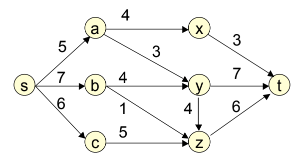
CSE 421
Introduction to Algorithms
Winter 2004

The Network Flow Problem

The Network Flow Problem



How much stuff can flow from s to t?

Net Flow: Formal Definition

Given:

A digraph G = (V,E)

Two vertices s,t in V (source & sink)

A *capacity* c(u,v) ≥ 0 for each (u,v) ∈ E

(and c(u,v) = 0 for all non-edges (u,v))

Find:

A *flow function* f: V x V → R s.t., for all u,v:

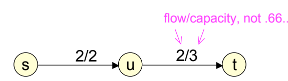
- f(u,v) ≤ c(u,v) [Capacity Constraint]
- f(u,v) = -f(v,u) [Skew Symmetry]
- if u ≠ s,t, f(u,V) = 0 [Flow Conservation]

Maximizing total flow |f| = f(s,V)

Notation:

$f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$

Example: A Flow Function



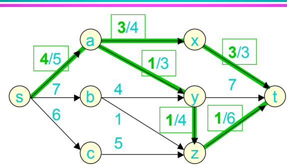
f(s,u) = f(u,t) = 2

f(u,s) = f(t,u) = -2 *(Why?)*

f(s,t) = -f(t,s) = 0 *(In every flow function for this G. Why?)*

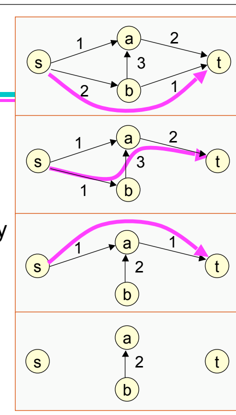
$f(u,V) = \sum_{v \in V} f(u,v) = f(u,s) + f(u,t) = -2 + 2 = 0$

Example: A Flow Function



- Not shown: f(u,v) if ≤ 0
- Note: max flow ≥ 4 since f is a flow function, with |f| = 4

Max Flow via a Greedy Alg?



While there is an s → t path in G

Pick such a path, p

Find c_p, the min capacity of any edge in p

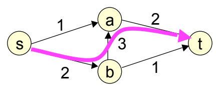
Subtract c_p from all capacities on p

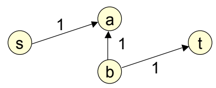
Delete edges of capacity 0

Max Flow via a Greedy Alg?

This does NOT always find a max flow:

If you pick s →b →a →t first,





Flow stuck at 2. But flow 3 possible.

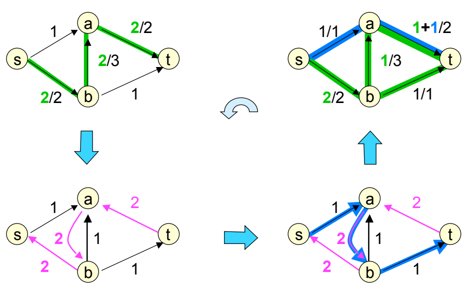
A Brief History of Flow

#	year	discoverer(s)	bound
1	1951	Dantzig	$O(n^2mU)$
2	1955	Ford & Fulkerson	$O(nmU)$
3	1970	Dinitz Edmonds & Karp	$O(nm^2)$
4	1970	Dinitz	$O(n^2m)$
5	1972	Edmonds & Karp Dinitz	$O(m^2 \log U)$
6	1973	Dinitz Gabow	$O(nm \log U)$
7	1974	Karzanov	$O(n^3)$
8	1977	Cherkassky	$O(n^2\sqrt{m})$
9	1980	Galil & Naamad	$O(nm \log^2 n)$
10	1983	Sleator & Tarjan	$O(nm \log n)$
11	1986	Goldberg & Tarjan	$O(nm \log(n^2/m))$
12	1987	Ahuja & Orlin	$O(nm + n^2 \log U)$
13	1987	Ahuja et al.	$O(nm \log(n\sqrt{\log U}/(m+2))$
14	1989	Cheriyan & Hagerup	$E(nm + n^2 \log^2 n)$
15	1990	Cheriyan et al.	$O(n^3/\log n)$
16	1990	Alon	$O(nm + n^{8/3} \log n)$
17	1992	King et al.	$O(nm + n^{2+\epsilon})$
18	1993	Phillips & Westbrook	$O(nm(\log_{m/n} n + \log^{2+\epsilon} n))$
19	1994	King et al.	$O(nm \log_{m/(n \log n)} n)$
20	1997	Goldberg & Rao	$O(m^{3/2} \log(n^2/m) \log U)$ $O(n^{2/3} m \log(n^2/m) \log U)$

n = # of vertices
m= # of edges
U = Max capacity

Source: Goldberg & Rao, FOCS '97

Greed Revisited



Residual Capacity

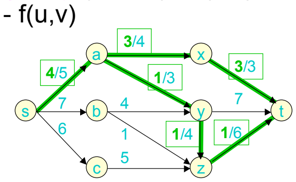
- The *residual capacity* (w.r.t. f) of (u,v) is $c_f(u,v) = c(u,v) - f(u,v)$
- E.g.:

$c_f(s,b) = 7;$

$c_f(a,x) = 1;$

$c_f(x,a) = 3;$

$c_f(x,t) = 0$ (a saturated edge)



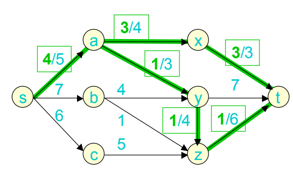
Residual Networks & Augmenting Paths

- The *residual network* (w.r.t. f) is the graph $G_f = (V,E_f)$, where

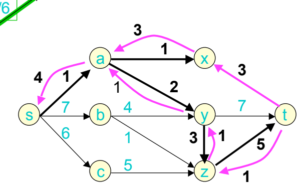
$$E_f = \{ (u,v) \mid c_f(u,v) > 0 \}$$

- An *augmenting path* (w.r.t. f) is a simple $s \rightarrow t$ path in G_f.

A Residual Network

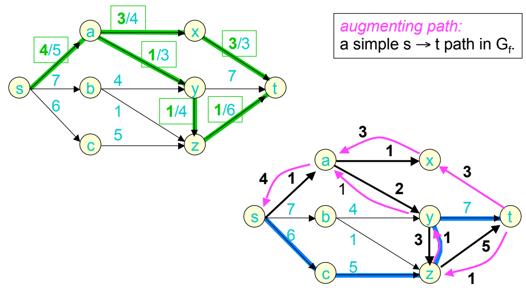


residual network: the graph $G_f = (V,E_f)$, where $E_f = \{ (u,v) \mid c_f(u,v) > 0 \}$



An Augmenting Path

augmenting path:
a simple s → t path in G_f.

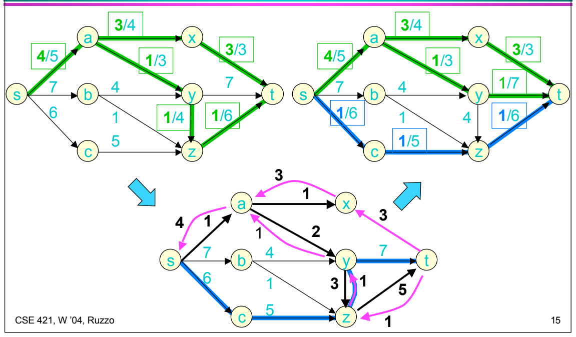


Lemma 1

If f admits an augmenting path p, then f is not maximal.

Proof: "obvious" -- augment along p by c_p, the min residual capacity of p's edges.

Augmenting A Flow



Lemma 1′: Augmented Flows are Flows

If f is a flow & p an augmenting path of capacity c_p, then f' is also a valid flow, where

$$f'(u,v) = \begin{cases} f(u,v)+c_p, & \text{if } (u,v) \text{ in path } p \\ f(u,v)-c_p, & \text{if } (v,u) \text{ in path } p \\ f(u,v), & \text{otherwise} \end{cases}$$

Proof:

a) Flow conservation -- easy
b) Skew symmetry -- easy
c) Capacity constraints – pretty easy

Lma 1′: Augmented Flows are Flows

$$f'(u,v) = \begin{cases} f(u,v)+c_p, & \text{if } (u,v) \text{ in path } p \\ f(u,v)-c_p, & \text{if } (v,u) \text{ in path } p \\ f(u,v), & \text{otherwise} \end{cases}$$

f a flow & p an aug path of cap c_p, then f' also a valid flow.

Proof (Capacity constraints):

(u,v), (v,u) not on path: no change

(u,v) on path:

$$\begin{aligned} f'(u,v) &= f(u,v) + c_p \\ &\le f(u,v) + c_f(u,v) \\ &= f(u,v) + c(u,v) - f(u,v) \\ &= c(u,v) \end{aligned}$$

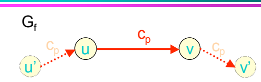
$$\begin{aligned} f'(v,u) &= f(v,u) - c_p \\ &\le f(v,u) \\ &= -f(u,v) \\ &\le c(v,u) \end{aligned}$$

Residual Capacity:
$0 \le c_p \le c_f(u,v) = c(u,v) - f(u,v)$
Cap Constraints:
$-c(v,u) \le f(u,v) \le c(u,v)$

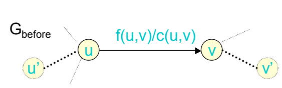
Lemma 1 Example—Case 1

Let (u,v) be any edge in augmenting path. Note

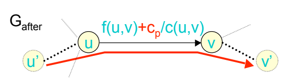
$c_f(u,v) = c(u,v) – f(u,v) \ge c_p > 0$



Case 1: f(u,v) ≥ 0:



Add forward flow

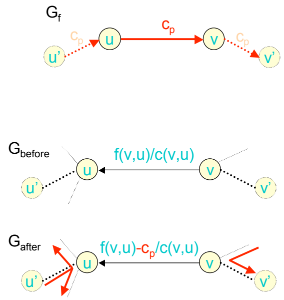


Lemma 1 Example — Case 2

Let (u,v) be any edge in augmenting path. Note $c_f(u,v) = c(u,v) - f(u,v) \geq c_p > 0$

Case 2: $f(u,v) \leq -c_p$:
$f(v,u) = -f(u,v) \geq c_p$

Cancel/redirect reverse flow

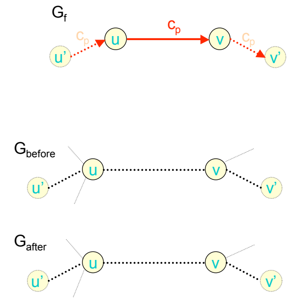


Lemma 1 Example — Case 3

Let (u,v) be any edge in augmenting path. Note $c_f(u,v) = c(u,v) - f(u,v) \geq c_p > 0$

Case 3: $-c_p < f(u,v) < 0$:

???

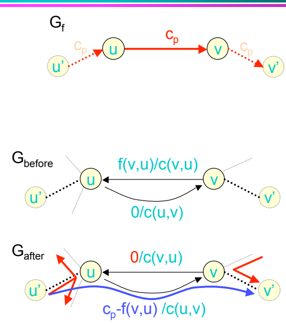


Lemma 1 Example — Case 3

Let (u,v) be any edge in augmenting path. Note $c_f(u,v) = c(u,v) - f(u,v) \geq c_p > 0$

Case 3: $-c_p < f(u,v) < 0$
$c_p > f(v,u) > 0$:

Both:
cancel/redirect reverse flow
and
add forward flow



Ford-Fulkerson Method

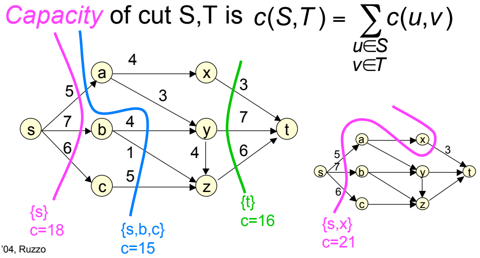
While G_f has an augmenting path, augment

Questions:

» Does it halt?
» Does it find a maximum flow?
» How fast?

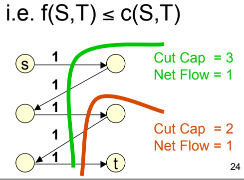
Cuts

- A partition S,T of V is a *cut* if $s \in S$, $t \in T$
- *Capacity* of cut S,T is $c(S,T) = \sum_{u \in S, v \in T} c(u,v)$



Lemma 2

- For any flow f and any cut S,T,
  - the net flow across the cut equals the total flow, i.e., $|f| = f(S,T)$, and
  - the net flow across the cut cannot exceed the capacity of the cut, i.e. $f(S,T) \leq c(S,T)$
- Corollary:
  Max flow ≤ Min cut



Max Flow / Min Cut Theorem

For any flow f, the following are equivalent

(1) $|f| = c(S,T)$ for some cut S,T (a min cut)

(2) f is a maximum flow

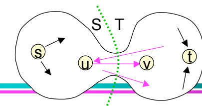
(3) f admits no augmenting path

Proof:

(1) ⇒ (2): corollary to lemma 2

(2) ⇒ (3): contrapositive of lemma 1

(3) ⇒ (1)



S = { u | ∃ an augmenting path from s to u }

T = V - S; $s \in S$, $t \in T$

For any (u,v) in S × T, ∃ an augmenting path from s to u, but not to v.

∴ (u,v) has 0 residual capacity:

$(u,v) \in E \Rightarrow$ saturated $\quad f(u,v) = c(u,v)$

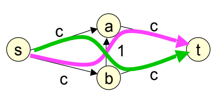
$(v,u) \in E \Rightarrow$ no flow $\quad f(u,v) = 0 = -f(v,u)$

This is true for every edge crossing the cut, i.e.

$$|f| = f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) = \sum_{u\in S, v\in T, (u,v)\in E} f(u,v) = \sum_{u\in S, v\in T, (u,v)\in E} c(u,v) = c(S,T)$$

Corollaries & Facts

- If Ford-Fulkerson terminates, then it's found a max flow.
- It will terminate if c(e) integer or rational (but may not if they're irrational).
- However, may take exponential time, even with integer capacities:



$c = 10^{99}$, say

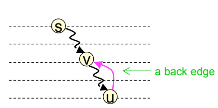
Edmonds-Karp Algorithm

- Use a shortest augmenting path (via Breadth First Search in residual graph)
- Time: $O(n\ m^2)$

BFS/Shortest Path Lemmas

Distance from s is never reduced by:

- Deleting an edge
  proof: no new (hence no shorter) path created
- Adding an edge (u,v), provided v is nearer than u
  proof: BFS is unchanged, since v visited before (u,v) examined

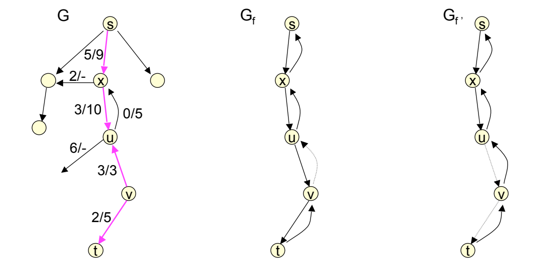


Lemma 3

Let *f* be a flow, G_f the residual graph, and *p* a shortest augmenting path. Then no vertex is closer to *s* after augmentation along *p*.

Proof: Augmentation only deletes edges, adds back edges

Augmentation vs BFS



Theorem 2

The Edmonds-Karp Algorithm performs O(mn) flow augmentations

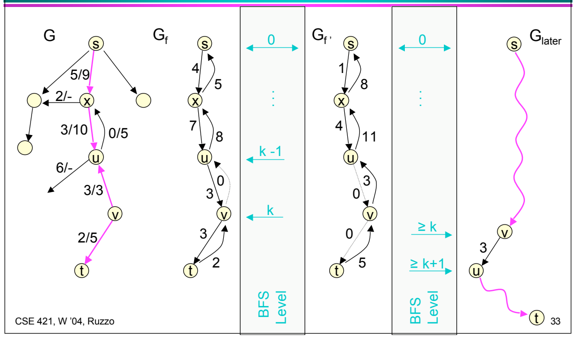
Proof:

{*u*,*v*} is critical on augmenting path *p* if it's closest to *s* having min residual capacity.

Won't be critical again until farther from *s*.

So each edge critical at most *n* times.

Augmentation vs BFS Level



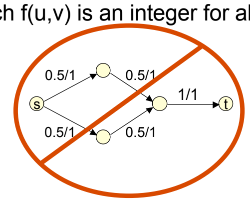
Corollary

Edmonds-Karp runs in O(nm^2)

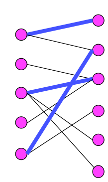
Flow Integrality Theorem

If all capacities are integers

- » The max flow has an integer value
- » Ford-Fulkerson method finds a max flow in which f(u,v) is an integer for all edges (u,v)



Bipartite Maximum Matching



Bipartite Graphs:

- G = (V,E)
- V = L ∪ R (L ∩ R = ∅)
- E ⊆ L × R

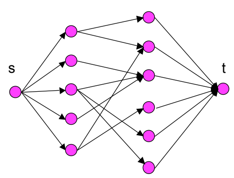
Matching:

- A set of edges M ⊆ E such that no two edges touch a common vertex

Problem:

- Find a matching M of maximum size

Reducing Matching to Flow



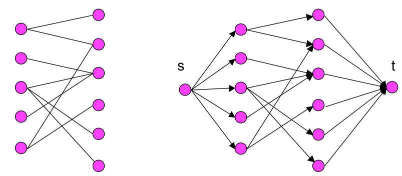
Given bipartite G, build flow network N as follows:

- Add source s, sink t
- Add edges s → L
- Add edges R→ t
- All edge capacities 1

Theorem:
Max flow iff max matching

Reducing Matching to Flow

Theorem: Max matching size = max flow value



M → f? Easy – send flow only through M
f → M? Flow integrality Thm, + cap constraints

Notes on Matching

- Max Flow Algorithm is probably overly general here
- But most direct matching algorithms use "augmenting path" type ideas similar to that in max flow – See text & homework
- Time $mn^{1/2}$ possible via Edmonds-Karp