

# CS 577 - Network Flow

Marc Renault

Department of Computer Sciences  
University of Wisconsin – Madison

Fall 2022

TopHat Section 001 Join Code: 102736

TopHat Section 002 Join Code: 897218



**WISCONSIN**  
UNIVERSITY OF WISCONSIN-MADISON

# NETWORK FLOW

# NETWORK FLOW

## Flow Problems

- Flow Network / Transportation Networks: Connected directed graph with water flowing / traffic moving through it.
- Edges have limited *capacities*.
- Nodes act as switches directing the flow.
- Many, many problems can be cast as flow problems.

### Ford-Fulkerson Method (1956)

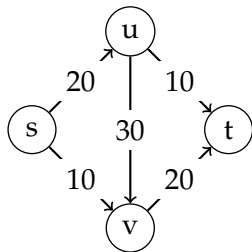


L R Ford Jr.



D. R. Fulkerson

# FLOW NETWORK



## Basic Flow Network

- Directed graph  $G = (V, E)$ .
- Each edge  $e$  has  $c_e \geq 0$ .
- Source  $s \in V$  and sink  $t \in V$ .
- Internal node  $V/\{s, t\}$ .

## Defining Flow

- Flow starts at  $s$  and exits at  $t$ .
- Flow function:  $f : E \rightarrow \mathbb{R}^+$ ;  $f(e)$  is the flow across edge  $e$ .
- Flow Conditions:

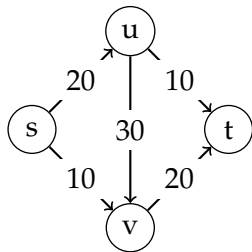
❶ Capacity: For each  $e \in E$ ,  $0 \leq f(e) \leq c_e$ .

❷ Conservation: For each  $v \in V/\{s, t\}$ ,

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

- Flow value  $v(f) = f^{\text{out}}(s) = f^{\text{in}}(t)$

# MAXIMUM-FLOW PROBLEM



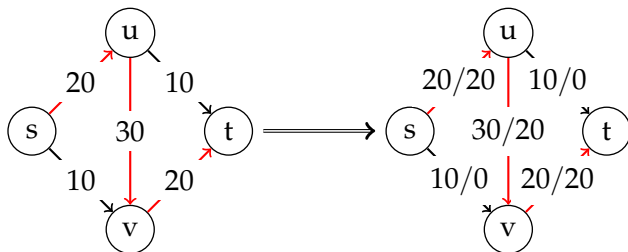
## Max-Flow

Given a flow network  $G$ , what is the maximum flow value, i.e., what is the flow  $f$  that maximizes  $v(f)$ ?

## Alternate View: Min-Cut

- A Cut: Partition of  $V$  into sets  $(A, B)$  with  $s \in A$  and  $t \in B$ .
- Flow from  $s$  to  $t$  must cross the set  $A$  to  $B$ .
- Cut capacity:  $c(A, B) = \sum_{e \text{ out of } A} c_e$
- Minimum-cut of  $G$ : The cut  $(A^*, B^*)$  that minimizes  $c(A^*, B^*)$  for  $G$ .
- The min-cut and max-flow are the same value for any flow network.

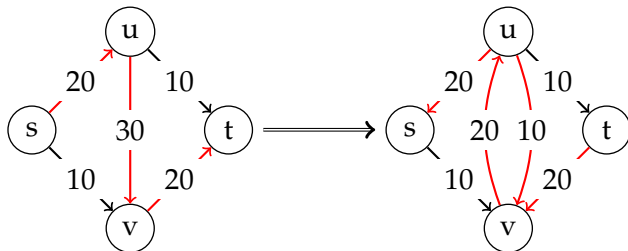
## DESIGNING THE APPROACH



### Basic Greedy Approach

- Initialize  $f(e) = 0$  for all edges.
- While there is a path from  $s$  to  $t$  with available capacity, push flow equal to the minimum available capacity along path.
- We need a mechanism to reverse flow...

# RESIDUAL GRAPH

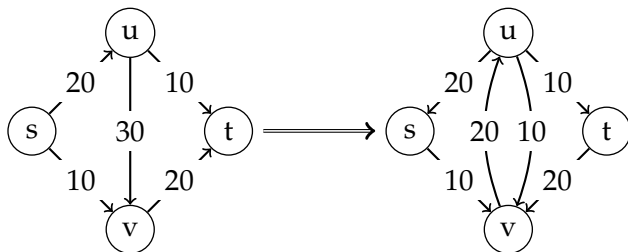


## Residual Graph

Given a flow network  $G$  and a flow  $f$  on  $G$ , we get the residual graph  $G_f$ :

- Same nodes as  $G$ .
- For edge  $(u, v)$  in  $E$ :
  - Add edge  $(u, v)$  with capacity  $c_e - f(e)$ .
  - Add edge  $(v, u)$  with capacity  $f(e)$ .

# AUGMENTING PATH



## Augmenting Path

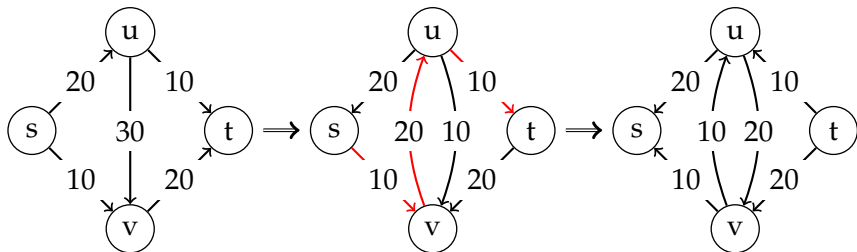
- A simple directed path from  $s$  to  $t$ .
- $\text{BOTTLENECK}(P, G_f)$ : Minimum residual capacity on augmenting path  $P$ .

## TopHat 3

List the nodes (separated by commas, i.e.  $s,u,t$ ) of an augmenting path in the example residual graph.



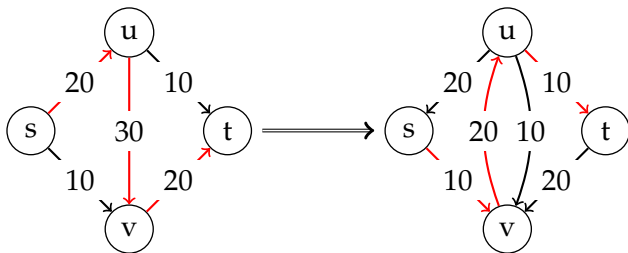
# AUGMENTING PATH



## Increasing the Flow along Augmenting Path

- Push  $\text{BOTTLENECK}(P, G_f) = q$  along path  $P$ :
  - Pushing  $q$  along a directed edge in  $G$ , increase flow by  $q$ .
  - Pushing  $q$  in opposite directed of edge in  $G$ , decreases flow by  $q$ .

# DESIGNING THE APPROACH



## Refined Greedy Approach

- Initialize  $f(e) = 0$  for all edges.
- While  $G_f$  contains an augmenting path  $P$ :
  - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f)$  along  $P$ .

# ANALYZING THE ALGORITHM

## CONSTANT INCREASE AND TERMINATION

### Observation 1

*If all capacities are integers, then all  $f(e)$ , residual capacities, and  $v(f)$  are integers at every iteration.*

### Refined Greedy Approach

- Initialize  $f(e) = 0$  for all edges.
- While  $G_f$  contains an augmenting path  $P$ :
  - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f)$  along  $P$ .

### TopHat 4

What technique should we use to prove the observation?

# ANALYZING THE ALGORITHM

## CONSTANT INCREASE AND TERMINATION

### Observation 1

*If all capacities are integers, then all  $f(e)$ , residual capacities, and  $v(f)$  are integers at every iteration.*

### Refined Greedy Approach

- Initialize  $f(e) = 0$  for all edges.
- While  $G_f$  contains an augmenting path  $P$ :
  - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f)$  along  $P$ .

### Lemma 1

$v(f') > v(f)$ , where  $v(f') = v(f) + \text{BOTTLENECK}(P, G_f)$  for an augmenting path  $P$  in  $G_f$ .

### Proof.

By definition of  $P$ , first edge of  $p$  is an out edge from  $s$  that we increase by  $\text{BOTTLENECK}(P, G_f) = q$ . By the law of conservation, this will give  $q$  more flow. □

# ANALYZING THE ALGORITHM

## CONSTANT INCREASE AND TERMINATION

### Observation 1

*If all capacities are integers, then all  $f(e)$ , residual capacities, and  $v(f)$  are integers at every iteration.*

### Refined Greedy Approach

- Initialize  $f(e) = 0$  for all edges.
- While  $G_f$  contains an augmenting path  $P$ :
  - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f)$  along  $P$ .

### Theorem 2

*Let  $C = \sum_{e \text{ out of } s} c_e$ , the FF method terminates in at most  $C$  iterations.*

### Proof.

From Lemma 1, the flow strictly increases at each iteration. Hence, the residual capacity out of  $s$  decreases by at least 1 at each iteration. □

# ANALYZING THE ALGORITHM

## RUNTIME

### Observation 2

*Since  $G$  is connected,  $m \geq n - 1$ . Hence,  $O(m + n) = O(m)$ .*

### Refined Greedy Approach

- Initialize  $f(e) = 0$  for all edges.
- While  $G_f$  contains an augmenting path  $P$ :
  - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f)$  along  $P$ .

### Theorem 3

*Suppose all capacities are integers. Then, runtime of  $O(mC)$ .*

### TopHat 7

Is this a polynomial bound? No, it is pseudo-polynomial.

# ANALYZING THE ALGORITHM

## RUNTIME

### Observation 2

Since  $G$  is connected,  
 $m \geq n - 1$ . Hence,  
 $O(m + n) = O(m)$ .

### Refined Greedy Approach

- Initialize  $f(e) = 0$  for all edges.
- While  $G_f$  contains an augmenting path  $P$ :
  - Update flow  $f$  by  
 $\text{BOTTLENECK}(P, G_f)$  along  $P$ .

### Theorem 3

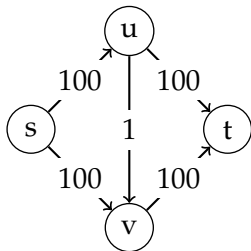
Suppose all capacities are integers. Then, runtime of  $O(mC)$ .

### Proof.

- Theorem 2: termination happens in at most  $C$  iterations.
- Work per iteration: Overall:  $O(m)$ 
  - ① Find an augmenting path: BFS or DFS:  $O(m + n)$ .
  - ② Update flow along path  $P$ :  $O(n)$ .
  - ③ Build new  $G_f$ :  $O(m)$ .



# CHOOSING GOOD AUGMENTING PATHS



## Idea

- Choose paths with large bottlenecks.
- Let  $G_f(\Delta)$  be a residual graph with edges of residual capacity  $\geq \Delta$ .

## Scaled Version

- Initialize  $f(e) = 0$  for all edges.
- Initialize  $\Delta := \max_i (2^i)$  such that  $2^i \leq \max_{e \text{ out of } s} (c_e)$ .
- While  $\Delta \geq 1$ :
  - While  $G_f(\Delta)$  contains an augmenting path  $P$ :
    - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f(\Delta))$  along  $P$ .
  - Set  $\Delta := \Delta/2$ .



# ANALYZING THE SCALED VERSION

## Scaled Version

- Initialize  $f(e) = 0$  for all edges.
- Initialize  $\Delta := \max_i (2^i)$  such that  $2^i \leq \max_{e \text{ out of } s} (c_e)$ .
- While  $\Delta \geq 1$ :
  - While  $G_f(\Delta)$  contains an augmenting path  $P$ :
    - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f(\Delta))$  along  $P$ .
  - Set  $\Delta := \Delta/2$ .

## Termination

- As before, inner loop always terminates.
- Outer loop advances to 1.

# ANALYZING THE SCALED VERSION

## Scaled Version

- Initialize  $f(e) = 0$  for all edges.
- Initialize  $\Delta := \max_i (2^i)$  such that  $2^i \leq \max_{e \text{ out of } s} (c_e)$ .
- While  $\Delta \geq 1$ :
  - While  $G_f(\Delta)$  contains an augmenting path  $P$ :
    - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f(\Delta))$  along  $P$ .
  - Set  $\Delta := \Delta/2$ .

## Advancement

- As before, inner loop always improves the flow.
- Since last outer iteration has  $\Delta = 1$ , this returns the same max-flow value as the non-scaled version.

# ANALYZING THE SCALED VERSION

## Scaled Version

- Initialize  $f(e) = 0$  for all edges.
- Initialize  $\Delta := \max_i (2^i)$  such that  $2^i \leq \max_{e \text{ out of } s} (c_e)$ .
- While  $\Delta \geq 1$ :
  - While  $G_f(\Delta)$  contains an augmenting path  $P$ :
    - Update flow  $f$  by  $\text{BOTTLENECK}(P, G_f(\Delta))$  along  $P$ .
  - Set  $\Delta := \Delta/2$ .

## Runtime

- Number of scaling phases:  $1 + \lceil \lg C \rceil$ .
- Number of augmenting phases per scaling phases:  $O(m)$ .
- Cost per augmentation:  $O(m)$ .
- Overall:  $O(m^2 \log C)$ .

TopHat 14: Is this polynomial? Yes, because  $\lceil \log C \rceil$  is the # of bits needed to encode  $C$ .

# STRONGLY POLYNOMIAL

## Definition

- Polynomial in the dimensions of the problem, not in the size of the numerical data.
- $m$  and  $n$  for max-flow.

## Fewest Edges Augmenting Path

$O(m^2n)$

- Edmonds-Karp (BFS) 1970
- Dinitz 1970

## Other Variations

- Dinitz 1970:  $O\left(\min\left\{n^{\frac{2}{3}}, m^{\frac{1}{2}}\right\} m\right)$ .
- Preflow-Push 1974/1986:  $O(n^3)$ .
- Best: Orlin 2013:  $O(mn)$

# MINIMUM CUT

# MAX-FLOW AND MIN-CUT

## Recall Cut

- A Cut: Partition of  $V$  into sets  $(A, B)$  with  $s \in A$  and  $t \in B$ .
- Cut capacity:  $c(A, B) = \sum_{e \text{ out of } A} c_e$ .

# MAX-FLOW AND MIN-CUT

## Lemma 4

Let  $f$  be any  $s - t$  flow and  $(A, B)$  be any  $s - t$  cut. Then,

$$v(f) = f^{\text{out}}(A) - f^{\text{in}}(A) = f^{\text{in}}(B) - f^{\text{out}}(B).$$

## Proof.

- By definition,  $f^{\text{out}}(A) = f^{\text{in}}(B)$  and  $f^{\text{in}}(A) = f^{\text{out}}(B)$ .
- By definition,  $v(f) = f^{\text{out}}(s)$

$$\begin{aligned} &= f^{\text{out}}(s) - f^{\text{in}}(s) \\ &= \sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v)) \end{aligned}$$

- Last line follows since  $\sum_{v \in A \setminus \{s\}} (f^{\text{out}}(v) - f^{\text{in}}(v)) = 0$ .

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



# MAX-FLOW AND MIN-CUT

## Lemma 4

*Let  $f$  be any  $s - t$  flow and  $(A, B)$  be any  $s - t$  cut. Then,*

$$v(f) = f^{\text{out}}(A) - f^{\text{in}}(A) = f^{\text{in}}(B) - f^{\text{out}}(B) .$$

## Lemma 5

*Let  $f$  be any  $s - t$  flow and  $(A, B)$  be any  $s - t$  cut. Then,*

$$v(f) \leq c(A, B).$$

## Proof.

$$\begin{aligned} v(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) \leq f^{\text{out}}(A) = \sum_{e \text{ out of } A} f(e) \\ &\leq \sum_{e \text{ out of } A} c_e = c(A, B) \end{aligned}$$

□



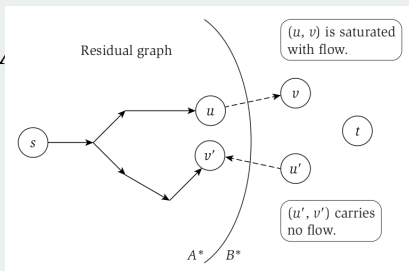
# MAX-FLOW EQUALS MIN-CUT

## Theorem 6

If  $f$  is a  $s - t$  flow such that there is no  $s - t$  path in  $G_f$ , then there is an  $s - t$  cut  $(A^*, B^*)$  in  $G$  for which  $v(f) = c(A^*, B^*)$ .

## Proof.

- Let  $A^*$  be the
- Let  $B^* = V \setminus A^*$



- Consider  $e = (u, v)$ : Claim  $f(e) = c_e$ .
  - If not, then  $s - v$  path in  $G_f$  which contradicts definition of  $A^*$  and  $B^*$ .

# MAX-FLOW EQUALS MIN-CUT

## Theorem 6

*If  $f$  is a  $s - t$  flow such that there is no  $s - t$  path in  $G_f$ , then there is an  $s - t$  cut  $(A^*, B^*)$  in  $G$  for which  $v(f) = c(A^*, B^*)$ .*

## Proof.

- Let  $A^*$  be the set of nodes for which  $\exists$  an  $s - v$  path in  $G_f$ .  
Let  $B^* = V \setminus A^*$ .
- Consider  $e = (u, v)$ : Claim  $f(e) = c_e$ .
- Consider  $e = (u', v')$ : Claim  $f(e) = 0$ .
- Therefore,

$$\begin{aligned} v(f) &= f^{\text{out}}(A^*) - f^{\text{in}}(A^*) \\ &= \sum_{e \text{ out } A^*} c_e - 0 \\ &= c(A^*, B^*) \end{aligned}$$



# MAX-FLOW EQUALS MIN-CUT

## Theorem 6

*If  $f$  is a  $s - t$  flow such that there is no  $s - t$  path in  $G_f$ , then there is an  $s - t$  cut  $(A^*, B^*)$  in  $G$  for which  $v(f) = c(A^*, B^*)$ .*

## Corollary 7

*Let  $f$  be flow from  $G_f$  with no  $s - t$  path. Then,  $v(f) = c(A^*, B^*)$  for minimum cut  $(A^*, B^*)$ .*

## Proof.

- By way of contradiction, assume  $v(f') > v(f)$ . This implies that  $v(f') > c(A^*, B^*)$  which contradicts Lemma 5.
- By way of contradiction, assume  $c(A, B) < c(A^*, B^*)$ . This implies that  $c(A, B) < v(f)$  which contradicts Lemma 5.



# MAX-FLOW EQUALS MIN-CUT

## Theorem 6

*If  $f$  is a  $s - t$  flow such that there is no  $s - t$  path in  $G_f$ , then there is an  $s - t$  cut  $(A^*, B^*)$  in  $G$  for which  $v(f) = c(A^*, B^*)$ .*

## Corollary 7

*Let  $f$  be flow from  $G_f$  with no  $s - t$  path. Then,  $v(f) = c(A^*, B^*)$  for minimum cut  $(A^*, B^*)$ .*

## Corollary 8

*Ford-Fulkerson method produces the maximum flow since it terminate when residual graph has no  $s - t$  paths.*

# FINDING THE MIN-CUT

## Theorem 9

*Given a maximum flow  $f$ , an  $s - t$  cut of minimum capacity can be found in  $O(m)$  time.*

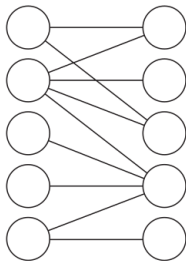
## Proof.

- Construct residual graph  $G_f$  ( $O(m)$  time).
- BFS or DFS from  $s$  to determine  $A^*$  ( $O(m + n)$  time).
- $B^* = V \setminus A^*$  ( $O(n)$  time).



# BIPARTITE MATCHING

# BIPARTITE MATCHING PROBLEM



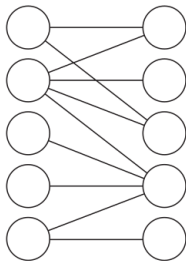
## Definition

- Bipartite Graph  $G = (V = X \cup Y, E)$ .
- All edges go between  $X$  and  $Y$ .
- Matching:  $M \subseteq E$  s.t. a node appears in only one edge.
- Goal: Find largest matching (cardinality).

## Reduction to Max-Flow Problem

- Goal: Create a flow network based on the the original problem.
- The solution to the flow network must correspond to the original problem.
- The reduction should be efficient.

# BIPARTITE MATCHING PROBLEM



## Definition

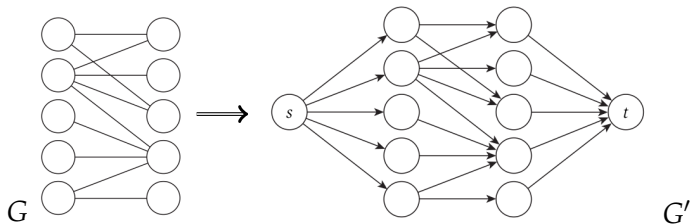
- Bipartite Graph  $G = (V = X \cup Y, E)$ .
- All edges go between  $X$  and  $Y$ .
- Matching:  $M \subseteq E$  s.t. a node appears in only one edge.
- Goal: Find largest matching (cardinality).

## Reduction to Max-Flow Problem

- How can the problem be encoded in a graph?
- Source/sink: Are they naturally in the graph encoding, or do additional nodes and edges have to be added?
- For each edge: What is the direction? Is it bi-directional? What is the capacity?



# BIPARTITE MATCHING TO FLOW NETWORK



- Add source connected to all  $X$ .
- Add sink connected to all  $Y$ .
- Original edges go from  $X$  to  $Y$ .
- Capacity of all edges is 1.

# BIPARTITE MATCHING TO FLOW NETWORK

## Theorem 10

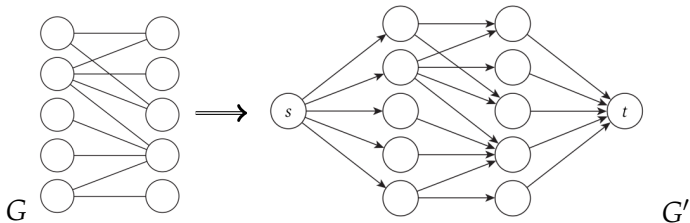
*$|M^*|$  in  $G$  is equal to the max-flow of  $G'$ , and the edges carrying the flow correspond to the edges in the maximum matching.*

## Proof.

- $s$  can send at most 1 unit of flow to each node in  $X$ .
- Since  $f^{\text{in}} = f^{\text{out}}$  for internal nodes,  $Y$  nodes can have at most 1 flow from 1 node in  $X$ .



# BIPARTITE MATCHING TO FLOW NETWORK



## Runtime

- Assume  $n = |X| = |Y|, m = |E|$ .
- Overall:  $O(mn)$ .
- Basic FF method bound:  $O(mC)$ , where  $C = f^{\text{out}}(S) \leq n$ .

# EDGE-DISJOINT PATHS

# EDGE-DISJOINT PATHS

## Problem

Given a graph  $G = (V, E)$  and two distinguished nodes  $s$  and  $t$ , find the number of edge-disjoint paths from  $s$  to  $t$ .

## Flow Network

- Directed Graph:
  - $s$  is the source and  $t$  is the sink.
  - Add capacity of 1 to every edge.
- Undirected Graph:
  - For each undirected edge  $(u, v)$ , convert to 2 directed edges  $(u, v)$  and  $(v, u)$ .
  - Apply directed graph transformation.

# EDGE-DISJOINT PATHS ANALYSIS

## Observation 3

*If there are  $k$  edge-disjoint paths in  $G$  from  $s - t$ , then the max-flow is  $k$  in  $G'$ .*

## Runtime

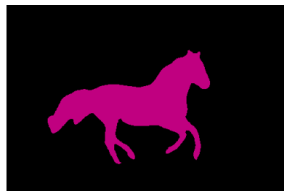
- Basic FF method:  $O(mC) = O(mn)$ .

## Path Decomposition

- Let  $f$  be a max-flow for this problem. How can we recover the  $k$  edge-disjoint paths?
- DFS from  $s$  in  $f$  along edges  $e$ , where  $f(e) = 1$ :
  - ➊ Find a simple path  $P$  from  $s$  to  $t$ : set flow to 0 along  $P$ ; continue DFS from  $s$ .
  - ➋ Find a path  $P$  with a cycle  $C$  before reaching  $t$ : set flow to 0 along  $C$ ; continue DFS from start of cycle.

# IMAGE SEGMENTATION

# IMAGE SEGMENTATION



## Problem

Let  $P$  be the set of pixels in an image. We would like to separate  $P$  into set  $A$  and  $B$ , where  $A$  are the foreground pixels and  $B$  are the background pixels.

For pixel  $i$ :

- $a_i > 0$  is the likelihood of  $i$  being in the foreground.
- $b_i > 0$  is the likelihood of  $i$  being in the background.
- For each adjacent pixel  $j$ :  $p_{ij} = p_{ji}$  is a separation penalty paid when  $i$  and  $j$  are not both  $\in A$  or  $\in B$ .



# IMAGE SEGMENTATION

## Problem

Let  $P$  be the set of pixels in an image.

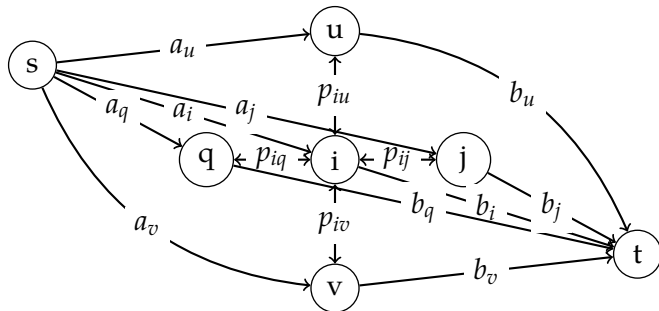
For pixel  $i$ :

- $a_i > 0$  is the likelihood of  $i$  being in the foreground.
- $b_i > 0$  is the likelihood of  $i$  being in the background.
- For each adjacent pixel  $j$ :  $p_{ij} = p_{ji}$  is a separation penalty paid when  $i$  and  $j$  are not both  $\in A$  or  $\in B$ .

## Goal

- Maximize  $q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{i, j \in P: |A \cap \{i, j\}| = 1} p_{ij}$
- Let  $Q = \sum_{i \in P} (a_i + b_i)$ .  
 $q(A, B) = Q - \sum_{i \in B} a_i - \sum_{j \in A} b_j - \sum_{i, j \in P: |A \cap \{i, j\}| = 1} p_{ij}$
- Equivalent goal:  
 Minimize  $\sum_{i \in B} a_i + \sum_{j \in A} b_j + \sum_{i, j \in P: |A \cap \{i, j\}| = 1} p_{ij}$ .

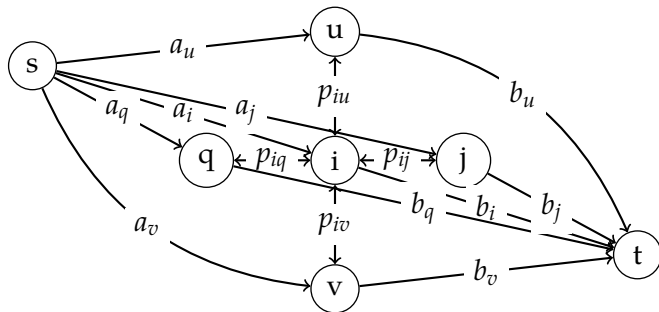
# ALGORITHM DESIGN



## Reduction

- Each pixel becomes a node.
- Add edges between neighbours  $i$  and  $j$  with capacity  $p_{ij}$ .
- Add a source  $s$  and connect to all nodes  $i$  with capacity  $a_i$ .
- Add a sink  $t$  and connect all nodes  $i$  with capacity  $b_i$  to  $t$ .

# ALGORITHM DESIGN



## Solution

- Min-cut will minimize  $\sum_{i \in B} a_i + \sum_{j \in A} b_j + \sum_{i,j \in P: |A \cap \{i,j\}|=1} p_{ij}$ .
- Consider  $i \in A$ : Foreground and contributes  $b_i$  to cut.
- Consider  $j \in B$ : Background and contributes  $a_i$  to cut.
- Consider  $i \in A, j \in B$  and  $i, j$  adjacent: contributes  $p_{ij}$  to cut.

# NODE DEMAND AND LOWER BOUNDS

# FLOW NETWORK EXTENSION

## ADDING NODE DEMAND

### Flow Network with Demand

- Each node has a demand  $d_v$ :
  - if  $d_v < 0$ : a source that demands  $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$ .
  - if  $d_v = 0$ : internal node ( $f^{\text{in}}(v) - f^{\text{out}}(v) = 0$ ).
  - if  $d_v > 0$ : a sink that demands  $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$ .
- $S$  is the set of sources ( $d_v < 0$ ).
- $T$  is the set of sinks ( $d_v > 0$ ).

### Flow Conditions

- ❶ Capacity: For each  $e \in E$ ,  $0 \leq f(e) \leq c_e$ .
- ❷ Conservation: For each  $v \in V$ ,  $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$ .

### Goal

Feasibility: Does there exist a flow that satisfies the conditions?

# FEASIBILITY

## Goal

Feasibility: Does there exist a flow that satisfies the conditions?

## Lemma 10

*If there is a feasible flow, then  $\sum_{v \in V} d_v = 0$ .*

## Proof.

- Suppose that  $f$  is a feasible flow, then, by definition, for all  $v$ ,  $d_v = f^{\text{in}}(v) - f^{\text{out}}(v)$ .
- For every edge  $e = (u, v)$ ,  $f_e^{\text{out}}(u) = f_e^{\text{in}}(v)$ . Hence,  $f_e^{\text{in}}(v) - f_e^{\text{out}}(u) = 0$ .
- $\sum_{v \in V} d_v = 0$ .



# FEASIBILITY

## Goal

Feasibility: Does there exist a flow that satisfies the conditions?

## Lemma 10

*If there is a feasible flow, then  $\sum_{v \in V} d_v = 0$ .*

## Corollary 11

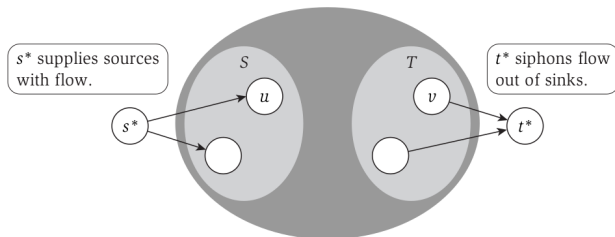
*If there is a feasible flow, then*

$$D = \sum_{v: d_v > 0} d_v = \sum_{v: d_v < 0} -d_v$$

## Not iff

Feasibility  $\implies \sum_{v \in V} d_v = 0$ , but  $\sum_{v \in V} d_v = 0 \not\Rightarrow$  feasibility.

# REDUCTION TO MAX-FLOW



## Reduction from $G$ (demands) to $G'$ (no demands)

- Super source  $s^*$ : Edges from  $s^*$  to all  $v \in S$  with  $d_v < 0$  with capacity  $-d_v$ .
- Super sink  $t^*$ : Edges from all  $v \in T$  with  $d_v > 0$  with capacity  $d_v$  to  $t^*$ .
- Maximum flow of  $D = \sum_{v:d_v>0 \in V} d_v = \sum_{v:d_v<0 \in V} -d_v$  in  $G'$  shows feasibility.



# ANOTHER FLOW NETWORK EXTENSION

## ADDING FLOW LOWER BOUND

### Adding Lower Bound

- For each edge  $e$ , define a lower bound  $\ell_e$ , where  $0 \leq \ell_e \leq c_e$ .

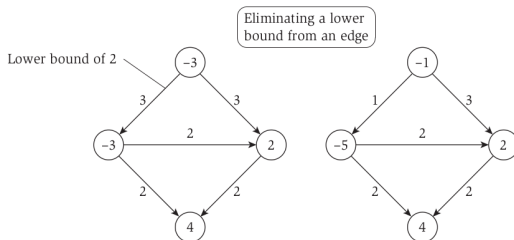
### Flow Conditions

- i Capacity: For each  $e \in E$ ,  $\ell_e \leq f(e) \leq c_e$ .
- ii Conservation: For each  $v \in V$ ,  $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$ .

### Goal

Feasibility: Does there exist a flow that satisfies the conditions?

# REDUCTION TO ONLY DEMAND



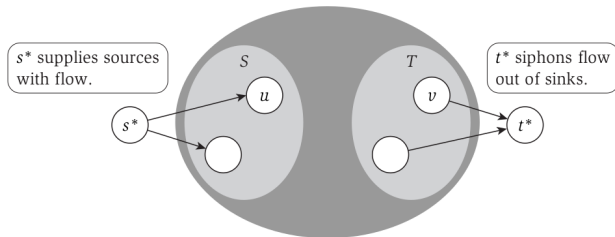
## Step 1: Reduction from $G$ (demand + LB) to $G'$ (demand)

- Consider an  $f_0$  that sets all edge flows to  $\ell_e$ :

$$L_v = f_0^{\text{in}}(v) - f_0^{\text{out}}(v).$$

- if  $L_v = d_v$ : Condition is satisfied.
  - if  $L_v \neq d_v$ : Imbalance.
- For  $G'$ :
  - Each edge  $e$ ,  $c'_e = c_e - \ell_e$  and  $\ell_e = 0$ .
  - Each node  $v$ ,  $d'_v = d_v - L_v$ .

# REDUCTION TO ONLY DEMAND



## Step 2: Reduction from $G'$ (demand) to $G''$ (no demand)

- Super source  $s^*$ : Edges from  $s^*$  to all  $v \in S$  with  $d_v < 0$  with capacity  $-d_v$ .
- Super sink  $t^*$ : Edges from all  $v \in T$  with  $d_v > 0$  with capacity  $d_v$  to  $t^*$ .
- Maximum flow of  $D = \sum_{v:d_v>0 \in V} d_v = \sum_{v:d_v<0 \in V} -d_v$  in  $G'$  shows feasibility.

# SURVEY DESIGN

# SURVEY DESIGN

## Problem

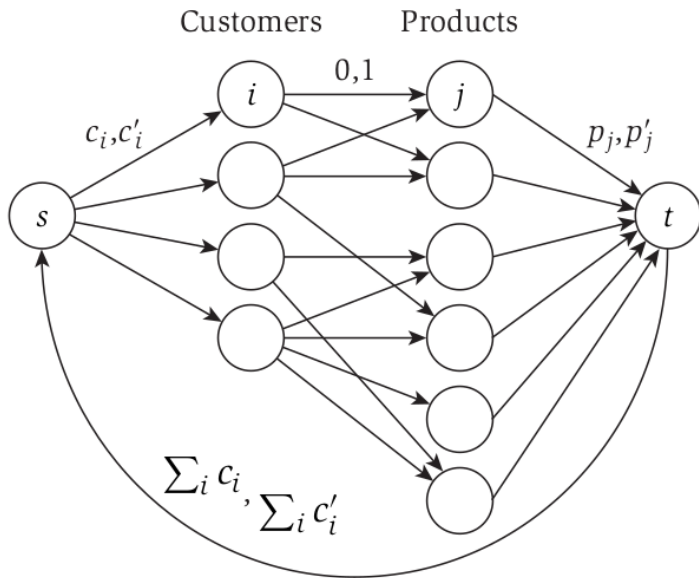
- Study of consumer preferences.
- A company, with  $k$  products, has a database of  $n$  customer purchase histories.
- Goal: Define a product specific survey.



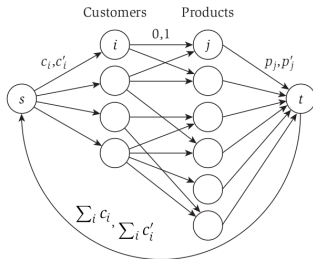
## Survey Rules

- Each customer receives a survey based on their purchases.
- Customer  $i$  will be asked about at least  $c_i$  and at most  $c'_i$  products.
- To be useful, each product must appear in at least  $p_i$  and at most  $p'_i$  surveys.

# ALGORITHM DESIGN



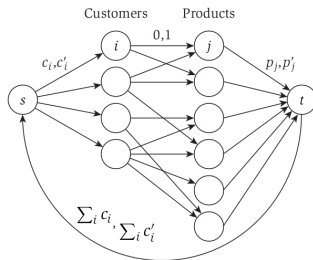
# ALGORITHM DESIGN



## Reduction

- Bipartite Graph: Customers to products with min of 0 and max of 1.
- Add  $s$  with edges to customer  $i$  with min of  $c_i$  and max of  $c_i'$ .
- Add  $t$  with edges from product  $j$  with min  $p_j$  and max of  $p_j'$ .
- Edge  $(t, s)$  with min  $\sum_i c_i$  and max  $\sum_i c_i'$ .
- All nodes have a demand of 0.

# ALGORITHM DESIGN



## Solution

- Feasibility means it is possible to meet the constraints.
- Edge  $(i, j)$  carries flow if customer  $i$  asked about product  $j$ .
- Flow  $(t, s)$  overall # of questions.
- Flow  $(s, i)$  # of products evaluated by customer  $i$ .
- Flow  $(j, t)$  # of customers asked about product  $j$ .



# AIRLINE SCHEDULING

# AIRLINE SCHEDULING

Flights: (2 airplanes)

- ① Boston (6 am) – Washington DC (7 am)
- ② Philadelphia (7 am) – Pittsburgh (8 am)
- ③ Washington DC (8 am) – Los Angeles (11 am)
- ④ Philadelphia (11 am) – San Francisco (2 pm)
- ⑤ San Francisco (2:15 pm) – Seattle (3:15 pm)
- ⑥ Las Vegas (5 pm) – Seattle (6 pm)

## Simple Version

- Scheduling a fleet of  $k$  airplanes.
- $m$  flight segments, for segment  $i$ :
  - Origin and departure time.
  - Destination and arrival time.

# AIRLINE SCHEDULING

Flights: (2 airplanes)

- ① Boston (6 am) – Washington DC (7 am)
- ② Philadelphia (7 am) – Pittsburgh (8 am)
- ③ Washington DC (8 am) – Los Angeles (11 am)
- ④ Philadelphia (11 am) – San Francisco (2 pm)
- ⑤ San Francisco (2:15 pm) – Seattle (3:15 pm)
- ⑥ Las Vegas (5 pm) – Seattle (6 pm)

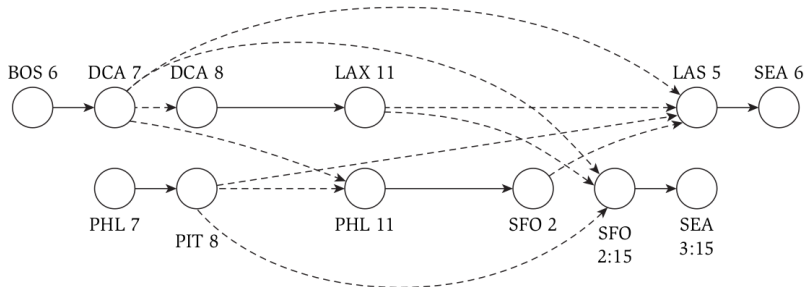
## Rules

The same plane can be used for flight  $i$  and  $j$  if:

- $i$  destination is the same as  $j$  origin and there is enough time for maintenance between  $i$  arrival and  $j$  departure;
- Or, there is enough time for maintenance and to fly from  $i$  destination to  $j$  origin.

How might you represent this as a graph?

# ALGORITHM DESIGN



$k = 2$  planes

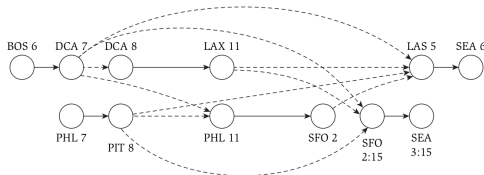
**Exercise: Reduce to a flow network**

Hint: Use lower bounds and demand.

- TH17: Are  $s$ - $t$  new nodes?
- TH18: What is the max capacity of the edges from  $G$ ?

# ALGORITHM DESIGN

$k = 2$  planes

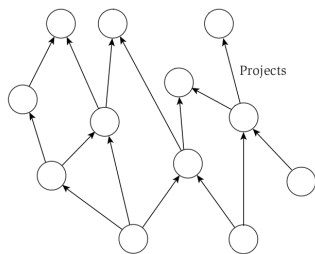


## Reduction

- Units of flow correspond to airplanes.
- Each edge of a flight has capacity  $(1, 1)$ .
- Each edge between flights has capacity of  $(0, 1)$ .
- Add node  $s$  with edges to all origins with capacity of  $(0, 1)$ .
- Add node  $t$  with edges from all destinations with cap  $(0, 1)$ .
- Edge  $(s, t)$  with a min of 0 and a max of  $k$ .
- Demand:  $d_s = -k, d_t = k, d_v = 0 \forall v \in V \setminus \{s, t\}$ .

# PROJECT SELECTION

# PROJECT SELECTION

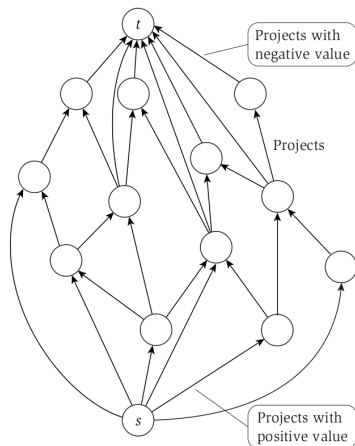


Use Min-Cut to solve this problem.

## Problem

- Set of projects:  $P$ .
- Each  $i \in P$ : profit  $p_i$  (which can be negative).
- Directed graph  $G$  encoding precedence constraints.
- Feasible set of projects  $A$ :  $\text{PROFIT}(A) = \sum_{i \in A} p_i$ .
- Goal: Find  $A^*$  that maximizes profit.

# ALGORITHM DESIGN

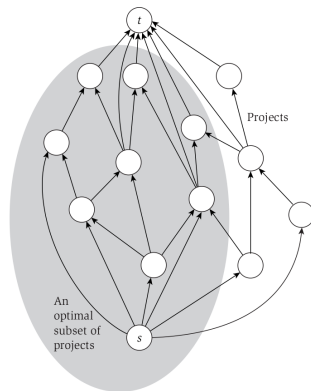


## Reduction

- Use Min-Cut
- Add  $s$  with edge to every project  $i$  with  $p_i > 0$  and capacity  $p_i$ .
- Add  $t$  with edge from every project  $i$  with  $p_i < 0$  and capacity  $-p_i$ .
- Max-flow is  $\leq C = \sum_{i \in P: p_i > 0} p_i$ .
- For edges of  $G$ , capacity is  $\infty$  (or  $C + 1$ ).



# ALGORITHM ANALYSIS



## Observation 4

If  $c(A', B') \leq C$ , then  $A = A' \setminus \{s\}$  satisfies precedence as edges of  $G$  have capacity  $> C$ .

## Lemma 12

Let  $(A', B')$  be a cut satisfies precedence; then  
 $c(A', B') = C - \sum_{i \in A} p_i$ .

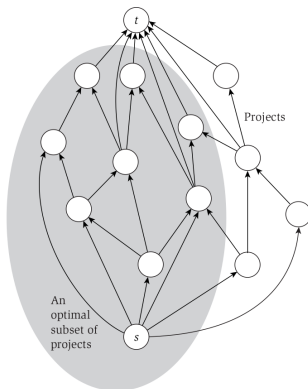
## Proof.

Consider the different edges:

- $(i, t)$ :  $-p_i$  for  $i \in A$ .
- $(s, i)$ :  $p_i$  for  $i \notin A$ .

$$c(A', B') = \sum_{i \in A: p_i < 0} -p_i + C - \sum_{i \in A: p_i > 0} p_i = C - \sum_{i \in A} p_i \quad \square$$

# ALGORITHM ANALYSIS



## Theorem 12

If  $(A', B')$  is a min-cut in  $G'$ , then  $A = A' \setminus \{s\}$  is an optimal solution.

## Proof.

- Obs:  $c(A', B') = C - \sum_{i \in A} p_i$  means feasible.  

$$c(A', B') = C - \text{PROFIT}(A)$$

$$\iff \text{PROFIT}(A) = C - c(A', B')$$
- Given that  $c(A', B')$  is a minimum, profit is maximized as  $C$  is a constant.



# BASEBALL ELIMINATION

# BASEBALL ELIMINATION

	Wins	Games Left
New York	92	NY Yankees vs Toronto
Toronto	91	Toronto vs Baltimore
Baltimore	91	Baltimore vs Boston
Boston	90	Boston vs Toronto Boston vs NY Yankees

## Why is Boston eliminated?

Case analysis:

- Boston must win its 2 remaining games.
- New York must lose its 2 remaining games.
- This leaves Toronto vs Baltimore: So one of Toronto or Baltimore will end with 93 wins.

# BASEBALL ELIMINATION

	Wins	Games Left
New York	92	NY Yankees vs Toronto
Toronto	91	Toronto vs Baltimore
Baltimore	91	Baltimore vs Boston
Boston	90	Boston vs Toronto
		NY Yankees vs Baltimore

## Why is Boston eliminated?

Analytical approach:

- Boston can finish with  $\leq 92$  wins.
- Currently, other 3 teams have 274 combined wins with 3 remaining games between them:
  - Overall, at the end, there will be 277 combined wins between the other 3 teams.
  - Average of  $92 \frac{1}{3}$  wins which implies that one team will have at least  $92 \frac{1}{3} \Rightarrow 93$  wins.

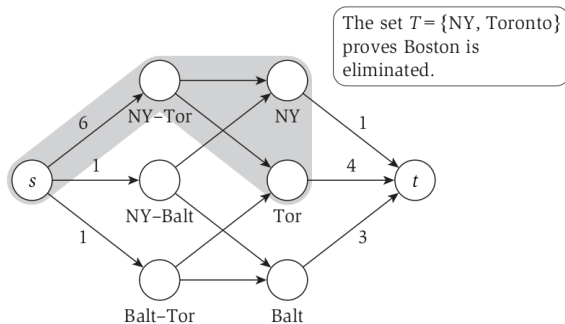
# BASEBALL ELIMINATION



## Problem

- A set  $S$  of teams.
- For each team  $x \in S$ :  $w_x$  is the # of wins.
- For each pair  $x, y \in S$ :  $g_{xy}$  is # of games left btw  $x$  and  $y$ .
- Goal: Decide if team  $z$  has been eliminated.

# ALGORITHM DESIGN

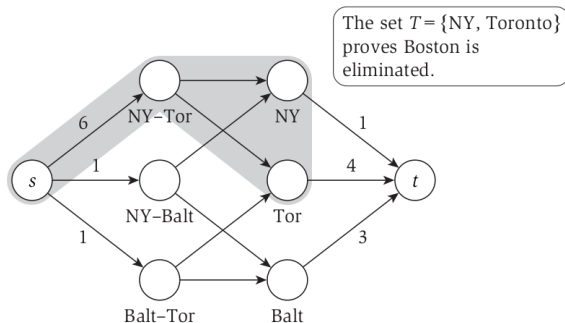


Let  $m$  be the max # of wins for  $z$ ,  
 $S' = S \setminus \{z\}$ , and  
 $g^* = \sum_{x,y \in S'} g_{xy}$ .

## Reduction

- Nodes:
  - Source  $s$ , sink  $t$ .
  - $v_x$  for each  $x \in S'$ .
  - $u_{xy}$  for each pair  $x, y \in S'$ .

# ALGORITHM DESIGN



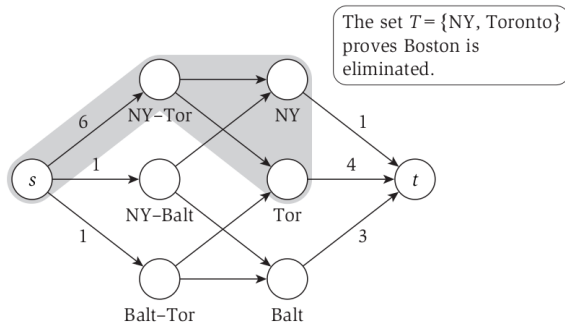
Let  $m$  be the max # of wins for  $z$ ,  
 $S' = S \setminus \{z\}$ , and  
 $g^* = \sum_{x,y \in S'} g_{xy}$ .

## Reduction

- Edges:
  - For each  $v_x$ :  $(v_x, t)$  with capacity  $m - w_x$ .
  - For each  $u_{xy}$ :
    - $(s, u_{xy})$  with capacity  $g_{xy}$ .
    - $(u_{xy}, v_x)$  and  $(u_{xy}, v_y)$  with capacity  $\infty$  (or  $g_{xy}$ ).



# ALGORITHM DESIGN



Let  $m$  be the max # of wins for  $z$ ,  
 $S' = S \setminus \{z\}$ , and  
 $g^* = \sum_{x,y \in S'} g_{xy}$ .

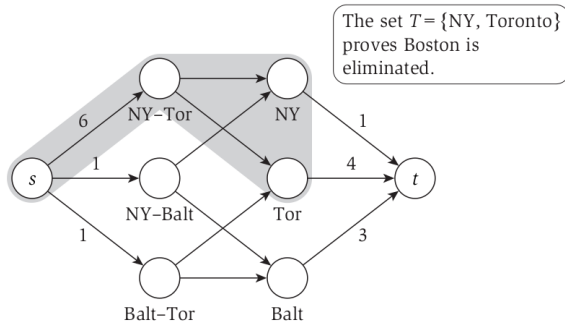
## Solution

- $v(f) = g^*$ :  $z$  is not eliminated.

$$v(f) = g^* = f^{\text{in}}(t) \leq \sum_{x \in S'} (m - w_x) = m|S'| - \sum_{x \in S'} w_x$$

$$\iff \sum_{x,y \in S'} g_{xy} \leq m|S'| - \sum_{x \in S'} w_x$$

# ALGORITHM DESIGN



Let  $m$  be the max # of wins for  $z$ ,  
 $S' = S \setminus \{z\}$ , and  
 $g^* = \sum_{x,y \in S'} g_{xy}$ .

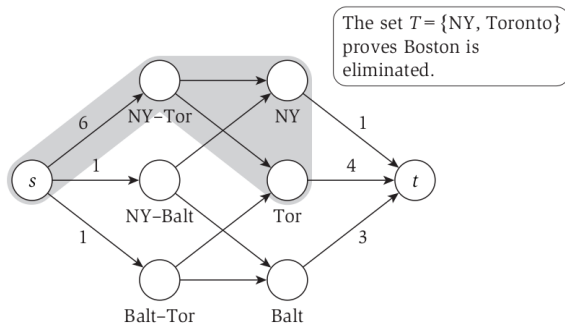
## Solution

- $v(f) = g^*$ :  $z$  is not eliminated.

$$v(f) = g^* = f^{\text{in}}(t) \leq \sum_{x \in S'} (m - w_x) = m|S'| - \sum_{x \in S'} w_x$$

$$\iff m|S'| \geq \sum_{x,y \in S'} g_{xy} + \sum_{x \in S'} w_x$$

# ALGORITHM DESIGN



Let  $m$  be the max # of wins for  $z$ ,  
 $S' = S \setminus \{z\}$ , and  
 $g^* = \sum_{x,y \in S'} g_{xy}$ .

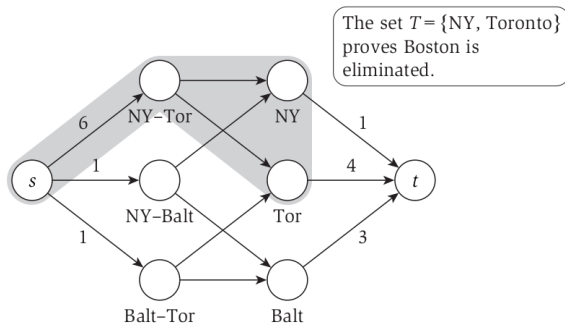
## Solution

- $v(f) = g^*$ :  $z$  is not eliminated.

$$v(f) = g^* = f^{\text{in}}(t) \leq \sum_{x \in S'} (m - w_x) = m|S'| - \sum_{x \in S'} w_x$$

$$\iff m \geq \left( \sum_{x,y \in S'} g_{xy} + \sum_{x \in S'} w_x \right) / |S'|$$

# ALGORITHM DESIGN

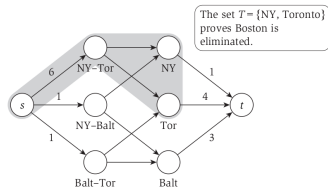


Let  $m$  be the max # of wins for  $z$ ,  
 $S' = S \setminus \{z\}$ , and  
 $g^* = \sum_{x,y \in S'} g_{xy}$ .

## Solution

- $v(f) = g^*$ :  $z$  is not eliminated.
- $v(f) < g^*$ :  $z$  is eliminated.

# SOLUTION CHARACTERIZATION



Let  $m$  be the max # of wins for  $z$ ,  $S' = S \setminus \{z\}$ , and

$$g^* = \sum_{x,y \in S'} g_{xy}.$$

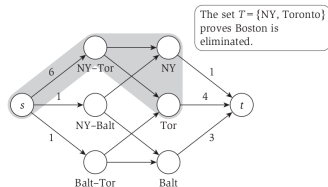
## Theorem 13

Suppose  $z$  has been eliminated. Then, there is a set of items  $T \subseteq S'$  such that:  $m|T| < \sum_{x,y \in T} g_{xy} + \sum_{x \in T} w_x$

## Proof.

- Let  $(A, B)$  be a min-cut with  $c(A, B) = g' \leq \min\{\sum_{x,y \in S'} g_{xy}, \sum_{x \in S'} m - w_x\}$ .
- Consider a  $u_{xy} \in A, x \in T$ , and  $y \notin T$  (WLOG).
  - Contradiction:  $c(u_{xy}, y) = \infty$ .

# SOLUTION CHARACTERIZATION



Let  $m$  be the max # of wins for  $z$ ,  $S' = S \setminus \{z\}$ , and

$$g^* = \sum_{x,y \in S'} g_{xy}.$$

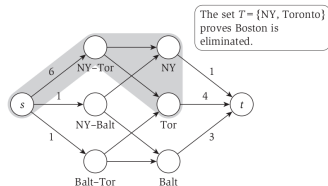
## Theorem 13

Suppose  $z$  has been eliminated. Then, there is a set of items  $T \subseteq S'$  such that:  $m|T| < \sum_{x,y \in T} g_{xy} + \sum_{x \in T} w_x$

## Proof.

- Let  $(A, B)$  be a min-cut with  $c(A, B) = g' \leq \min\{\sum_{x,y \in S'} g_{xy}, \sum_{x \in S'} m - w_x\}$ .
- Consider a  $u_{xy} \notin A$ , and  $x, y \in T$ .
  - Contradiction:  $c(A \cup \{u_{xy}\}, B \setminus \{u_{xy}\}) = c(A, B) - g_{xy}$ .

# SOLUTION CHARACTERIZATION



Let  $m$  be the max # of wins for  $z$ ,  $S' = S \setminus \{z\}$ , and  $g^* = \sum_{x,y \in S'} g_{xy}$ .

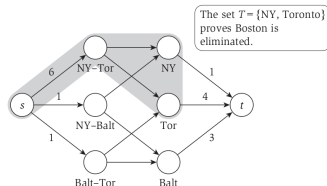
## Theorem 13

Suppose  $z$  has been eliminated. Then, there is a set of items  $T \subseteq S'$  such that:  $m|T| < \sum_{x,y \in T} g_{xy} + \sum_{x \in T} w_x$

## Proof.

- Let  $(A, B)$  be a min-cut with  $c(A, B) = g' \leq \min\{\sum_{x,y \in S'} g_{xy}, \sum_{x \in S'} m - w_x\}$ .
- $c(A, B) = g' = m|T| - \sum_{x \in T} w_x + \sum_{x,y \notin T} g_{xy}$

# SOLUTION CHARACTERIZATION



Let  $m$  be the max # of wins for  $z$ ,  $S' = S \setminus \{z\}$ , and  $g^* = \sum_{x,y \in S'} g_{xy}$ .

## Theorem 13

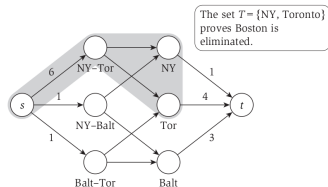
Suppose  $z$  has been eliminated. Then, there is a set of items  $T \subseteq S'$  such that:  $m|T| < \sum_{x,y \in T} g_{xy} + \sum_{x \in T} w_x$

## Proof.

- Let  $(A, B)$  be a min-cut with  $c(A, B) = g' \leq \min\{\sum_{x,y \in S'} g_{xy}, \sum_{x \in S'} m - w_x\}$ .
- $c(A, B) = g' = m|T| - \sum_{x \in T} w_x + g^* - \sum_{x,y \in T} g_{xy}$   
 $\iff 0 > m|T| - \sum_{x \in T} w_x - \sum_{x,y \in T} g_{xy}$  as  $g' < g^*$



# SOLUTION CHARACTERIZATION



Let  $m$  be the max # of wins for  $z$ ,  $S' = S \setminus \{z\}$ , and

$$g^* = \sum_{x,y \in S'} g_{xy}.$$

## Theorem 13

Suppose  $z$  has been eliminated. Then, there is a set of items  $T \subseteq S'$  such that:  $m|T| < \sum_{x,y \in T} g_{xy} + \sum_{x \in T} w_x$

## Proof.

- Let  $(A, B)$  be a min-cut with  $c(A, B) = g' \leq \min\{\sum_{x,y \in S'} g_{xy}, \sum_{x \in S'} m - w_x\}$ .
- $c(A, B) = g' = m|T| - \sum_{x \in T} w_x + g^* - \sum_{x,y \in T} g_{xy}$   
 $\iff m|T| < \sum_{x \in T} w_x + \sum_{x,y \in T} g_{xy}$

