Maximum Flow Applications

Maximum Flow Applications Contents

Max flow extensions and applications.

- Disjoint paths and network connectivity.
- Bipartite matchings.
- Circulations with upper and lower bounds.
- Census tabulation (matrix rounding).
- Airline scheduling.
- Image segmentation.
- Project selection (max weight closure).
- Baseball elimination.

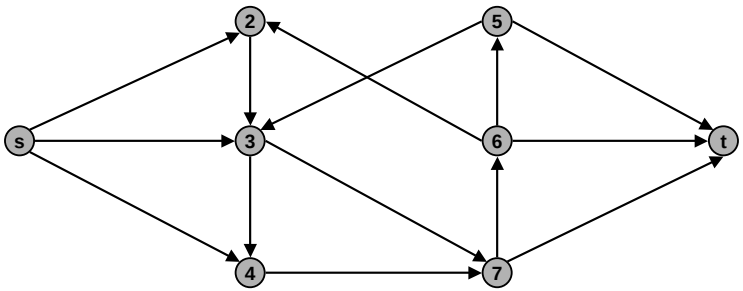
Disjoint Paths

Disjoint path network: G = (V, E, s, t).

- Directed graph (V, E), source s, sink t.
- Two paths are edge-disjoint if they have no arc in common.

Disjoint path problem: find max number of edge-disjoint s-t paths.

- Application: communication networks.

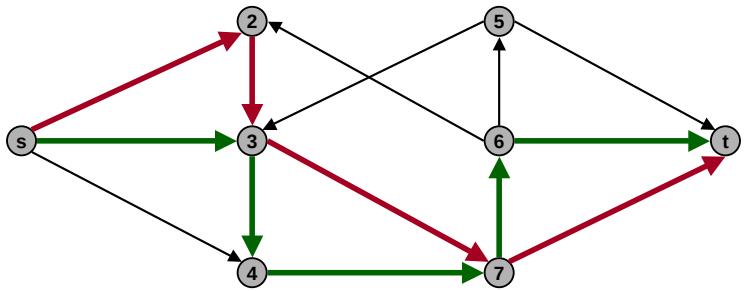


Disjoint Paths

Disjoint path network: G = (V, E, s, t).

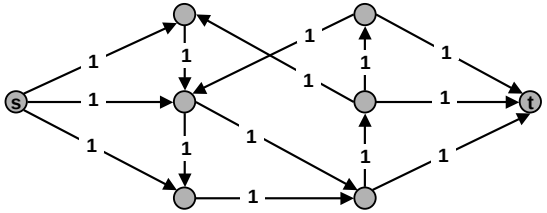
- Directed graph (V, E), source s, sink t.
- Two paths are edge-disjoint if they have no arc in common.

Disjoint path problem: find max number of edge-disjoint s-t paths.



Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



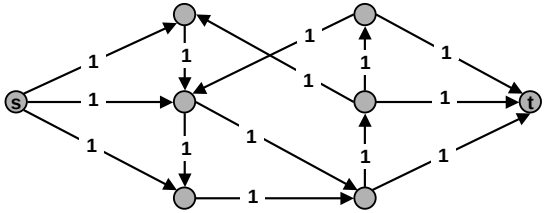
Theorem. There are k edge-disjoint paths from s to t if and only if the max flow value is k.

Proof. $\Rightarrow$

- Suppose there are k edge-disjoint paths $P_1, \ldots, P_k$.
- Set f(e) = 1 if e participates in some path P_i ; otherwise, set f(e) = 0.
- Since paths are edge-disjoint, f is a flow of value k.

Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



Theorem. There are k edge-disjoint paths from s to t if and only if the max flow value is k.

Proof. $\Leftarrow$

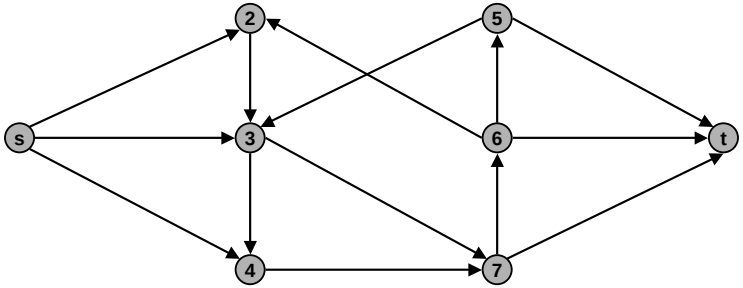
- Suppose max flow value is k. By integrality theorem, there exists {0, 1} flow f of value k.
- Consider edge (s,v) with f(s,v) = 1.
  - by conservation, there exists an arc (v,w) with f(v,w) = 1
  - continue until reach t, always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths.

Network Connectivity

Network connectivity network: G = (V, E, s, t) .

- Directed graph (V, E), source s, sink t.
- A set of edges F $\subseteq$ E **disconnects t from s** if all s-t paths uses at least on edge in F.

Network connectivity: find min number of edges whose removal disconnects t from s.

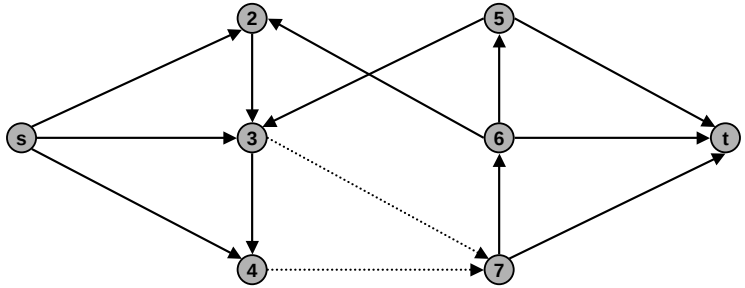


Network Connectivity

Network connectivity network: G = (V, E, s, t) .

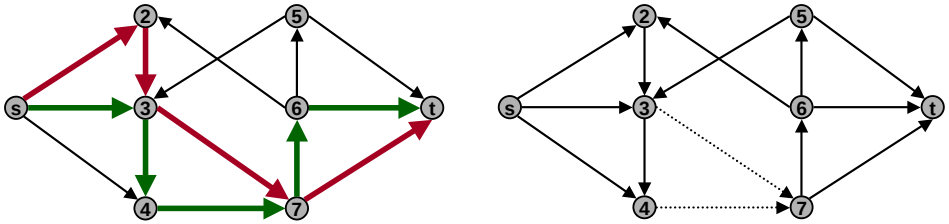
- Directed graph (V, E), source s, sink t.
- A set of edges F $\subseteq$ E **disconnects t from s** if all s-t paths uses at least on edge in F.

Network connectivity: find min number of edges whose removal disconnects t from s.



Disjoint Paths and Network Connectivity

Menger's Theorem (1927). The max number of edge-disjoint s-t paths is equal to the min number of arcs whose removal disconnects t from s.

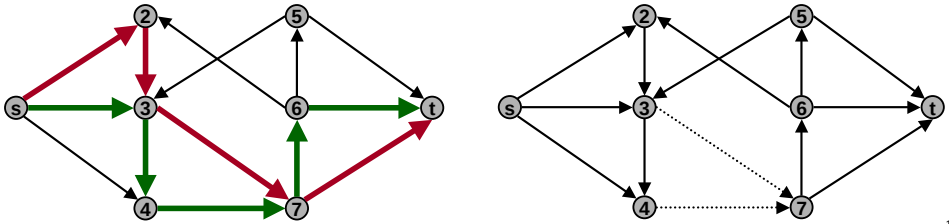


Disjoint Paths and Network Connectivity

Menger's Theorem (1927). The max number of edge-disjoint s-t paths is equal to the min number of arcs whose removal disconnects t from s.

Proof. ⇐

- Suppose the removal of $F \subseteq E$ disconnects t from s, and $|F| = k$.
- All s-t paths use at least one edge of F. Hence, the number of edge-disjoint paths is at most k.

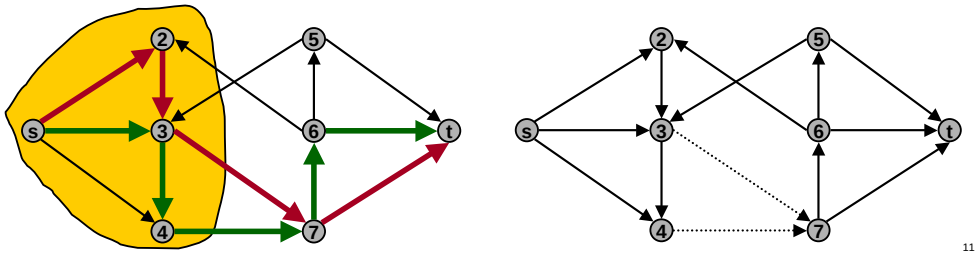


Disjoint Paths and Network Connectivity

Menger's Theorem (1927). The max number of edge-disjoint s-t paths is equal to the min number of arcs whose removal disconnects t from s.

Proof. ⇒

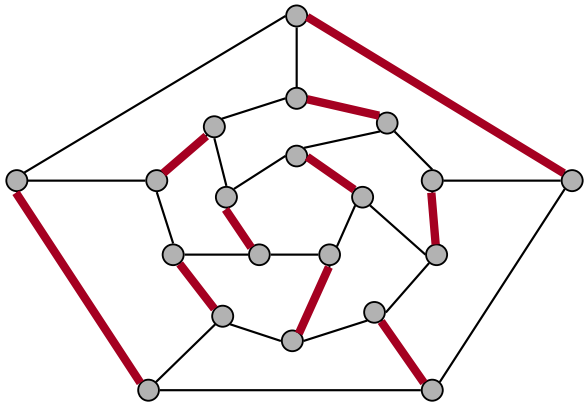
- Suppose max number of edge-disjoint paths is k.
- Then max flow value is k.
- Max-flow min-cut ⇒ cut (S, T) of capacity k.
- Let F be set of edges going from S to T.
- |F| = k, and definition of cut implies F disconnects t from s.



Matching

Matching.

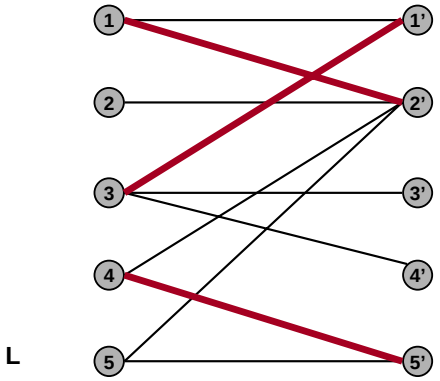
- Input: undirected graph G = (V, E).
- $M \subseteq E$ is a matching if each node appears in at most edge in M.
- Max matching: find a max cardinality matching.



Bipartite Matching

Bipartite matching.

- Input: undirected, bipartite graph G = (L ∪ R, E).
- M ⊆ E is a matching if each node appears in at most edge in M.
- Max matching: find a max cardinality matching.



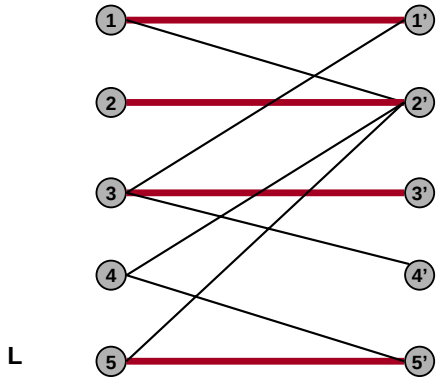
Matching

1-2', 3-1', 4-5'

Bipartite Matching

Bipartite matching.

- Input: undirected, bipartite graph G = (L ∪ R, E).
- M ⊆ E is a matching if each node appears in at most edge in M.
- Max matching: find a max cardinality matching.



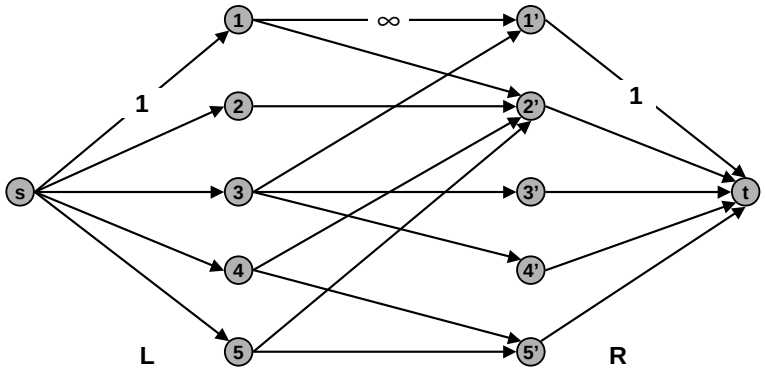
Matching

1-1', 2-2', 3-3', 4-4'

Bipartite Matching

Max flow formulation.

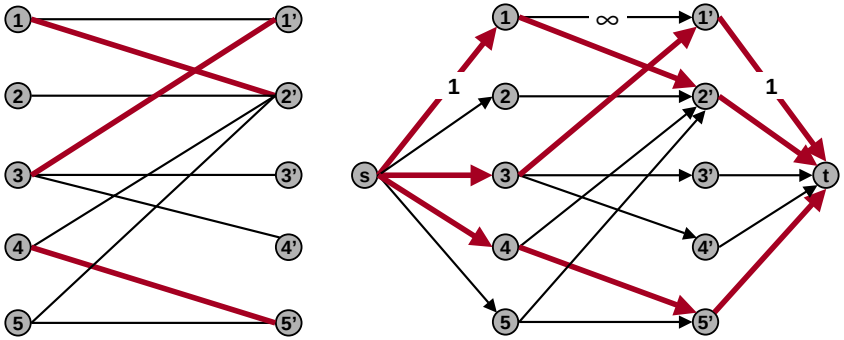
- Create directed graph G' = (L ∪ R ∪ {s, t}, E').
- Direct all arcs from L to R, and give infinite (or unit) capacity.
- Add source s, and unit capacity arcs from s to each node in L.
- Add sink t, and unit capacity arcs from each node in R to t.



Bipartite Matching: Proof of Correctness

Claim. Matching in G of cardinality k induces flow in G' of value k.

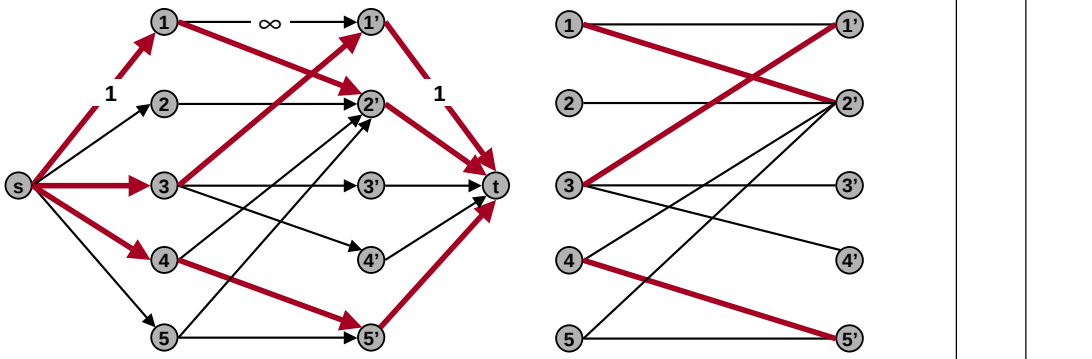
- Given matching M = { 1 - 2', 3 - 1', 4 - 5' } of cardinality 3.
- Consider flow that sends 1 unit along each of 3 paths:
  s - 1 - 2' - t, s - 3 - 1' - t, s - 4 - 5' - t.
- f is a flow, and has cardinality 3.



Bipartite Matching: Proof of Correctness

Claim. Flow f of value k in G' induces matching of cardinality k in G.

- By integrality theorem, there exists {0, 1}-valued flow f of value k.
- Consider M = set of edges from L to R with f(e) = 1.
  - each node in L and R participates in at most one edge in M
  - |M| = k: consider cut (L ∪ s, R ∪ t)



Perfect Matching

Perfect matching.

- Input: undirected graph G = (V, E).
- A matching M ⊆ E is perfect if each node appears in exactly one edge in M.

Perfect bipartite matching.

- Input: undirected, bipartite graph G = (L ∪ R, E), |L| = |R| = n.
- Can determine if bipartite graph has perfect matching by running matching algorithm.

Is there an easy way to convince someone that a bipartite graph does not have a perfect matching?

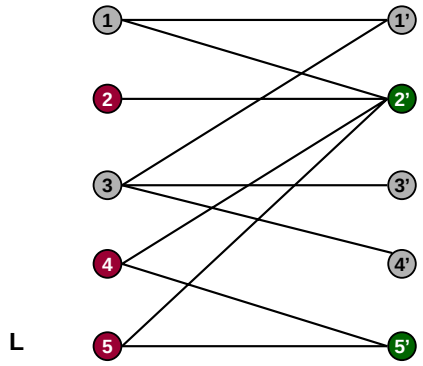
- Need good characterization of such graphs.

Perfect Matching

Let X be a subset of nodes, and let N(X) be the set of nodes adjacent to nodes in x.

Observation. If a bipartite graph G = (L ∪ R, E), has a perfect matching, then |N(X)| ≥ |X| for every X ⊆ L.

- Each node in X has to be matched to a different node in N(X).

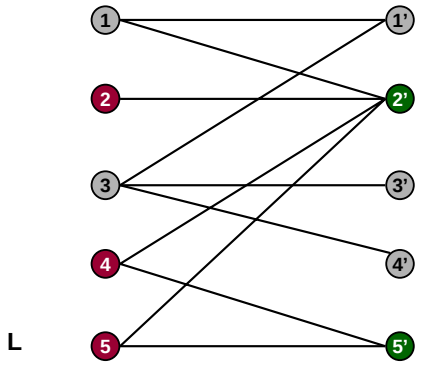


No perfect matching:
X = {2, 4, 5}, N(X) = {2', 5'}.

Perfect Matching

Hall's Theorem. Let G = (L ∪ R, E) be a bipartite graph with |L| = |R|. Then, either

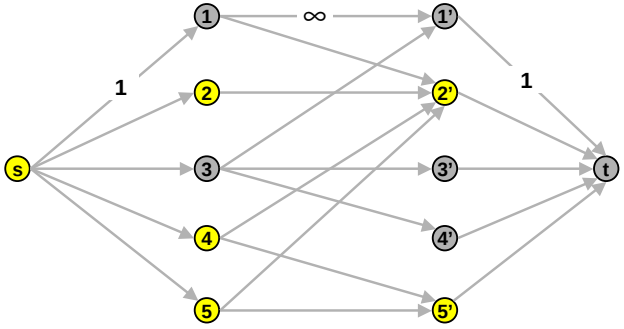
- (i) G either has a perfect matching, or
- (ii) There exists a subset X ⊆ L such that |N(X)| < |X|.



Perfect Matching

Proof. Suppose G does not have perfect matching. Then, there exists a subset $X \subseteq L$ such that $|N(X)| < |X|$.

- Let (S, T) be min cut. By max-flow min-cut, cap(S, T) < n.

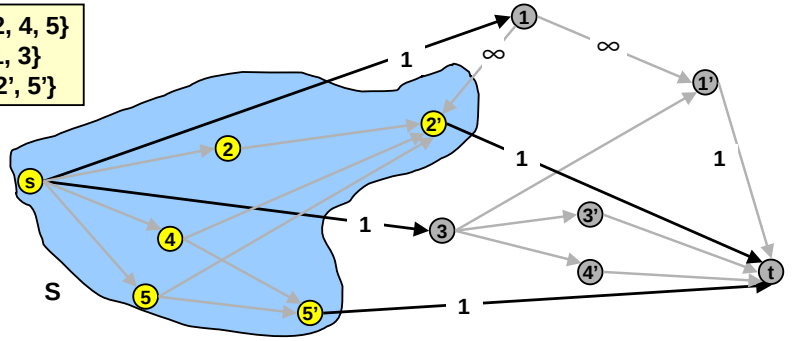


Perfect Matching

Proof. Suppose G does not have perfect matching. Then, there exists a subset $X \subseteq L$ such that $|N(X)| < |X|$.

- Let (S, T) be min cut. By max-flow min-cut, cap(S, T) < n.
- Define $X = L_S = L \cap S$, $L_T = L \cap S$, $R_S = R \cap S$.
- cap(S, T) = $|L_T| + |R_S| \Rightarrow |R_S| < |L_S|$.
- For all arcs $(v, w) \in E$: $v \in S \Rightarrow w \in S$. (min cut can't use ∞ arcs)
  - $N(L_S) \subseteq R_S \Rightarrow |N(L_S)| \leq |R_S|$.

$L_S = \{2, 4, 5\}$
$L_T = \{1, 3\}$
$R_S = \{2', 5'\}$

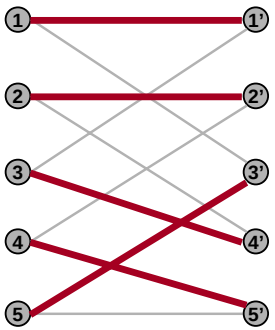


Dancing Problem (k-Regular Bipartite Graph)

Dancing problem.

- Exclusive Ivy league party attended by n men and n women.
- Each man knows exactly k women.
- Each woman knows exactly k men.
- Acquaintances are mutual.
- Is it possible to arrange a dance so that each man dances with a different woman that he knows?

Mathematical reformulation: does every k-regular bipartite graph have a perfect matching?



Dancing Problem (k-Regular Bipartite Graph)

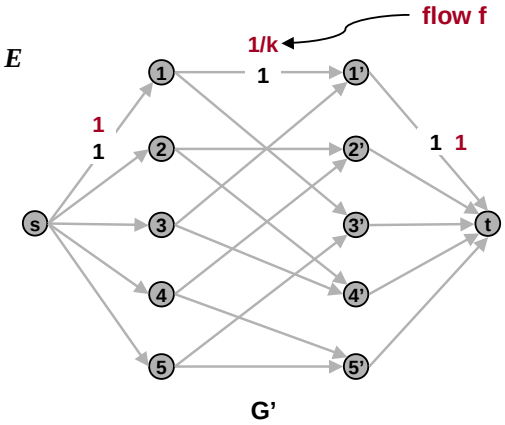
Mathematical reformulation: does every k-regular bipartite graph have a perfect matching?

Slick solution:

- Size of max matching is equal to max flow in network G'.
- Consider following flow:

$$f(v,w) = \begin{cases} 1/k & \text{if } (v,w) \in E \\ 1 & \text{if } v = s \\ 1 & \text{if } w = t \\ 0 & \text{otherwise} \end{cases}$$

- f is a flow and | f | = n.
- Integrality theorem $\Rightarrow$ integral flow of value n $\Rightarrow$ perfect matching.

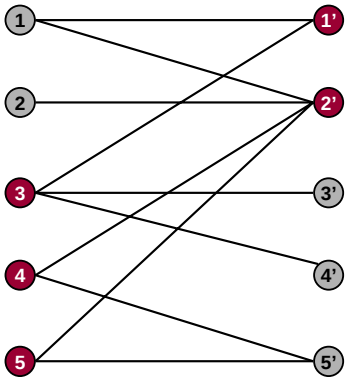


G'

Vertex Cover

Given an undirected graph G = (V, E), a vertex cover is a subset of vertices C ⊆ V such that:

- Every arc (v, w) ∈ E has either v ∈ C or w ∈ C or both.

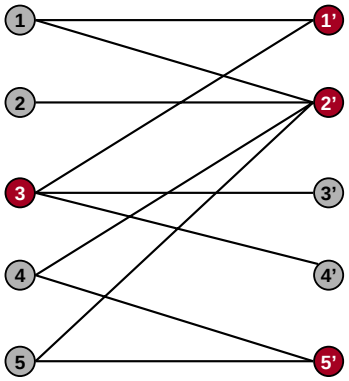


C = { 3, 4, 5, 1', 2' }
|C| = 5

Vertex Cover

Given an undirected graph G = (V, E), a vertex cover is a subset of vertices C ⊆ V such that:

- Every arc (v, w) ∈ E has either v ∈ C or w ∈ C or both.



C = { 3, 1', 2', 5' }
|C| = 4

Vertex Cover

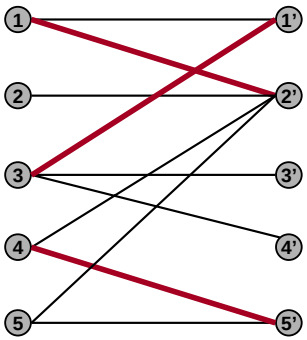
Given an undirected graph G = (V, E), a vertex cover is a subset of vertices C ⊆ V such that:

- Every arc (v, w) ∈ E has either v ∈ C or w ∈ C or both.

Observation. Let M be a matching, and let C be a vertex cover. Then, |M| ≤ |C|.

- Each vertex can cover at most one edge in any matching.

M = { 1-2', 3-1', 4-5' }
|M| = 3



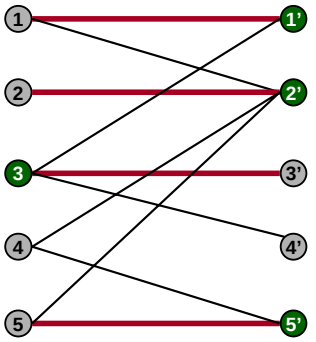
Vertex Cover: König-Egerváry Theorem

Given an undirected graph G = (V, E), a vertex cover is a subset of vertices C ⊆ V such that:

- Every arc (v, w) ∈ E has either v ∈ C or w ∈ C or both.

König-Egerváry Theorem: In a bipartite, undirected graph the max cardinality of a matching is equal to the min cardinality of a vertex cover.

M* = { 1-1', 2-2', 3-3', 5-5' }
|M*| = 4



C* = { 3, 1', 2', 5' }
|C*| = 4

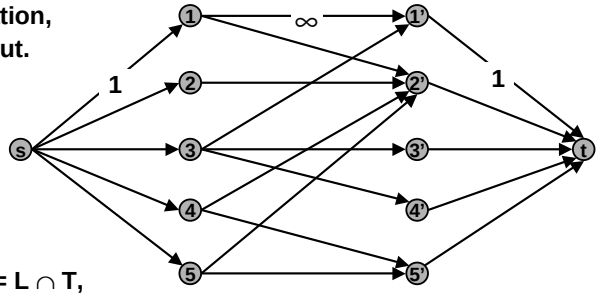
L

R

Vertex Cover: Proof of König-Egerváry Theorem

König-Egerváry Theorem: In a bipartite, undirected graph, the sizes of max matching and min vertex cover are equal.

- Suffices to find matching M* and cover C* such that |M*| = |C*|.
- Use max flow formulation, and let (S, T) be min cut.

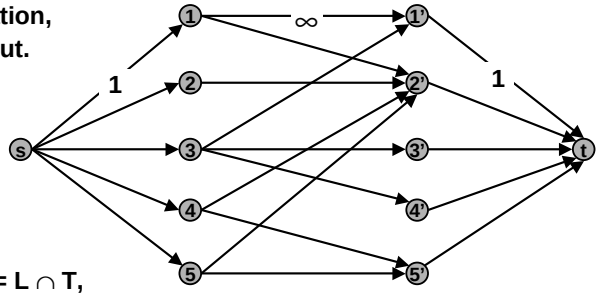


- Define $L_S = L \cap S$, $L_T = L \cap T$, $R_S = R \cap S$, $R_T = R \cap T$, and $C^* = L_T \cup R_S$.
- Claim 1. C* is a vertex cover.
  - consider $(v, w) \in E$
  - $v \in L_S$, $w \in R_T$ impossible since infinite capacity
  - thus, $v \in L_T$ or $w \in R_S$ or both

Vertex Cover: Proof of König-Egerváry Theorem

König-Egerváry Theorem: In a bipartite, undirected graph, the sizes of max matching and min vertex cover are equal.

- Suffices to find matching M* and cover C* such that |M*| = |C*|.
- Use max flow formulation, and let (S, T) be min cut.



- Define $L_S = L \cap S$, $L_T = L \cap T$, $R_S = R \cap S$, $R_T = R \cap T$, and $C^* = L_T \cup R_S$.
- Claim 1. C* is a vertex cover.
- Claim 2. |C*| = |M*|.
  - max-flow min-cut theorem $\Rightarrow$ |M*| = cap(S, T)
  - only arcs of form (s, v) or (w, t) contribute to cap(S, T)
  - $|M^*| = u(S, T) = |L_T| + |R_S| = |C^*|$.

Bipartite Matching and Vertex Cover

Which max flow algorithm to use for bipartite matching / vertex cover?

- Generic augmenting path: O(m |f *|) = O(mn).
- Capacity scaling: $O(m^2 \log U) = O(m^2)$.
- Shortest augmenting path: $O(m n^2)$.

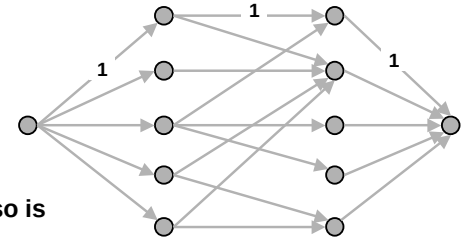
Seems to indicate "more clever" algorithms are not as good as we first thought.

No - just need more clever analysis! For bipartite matching, shortest augmenting path algorithm runs in $O(m n^{1/2})$ time.

Unit Capacity Simple Networks

Unit capacity simple network.

- Every arc capacity is one.
- Every node has either:
  (i) at most one incoming arc, or
  (ii) at most one outgoing arc.
- If G is simple unit capacity, then so is G_f, assuming f is {0, 1} flow.



Shortest augmenting path algorithm.

- Normal augmentation: length of shortest path doesn't change.
- Special augmentation: length of shortest path strictly increases.

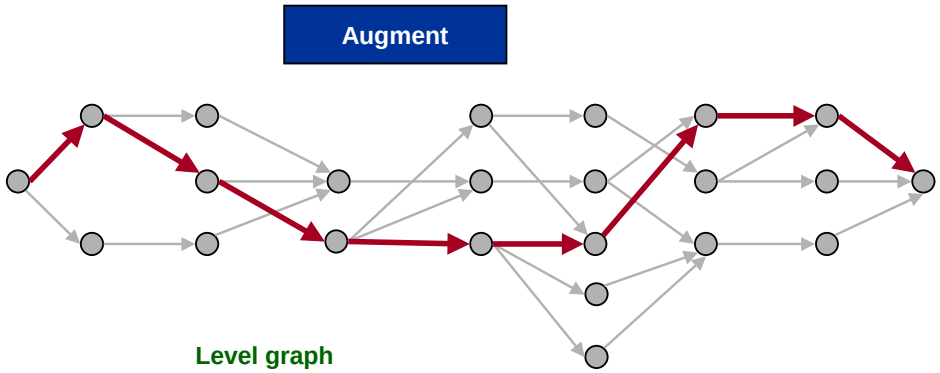
Theorem. Shortest augmenting path algorithm runs in $O(m n^{1/2})$ time.

- L1. Each phase of normal augmentations takes O(m) time.
- L2. After at most $n^{1/2}$ phases, $|f| \geq |f^*| - n^{1/2}$.
- L3. After at most $n^{1/2}$ additional augmentations, flow is optimal.

Unit Capacity Simple Networks

Lemma 1. Phase of normal augmentations takes O(m) time.

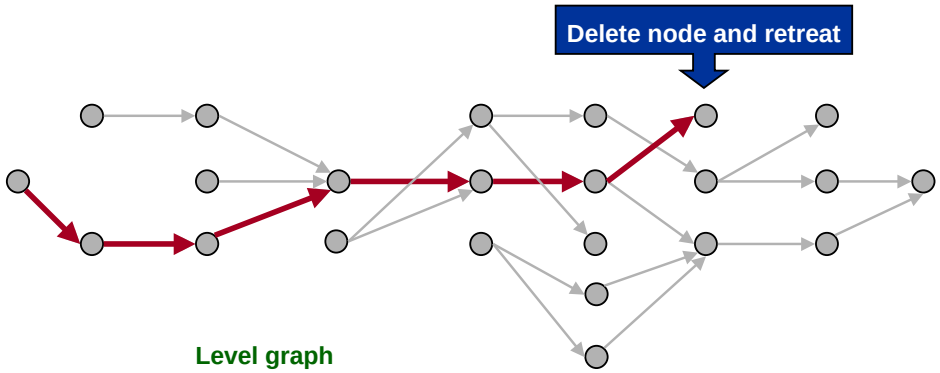
- Start at s, advance along an arc in L_G until reach t or get stuck.
  - if reach t, augment and delete ALL arcs on path
  - if get stuck, delete node and go to previous node



Unit Capacity Simple Networks

Lemma 1. Phase of normal augmentations takes O(m) time.

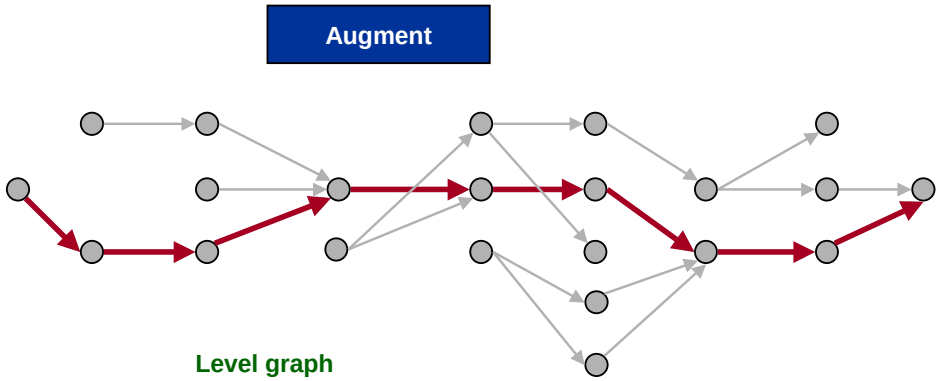
- Start at s, advance along an arc in L_G until reach t or get stuck.
  - if reach t, augment and delete ALL arcs on path
  - if get stuck, delete node and go to previous node



Unit Capacity Simple Networks

Lemma 1. Phase of normal augmentations takes O(m) time.

- Start at s, advance along an arc in L_G until reach t or get stuck.
  - if reach t, augment and delete ALL arcs on path
  - if get stuck, delete node and go to previous node

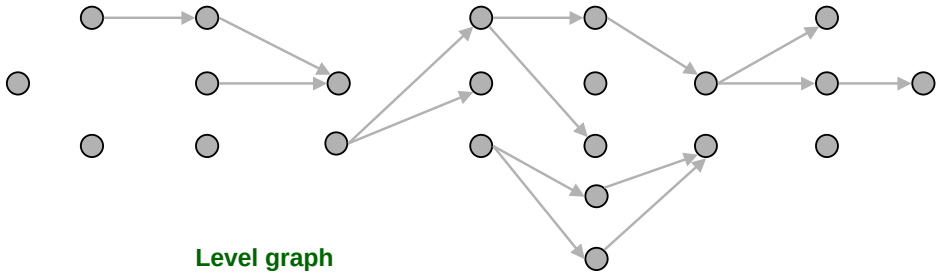


Unit Capacity Simple Networks

Lemma 1. Phase of normal augmentations takes O(m) time.

- Start at s, advance along an arc in L_G until reach t or get stuck.
  - if reach t, augment and delete ALL arcs on path
  - if get stuck, delete node and go to previous node

STOP
Length of shortest path has increased.



Unit Capacity Simple Networks

Lemma 1. Phase of normal augmentations takes O(m) time.

- Start at s, advance along an arc in L_G until reach t or get stuck.
  - if reach t, augment and delete ALL arcs on path
  - if get stuck, delete node and go to previous node

- O(m) running time.
  - O(m) to create level graph
  - O(1) per arc, since each arc traversed at most once
  - O(1) per node deletion

Unit Capacity Simple Networks

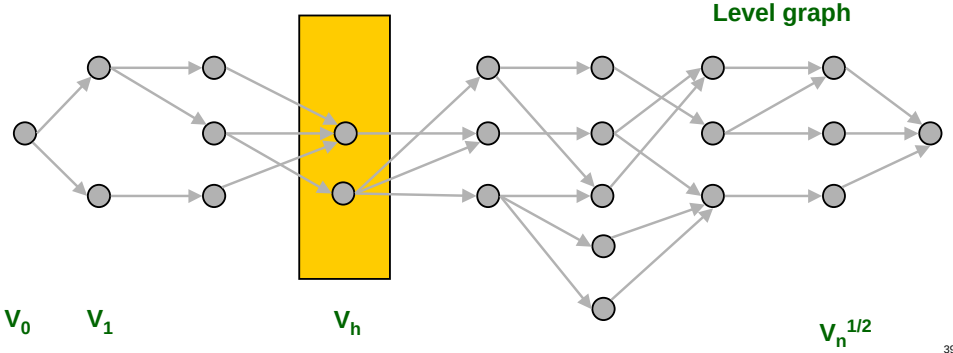
AdvanceRetreat(V, E, f, s, t)

```
ARRAY pred[v ∈ V]
L_G ← level graph of G_f
v ← s, pred[v] ← nil

REPEAT
   WHILE (there exists (v,w) ∈ L_G)
      pred[w] ← v, v ← w                    advance
      IF (v = t)
         P ← path defined by pred[]
         f ← augment(f, P)                  augment
         update L_G
         v ← s, pred[v] ← nil
   delete v from L_G                        retreat
UNTIL (v = s)

RETURN f
```


Unit Capacity Simple Networks

Lemma 2. After at most $n^{1/2}$ phases, $|f| \geq |f^*| - n^{1/2}$.

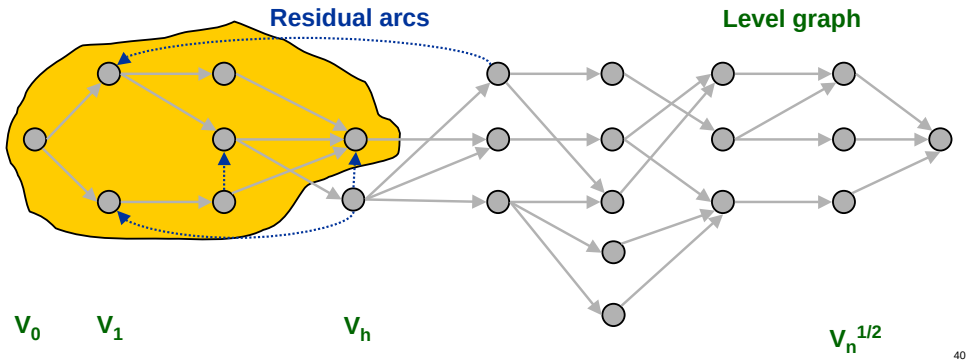
- After $n^{1/2}$ phases, length of shortest augmenting path is $> n^{1/2}$.
- Level graph has more than $n^{1/2}$ levels.
- Let $1 \leq h \leq n^{1/2}$ be layer with min number of nodes: $|V_h| \leq n^{1/2}$.



Unit Capacity Simple Networks

Lemma 2. After at most $n^{1/2}$ phases, $|f| \geq |f^*| - n^{1/2}$.

- After $n^{1/2}$ phases, length of shortest augmenting path is $> n^{1/2}$.
- Level graph has more than $n^{1/2}$ levels.
- Let $1 \leq h \leq n^{1/2}$ be layer with min number of nodes: $|V_h| \leq n^{1/2}$.
- $S := \{v : \ell(v) < h\} \cup \{v : \ell(v) = h$ and v has ≤ 1 outgoing residual arc$\}$.
- $cap_f(S, T) \leq |V_h| \leq n^{1/2} \Rightarrow |f| \geq |f^*| - n^{1/2}$.



San Francisco Chronicle

Sports Online

SPORTING G

49ers, Young Get Big Bre

Quarterback m

Giants Officially Leave the NL West Race

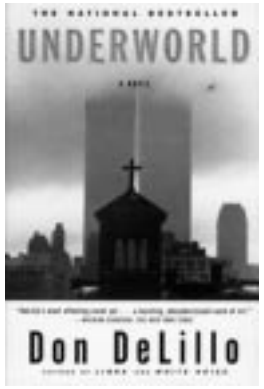
Finessing to Place For Giants' New Stadium

CARDINALS 6 GIANTS 2

Baseball Elimination

Over on the radio side the producer's saying,

- "See that thing in the paper last week about Einstein? . . . Some reporter asked him to figure out the mathematics of the pennant race. You know, one team wins so many of their remaining games, the other teams win this number or that number. What are the myriad possibilities? Who's got the edge?"
- "The hell does he know?"
- "Apparently not much. He picked the Dodgers to eliminate the Giants last Friday."



Baseball Elimination

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}			
				Atl	Phi	NY	Mon
Atlanta	83	71	8	-	1	6	1
Philly	80	79	3	1	-	0	2
New York	78	78	6	6	0	-	0
Montreal	77	82	3	1	2	0	-

Which teams have a chance of finishing the season with most wins?

- Montreal eliminated since it can finish with at most 80 wins, but Atlanta already has 83.
- $w_i + r_i < w_j \Rightarrow$ team i eliminated.
- Only reason sports writers appear to be aware of.
- Sufficient, but not necessary!

Baseball Elimination

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}			
				Atl	Phi	NY	Mon
Atlanta	83	71	8	-	1	6	1
Philly	80	79	3	1	-	0	2
New York	78	78	6	6	0	-	0
Montreal	77	82	3	1	2	0	-

Which teams have a chance of finishing the season with most wins?

- Philly can win 83, but still eliminated . . .
- If Atlanta loses a game, then some other team wins one.

Answer depends not just on how many games already won and left to play, but also on whom they're against.

Baseball Elimination

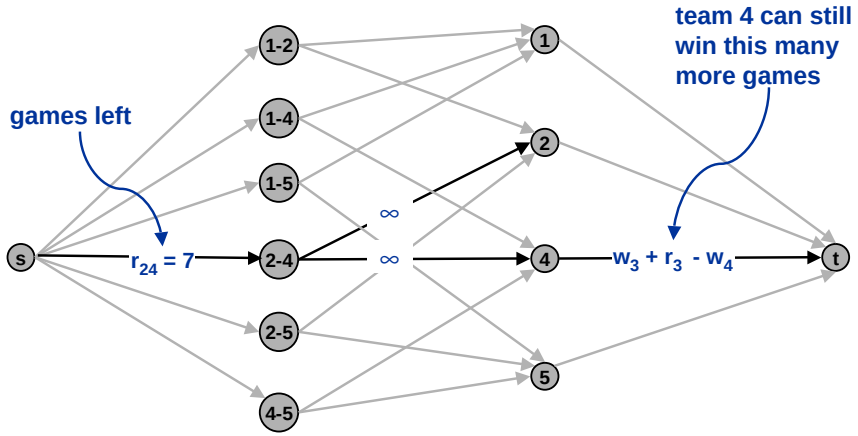
Baseball elimination problem.

- Set of teams X.
- Distinguished team $x \in X$.
- Team i has won w_i games already.
- Teams i and j play each other r_{ij} additional times.
- Is there any outcome of the remaining games in which team x finishes with the most (or tied for the most) wins?

Baseball Elimination: Max Flow Formulation

Can team 3 finish with most wins?

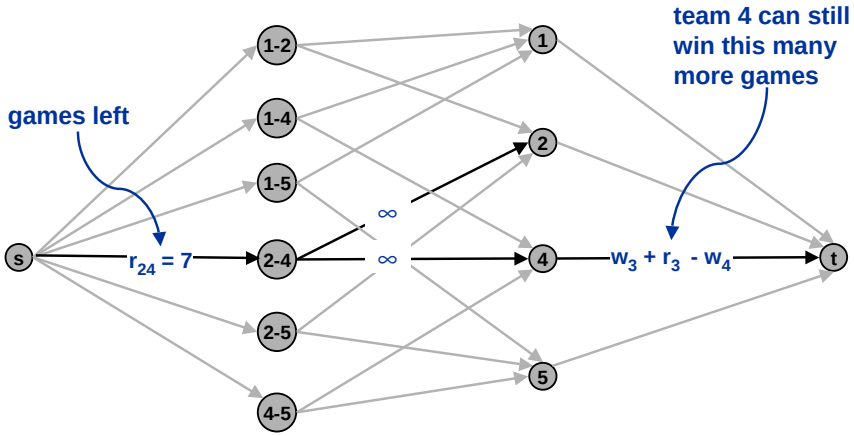
- Assume team 3 wins all remaining games $\Rightarrow$ $w_3 + r_3$ wins.
- Divvy remaining games so that all teams have $\leq$ $w_3 + r_3$ wins.



Baseball Elimination: Max Flow Formulation

Theorem. Team 3 is not eliminated if and only if max flow saturates all arcs leaving source.

- Integrality theorem $\Rightarrow$ each remaining game between i and j added to number of wins for team i or team j.
- Capacity on (v, t) arcs ensure no team wins too many games.



Baseball Elimination: Explanation for Sports Writers

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}				
				NY	Bal	Bos	Tor	Det
NY	75	59	28	-	3	8	7	3
Baltimore	71	63	28	3	-	2	7	4
Boston	69	66	27	8	2	-	0	0
Toronto	63	72	27	7	7	0	-	-
Detroit	49	86	27	3	4	0	0	-

AL East: August 30, 1996

Which teams have a chance of finishing the season with most wins?

- Detroit could finish season with 49 + 27 = 76 wins.

Baseball Elimination: Explanation for Sports Writers

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}				
				NY	Bal	Bos	Tor	Det
NY	75	59	28	-	3	8	7	3
Baltimore	71	63	28	3	-	2	7	4
Boston	69	66	27	8	2	-	0	0
Toronto	63	72	27	7	7	0	-	-
Detroit	49	86	27	3	4	0	0	-

AL East: August 30, 1996

Which teams have a chance of finishing the season with most wins?

- Detroit could finish season with 49 + 27 = 76 wins.

Consider subset R = {NY, Bal, Bos, Tor}

- Have already won w(R) = 278 games.
- Must win at least r(R) = 27 more.
  - Average team in R wins at least 305/4 > 76 games.

Baseball Elimination: Explanation for Sports Writers

Certificate of elimination.

$$R \subseteq X, \quad w(R) := \overbrace{\sum_{i \in R} w_i}^{\#\text{ wins}}, \quad r(R) := \overbrace{\frac{1}{2} \sum_{i,j \in R} r_{ij}}^{\#\text{ remaining games}},$$

If $\overbrace{\frac{w(R)+r(R)}{|R|}}^{\text{LB on avg \# games won}} > w_x + r_x$ then x is eliminated (by R).

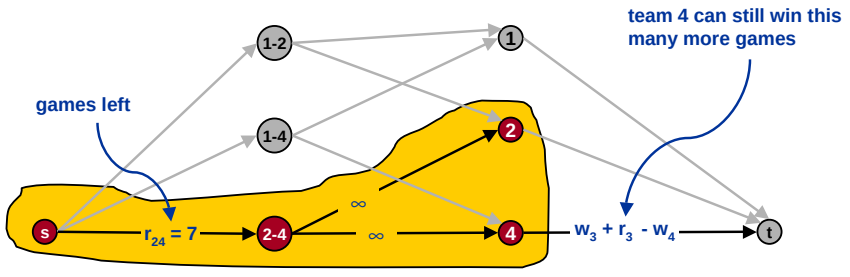
Theorem (Hoffman-Rivlin, 1967). Team x is eliminated if and only if there exists a subset R that eliminates x.

- Proof idea. Let R = team nodes on source side of min cut.

Baseball Elimination: Explanation for Sports Writers

Proof of Theorem.

- Use max flow formulation, and consider min cut (S, T).
- Define R = team nodes on source side of min cut = T ∩ S.
- Claim. i-j ∈ S if and only if i ∈ R and j ∈ R.
  - infinite capacity arcs ensure if i-j ∈ S then i ∈ S and j ∈ S
  - if i ∈ S and j ∈ S but i-j ∈ T, then adding i-j to S decreases capacity of cut

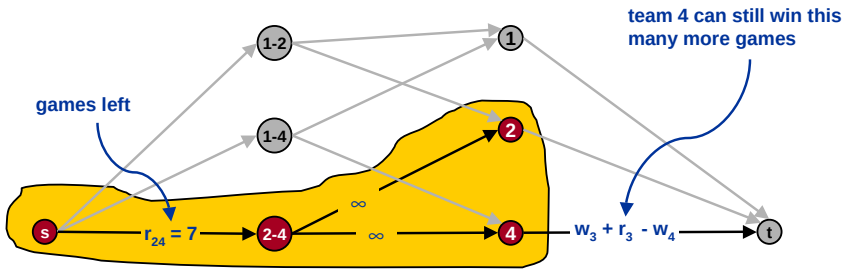


Baseball Elimination: Explanation for Sports Writers

Proof of Theorem.

- Use max flow formulation, and consider min cut (S, T).
- Define R = team nodes on source side of min cut = T ∩ S.
- Claim. i-j ∈ S if and only if i ∈ R and j ∈ R.

$$\begin{aligned} r(X - \{x\}) &> cap(S,T) \\ &= r(X - \{x\}) - r(R) + \sum_{i \in R} (w_x + r_x - w_i) \end{aligned}$$



Baseball Elimination: Explanation for Sports Writers

Proof of Theorem.

- Use max flow formulation, and consider min cut (S, T).
- Define R = team nodes on source side of min cut = T ∩ S.
- Claim. i-j ∈ S if and only if i ∈ R and j ∈ R.

$$\begin{aligned} r(X - \{x\}) &> cap(S, T) \\ &= r(X - \{x\}) - r(R) + \sum_{i \in R} (w_x + r_x - w_i) \\ &= r(X - \{x\}) - r(R) - w(R) + |R|(w_x + r_x) \end{aligned}$$

- Rearranging terms:

$$w_x + r_x < \frac{w(R) + r(R)}{|R|}.$$

Circulation with Demands

Circulation with demands.

- Directed graph G = (V, E).
- Arc capacities u(e), e ∈ E.
- Node supply and demands d(v), v ∈ V.
  - demand if d(v) > 0; supply if d(v) < 0; transshipment if d(v) = 0

A circulation is a function f: E → ℜ that satisfies:

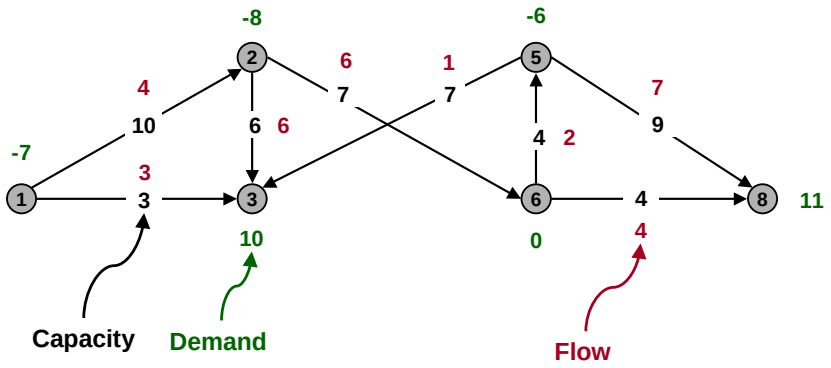
- For each e ∈ E: $0 \le f(e) \le u(e)$ (capacity)
- For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem: given (V, E, u, d), does there exist a circulation?

Circulation with Demands

A circulation is a function f: E → ℜ that satisfies:

- For each e ∈ E: $0 \le f(e) \le u(e)$ (capacity)
- For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)



Circulation with Demands

A circulation is a function f: E → ℜ that satisfies:

- For each e ∈ E: $0 \le f(e) \le u(e)$ (capacity)
- For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Necessary condition: sum of supplies = sum of demands.

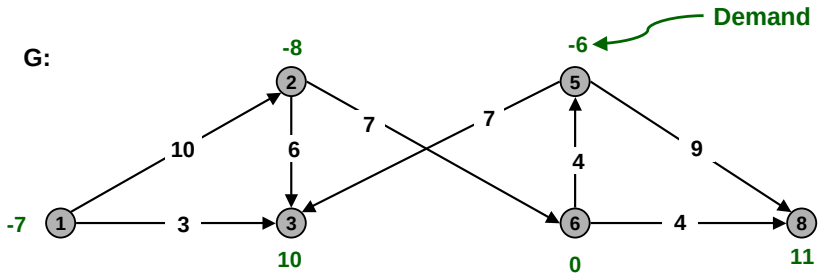
$$\sum_{v : d(v) > 0} d(v) = \sum_{v : d(v) < 0} -d(v) =: D$$

- Proof: sum conservation constraints for every demand node v.

Circulation with Demands

Max flow formulation.

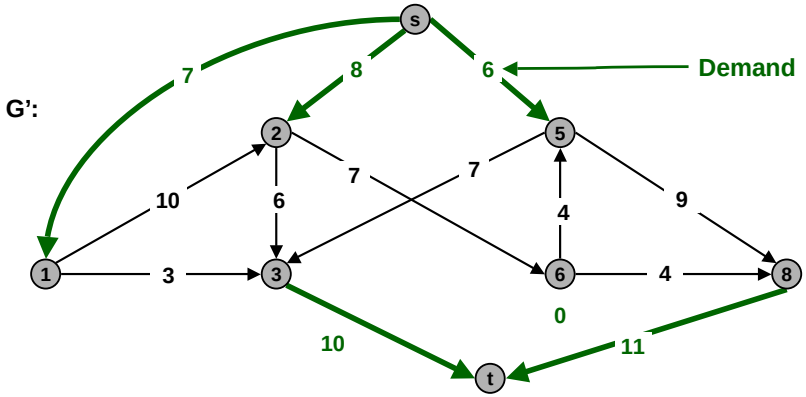
- Add new source s and sink t.
- For each v with d(v) < 0, add arc (s, v) with capacity -d(v).
- For each v with d(v) > 0, add arc (v, t) with capacity d(v).



Circulation with Demands

Max flow formulation.

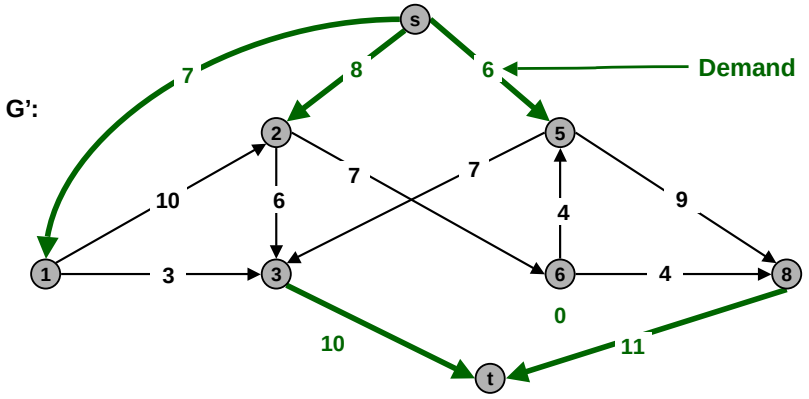
- Add new source s and sink t.
- For each v with d(v) < 0, add arc (s, v) with capacity -d(v).
- For each v with d(v) > 0, add arc (v, t) with capacity d(v).



Circulation with Demands

Max flow formulation.

- Graph G has circulation if and only if G' has max flow of value D (saturates all arcs leaving s and entering t).



Circulation with Demands

Max flow formulation.

- Graph G has circulation if and only if G' has max flow of value D (saturates all arcs leaving s and entering t).
- Moreover, if all capacities and demands are integers, and there exists a circulation, then there exists one that is integer-valued.

Characterization.

- Given (V, E, u, d), there does not exists a circulation if and only if there exists a node partition (A, B) such that:

$$\sum_{v \in B} d(v) > u(A, B).$$

  - demand by nodes in B exceeds supply of nodes in B plus max capacity of arcs going from A to B
- Proof idea: look at min cut in G'.

Circulation with Demands and Lower Bounds

Feasible circulation.

- Directed graph G = (V, E).
- Arc capacities u(e) and lower bounds $\ell(e)$, $e \in E$.
  - force flow to make use of certain edges
- Node supply and demands d(v), $v \in V$.

A circulation is a function $f: E \to \Re$ that satisfies:

- For each $e \in E$: $\ell(e) \le f(e) \le u(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

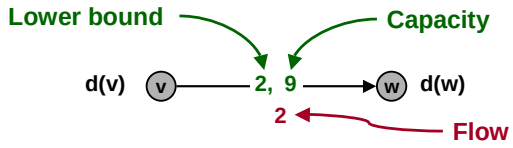
Given (V, E, ℓ, u, d), is there a circulation?

Circulation with Demands and Lower Bounds

A circulation is a function $f: E \to \Re$ that satisfies:

- For each $e \in E$: $\ell(e) \le f(e) \le u(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) = d(v)$ (conservation)

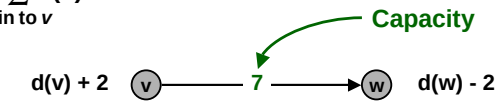
Idea: model lower bounds with supply / demands.



Create network G'.

$u'(e) = u(e) - \ell(e)$

$d'(v) = d(v) + \sum_{e \text{ out of } v} \ell(e) - \sum_{e \text{ in to } v} \ell(e)$



Circulation with Demands and Lower Bounds

A circulation is a function $f: E \to \Re$ that satisfies:

- For each $e \in E$: $\ell(e) \le f(e) \le u(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) = d(v)$ (conservation)

Create network G'.

$u'(e) = u(e) - \ell(e)$

$d'(v) = d(v) + \sum_{e \text{ out of } v} \ell(e) - \sum_{e \text{ in to } v} \ell(e)$

Theorem. There exists a circulation in G if and only if there exists a circulation in G'. If all demands, capacities, and lower bounds in G are integers, then there is a circulation in G that is integer-valued.

- Proof idea: f(e) is a circulation in G if and only if f'(e) = f(e) - $\ell(e)$ is a circulation in G'.

Matrix Rounding

Feasible matrix rounding.

- Given a p x q matrix $D = \{d_{ij}\}$ of real numbers.
- Row i sum = a_i, column j sum b_j.
- Round each d_{ij}, a_i, b_j up or down to integer so that sum of rounded elements in each row (column) equal row (column) sum.
- Original application: publishing US Census data.

Theorem: for any matrix, there exists a feasible rounding.

3.14	6.8	7.3	17.24
9.6	2.4	0.7	12.7
3.6	1.2	6.5	11.3
16.34	10.4	14.5	

Original Data

3	7	7	17
10	2	1	13
3	1	7	11
16	10	15	

Possible Rounding

Matrix Rounding

Feasible matrix rounding.

- Given a p x q matrix D = $\{d_{ij}\}$ of real numbers.
- Row i sum = a_i, column j sum b_j.
- Round each d_{ij}, a_i, b_j up or down to integer so that sum of rounded elements in each row (column) equal row (column) sum.
- Original application: publishing US Census data.

Theorem: for any matrix, there exists a feasible rounding.

- Note: "threshold rounding" doesn't work.

0.35	0.35	0.35	1.05
0.55	0.55	0.55	1.65
0.9	0.9	0.9	

Original Data

0	0	1	1
1	1	0	2
1	1	1	

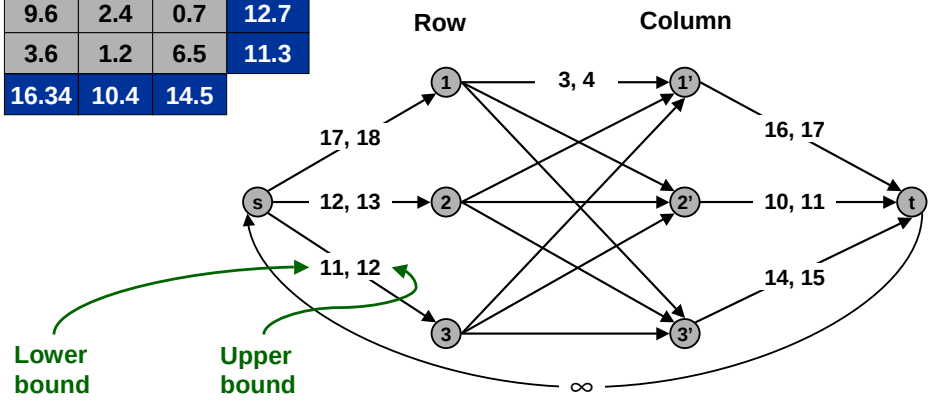
Possible Rounding

Matrix Rounding

Max flow formulation.

- Original data provides circulation (all demands 0).
- Integrality theorem $\Rightarrow$ there always exists feasible rounding!

3.14	6.8	7.3	17.24
9.6	2.4	0.7	12.7
3.6	1.2	6.5	11.3
16.34	10.4	14.5	



Airline Scheduling

Airline scheduling.

- Complex computational problem faced by nation's airline carriers.
- Produces schedules for thousands of routes each day that are efficient in terms of:
  - equipment usage, crew allocation, customer satisfaction
  - in presence of unpredictable issues like weather, breakdowns
- One of largest consumers of high-powered algorithmic techniques.

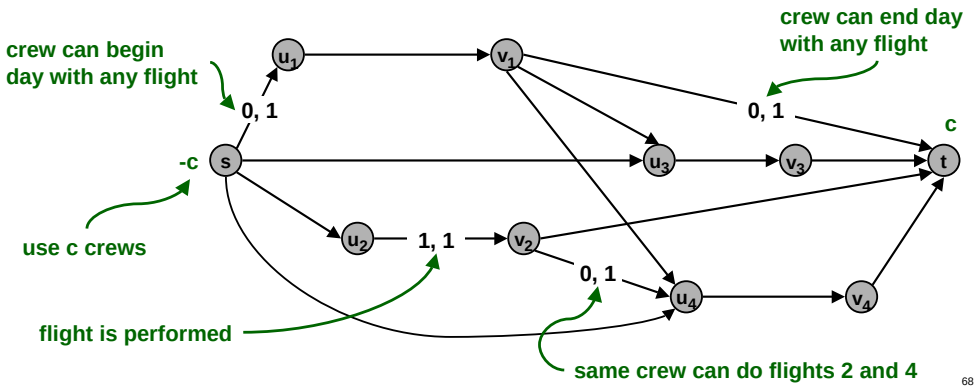
"Toy problem."

- Manage flight crews by reusing them over multiple flights.
- Input: list of cities V.
- Travel time t(v, w) from city v to w.
- Flight i: (o_i, d_i, t_i) consists of origin and destination cities, and departure time.

Airline Scheduling

Max flow formulation.

- For each flight i, include two nodes u_i and v_i.
- Add source s with demand -c, and arcs (s, u_i) with capacity 1.
- Add sink t with demand c, and arcs (v_j, t) with capacity 1.
- For each i, add arc (u_i, v_i) with lower bound and capacity 1.
- if flight j reachable from i, add arc (v_i, u_i) with capacity 1.



Airline Scheduling: Running Time

Running time.

- k = number of flights.
- O(k) nodes, $O(k^2)$ edges.
- At most k crews needed $\Rightarrow$ solve k max flow problems.
- Arc capacities between 0 and k $\Rightarrow$ at most k augmentations per max flow computation.
- Overall time = $O(k^4)$.

Remarks.

- Can solve in $O(k^3)$ time by formulating as "minimum flow problem."

Airline Scheduling: Postmortem

Airline scheduling.

- We solved toy problem.
- Real problem addresses countless other factors:
  - union regulations: e.g., flight crews can only fly certain number of hours in given interval
  - need optimal schedule over planning horizon, not just 1 day
  - deadheading has a cost
  - simultaneously trying to re-work flight schedule and re-optimize fare structure

Message.

- Our solution is a generally useful technique for efficient re-use of limited resources but trivializes real airline scheduling problem.
- Flow techniques useful for solving airline scheduling problems, and are genuinely used in practice.
- Running an airline efficiently is a very difficult problem.

Image Segmentation

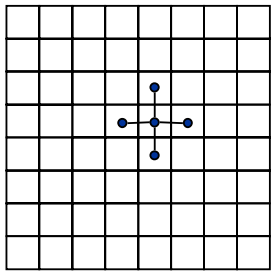
Image segmentation.

- Central problem in image processing.
- Divide image into coherent regions.
  - three people standing in front of complex background scene
  - identify each of three people as coherent objects

Image Segmentation

Foreground / background segmentation.

- Label each pixel in picture as belonging to foreground or background.
- V = set of pixels, E = pairs of neighboring pixels.
- a_v = likelihood pixel v in foreground.
- b_v = likelihood pixel v in background.
- p_{vw} = separation penalty for labeling one of v and w as foreground, and the other as background.



Goals.

- Accuracy: if $a_v > b_v$, in isolation we prefer to label pixel v in foreground.
- Smoothness: if many neighbors of v are labeled foreground, we should be inclined to label v as foreground.
- Find partition (S, T) that maximizes: $\sum_{v \in S} a_v + \sum_{v \in T} b_v - \sum_{\{v,w\} \in E \atop |S \cap \{v,w\}| = 1} p_{vw}$

Image Segmentation

Formulate as min cut problem.

- Maximization.
- No source or sink.
- Undirected graph.

Turn into minimization problem.

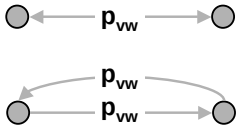
- Since $\sum_{v \in V} a_v + \sum_{v \in V} b_v$ is a constant,

maximizing $\sum_{v \in S} a_v + \sum_{v \in T} b_v - \sum_{\substack{\{v,w\} \in E \\ |S \cap \{v,w\}| = 1}} p_{vw}$

is equivalent to minimizing $\sum_{v \in T} a_v + \sum_{v \in S} b_v + \sum_{\substack{\{v,w\} \in E \\ |S \cap \{v,w\}| = 1}} p_{vw}$

Image Segmentation

Formulate as min cut problem: G' = (V', E').



- Maximization.
- No source or sink.
  - add source to correspond to foreground
  - add sink to correspond to background
- Undirected graph.
  - add two anti-parallel arcs

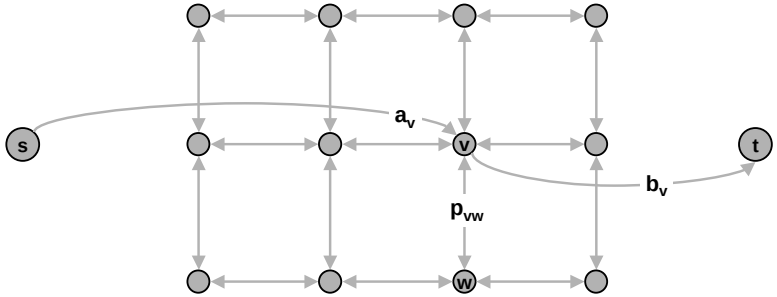
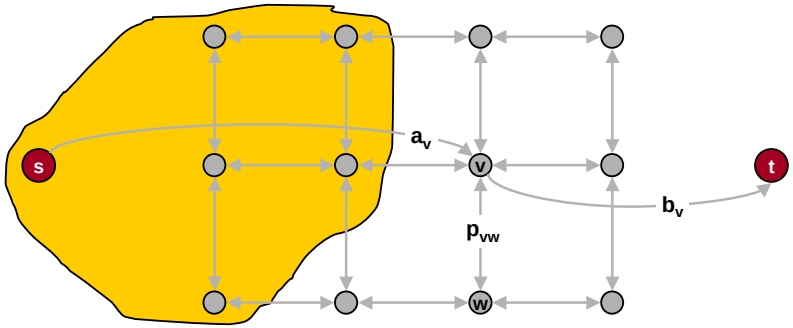


Image Segmentation

Consider min cut (S, T) in resulting graph.

- $cap(S, T) = \sum_{v \in T} a_v + \sum_{v \in S} b_v - \sum_{\substack{(v,w) \in E \\ v \in S,\ w \in T}} p_{vw}$

- Precisely the quantity we want to minimize.
  - note if v and w on different sides, p_{vw} counted exactly once



G'

Project Selection

Projects with prerequisites.

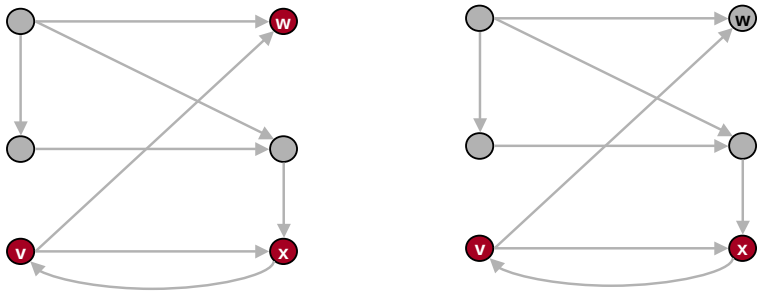
- Set P of possible projects. Project v has associated revenue p_v.
  - some projects generate money: create interactive e-commerce interface, redesign cs web page
  - others cost money: upgrade computers, get site license for encryption software
- Set of prerequisites E. If (v, w) ∈ E, can't do project v and unless also do project w.
  - can't start on e-commerce opportunity unless you've got encryption software
- A subset of projects A ⊆ P is feasible if the prerequisite of every project in A also belongs to A.
  - for each v ∈ P, and (v, w) ∈ E, we have w ∈ P

Project selection (max weight closure) problem: choose a feasible subset of projects to maximize revenue.

Project Selection

Prerequisite graph.

- Include an arc from v to w if can't do v without also doing w.
- {v, w, x} is feasible subset of projects.
- {v, x} is infeasible subset of projects.



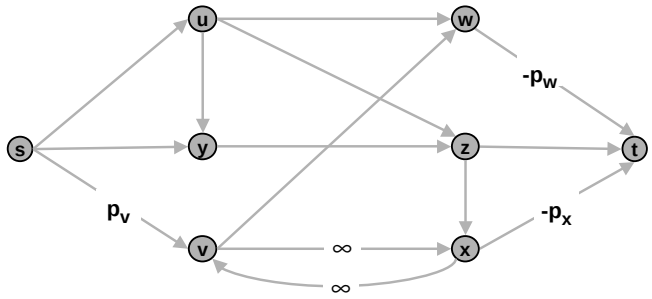
Feasible

Infeasible

Project Selection

Project selection formulation.

- Assign infinite capacity to all prerequisite arcs.
- Add arc (s, v) with capacity p_v if $p_v > 0$.
- Add arc (v, t) with capacity $-p_v$ if $p_v < 0$.
- For notational convenience, define $p_s = p_t = 0$.



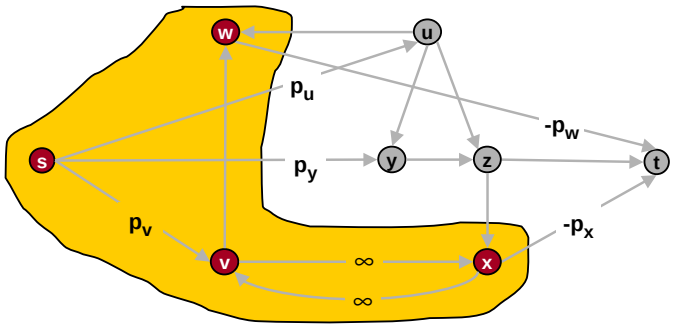
Project Selection

Claim. (S, T) is min cut if and only if S \ {s} is optimal set of projects.

- Infinite capacity arcs ensure S \ {s} is feasible.
- Max revenue because:

$$cap(S, T) = \sum_{v \in T:\, p_v > 0} p_v + \sum_{v \in S:\, p_v < 0} (-p_v)$$

$$= \underbrace{\sum_{v:\, p_v > 0} p_v}_{\text{constant}} - \sum_{v \in S} p_v$$



Project Selection

Open-pit mining (studied since early 1960's).

- Blocks of earth are extracted from surface to retrieve ore.
- Each block v has net value p_v = value of ore - processing cost.
- Can't remove block v before w or x.