Algorithms
Professor John Reif

ALG 5.3

Flow Algorithms:

(a) Max-flow, min-cut Theorem
(b) Augmenting Paths
(c) 0 - 1 flow
(d) Vertex Connectivity
(e) Planar Flow

Main Reading Selections:
CLR, Chapter 27

Auxillary Reading Selections:
Handout: "Network Flows"
Combinatorial Optimization, by Eugene Lawler, Holt, Rinehart, Winston, 1976.

Network Definition:

digraph $G = (V, E)$

distinguished vertices:

source $s \in V$

sink $t \in V$

edge capacities: $c: E \to R^+$

Reverse Edges: $E^R =$ reverse of edges in E

$u \longrightarrow v$ $\quad$ $u \longleftarrow v$

(u, v) $\quad$ $(u, v)^R$ $\quad$ $(u, v)^R = (v, u)$

Flow $f: (E \cup E^R) \to R^+$

$\forall e \in E$ $\quad$ (1) $f(e) = -f(e^R)$

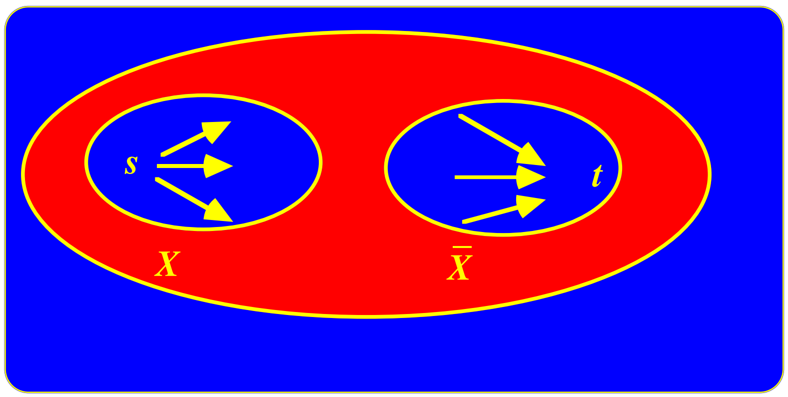
(2) $f(e) \leq c(e)$

$\forall v \in V - \{s, t\}$ $\quad$ (3) $\sum_{(v,u) \in E} f(v, u) = 0$

Flow f

Value $(f) = \sum_{v \in V} f(s, v)$

= sum of flow from source s



cut $X, \overline{X}$ is partition of V

where $s \in X,\ t \in \overline{X}$

$$f(X, \overline{X}) = \sum_{\substack{v \in X \\ u \in \overline{X}}} f(v, u)$$

Lemma: The flow across any cut $X, \overline{X}$ is equal to the $value(f)$.

proof: $f(X, \overline{X}) = \sum_{\substack{v \in X \\ u \in \overline{X}}} f(u, v) = \sum_{\substack{v \in X \\ w \in \overline{X}}} f(v, w)$

$$= \sum_{v, w \in X} f(v, w) = value(f) - 0 = value(f).$$

Q.E.D.

residual capacity of edge e:

$$res(e) = c(e) - f(e)$$

residual graph R: use modified capacities

$$c'(e) = res(e) \text{ for } res(e) > 0$$

augmenting path p for flow f is path in R from s to t

$$\underline{res(p)} = \min_{e \in p}(res(e))$$

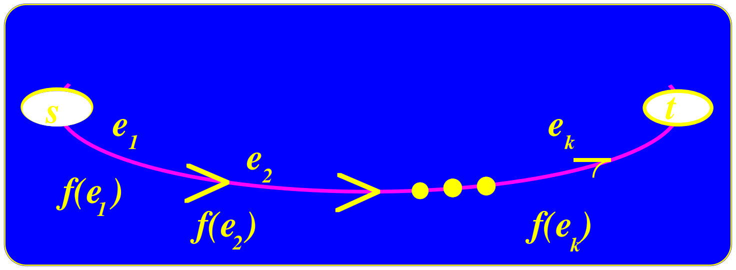
Lemma: R has max flow value $value(f^*) - value(f)$, where f^* is the max flow of G.

proof: If f' is flow in R, then $f + f'$ is flow of G. Also, $f' = f^* - f$ is a flow in R.

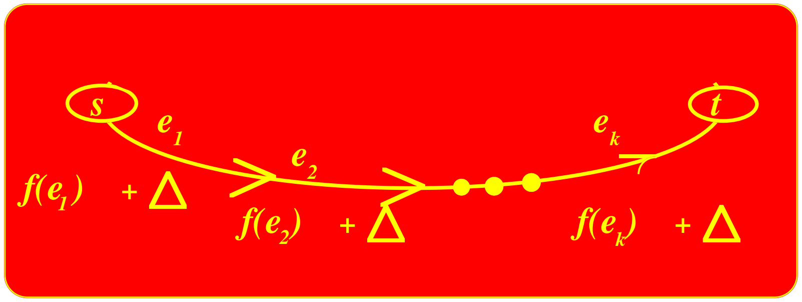
Q.E.D.

Flow f

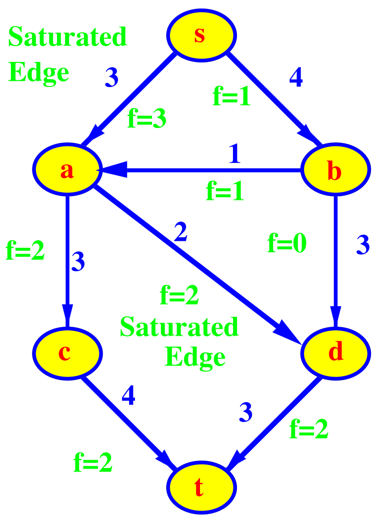
on path $p = (e_1, e_2, \ldots, e_k)$

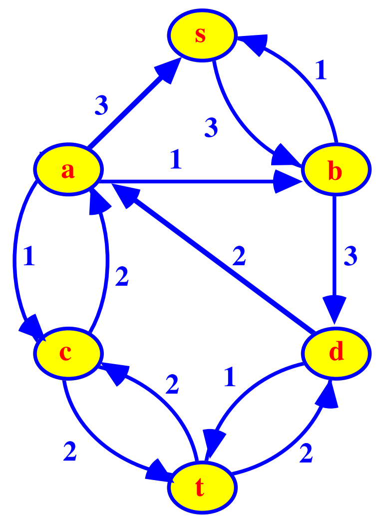


residual $res(p) = \Delta = \min_{e \in p}(c(e) - f(e))$



gives Augmented flow $f + res(p)$





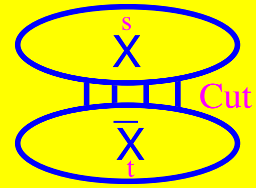
Augmenting Path (s,b,d,a,c,t)

Network with Flow **Residual Network**

min cut: cut of minimum capacity

max flow: $\max(value(f))$

f is flow



Ford - Fulkerson©s:

Theorem: The max flow f is equal to the min cut $X, \overline{X}$.

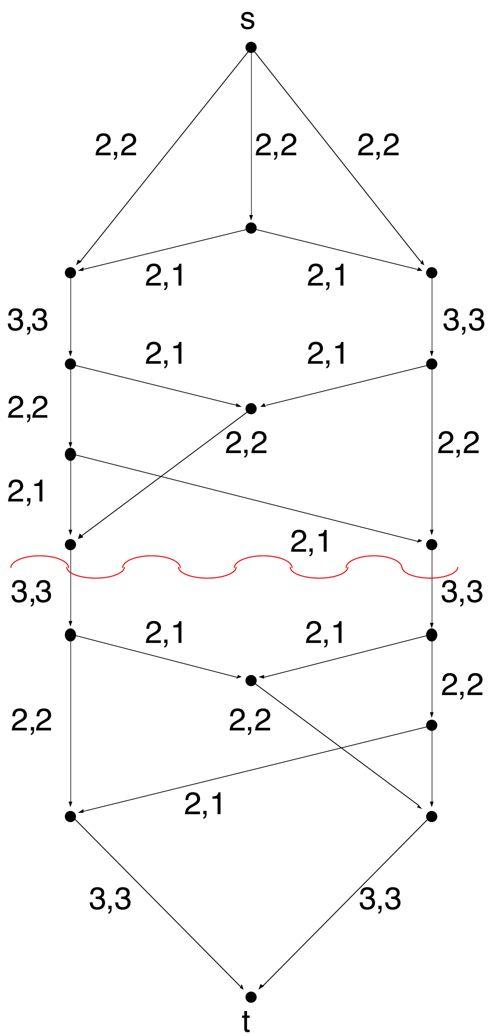
proof: (1) If f is max flow, then there can be no augmenting path from s to t. Let X = vertices in V, reachable from s in residual graph R.

$\overline{X} = V - X$

$$Value(f) = \sum_{\substack{u \in X \\ v \in \overline{X}}} f(u,v) = \sum_{\substack{u \in X \\ v \in \overline{X}}} c(u,v) = c(X, \overline{X})$$

(2) Clearly, f has value at most $c(X, \overline{X})$ for any cut $X, \overline{X}$.

Q.E.D.



Max Flow = Min Cut = 6

Edges Labeled (Capacity, Flow)

<u>Lemma</u>: At most $|E|$ flow augmentations are required to construct max. flow.

<u>proof</u>: Suppose f^* is max flow in G.
Let G^* be subgraph of G with pos. flow.

$i \leftarrow 0$

<u>while</u> $\exists$ path from s to t **<u>do</u>**

- $i \leftarrow i+1$
- find a path p_i from s to t in G^*
- let $\Delta_i = \min_{e \in p_i} f^*(e)$
- **<u>for</u>** all $e \in p$ **<u>do</u>** $f^*(e) \leftarrow f^*(e) - \Delta_i$
    - **<u>if</u>** $f^*(e) \le 0$, **<u>then</u>** delete e from G^*
    - **<u>od</u>**
- **<u>od</u>**

<u>Note</u>: Deletes at least on edge per step!

Definitions

given flow f

saturated edge e has $f(e)=c(e)$

blocking flow f every path from s to t

has saturated edge

(so can not augment flow!)

Idea: Re-route flow

Level Graph L subgraph of R

level (v) = length of shortest path from s to v in R

L contains only edges $(v,u) \in R$ s.t.

level $(u) =$ level $(v)+1$

Note

L gives shortest augmenting paths

Construct L in $O(|V|+|E|)$ time by

Breadth First Search of R

Dinic© Flow Algorithm

Input: network $G=(V,E),s,t$

capacities $c_i:\left(E \cup E^R\right) \rightarrow R^+$

Initialization: $\forall e,\ f(e) \leftarrow 0$

Loop:

[1] Construct level graph L for f by Breadth First Search.

[2] By augmentations, find blocking flow f' in L from f.

[3] $\forall e,\ f(e) \leftarrow f(e)+f'(e)$

[4] If t is not in level graph,

then return f

else go to [1]

Theorem

Dinic's Algorithm halts after $|V|$ blocking steps.

Proof

Suppose f is current flow with

R = residual graph (currently)

level(v) = min length path from s to v in R

R' = new residual graph

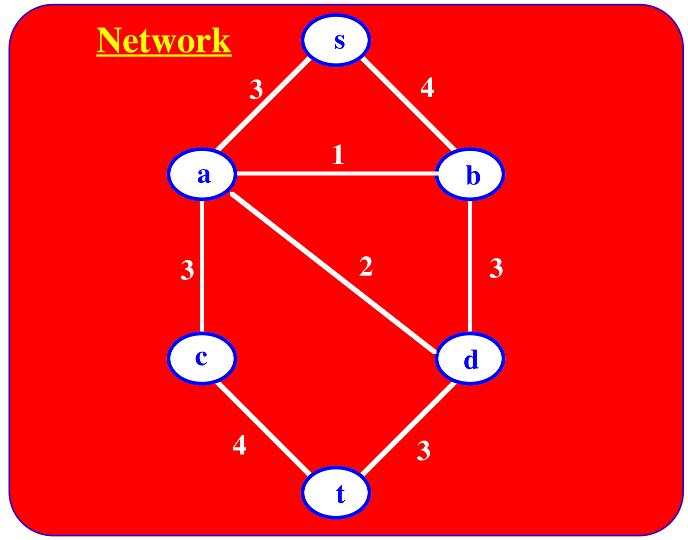
level'(v) = min length of path s to v in R'

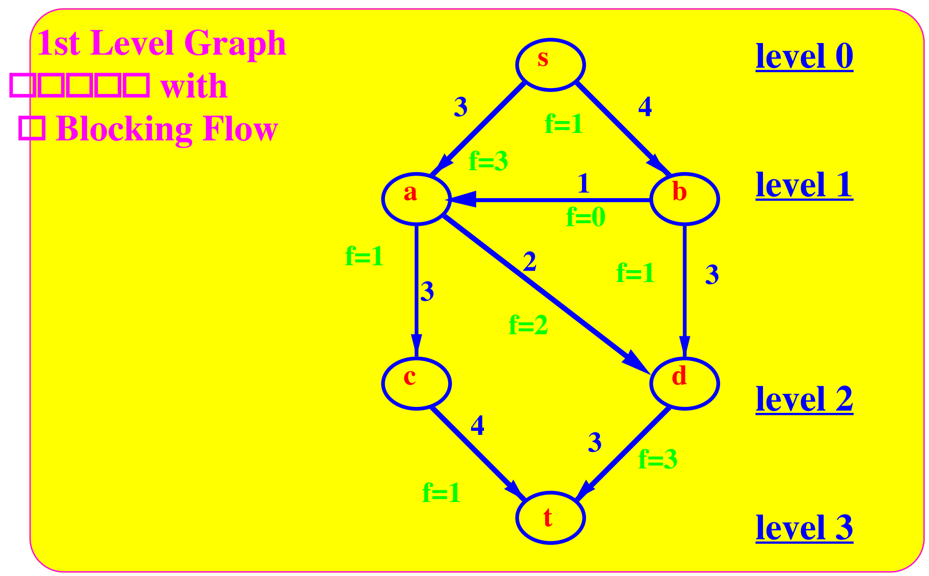
Claim $\text{level}'(t) > \text{level}(t)$

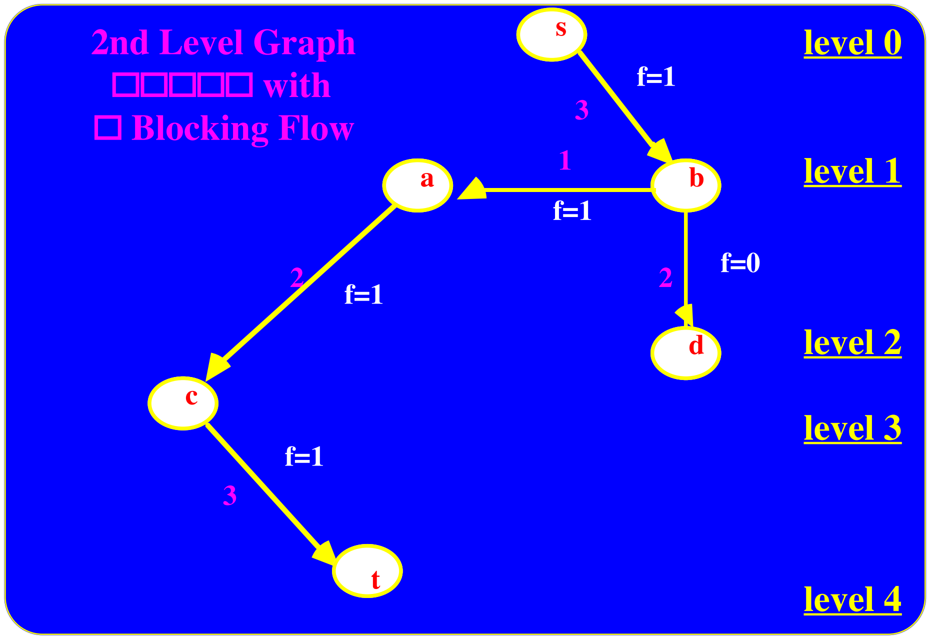
Proof (by contradiction)

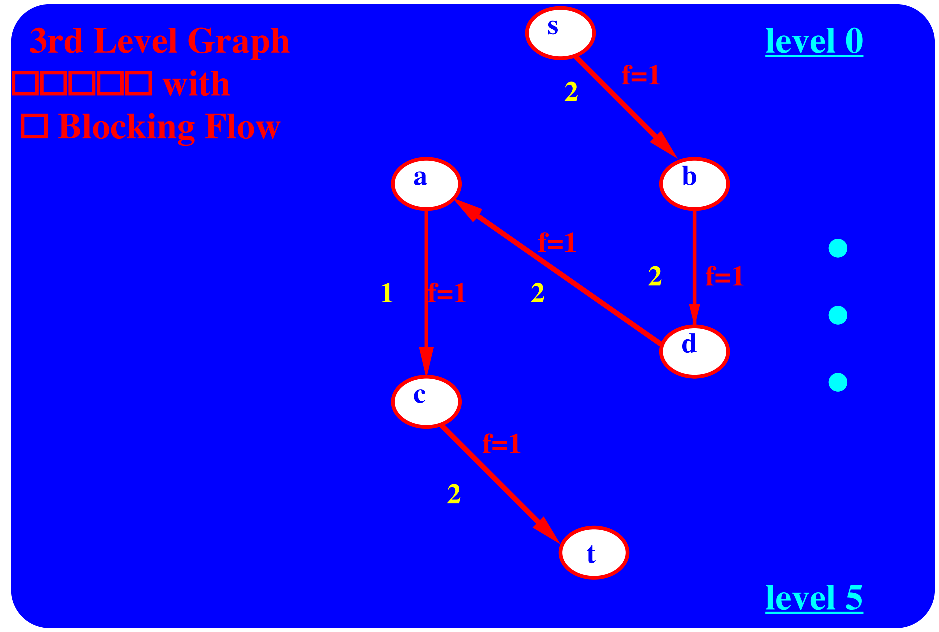
If level(t) = level'(t), then level(w) = level(v) + 1 for every edge $(v,w) \in L$. This contradicts the fact that $\geq$ one edge is saturated (on the blocking flow) on any path p in L.

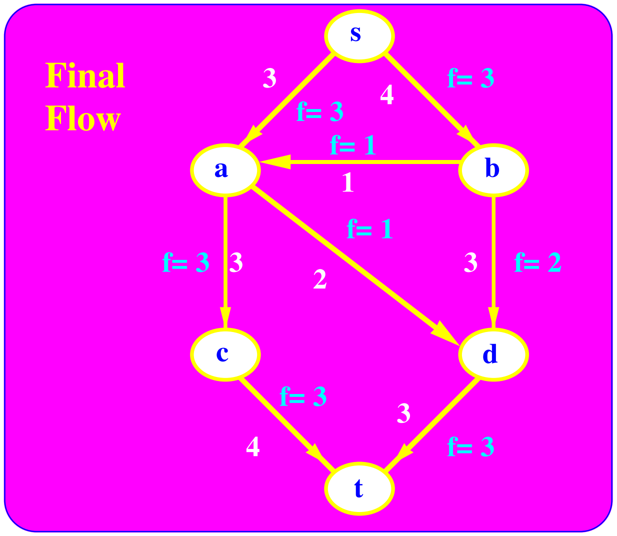
Q.E.D., Hence n steps suffice for the algorithm






2nd Level Graph
□□□□□ with
□ Blocking Flow
level 0
level 1
level 2
level 3
level 4
s
a
b
c
d
t
f=1
3
1
f=1
2
f=1
2
f=0
f=1
3


3rd Level Graph
□□□□□ with
□ Blocking Flow
level 0
level 5
s
a
b
c
d
t
f=1
2
f=1
1
f=1
2
2
f=1
f=1
2



Finding a Blocking Flow
(Karzanov)

Preflow j:

(1) Satisfies capacities' constraints

(2) May have unbalanced verticies

where $\Delta f(u) = \sum_{u \in v} f(u,v) > 0$

Wave method:

- begin with blocking preflow f
  (saturates on edge on every path *s* to *t*
- balance vertices so $\Delta f(v) = 0$ to get blocking flow

To balance blocked vertex v:

Repeat (until Δ f (v) =0) do

choose edge (u,v) with f(u,v) > 0

decrease f(u,v) by min (f(uv), Δ f(v))

or

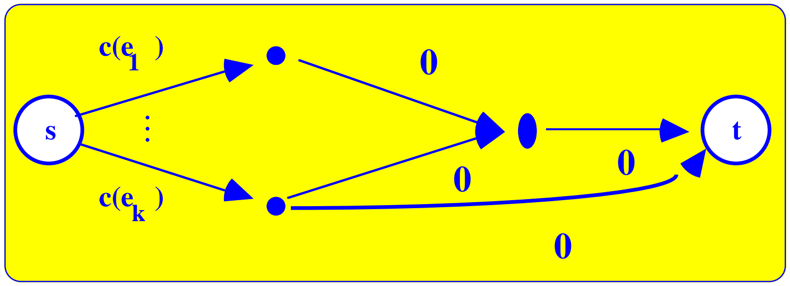
To attempt to <u>balance unblocked vertex</u> v:

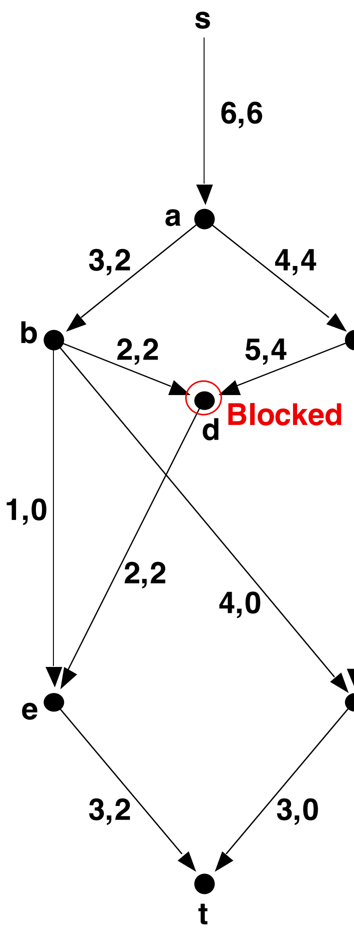
<u>Repeat</u> (until $\Delta f(v)=0$, or there is not an unsaturated edge (v,w) where w is unblocked).

<u>do</u> choose some such edge (v,w) and decrease $f(v,w)$ by $\min(C(v,w)-f(v,w), \Delta f(v))$.

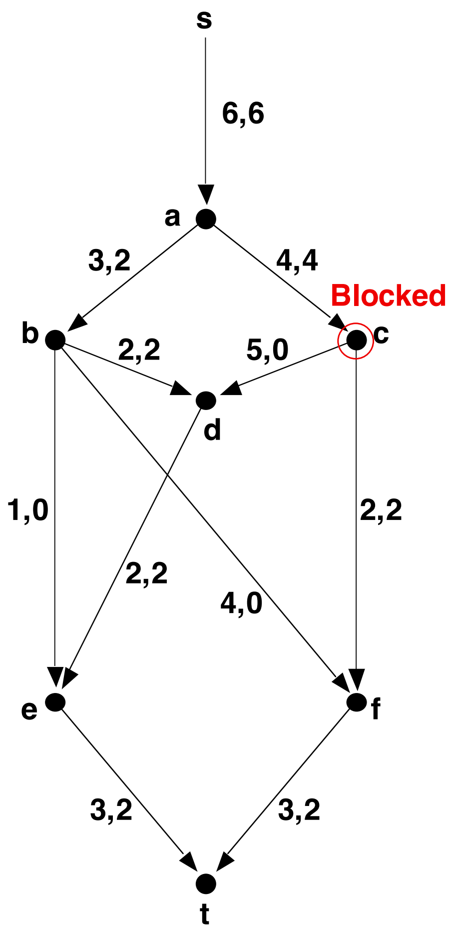
<u>Wave Algorithm for Blocking Flow</u>

<u>Initialize</u>: with preflow that saturates every edge out of s and otherwise 0.



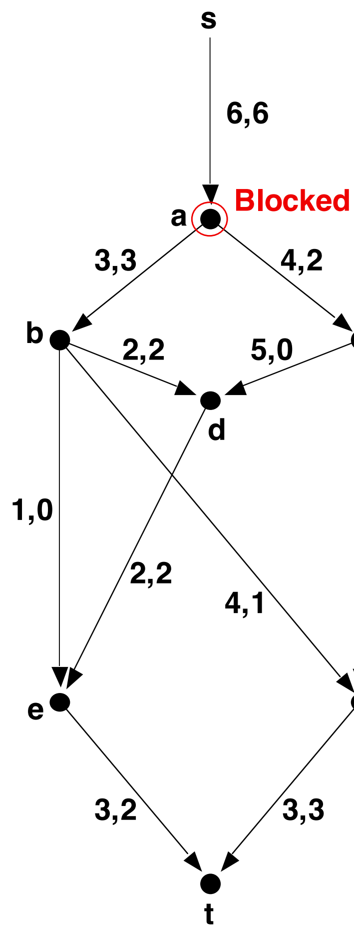


Increased Flow

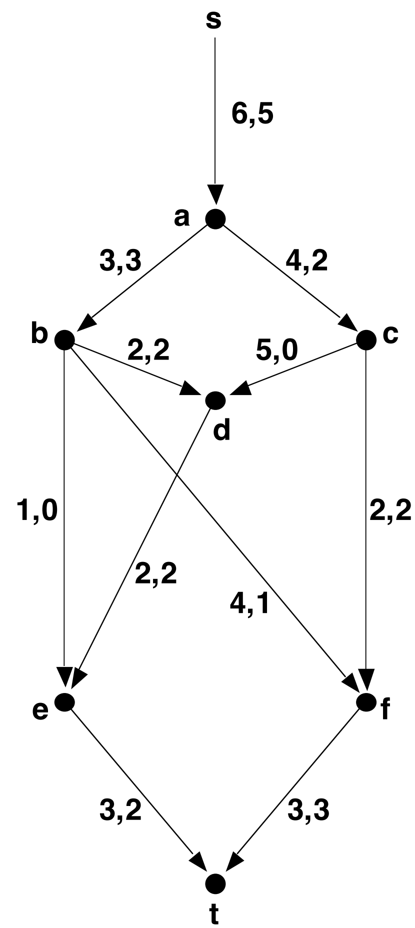


Decreased Flow Balanced *d*

Edges Labeled (Capacity, Flow)



Decreased Flow Balanced *c*



Decreased Flow All Balanced

Edges Labeled (Capacity, Flow)

set s blocked, and set $V-\{s\}$ all unblocked.

Repeat until there are no unbalanced vertices do

Increase flow: Scan all vertices between t,s in topological order, balancing every vertex v that is unbalanced and unblocked. (If balancing fails, make v blocked.)

Decrease flow: Scan vertices in reverse topological order, balancing each vertex that is unbalanced and blocked.

Theorem: Wave Algorithm computes a blocking flow in $O(n^2)$ time (and hence a max flow in $O(n^3)$ time).

Proof (use invariants)**:**

(1) If v blocked $\Rightarrow$ every path from v to t has saturated edge.

(2) The preflows constructed by algorithm are blocking.

Modify: s blocked, and departing edges saturated.

Inductive Step:

(a) Scanning in topological order in increase flow guarantees no unblocked, unbalanced vertices.

(b) Scanning in reverse topological order guarentees every blocked vertex gets balanced.

Note: Each step blocks at least 1 vertex $\Rightarrow$ at most n steps flow on edge e increases and decreases at most once $\Rightarrow$ total time $O(|V|^2+|E|)=O(|V|^2)$

Improved Flow Algorithms

Sleater - Tarjan use data structures to decrease blocking flow algorithms to $O(|E|\log|V|)$ time, giving…

Theorem

Max flow can be computed in $O(|V||E|\log|V|)$ time.

Special Case:

<u>0 − 1 Flow</u>, if $\forall e \in E, c(e) = 1$

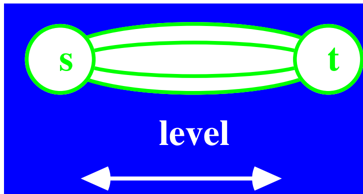
Theorem (Evan and Tarjan)

0 - 1 Flow requires min $(|V|^{2/3}, |E|^{1/2})$ blocking steps of Dinic's Algorithm, so total time $O(\min(|V|^{2/3}, |E|^{1/2})|E|\log(V))$.

<u>Unit Network</u>: All capacities $\in Z$ and every vertex v other than s or t has $\begin{cases}\text{single entering edge or} \\ \text{single departing edge.}\end{cases}$

<u>Claim</u>: If Unit Network G has max flow f, then max level is $\le \left(|V| / value(f)\right) + 1$

<u>Proof</u>: G can be decomposed into $value(f)$ vertex-disjoint paths from s to t.



so value(f)• (level-1) ≤ |V|

Theorem: Dinic's Algorithm has $O\left(|V|^{1/2}\right)$ steps on unit networks.

Proof: (1) If $value(f) \le |V|^{1/2} \Rightarrow \# \, steps \le |V|^{1/2}$

(2) If $value(f) > |V|^{1/2} \Rightarrow level \le \frac{|V|}{|V|^{1/2}} + 1,$

so # steps $\le O\left(|V|^{1/2}\right)$.

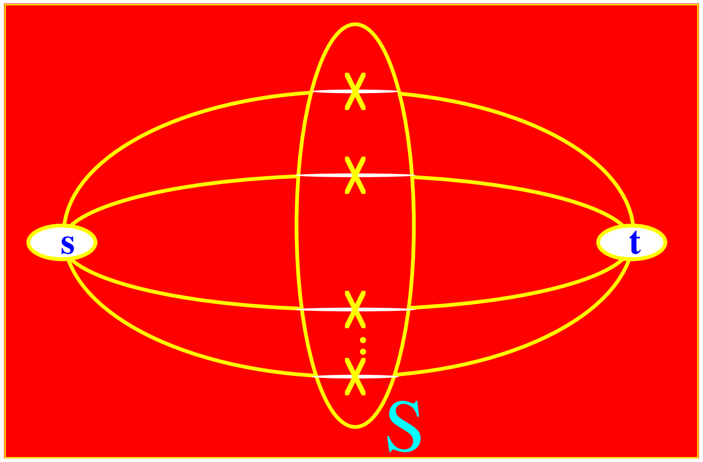
Q.E.D.

Total Time Unit Flow is $O\left(|V|^{1/2} |E| \log |E|\right)$.

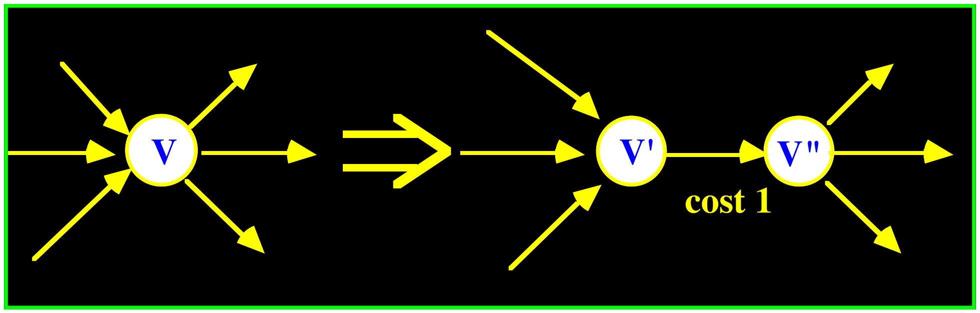
$(s-t)$ Vertex Separator $S \le V$:

if all paths from s to t contain $v \in S$.

Menger's Theroem: The size of the smallest s,t Vertex Separator S is exactly the same as the number of vertex disjoint paths from s to t.



Transform <u>Vertex Connectivity</u> to Unit <u>Network Flow Problem</u>



Total Time $O\left(|V|^{1/2}|E|\log(E)\right)$ to compute

s - t Vertex Connectivity $N(s,t)$ (from s to t).

$N(s,t)$ = number of disjoint paths from s to t.

N(u,v) = min vertex cut size for (G,u,v)

G undirected

<u>Vertex Connectivity:</u> $c(G)$

$$c(G) = \begin{cases} n-1 & \text{if } G \text{ is complete graph} \\ \underset{\substack{u,v \in V \\ (u,v) \notin E}}{\underline{\text{Min}}} N(u,v) & \text{else} \end{cases}$$

Lemma

$$c(G) \le 2|E| / |V|$$

Proof

Connectivity $\le \min_{v \in V}$ degree v,

but $\sum_{v \in V} \text{degree}(v) = 2|E|$

Hence, $c(G) \le 2|E| / |V|$

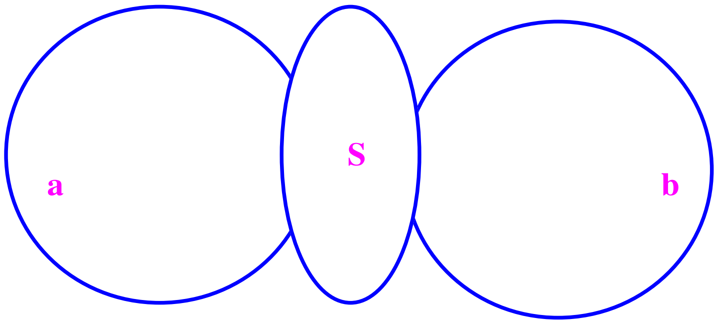
Q.E.D.

(also true for edge connectivity)

Lemma: If S is (u,v) Vertex Separator with $|S| = c(G)$, then

$$c(G) = \min_{(a,b) \notin E} N(a,b) \text{ for all } a \in V - S$$

Proof: $G - S$ has at least 2-components



Let b be any node in a component of $G - S$ which does not have a.

Thus, $N(a,b) \le |S| = c(G)$.

Q.E.D.

Idea: Choose random $a \in V$.

$$O\left(\log\left(\frac{1}{\varepsilon}\right)|V|^{3/2}|E|\log|E|\right)$$

Randomized Algorithm for Vertex Connectivity (Mehlhorn & Students)

Input: $G = (V, E)$, error bound, ε, $0 < \varepsilon < 1$

[0] $\mu \leftarrow |V| - 2$

[1] for $i = 1, 2, \ldots$ until $i \ge \log\left(\frac{1}{\varepsilon}\right) \Big/ \log\left(\frac{|V|}{\mu}\right)$

do select $a_i \in V$ at random

$\mu \leftarrow \min\left(\mu, \min_{b \in V} N(a_i, b)\right)$

od

[2] output μ

<u>Theorem</u>: Prob$(\mu \neq c(G)) \leq \varepsilon$

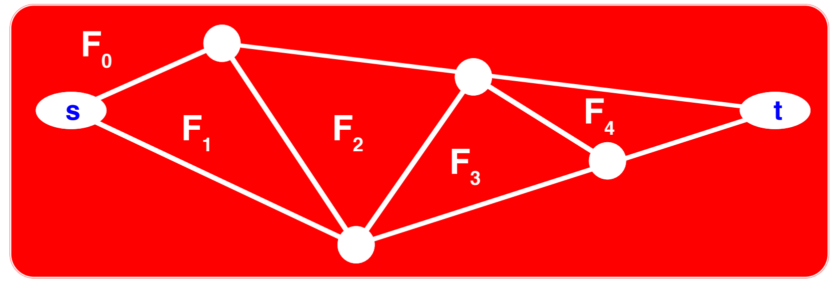
<u>Proof</u>: Let S be a Vertex Separator with $|S| = c(G)$. If $\mu > c(G)$, then $a_1, a_2, \ldots, a_k$ all belong to S, where

$$k \geq \frac{\log\left(\frac{1}{\varepsilon}\right)}{\log\left(\frac{|V|}{c(G)}\right)}$$

Hence, prob $(\mu > c(G)) =$ prob $(a_1, \ldots, a_k \in S)$

$$= \left(\frac{|S|}{|V|}\right)^k = \left(\frac{c(G)}{|V|}\right)^k = 2^{-\log\left(\frac{1}{\varepsilon}\right)} = 2^{\log \varepsilon} = \varepsilon.$$

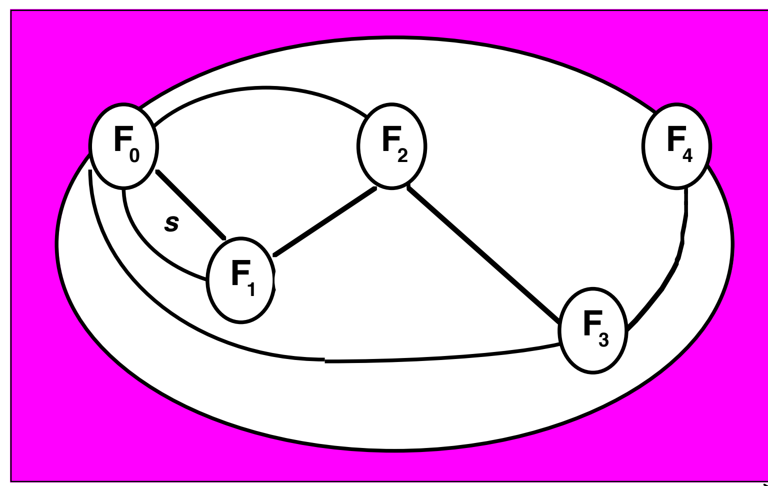
$G = (V, E)$ is a <u>planar graph</u> if G can be embedded on plane so no two edges cross.



<u>Dual</u>: $D(G) = (F, D(E))$

$F =$ faces of embedding

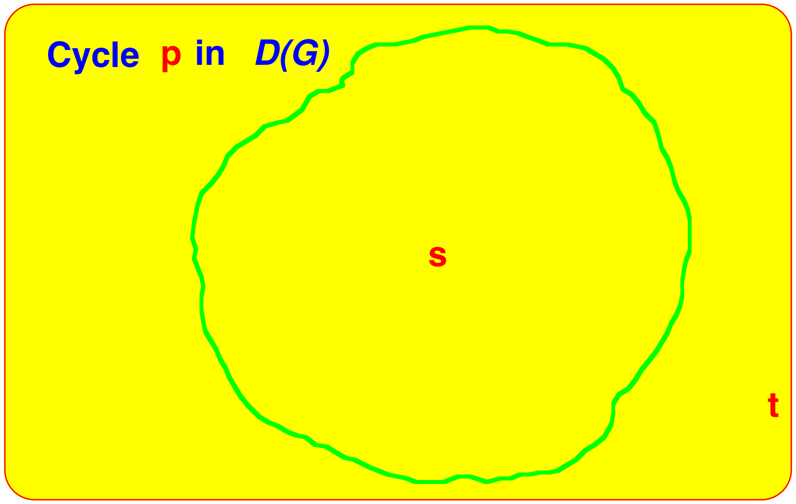
$D(E) = \{\{F_i, F_j\} | e \in E$ is between $F_i, F_j\}$



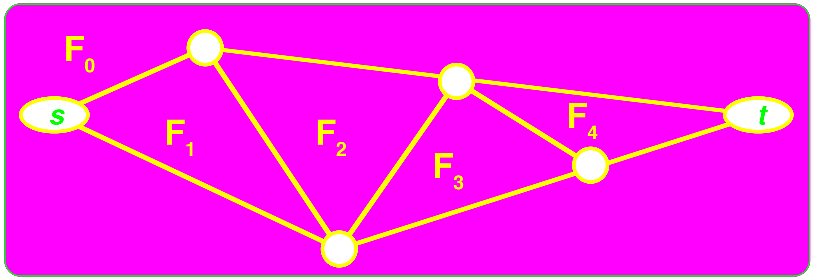
Lemma: If G is a planar embedded network, then max flow in G is same as min cost cycle in $D(G)$ separating s,t.

Proof: We assume $c(F_i, F_j) = c(e)$, if e is between F_i, F_j. Then, by min-cost cut theorem, flow value = min cut $X, \overline{X}$ between s,t

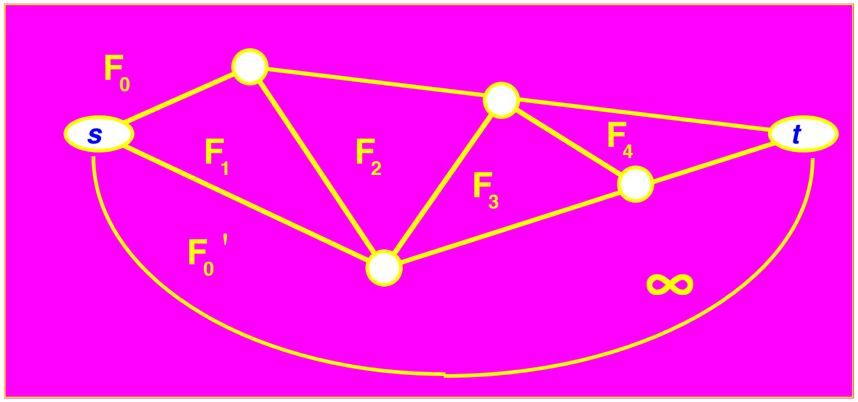
= min cost cycle in $D(G)$ separating s,t



G is outerplanar embedded if the planar embedding has face F_0 incident to all vertices.



Idea: To reduce to Min Cost Path Add new edge (s,t) with weight ∞.



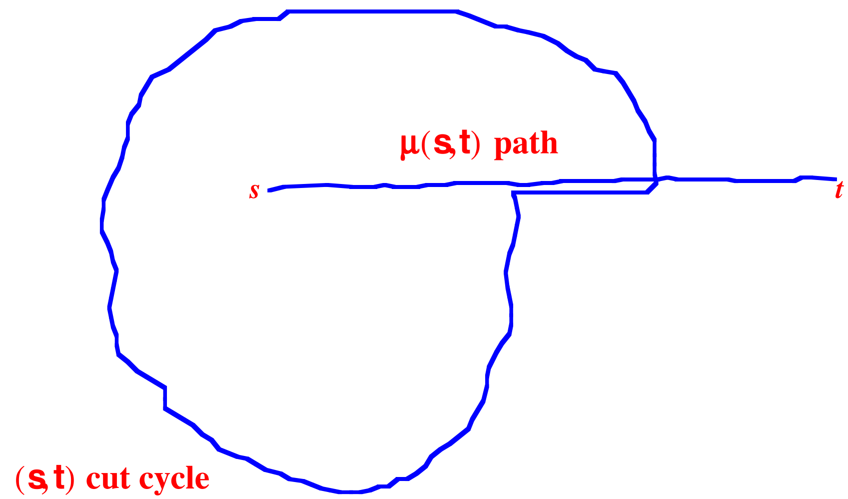
Find min cost path from F_0 to F_0©in $D(G)$

$= \min\ s \cdot t$ cut in G

$=$ max flow value in G

<u>Theorem</u>: If G is <u>outerplanar</u>, we can find max flow in $O(|V|\log|V|)$ time.

<u>Lemma</u>: [Reif] If $\mu(s,t)$ is a minimum cost path in $D(G)$ from a face bounding on s to a face bounding on t, then any min cost cycle in $D(G)$ separating s,t must contain an edge of $\mu(s,t)$.



Proof

Suppose not. Then we can shortcut any cycle of $D(G)$ separating s,t to get a shorter one, using edges of the $\mu(s,t)$ path.

<u>Theorem</u>: [Reif] The min cost flow in a planar graph can be computed in $O\left(|V|\log^2|V|\right)$ time.

<u>Proof</u>: Idea: use $\mu(s,t)$ cut in $D(G)$ to guide a recursive divide and conquer algorithm. On each step, divide the $\mu(s,t)$ path in half and solve the problem on each half, separately, using s,t cut as separator.

Requires $O(\log|V|)$ steps

Each step $O(|V|\log|V|)$ time