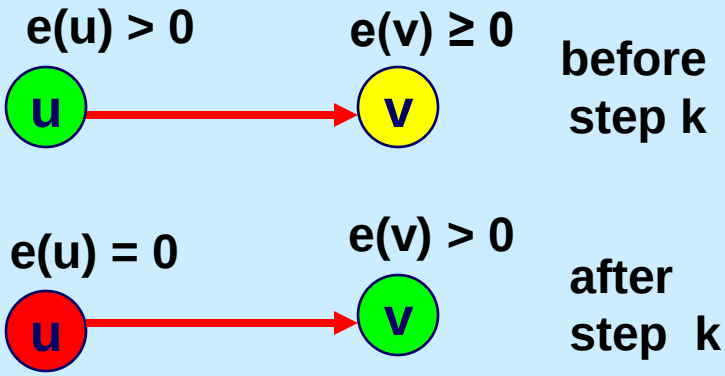
We will use Formula 1 to bound nonsaturating pushes.

Losses in potential = Initial potential + gains in potential - Final potential

Bounds on δ_k

Consider the case that the k-th step is a nonsaturating push in arc (u,v).

$d_{k+1} = d_k$



$\Phi_k = \sum_{j \in \text{Active}(k)} d_k(j)$

$\Phi_{k+1} = \sum_{j \in \text{Active}(k+1)} d_{k+1}(j).$

$\delta_k = \Phi_{k+1} - \Phi_k = d_{k+1}(v) - d_k(u) - (0 \text{ or } d(v))$

$\leq d_k(v) - d_k(u) = -1.$

Let NSAT be the number of nonsaturating pushes.

NSAT ≤ Losses

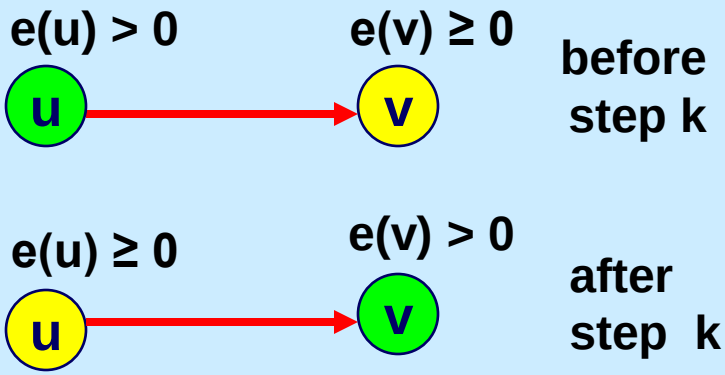
Bounds on δ_k

Consider the case that the k-th step is a saturating push.

$d_{k+1} = d_k$

$\Phi_k = \sum_{j \in \text{Active}(k)} d_k(j)$

$\Phi_{k+1} = \sum_{j \in \text{Active}(k+1)} d_{k+1}(j).$



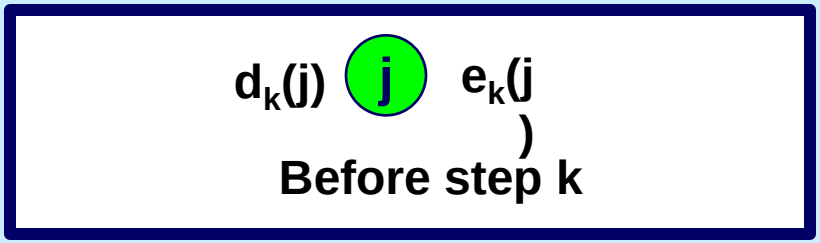
$$\delta_k = \Phi_{k+1} - \Phi_k = d_{k+1}(v) + [d_{k+1}(u) \text{ or } 0] - d_k(u) - [d_k(v) \text{ or } 0]$$
$$\leq d_{k+1}(v) + d_{k+1}(u) - d_k(u) = d_{k+1}(v) \leq n.$$

Let SAT be the number of saturating pushes.

The contribution of saturating pushes to gains in potential is at most $n\,(SAT) \leq n^2 m$.

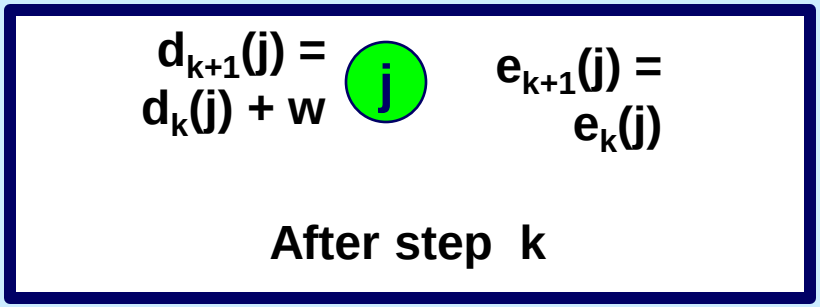
Bounds on δ_k

Consider the case that the k-th step is an increase of d(v) by w.



$$\Phi_k = \sum_{j \in \text{Active}(k)} d_k(j)$$

$$\Phi_{k+1} = \sum_{j \in \text{Active}(k+1)} d_{k+1}(j).$$



$$\delta_k = \Phi_{k+1} - \Phi_k = d_{k+1}(v) - d_k(v) = w.$$

But each node can increase its distance by at most n.

The contribution of distance label increases to gains in potential is at most n per node and at most n^2 in all.

Bounding NSAT

Losses = Initial Potential + Gains - Final Potential

$\Phi_k = \sum_{j \in \text{Active}(k)} d_k(j)$

NSAT ≤ Losses
Initial Potential ≤ n^2
Gains ≤ $n^2m + n^2$
Final Potential ≥ 0

NSAT ≤ $n^2 + n^2m + n^2 - 0 = O(n^2m)$

Research notes on preflow push

- Pushing from the active node with the largest distance label leads to $O(n^2 m^{.5})$ nonsat pushes.
- A very efficient data structure called dynamic trees reduces the running time to $O(nm \log n^2/m)$. Goldberg-Tarjan (1986)
- The "excess scaling technique" of Ahuja and Orlin (1989) reduced the running time to $O(nm + n^2 \log U)$.
- Ahuja, Orlin, and Tarjan (1989): further very small improvements.
- Goldberg and Rao (1998). An even more efficient algorithm for max flows.

15.082J / 6.855J / ESD.78J Network Optimization

Fall 2010