CS 577 - Network Flow

Marc Renault

Department of Computer Sciences
University of Wisconsin – Madison

Fall 2024

TopHat Section 001 Join Code: 728978
TopHat Section 002 Join Code: 412331

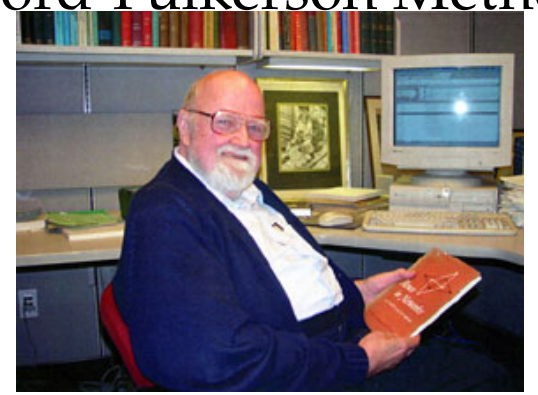
Network Flow

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.

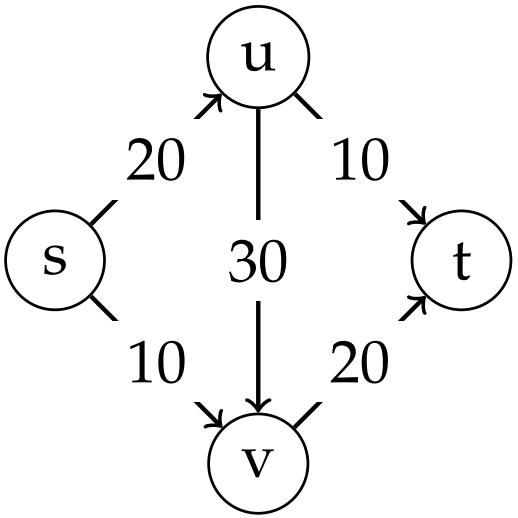
Ford-Fulkerson Method (1956)



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 $V \setminus \{s, t\}$.

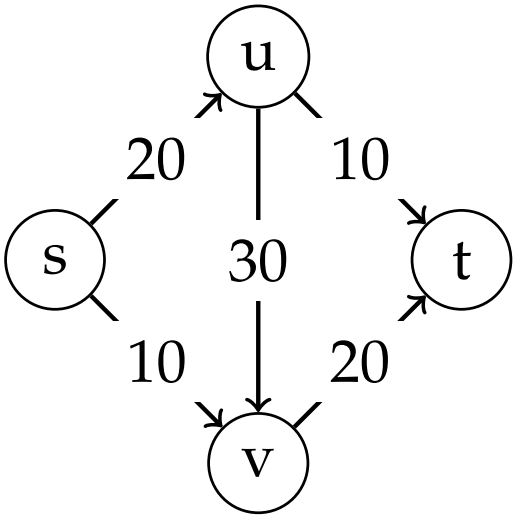
Defining Flow

- Flow starts at s and exits at t.
- Flow function: $f : E \to R^+; f(e)$ is the flow across edge e.
- Flow Conditions:
  - i Capacity: For each $e \in E$, $0 \leq f(e) \leq c_e$.
  - ii Conservation: For each $v \in V \setminus \{s, t\}$,

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

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

Maximum-Flow Problem



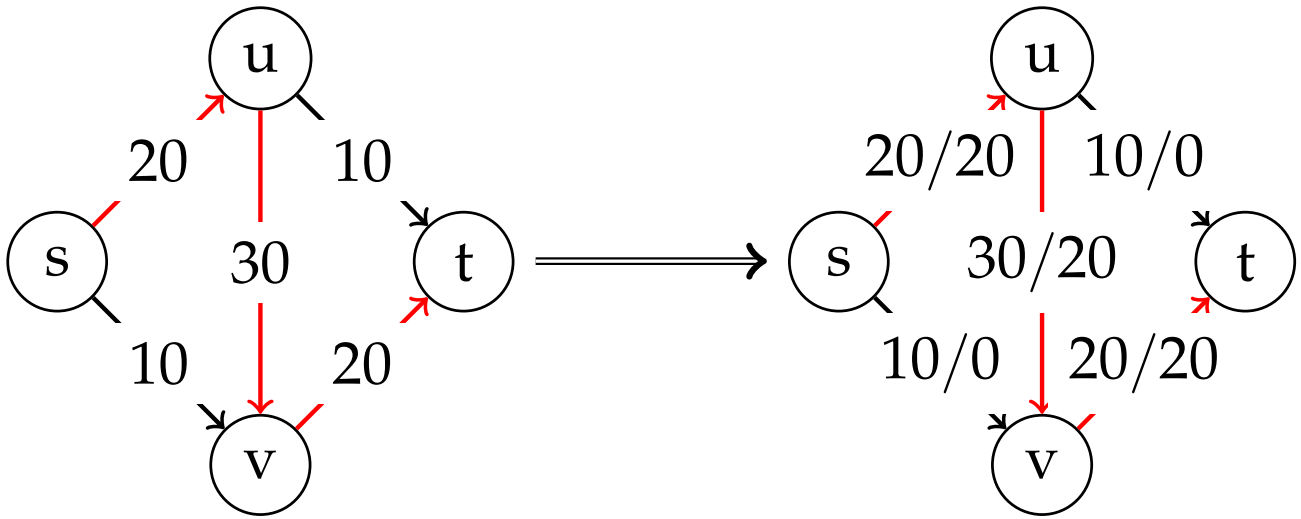
Max-Flow

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.

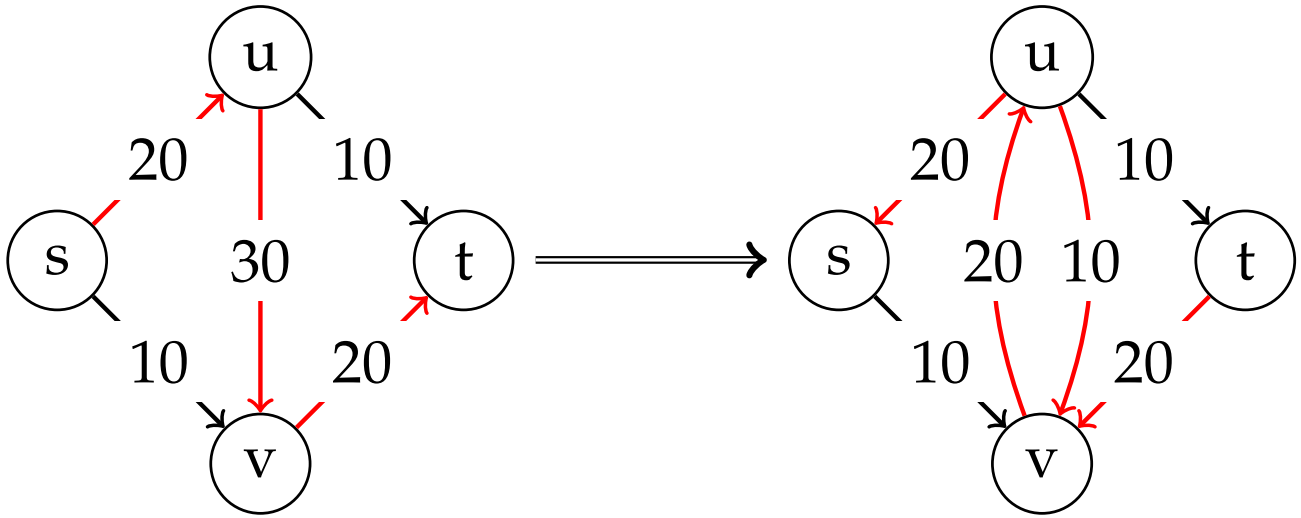
Designing the Approach



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

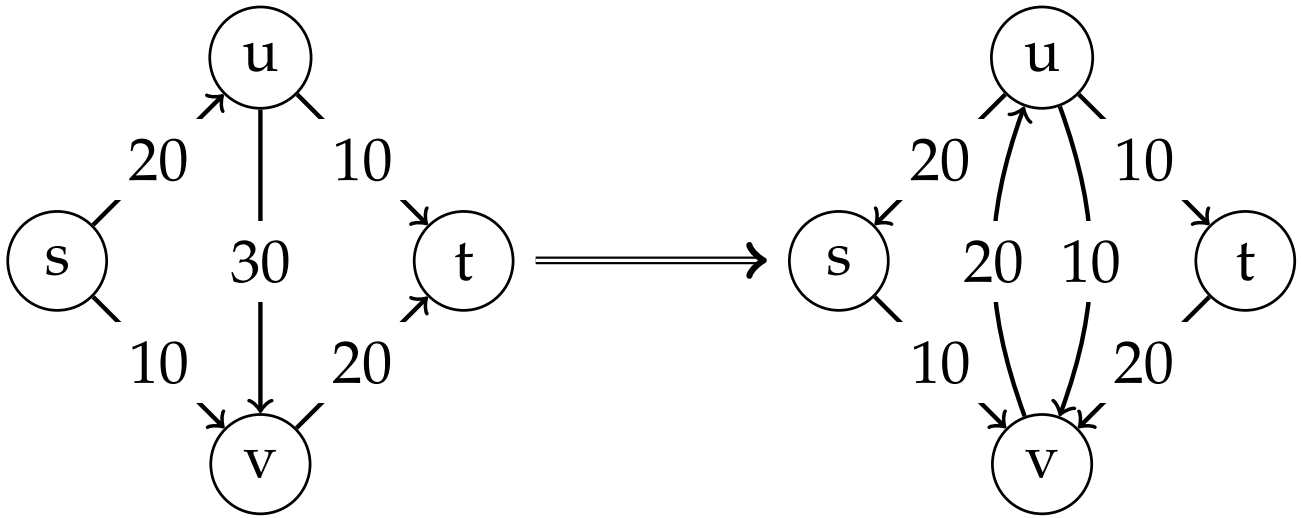


Residual Graph

Given a flow network G and a flow f on G, we define 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



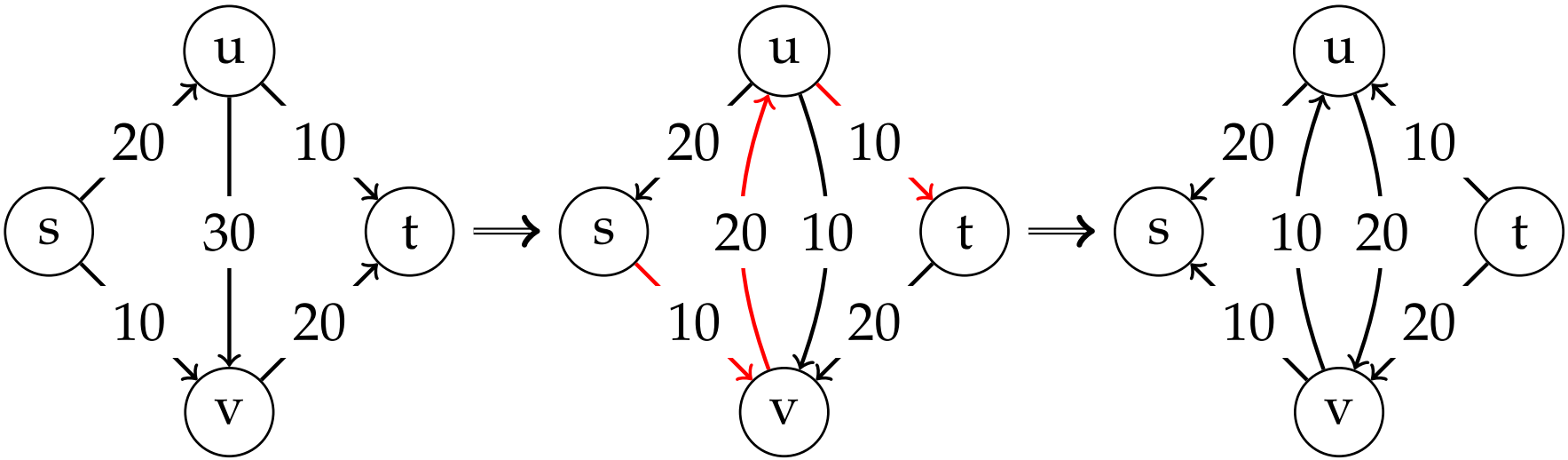
Augmenting Path

- A simple directed path from s to t.
- 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.

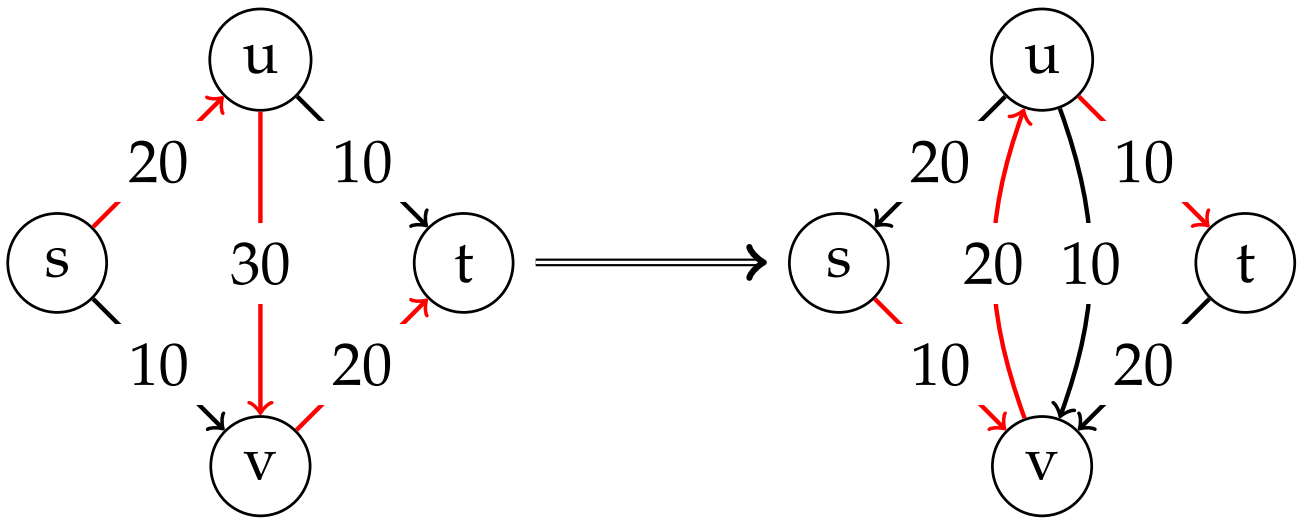
Augmenting Path



Increasing the Flow along Augmenting Path

- Push $\textsc{bottleneck}(P, G_f) = q$ 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 $\textsc{bottleneck}(P, G_f)$ 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 $\textsc{bottleneck}(P, G_f)$ 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 $\textsc{bottleneck}(P, G_f)$ along P.

Lemma 1

$v(f') > v(f)$, where $v(f') = v(f) + \textsc{bottleneck}(P, G_f)$ 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 $\textsc{bottleneck}(P, G_f) = q$. 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 $\textsc{bottleneck}(P, G_f)$ 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 $\textsc{bottleneck}(P, G_f)$ 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 $\textsc{bottleneck}(P, G_f)$ along P.

Theorem 3

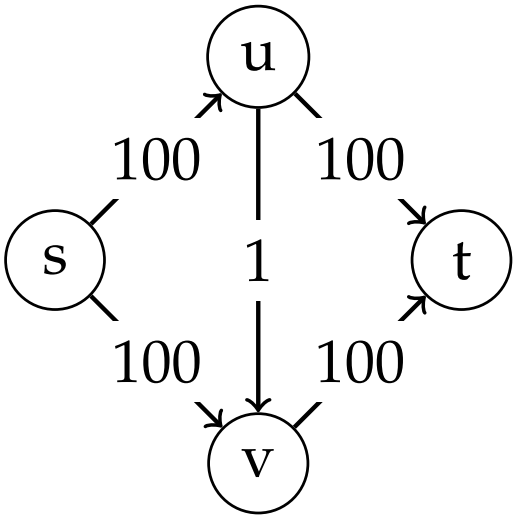
Suppose all capacities are integers. Then, runtime of $O(mC)$.

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 $\Delta := \max_i \left(2^i\right)$ 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 $\textsc{bottleneck}(P, G_f(\Delta))$ along P.
  - Set $\Delta := \Delta/2$.

Analyzing the Scaled Version

Scaled Version

- Initialize $f(e) = 0$ for all edges.
- Initialize $\Delta := \max_i \left(2^i\right)$ 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 $\textsc{bottleneck}(P, G_f(\Delta))$ 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 $\Delta := \max_i \left(2^i\right)$ 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 $\textsc{bottleneck}(P, G_f(\Delta))$ 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 $\Delta := \max_i \left(2^i\right)$ 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 $\textsc{bottleneck}(P, G_f(\Delta))$ 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,

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

Proof.

- By definition, $f^{\text{out}}(A) = f^{\text{in}}(B)$ and $f^{\text{in}}(A) = f^{\text{out}}(B)$.
- By definition,
$$\begin{aligned} v(f) &= f^{\text{out}}(s) \\ &= f^{\text{out}}(s) - f^{\text{in}}(s) \\ &= \sum_{v \in A} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right) \end{aligned}$$
- Last line follows since $\sum_{v \in A \setminus \{s\}} \left(f^{\text{out}}(v) - f^{\text{in}}(v)\right) = 0$.

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

□

Max-Flow and Min-Cut

Lemma 4

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

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

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.

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

$$\leq \sum_{e \text{ out of } A} c_e = c(A,B) \qquad \square$$

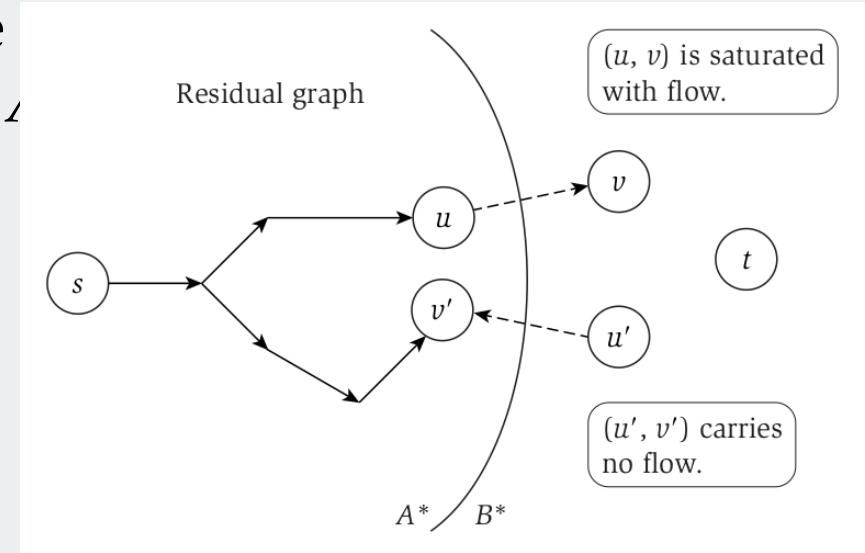
Max-Flow equals Min-Cut

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 … path in G_f. 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.

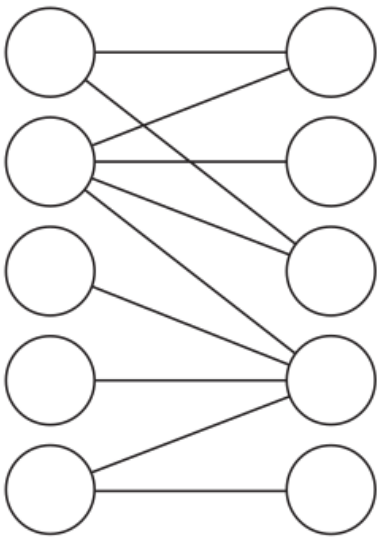
Proof.

- 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



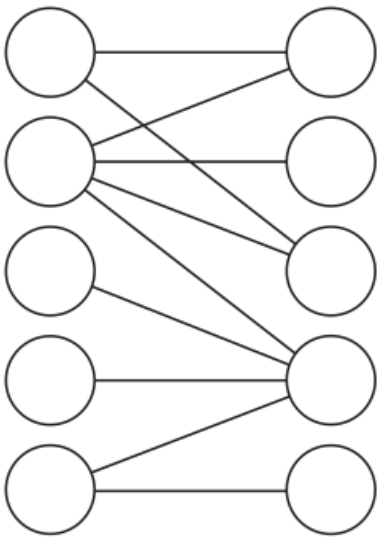
Definition

- 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



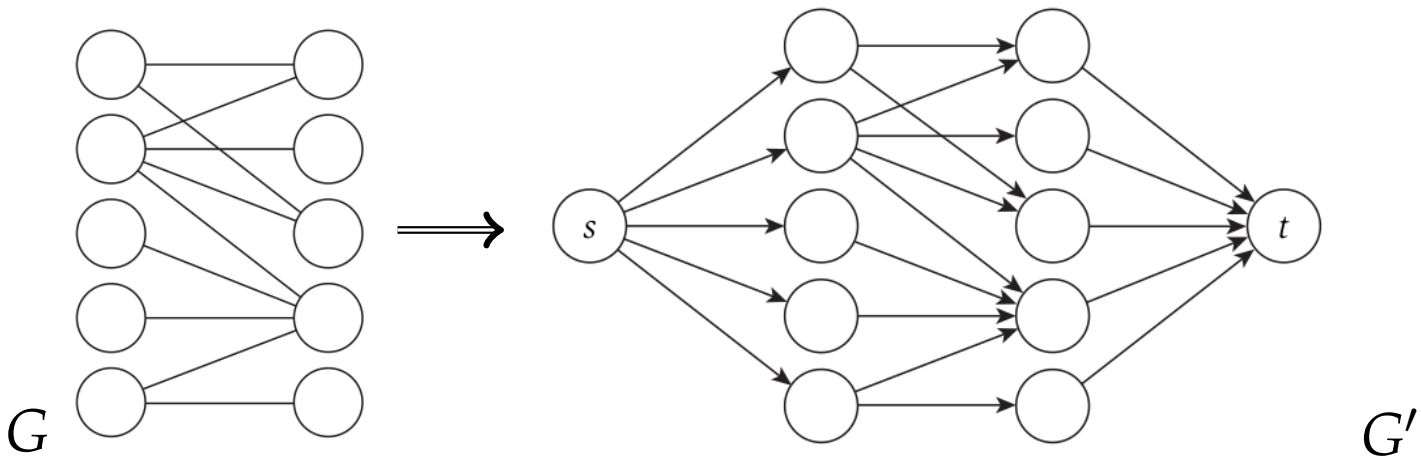
Definition

- 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

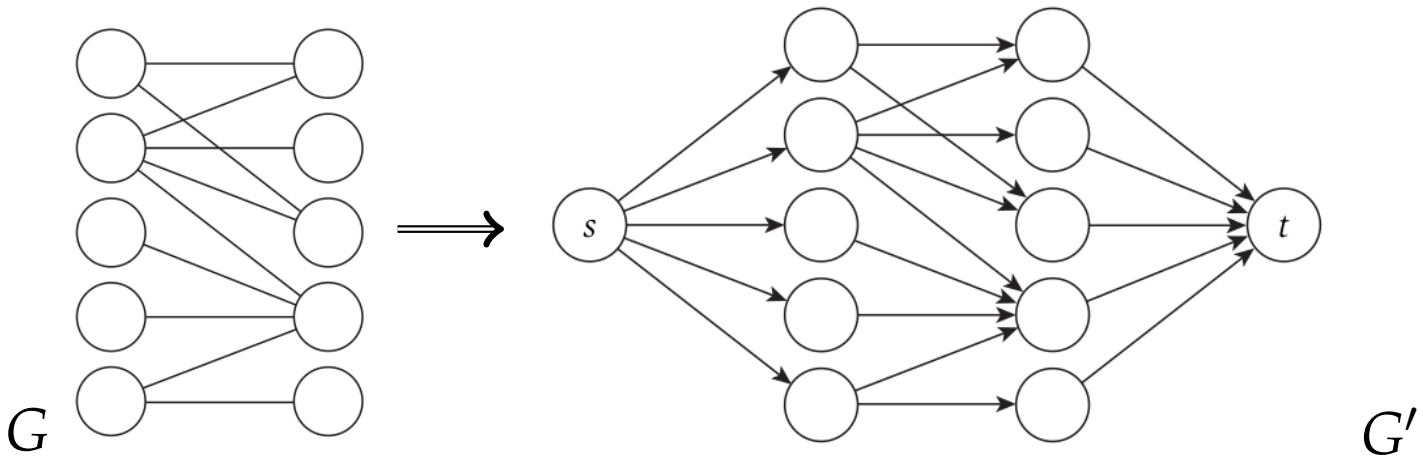
$|M^*|$ in G is equal to the max-flow of G', and the edges carrying the flow correspond to the edges in the maximum matching.

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 $C = n$.

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

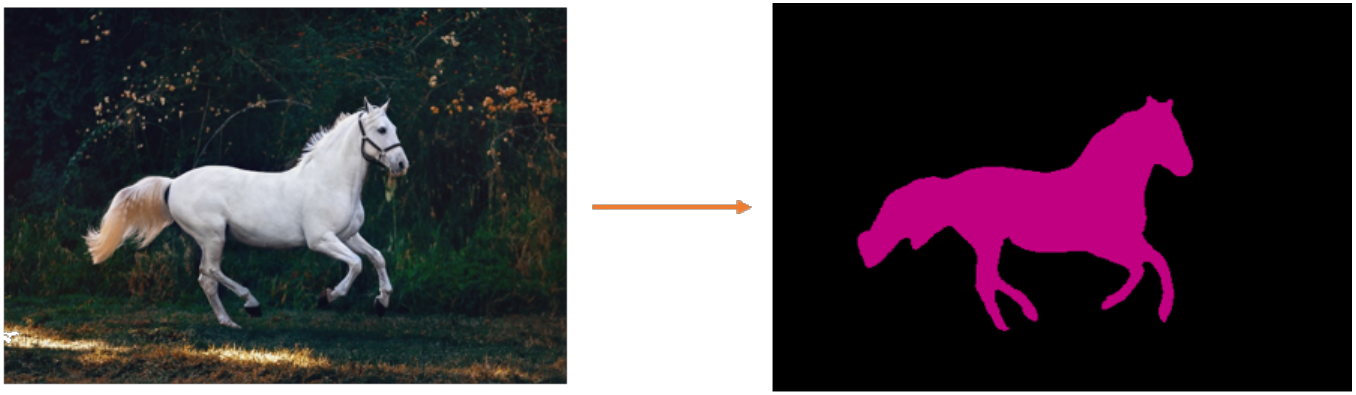
- Basic FF method: $O(mC) = O(mn)$.

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 $f(e) = 1$:
  1. Find a simple path P from s to t: set flow to 0 along P; continue DFS from s.
  2. 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

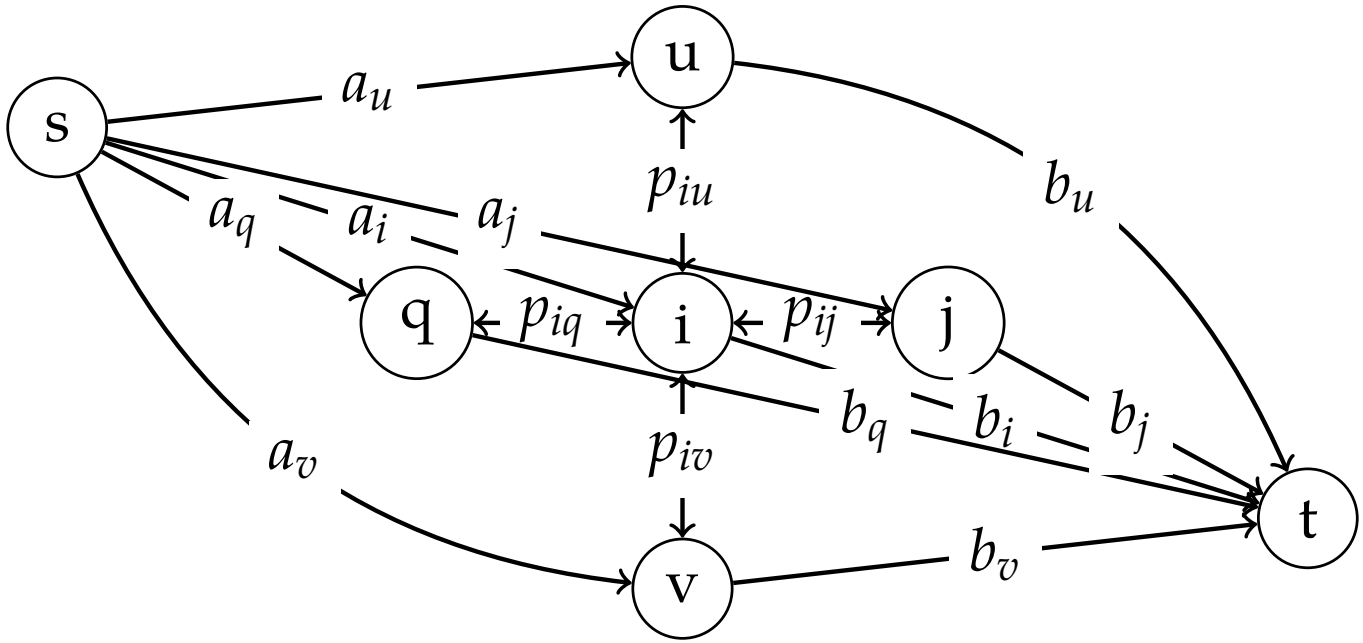
Let P be the set of pixels in an image.
For pixel i:

- $a_i > 0$ 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 $\sum_{i \in B} a_i + \sum_{j \in A} b_j + \sum_{i,j \in P : |A \cap \{i,j\}| = 1} p_{ij}$.

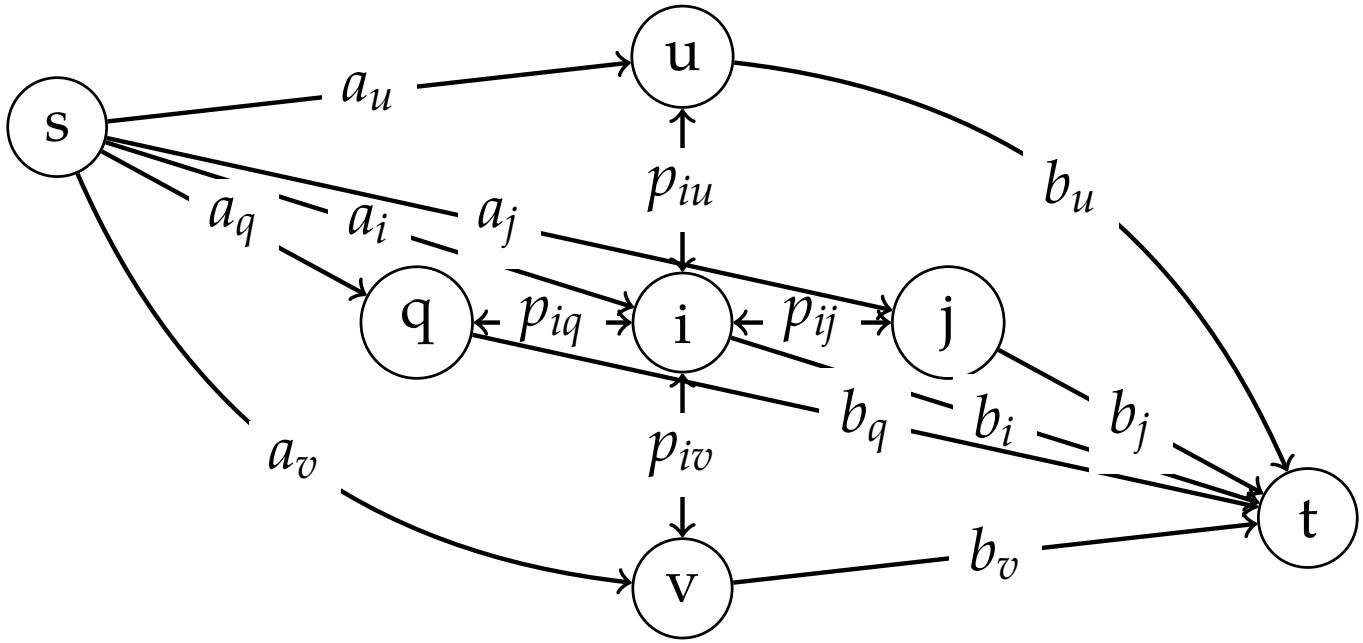
Algorithm Design



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

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

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?

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

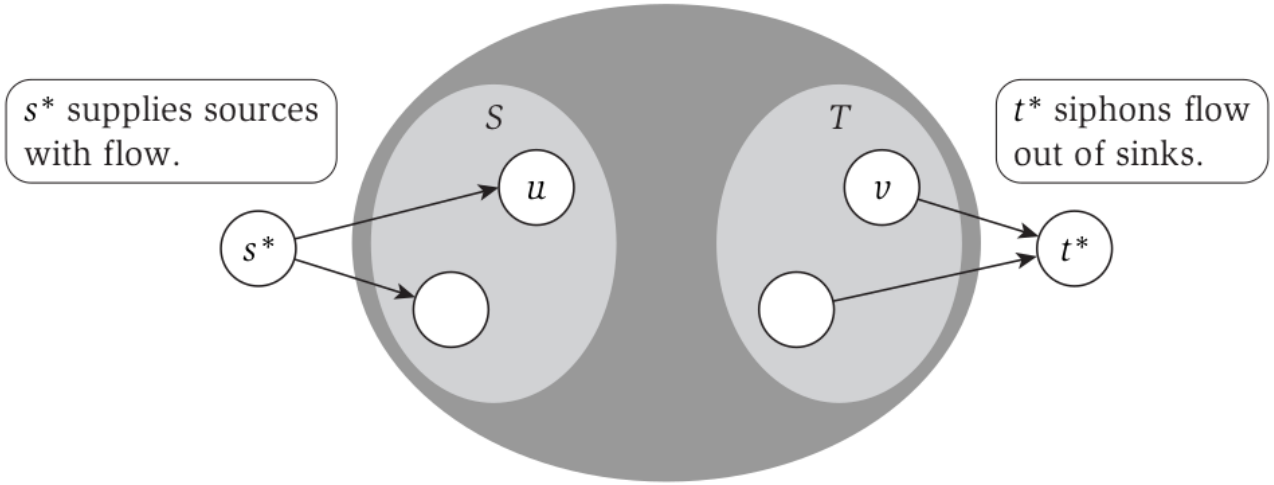
If there is a feasible flow, then

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

Not iff

Feasibility $\implies$ $\sum_{v \in V} d_v = 0$, but $\sum_{v \in V} d_v = 0 \not\implies$ 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 ℓ_e, where $0 \leq \ell_e \leq c_e$.

Flow Conditions

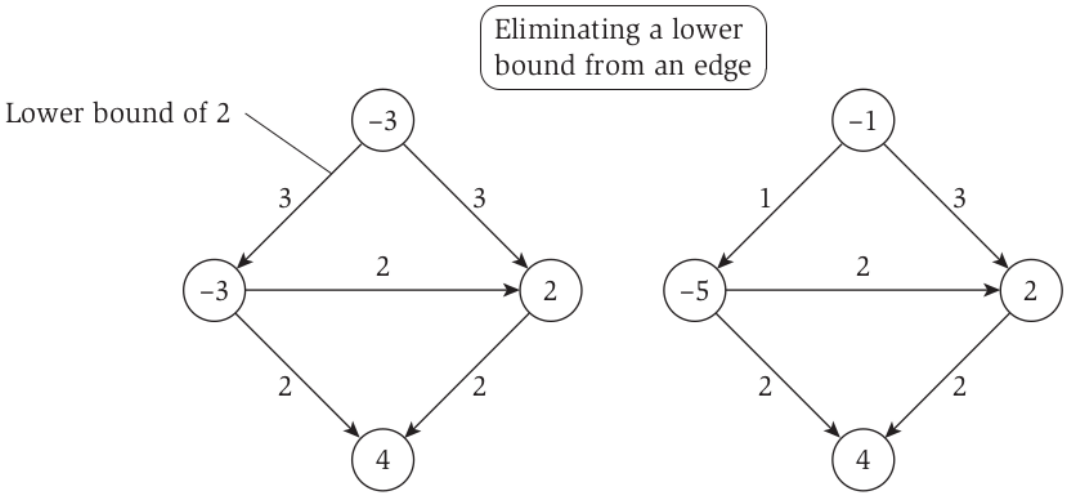
i. Capacity: For each $e \in E$, $\ell_e \leq f(e) \leq c_e$.

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?

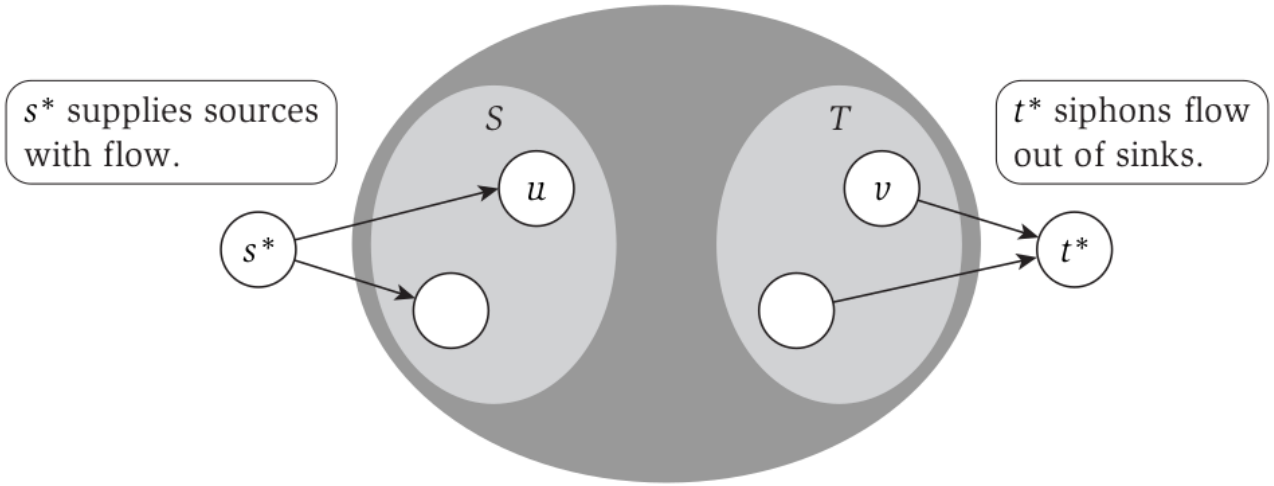
Reduction to Only Demand



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

- Consider an f_0 that sets all edge flows to ℓ_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

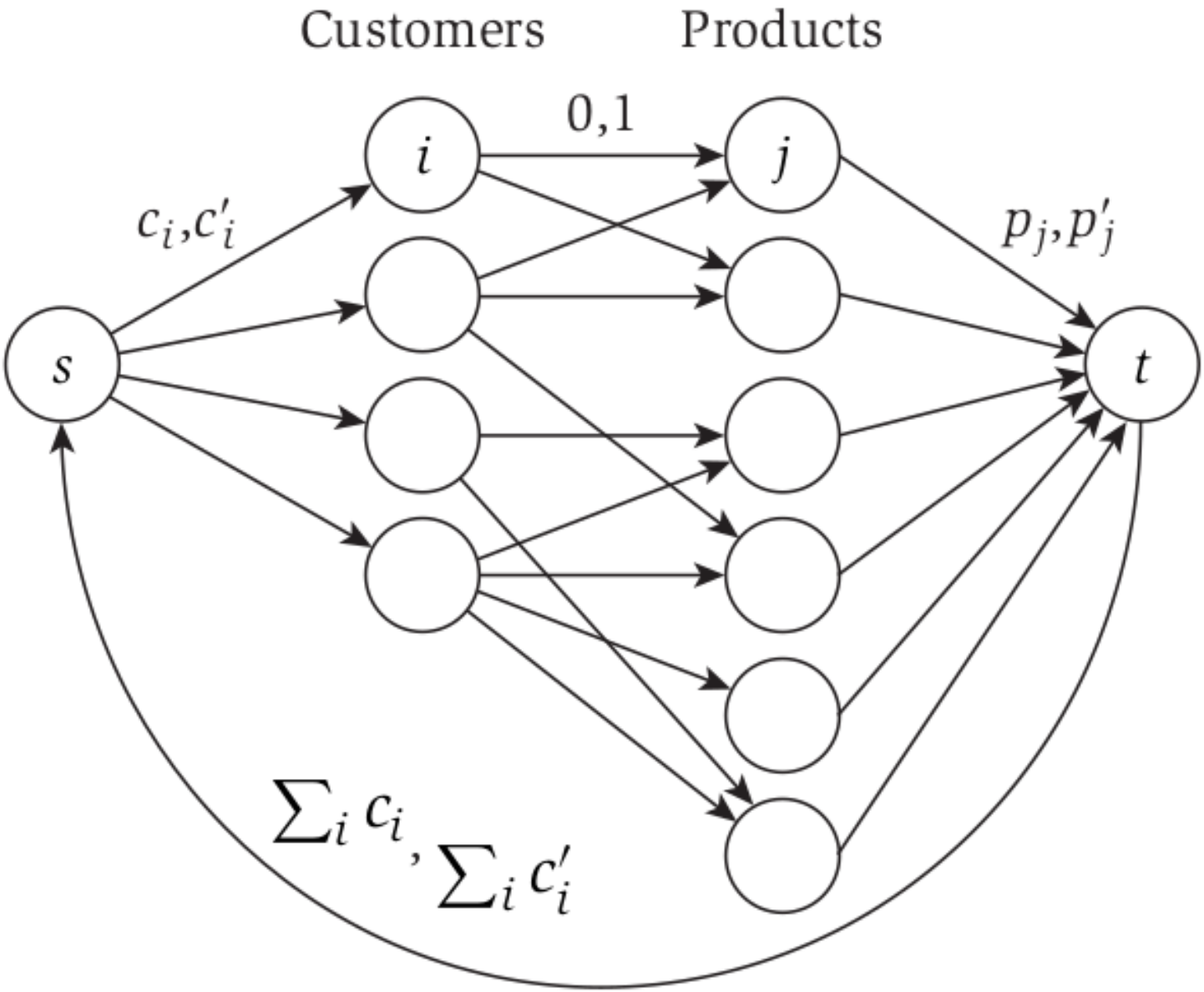
Problem

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

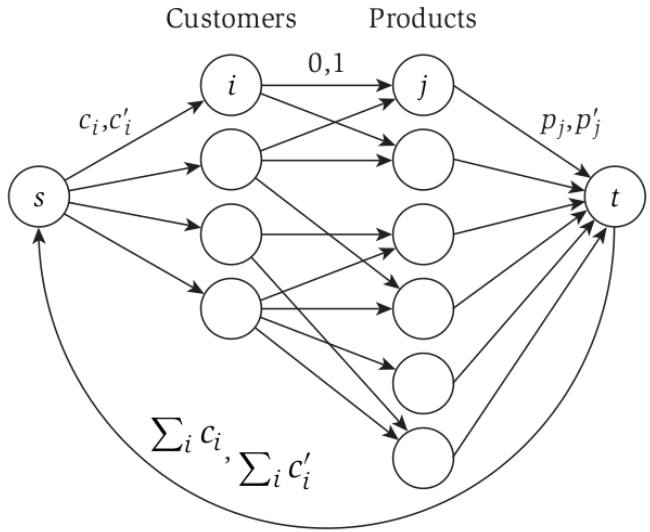
Survey Rules

- 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



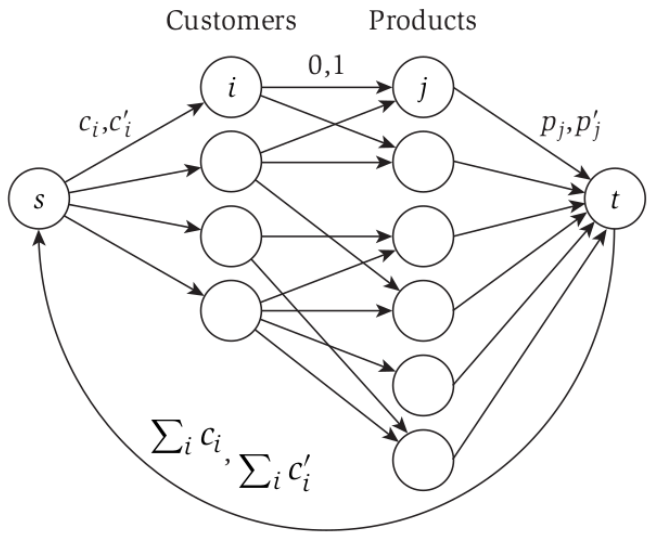
Algorithm Design



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)

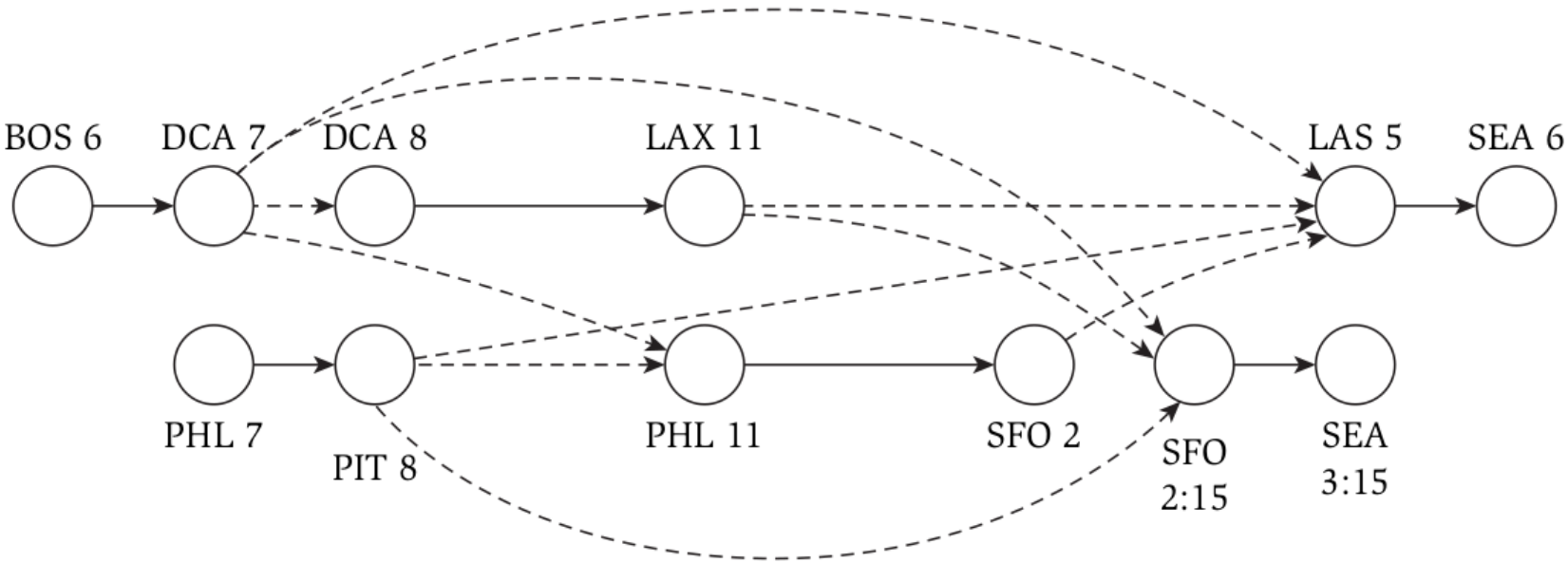
Rules

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

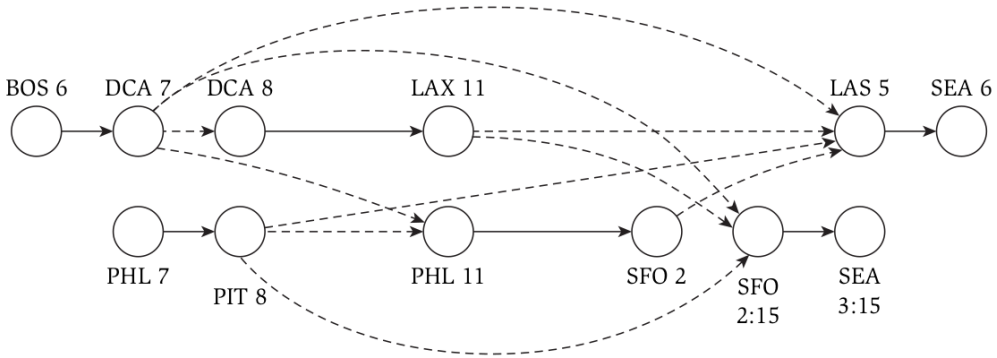
Exercise: Reduce to a flow network

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

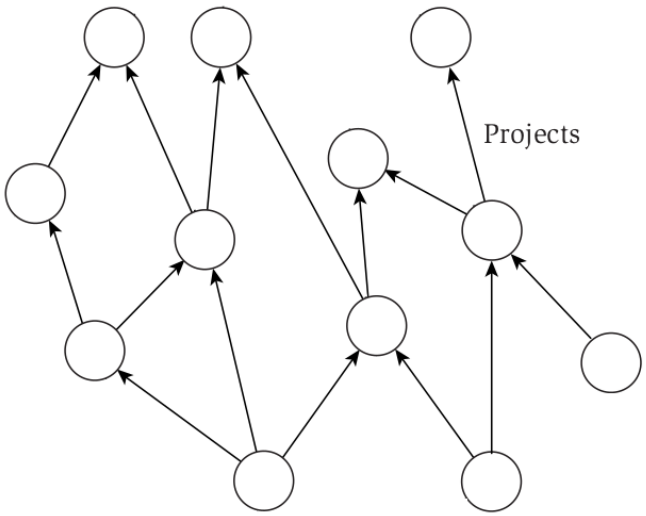


Reduction

- 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

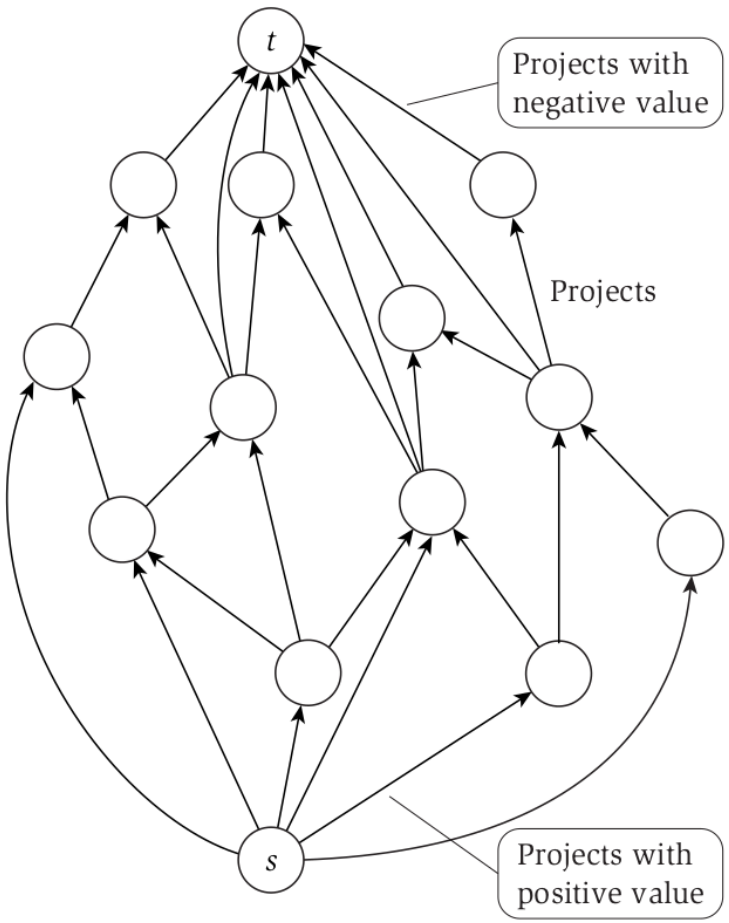


Use Min-Cut to solve this problem.

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: $\textsc{profit}(A) = \sum_{i \in A} p_i$.
- Goal: Find A^* that maximizes profit.

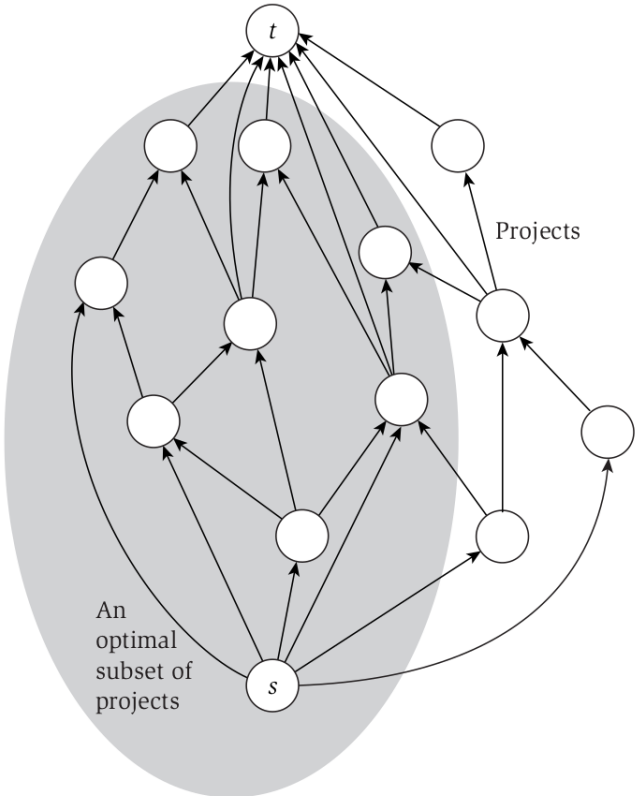
Algorithm Design



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 ∞ (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 $c(A', B') = C - \sum_{i \in A} p_i$.

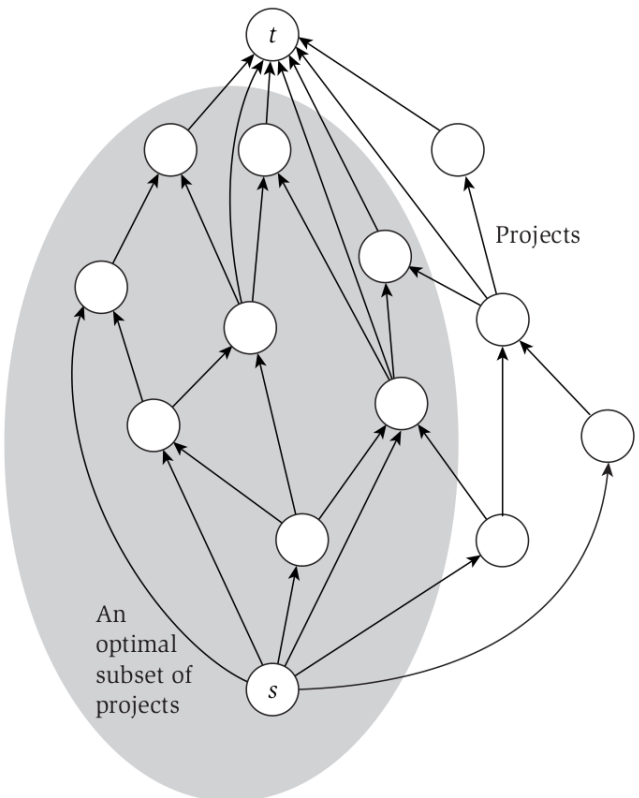
Proof.

Consider the different edges:

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

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

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.

$$c(A', B') = C - \textsc{profit}(A)$$

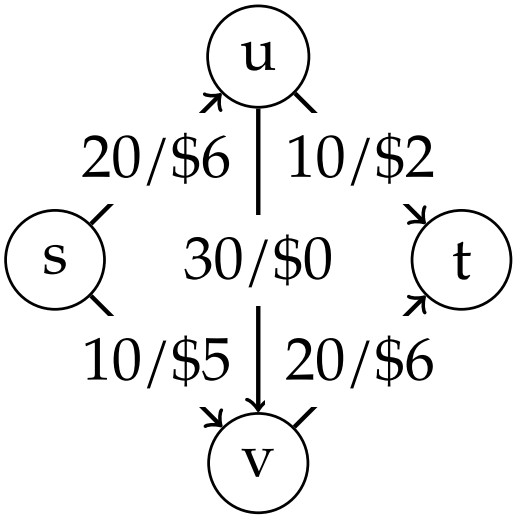
$$\iff \textsc{profit}(A) = C - c(A', B')$$

- Given that $c(A', B')$ is a minimum, profit is maximized as C is a constant.

□

Min-Cost Max Flow

Flow Network with Cost



Flow Network with Cost

- Directed graph $G = (V, E)$.
- Each edge e has $c_e \geq 0$ and a cost $\$_e \geq 0$.
  - $\$_e$ is the cost per unit of flow.

Defining Flow

- Flow starts at s and exits at t.
- Flow function: $f : E \to R^+; f(e)$ is the flow across edge e.
- Flow Conditions:
  1. Capacity: For each $e \in E, 0 \leq f(e) \leq c_e$.
  2. Conservation: For each $v \in V \setminus \{s, t\}$,
$$\sum_{e \text{ into } v} f(e) = f^{\text{in}}(v) = f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$$
- Flow value $v(f) = f^{\text{out}}(s) = f^{\text{in}}(t)$.

Flow Network with Cost

Greedy Approach

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

Note: In G_F, let e' be the reverse edge of e. The $\$_{e'} = -\$_e$.

How do we find the cheapest augmenting path?

- Bellman-Ford shortest path
  - Negative costs and it is possible to show that there will be no negative cycles.
- Special Case for Flow Networks: It is possible to modify the weights to remove negative costs and use Dijkstra's to improve the runtime.