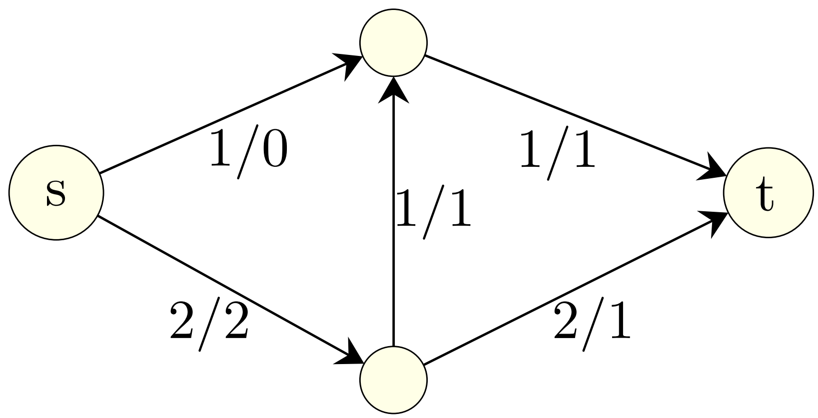
The Maximum Flow Problem

nput:

- a directed graph $G = (V, E)$, source node $s \in V$, sink node $t \in V$
- edge capacities $cap : E \longrightarrow I\!R_{\geq 0}$



oal:

- compute a flow of maximal value, i.e.,
- a function $f : E \longrightarrow I\!R_{\geq 0}$ satisfying the capacity constraints and the flow conservation constraints

(1) $$0 \leq f(e) \leq cap(e) \quad \text{for every edge } e \in E$$

(2) $$\sum_{e;target(e)=v} f(e) = \sum_{e;source(e)=v} f(e) \quad \text{for every node } v \in V \setminus \{s, t\}$$

- and maximizing the net flow into t.

Further Reading

he main sources for the lectures are the books [8, 1, 9]. The original ublications on the preflow-push algorithm are [6, 2]. [5] describes the currently est flow algorithm for integral capacities. Papers by the instructors are [3, 7, 4].

R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

R.K. Ahuja and J.B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operation Research*, 37:748–759, 1989.

J. Cheriyan, T. Hagerup, and K. Mehlhorn. An $o(n^3)$-time maximum flow algorithm. *SIAM Journal of Computing*, 25(6):1144–1170, 1996.

J. Cheriyan and K. Mehlhorn. An analysis of the highest-level selection rule in the preflow-push max-flow algorithm. *IPL*, 69:239–242, 1999. `www.mpi-sb.mpg.de/~mehlhorn/ftp/maxflow.ps`.

A.V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *JACM*, 45(5), 1998.

A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35:921–940, 1988.

T. Hagerup, P. Sanders, and J. Träff. An implementation of the binary blocking flow algorithm. In *Proceedings of the 2nd Workshop on Algorithm Engineering (WAE'98)*, pages 143–154. Max-Planck-Institut für Informatik, 1998.

K. Mehlhorn. *Data Structures and Algorithms*. Springer, 1984.

K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999. 1018 pages.

Some Notation and First Properties

- the excess of a node v: $excess(v) = \sum_{e; target(e)=v} f(e) - \sum_{e; source(e)=v} f(e)$
- in a flow: all nodes except s and t have excess zero.
- the value of a flow $= val(f) = excess(t)$

learly: the net flow into t is equal to the next flow out of s.

emma 1 $excess(t) = -excess(s)$

he proof is short and illustrates an important technique

$$excess(s) + excess(t) = \sum_{v \in V} excess(v) = 0$$

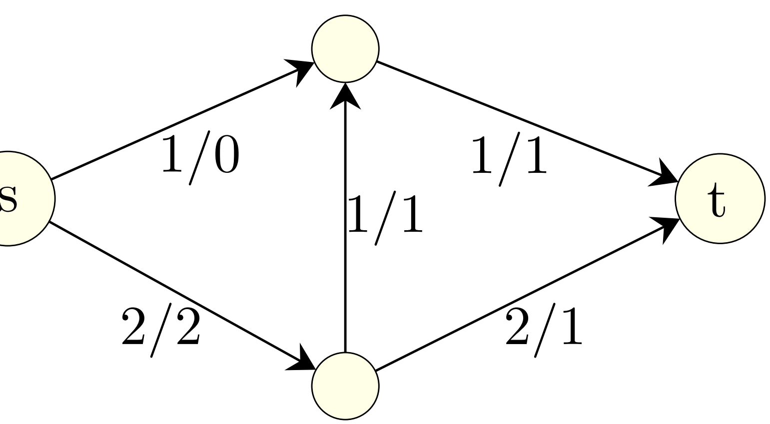
- the first equality holds since $excess(v) = 0$ for $v \neq s, t$.
- the second equality holds since the flow across any edge $e = (v, w)$ appears twice in this sum
  - positively in $excess(w)$ and negatively in $excess(v)$

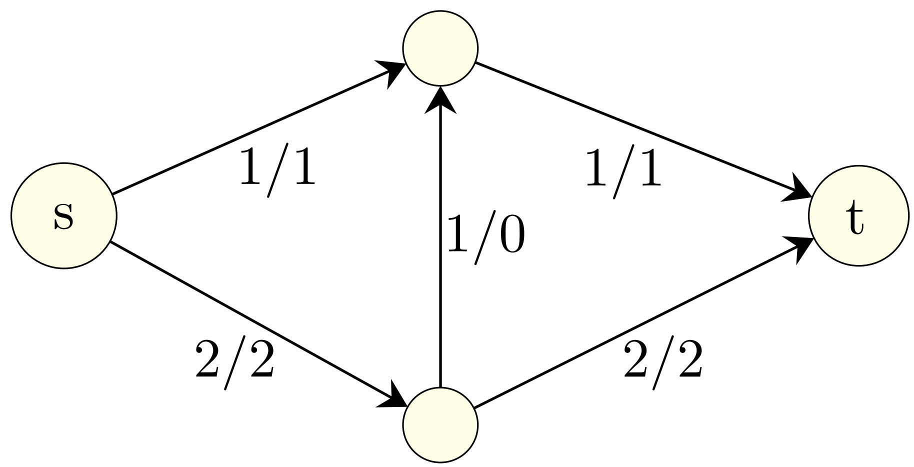
Cuts

- a subset S of the nodes is called a cut. Let $T = V \setminus S$
- S is called an (s,t)-cut if $s \in S$ and $t \in T$.
- the *capacity* of a cut is the total capacity of the edges leaving the cut,

$$cap(S) = \sum_{e \in E \cap (S \times T)} cap(e).$$

- a cut S is called *saturated* if $f(e) = cap(e)$ for all $e \in E \cap (S \times T)$ and $f(e) = 0$ for all $e \in E \cap (T \times S)$.





Lemma 2 *For any flow f and any (s,t)-cut*

- $val(f) \le cap(S)$.
- *if S is saturated,* $val(f) = cap(S)$.

Proof: We have

$$\begin{aligned} val(f) &= -excess(s) = -\sum_{u \in S} excess(u) \\ &= \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in E \cap (T \times S)} f(e) \le \sum_{e \in E \cap (S \times T)} cap(e) \\ &= cap(S). \end{aligned}$$

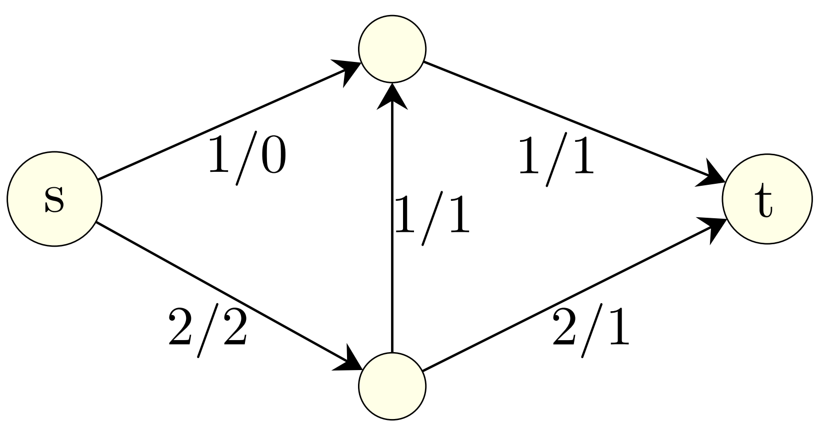
For a saturated cut, the inequality is an equality. ∎

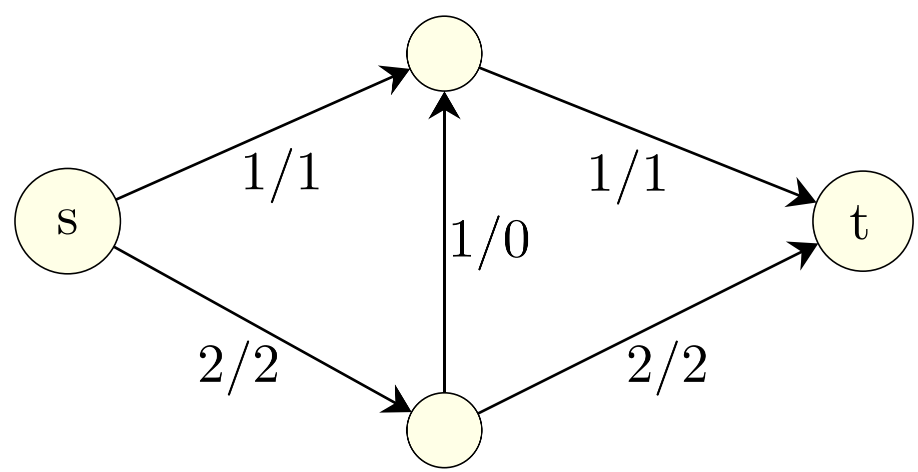
Remarks:

- A saturated cut proves the optimality of a flow.
- For every maximal flow there is a saturated cut proving its optimality ($\Longrightarrow$)

The Residual Network

- let f be a flow in $G = (V, E)$
- the *residual network* G_f captures possible changes to f
  - same node set as G
  - for every edge $e = (v, w)$ up to two edges e' and e'' in G_f
    * if $cap(e) < f(e)$, we have an edge $e' = (v, w) \in G_f$
      residual capacity $r(e') = cap(e) - f(e)$.
    * if $f(e) > 0$, we have an edge $e'' = (w, v) \in G_f$
      residual capacity $r(e'') = f(e)$.
- two flows and the corresponding residual networks

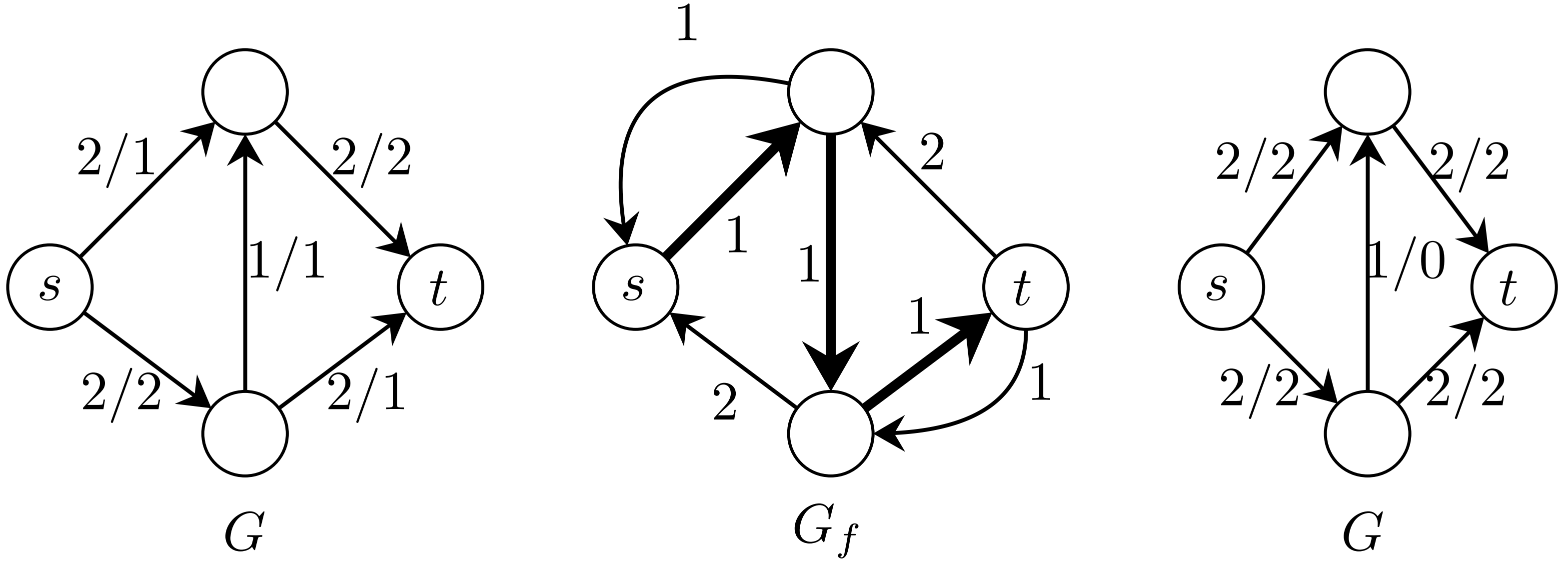




heorem 1 (Maximum Flows and the Residual Graph) *Let f be an $,t)$-flow, let G_f be the residual network with respect to f, and let S be the set of odes that are reachable from s in G_f.*

) *If $t \in S$ then f is not maximum.*

) *If $t \notin S$ then S is a saturated cut and f is maximum.*



An illustration of part a)

Maximum Flows and the Residual Graph: Part a

t is reachable from s in G_f, f is not maximal

- Let p be any simple path from s to t in G_f
- Let δ be the minimum residual capacity of any edge of p. Then $\delta > 0$.
- We construct a flow f' of value $val(f) + \delta$. Let (see Figure on preceding slide)

$$f'(e) = \begin{cases} f(e) + \delta & \text{if } e' \text{ is in } p \\ f(e) - \delta & \text{if } e'' \text{ is in } p \\ f(e) & \text{if neither } e' \text{ nor } e'' \text{ belongs to } p. \end{cases}$$

- f' is a flow and $val(f') = val(f) + \delta$.

a path in G_f: $s \longrightarrow v_1 \longrightarrow v_2 \longrightarrow v_3 \longrightarrow v_4 \longrightarrow v_5 \longrightarrow t$

the corresponding path in G:

Maximum Flows and the Residual Graph: Part b

t cannot be reached from s in G_f, f is maximal.

- Let S be the set of nodes reachable from s and let $T = V \setminus S$.
- There is no edge (v, w) in G_f with $v \in S$ and $w \in T$.
- Hence
  - $f(e) = cap(e)$ for any e with $e \in E \cap (S \times T)$ and
  - $f(e) = 0$ for any e with $e \in E \cap (T \times S)$
- Thus S is saturated and f is maximal.

G_f G

Max-Flow-Min-Cut Theorem

heorem 2 (Max-Flow-Min-Cut Theorem)

$$\max\{\mathit{val}(f)\ ;\ f\ \mathit{is\ a\ flow}\} = \min\{\mathit{cap}(S)\ ;\ S\ \mathit{is\ an}\ (s,t)\text{-}\mathit{cut}\}$$

roof:

- $\leq$ is the content of Lemma 2, part (a).
- let f be a maximum flow
  - then there is no path from s to t in G_f and
  - the set S of nodes reachable from s form a saturated cut
  - hence $\mathit{val}(f) = \mathit{cap}(S)$ by Lemma 2, part (b).

∎

theorem of the form above is called a *duality theorem.*

The Ford-Fulkerson Algorithm

- start with the zero flow, i.e., $f(e) = 0$ for all e.
- construct the residual network G_f
- check whether t is reachable from s.
  - if not, stop
  - if yes, increase flow along an augmenting path, and iterate

- each iteration takes time $O(n + m)$
- if capacities are arbitrary reals, the algorithm may run forever
- it does well in the case of integer capacities

Integrality Theorem

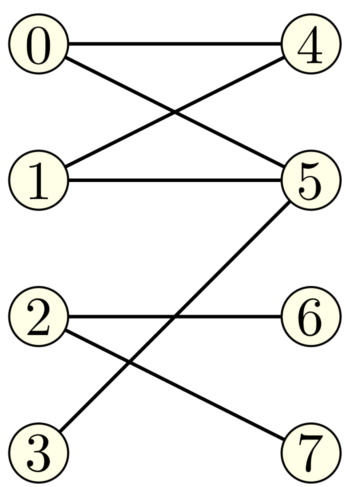
- assume integral capacities, say in $[0\,..\,C]$
- let $v^* =$ value of the maximum flow $\leq deg(s) \cdot C \leq nC$
- **Claim:** all flows constructed are integral (and hence final flow is integral)

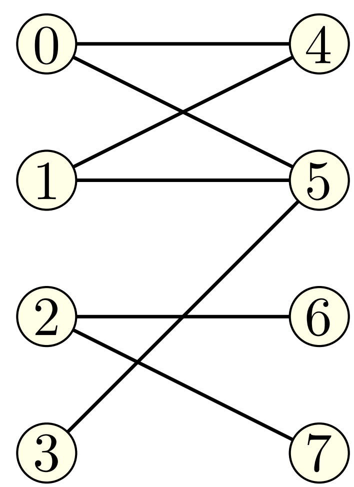
  Proof: We use induction on the number of iterations.
  - the initial flow (= all-zero-flow) is integral.
  - if current flow is integral, residual capacities are integral and hence next flow is integral
- every augmentation increases flow value by at least one
- running time is $O((n+m)v^*)$; this is good if v^* is small

heorem 3 *If edge capacities are integral, there exists an integral maximal flow. loreover, the algorithm of Ford and Fulkerson finds it in time $O((n+m)v^*)$, here v^* is the value of the maximum flow.*

Bipartite Matching

- given a bipartite graph $G = (A \cup B, E)$, find a maximal matching
- matching M, a subset of the edges, no two of which share an endpoint
- reduces easily to network flow
  - add a source s, edges (s, a) for $a \in A$, capacity one
  - add a sink t, edges (b, t) for $b \in B$, capacity one
  - direct edges in G from A to B, capacity $+\infty$
  - integral flows correspond to matchings
  - Ford-Fulkerson takes time $O(nm)$ since $v^* \leq n$, can be improved to $O(\sqrt{n}m)$





The Theorem of Hall

heorem 4 *A bipartite graph $G = (A \cup B, E)$ has an A-perfect matching (= a atching of size $|A|$) iff for every subset $A' \subset A$, $|\Gamma(A')| \geq |A'|$, where $\Gamma(A')$ is e set of neighbors of the nodes in A'.*

ndition is clearly necessary; we need to show sufficiency

- assume that there is no A-perfect matching
- then flow in the graph defined on preceding slide is less than $|A|$
- and hence minimum cut has capacity less than $|A|$.
- consider a minimum (s, t)-cut (S, T).
- let $A' = A \cap S$, $A'' = A \cap T$, $B' = B \cap S$, $B'' = B \cap T$

- no (!!!) edge from A' to B'' and hence $\Gamma(A') \subseteq B'$
- flow $= |B'| + |A''| < |A| = |A'| + |A''|$
- thus $|B'| < |A'|$

A Theoretical Improvement for Integral Capacities

- modify Ford-Fulkerson by always augmenting along a flow of maximal residual capacity
- **Theorem 5** *running time becomes:* $T = O((m + m\log\lceil v^*/m\rceil)m\log m)$
- i.e., v^*-term in time bound is essentially replaced by $m\log v^*\log m$; this is good for large v^* (namely, if $v^* \geq m\log v^*\log m$)
- practical value is minor (since we will see even better methods later), but proof method is interesting
- **Lemma 3** *Max-res-cap-path can be determined in time* $O(m\log m)$.
- **Lemma 4** $O(m + m\log\lceil v^*/m\rceil)$ *augmentations suffice*

emma 5 *Max-res-cap-path can be determined in time* $O(m \log m)$.

- sort the edges of G_f in decreasing order of residual capacity
- let e_1, e_2, ... , $e_{m'}$ be the sorted list of edges
- want to find the minimal i such that $\{e_1, \ldots, e_i\}$ contains a path from s to t
- for fixed i we can test existence of path in time $O(n + m)$
- determine i by binary search in $O(\log m)$ rounds.

emma 6 *$O(m + m\log\lceil v^*/m\rceil)$ augmentations suffice*

- a flow can be decomposed into at most m paths
  - start with a maximal flow f
  - repeatedly construct a path from s to t, saturate it, and subtract from f
- augmentation along max-res-cap-path increases flow by at least $1/m$ of dist to v^*
- let g_i be the diff between v^* and the flow value after the i-th iteration
- $g_0 = v^*$
- if $g_i > 0$, $g_{i+1} \leq g_i - \max(1, g_i/m) \leq \min(g_i - 1, (1 - 1/m)g_i)$
- $g_i \leq (\frac{m-1}{m})^i g_0$ and hence $g_i \leq m$ if i is such that $(\frac{m-1}{m})^i g_0 \leq m$.
- this is the case if $i \geq \log_{m/(m-1)}(v^*/m) = \frac{\log(v^*/m)}{\log m/(m-1)}$
- $\log(m/(m-1)) = \log(1 + 1/(m-1)) \geq 1/(2m)$ for $m \geq 10$
- number of iterations $\leq m + 2m\log(v^*/m)$

Dinic's Algorithm (1970), General Capacities

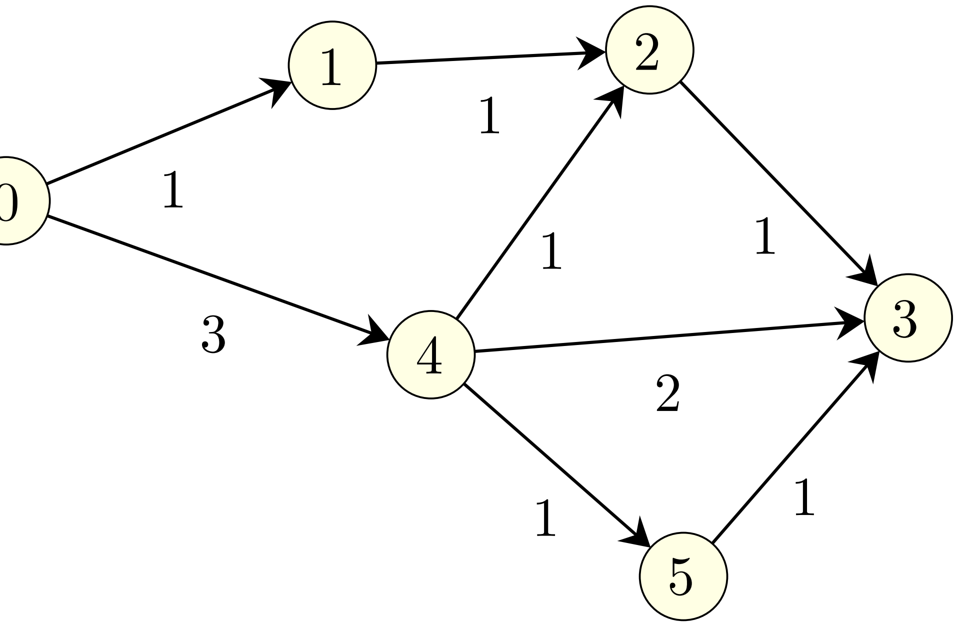
- start with the zero flow f
- construct the layered subgraph L_f of G_f
- if t is not reachable from s, stop
- construct a blocking flow f_b in L_f and augment to f, repeat

- in L_f nodes are on layers according to their BFS-distance from s and only edges going from layer i to layer $i+1$ are retained
- L_f is constructed in time $O(m)$ by BFS
- blocking flow: a flow which saturates one edge on every path from s to t

- the number of rounds is at most n, since the depth of L_f grows in each round (without proof, but see analysis of # of saturating pushes in preflow-push alg)
- a blocking flow can be computed in time $O(nm)$
- $T = O(n^2m)$

An Example Run of Dinic's Algorithm

will illustrate the sequence of residual graphs and residual level graphs.



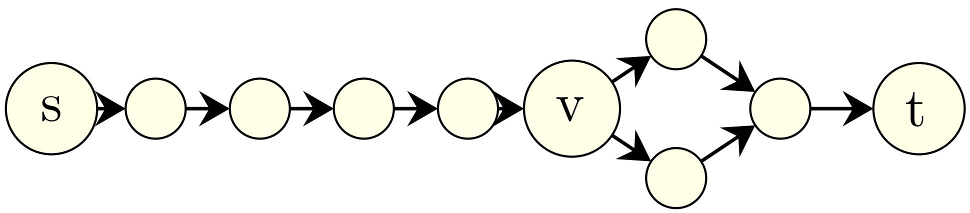
The Computation of Blocking Flows

- maintain a path p starting at s, initially $p = \epsilon$, let $v = \mathit{tail}(p)$
- if $v = t$, increase f_b by saturating p, remove saturated edges, set p to the empty path (**breakthrough**)
- if $v = s$ and v has no outgoing edge, stop
- if $v \neq t$ and v has an outgoing edge, **extend** p by one edge
- if $v \neq t$ and v has no outgoing edge, **retreat** by removing last edge from p.

- running time is $\#_{extends} + \#_{retreats} + n \cdot \#_{breakthroughs}$
- $\#_{breakthroughs} \leq m$, since at least one edge is saturated
- $\#_{retreats} \leq m$, since one edge is removed
- $\#_{extends} \leq \#_{retreats} + n \cdot \#_{breakthroughs}$, since a retreat cancels one extend and a breakthrough cancels n extends
- running time is $O(m + nm) = O(nm)$

Preflow-Push Algorithms

- f is a preflow (Karzanov (74)): $\mathit{excess}(v) \geq 0$ for all $v \neq s, t$
- residual network with respect to a preflow is defined as for flows
- Idea: preflows give additional flexibility



- manipulate a preflow by operation $\mathit{push}(e, \delta)$
  - Preconditions:
    - e is residual, i.e., $e = (v, w) \in E_f$
    - v has excess, i.e, $\mathit{excess}(v) > 0$
    - δ is feasible, i.e, $\delta \leq \min(\mathit{excess}(v), \mathit{res}_f(e))$
  - Action: push δ units of flow from v to w
    - decrease $\mathit{excess}(v)$ by δ, increase $\mathit{excess}(w)$ by δ, modify f and adapt E_f (remove e if it now saturated, add its reversal)
- **Question:** Which push to make?
- **Answer:** push towards t, but what does this mean?

The Level Function (Goldberg/Tarjan)

- a simple and highly effective notion of "towards t"
- arrange the nodes on levels, $d(v)$ = level number of $v \in \mathbb{N}$
- at all times: $d(t) = 0$, $d(s) = n$
- call an edge $e = (v, w)$ *eligible* iff $e \in E_f$ and $d(w) < d(v)$
- and only push across eligible edges, i.e., from higher to lower level

Question: What to do when v has positive excess but no outgoing eligible edge?

Answer: lift it up, i.e., increase $d(v)$ by one (relabel v)

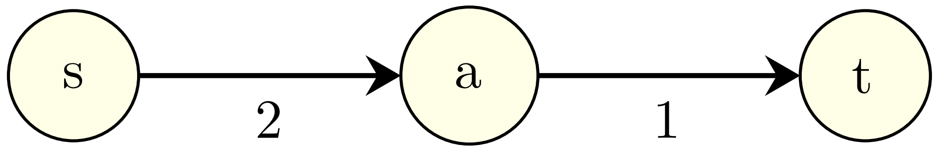
The Generic Push-Relabel Algorithm

t $f(e) = cap(e)$ for all edges with $source(e) = s$;
t $f(e) = 0$ for all other edges;
t $d(s) = n$ and $d(v) = 0$ for all other nodes;

hile there is a node $v \neq s, t$ with positive excess
 let v be any such node node;
 if there is an eligible edge $e = (v, w)$ in G_f
 { push δ across e for some $\delta \leq \min(excess(v), res\_cap(e))$; }
 else
 { relabel v; }

- obvious choice for δ: $\delta = \min(excess(v), res\_cap(e))$
- push with $\delta = res\_cap(e)$ — *saturating push*
- push with $\delta < res\_cap(e)$ — *non-saturating push*
- need to bound the number of relabels, the number of pushes, need to explain how to find nodes with positive excess and eligible edges

A Sample Run



nd here comes the sequence of residual graphs (residual capacities are shown)

No Steep Edges

n edge $e = (v, w) \in G_f$ is called *steep* if $d(w) < d(v) - 1$, i.e., if it reaches down
y two or more levels.

emma 7 *The algorithm maintains a preflow and does not generate steep edges.*
he nodes s and t stay on levels 0 and n, respectively.

roof:

- the algorithm maintains a preflow by the restriction on δ
- after initialization: edges in G_f go sidewards or upwards
- when v is relabeled, no edge in G_f out of v goes down. After relabeling, edges out of v go down at most one level.
- a push across an edge $e = (v, w) \in G_f$ may add the edge (w, v) to G_f. This edge goes up.
- s and t are never relabeled

∎

The Maximum Level Stays Below $2n$

emma 8 *If v is active then there is a path from v to s in G_f. No distance label ver reaches $2n$.*

roof: Let S be the set of nodes that are reachable from v in G_f and let $= V \setminus S$. Then

$$\sum_{u \in S} excess(u) = \sum_{e \in E \cap (T \times S)} f(e) - \sum_{e \in E \cap (S \times T)} f(e),$$

here is no edge $(v, w) \in G_f$ with $v \in S$ and $w \notin S$. Thus, $f(e) = 0$ for every $\in E \cap (T \times S)$. We conclude $\sum_{u \in S} excess(v) \leq 0$.

nce s is the only node whose excess may be negative and since $excess(v) > 0$ we ust have $s \in S$.

ssume that a node v is moved to level $2n$. Since only active nodes are relabeled is implies the existence of a path (and hence simple path) in G_f from a node on vel $2n$ to s (which is on level n). Such a path must contain a steep edge. ∎

Partial Correctness

heorem 6 *When the algorithm terminates, it terminates with a maximum flow.*

roof: When the algorithm terminates, all nodes different from s and t have
xcess zero and hence the algorithm terminates with a flow. Call it f.

n G_f there can be no path from s to t since any such path must contain a steep
dge (since s is on level n, t is on level 0). Thus, f is a maximum flow by the
ax-flow-min-cut theorem. ∎

n order to prove termination, we bound the number of relabels, the number of
turating pushes and the number of non-saturating pushes.

he former two quantities are easily bounded.

Ve have to work harder for the number of non-saturating pushes.

On the Number of Relabels and Saturating Pushes

emma 9 *There are at most $2n^2$ relabels and at most nm saturating pushes.*

roof:

- no distance label ever reaches $2n$.
- therefore, each node is relabeled at most $2n$ times
- the number of relabels is therefore at most $2n^2$.

- a saturating push across an edge $e = (v, w) \in G_f$ removes e from G_f.
- **Claim:** v has to be relabeled at least twice before the next push across e and hence there can be at most n saturating pushes across any edge.
  - only a push across e^{rev} can again add e to G_f.
  - for this to happen w must be lifted by two levels, ...

∎

On the Number of Non-Saturating Pushes: Scaling

```
* scaling push-relabel algorithm (Ahuja-Orlin) for integral capacities */
t f(e) = cap(e) for all edges with source(e) = s and f(e) = 0 for all other edges;
t d(s) = n and d(v) = 0 for all other nodes;
t Δ = 2^⌈log max_e cap(e)⌉;

hile ( Δ > 1 )
 while there is a node v ≠ s,t with excess(v) ≥ Δ/2
 { let v be the lowest (!!!) such node;
    if there is an eligible edge e = (v,w) in G_f
    { push δ across e for δ = min(Δ/2, res_cap(e)); }
    else
    { relabel v; }
 }
 Δ = Δ/2;
```

- excesses are bounded by Δ, i.e., at all times and for all $v \neq t$: $excess(v) \leq \Delta$
- a non-saturating push moves $\Delta/2$ units of flow

On the Number of Non-Sat Pushes in Ahuja-Orlin

emma 10 *The number of non-saturating pushes is at most* $4n^2 + 4n^2\lceil \log U \rceil$, *here* U *is the largest capacity*

Ve use a potential function argument (let $V' = V \setminus \{s,t\}$)

$$\Phi = \sum_{v \in V'} d(v) \frac{excess(v)}{\Delta}$$

- $\Phi \geq 0$ always, $\Phi = 0$ initially
- total decrease of $\Phi \leq$ total increase of Φ
- a relabel increases Φ by at most one
- every push decreases Φ
- a non-saturating push decreases Φ by $1/2$
- a change of Δ increases Φ by at most $2n^2$
- Δ is changed $\lceil \log U \rceil$ times
- $(1/2)\#_{non\ sat\ pushes} \leq$ total decrease $\leq$ total increase $\leq 2n^2 + 2n^2\lceil \log U \rceil$

Maintaining the Set of High Excess Nodes

- we have $2n$ buckets, one for each level
- the i-the bucket B_i contains all nodes v with $d(v) = i$ and $excess(v) \geq \Delta/2$
- at the beginning of a Δ-phase: initialize buckets by a scan over all nodes
- maintain a index i^*, buckets B_i with $i < i^*$ are empty
- search for a high excess node: advance i^* until a non-empty bucket is found
- pushes may require to decrease i^* by one
- summary: total number of changes of $i^* \leq 2n +$ number of pushes

The Search for Eligible Edges

- every node v stores the list of all edges (out and in) incident to it
- every node stores its height
- every edge stores its capacity and the current flow across it
- an out-edge $e = (v, w)$ is eligible for pushing out of v iff $f(e) < cap(e)$ and $d(w) < d(v)$
- an in-edge $e = (w, v)$ is eligible for pushing out of v iff $f(e) > 0$ and $d(w) < d(v)$
- **L 11** *An edge can become eligible for pushing out of v only by a relabel of v*
  - consider a non-eligible out-edge $e = (v, w)$, i.e., either $d(w) \geq d(v)$ or $f(e) = cap(e)$.
  - the latter condition can only be changed by a push across the reversal of e.
  - such a push is only possible if $d(w) > d(v)$. Hence e cannot be eligible after the push.

The Search for Eligible Edges

- every node maintains a pointer into its edge list (= the current edge)
- **invariant:** no edge to the left of the current edge is eligible
- in order to search for an eligible edge for pushing out of v, v advances its current edge pointer until
  - either an eligible edge is found
  - or the end of the list is reached. Then v is relabeled and the current edge pointer is reset to the beginning of the list
- correctness follows from Lemma on preceding slide
- time is $O(deg(v))$ between relabels of v and hence
- total time required to search for eligible edges $= 2n \cdot \sum_v deg(v) = O(nm)$

On the Number of Non-Sat Pushes in the Generic Algorithm

- pushes are made as large as possible, i.e., $\Delta = \min(excess(v), res\_cap(e))$
- (persistence) when an active node v is selected, pushes out of v are performed until either v becomes inactive (because of a non-saturating push out of v) or until there are no eligible edges out of v anymore. In the latter case v is relabeled.
- we study three rules for the selection of active nodes

  ***Arbitrary*:** an arbitrary active node is selected.
  $\#_{non\ sat\ pushes} = O(n^2 m)$, Goldberg and Tarjan

  ***FIFO*:** the active nodes are kept in a queue and the first node in the queue is always selected. When a node is relabeled or activated the node is added to the rear of the queue, $\#_{non\ sat\ pushes} = O(n^3)$, Goldberg

  ***Highest-Level*:** an active node on the highest level, i.e., with maximal d-value is selected, $\#_{non\ sat\ pushes} = O(n^2\sqrt{m})$, Cheriyan and Maheshwari
- in all three cases: running time of preflow-push is $O(nm + \#_{non\ sat\ pushes})$

The Arbitrary Rule

emma 12 *When the Arbitrary-rule is used, the number of non-saturating ushes is* $O(n^2m)$.

roof:

$$\Phi = \sum_{v \in V'; v \text{ is active}} d(v).$$

- $\Phi \geq 0$ always, and $\Phi = 0$ initially.
- a non-saturating push decreases Φ by at least one, since it deactivates the source of the push (may activate the target)
- a relabeling increases Φ by one.
- a saturating push increases Φ by at most $2n$, since it may activate the target
- total increase of $\Phi \leq n^2 + nm2n = n^2(1 + 2m)$
- $\#_{non\ sat\ pushes} \leq$ total increase of Φ

∎

The FIFO Rule

- active nodes are in a queue, head of queue is selected for pushing/relabeling
- relabeled and activated nodes are added to the rear of the queue
- we split the execution into phases
- first phase starts at the beginning of the execution
- a phase ends when all nodes that were active at the beginning of the phase have been selected from the queue
- each node is selected at most once in each phase: $\#_{non\ sat\ pushes} \leq n \cdot \#_{phases}$

emma 13 *When the FIFO-rule is used, the number of phases is $O(n^2)$.*

roof: Use $\Phi = \max\{d(v)\ ;\ v \text{ is active }\}$

- $\Phi \geq 0$ always, and $\Phi = 0$ initially.
- a phase containing no relabel operation decreases Φ by at least one, since all nodes on the highest level become inactive.
- a phase containing a relabel operation increases Φ by at most one, since a relabel increases the highest level by at most one.

∎

emma 14 *When the Highest-Level-rule is used,* $\#_{non\ sat\ pushes} = O(n^2\sqrt{m})$.

/arning: Proof in Ahuja/Magnanti/Orlin is wrong, proof here Cheriyan/M

- let $K = \sqrt{m}$. For a node v, let $d'(v) = |\{w; d(w) \leq d(v)\}|/K$.
- potential function $\Phi = \sum_{v;v \text{ is active}} d'(v)$.
- execution is split into phases
- phase = all pushes between two consecutive changes of $d^* = \max\{d(v)\ ;\ v \text{ is active }\}$
- phase is *expensive* if it contains more than K non-sat pushes, *cheap* otherwise.

/e show:

.) The number of phases is at most $4n^2$.

2) The number of non-saturating pushes in cheap phases is at most $4n^2K$.

3) $\Phi \geq 0$ always, and $\Phi \leq n^2/K$ initially.

4) A relabeling or a sat push increases Φ by at most n/K.

5) A non-saturating push does not increase Φ.

6) An expensive phase with $Q \geq K$ non-sat pushes decreases Φ by at least Q.

(1) The number of phases is at most $4n^2$.

(2) The number of non-saturating pushes in cheap phases is at most $4n^2K$.

(3) $\Phi \geq 0$ always, and $\Phi \leq n^2/K$ initially.

(4) A relabeling or a sat push increases Φ by at most n/K.

(5) A non-saturating push does not increase Φ.

(6) An expensive phase with $Q \geq K$ non-sat pushes decreases Φ by at least Q.

- Suppose that we have shown (1) to (6).
- (4) and (5) imply total increase of $\Phi \leq (2n^2 + mn)n/K$
- above + (3): total decrease can be at most this number plus n^2/K
- $\#_{non\ sat\ pushes\ in\ expensive\ phases} \leq (2n^3 + n^2 + mn^2)/K$.
-

above + (2)	$\#_{non\ sat\ pushes} \leq (2n^3 + n^2 + mn^2)/K + 4n^2K$
since $n \leq m$:	$\#_{non\ sat\ pushes} \leq 4mn^2/K + 4n^2K = 4n^2(m/K + K)$
$K = \sqrt{m}$:	$\#_{non\ sat\ pushes} \leq 8n^2\sqrt{m}$.

Proving (1) to (5)

1) The number of phases is at most $4n^2$:

we have $d^* = 0$ initially, $d^* \geq 0$ always, and only relabels increase d^*. Thus, d^* is increased at most $2n^2$ times, decreased no more than this, and hence changed at most $4n^2$ times.

2) The number of non-saturating pushes in cheap phases is at most $4n^2 K$:

follows immediately from (1) and the definition of a cheap phase.

3) $\Phi \geq 0$ always, and $\Phi \leq n^2/K$ initially: obvious

4) A relabeling or a sat push increases Φ by at most n/K:

follows from the observation that $d'(v) \leq n/K$ for all v and at all times. Also a relabel of v cannot increase any of the other labels $d'(w)$.

5) A non-saturating push does not increase Φ:

observe that a non-sat push across an edge (v, u) deactivates v, activates u (if it is not already active), and that $d'(u) \leq d'(v)$.

Proving (6)

6) An expensive phase with $Q \geq K$ non-sat pushes decreases Φ by at least Q:

- consider an expensive phase containing $Q \geq K$ non-sat pushes.
- d^* is constant during a phase and hence all Q non-saturating pushes must be out of nodes at level d^*.
- The phase is finished either because level d^* becomes empty or because a node is moved from level d^* to level $d^* + 1$.
- In either case, we conclude that level d^* contains $Q \geq K$ nodes at all times during the phase.
- Thus, each non-saturating push in the phase decreases Φ by at least one (since $d'(u) \leq d'(v) - 1$ for a push from v to u).

Heuristic Improvements, see LEDAbook, pages 465 – 487

- at the start of the alg, all edges out of s are saturated
- some of the flow pushed into the network will make it to t
- part of the flow must be routed back to s
- this requires to lift (some, many) nodes above level n
- thus running time is $\Omega(n^2)$ even if total number of pushes is small
- heuristic improvements attempt to lift nodes faster that the standard relabeling procedure

ggressive local relabeling: when a node is relabeled, continue to relabel it until there is an eligible edge out of it, i.e.,

$$\text{set } d(v) \text{ to } 1 + \min\{d(w)\ ;\ (v,w) \in G_f\}$$

aggressive local relabeling has cost $O(1)$, it may increase $d(v)$ by more than one.

Heuristic Improvements, Continued

obal relabeling: after $O(m)$ edge inspections, update the dist-values of all nodes by setting

$$d(v) = \begin{cases} \mu(v,t) & \text{if there is a path from } v \text{ to } t \text{ in } G_f \\ n + \mu(v,s) & \text{if there is a path from } v \text{ to } s \text{ in } G_f \text{ but no} \\ & \text{path from } v \text{ to } t \text{ in } G_f \\ 2n-1 & \text{otherwise} \end{cases}$$

Here $\mu(v,t)$ and $\mu(v,s)$ denote the lengths (= number of edges) of the shortest paths from v to t, respectively s, in G_f.

global relabeling has cost $O(m)$.

ap heuristic: when a level i, $1 \le i < n$, becomes empty (because we lift the last node on this level to a higher level),
lift all nodes in levels $i+1$ to $n-1$ to level n.

gap heuristic has cost proportional to the number of nodes moved to level n.

Experimental Findings

Gen	Rule	BASIC	HL	LRH	GRH	GAP	LEDA
rand	FF	5.84	6.02	4.75	0.07	0.07	—
		33.32	33.88	26.63	0.16	0.17	—
	HL	6.12	6.3	4.97	0.41	0.11	0.07
		27.03	27.61	22.22	1.14	0.22	0.16
	MF	5.36	5.51	4.57	0.06	0.07	—
		26.35	27.16	23.65	0.19	0.16	—

nd = random graphs, we used $n = 1000$ and $n = 2000$, $m = 3n$.

F = first-in-first-out selection rule

L = highest level selection rule

F = modified FF-rule (relabels reinsert at front, pushes insert at end)

ASIC = generic preflow push

L = nodes above level n are treated slightly differently (not explained in lectures)

RH = aggressive local relabeling

LH = global relabeling heuristic

AP = gap heuristic

EDA = improved storage organization

Asymptotics of our Implementations

Gen	Rule	GRH			GAP			LEDA		
rand	FF	0.16	0.41	1.16	0.15	0.42	1.05	—	—	—
	HL	1.47	4.67	18.81	0.23	0.57	1.38	0.16	0.45	1.09
	MF	0.17	0.36	1.06	0.14	0.37	0.92	—	—	—
CG1	FF	3.6	16.06	69.3	3.62	16.97	71.29	—	—	—
	HL	4.27	20.4	77.5	4.6	20.54	80.99	2.64	12.13	48.52
	MF	3.55	15.97	68.45	3.66	16.5	70.23	—	—	—
CG2	FF	6.8	29.12	125.3	7.04	29.5	127.6	—	—	—
	HL	0.33	0.65	1.36	0.26	0.52	1.05	0.15	0.3	0.63
	MF	3.86	15.96	68.42	3.9	16.14	70.07	—	—	—
AMO	FF	0.12	0.22	0.48	0.11	0.24	0.49	—	—	—
	HL	0.25	0.48	0.99	0.24	0.48	0.99	0.12	0.24	0.52
	MF	0.11	0.24	0.5	0.11	0.24	0.48	—	—	—

G1, CG2, and AMO are problem generators, see LEDAbook for details.

or each generator we ran the cases $n = 5000 \cdot 2^i$ for $i = 0$, 1, and 2.

or the random graph generator we used $m = 3n$.