CS 577 - Network Flow

Manolis Vlatakis

Department of Computer Sciences
University of Wisconsin – Madison

Fall 2024

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Network Flow

Network Flow

Flow Problems

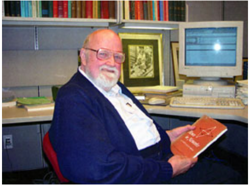
- Flow Network / Transportation Networks: Connected directed graph with water flowing / traffic moving through it.
- Edges have limited capacities.
- Nodes act as switches directing the flow.
- Many, many problems can be cast as flow problems.

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.

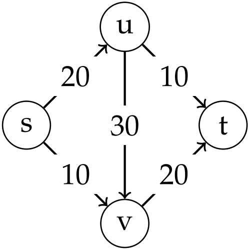
Ford-Fulkerson Method (1956)



L R Ford Jr.

D. R. Fulkerson

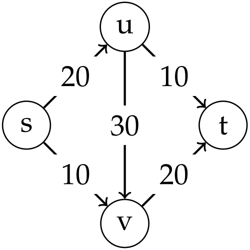
Flow Network



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

Flow Network



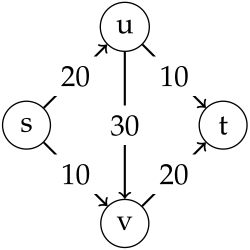
Basic Flow Network

- 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 Network



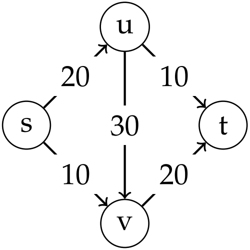
Basic Flow Network

- 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 Network



Basic Flow Network

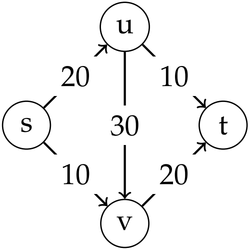
- Directed graph $G = (V, E)$.
- Each edge e has $c_e \geq 0$.
- Source $s \in V$ and sink $t \in V$.
- Internal node $V \setminus \{s, t\}$.

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

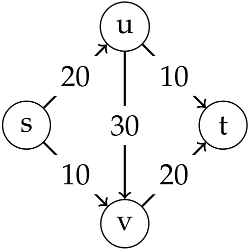
Defining Flow

- Flow starts at s and exits at t.
- Flow function: $f : E \to R^+; f(e)$ is the flow across edge e.
- Flow Conditions:
  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 $v(f) = f^{\text{out}}(s) = f^{\text{in}}(t)$.

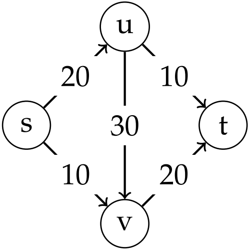
Maximum-Flow Problem



Max-Flow

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

Maximum-Flow Problem



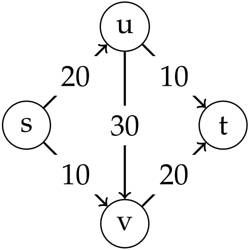
Max-Flow

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 $t \in B$.

Maximum-Flow Problem



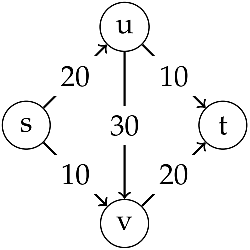
Max-Flow

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.

Maximum-Flow Problem



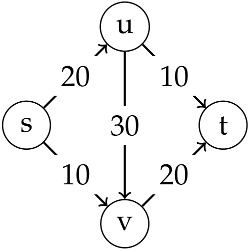
Max-Flow

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

Maximum-Flow Problem



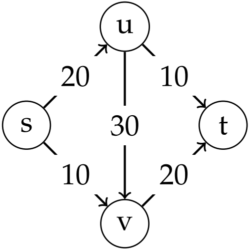
Max-Flow

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.

Maximum-Flow Problem



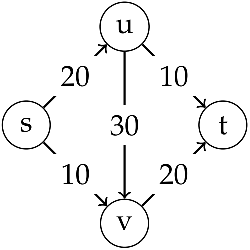
Max-Flow

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

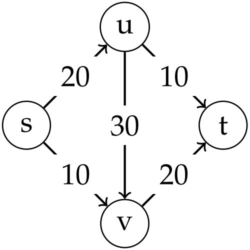
- A Cut: Partition of V into sets (A, B) with $s \in A$ and $t \in B$.
- Flow from s to t must cross the set A to B.
- Cut capacity: $c(A, B) = \sum_{e \text{ out of } A} c_e$
- Minimum-cut of G: The cut (A^*, B^*) that minimizes $c(A^*, B^*)$ for G.
- The min-cut and max-flow are the same value for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

Designing the Approach



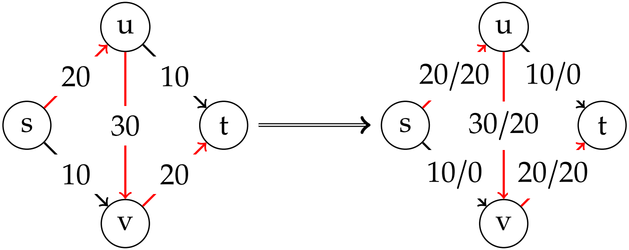
🤔What is the max-flow value in the example?

Designing the Approach



🤔What is the min-cut value in the example?

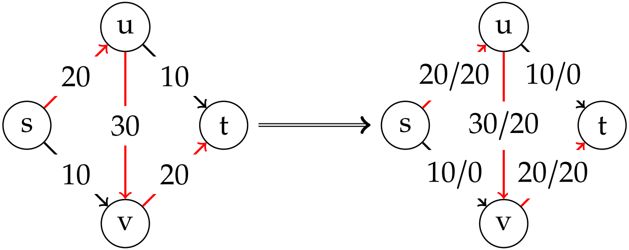
Designing the Approach



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.

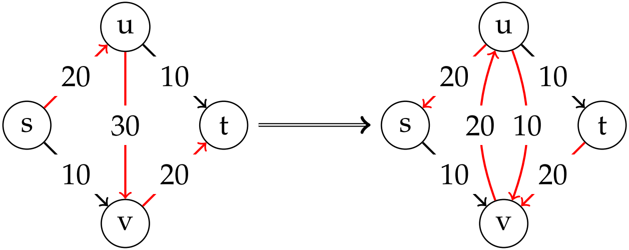
Designing the Approach



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

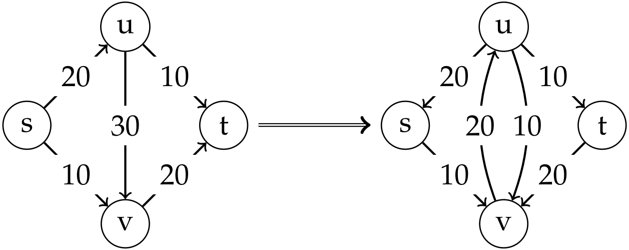


Residual Graph

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 $f(e)$.

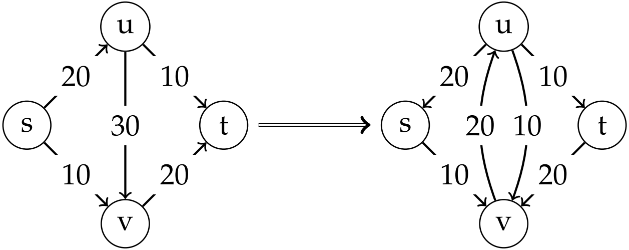
Augmenting Path



Augmenting Path

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

Augmenting Path

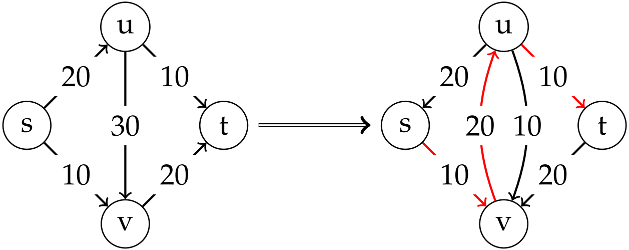


Augmenting Path

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

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

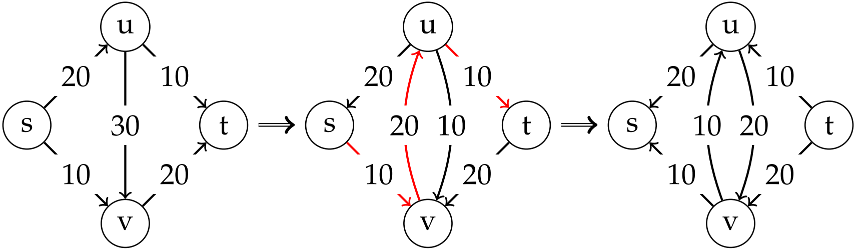
Augmenting Path



Increasing the Flow along Augmenting Path

- Push 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.

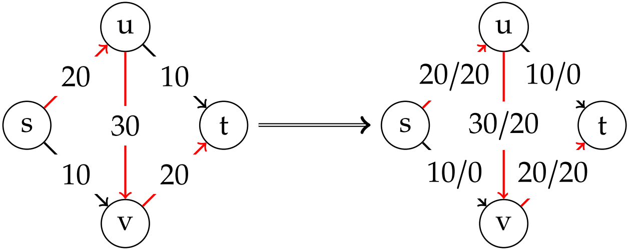
Augmenting Path



Increasing the Flow along Augmenting Path

- Push 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.

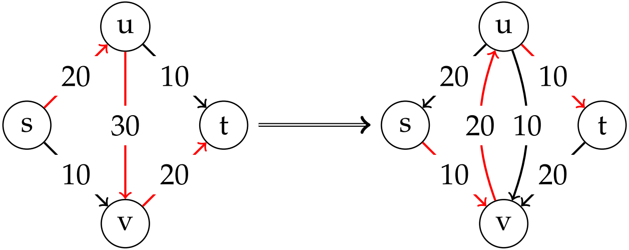
Designing the Approach



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...

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.

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.

🤔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.

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.

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.

🤔What technique?

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

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.

Theorem 2

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

Analyzing the Algorithm

Runtime

Observation 2

Since G is connected,
$m \geq$ 🤔*Dominating Factor?*.

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

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.

Analyzing the Algorithm

Runtime

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.

Observation 2

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

Theorem 3

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

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 $O(mC)$.

🤔Is this a polynomial bound?

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 BOTTLENECK(P, G_f) along P.

Theorem 3

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

🤔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.

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:

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:
  1. Find an augmenting path: 🤔How can we do that?

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:
  1. Find an augmenting path: BFS or DFS: $O(m + n)$.

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:
  1. Find an augmenting path: BFS or DFS: $O(m + n)$.
  2. Update flow along path P: 🤔Time bound?

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 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:
    1. Find an augmenting path: BFS or DFS: $O(m + n)$.
    2. Update flow along path P: $O(n)$.

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:
  1. Find an augmenting path: BFS or DFS: $O(m + n)$.
  2. Update flow along path P: $O(n)$.
  3. Build new G_f: 🤔Time bound?

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 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:
  1. Find an augmenting path: BFS or DFS: $O(m + n)$.
  2. Update flow along path P: $O(n)$.
  3. Build new G_f: $O(m)$.

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

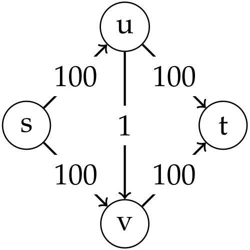
Suppose all capacities are integers. Then, runtime of $O(mC)$.

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: $O(m)$.

□

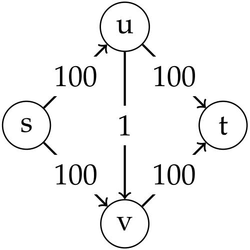
Choosing Good Augmenting Paths



Idea

- Choose paths with large bottlenecks.

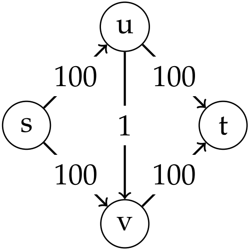
Choosing Good Augmenting Paths



Idea

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

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: .

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: $1 + \lceil \lg C \rceil$.

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:.

Analyzing the Scaled Version

Scaled Version

- Initialize $f(e) = 0$ for all edges.
- Initialize $\Delta := \max_i \left(2^i\right)$ such that $2^i \le \max_{e \text{ out of } s}(c_e)$.
- While $\Delta \ge 1$:
  - 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:$O(m)$.

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: .

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: $O(m)$.

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: $O(m^2 \log C)$.

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: $O(m^2 \log C)$.

🤔Is this polynomial?

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: $O(m^2 \log C)$.

🤔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.

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

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

Recall Cut

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

Lemma 4

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

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

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

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,

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

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

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 $v(f) = c(A^*, B^*)$.

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 $B^* = V \setminus A^*$.

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 $B^* = V \setminus A^*$.
- (A^*, B^*) is an $s-t$ cut:
  - Partition of V
  - $s \in A^*$ and $t \in B^*$

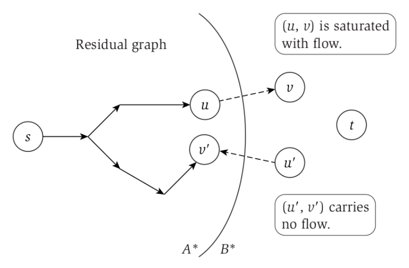
Max-Flow equals Min-Cut

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 path in G_f.
  Let $B^* = V \setminus A$



(u, v) is saturated with flow.

(u', v') carries no flow.

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

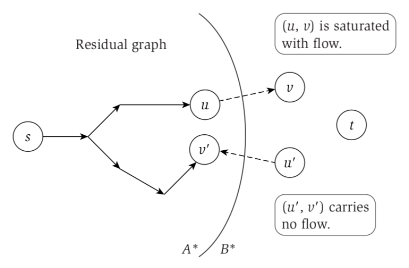
Max-Flow equals Min-Cut

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 path in G_f.
  Let $B^* = V \setminus A$



- Consider $e = (u', v')$: Claim $f(e) = 0$.
  - If not, then $s - u'$ 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^*).

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.

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.

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.

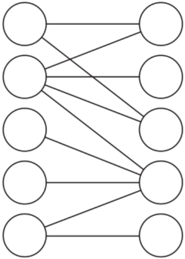
Proof.

- 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

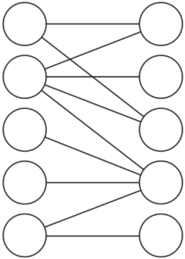
Bipartite Matching Problem



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).

Bipartite Matching Problem



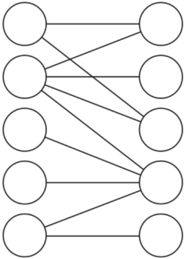
Definition

- 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



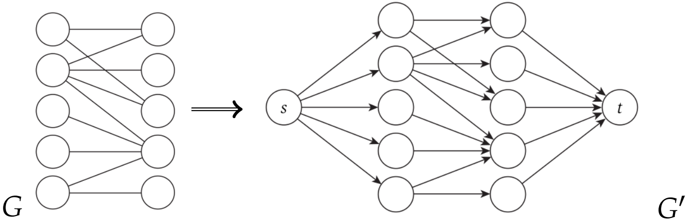
Definition

- 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

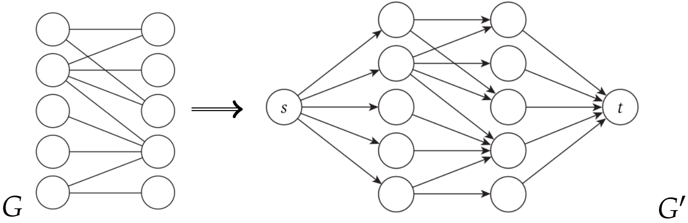
- How can the problem be encoded in a graph?
- Source/sink: Are they naturally in the graph encoding, or do additional nodes and edges have to be added?
- For each edge: What is the direction? Is it bi-directional? What is the capacity?

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.

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.

Bipartite Matching to Flow Network

Theorem 10

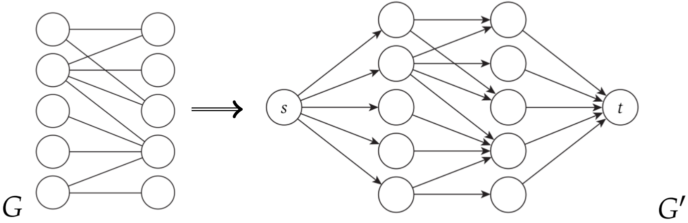
$|M^*|$ in G is equal to the max-flow of G', and the edges carrying the flow correspond to the edges in the maximum matching.

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.

□

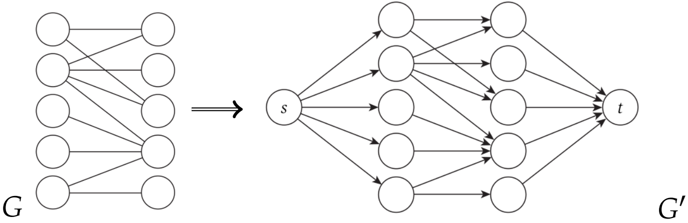
Bipartite Matching to Flow Network



Runtime

- Assume $n = |X| = |Y|, m = |E|$.

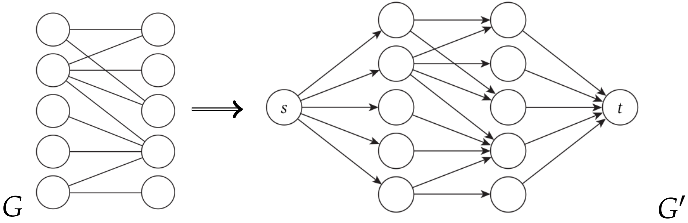
Bipartite Matching to Flow Network



Runtime

- Assume $n = |X| = |Y|, m = |E|$.
- Overall: .

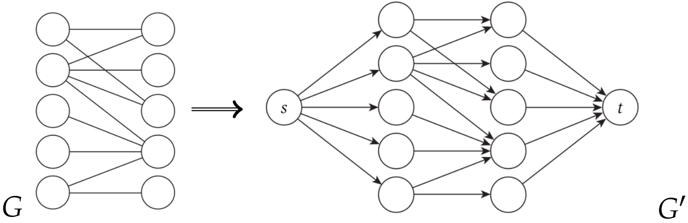
Bipartite Matching to Flow Network



Runtime

- Assume $n = |X| = |Y|, m = |E|$.
- Overall: $O(mn)$.

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.

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:

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.

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:

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'.

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: $O(mC) = O(mn)$.

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?

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

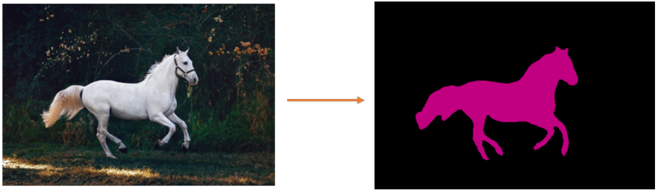
- Basic FF method: $O(mC) = O(mn)$.

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 $q(A,B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{i,j \in P : |A \cap \{i,j\}|=1} p_{ij}$

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 $Q = \sum_{i \in P}(a_i + b_i)$. Express $q(A, B)$ using Q.

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

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

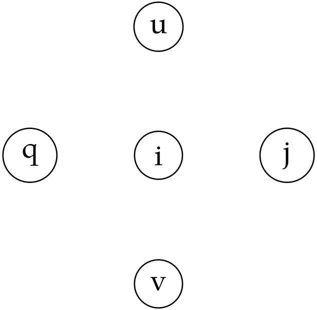
- Maximize $q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{i,j \in P : |A \cap \{i,j\}| = 1} p_{ij}$
- Let $Q = \sum_{i \in P}(a_i + b_i)$.
  $q(A, B) = Q - \sum_{i \in B} a_i - \sum_{j \in A} b_j - \sum_{i,j \in P : |A \cap \{i,j\}| = 1} p_{ij}$
- Equivalent goal:
  Minimize $\sum_{i \in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

Algorithm Design

Reduction

- How can we represent this problem as a graph? What are the nodes?

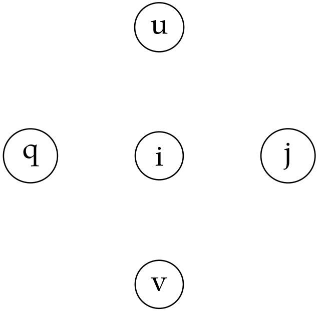
Algorithm Design



Reduction

- Each pixel becomes a node.

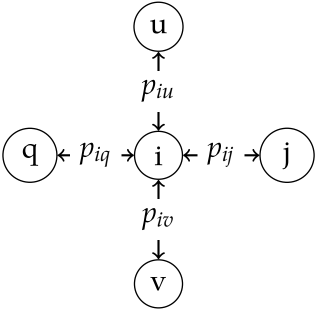
Algorithm Design



Reduction

- Each pixel becomes a node.
- What about the edges?

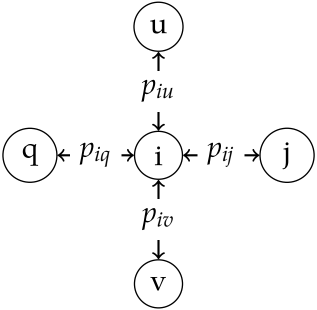
Algorithm Design



Reduction

- Each pixel becomes a node.
- Add edges between neighbours i and j with capacity p_{ij}.

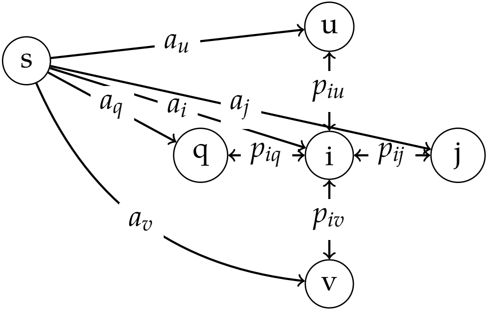
Algorithm Design



Reduction

- Each pixel becomes a node.
- Add edges between neighbours i and j with capacity p_{ij}.
- What about source and target?

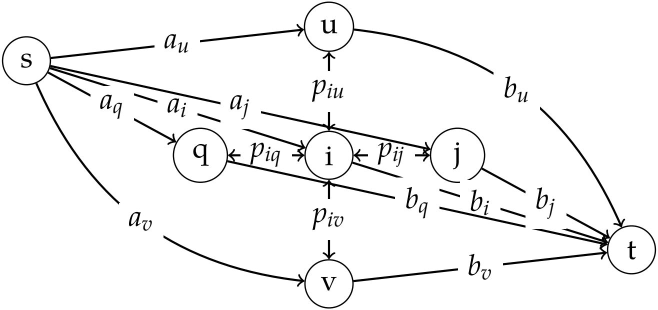
Algorithm Design



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.

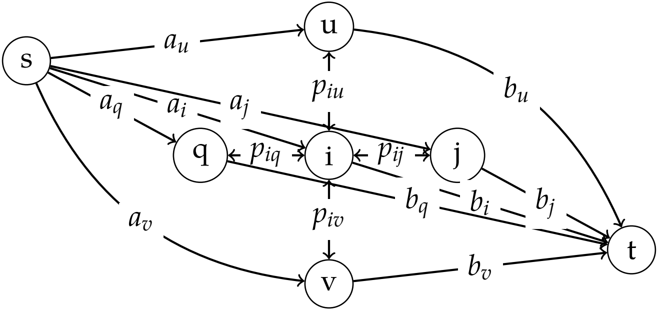
Algorithm Design



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.

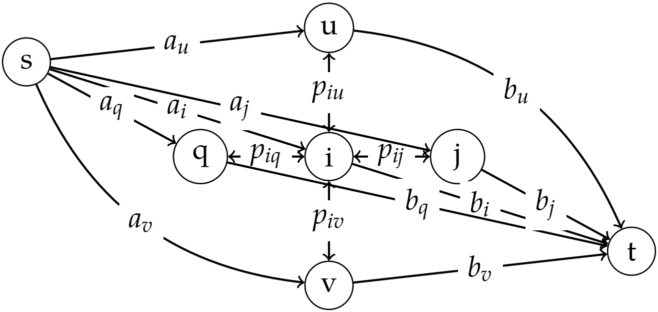
Algorithm Design



Solution

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

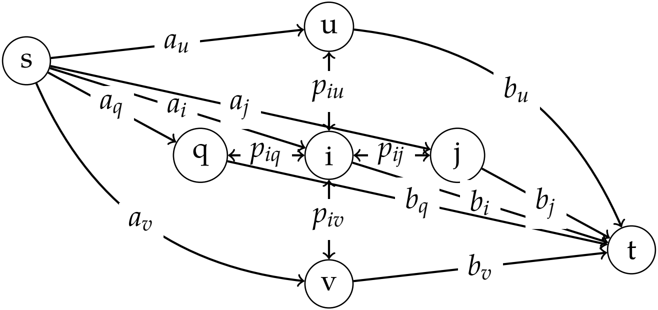
Algorithm Design



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.

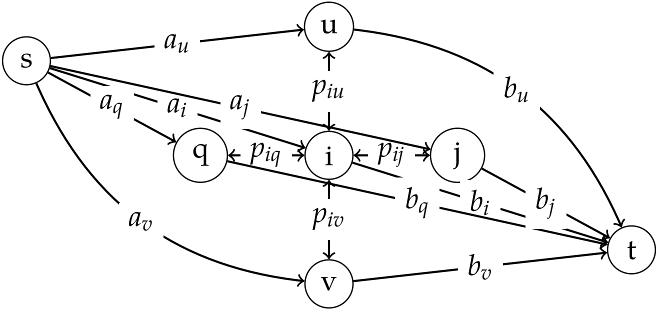
Algorithm Design



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.

Algorithm Design



Solution

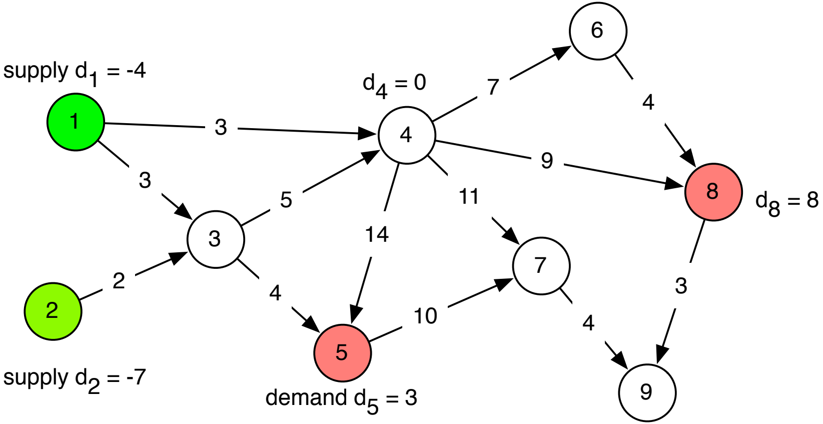
- Min-cut will minimize $\sum_{i \in B} a_i + \sum_{j \in A} b_j + \sum_{i,j \in P : |A \cap \{i,j\}|=1} p_{ij}$.
- Consider $i \in A$: Foreground and contributes b_i to cut.
- Consider $j \in B$: Background and contributes a_i to cut.
- Consider $i \in A, j \in B$ and i, j adjacent: contributes p_{ij}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

Node Demand and Lower Bounds

Circulations with Demands

- Suppose we have multiple sources and multiple sinks.
- Each sink wants to get a certain amount of flow (its **demand**).
- Each source has a certain amount of flow to give (its **supply**).
- We can represent supply as **negative demand**.

Demand Example



Constraints

Goal: find a flow f that satisfies:

1. Capacity constraints: For each $e \in E$, $0 \leq f(e) \leq c_e$.
2. Demand constraints: For each $v \in V$,

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

The demand d_v is the excess flow that should come into node.

Sources and Sinks

Let S be the set of nodes with **negative** demands (supply).

Let T be the set of nodes with **positive** demands (demand).

In order for there to be a feasible flow, we must have:

$$\sum_{s \in S} -d_s = \sum_{t \in T} d_t$$

Let $D = \sum_{t \in T} d_t$.

Reduction

How can we turn the **circulation with demands** problem into the maximum flow problem?

Reduction

How can we turn the **circulation with demands** problem into the maximum flow problem?

1. Add a new source s^* with an edge (s^*, s) from s^* to every node $s \in S$.

2. Add a new sink t^* with an edge (t, t^*) from t^* to every node $t \in T$.

Reduction

How can we turn the **circulation with demands** problem into the maximum flow problem?

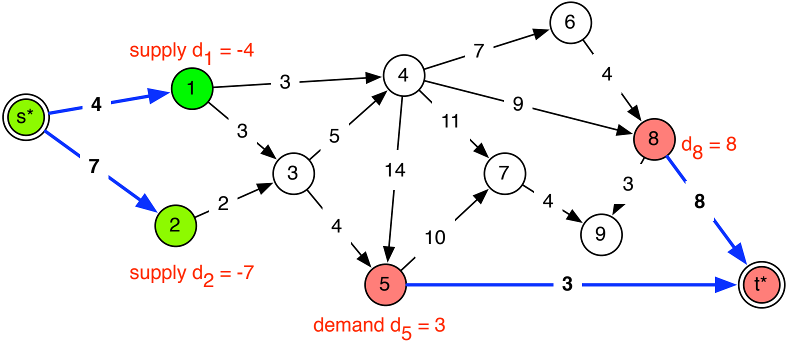
1. Add a new source s^* with an edge (s^*, s) from s^* to every node $s \in S$.

2. Add a new sink t^* with an edge (t, t^*) from t^* to every node $t \in T$.

The capacity of edges $(s^*, s) = -d_s$ (since $d_s < 0$, this is $+$ve)

The capacity of edges $(t, t^*) = d_t$.

Circulation Reduction Example



Feasible circulation if and only if there is a flow of value $D = \sum_{t \in T} d_t$.

Notes

Intuition:

- Capacity of edges (s^*, s) limit the supply for source nodes s.
- Capacity of edges (t, t^*) require that d_t flow reaches each t.

Hence, we can use max-flow to find these circulations.

Lower Bounds

Another extension: what if we want **lower** bounds on what flow goes through some edges?

In other words, we want to require that some edges are used.

Goal: find a flow f that satisfies:

1. Capacity constraints: For each $e \in E$, $\ell_e \le f(e) \le c_e$.
2. Demand constraints: For each $v \in V$,

$$f^{\mathrm{in}}(v) - f^{\mathrm{out}}(v) = d_v.$$

Lower Bounds

Suppose we defined an initial flow f_0 by setting the flow along each edge equal to the lower bound. In other words: $f_0(e) = \ell_e$.

This flow satisfies the capacity constraints, but not the demand constraints.

Define: $L_v = f_0^{\mathrm{in}}(v) - f_0^{\mathrm{out}}(v)$.

Recall that the demand constraints say that $f^{\mathrm{in}}(v) - f^{\mathrm{out}}(v) = d_v$. Hence, L_v is equal to the amount of the demand that f_0 satisfies at node v.

New Graph

For each node, our flow f_0 satisfies L_v of its demand, hence we have:

New demand constraints:

$$f^{\mathrm{in}}(v) - f^{\mathrm{out}}(v) = d_v - L_v$$

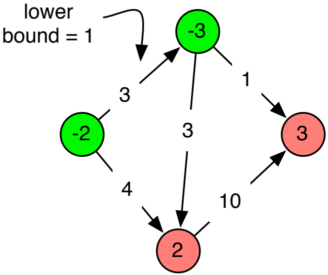
Also, f_0 uses some of the edge capacities already, so we have:

New capacity constraints:

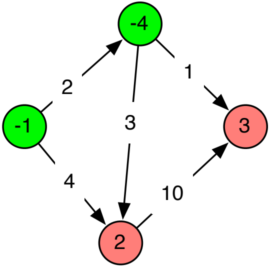
$$0 \leq f(e) \leq c_e - \ell_e$$

These constraints give a standard instance of the circulation problem.

Lower Bound Example



(a) Small instance where one edge has a lower bound. This makes the most obvious flow not feasible.



(b) After transformation, we have an equivalent instance with no lower bounds.

Reduction:

Given a circulation instance G with lower bounds, we:

1. subtract ℓ_e from the capacity of each edge e, and
2. subtract L_v from the demand of each node v.
   (This may create some new "sources" or "sinks".)

We then solve the circulation problem on this new graph to get a flow f'.

To find the flow that satisfies the original constraints, we add ℓ_e to every $f'(e)$.

Summary

We can efficiently find a feasible flow for the following general problem:

Circulations with demands and lower bounds

Given:

- a directed graph G
- a nonnegative lower bound ℓ_e for each edge $e \in G$
- a nonnegative upper bound $c_e \geq \ell_e$ for each edge $e \in G$
- and a demand d_v for every node

Find: a flow f such that

- $\ell_e \leq f(e) \leq c_e$ for every e, and
- $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$ for every v.

Serial Reductions…

We designed the algorithm for this general problem by reducing Circulation with Lower Bounds problem to the Circulation without Lower Bounds problem. We in turn reduced that problem to the Max Flow problem.

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 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 $v \in V$, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

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 $e \in E$, $0 \le f(e) \le c_e$.
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 $\sum_{v \in V} d_v = 0$.

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, $d_v = f^{\text{in}}(v) - f^{\text{out}}(v)$.

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, $f_e^{\text{in}}(v) - f_e^{\text{out}}(u) = 0$.

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

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

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

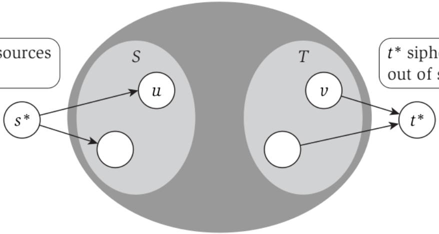
If there is a feasible flow, then

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

Not iff

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

Reduction to Max-Flow



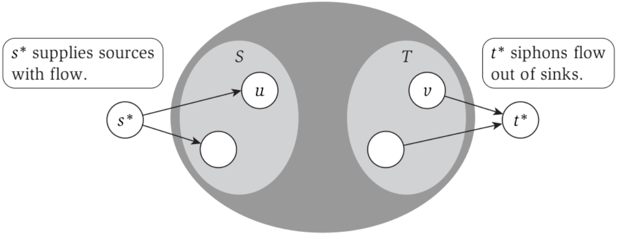
s^* supplies sources with flow.

t^* siphons flow out of sinks.

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 $-d_v$.

Reduction to Max-Flow

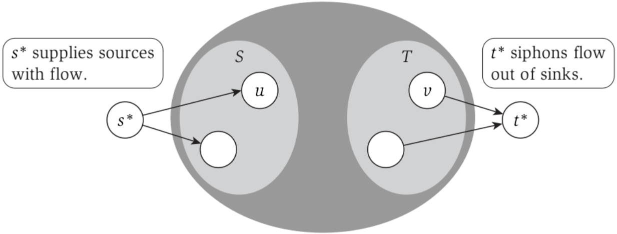


s^* supplies sources with flow.
S
T
u
v
s^*
t^*
t^* siphons flow out of sinks.

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^*.

Reduction to Max-Flow


s^* supplies sources with flow.
S
u
s^*
T
v
t^*
t^* siphons flow out of sinks.

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 $0 \le \ell_e \le c_e$.

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

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

Another Flow Network Extension

Adding Flow Lower Bound

Adding Lower Bound

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

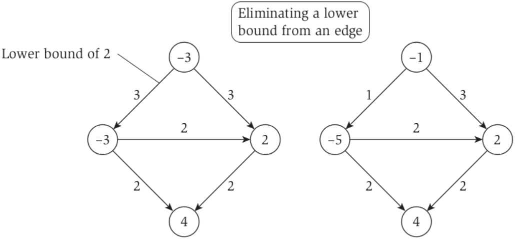
Flow Conditions

1. Capacity: For each $e \in E$, $\ell_e \leq f(e) \leq c_e$.
2. 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?

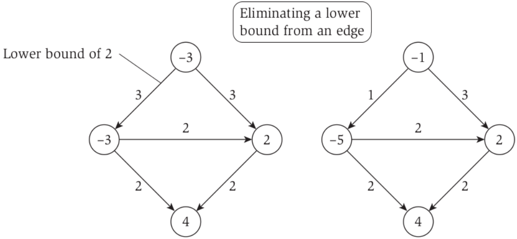
Reduction to Only Demand



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.

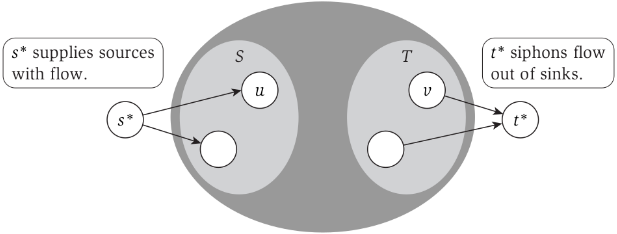
Reduction to Only Demand



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


s^* supplies sources with flow.
S
T
t^* siphons flow out of sinks.
u
v
s^*
t^*

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 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.

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.

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.

Survey Design

Problem

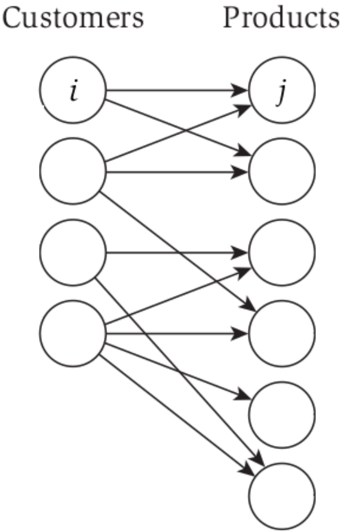
- Study of consumer preferences.
- A company, with k products, has a database of n customer purchase histories.
- Goal: Define a product specific survey.

🤔What type of graph to use?

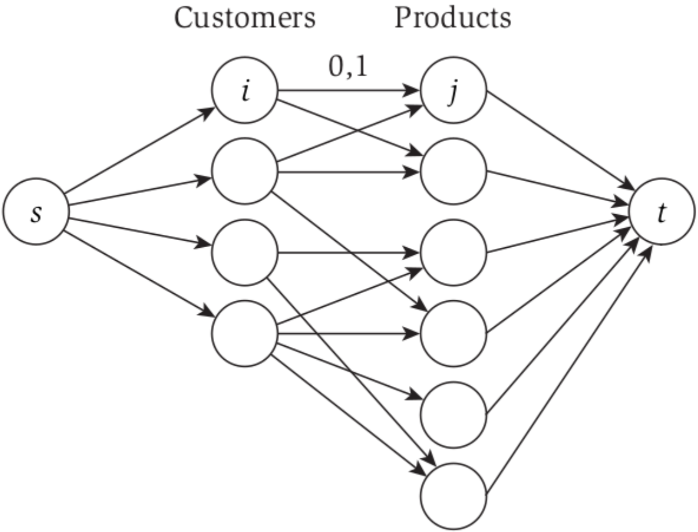
Survey Rules

- 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.

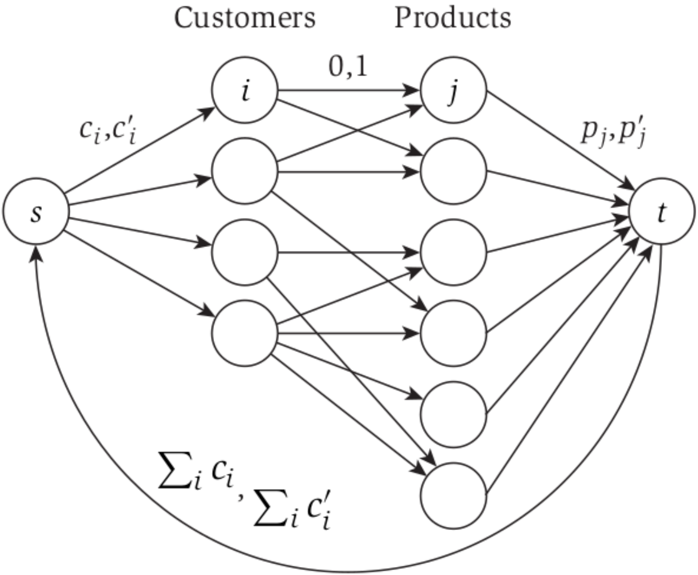
Algorithm Design



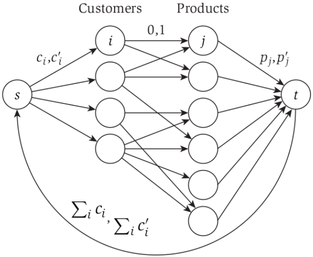
Algorithm Design



Algorithm Design



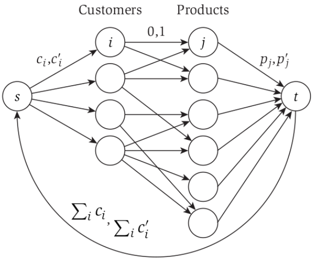
Algorithm Design



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;

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.

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)

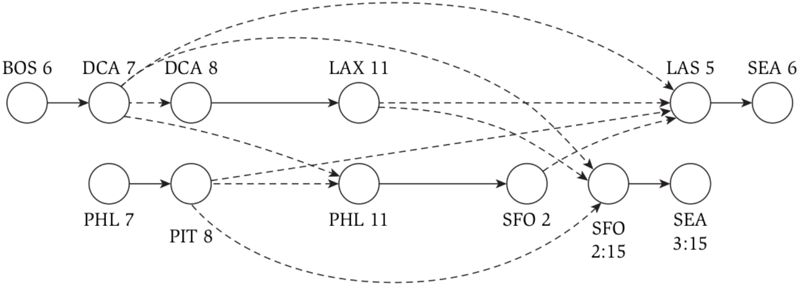
Rules

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

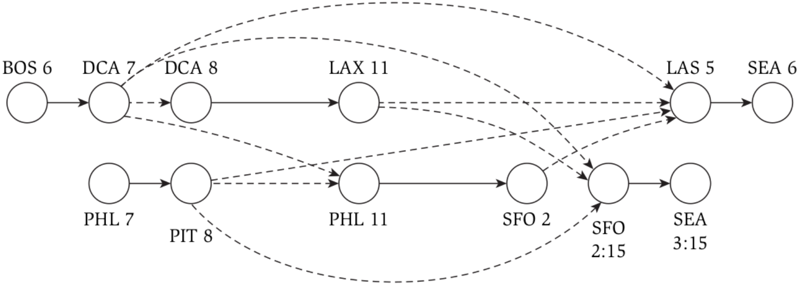


$k = 2$ planes

Exercise: Reduce to a flow network

Hint: Use lower bounds and demand.

Algorithm Design



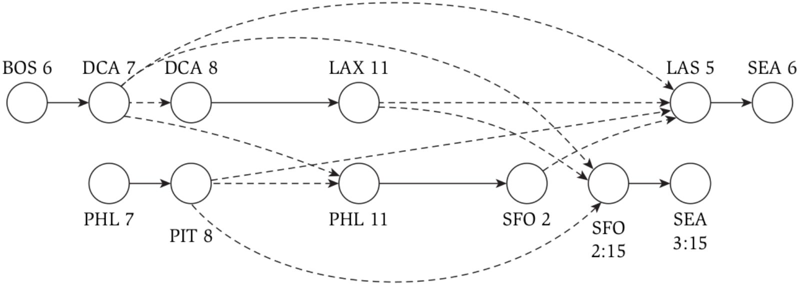
$k = 2$ planes

Exercise: Reduce to a flow network

Hint: Use lower bounds and demand.

- 🤔 Are s-t new nodes?
- 🤔 What is the max capacity of the edges from G?

Algorithm Design



$k = 2$ planes

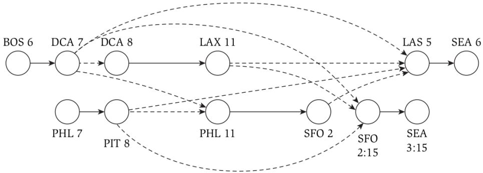
Exercise: Reduce to a flow network

Hint: Use lower bounds and demand.

- 🤔 In the example, how many edges out from s?
- 🤔 In the example, how many edges in to t?

Algorithm Design

$k = 2$ planes

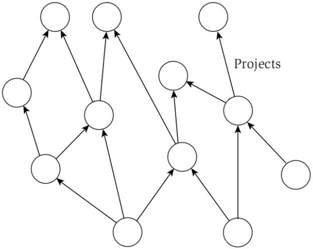


Reduction

- 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

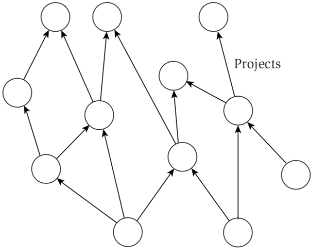
Project Selection



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.

Project Selection

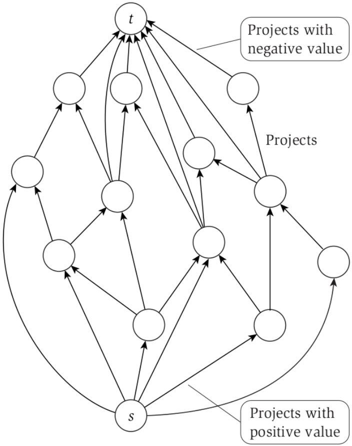


Use Min-Cut to solve this problem.

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.

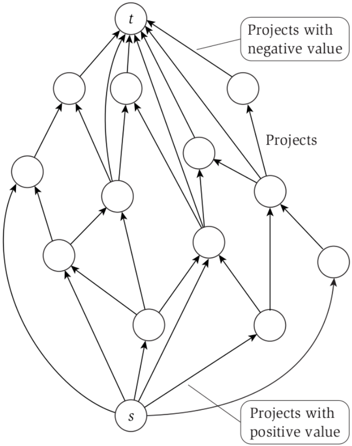
Algorithm Design



Reduction

- Use Min-Cut

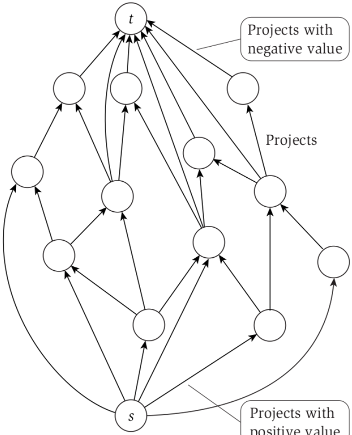
Algorithm Design



Reduction

- Use Min-Cut
- Add s with edge to every project i with $p_i > 0$ and capacity p_i.

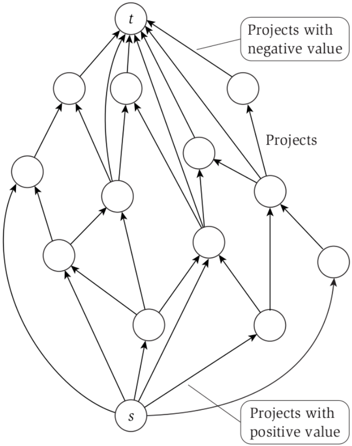
Algorithm Design



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 $-p_i$.

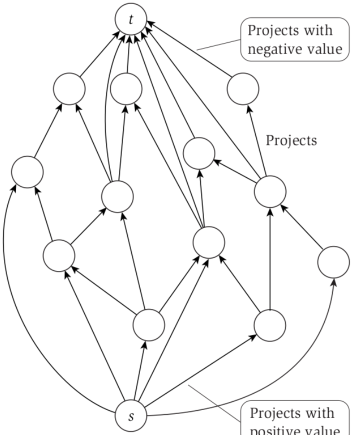
Algorithm Design



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 $-p_i$.
- $C = \sum_{i \in P : p_i > 0} p_i$

Algorithm Design

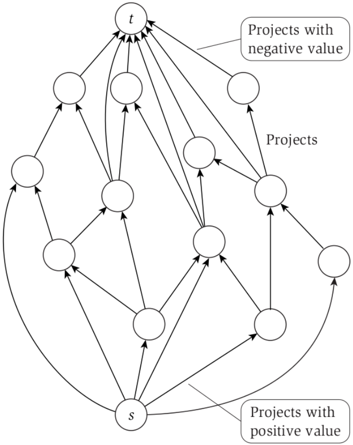


Reduction

- 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 $-p_i$.
- $C = \sum_{i \in P : p_i > 0} p_i$

🤔: What is the capacity of the cut $(\{s\}, P \cup \{t\})$?

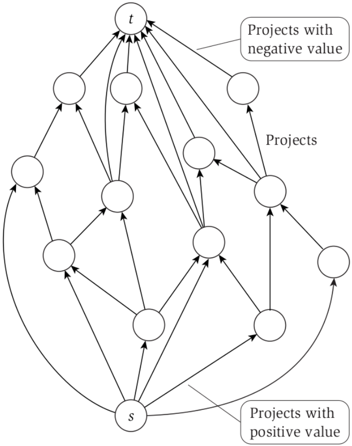
Algorithm Design



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 $\leq C = \sum_{i \in P : p_i > 0} p_i$ which is the capacity $(\{s\}, P \cup \{t\})$

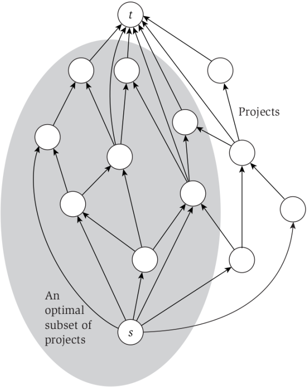
Algorithm Design



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 $C + 1$).

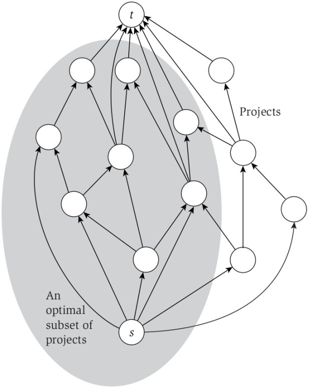
Algorithm Analysis



Observation 4

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

Algorithm Analysis



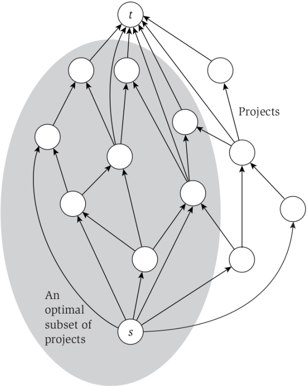
Observation 4

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

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

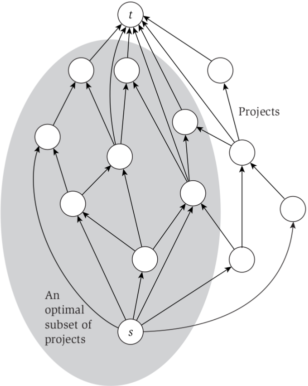
Proof.

Consider the different edges:

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

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

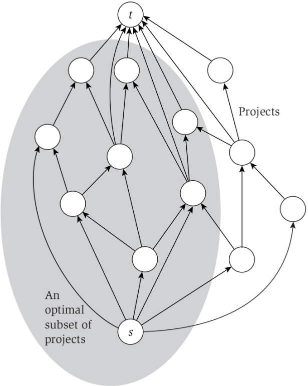
Algorithm Analysis



Theorem 12

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

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.

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

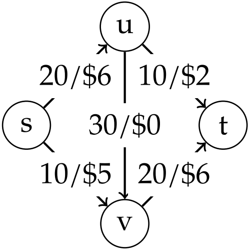
$$\iff \textsc{profit}(A) = C - c(A', B')$$

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

□

Min-Cost Max Flow

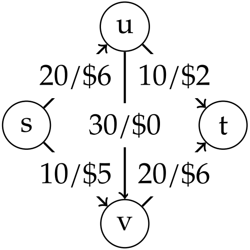
Flow Network with Cost



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.

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.

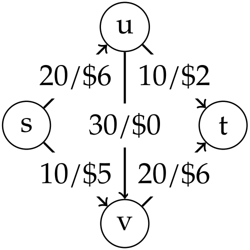
Defining Flow

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

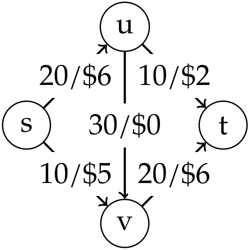
Flow Network with Cost



Min-Cost Max-Flow

Given a flow network G, what is the flow f of minimum cost that maximizes $v(f)$?

Flow Network with Cost



Min-Cost Max-Flow

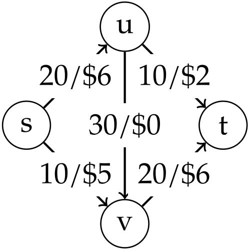
Given a flow network G, what is the flow f of minimum cost that maximizes $v(f)$?

Greedy Approach

How do we make this give us the min-cost max-flow?

- 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.

Flow Network with Cost



Min-Cost Max-Flow

Given a flow network G, what is the flow f of minimum cost that maximizes $v(f)$?

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 $\$_{e'} = -\$_e$.

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?

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

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.

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 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.

Baseball Elimination

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

🤔: Is Boston Eliminated?

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

🤔: Is Boston Eliminated? Yes.

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

Why is Boston eliminated?

Case analysis:

- Boston must win its 2 remaining games.

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

Why is Boston eliminated?

Case analysis:

- Boston must win its 2 remaining games.
- New York must lose its 2 remaining games.

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

Why is Boston eliminated?

Case analysis:

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

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

Why is Boston eliminated?

Analytical approach:

- Boston can finish with ≤ 92 wins.

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

Why is Boston eliminated?

Analytical approach:

- Boston can finish with ≤ 92 wins.
- Currently, other 3 teams have 274 combined wins with 3 remaining games between them:

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

Why is Boston eliminated?

Analytical approach:

- Boston can finish with ≤ 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.

Baseball Elimination

	Wins	Games Left
New York	92	NYY vs TOR
Toronto	91	TOR vs BAL
Baltimore	91	BAL vs BOS
Boston	90	BOS vs TOR
		NYY vs BAL

Why is Boston eliminated?

Analytical approach:

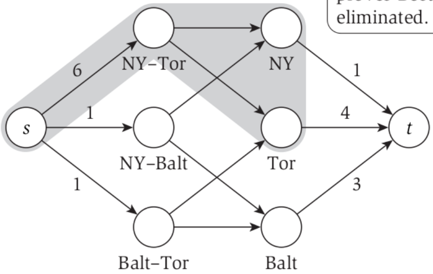
- Boston can finish with ≤ 92 wins.
- Currently, other 3 teams have 274 combined wins with 3 remaining games between them:
    - Overall, at the end, there will be 277 combined wins between the other 3 teams.
    - Average of 92 1/3 wins which implies that one team will have at least 92 1/3 $\implies$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



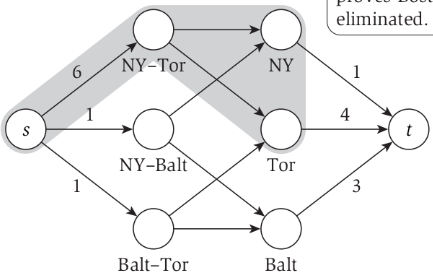
The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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



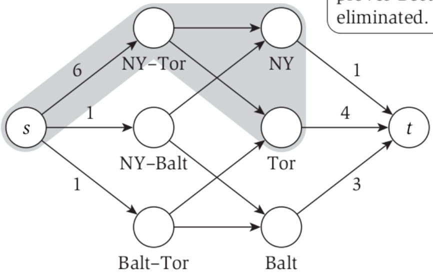
The set $T = \{\text{NY}, \text{Toronto}\}$ proves Boston is eliminated.

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 ∞ (or g_{xy}).

Algorithm Design



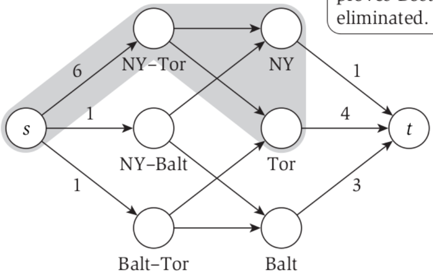
The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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.

Algorithm Design



The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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

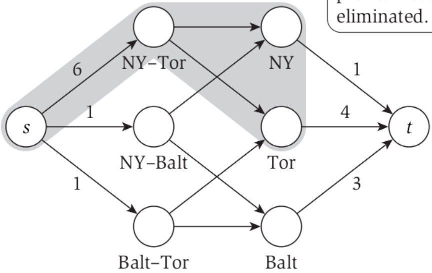
Solution

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

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

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

Algorithm Design



The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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

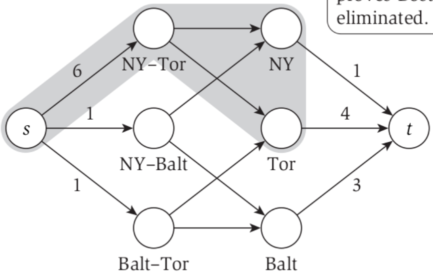
Solution

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

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

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

Algorithm Design



The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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

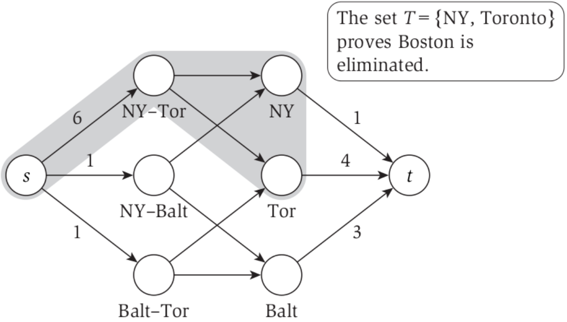
Solution

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

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

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

Algorithm Design



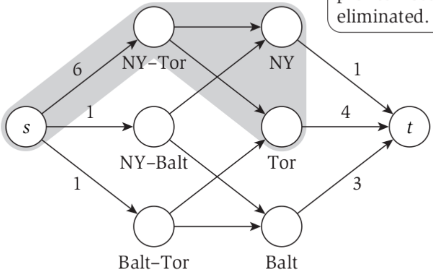
The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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



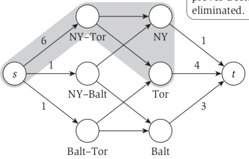
The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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: $m|T| < \sum_{x,y \in T} g_{xy} + \sum_{x \in T} w_x$

Solution Characterization



The set $T = \{\text{NY}, \text{Toronto}\}$ proves Boston is eliminated.

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

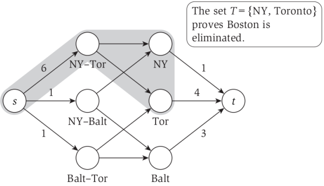
Theorem 13

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 $c(A, B) = g' \leq \min\{\sum_{x,y \in S'} g_{xy}, \sum_{x \in S'} m - w_x\}$.

Solution Characterization



The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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

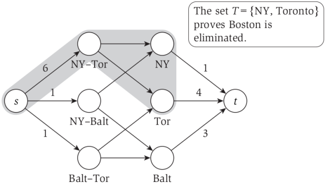
Theorem 13

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


The set $T = \{$NY, Toronto$\}$ proves Boston is eliminated.
s
NY–Tor
NY–Balt
Balt–Tor
NY
Tor
Balt
t
6
1
1
1
4
3

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

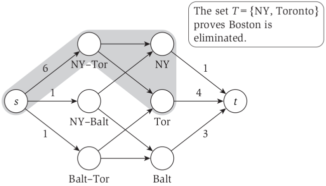
Theorem 13

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



The set $T = \{\text{NY, Toronto}\}$ proves Boston is eliminated.

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

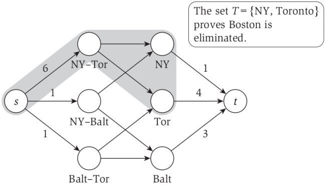
Theorem 13

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



The set $T = \{\text{NY}, \text{Toronto}\}$ proves Boston is eliminated.

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

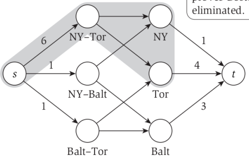
Theorem 13

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

Solution Characterization



The set $T = \{$NY, Toronto$\}$ proves Boston is eliminated.

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

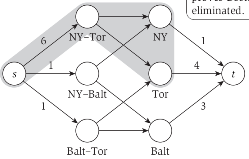
Theorem 13

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



The set $T = \{\text{NY}, \text{Toronto}\}$ proves Boston is eliminated.

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

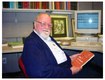
□

Appendix

References

Image Sources I

https://upload.wikimedia.org/wikipedia/en/2/25/Delbert_Ray_Fulkerson.png



https://angelberh7.wordpress.com/2014/10/08/biografia-de-lester-randolph-ford-jr/

https://getthematic.com/insights/customer-survey-design/

https://://hexaware.com/industries/travel/airlines/

Image Sources II

http://bluejayhunter.com/2010/01/which-team-was-better-92-or-93-blue.html

https://brand.wisc.edu/web/logos/

https://learnopencv.com/wp-content/uploads/2019/06/semantic-segmentation-examples.png