Lecture 9 - Network Flows

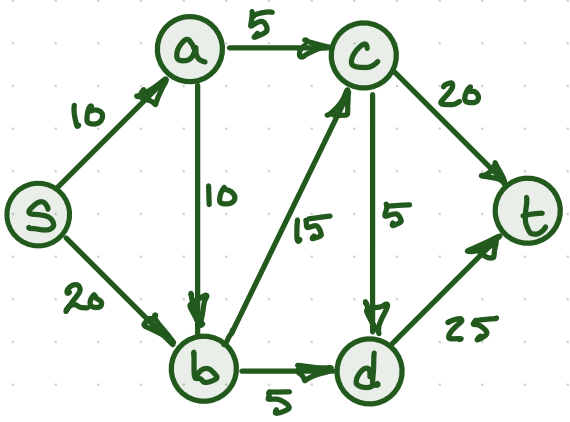
CMPT 307 D-1 Fall 2024

Instructor: David Mitchell

Flow Networks

Directed Graph $G=\langle V, E\rangle$ with:

- Edge capacities $c: E \to \mathbb{R}^+$,
- source vertex s,
- sink vertex t
- Internal nodes $= V - \{s, t\}$



- Used to model flow of: traffic, data, material, fluids, etc
  in networks of: roads, cable, conveyors, pipes, etc.

For simplicity, we assume:

- in-degree(s) = out-degree(t) = 0
- capacities are integers
- every vertex is reachable from s

Flow in Network G

s-t flow:

$f : E \to \mathbb{R}^+$ // $f(e)$ = amount of flow carried by e.

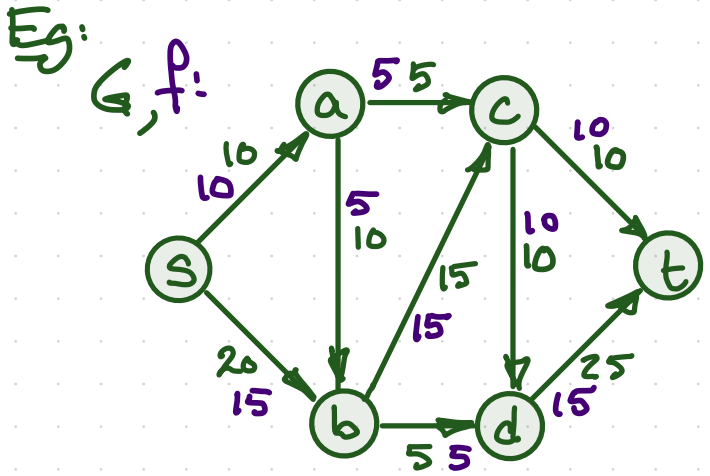
s.t. · $0 \le f(e) \le c_e$, for every $e \in E$. // capacity condition.

· $f^{in}(v) = f^{out}(v)$, for every $v \in V - \{s, t\}$ // conservation condition.

Where: $f^{in}(v) = \sum_{e \text{ into } v} f(e)$; $f^{out}(v) = \sum_{e \text{ out of } v} f(e)$.

value of flow f: $v(f) = f^{out}(s)$.

Eg: G, f:



$v(f) = f^{out}(s) = 25$

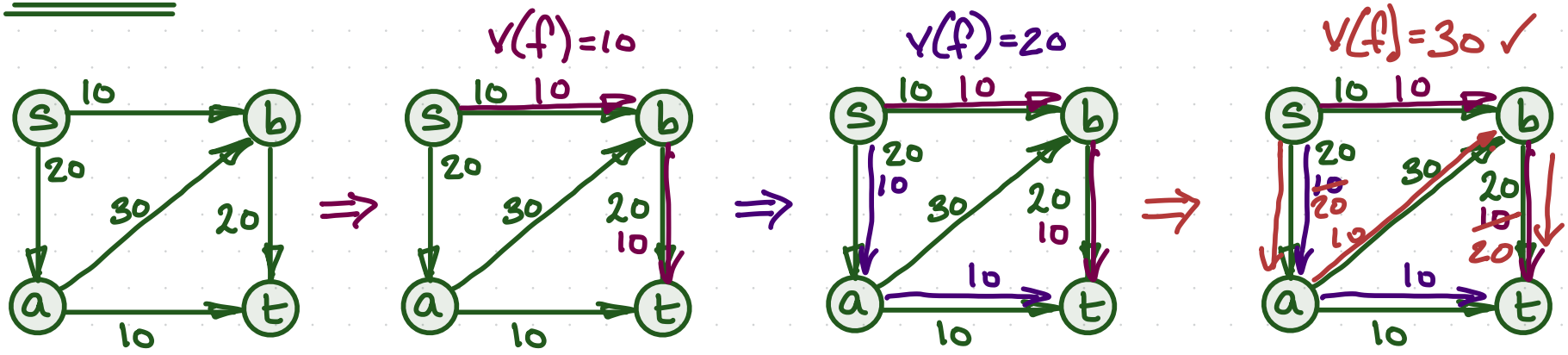
Maximum Flow Problem

Instance: Flow network G

Question: What is the max. value of a flow for G?

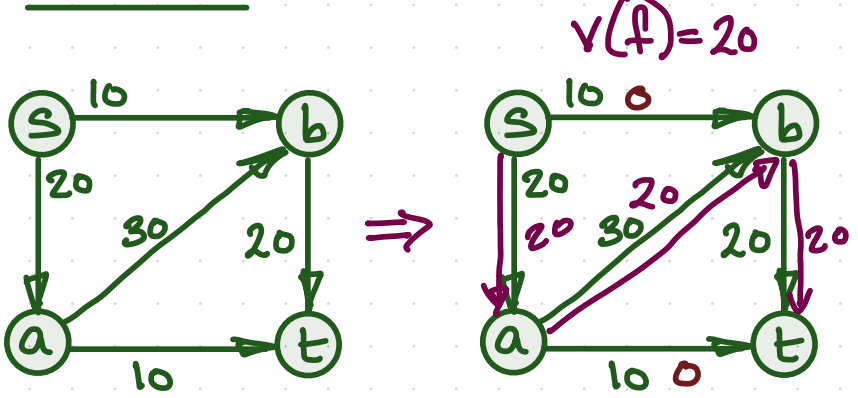
- There is no known dynamic programming algorithm for this problem, and simple greedy methods do not work.
- The algorithm we give is an iterative improvement algorithm:
  - start with an easy-to-find non-optimal solution
  - repeatedly improve it

Consider:



Here, simple greedy improvements lead to optimality.

But also:

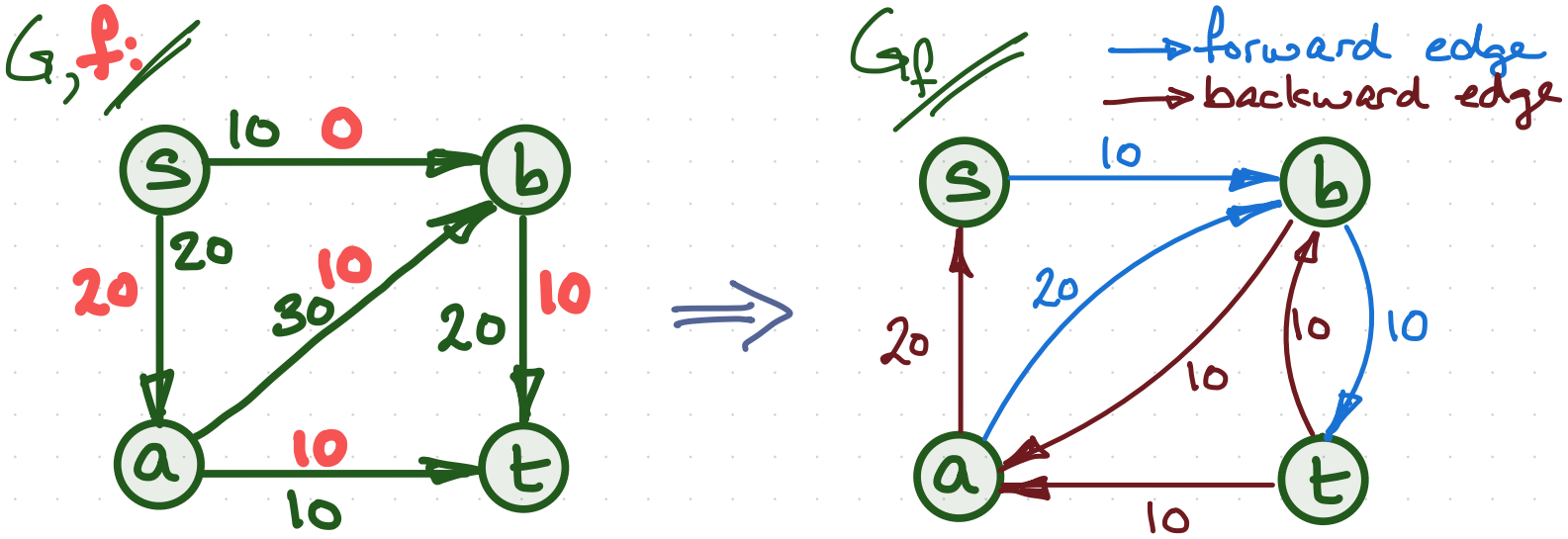


← Cannot improve this without "taking back" some of the a→b flow.

Residual Graph for G and f

$G_f = (V, E_f)$, where E_f contains

1. (u,v) with "residual capacity" $(c_e - f(e))$, if $(u,v) \in E$, $f(e) < c_e$.
   // "forward edges" represent "spare capacity" we could use.
2. (v,u) with "residual capacity" $f(e)$, if $(u,v) \in E$, $f(e) > 0$.
   // "backward edges" represent flow we could "take back".



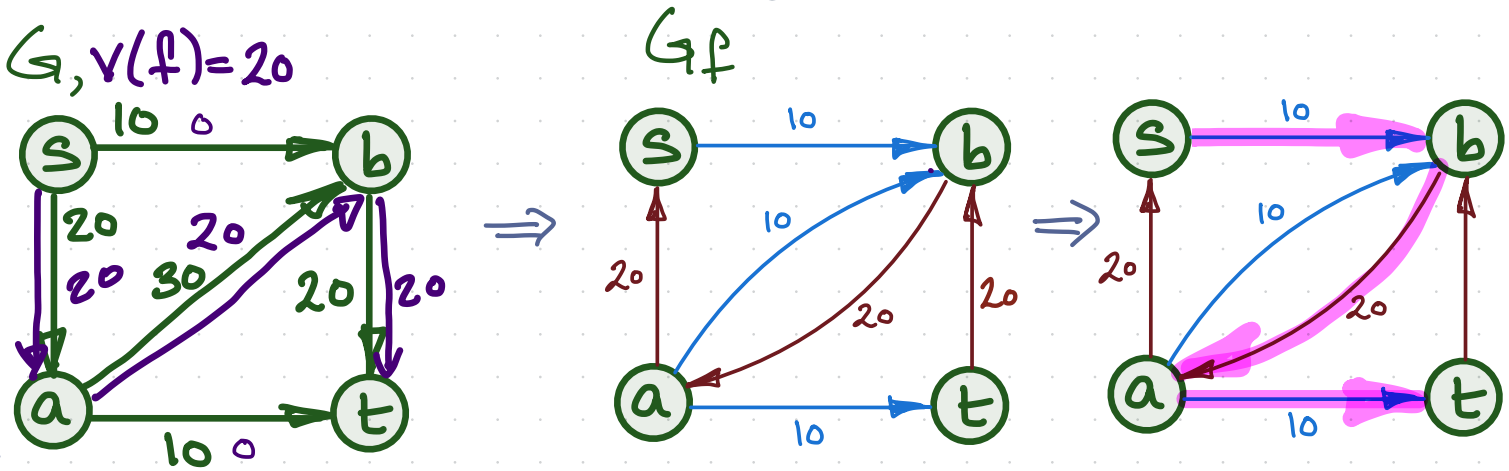
G with f

Residual Graph for G,f

Notice: We can increase the flow on the path $s \to b \to t$.

Augmenting Paths

- An s-t path in G_f is called an augmenting path.
- The min. residual capacity on the path is the bottleneck.



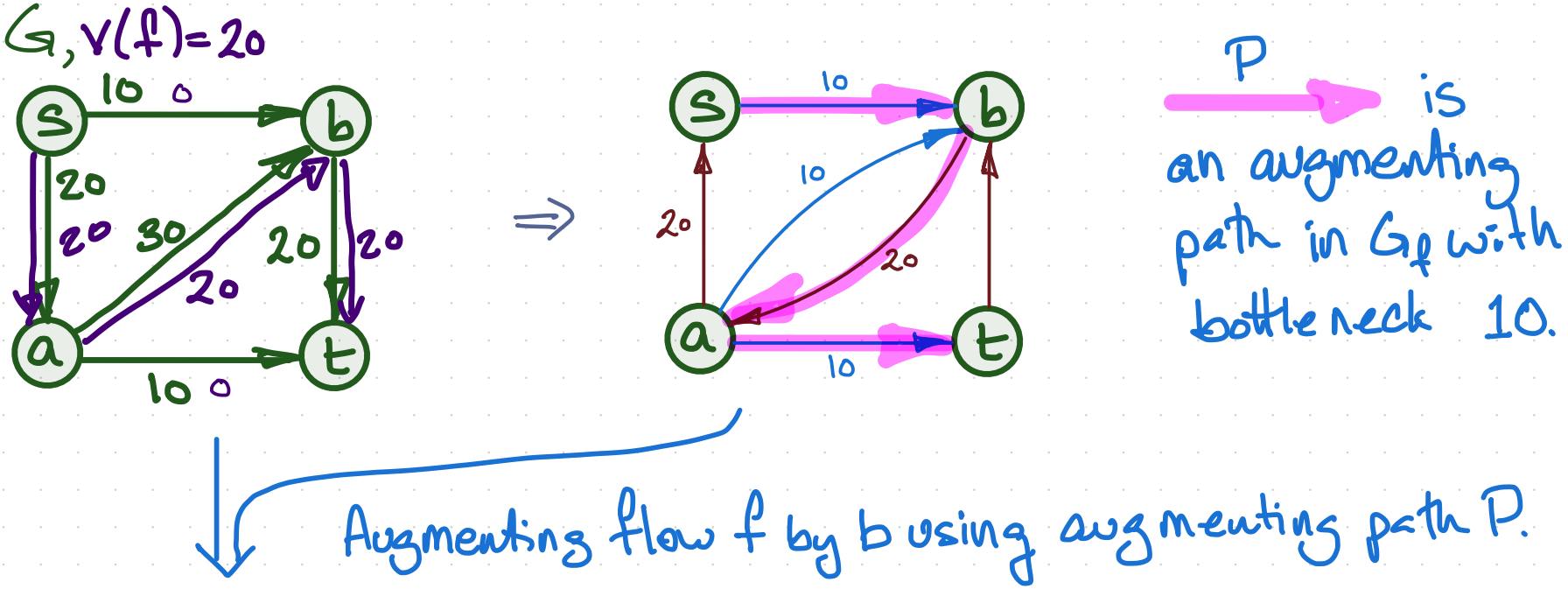
Is an s-t path with minimum residual capacity 10

We call it an augmenting path with bottleneck 10.

this is the example ▲ showing greedy fails, slide 5.

Augmenting Paths

- An augmenting path in G_f with bottleneck b gives us a way to increase (augment) the flow f by b.



Augmenting a Flow

```
augment(f, P){ // flow f; P an augmenting path
    b = bottleneck(P, f)
    for each forward edge e=(u,v) ∈ P
        f(e) = f(e) + b
    for each backward edge (u,v) ∈ P
        e = (v,u)
        f(e) = f(e) - b
    return f
}
```

Q: Suppose G is a network, f a flow for G, P an augmenting path in G_f.

Is augment(f,P) always a flow for G with $v(\text{augment}(f,P)) > v(f)$?

Augmenting a Flow

Fact 1) $f' = \text{augment}(f, P)$ is a flow

Pf: We must show capacity and conservation conditions hold for f'.

Capacity) Need to show that, for every edge e, $0 \le f'(e) \le c_e$

Let $b = \text{bottleneck}(P, f)$. Observe that, for any edge e in P:

- if e is a forward edge, $0 \le b \le c_e - f(e)$
- if e is a backward edge, $0 \le b \le f(e)$

Suppose $e = (u,v) \in E$, and consider 3 cases:

1) If (u,v) is a forward edge of P, then ← the residual cap.

$$0 \le f(e) \le f'(e) = f(e) + b \le f(e) + (c_e - f(e)) = c_e.$$

2) If (v,u) is a backward edge of P, then

$$c_e \ge f(e) \ge f'(e) = f(e) - b \ge f(e) - f(e) = 0.$$

← the residual cap.

3) If neither 1 nor 2 holds // no (u,v) or (v,u) in P.

$$c_e \ge f'(e) - f(e) \ge 0$$

Fact 1 "example" of capacity condition.

Suppose $e=(u,v) \in E$, and consider 3 cases:

1) If (u,v) is a forward edge of P, then ← the residual cap.

$$0 \le f(e) \le f'(e) = f(e) + b \le f(e) + (c_e - f(e)) = c_e.$$

G,f: $c_e = 5$, $f(e) = 3$ (u → v)

G_f: $c_e - f(e) = 5 - 3 = 2$ (u → v)

the residual cap. of (u,v) in G_f.

Increase flow by the bottleneck value b. We know $0 < b \le 2 = c_e - f(e)$

Final flow is: $f'(e) = f(e) + b$, so

$$0 \le f(e) < f'(e) = f(e) + b \le f(e) + 2 \le f(e) + (c_e - f(e)) = c_e$$

2) If (v,u) is a backward edge of P, then

$$c_e \ge f(e) \ge f'(e) = f(e) - b \ge f(e) - f(e) = 0$$

G,f: $c_e = 5$, $f(e) = 3$ (u → v)

residual capacity of (v,u)

$f(e) = 3$ (u ← v)

Decrease flow by b, where $0 < b \le 3 = f(e)$

$$c_e \ge f(e) \ge f'(e) = f(e) - b \ge f(e) - 3 = f(e) - f(e) = 0$$

3) $f'(e) = f(e)$. // "e not in P." (more precisely, if $e=(u,v)$, neither (u,v) nor (v,u) in P)

Augmenting a Flow

Fact 1) $f' = \text{augment}(f, P)$ is a flow (proof continued)

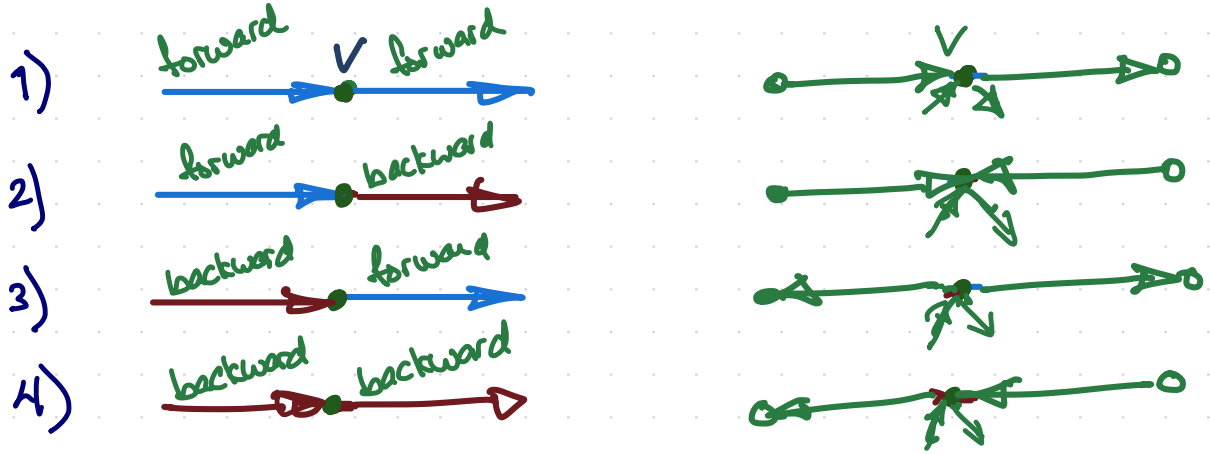
Pf: We must show capacity and conservation conditions hold for f'.

(Conservation)

Need to show that $f'^{in}(v) = f'^{out}(v)$, for every $v \in V - \{s, t\}$

Exercise.

Hint: If we consider a vertex v on augmenting path P, there are 4 cases in terms of forward or backward edges:

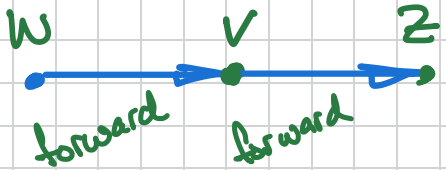


Prove each case.

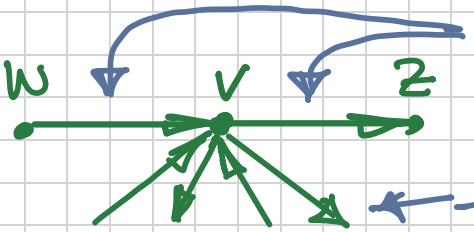
(Next 2 slides have cases 1,2 in minute detail. Do the others!)

Case 1:

Path P in G_f



Corresponding Edges in G



we increase the flow on both these edges by b.

← there may be many edges entering/leaving v.

We need to show that $f'_{in}(v) = f'_{out}(v)$.

Suppose (w,v) and (v,z) are the edges of P that are incident to v. Then:

$f'_{in}(v) = \sum_{(u,v)\in E} f'(u,v)$ // def'n of $f'_{in}(v)$

$= \sum_{\substack{(u,v)\in E \\ u\neq w}} f'(u,v) + f'(w,v)$ // separating out // (w,v)

$= \sum_{\substack{(u,v)\in E \\ u\neq w}} f'(u,v) + f(w,v) + b$ // flow on (w,v) // increased by b.

$= \sum_{\substack{(u,v)\in E \\ u\neq w}} f(u,v) + f(w,v) + b$ // flow on other edges // into v unchanged.

$= \sum_{(u,v)\in E} f(u,v) + b$

$= f_{in}(v) + b$ // def'n of $f_{in}(v)$.

$= f_{out}(v) + b$ // f is a flow, so // $f_{out}(v) = f_{in}(v)$.

⋮

b

⋮

$= \sum_{(v,u)\in E} f(v,u) + b$ // def'n of $f_{out}(v)$

$= \sum_{\substack{(v,u)\in E \\ u\neq z}} f(v,u) + f(v,z) + b$ // separating out (v,z)

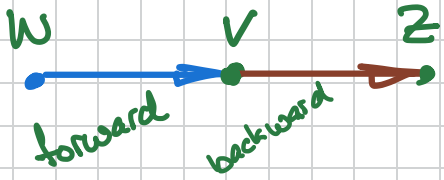
$= \sum_{\substack{(v,u)\in E \\ u\neq z}} f(v,u) + f'(v,z)$ // flow on (v,z) increased // by b.

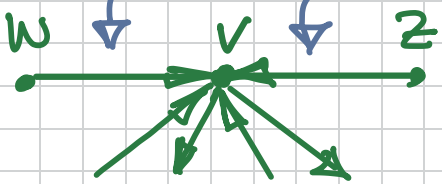
$= \sum_{\substack{(v,u)\in E \\ u\neq z}} f'(v,u) + f'(v,z)$ // flow on other edges // out of v unchanged

$= \sum_{(v,u)\in E} f'(v,u)$ //

$= f'_{out}(v)$. // def'n of $f'_{out}(v)$.

Case 2: Path P in G_f — Corresponding Edges in G





we increase the flow on (w,v) by b.

we decrease the flow on (v,z) by b.

We need to show that $f'_{in}(v) = f'_{out}(v)$.

Suppose (w,v) and (v,z) are the edges of P that are incident to v. Then:

$$f'_{in}(v) = \sum_{(u,v)\in E} f'(u,v) \quad \text{// def'n of } f'_{in}(v)$$

$$= \sum_{\substack{(u,v)\in E \\ u\neq w}} f'(u,v) + f'(w,v) + f'(z,v) \quad \text{// separating out (w,v) and (z,v)}$$

$$= \sum_{\substack{(u,v)\in E \\ u\neq w,z}} f'(u,v) + f(w,v) + b + f'(z,v) \quad \text{// flow on (w,v) increased by b.}$$

$$= \sum_{\substack{(u,v)\in E \\ u\neq w,z}} f'(u,v) + f(w,v) + b + f(z,v) - b \quad \text{// flow on (z,v) decreased by b}$$

$$= \sum_{\substack{(u,v)\in E \\ u\neq w,z}} f'(u,v) + f(w,v) + f(z,v) \quad \text{// } b-b=0$$

$$= \sum_{\substack{(u,v)\in E \\ u\neq w,z}} f(u,v) + f(w,v) + f(z,v) \quad \text{// flow on other edges into v unchanged.}$$

⋮

$$= \sum_{(u,v)\in E} f(u,v) \quad \text{// def'n of } f_{in}(v).$$

$$= f_{in}(v)$$

$$= f_{out}(v) \quad \text{// f is a flow, so } f_{in}(v) = f_{out}(v).$$

$$= \sum_{(v,u)\in E} f(v,u) \quad \text{// def'n of } f_{out}(v)$$

$$= \sum_{(v,u)\in E} f'(v,u) \quad \text{// flows out of v did not change}$$

$$= f'_{out}(v). \quad \text{// def'n of } f'_{out}(v).$$

□

The Ford-Fulkerson Algorithm


```
FF(G){ // G is a flow network
    f(e) = 0 for all e in G.
    while there is an s-t path in Gf {
        P = a simple s-t path in Gf
        f = augment(f,P)
    }
    return f // f is a max. flow for G.
}
```

We want to show:

1) FF always terminates ✓
2) When FF terminates, f is optimal ? Next
3) FF is efficient - ? (not this version)

Ford-Fulkerson: Termination

Fact 2) All flow values $f(e)$ and residual capacities in G_f during execution of FF are integers.

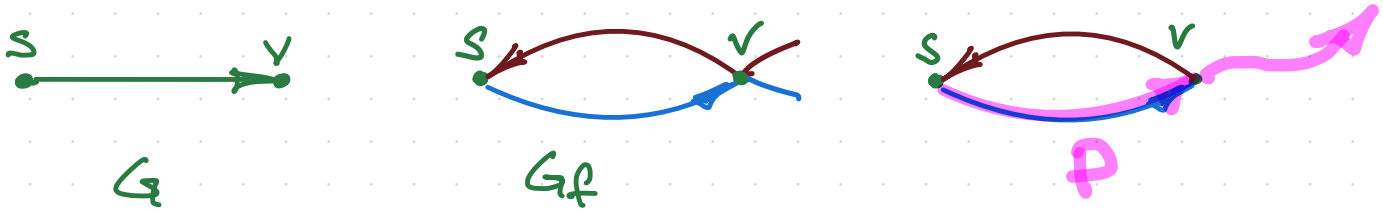
Pf: By induction on i, the # iterations of the while loop.

- $i=0$) flows are 0, residual capacities are all edge capacities, which we assumed to be integers.
- Assume all residual capacities are integers on i^{th} iteration, $i \geq 0$. ← flows and.
- In the i^{th} iteration, b was an integer, since it was a residual capacity. So the flows on the $i+1^{st}$ iteration are integers. The capacities are also integers, so the new residual capacities will be integers also.

Ford-Fulkerson: Termination

Fact 3) If f is a flow in G and P a simple s-t path in G_f, and $f' = \text{augment}(f, P)$, then $v(f') > v(f)$.

Pf: The first edge e of P in G_f is out of s, i.e., (s,v) for some v. Since s has no entering edge, e is a forward edge.



We increase the flow on this edge by $\text{bottleneck}(P,f)$, and we do not change the flow on any other edge incident to s. Then, since $\text{bottleneck}(P,f) \geq 1$;*

$$v(f') = v(f) + \text{bottleneck}(P,f) > v(f)$$

□

* residual capacities are all positive.

Ford-Fulkerson: Termination

Theorem: Let G be a flow network with all capacities integers, and

$$C = \sum_{e \text{ out of } s} c_e.$$

Then FF terminates in at most C iterations.

Pf: Initially $v(f) = 0$. In every iteration $v(f)$ increases by at least one, and $v(f)$ cannot be greater than C. ∎

Notice: Integer capacities are essential in this proof.

Ford-Fulkerson: Time Complexity

Cor: FF can be implemented to run in time $O(mC)$

Pf: There are at most C iterations. Each iteration requires updating G_f, finding an s-t path in G_f, and then updating f. These can all be done in time $O(m+n)$, which under our assumption that every node is reachable from s is $O(m)$.

Ex: Justify the second part of the last sentence.

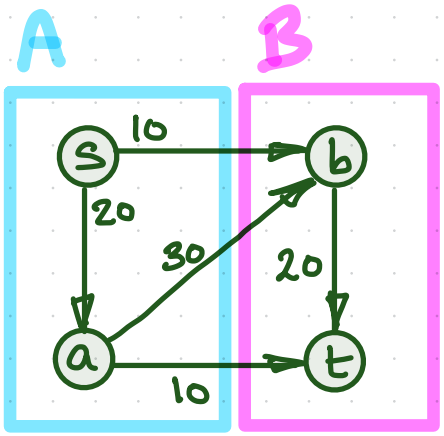
Note: This time bound is very poor, because the value of C can be very large relative to the size of input. In particular, it is not polynomial.

Flows and Cuts

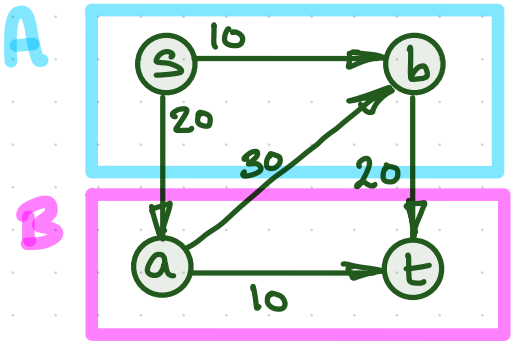
s-t Cut in G: A bipartition (A, B) of V s.t. $s \in A$, $t \in B$.

capacity of cut (A, B): $c(A, B) = \sum_{\substack{e \text{ out} \\ \text{of } A}} c_e$ // total capacity of // edges leaving A.

Eg/



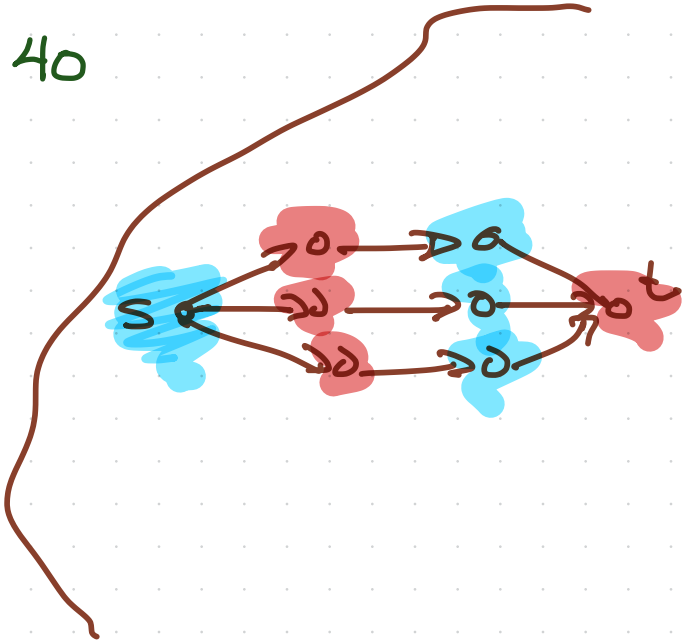
$c(A, B) = 50$



$c(A, B) = 40$

Define: for $A \subseteq V$, $f^{out}(A) = \sum_{\substack{e \text{ out} \\ \text{of } A}} f(e)$;

$$f^{in}(A) = \sum_{\substack{e \text{ into} \\ A}} f(e),$$



Flows and Cuts

Fact 4) If f is an s-t flow and (A,B) an s-t cut, then $v(f) = f^{out}(A) - f^{in}(A)$.

Pf: $v(f) = f^{out}(s)$ // Def'n.

$= f^{out}(s) - f^{in}(s)$ // $f^{in}(s) = 0$

$= \sum_{v \in A} (f^{out}(v) - f^{in}(v))$ // $f^{out}(v) - f^{in}(v) = 0$ for every internal // node, so for all $v \in A - s$.

$= \sum_{v \in A} f^{out}(v) - \sum_{v \in A} f^{in}(v)$ // (*)

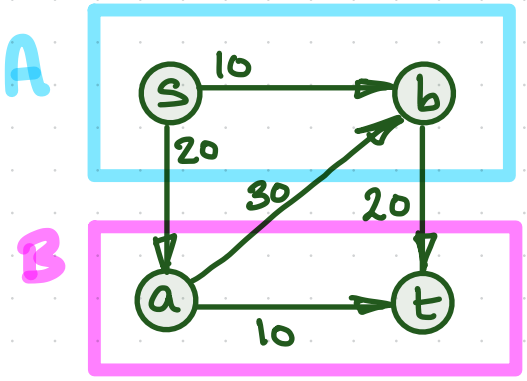
If $e = (u,v) \in E$ and $u, v \in A$ then $f(e)$ occurs once in each summation in (*). These occurrences cancel, so we have:

$$v(f) = \sum_{v \in A} f^{out}(v) - \sum_{v \in A} f^{in}(v)$$

$$= \sum_{e \text{ in } A} f(e) - \sum_{e \text{ in } A} f(e) + \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e)$$

$$= f^{out}(A) - f^{in}(A) \quad \square$$



Flows and Cuts

Fact 4) If f is an s-t flow and (A,B) an s-t cut, then $v(f) = f^{out}(A) - f^{in}(A)$.

Pf: $v(f) = f^{out}(s)$ // Def'n.

$= f^{out}(s) - f^{in}(s)$ // $f^{in}(s) = 0$

$= \sum_{v \in A} (f^{out}(v) - f^{in}(v))$ // $f^{out}(v) - f^{in}(v) = 0$ for every internal node, so for all $v \in A - s$.

$= \sum_{v \in A} f^{out}(v) - \sum_{v \in A} f^{in}(v)$ // (*)

$= \sum_{v \in A} \sum_{\substack{u \\ (v,u) \in E \\ e = (v,u)}} f(e) - \sum_{v \in A} \sum_{\substack{u \\ (u,v) \in E \\ e = (u,v)}} f(e)$ // Def'n of $f^{out}(v)$, $f^{in}(v)$

$= \sum_{\substack{(v,u) \in E, \\ v \in A}} f(v,u) - \sum_{\substack{(u,v) \in E \\ v \in A}} f(u,v)$ // writing $f(u,v)$ for $f((u,v))$

$= \sum_{\substack{(v,u) \in E \\ v,u \in A}} f(v,u) + \sum_{\substack{(v,u) \in E \\ v \in A, u \notin A}} f(v,u) - \sum_{\substack{(u,v) \in E \\ u,v \in A}} f(u,v) - \sum_{\substack{(u,v) \in E \\ v \in A, u \notin A}} f(u,v)$

$= \sum_{\substack{(v,u) \in E \\ v \in A, u \notin A}} f(v,u) - \sum_{\substack{(u,v) \in E \\ v \in A, u \notin A}} f(u,v)$ // $\sum_{\substack{(v,u) \in E \\ v,u \in A}} f(v,u) = \sum_{\substack{(u,v) \in E \\ u,v \in A}} f(u,v)$

$= f^{out}(A) - f^{in}(A)$ // Def'n of $f^{out}(A)$, $f^{in}(A)$. ∎

Flows and Cuts

Observations:

- For $A=\{s\}$, Fact 4 gives us $v(f) = f^{out}(s) - f^{in}(s) = f^{out}(s)$ // the def'n of $v(f)$.
- If (A,B) is an s-t cut, $f^{out}(A) = f^{in}(B)$ and $f^{in}(A) = f^{out}(B)$.
  // because edges out of A = edges into B & vice versa.
- It follows that $v(f) = f^{in}(t)$. // Set $B=\{t\}$.

Fact 5) For every flow f and every s-t cut (A,B), $v(f) \leq c(A,B)$.

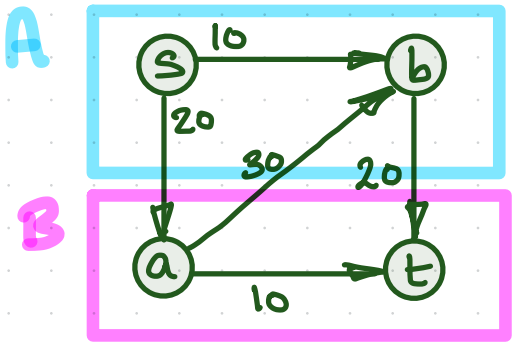
Pf: $v(f) = f^{out}(A) - f^{in}(A)$ // Fact 4.

$\leq f^{out}(A)$ // $f^{in}(A) \geq 0$

$= \sum_{\substack{e \text{ out} \\ \text{of } A}} f(e)$ // def'n of f^{out}

$\leq \sum_{\substack{e \text{ out} \\ \text{of } A}} c_e$ // capacity condition

$= c(A,B)$. // def'n. of $c(A,B)$.



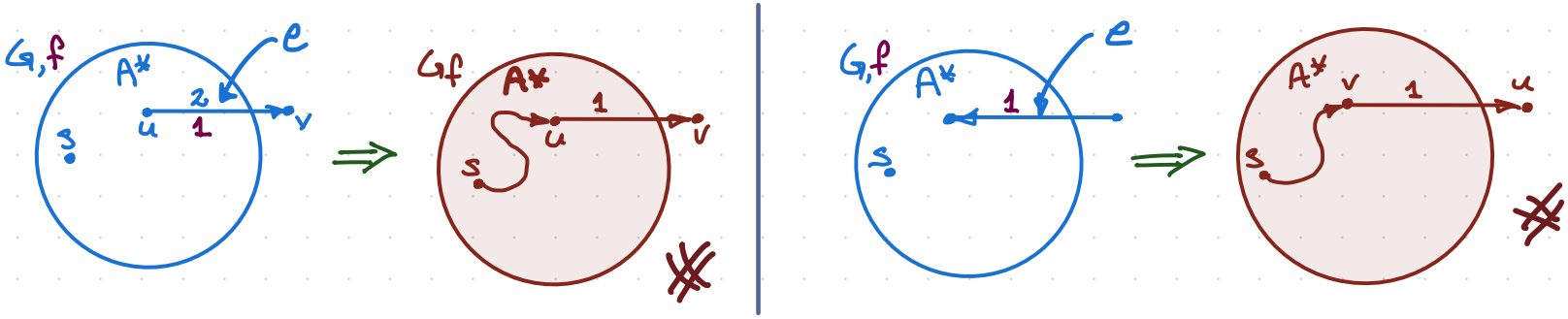
Flows, Cuts & Augmenting Paths

Lemma: Let f be an s-t flow s.t. there is no s-t path in G_f. Then there is an s-t cut (A^*, B^*) s.t. $v(f) = c(A^*, B^*)$.

Pf: Let $A^* = \{ v \mid v \text{ is reachable from } s \text{ in } G_f \}$; $B^* = V - A^*$. (A^*, B^*) is a partition of V, $s \in A^*$, and since t is not reachable from s, $t \in B^*$. So (A^*, B^*) is an s-t cut.

Suppose $e = (u, v)$ is an edge out of A^*. If $f(e) < c_e$, then e is a forward edge in G_f, and there is an s-v path in G_f, which is a contradiction because $v \in B^*$. So $f(e) = c_e$ for edges e out of A^*. (*)

Suppose $e = (u, v)$ is an edge into A^*. If $f(e) > 0$, then G_f has a backward edge (v, u), and there is an s-v path in G_f, which is a contradiction. So $f(e) = 0$ for every edge e into A^*. (**)



Flows, Cuts & Augmenting Paths

Lemma: Let f be an s-t flow s.t. there is no s-t path in G_f. Then there is an s-t cut (A^*, B^*) s.t. $v(f) = c(A^*, B^*)$.

Pf: continued:

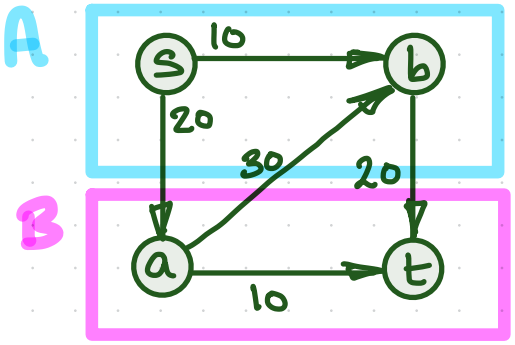
We have: $v(f) = f^{out}(A^*) - f^{in}(A^*)$ // Fact 4

$= \sum_{e \text{ out of } A^*} f(e) - \sum_{e \text{ into } A^*} f(e)$ // defn f^{out}, f^{in}

$= \sum_{e \text{ out of } A^*} c_e$ // by (*) and (**) above

$= c(A^*, B^*)$ // def'n of $c(A,B)$.

$\square$



Ford-Fulkerson: Optimality.

Thm: The flow returned by FF is optimal.

Pf: Let f^* be the flow returned by FF.

The termination condition implies that G_{f^*} has no s-t path, so by the lemma, there is an s-t cut (A, B) s.t. $v(f^*) = c(A, B)$.

Then f^* must be optimal, as a flow with greater value would contradict Fact 5. ∎

Max-Flow Min-Cut Theorem

Thm: For every flow network, there is a flow f and a cut (A, B) such that:

1) $v(f) = c(A, B)$
2) f is a maximum flow
3) (A, B) is a minimum capacity cut

"Max flow = Min Cut"

Pf. (1), (2) are established by the Lemma (or optimality of FF.)
Minimality of (A, B) follows because a cut of smaller capacity would contradict Fact 5.

Further Facts

Cor: From a max. flow we can construct a min. cut in time $O(m)$.

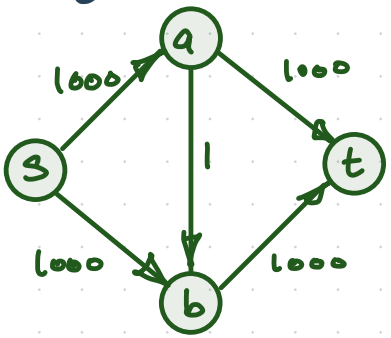
Pf: Execute the construction from proof of Fact 5.

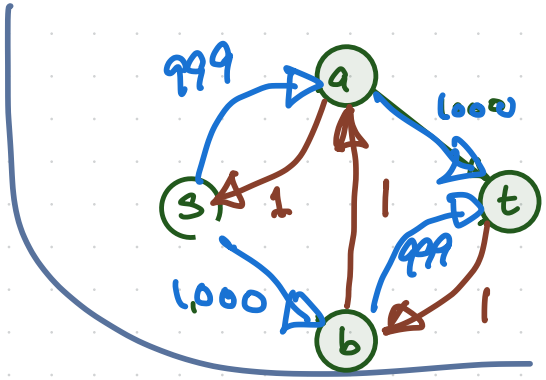
Cor: If all capacities of a flow network are integers, then there is a max. flow in which the flow on every edge is integral.

Pf: Every flow produced by FF, including the last one, will have integral edge flows (Fact 2).

Choosing Bad Augmenting Paths

Consider this network:





FF can solve it (i.e, find the max. flow f with $v(f) = 2000$) in two iterations:

1) Choose augmenting path $s \to a \to t$
2) Choose augmenting path $s \to b \to t$

But: FF might instead do this

1) Choose $P_1 = s \to a \to b \to t$, adding a flow of 1.
2) Choose $P_2 = s \to b \to a \to t$, " " " "
3) Choose $P_1 = s \to a \to b \to t$, " " " "
4) Choose $P_2 = s \to b \to a \to t$ " " " " take time

So: In a network with total capacity C on edges leaving S, it could take $\Omega(C)$ iterations, which may be exponential in the input size.

Scaling Version of FF

Idea: Search first for augmenting paths with large bottleneck.

Define: $G_f(\Delta)$ = subgraph of G_f containing only edges with residual capacity $\geq \Delta$.

// An augmenting path P in $G_f(\Delta)$ has bottleneck $\geq \Delta$.

```
Scaling-Max-Flow {
    f(e) = 0 for all e ∈ G.
    Δ = max. power of 2 s.t. Δ ≤ max { c_e | e ∈ E }
    while Δ ≥ 1 {
        while G_f(Δ) has an s-t path
            P = a simple s-t path in G_f(Δ)
            f = augment(f, P)
        }
        Δ = Δ/2
    }
    return f
}
```

Complexity of Scaling Max-Flow Alg.

Thm: The Scaling Max-Flow (SMF) algorithm:

1) returns a max flow
2) does at most $2m(1 + \lceil \log_2 C \rceil)$ augmentations
3) can be implemented to run in time $O(m^2 \log C)$.

This is polynomial in the input size

Proof Sketch

1) SMF is a special case of FF: residual capacities are always integer, so when $\Delta = 1$, $G_f(\Delta) = G_f$.
2) The algorithm uses $\sim \log_2 C$ values of Δ.
   During the phase where $\Delta = k$, each augmentation increases the flow by at least k. With some work, one can show there are $\leq 2m$ augmentations for each value of Δ.
3) Each augmentation takes time $O(m)$, and there are $O(m \log C)$ of them.

Bipartite Matching

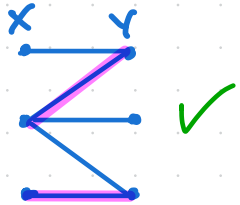
- A matching in graph $G = (V, E)$ is a set $M \subseteq E$ s.t. no vertex occurs twice in M:

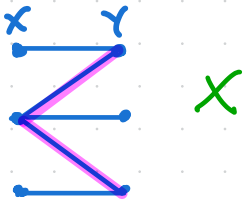
 ✓  ✗

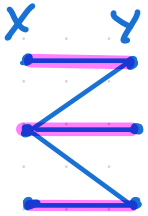
A matching M is perfect if $|M| = |V|/2$. // every vertex is matched

- A bipartite graph is a graph $G = (X \cup Y, E)$ s.t. $X \cap Y = \emptyset$ and $(u,v) \in E \Rightarrow u \in X$ and $v \in Y$ or $v \in X$ and $u \in Y$

- A matching in a bipartite graph $G = (X \cup Y, E)$ is a set $M \subseteq E$ s.t. no vertex occurs twice in M:



✓


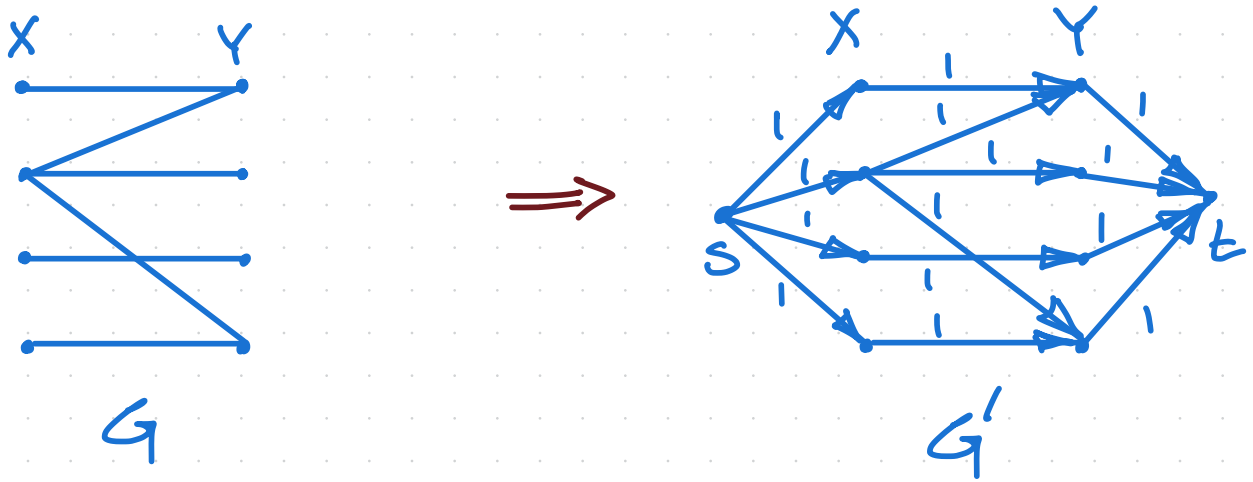
✗


A bipartite matching M is perfect if $|M| = |X| = |Y|$

- Many problems involve finding maximum or perfect matchings.

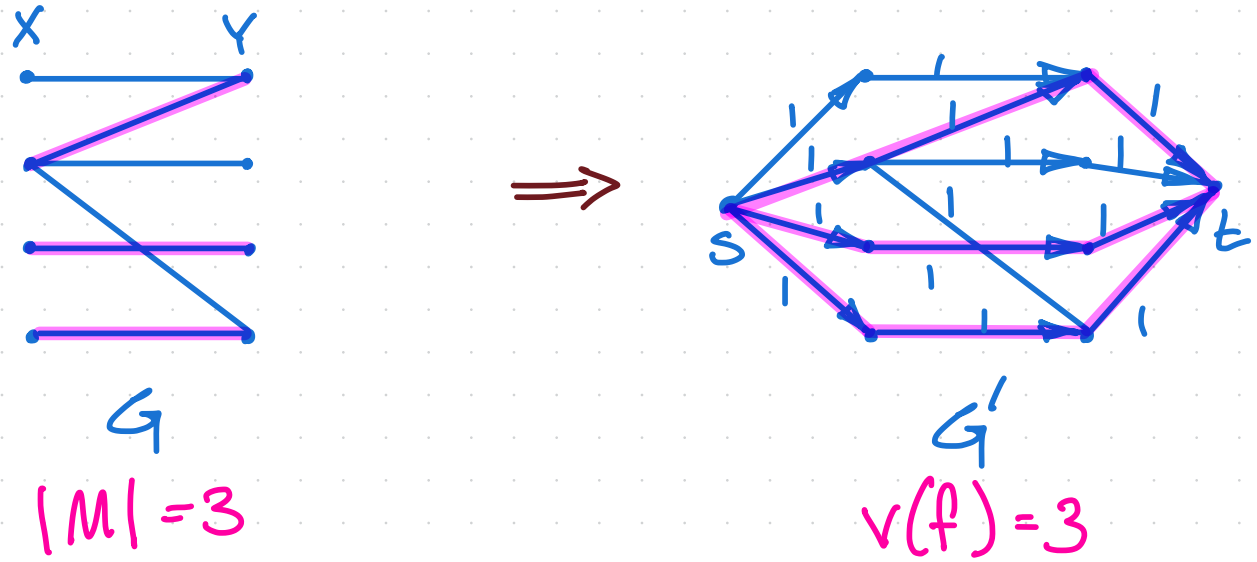
Reducing Bipartite Matching to Network Flow

- For $G = (X \cup Y, E)$, let $G' = (V, E', c)$ be the flow network
  with
  - $V = X \cup Y \cup \{s, t\}$
  - $E' = \{(u,v) : u \in X, v \in Y, (u,v) \in E\}$
    $\cup \{(s,u) : u \in X\}$
    $\cup \{(v,t) : v \in Y\}$.
  - $c_e = 1$ for each $e \in E'$



Reducing Bipartite Matching to Network Flow

- Fact: G has a matching of size k iff G' has a flow of value k.



G

$|M| = 3$

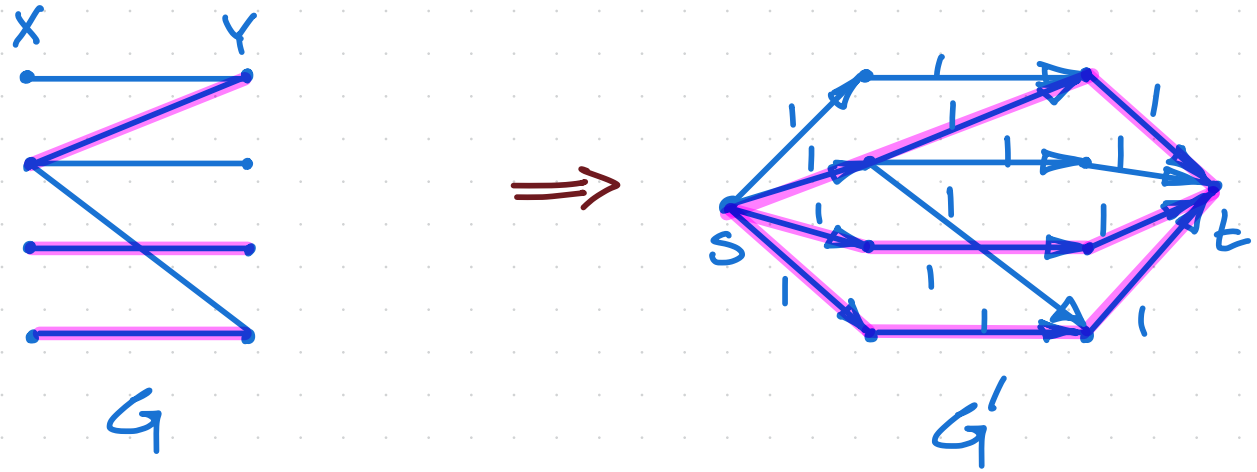
G'

$v(f) = 3$

Ex: Prove it.

Reducing Bipartite Matching to Network Flow

- Fact: G has a matching of size k iff G' has a flow of value k.



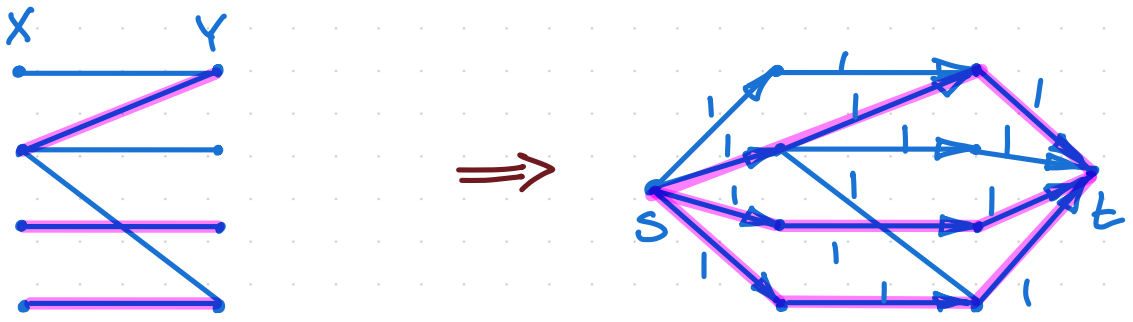
G $\qquad$ G'

$|M| = 3$ $\qquad$ $v(f) = 3$

Ex: Prove it.

Let $f: G \to G'$ as above.

1. Suppose G has a matching of size k.
   Construct a flow f with $v(f) = k$ for G'.
2. Suppose G' has a flow f with $v(f) = k$.
   Construct a matching of size k in G.



G $\qquad$ G'

$|M| = 3$ $\qquad$ $v(f) = 3$

1) i (let $L \subseteq X$ be the vertices in X that appear in M.
For each edge (s, x), with $x \in L$, let $f_{(s,x)} = 1$.

ii (Define flows on edges from X to Y.

iii (Define flows on edges from Y to t.

iv (Show the result is a flow f.

v (Show the $v(f) = k$.

2) ...

Reducing Bipartite Matching to Network Flow

Fact: We can solve the Maximum Matching and Perfect Matching problems for bipartite graphs in time $O(m \cdot n)$ using the Ford-Fulkerson algorithm.

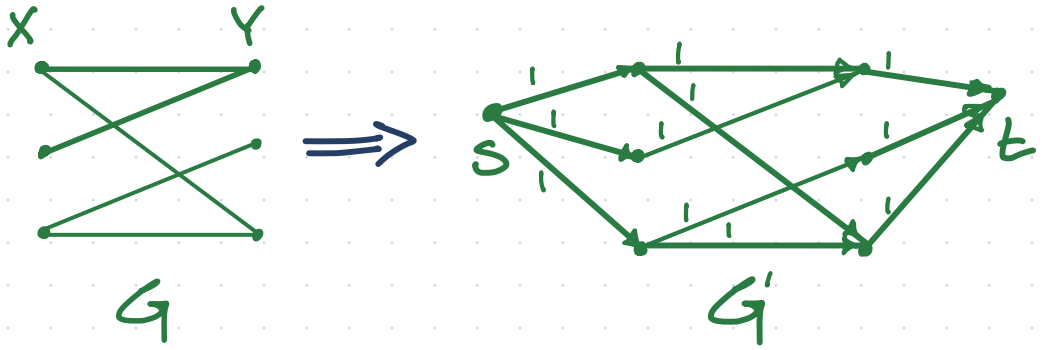
Pf: Given bipartite graph G, we can construct G' and run F.F. on G'. Let $n = |X|$.

We have:

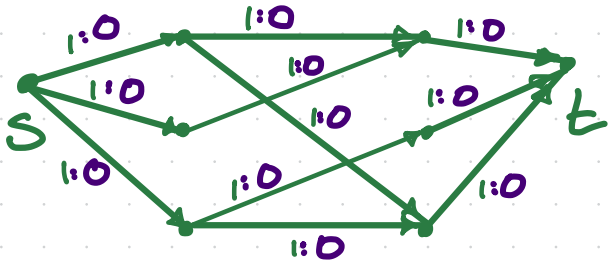
- The max. matching has size $k \Leftrightarrow$ the max flow in G' is k $\Leftrightarrow$ FF run on G' returns k.
- G has a perfect matching $\Leftrightarrow$ G' has a flow of value n $\Leftrightarrow$ FF returns n.
- FF runs in time $O(C \cdot m)$ where $C = \sum_{\substack{e \text{ out} \\ \text{of } s}} c_e$.

In G', $\sum_{\substack{e \text{ out} \\ \text{of } s}} c_e = n$, giving running time $O(m \cdot n)$ $\square$

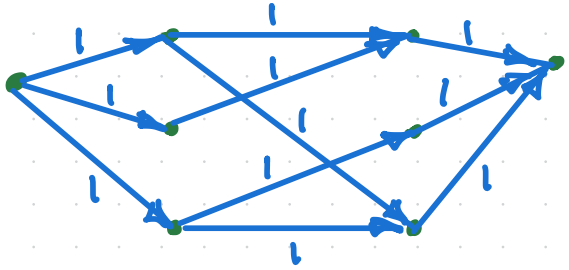
Solving Bipartite Matching with Ford-Fulkerson - Example



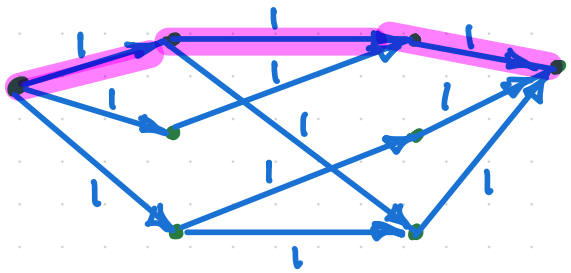
$G, f - v(f) = 0$



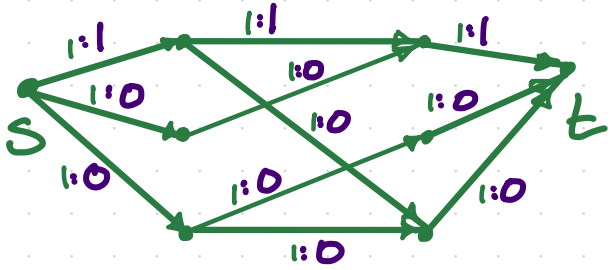
G_f



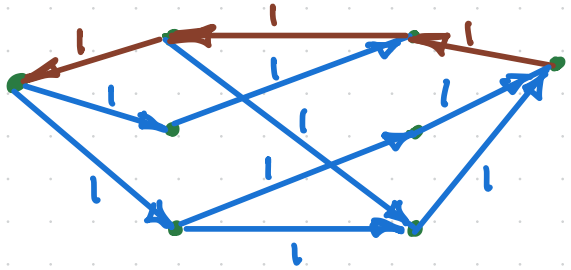
P - b = 1



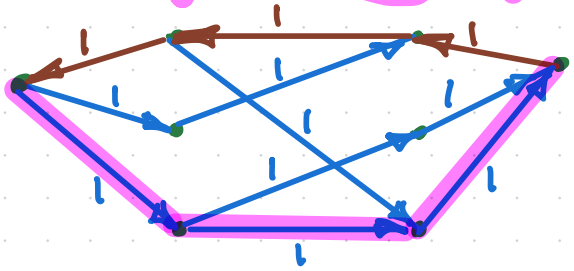
$G, f - v(f) = 1$



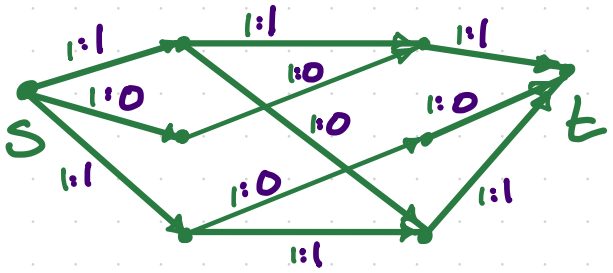
G_f



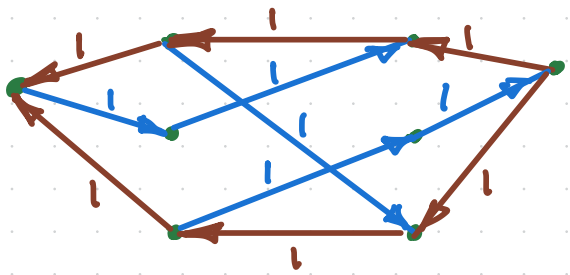
P - b = 1



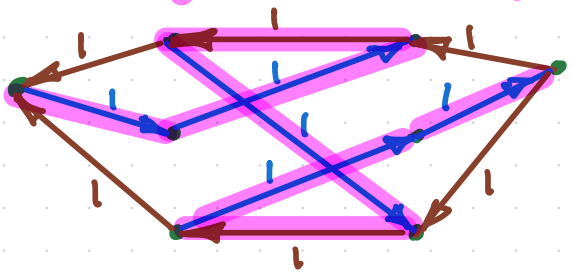
G, f — $v(f) = 2$



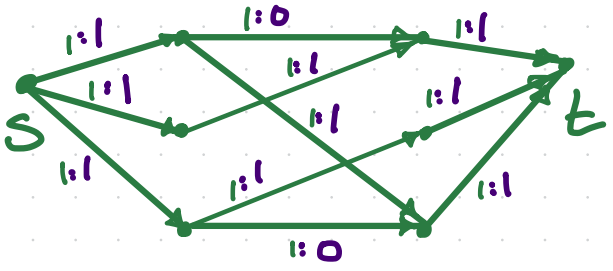
G_f



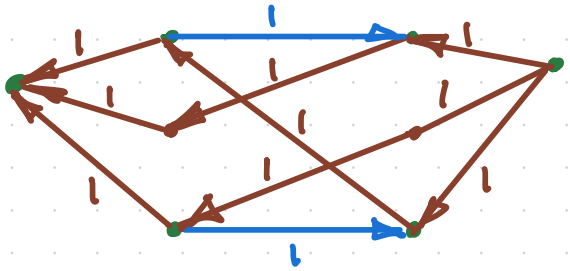
P-bal



G, f — $v(f) = 3$

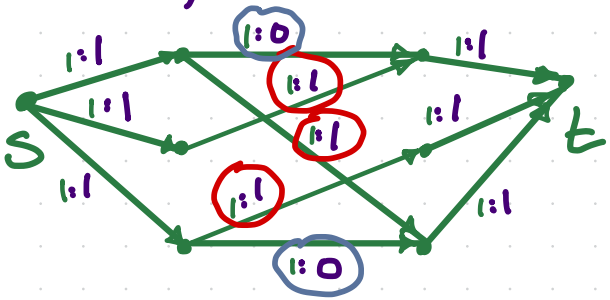


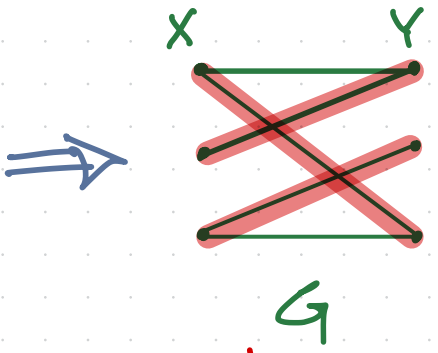
G_f



No s-t path.
f is optimal.

G, f — $v(f) = 3$





G

With a maximum matching.

End