Network Flows (Ch 7)

- **Maximum Flow Problem, Basic Notions and Technologies**
- **Ford-Falkerson Algorithm for Max-flow**
- **Applications of Max-flow**

  Matching in graphs, disjoint paths

  Circulation with Demand (Max-flow with Multiple Sources and Sinks)

  Airline scheduling, image segmentation, project selection, sports team elimination

  Minimum weight matching, minimum cost max-flow

The lecture notes/slides are adapted from those associated with the text book by J. Kleinberg and E. Tardos.

Maximum Flow Problem, Basic Notions and Technologies

- **Flow network: a digraph $G(V, E)$, each arc e has a capacity $c(e) \geq 0$, a source node $s \in V(G)$ and a sink node $t \in V(G)$. Nodes other than s and t are called internal nodes.**
- **Flow: a function $f : E \to R^+$ s.t.**
  - **(Capacity condition) for $e \in E(G), 0 \leq f(e) \leq c(e)$;**
  - **(Conservation condition) for any node $v \in V(G) \setminus \{s, t\}$,**

$$\sum_{e=(u,v)\in E(G)} f(e) = \sum_{e=(v,w)\in E(G)} f(e).$$

The value of the flow is

$$\mathrm{val}(f) = \sum_{e=(s,u)\in E(G)} f(e) - \sum_{e=(u,s)\in E(G)} f(e).$$

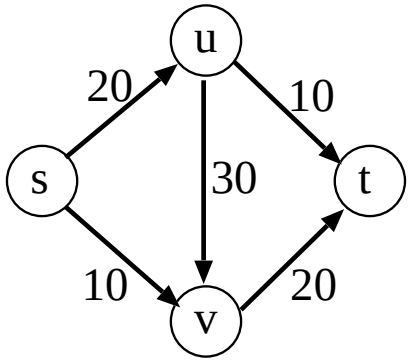
- **Maximum flow problem: find a flow on G of maximum value,**

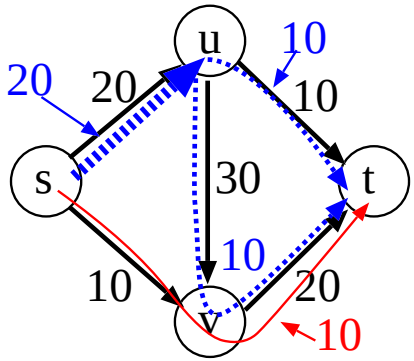
$$\max \quad \mathrm{val}(f) = \sum_{e=(s,u)\in E(G)} f(e) - \sum_{e=(u,s)\in E(G)} f(e) \quad \textbf{subject to}$$

$$\forall e \in E(G), 0 \le f(e) \le c(e) \quad \textbf{Capacity condition}$$

$$\forall v \in V(G) \setminus \{s,t\},$$

$$\sum_{e=(u,v)\in E(G)} f(e) - \sum_{e=(v,w)\in E(G)} f(e) = 0 \quad \textbf{Conservation condition.}$$

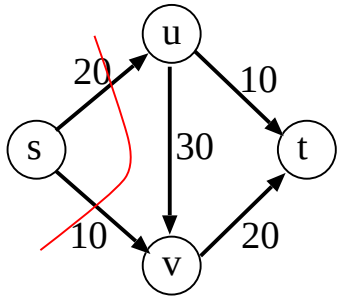




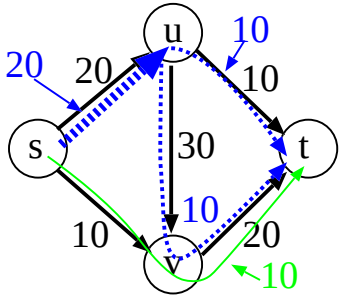
Max-flow=30

Minimum cut

- **A st-cut (cut) of flow network G is a subset C of $E(G)$ that partitions $V(G)$ into two sets A and B s.t. $s \in A, t \in B$ and any path $s \to t$ has an arc in C. We denote C by (A, B).**
- **The capacity of cut (A, B) is $c(A, B) = \sum_{\substack{e=(u,v) \\ u \in A, v \in B}} c(e)$.**
- **Minimum cut problem: find a cut of minimum capacity**
- **Value of a max-flow = capacity of a min-cut (Max-flow min-cut theorem).**

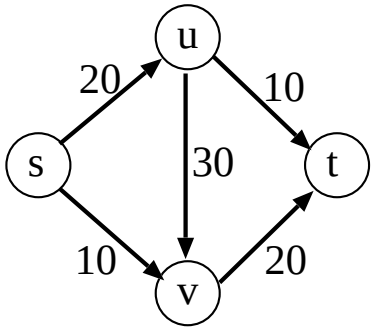


Cut (A,B), A={s},B={u,v,t}
capacity of (A,B): 30

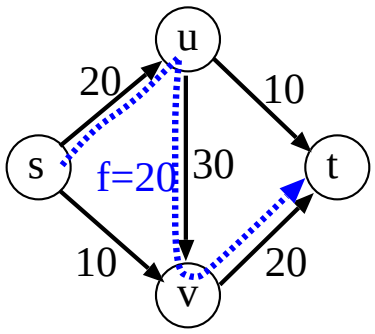


Max-flow=30

- **A simple greedy algorithm for max-flow**
  - **Start from a flow $f(e) = 0$ for each $e \in E(G)$.**
  - **Find a $s \to t$ path P with $f(e) < c(e)$ for $e \in P$.**
  - **Augment flow f along path P.**
  - **Repeat the above until no improvement possible.**
- **Simple greedy may not find the max-flow. A problem is that once a flow on an edge is increased, the flow is never decreased.**



Flow network

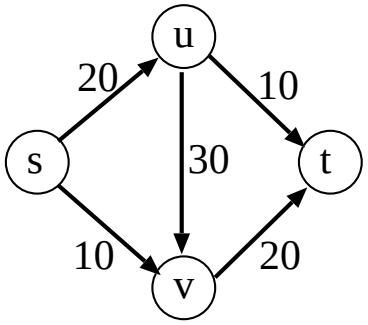


A flow of value 20

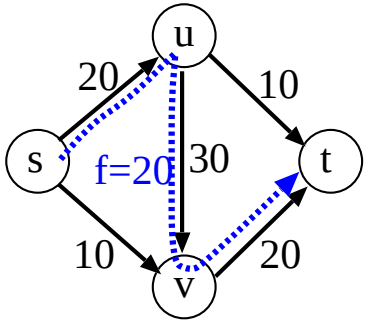
Residual graph

- **Given flow network G and flow f, the residual graph G_f w.r.t. f:**
  - $V(G_f) = V(G)$;
  - **for each arc $e = (u, v) \in E(G)$ with $f(e) < c(e)$, $(u, v) \in E(G_f)$ with capacity $c(e) - f(e)$ (forward arc);**
  - **for each arc $e = (u, v) \in E(G)$ with $f(e) > 0$, $(v, u) \in E(G_f)$ with capacity $f(e)$ (backward arc).**

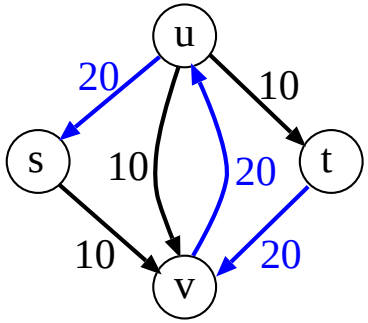
Capacity of arcs in G_f is called residual capacity.



Flow network



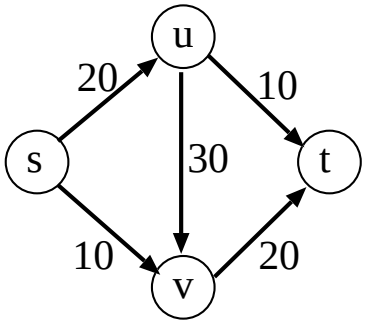
A flow of value 20



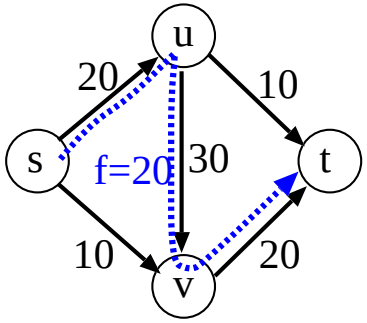
Residual graph
black: forward arc
blue: backward arc

Augmenting path

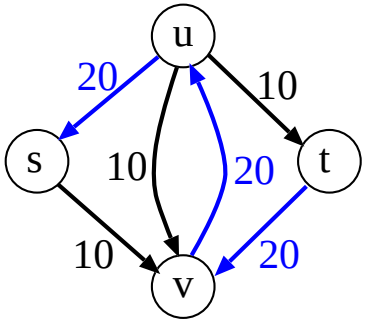
- **Augmenting path: a simple $s \to t$ path in the residual network G_f.**
- **The bottleneck capacity of an augmenting path P is the minimum residual capacity of any arc in P, denoted by $\text{bottleneck}(f, P)$ (δ).**



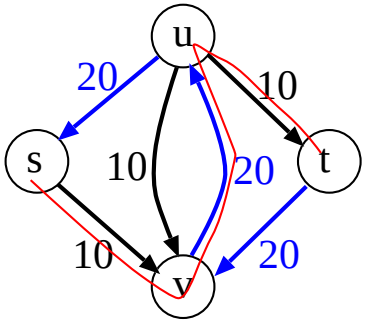
Flow network G



A flow f of value 20



Residual graph G_f
black: forward arc
blue: backward arc



Augmnting path P
δ=10

Augment(f, P)

$\delta = \text{bottleneck}(f, P)$;

for each arc $e \in E(G)$ **do** $f'(e) = f(e)$;

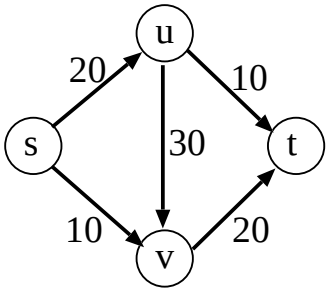
for each arc $e = (u, v) \in P$ **do**

if e **is a forward arc in** G_f **then increase** $f'(e)$ **by** δ

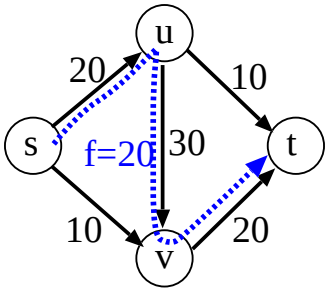
else decrease $f'(e)$ **by** δ

Return f'

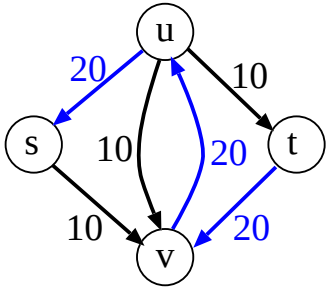
- **Key property: Let f be a flow in G and P be an augmenting path in G_f. Then f' computed by the algorithm Augment(f, P) is a flow in G.**



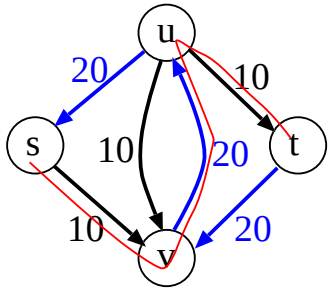
Flow network G



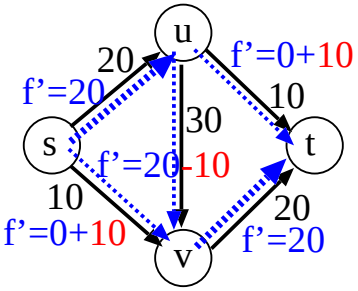
A flow f of value 20



Residual graph G_f
black: forward arc
blue: backward arc



Augmnting path P
δ=10



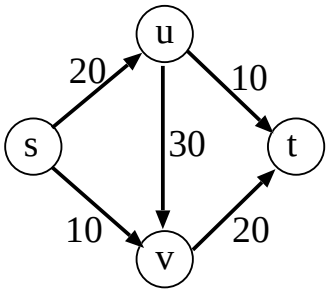
Function f' computed by Augment algorithm is a flow in G.

Proof. For capacity condition, it suffices to check every arc e of P only. By definition, $\delta = \text{bottleneck}(f, P) \leq$residual capacity of every $e \in P$. If e is a forward arc, then

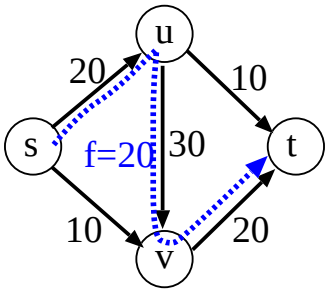
$$0 \leq f(e) \leq f'(e) = f(e) + \delta \leq f(e) + (c(e) - f(e)) = c(e).$$

If e is a backward arc, then

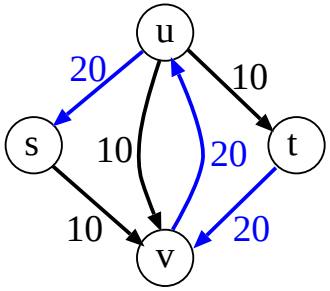
$$c(e) \geq f(e) \geq f'(e) = f(e) - \delta \geq f(e) - f(e) = 0$$



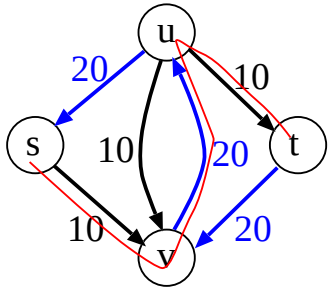
Flow network G



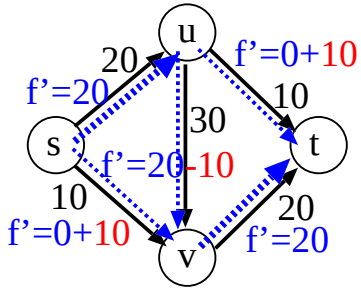
A flow f of value 20



Residual graph G_f
black: forward arc
blue: backward arc



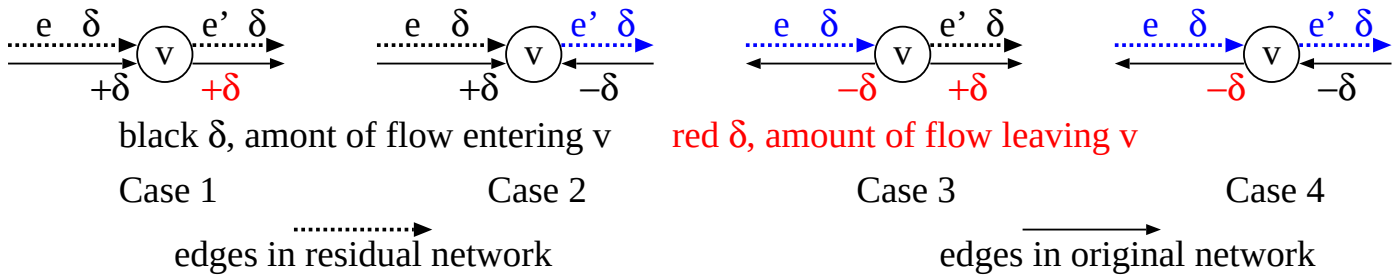
Augmnting path P
δ=10



For conservation condition, we check that for each node v in P, the additional amount of flow, 0 or δ or $-\delta$, entering v equals to the additional amount of flow, 0 or δ or $-\delta$, leaving v. Let $(u, v)(v, w)$ (denote (u, v) by e and (v, w) by e') be the subpath of P containing v. There are four cases:

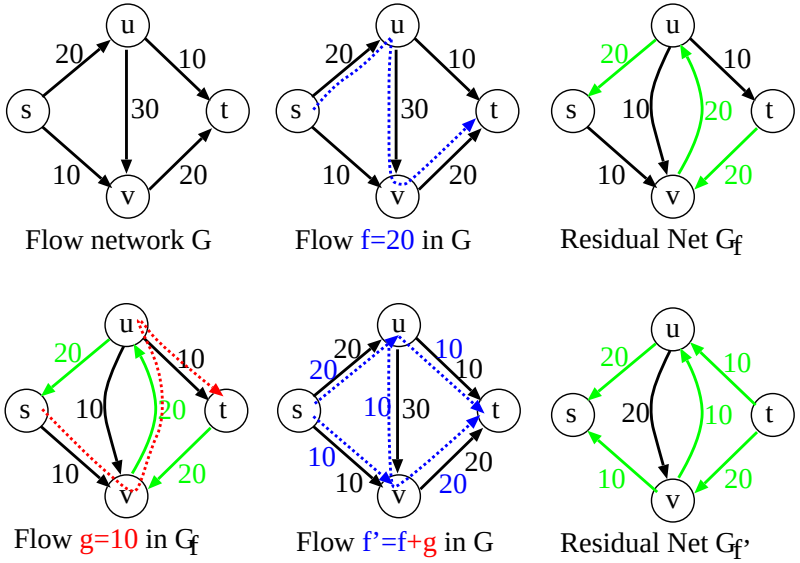
1. e is forward edge and e' is forward edge, δ additional amount of flow entering v and δ additional flow leaving v.
2. e is forward edge and e' is backward edge, 0 additional amount of flow entering v and 0 additional flow leaving v.
3. e is backward edge and e' is forward edge, 0 additional amount of flow entering v and 0 additional flow leaving v.
4. e is backward edge and e' is backward edge, $-\delta$ additional amount of flow entering v and $-\delta$ additional flow leaving v.

□



Example of improving flow by residual graph and augmenting path

- **Start with a flow f along path $(s,u),(u,v),(v,t)$ in G with $f(s,u)=f(u,v)=f(v,t)=20$.**
- **Construct the residual graph G_f w.r.t. f.**
- **Find a flow g along augmenting path $(s,v),(v,u),(u,t)$ in G_f with $g(s,v)=g(v,u)=g(u,t)=10$.**
- **Construct the residual graph $G_{f'}$ w.r.t. $f'=f+g$.**
- **No flow>0 can be found in $G_{f'}$, return f'.**



Ford-Falkerson algorithm [Ford-Fulkerson 1955]

- **Start with $f(e) = 0$ for $e \in E(G)$.**
- **Improve f on an augmenting path $s \to t$ in residual graph G_f.**
- **Repeat the above until no augmenting path can be found.**

Max-Flow(f, P)
Input: A flow network G.
Output: A max-flow f in G.
  for each arc $e \in E(G)$ do $f(e) = 0$;
  while $\exists$ $s \to t$ path in residual graph G_f do
    Let P be a simple $s \to t$ path in G_f;
    $f' =$Augment(f, P); $G_f = G_{f'}$; $f = f'$;
  end while
  Return f

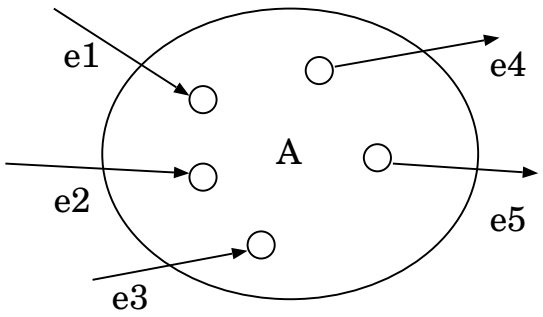
Max-flow vs min-cut

- **For a flow f in G and $A \subset V(G)$,**

  $f^{out}(A) = \sum_{\substack{e=(u,v) \\ u \in A, v \notin A}} f(e)$,

  $f^{in}(A) = \sum_{\substack{e=(v,w) \\ v \notin A, w \in A}} f(e)$.

  For $A = \{v\}$, denote $f^{out}(A)$ by $f^{out}(v)$ and $f^{in}(A)$ by $f^{in}(v)$.



$f^{out}(A)=f(e4)+f(e5)$

$f^{in}(A)=f(e1)+f(e2)+f(e3)$

For any flow f and any cut (A, B), $\mathrm{val}(f) = f^{out}(A) - f^{in}(A)$.

Proof. For any $v \notin \{s, t\}$, $f^{out}(v) - f^{in}(v) = 0$ (conservation condition) and $\mathrm{val}(f) = f^{out}(s) - f^{in}(s)$.

$\sum_{v \in A} f^{out}(v) - f^{in}(v) = f^{out}(s) - f^{in}(s) + \sum_{v \in A \setminus \{s\}} f^{out}(v) - f^{in}(v) = \mathrm{val}(f)$.

Let $f(e)$ be a flow on arc (u, v) with $u \in A$ or $v \in A$. The contribution by $f(e)$ to $\sum_{v \in A} f^{out}(v) - f^{in}(v)$:

- 0 for $u, v \in A$ as $f(e)$ is included in $f^{out}(u)$ and in $-f^{in}(v)$
- $f(e)$ for $u \in A$ and $v \in B$ as $f(e)$ is included in $f^{out}(u)$ only
- $-f(e)$ for $v \in A$ and $u \in B$ as $f(e)$ is included in $-f^{in}(v)$ only.

$$\begin{aligned} \mathrm{val}(f) &= \sum_{v \in A} f^{out}(v) - f^{in}(v) \\ &= \sum_{\substack{e=(u,v) \\ u \in A, v \in B}} f(e) - \sum_{\substack{e=(v,w) \\ v \in B, w \in A}} f(e) = f^{out}(A) - f^{in}(A). \end{aligned}$$

□

- **Weak duality. For any flow f and any cut (A, B), $\mathrm{val}(f) \leq c(A, B)$.**

  Proof. $\mathrm{val}(f) = f^{out}(A) - f^{in}(A) \leq f^{out}(A) \leq c(A, B)$. □

- **Max-flow min-cut theorem. Value of a max-flow equals to capacity of a min-cut.**

  Proof. Claim: For any flow f and any cut (A, B), if $\mathrm{val}(f) = c(A, B)$ then f is a max-flow and (A, B) is a min-cut. By weak duality,

  for any flow f', $\mathrm{val}(f') \leq c(A, B) = \mathrm{val}(f)$;

  for any cut (A', B'), $c(A, B) = \mathrm{val}(f) \leq c(A', B')$.

  So the claim is true. From the claim, the theorem holds. □

- **Augmenting path theorem. Let f be a flow s.t. there is no $s \to t$ path in G_f. Then f is a max-flow.**

  Proof. We show that there is a cut (A, B) s.t. $\mathrm{val}(f) = c(A, B)$ and then the theorem follows from the max-flow and min-cut theorem.

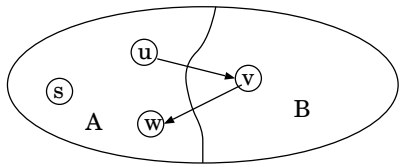
  Let $A = \{u | u$ is reachable from s in $G_f\}$ and $B = V(G) \setminus A$. Then (A, B) is a cut.

  For any arc $e = (u, v) \in E(G)$ with $u \in A$ and $v \in B$, $f(e) = c(e)$ because otherwise (u, v) would be a forward arc in G_f, a contradiction to the choice of A.

  For any arc $e' = (v, w) \in E(G)$ with $v \in B$ and $w \in A$, $f(e') = 0$ because otherwise (w, v) would be a backward arc in G_f, a contradiction to the choice of A. Therefore,

$$\begin{aligned}
\mathrm{val}(f) &= \sum_{v \in A} f^{out}(v) - f^{in}(v) \\
&= \sum_{\substack{e=(u,v) \\ u \in A, v \in B}} f(e) - \sum_{\substack{e=(v,w) \\ v \in B, w \in A}} f(e) = \sum_{\substack{e=(u,v) \\ u \in A, v \in B}} c(e) - 0 = c(A, B).
\end{aligned}$$

□



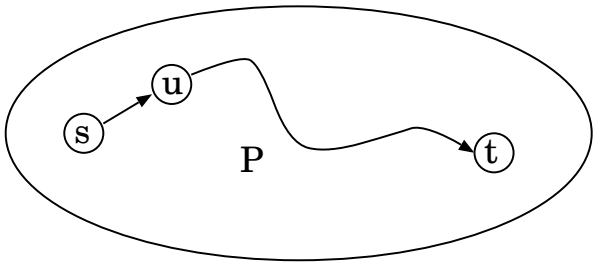
Theorem. [Ford-Fulkerson 1955] For a flow network G with integer capacities, let $C = \sum_{e=(s,v)\in E(G)} c(e)$ be the total capacity of arcs leaving s. Ford-Falkerson algorithm computes a max-flow in $O(mC)$ time.

Proof. From the augmenting path theorem, the algorithm computes a max-flow when it terminates.

(1) For a flow f in G, let $\mathrm{val}(f) = \sum_{e=(s,v)\in E(G)} f(e)$ be the value of f. Then, at each step of the algorithm, $\mathrm{val}(f)$ is an integer.

(2) Let P be an augmenting path in G_f. Then the arc $e = (s, u)$ leaving s in P is a forward arc, implying $f'(e) = f(e) + \mathrm{bottleneck}(P, f)$. Further, e is the only arc in P leaving s, implying $\mathrm{val}(f') = \mathrm{val}(f) + \mathrm{bottleneck}(P, f) > \mathrm{val}(f)$ since $\mathrm{bottleneck}(P, f) > 0$.

From (1) and (2), the algorithm terminates in at most C iterations of while loop. In each iteration of while loop, G_f can be computed in $O(m)$ time since $|E(G_f)| \leq 2m$. P can be found in $O(m)$ time by BFS. Augmenting takes $O(m)$ time. □

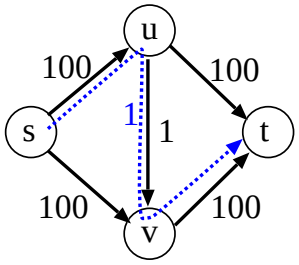


Max-flow and min-cut properties

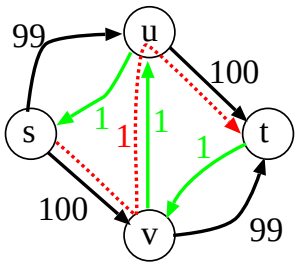
- **Ford-Falkerson algorithm computes a max-flow if all arc capacities are integer.**
- **In a flow network, the value of a max-flow equals to the capacity of a min-cut (max-flow min-cut theorem).**
- **In a flow network, a flow f is a max-flow iff there is no augmenting path (augmenting path theorem).**
- **Given a max-flow, a min-cut can be computed in $O(m)$ time.**
- **If all arc capacities in a flow network are integers, then there is a max-flow f with $f(e)$ an integer for every arc e.**

Faster algorithm for Max-Flow [Karp 1972, Dinitz 1970]

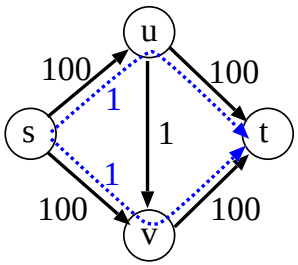
- **Running time $O(mC)$ may not be polynomial in an input instance size.**
- **Select an augmenting path with a large bottleneck can help.**
- **For a scaling parameter Δ, let $G_f(\Delta)$ be the subgraph of the residual graph G_f consisting of arcs with residual capacity$\geq \Delta$.**



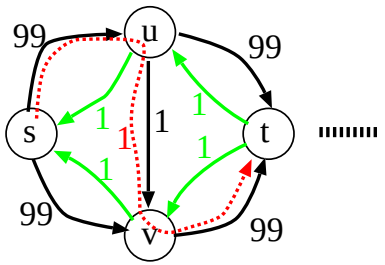
Flow network G
and flow f=1



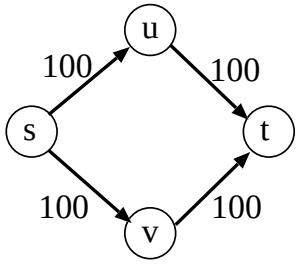
Residual net G_f
and flow g=1



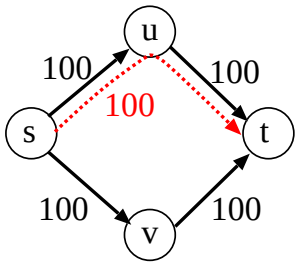
Flow network G
and flow f=2



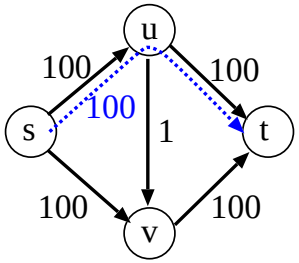
Residual net G_f
and flow g=1



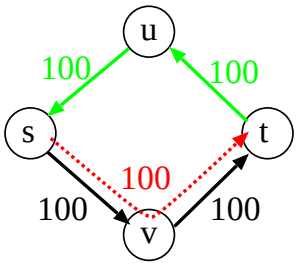
$G_f(\Delta)$ for f=0 and Δ=2



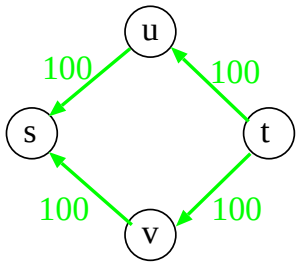
g=100



Flow network G
and flow f=100



Residual net $G_f(\Delta)$
and flow g=100



Scaling-Max-Flow(G)

Input: A flow network G.

Output: A max-flow f in G.

for each arc $e \in E(G)$ **do** $f(e) = 0$;

$\Delta = max\{2^i | 2^i \leq \max_{e=(s,u) \in E(G)} c(e)\}$;

while $\Delta \geq 1$ **do**

while $\exists\ s \rightarrow t$ path P in residual graph $G_f(\Delta)$ **do**

$f' =$Augmenting(f, P); $G_f = G_{f'}$; $f = f'$;

end while

$\Delta = \Delta/2$;

end while

Return f

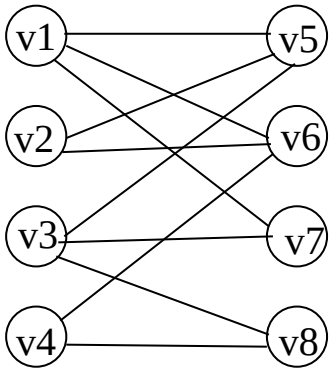
For a flow network G with m arcs and integer capacities, Scaling-Max-Flow algorithm finds a max-flow in a most $2m(1 + \lceil \log C \rceil)$ augmentations, $C = \max_{e \in E(G)} c(e)$. The algorithm can be implemented in $O(m^2 \log C)$ time.

Applications of Max-Flow

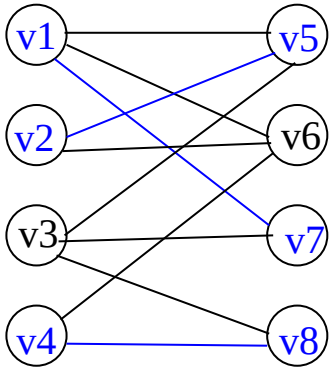
- **Matching problem in graphs**
- **Disjoint paths problem in graphs**
- **Circulation with demand problem (flow with multiple sources/sinks)**
- **Airline scheduling problem**
- **Image segmentation problem**
- **Project selection problem**
- **Sports team elimination problem**
- **Minimum weight matching problem**
- **Minimum cost max-flow problem**

Matching Problem in Graphs

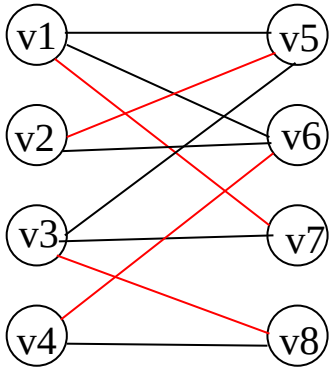
- **A matching M of a graph G is a set of edges s.t. each node of G is incident to at most one edge in M. A matching M is perfect if each node of G is incident to one edge of M.**
- **Bipartite graph G, V can be partitioned into two subsets X and Y s.t. for each edge $\{u, v\}$ of G, $u \in X$ and $v \in Y$. X and Y are called a bipartition of G.**
- **Bipartite matching problem: find a matching in a bipartite G of maximum size.**



A bipartite graph G



Blue edges form
a matching

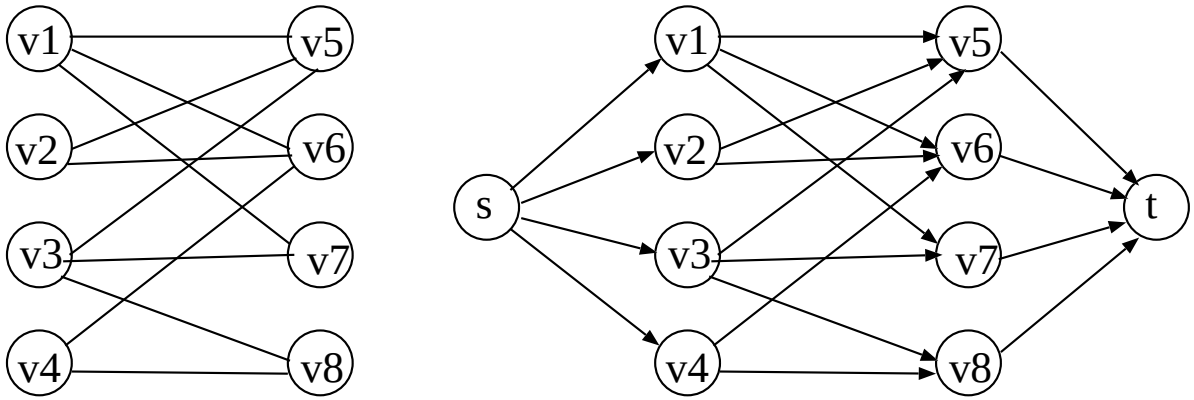


Red edges form
a perfect matching

Network flow algorithm for bipartite matching

Reduce the bipartite matching problem in G with bipartition X and Y to network flow problem in G' constructed below:

- **orient every edge from X to Y;**
- **add source s and sink t;**
- **add an arc (s, u) for every $u \in X$ and an arc (v, t) for every $v \in Y$;**
- **set the capacity of every arc to 1.**



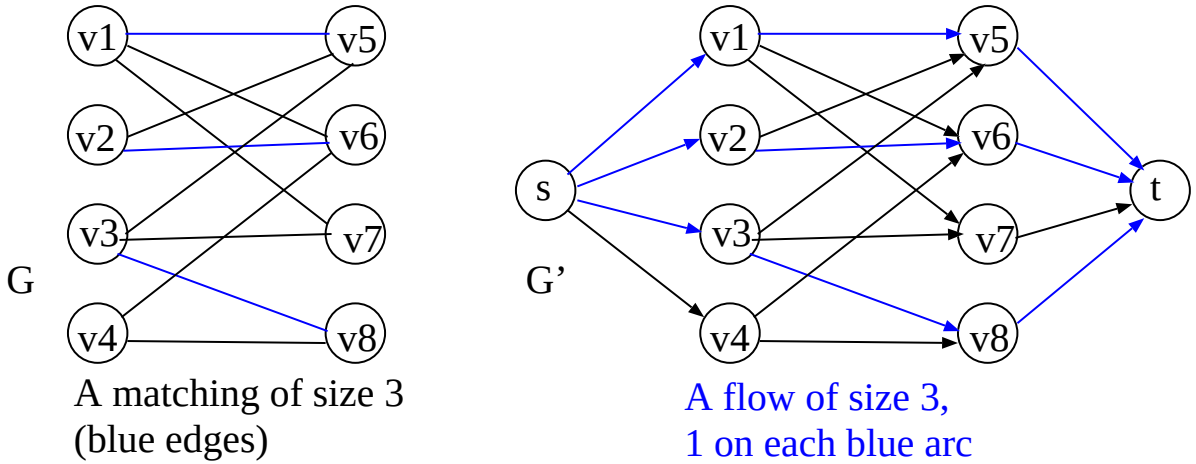
Convert bipartite matching problem to max-flow problem

There is a matching of size k in G iff there is a flow in G' of value k.

Proof. Let M be a matching in G of size k and $V(M)$ be the set of end nodes of edges in M. Let $f : E(G) \to R^+$ be defined as $f(e) = 1$ for $e = \{s, u\}$ with $u \in X \cap V(M)$, $e \in M$ and $e = \{v, t\}$ with $v \in Y \cap V(M)$; and $f(e) = 0$ for other arcs e. Then f is a flow of value k.

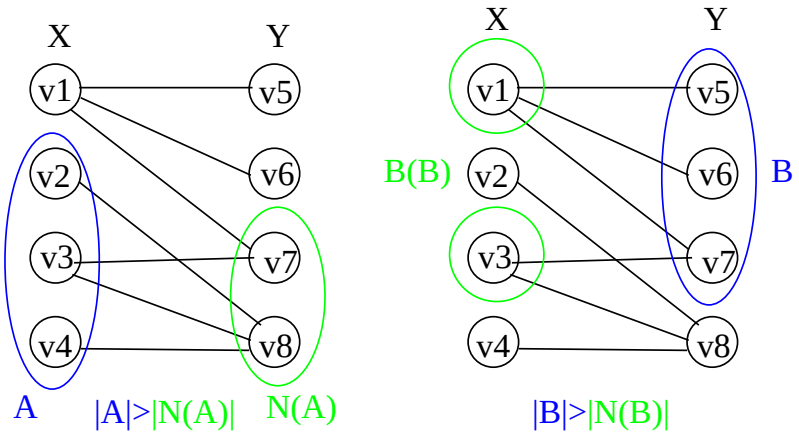
Let f be a flow of value k in G'. Since arc capacities in G' are integers, k is an integer and $f(e)$ is either 0 or 1 for each arc e. Let M be the set of arcs (u, v) with $f(e) = 1$, $u \in X$ and $v \in Y$. Then $|M| = k$ and M is a matching. □

Bipartite matching problem can be solved in $O(mn)$ time.



Perfect matching in bipartite graph

- **For G with bipartition X and Y, if G has a perfect matching, then $|X| = |Y|$.**
- **For $A \subseteq V(G)$, let $N(A) = \{v|(u,v) \in E(G), u \in A\}$ be the set of neighbor nodes of A. If there is $A \subseteq X$ (or $B \subseteq Y$) s.t. $|A| > |N(A)|$ ($|B| > |N(B)|$), then G does not have a perfect matching.**



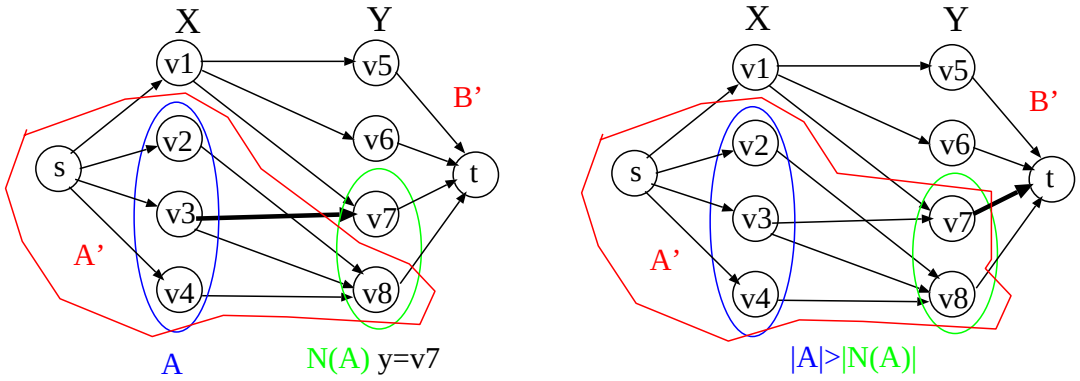
Theorem. [Frobenius 1917, Hall 1935] If $|X| = |Y|$, for any $A \subseteq X$ $|A| \leq |N(A)|$, and for any $B \subseteq Y$ $|B| \leq |N(B)|$, then G has a perfect matching.

Proof. Assume a maximum matching M in G has size $|M| = k < n = |X|$. We find a subset $A \subseteq X$ s.t. $|A| > |N(A)|$ to get a contradiction.

A max-flow in G' has value $k < n$. There is a min-cut (A', B') with capacity k and $s \in A'$.
Let $A = A' \cap X$. If $\exists y \in B' \cap N(A)$, then $(A' \cup \{y\}, B' \setminus \{y\})$ is a new cut with capacity k.
Take this new cut as (A', B'), repeat above to get a cut (A', B') with capacity k and $N(A) \subseteq A'$.
For every arc $e = (u, v)$ with $u \in A'$ and $v \notin A'$, either $e = (s, v)$ or $e = (u, t)$.
Then $c(A', B') = |X \cap B'| + |Y \cap A'|$.
Since $|X \cap B'| = n - |A|$ and $|Y \cap A'| = |N(A)|$, $c(A', B') = k < n$ implies

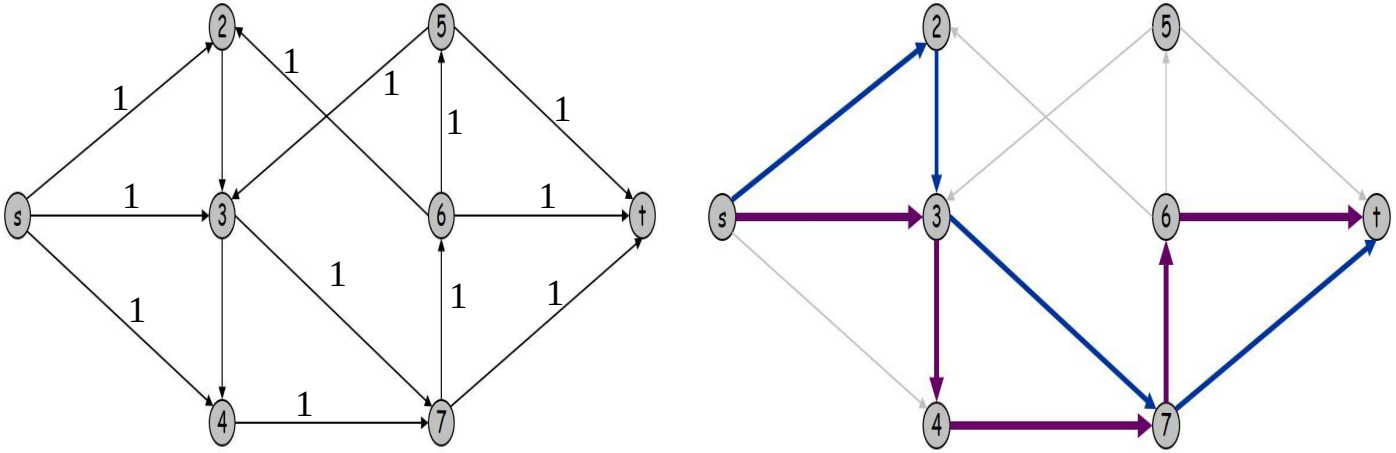
$$c(A', B') = |X \cap B'| + |Y \cap A'| = n - |A| + |N(A)| < n.$$

We get $|A| > |N(A)|$, a contradiction. □



Disjoint Paths

- **A set of paths are edge-disjoint if they do not have a common edge (arc).**
- **Directed $s \to t$ edge-disjoint paths problem: Given two nodes s and t in a digraph G, find a maximum number of edge-disjoint paths from s to t.**
- **Given a disjoint paths instance (G, s, t), we construct a max-flow instance (G, s, t), s is the source, t is the sink and each arc of G has capacity 1.**



- **The maximum number edge-disjoint $s \to t$ paths equals the max-flow value.**

  Proof. Assume there are k edge-disjoint $s \to t$ paths. Let $f(e) = 1$ for each arc e in the disjoint paths and $f(e) = 0$ otherwise. Then f is a flow of value k.

  Assume there is a flow f of value k s.t. $f(e) = 0, 1$. We show there are k edge-disjoint $s \to t$ paths by induction. Induction base $k = 0$ holds.

  Induction hypothesis: assume the statement is true for $k - 1 \geq 0$.

  Induction step: Let f be a flow of value k. We find a sequence of arcs, starting from s: Take any arc $e = (s, u)$ with $f(e) = 1$. By conservation property, there is an arc $e_1 = (u, v)$ with $f(e_1) = 1$. Continue until either we (1) reach t or (2) reach a node v for the second time.

  For (1), we get a path P from s to t. Set $f(e) = 0$ for each $e \in P$. We get a flow of value $k - 1$. By the induction hypothesis, there are $k - 1$ disjoint paths from s to t. Further the $k - 1$ disjoint paths do not have a common arc with P. Thus, there are k disjoint paths from s to t.

  For (2), remove the cycle between the two appearances of v to get a path P. □

- **The induction procedure above is called path decomposition.**

Find directed edge-disjoint paths

- **Algorithm**

  Apply Ford-Fulkerson algorithm to find a max-flow f.

  Apply the path decomposition procedure to find the disjoint paths

- **A maximum set of edge-disjoint paths from s to t in a digraph can be computed in (mn) time.**

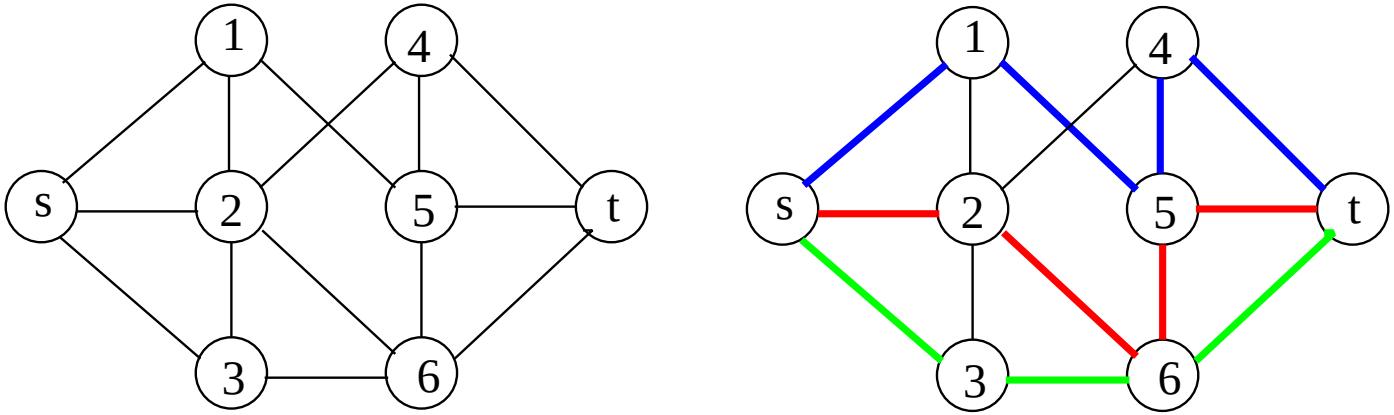
- **Network connectivity: Given a digraph $G(V, E)$ and two nodes s, t, find a minimum number of arcs whose removal disconnects t from s.**
- **Theorem. [Menger 1927] The maximum number of edge-disjoint $s \to t$ paths equals the minimum number of edges whose removal disconnects t from s.**

  Proof. Let C be a set of arcs whose removal disconnects t from s and $|C| = k$. Then each $s \to t$ has an arc in C, implying the number of edge-disjoint $s \to t$ paths is at most k.

  Assume the maximum number of edge-disjoint $s \to t$ paths is k. Then the max-flow value is k, implying a min-cut C of size k. Then removing C disconnects t from s. □

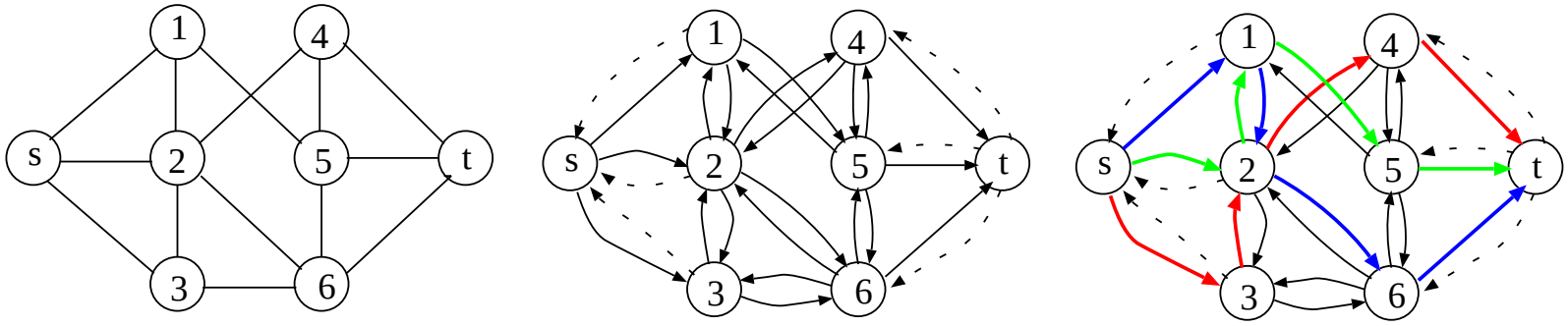
Undirected edge-disjoint paths

- **Undirected edge-disjoint $s-t$ paths problem: Given two nodes s and t in a graph G, find a maximum number of edge-disjoint paths between s and t.**



Undirected disjoint paths vs flows

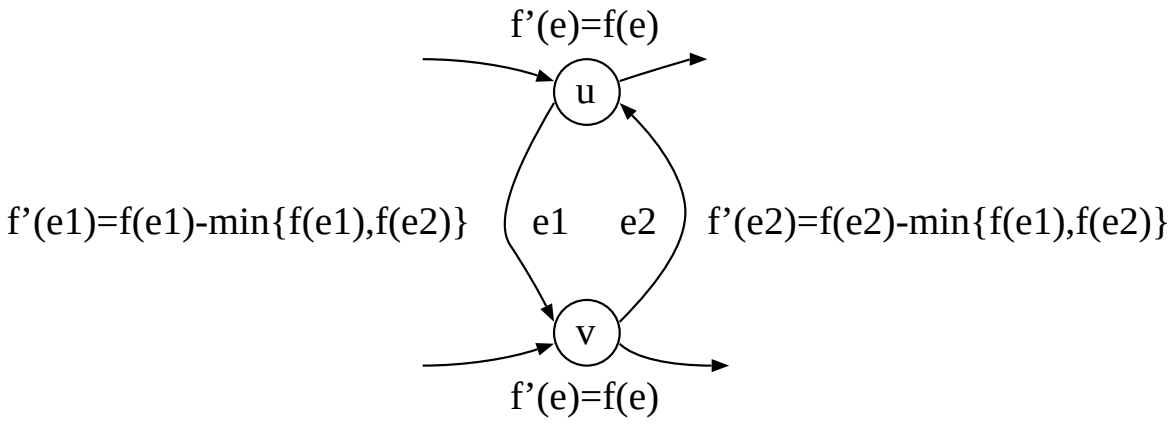
- **Reduce the problem in graph G to the problem in digraph $\vec{G}$:**
  - **Replace every edge $\{u, v\}$ by two arcs (u, v) and (v, u), and remove every arc (u, s) and every arc (t, v) to get $\vec{G}$.**
  - **Find edge-disjoint paths in $\vec{G}$ by flow algorithms.**
  - **Replace each of arcs (u, v) and (v, u) in the paths in $\vec{G}$ by edge $\{u, v\}$ to get paths in G.**
- **Hurdle: edge-disjoint paths in $\vec{G}$ may have arcs (u, v) and (v, u). Such a set of paths do not give a set of edge-disjoint paths in G.**



- **For any flow network, there is a max-flow f s.t. for every pair of opposite arcs $e_1 = (u, v)$ and $e_2 = (v, u)$, either $f(e_1) = 0$ or $f(e_2) = 0$.**

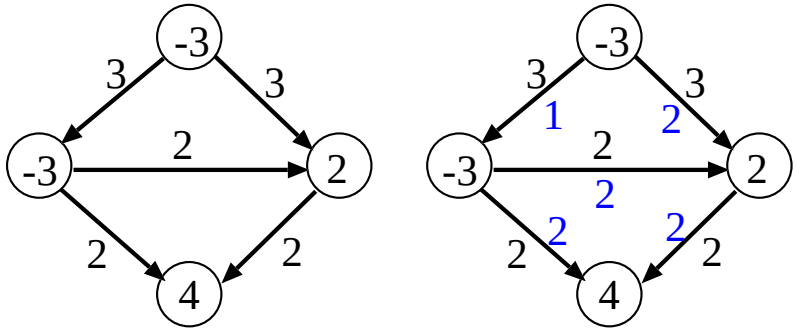
  Proof. Assume for a max-flow f, $f(e_1) > 0$ and $f(e_2) > 0$ for some $e_1 = (u, v)$ and $e_2 = (v, u)$. Let f' be a function with $f'(e_1) = f(e_1) - \min\{f(e_1), f(e_2)\}$, $f'(e_2) = f(e_2) - \min\{f(e_1), f(e_2)\}$ and $f'(e) = f(e)$ for $e \neq e_1, e_2$. Then f' is a flow, $\mathrm{val}(f') = \mathrm{val}(f)$ and either $f'(e_1) = 0$ or $f'(e_2) = 0$. □

- **Undirected $s - t$ edge-disjoint paths problem can be solved in $O(mn)$ time.**



Circulation with Demands

- **In a digraph G, each arc e has a capacity $c(e) \geq 0$ and each node v has a demand $d(v)$. If $d(v) > 0$, then v is a sink and requires the flow entering v greater than the flow leaving v. If $d(v) < 0$, then v is a source and requires the flow leaving v greater than the flow entering v.**
- **A feasible circulation with demands $\{d(v)|v \in V(G)\}$ is a function $f : E(G) \to R^+$ s.t. for each arc $e \in E(G)$, $0 \leq f(e) \leq c(e)$ (capacity condition), and for each node v, $f^{in}(v) - f^{out}(v) = d(v)$ (demand condition).**
- **Problem: Given an instance G, is there a feasible circulation in G?**



A necessary condition: If there is a feasible circulation in G, then $\sum_{v \in V(G)} d(v) = 0$.

Proof. Assume f is a feasible circulation. Then

$$\sum_{v \in V(G)} d(v) = \sum_{v \in V(G)} (f^{in}(v) - f^{out}(v)) = \sum_{e \in E(G)} (f(e) - f(e)) = 0.$$

□

Circulation vs flow

- **Given G, let $S = \{u|d(u) < 0\}$ be the set of sources, $T = \{v|d(v) > 0\}$ the set of sinks, and $D = \sum_{v \in T} d(v) = \sum_{u \in S} -d(u)$.**
- **Construct a flow network G' by adding a new source s with $d(s) = -D$, an arc (s, u) for each $u \in S$ with capacity $-d(u)$, a new sink t with demand D and an arc (v, t) for each $v \in T$ with capacity $d(v)$.**
- **G has a feasible circulation iff G' has a max-flow of value D.**

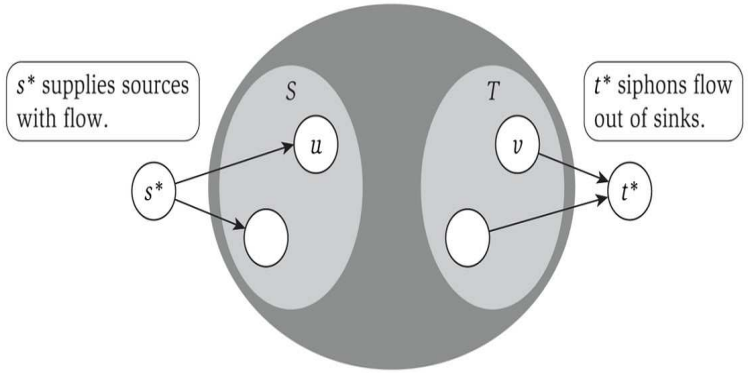
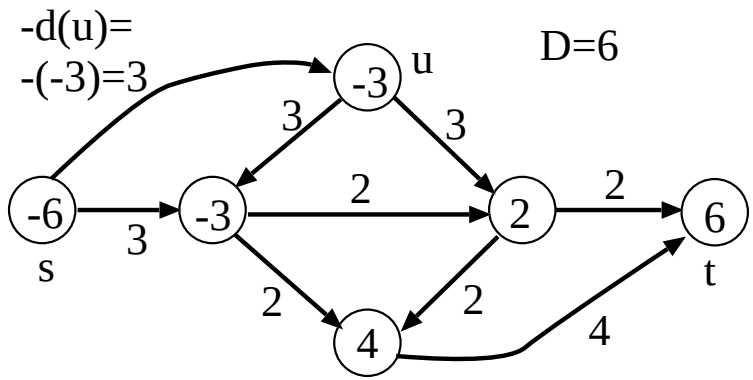


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.



G has a feasible circulation iff G' has a max-flow of value D. If G has a feasible circulation and all capacities and demands are integer, then G has an integer feasible circulation.

Proof. Any flow in G' has value$\leq D$.

Assume G has a feasible circulation f. For each arc $e = (s, u)$ in G', let $f(e) = -d(u)$, and for each arc $e = (v, t)$ in G', let $f(e) = d(v)$. Then f is a flow of value D in G'.

Assume G' has a flow f of value D. Removing the arcs (s, u) and (v, t), we get a circulation f in G with $f^{in}(v) - f^{out}(v) = d(v)$ for every $v \in V(G)$ and $0 \leq f(e) \leq c(e)$ for every $e \in e(G)$. □

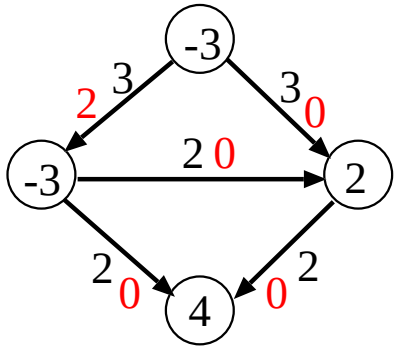
Circulation with demands and lower bounds

- **Input instance: digraph G, each arc e has a capacity $c(e) \geq 0$, each node v has a demand $d(v)$ and each arc e has a lower bound $l(e)$ with $0 \leq l(e) \leq c(e)$.**
- **A circulation f in G is feasible if**

  for each $e \in E(G)$, $l(e) \leq f(e) \leq c(e)$ (capacity condition), and

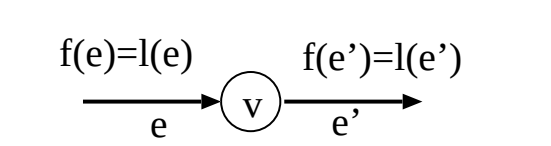
  for each node $v \in V(G)$, $f^{in}(v) - f^{out}(v) = d(v)$ (demand condition).
- **Problem: given an instance G, does G has a feasible circulation?**

  The problem can be reduced to the original circulation with demands problem.



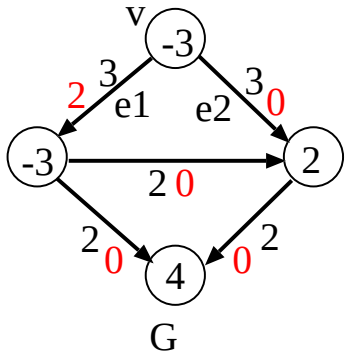
- **Remove lower bound: assign each arc e with lower bound $l(e)$ a "flow value" $f(e) = l(e)$, update node demand and arc capacity to get a new network.**
- **Let f be a function with $f(e) = l(e)$ for each $e \in E(G)$.**

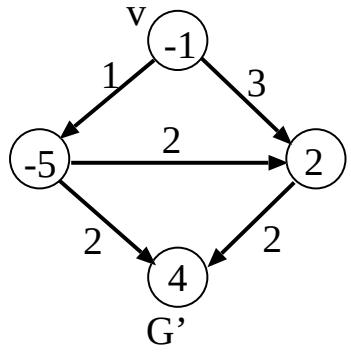
  Construct flow network G' as follows: $V(G') = V(G)$, $E(G') = E(G)$, capacity of each arc e is changed to $c'(e) = c(e) - l(e)$, demand of each node v is changed to $d'(v) = d(v) - [f^{in}(v) - f^{out}(v)]$ and the lower bounds are removed (changed to 0).



d'(v)=d(v)-[f in (v)-f out (v)]
=d(v)-[l(e)-l(e')]

c'(e)=c(e)-l(e) c'(e')=c(e')-l(e')





d'(v)=d(v)-[0-2]
=(-3)-(-2)=-1

Let G be a flow network with lower bounds and G' be the flow network with lower bounds removed. There is a feasible circulation in G iff there is a feasible circulation in G'. If all demands, capacities and lower bounds in G are integers, and there is a feasible circulation in G, then there is an integer feasible circulation in G.

Proof. Assume there is circulation f in G. Let $f'(e) = f(e) - l(e)$ for each arc e to be a circulation in G'. Since $f(e) \leq c(e)$, $f'(e) \leq c(e) - l(e) = c'(e)$, satisfying the capacity condition, and

$$(f')^{in}(v) - (f')^{out}(v) = \sum_{e=(u,v)\in E} (f(e) - l(e)) - \sum_{e'=(v,w)\in E} (f(e') - l(e'))$$

$$= [\sum_{e=(u,v)\in E} f(e) - \sum_{e'=(v,w)\in E} f(e')] - [\sum_{e=(u,v)\in E} l(e) - \sum_{e'=(v,w)\in E} l(e')]$$

$$= d(v) - [f^{in}(v) - f^{out}(v)] = d'(v),$$

satisfying the demand condition. Thus f' is a feasible circulation in G'.

Assume there is circulation f' in G'. Let $f(e) = f'(e) + l(e)$ for each arc e to be a circulation in G. Then $l(e) \leq f(e) \leq c'(e) + l(e) = c(e)$, satisfying the capacity condition, and

$$f^{in}(v) - f^{out}(v) = \sum_{e=(u,v)\in E} (l(e) + f'(e)) - \sum_{e'=(v,w)\in E} (l(e') + f'(e'))$$

$$= [\sum_{e=(u,v)\in E} f'(e) - \sum_{e'=(v,w)\in E} f'(e')] + [\sum_{e=(u,v)\in E} l(e) - \sum_{e'=(v,w)\in E} l(e')]$$

$$= d'(v) + [f^{in}(v) - f^{out}(v)] = d(v),$$

satisfying the demand condition. Thus f is a feasible circulation in G. □

Airline Scheduling

- **An airline carrier wants to serve a set of flights. An example,**

Boston (6am) $\rightarrow$ Washington DC (7am)

Philadelphia (7am) $\rightarrow$ Pittsburg (8am)

Washington DC (8am) $\rightarrow$ Los Angeles (11am)

San Francisco (2:15pm) $\rightarrow$ Seattle (3:15pm)

Las Vegas (5pm) $\rightarrow$ Seattle (6pm)

Philadelphia (11am) $\rightarrow$ San Francisco (2pm)

- **A same plane can be used for flights i and j if**
  - **the destination of i is the origin of j and there is enough time for maintenance;**
  - **a flight can be added between the destination of i and origin of j with adequate time in between.**
- **Airline scheduling problem, given a set of flights to serve and k planes, is it possible to serve the flights.**

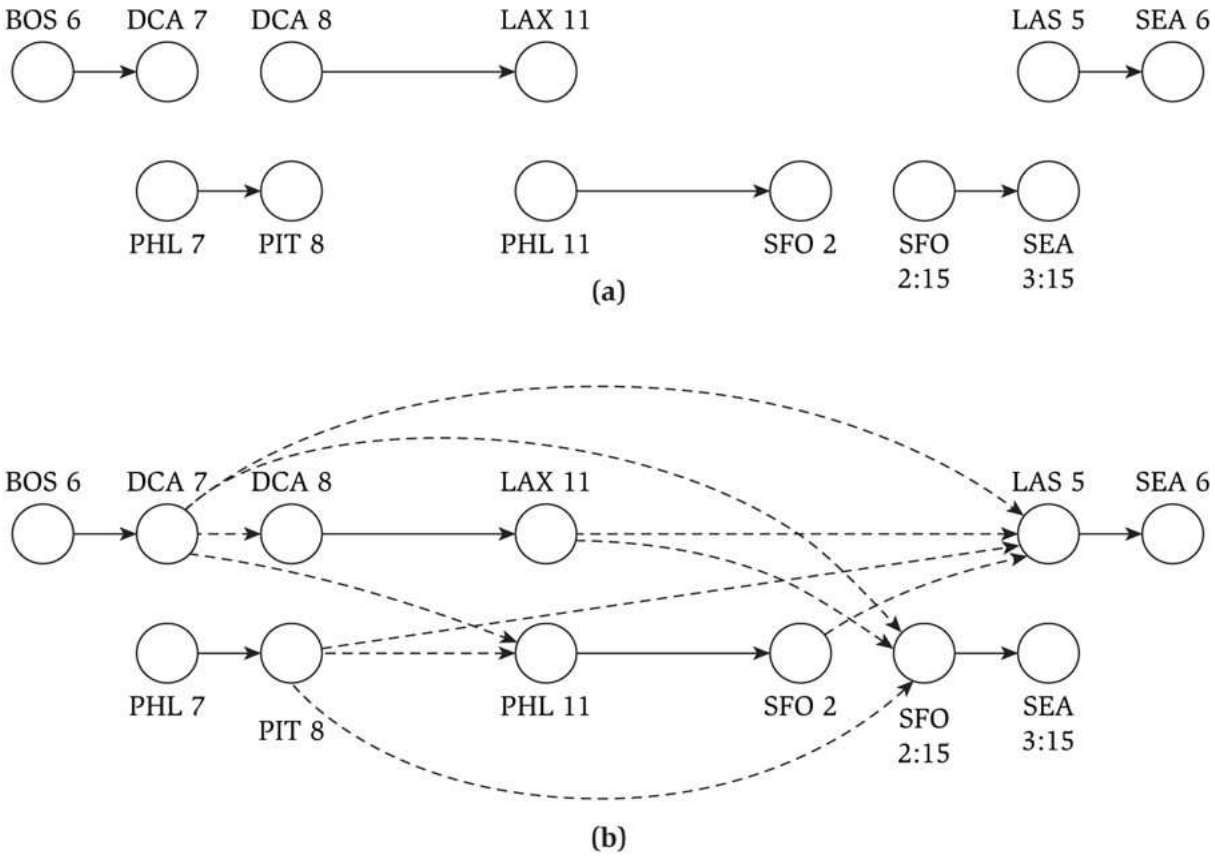


Figure 7.17 (a) A small instance of our simple Airline Scheduling Problem. (b) An expanded graph showing which flights are reachable from which others.

Airline scheduling vs flows

- **Flight j is reachable from flight i if a same plane can be used for i and j. For a set F of flights, let u_i be the origin and v_i be the destination of flight $i \in F$.**
- **Construct a flow network G, $V(G) = \{u_i, v_i | i \in F\} \cup \{s, t\}$,**

$$\begin{aligned} E(G) &= \{(u_i, v_i), (s, u_i), (v_i, t) | i \in F\} \\ &\cup \{(v_i, u_j) | j \text{ is reachable from } i\}, \end{aligned}$$

  each arc (u_i, v_i) has capacity 1 and lower bound 1, each of $(v_i, u_j), (s, u_i), (v_i, t)$ has capacity 1 and lower bound 0, node s has demand $-k$ and node t has demand k.

- **There is way to serve all flights by k planes iff there is a feasible circulation in G.**

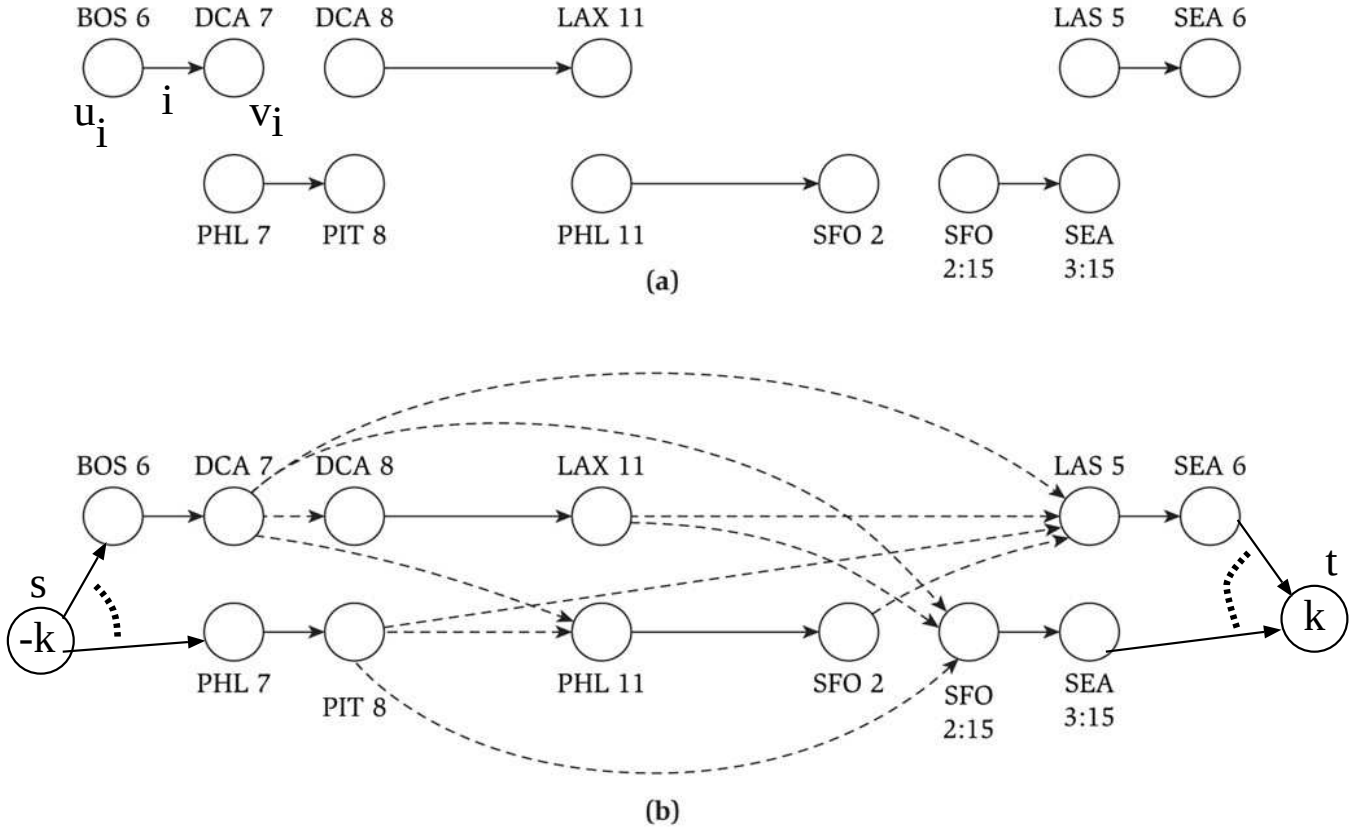
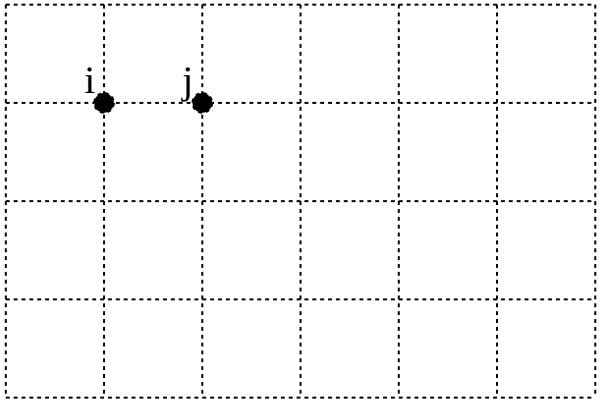


Figure 7.17 (a) A small instance of our simple Airline Scheduling Problem. (b) An expanded graph showing which flights are reachable from which others.

Image Segmentation

- **Problem: Separate objects on a digital image, which object each pixel belongs to. Simple case, decide if a pixel belongs to foreground or background.**
- **Graph model: Let G be a graph, $V(G)$ is the set of pixels and $E(G)$ has an edge $\{i, j\}$ if pixels i and j are neighbors (e.g., when pixels are arranged in a grid, each pixel has 2, 3 or 4 neighbors.**

  Each pixel i has a likelihood $a_i \geq 0$ belongs to foreground and a likelihood $b_i \geq 0$ belongs to background. For each edge $\{i, j\}$, there is a separation penalty $p_{ij} \geq 0$ for one of i, j in foreground and the other in background.



- **Image segmentation problem**

  Given a set of pixels with likelihoods and separation penalties, find a partition of the set of pixels into sets A and B (foreground and background) s.t.

$$q(A,B) = \sum_{i \in A} a_i + \sum_{i \in B} b_i - \sum_{\substack{\{i,j\} \in E \\ |A \cap \{i,j\}|=1}} p_{ij}$$

  is maximized. Such a partition is called an optimal labelling.

- **Algorithm idea**

  Find a minimum cut in G as an optimal labelling.

  Hurdles:

  (1) optimal labelling is a maximization problem,

  (2) no source/sink in G,

  (3) how to deal with likelihoods on nodes, and

  (4) G is undirected.

Clear hurdle (1): reduce the optimal labelling into a minimization problem

Let $Q = \sum_{i \in V}(a_i + b_i) = \sum_{i \in A} a_i + \sum_{i \in A} b_i + \sum_{i \in B} a_i + \sum_{i \in B} b_i$**. Then**

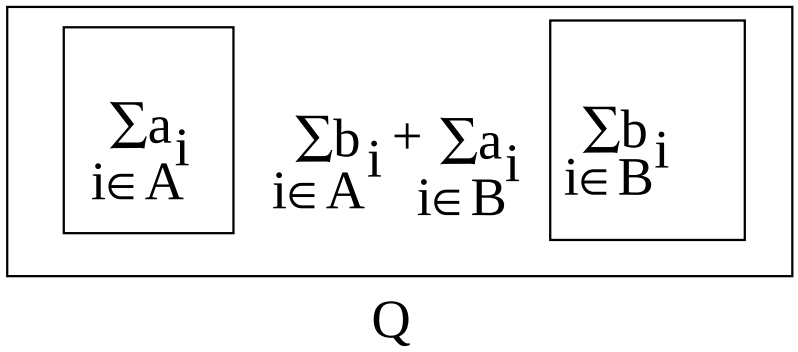
$$\sum_{i \in A} a_i + \sum_{i \in B} b_i = Q - (\sum_{i \in A} b_i + \sum_{i \in B} a_i).$$

So

$$q(A, B) = Q - (\sum_{i \in A} b_i + \sum_{i \in B} a_i) - \sum_{\substack{\{i,j\} \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}.$$

Thus, instead of maximizing $q(A, B)$**, we can minimize**

$$q'(A, B) = \sum_{i \in A} b_i + \sum_{i \in B} a_i + \sum_{\substack{\{i,j\} \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}.$$



Clear hurdle (2): Add source s, sink t, arcs $(s,i),(i,t)$ for each $i \in V(G)$.

Clear (3): assign arc (s,i) capacity a_i and arc (i,t) capacity b_i.

Clear (4): Replace each edge $\{i,j\}$ by arcs $(i,j),(j,i)$ and assign every arc capacity p_{ij}.

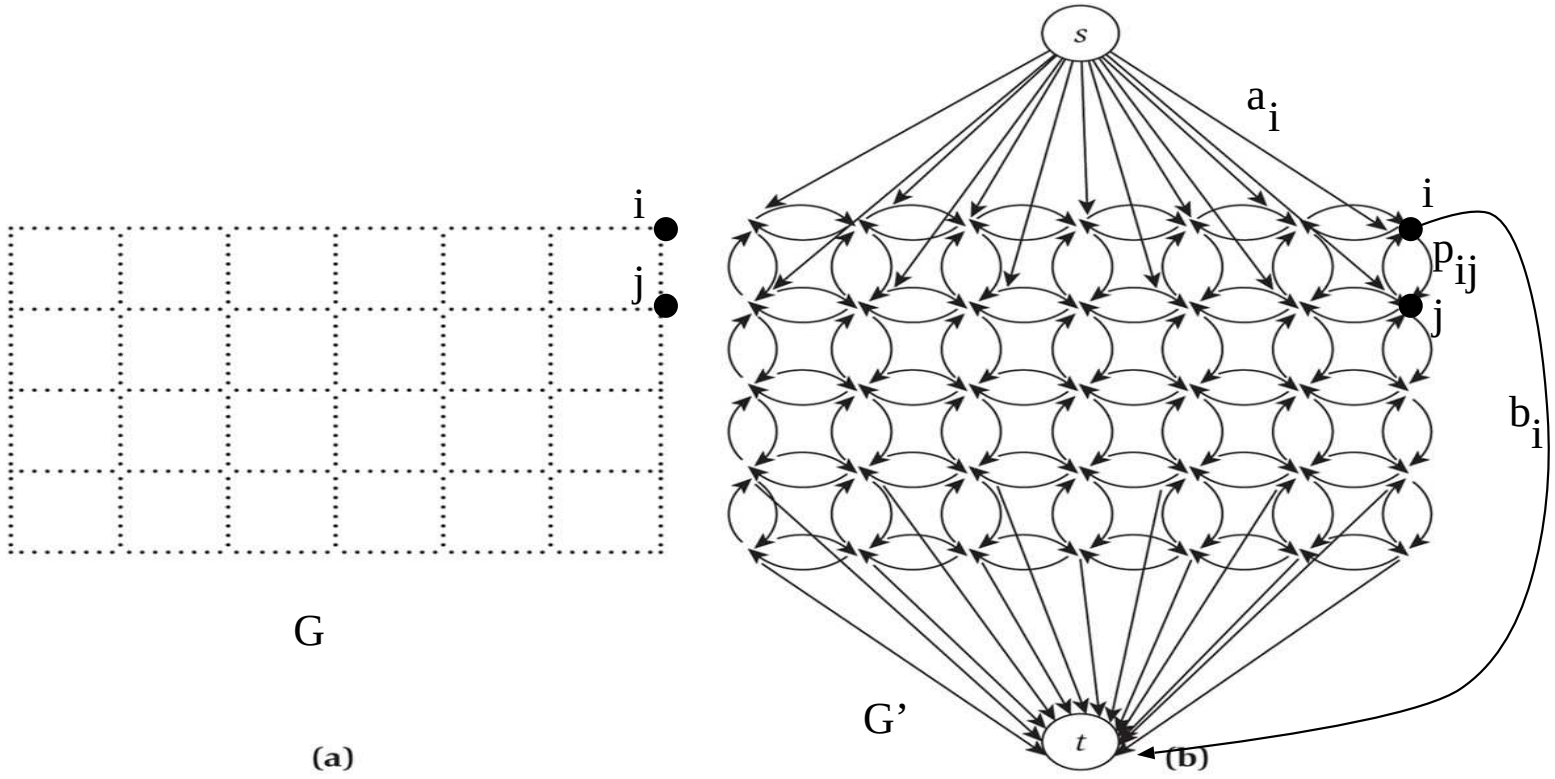


Figure 7.18 (a) A pixel graph. (b) A sketch of the corresponding flow graph. Not all edges from the source or to the sink are drawn.

Given a graph G for the optimal labelling problem, let G' be the digraph constructed in the previous slide. For a min-cut (A', B') in G', the partition (A, B) obtained in G by deleting s and t maximizes $q(A, B)$.

Proof. A cut $(A', B') = (A \cup \{s\}, B \cup \{t\})$ in G' gives a partition (A, B) in G. The capacity of $(A \cup \{s\}, B \cup \{t\})$ is

$$c(A \cup \{s\}, B \cup \{t\}) = \sum_{\substack{(s,j) \\ j \in B}} a_j + \sum_{\substack{(i,t) \\ i \in A}} b_i + \sum_{\substack{(i,j) \\ i \in A, j \in B}} p_{ij}.$$

□

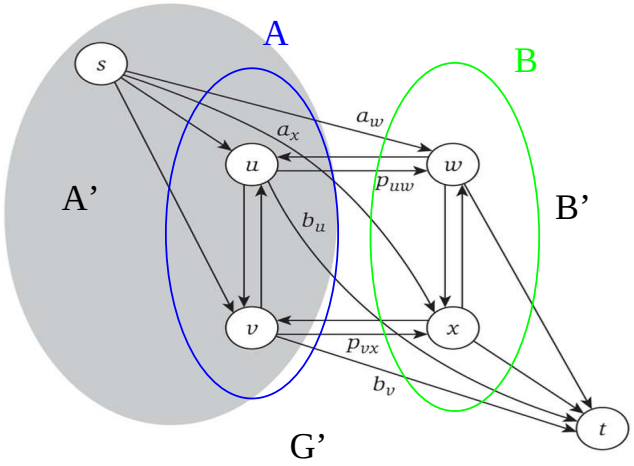
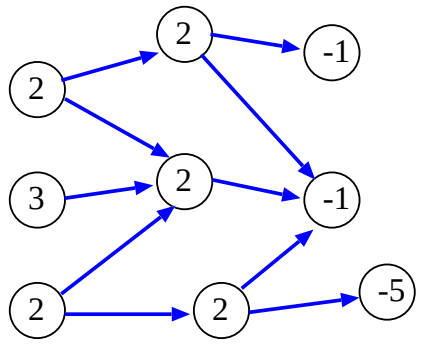


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

Project Selection

Problem definition

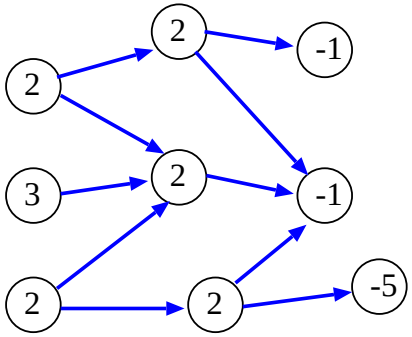
- **Given set P of projects, each project $u \in P$ has a revenue r_u and precedence constraints between projects defined by a DAG $G(P, E)$: $(u, v) \in E$ if v is a prerequisite of u.**
- **A subset $A \subseteq P$ is feasible if $\forall u \in A$ and every edge $(u, v) \in E$, $v \in A$. The revenue of A is $r(A) = \sum_{u \in A} r_u$.**
- **The project selection problem is to find a feasible $A \subseteq P$ with maximum revenue.**



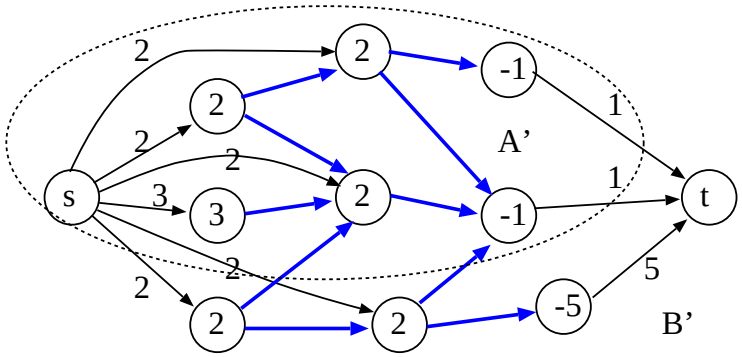
Projects and Prerequisite Graph G

Design algorithm

- **Given P and $G(P, E)$, construct a flow network $G'(V, E)$**
  - $V(G') = P(G) \cup \{s, t\}$ **(add source s and sink t).**
  - $E(G') = E(G) \cup \{(s, u)|r_u \geq 0\} \cup \{(v, t)|r_v < 0\}$
    (add edge (s, u) for $u \in P$ with $r_u \geq 0$ and edge (v, t) for $v \in P$ with $r_v < 0$).
  - **For $e = (s, u)$, capacity $c(e) = r_u$, for $e = (v, t)$, $c(e) = -r_v$.**
    Let $C = \sum_{e=(s,u)\in E(G')} c(e)$. For $e = (u, v) \in E(G)$, $c(e) = C + 1$.
- **Compute a minimum $s - t$ cut (A', B') of G' and output $A = A' \setminus \{s\}$.**



Projects and Prerequisite Graph G



Flow network G' and optimal solution
Each blue edge has capacity C+1=14

5

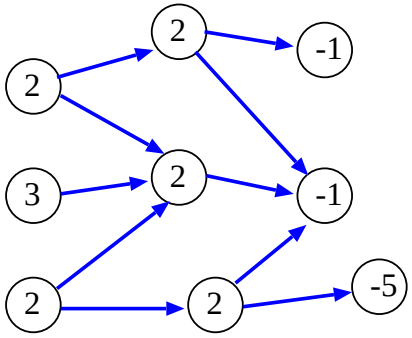
Analysis of algorithm

Lemma 1. For any $s-t$ cut (A', B') with $A = A' \setminus \{s\}$ satisfying the precedence constraints, the capacity $c(A', B') = C - r(A)$.

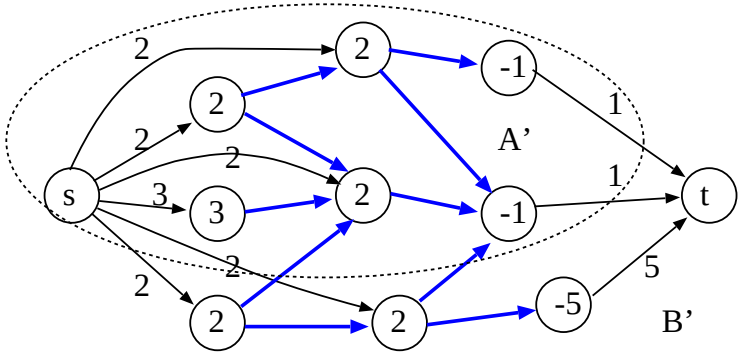
Proof. For every edge $(u, v) \in E(G)$, if $u \in A$ then $v \in A$ as A satisfying the precedence constraints. So, for each $(u, v) \in E(G')$ with $u \in A'$ and $v \in B'$, either $u = s$ or $u \in A$ and $v = t$. Then

$$\begin{aligned} c(A', B') &= \sum_{u \in B, r_u \geq 0} r_u + \sum_{v \in A, r_v < 0} (-r_v) \\ &= C - \sum_{u \in A, r_u \geq 0} r_u + \sum_{v \in A, r_v < 0} (-r_v) = C - r(A) \end{aligned}$$

□



Projects and Prerequisite Graph G



Flow network G' and optimal solution
Each blue edge has capacity C+1=14

Lemma 2. If (A', B') is an $s - t$ cut with $c(A', B') \leq C$, $A = A' \setminus \{s\}$ satisfying the precedence constraints.

Proof. For each edge $e = (u, v) \in E(G)$, if $u \in A$ then $v \in A$ as $c(e) = C + 1 > c(A', B')$. □

Theorem. If (A', B') is a minimum $s - t$ cut of G', then $A = A' \setminus \{s\}$ is an optimal solution to the project selection problem.

Proof. By Lemma 1 and Lemma 2, A is a feasible solution and $c(A', B') = C - r(A)$. Then $r(A) = C - c(A', B')$ has the maximum value as (A', B') is a minimum cut and C is a constant. □

Sports Team Elimination

Problem definition

- S : **a set of sports teams playing multiple rounds of round-robin games in a league. In each game between two teams, one wins and one loses, no draw. The champion is the team with the most wins when all games are finished.**
- **At a time point, each team x has w_x wins, each pair of teams x and y has g_{xy} games remaining.**
- **A team z is eliminated from the champion if z can not have the most wins after all remaining games.**

  The problem asks if team z has been eliminated at a given time point.

	NY	Baltimore	Toronto	Boston
wins w_x	90	88	87	79

g_{xy}	NY	Baltimore	Toronto	Boston
NY		1	6	4
Baltimore			1	4
Toronto				4
Boston				

Has Boston been eliminated?

Notations

- **Given S and a time point, let w_x be the number of wins and r_x be the number of remaining games for $x \in S$, g_{xy} be the number of remaining games between x and y for $x, y \in S$. The maximum number of possible wins for x is $m_x = w_x + r_x$.**

- **Observation: A team z has been eliminated if there is a team x with $w_x > m_z$. In general, a team z has been eliminated if there is a subset $T \subseteq S$ of teams s.t. $\sum_{x \in T} w_x + \sum_{x,y \in T} g_{xy} > m_z \times |T|$.**

  Proof. The total number of wins for teams in T is greater than $m_z \times |T|$, there is a team of T with total number of wins greater than m_z. □

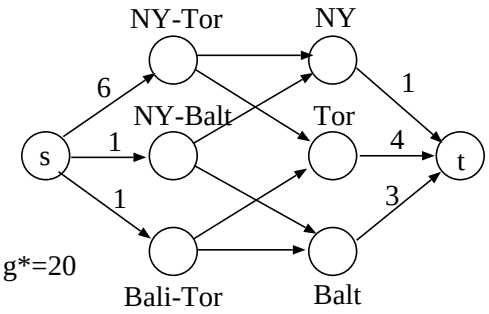
Design algorithm

- **For a team $z \in S$, construct a flow network G: Let $S' = S \setminus \{z\}$.**
  - **For every pair $x, y \in S'$ with $g_{xy} > 0$, create a node u_{xy};**
    for every $x \in S'$, create a node v_x; create source s and sink t.
  - **For every u_{xy}, create edges (s, u_{xy}) (u_{xy}, v_x) and (u_{xy}, v_y);**
    for every $x \in S'$, create edge (v_x, t).
  - **For every edge $e = (s, u_{xy})$, capacity $c(e) = g_{xy}$;**
    for every edge $e = (v_x, t)$, capacity $c(e) = m_z - w_x$;
    for each edge e of (u_{xy}, v_x) and (u_{xy}, v_y), capacity $c(e) = g^* = \sum_{x,y \in S'} g_{xy}$.
- **Find a max-flow f on G. If $v(f) < g^*$ then team z has been eliminated.**

	NY	Baltimore	Toronto	Boston
wins w_x	90	88	87	79

g_{xy}	NY	Baltimore	Toronto	Boston
NY		1	6	4
Baltimore			1	4
Toronto				4
Boston				

Has Boston been eliminated?



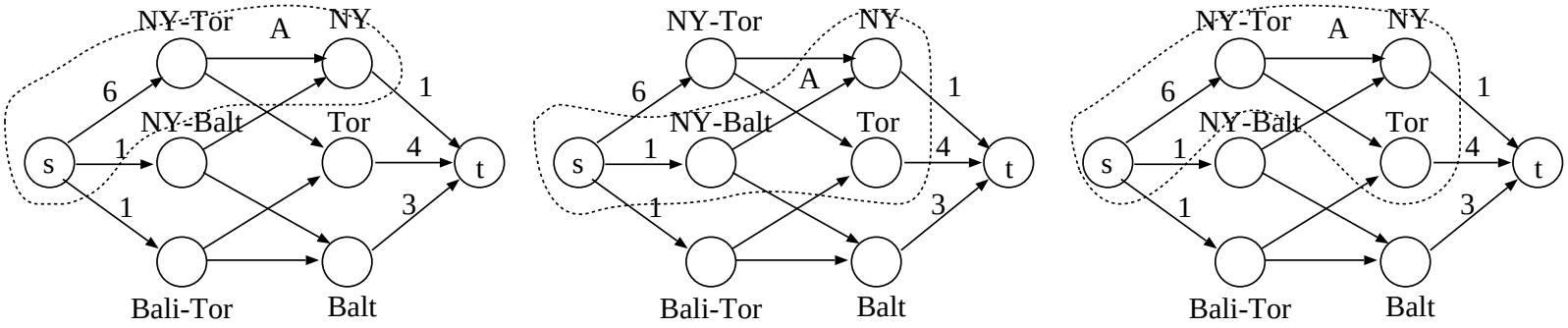
m_z=91, g*=20

Analysis of algorithm

Theorem. Team z has been eliminated if the max-flow value $v(f) < g^*$.

Proof. Assume $v(f) < g^*$. By max-flow min-cut theorem, there is a min-cut (A, B) of G with capacity $c(A, B) = v(f) < g^*$. For each node $v_x \in B$, node $u_{xy} \in B$, otherwise edge (u_{xy}, v_x) of capacity g^* is from A to B, contradicting to $c(A, B) < g^*$. For $v_x, v_y \in A$, $u_{xy} \in A$, otherwise let (A', B') be the cut with $A' = A \cup \{u_{xy}\}$ and $B' = B \setminus \{u_{xy}\}$, $c(A', B') = c(A, B) - g_{xy} < c(A, B)$, contradicting to (A, B) a minimum cut. So, every edge from A to B is one of the two types below:

- (v_x, t) with $v_x \in A$ and capacity $m_z - w_x$.
- (s, u_{xy}) with $u_{xy} \in B$, x or y not in A, and capacity g_{xy}.



Let T be the set of teams x with $v_x \in A$.

$$\begin{aligned} c(A,B) &= \sum_{x \in T}(m_z - w_x) + \sum_{x \notin T \text{ or } y \notin T} g_{xy} \\ &= m_z \times |T| - \sum_{x \in T} w_x + (g^* - \sum_{x,y \in T} g_{xy}). \end{aligned}$$

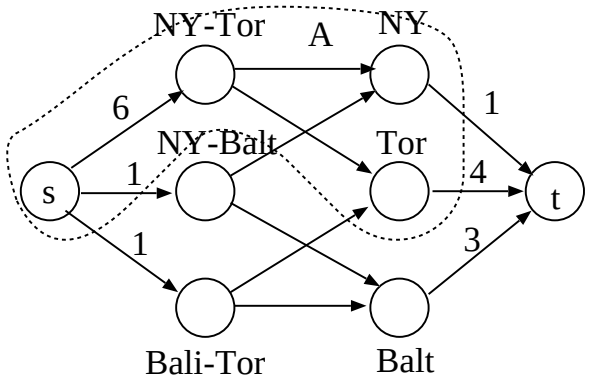
Since $c(A,B) = v(f) < g^*$,

$$m_z \times |T| - \sum_{x \in T} w_x - \sum_{x,y \in T} g_{xy} < 0,$$

giving

$$\sum_{x \in T} w_x + \sum_{x,y \in T} g_{xy} > m_z \times |T|.$$

By Observation, the theorem is proved. □



Minimum Weight Perfect Matching

Problem definition

- **Let $G(X \cup Y, E)$ be a directed bipartite graph with $|X| = |Y|$ and each edge $e = (u, v) \in E$ is from $u \in X$ to $v \in Y$ and assigned weight $w(e) \geq 0$.**
- **A subset $M \subseteq E$ is a matching if every node of G is incident to at most one edge of M. M is a perfect matching if each node of G is incident to exactly one edge of M. The weight of a matching M is $w(M) = \sum_{e \in M} w(e)$.**
- **Problem is to find a perfect matching M in G of minimum weight.**

Design algorithm

- **Given $G(X \cup Y, E)$, construct a flow network N:**
  - **Add source s and t to G.**
  - **For each $u \in X$, add edge $e = (s, u)$ of weight $w(e) = 0$ and capacity $c(e) = 1$.**
  - **For each $v \in Y$, add edge $e = (v, t)$ of weight $w(e) = 0$ and capacity $c(e) = 1$.**
  - **For every $e = (u, v) \in E$, $c(e) = 1$ and weight $w(e)$.**
- **For a flow f in N, the set of edges $e = (u, v)$ of $f(e) = 1$ is a matching M. The residual graph of N w.r.t. flow f is denoted as N_f.**
- **Algorithm PM to compute a perfect matching M of G:**
  - **Initialize $f(e) = 0$ for every edge of G. $N_f = N$.**
  - **while $\exists$ an augment path from s to t in N_f do**
    find an augment path from s to t;
    $f' =$Augment(f, P); $f = f'$; compute N_f;
  - **Output set of edges $e = (u, v)$ of $f(e) = 1$ in G as M.**

Algorithm PM computes a perfect matching of G if it has one. Our goal is to find a perfect matching of minimum weight. Idea: finding a minimum weight $s-t$ augment path in N_f w.r.t. the edge weight each time in Algorithm PM.

How to express weights of edges of N in N_f?

- **Let $w(e)$ be the weight of edge e in N.**
- **Edge weight in N_f: if edge e is a forward edge, then the weight of e is $w(e)$; if e is a backward edge, then the weight of e is $-w(e)$.**

Hurdles, N_f may have edges of negative weights.

- **Initially, $v(f)=0$ and $N_f=N$, there is no negative weight edge in N_f.**
- **When $v(f)>0$, N_f may have negative weight edges.**
- **A cycle C in N_f is a negative cycle if $\sum_{e\in C} w(e)<0$.**
- **If N_f has an $s-t$ path and does not have any negative cycle, then a minimum weight $s-t$ augment path can be computed by a shortest path algorithm.**

Properties to show N_f has no negative cycle.

For an edge weighted digraph $H(V, E)$, let $p : V(H) \to \mathbb{R}$ assigning each node $u \in V(H)$ a real number $p(u)$.

For each $e = (u, v) \in E(H)$, let $\hat{w}(e) = p(u) + w(e) - p(v)$.

For a cycle C in H, let $w(C) = \sum_{e \in C} w(e)$ and $\hat{w}(C) = \sum_{e \in C} \hat{w}(e)$.

Lemma 1. $w(C) = \hat{w}(C)$.

Proof. Assume C consists of edges $e_1, .., e_k$ with $e_i = (u_{i-1}, u_i)$ and $u_k = u_0$. Then

$$\begin{aligned} \hat{w}(C) &= \sum_{1 \le i \le k} p(u_{i-1}) + w(e_i) - p(u_i) \\ &= \sum_{1 \le i \le k} w(e_i) = w(C). \end{aligned}$$

□

For H and $p : V(H) \to \mathbb{R}$, let $\hat{H}$ be the graph with $V(\hat{H}) = V(H)$, $E(\hat{H}) = E(H)$, and edge $e = (u, v) \in E(\hat{H})$ has weight $\hat{w}(e) = p(u) + w(e) - p(v)$.

Lemma 2.

1. **H has a negative cycle iff $\hat{H}$ has a negative cycle.**
2. **Assume H does not have a negative cycle. A path P is a shortest path from s to t in H iff P is a shortest path from s to t in $\hat{H}$.**

Proof. Statement 1 follows from Lemma 1.

Proof of Statement 2:

Let P be a path from s to t consisting of edges $e_1, .., e_k$ with $e_i = (u_{i-1}, u_i)$, $1 \leq i \leq k$, $u_0 = s$ and $u_k = t$. Then $w(P) = \sum_{1 \leq i \leq k} w(e_i)$ and

$$\begin{aligned} \hat{w}(P) &= \sum_{1 \leq i \leq k} \hat{w}(e_i) = p(s) - p(t) + \sum_{1 \leq i \leq k} w(e_i) \\ &= p(s) - p(t) + w(P). \end{aligned}$$

For any path P' from s to t consisting of edges $e'_1, .., e'_j$ with $e'_i = (u_{i-1}, u_i)$, $1 \leq i \leq j$, $u_0 = s$ and $u_j = t$. Then $w(P') = \sum_{1 \leq i \leq j} w(e'_i)$ and

$$\begin{aligned} \hat{w}(P') &= \sum_{1 \leq i \leq j} \hat{w}(e'_i) = p(s) - p(t) + \sum_{1 \leq i \leq j} w(e'_i) \\ &= p(s) - p(t) + w(P'). \end{aligned}$$

If P is a shortest path in H, then $w(P) \leq w(P')$, $\hat{w}(P) \leq \hat{w}(P')$, and P is a shorest path in $\hat{H}$. Similalry, if P is a shortest path in $\hat{H}$, then P is a shortest path in H. □

Lemma 3. Let f be a flow with value $v(f) \geq 0$ in N and no negative cycle in N_f. For each edge $e = (u, v)$ in N_f, let $\hat{w}(e) = p(u) + w(e) - p(v) \geq 0$ be the modified weight of edge e for some $p : V(N_f) \to \mathbb{R}$. Let P be a shortest $s - t$ augment path in N_f w.r.t. $\hat{w}(e)$, f' be the new flow computed by subroutine Augment(f, P), and $N_{f'}$ be the residual network w.r.t. f'. There is a $p' : V(N_{f'}) \to \mathbb{R}$ s.t. for each edge $e = (u, v)$ in $H_{f'}$, $\hat{w}'(e) = p'(u) + w(e) - p'(v) \geq 0$.

Proof. For each node u, let $d_f(u)$ be the minimum distance from s to u in N_f w.r.t. $\hat{w}(e)$.
Let $p' : V(N_{f'}) \to \mathbb{R}$ with $p'(u) = d_f(u) + p(u)$.
We prove $\hat{w}'(e) = p'(u) + w(e) - p'(v) \geq 0$ for e in $H_{f'}$.
Let $M = \{e \in E(G) | f(e) = 1\}$ and $M' = \{e \in E(G) | f'(e) = 1\}$.

For $e = (u, v) \notin M'$, e is a forward edge in and $N_{f'}$. Since $d_f(v) \leq d_f(u) + \hat{w}(e)$,

$$\begin{aligned} \hat{w}'(e) &= p'(u) + w(e) - p'(v) = d_f(u) + p(u) + w(e) - d_f(v) - d(v) \\ &= d_f(u) + \hat{w}(e) - d_f(v) \geq d_f(u) + \hat{w}(e) - d_f(u) - \hat{w}(e) = 0. \end{aligned}$$

For $e = (u, v) \in M'$, e is a backward edge (v, u) in $N_{f'}$.
For $e \in M \cap M'$, e is a backward edge (v, u) in N_f

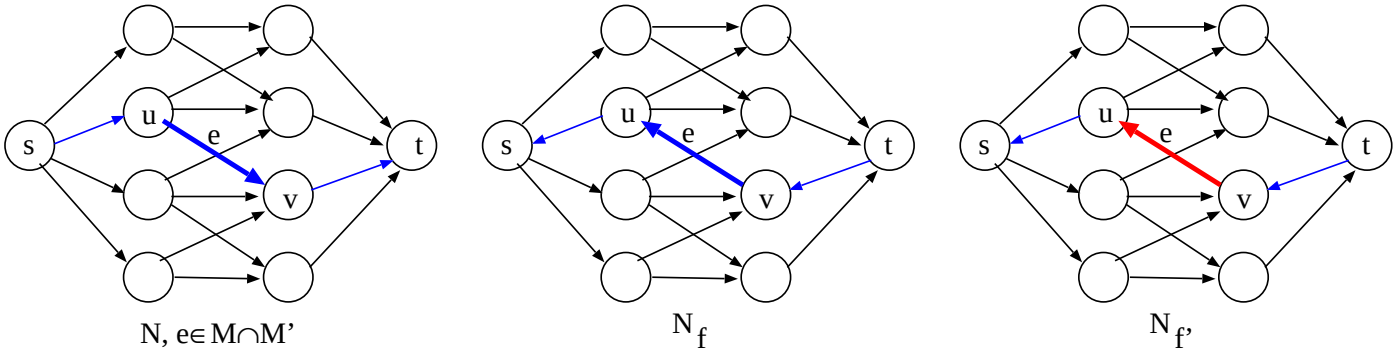
$$\hat{w}(e) = p(v) + (-w(e)) - p(u),$$

and

$$d_f(u) = d_f(v) - \hat{w}(e) = d_f(v) - (p(v) + (-w(e)) - p(u)).$$

Then

$$\begin{aligned} \hat{w}'(e) &= p'(v) + (-w(e)) - p'(u) = d_f(v) + p(v) + (-w(e)) - d_f(u) - p(u) \\ &= d_f(v) + p(v) + (-w(e)) - d_f(v) + p(v) + (-w(e)) - p(u) - p(u) \\ &= 2(p(v) + (-w(e)) - p(u)) = 2\hat{w}(e) \geq 0. \end{aligned}$$

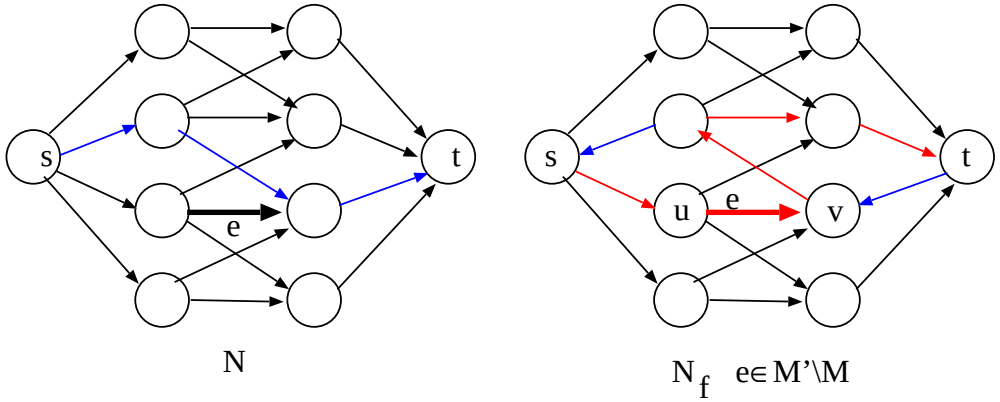


For edge $e = (u, v) \in M' \setminus M$, e is a forward edge (u, v) in N_f,

$$\hat{w}(e) = p(u) + w(e) - p(v),$$

and

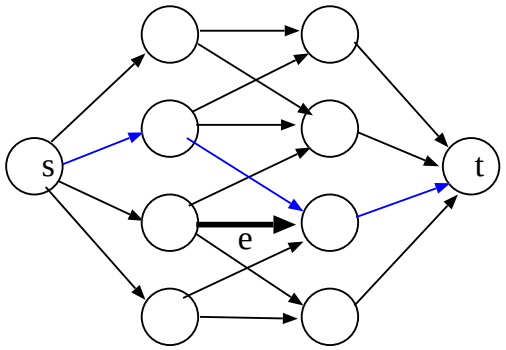
$$d_f(v) = d_f(u) + \hat{w}(e) = d_f(u) + p(u) + w(e) - p(v).$$



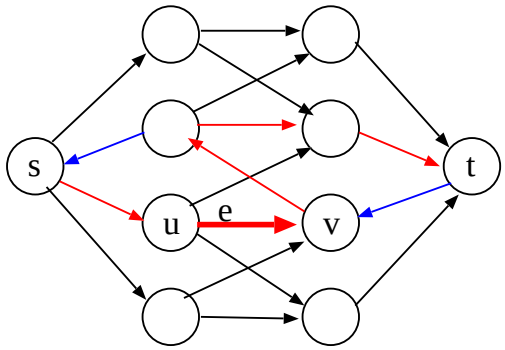
Since e is a backward edge (v, u) in $N_{f'}$,

$$\begin{aligned}\hat{w}'(e) &= p'(v) + (-w(e)) - p'(u) \\ &= d_f(v) + p(v) - w(e) - d_f(u) - p(u) \\ &= d_f(u) + p(u) + w(e) - p(v) + p(v) - w(e) - d_f(u) - p(u) = 0.\end{aligned}$$

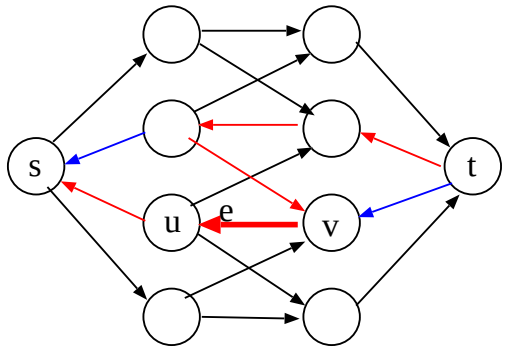
□



N



N_f $e \in M' \backslash M$



$N_{f'}$ $e \in M' \backslash M$

By Lemma 3, we can use Bellman-Ford algorithm to find minimum weight $s-t$ augment path.

Modify edge weights in N_f as $\hat{w}(e) \geq 0$ in the proof of Lemma 3, we can compute minimum weight $s-t$ augment path by Dijkstra algorithm (more efficient).

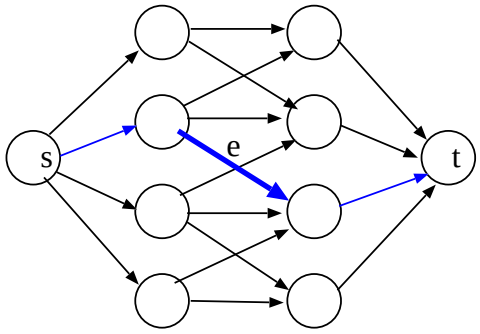
Algorithm MPM for minimum weight perfect matching

- **Given $G(X \cup Y, E)$, construct flow network N: $f(e) = 0$ for $e \in N$;**
  $M = \emptyset$; $N_f = N$; $p(u) = 0$ for every node in N_f;
- **while there is an augment path from s to t in N_f do**
  compute shortest distance $d_f(u)$ from s to every other node u in N_f;
  for each u in N_f, update $p(u) = d_f(u) + p(u)$;
  find a augment path P from s to t of minimum weight $\hat{w}(P)$ in N_f;
  $f' =$Augment(f, P); $f = f'$; compute N_f;
- **Output set of edges $e = (u, v)$ of $f(e) = 1$ in G as M.**

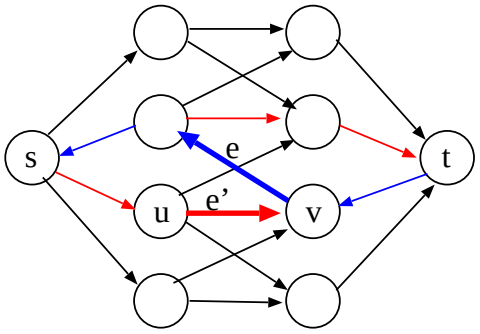
Theorem. Algorithm MPM computes a minimum weight perfect matching M in G.

Proof. For f with $v(f) = 0$, $\hat{w}(e) = w(e) \geq 0$ for every e in N_f. By Lemma 3, for every f computed by the algorithm, $\hat{w}(e) \geq 0$ for every e in N_f. So, N_f does not have any negative cycle and the algorithm finds a perfect matching.

Assume there is a perfect matching M' with $w(M') < w(M)$. Let $\Delta = (M \setminus M') \cup (M' \setminus M)$. Then edges of Δ form a set of cycles in N_f and $w(\Delta) = w(M') - w(M) < 0$, implying N_f has a negative cycle, contradiction. So, M is a minimum weight perfect matching. □



N, e∈M



N_f , e'∈M'\M

Minimum Cost Maximum Flow

- **Minimum cost flow: each arc e also has a cost $a(e)$, the cost of a flow f is**

$$a(f) = \sum_{e \in E(G)} a(e) \cdot f(e).$$

Given a flow value $\mathrm{val}(f)$, the minimum cost flow of value $\mathrm{val}(f)$ problem is

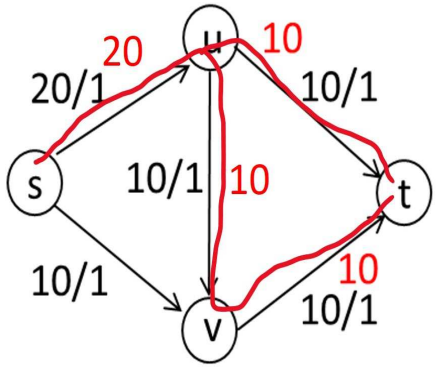
$$\min \quad \sum_{e \in E(G)} a(e) \cdot f(e) \qquad \textbf{subject to}$$

$$\forall e \in E(G), 0 \leq f(e) \leq c(e) \qquad \textbf{Capacity condition}$$

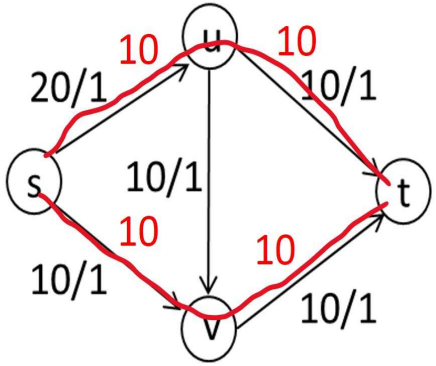
$$\forall v \in V(G),$$

$$\sum_{e=(u,v) \in E(G)} f(e) - \sum_{e=(v,w) \in E(G)} f(e) = d(v) \qquad \textbf{Demand condition}$$

- **Minimum cost maximum flow problem: find a minimum cost maximum flow in G.**



Max-flow f of value 20 and cost 50



Max-flow f' of value 20 and cost 40

Algorithm for min-cost max-flow

- **Given a flow f in G, the residual graph G_f w.r.t. f:**
  - $V(G_f) = V(G)$**;**
  - $\forall e = (u, v) \in E(G)$ **with** $0 \leq f(e) < c(e), (u, v) \in E(G_f)$ **with capacity** $c(e) - f(e)$ **and cost** $a(e)$ **(forward arc);**
  - $\forall e = (u, v) \in E(G)$ **with** $0 < f(e), (v, u) \in E(G_f)$ **with capacity** $f(e)$ **and cost** $-a(e)$ **(backward arc).**
- **Any flow f can be decomposed into paths/cycles in G_f.**

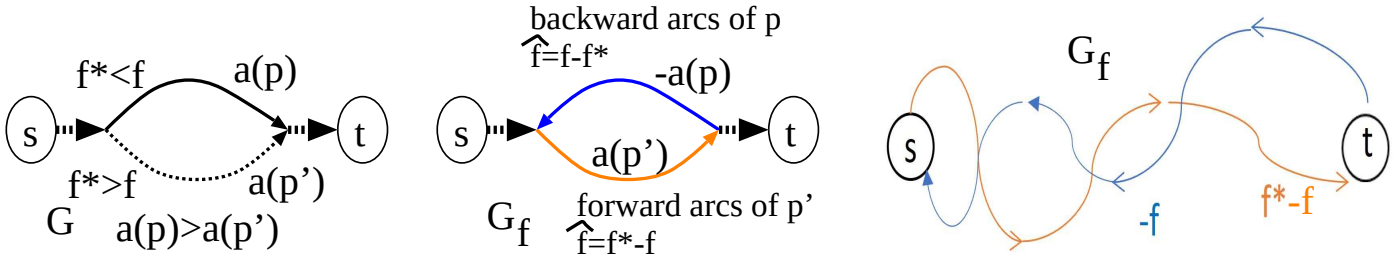
  The cost of a path (cycle) P (C) in G_f is $\sum_{e \in P} a(e)$ ($\sum_{e \in C} a(e)$).
- **A flow f is optimal if $a(f) \leq a(f')$ for any f' with $\mathrm{val}(f') = \mathrm{val}(f)$.**
- **Find optimal flow f: Intuition, given flow f, if there is a cycle C in G_f of negative cost, then increase the flow in C gives a flow f' with $\mathrm{val}(f') = \mathrm{val}(f)$ and $a(f') < a(f)$.**

  Such C is called augmenting cycle of negative cost (negative cycle).

- **A flow f is optimal iff f has no negative cycle in G_f.**

Proof. Assume that G_f has a negative cycle C. Then a new flow f' with $\mathrm{val}(f') = \mathrm{val}(f)$ can be obtained by increasing the flow on C. Since C has negative cost, $a(f') < a(f)$ and thus, f is not optimal.

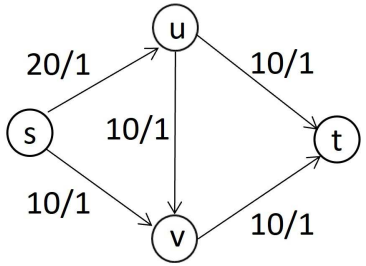
Assume that f is not optimal. Let f^* be a flow with $\mathrm{val}(f^*) = \mathrm{val}(f)$, $a(f^*) < a(f)$, and $\hat{f}(e) = f^*(e) - f(e)$ for every arc e of G. Then $\hat{f}$ is a feasible flow on G_f (if $f^*(e) - f(e) > 0$, then $\hat{f}(e)$ is a flow on the forward arc of e, if $f^*(e) - f(e) < 0$ then $\hat{f}(e)$ is a flow on the backward arc of e in G_f, capacity condition and conservation condition are satisfied). Since $a(f^*) < a(f)$, $\hat{f}$ is a flow in G_f of negative cost, implying G_f has a negative cycle. □



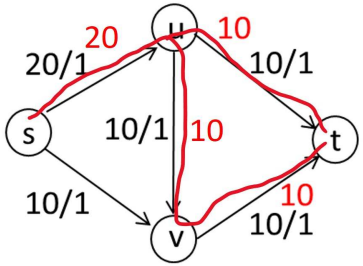
- **Find negative cycle in G_f: add a psuedo source s' and arc (s', v) for every $v \in V(G_f)$ to get $\hat{G}_f$ and run Bellman-Ford shortest path algorithm on $\hat{G}_f$ with s' as source.**
- **Algorithm**
  - **Find a maximum flow f in G.**
  - **Find negative cycle in G_f by Bellman-Ford shortest path algorithm.**

    If a negative cycle C is found then let e be a bottle arc in C and increase the flow on C to $c(e)$ to get a new flow f'. Notice that $\mathrm{val}(f') = \mathrm{val}(f)$ and arc e disappears in $G_{f'}$. Repeat this to eliminate every negative cycle.

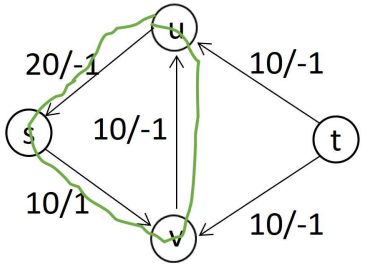
    Let $f = f'$ and repeat this step until there is no negative cycle in G_f.
  - **Output f.**



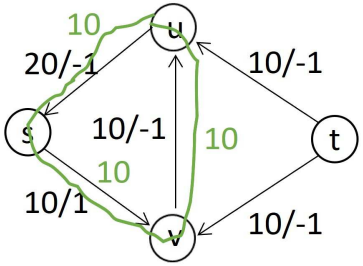
G, numbers on
arc: capacity/cost



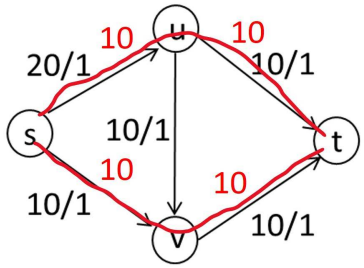
Max-flow f of value
20 and cost 50



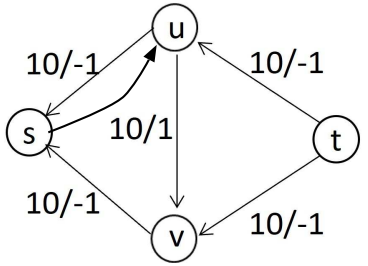
G_f has a negative cycle P



Create flow f" of value 10 on P
to 10 to get a new flow f'=f+f"



Max-flow f' of value
20 and cost 40



$G_{f'}$ has no negative cycle

- **Analysis of the algorithm**

  It takes $O(mn)$ time to detect and eliminate negative cycles in G_f for network G of m arcs and n vertices. Let $a(f)$ be the cost of a given flow f. The total time of the algorithm is $O(mna(f))$ since every update on f reduces the cost of f by at least one unit. Let k to the total capacity of all arcs and l be the total cost of all arcs. Then kl is an upper bound on any flow in G. So $O(mnkl)$ is an upper bound on the time of the algorithm.