Objectives

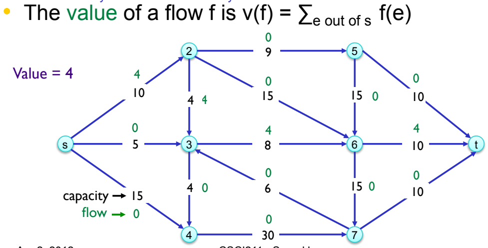
- Network Flow Applications
  - Circulation
  - Survey design
  - Airline scheduling
  - Capacity Scaling

Review

- What is a flow network?
- What is our usual goal given a flow network?
  - How do we reach that goal?
- What is the Ford-Fulkerson algorithm?
- What is the min-cut?
  - How does it relate to the max flow?

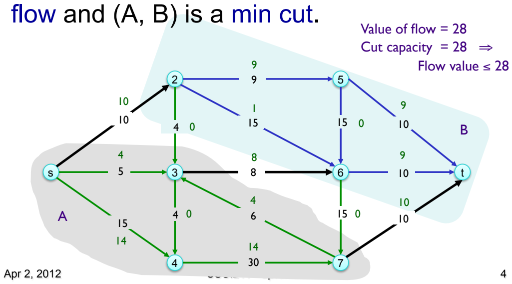
Review: Network Flows

- An **s-t flow** is a function that satisfies
  - ***Capacity* condition**: For each e ∈ E: $0 \le f(e) \le c(e)$
  - ***Conservation* condition**: For each v ∈ V – {s, t}: $\sum_{e \text{ into } y} f(e) = \sum_{e \text{ out of } y} f(e)$
- The value of a flow f is $v(f) = \sum_{e \text{ out of } s} f(e)$



Review: Certificate of Optimality

- Corollary. Let *f* be any flow, and let (A, B) be any cut. If v(f) = cap(A, B), then *f* is a max flow and (A, B) is a min cut.



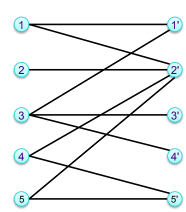
Power of Max Flow Problem

Some problems with non-trivial combinatorial searches can be formulated as **max flow** or **min cut** in a directed graph

Review: Bipartite Graph: Max Flow Formulation

Problem: find matching of largest possible size

How did we turn this into a max flow problem?

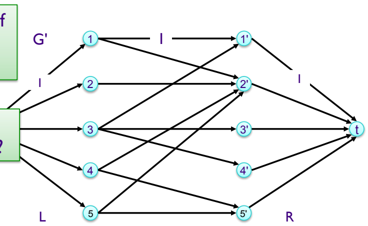


Review: Bipartite Graph: Max Flow Formulation

- Create digraph G' = (L ∪ R ∪ {s, t}, E')
- Direct all edges from L to R, and assign unit capacity
- Add source s, and unit capacity edges from s to each node in L
- Add sink t, and unit capacity edges from each node in R to t

What is cost of generating model?

What is C in this model?



Summary of Approach

1. Model problem as a flow network
2. Run Ford-Fulkerson algorithm
3. Analyze running time
   - Creating model
   - FF algorithm

Review: Circulation with Demands and Lower Bounds

Force flow to use certain edges

- Feasible circulation
  - Directed graph G = (V, E)
  - Edge capacities c(e) and lower bounds ℓ (e), e ∈ E
  - Node supply and demands d(v), v ∈ V

- Def. A *circulation* is a function that satisfies:
  - For each e ∈ E: 0 ≤ ℓ (e) ≤ f(e) ≤ c(e) (capacity)
  - For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem with lower bounds.
Given (V, E, ℓ, c, d), does a circulation exist?

Review: Survey Design

- Design survey asking consumers about products
- Can only survey a consumer about a product if they own it
  - Consumer can own multiple products
- Ask consumer i between c_i and c_i' questions
- Ask between p_j and p_j' consumers about product j

Goal: Design a survey that meets these specs, if possible.

How can we model this problem?

Bipartite Graph

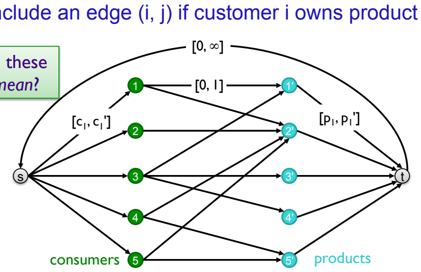
- Nodes: customers and products
- Edge between customer and product means customer owns product
- For each customer, range of # of products asked about
- For each product, range of # of customers asked about it

What does the flow represent?

Survey Design Algorithm

- Formulate as a circulation problem with lower bounds
  - Include an edge (i, j) if customer i owns product j

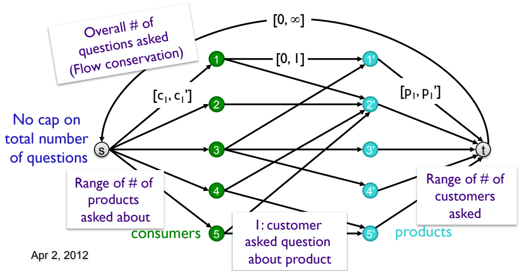
What do these edges *mean?*



Survey Design Algo

Alternative bounds on t→s?
How do we know if we can create a survey?
What is the survey?
How many solutions are there to this problem?

- Formulate as a circulatio bounds
  - Include an edge (i, j) if cu



Survey Solution

- If a feasible, integer flow solution, can create the survey
- Customer *i* will be surveyed about product *j* iff the edge (*i*,*j*) carries a unit of flow

7.9 AIRLINE SCHEDULING

Airline Scheduling

- **Scheduling goal**: efficient in terms of equipment usage, crew allocation, customer satisfaction, …
- **Our simplified problem:**
  - Flight segment: origin & destination airport, departure & arrival time
  - Use a plane for two flight segments (i, j) if
    - i's destination == j's origin & enough time to perform maintenance on plane OR
    - Add a flight segment in between that gets plane to j's origin with adequate time in between

Scheduling Planes

- Maintenance time: 1 hour

Number	Origin	Departure	Destination	Arrival
1	Boston	6 a.m.	DC	7 a.m.
2	Philadelphia	7 a.m.	Pittsburgh	8 a.m.
3	DC	8 a.m.	LAX	11 a.m.
4	Philadelphia	11 a.m.	San Francisco	2 p.m.
5	San Francisco	2:15 p.m.	Seattle	3:15 p.m.
6	Las Vegas	5 p.m.	Seattle	6 p.m.

What is a valid use of one plane for > 1 segment?

Scheduling Planes

- Maintenance time: 1 hour

Number	Origin	Departure	Destination	Arrival
1	Boston	6 a.m.	DC	7 a.m.
2	Philadelphia	7 a.m.	Pittsburgh	8 a.m.
3	DC	8 a.m.	LAX	11 a.m.
4	Philadelphia	11 a.m.	San Francisco	2 p.m.
5	San Francisco	2:15 p.m.	Seattle	3:15 p.m.
6	Las Vegas	5 p.m.	Seattle	6 p.m.

What is a valid use of one plane for > 1 segment?

1 → 3 → 5 ; 1 → 3 → 6

Problem Statement

- A flight j is *reachable* from flight i if it is possible to use the same plane for flight j as flight i

Goal: Determine if it's possible to serve all *m* flights using at most *k* planes

Scheduling Planes

- Maintenance time: 1 hour

Number	Origin	Departure	Destination	Arrival
1	Boston	6 a.m.	DC	7 a.m.
2	Philadelphia	7 a.m.	Pittsburgh	8 a.m.
3	DC	8 a.m.	LAX	11 a.m.
4	Philadelphia	11 a.m.	San Francisco	2 p.m.
5	San Francisco	2:15 p.m.	Seattle	3:15 p.m.
6	Las Vegas	5 p.m.	Seattle	6 p.m.

Could we schedule all flights from previous example with only 2 planes?

Scheduling Planes

- Maintenance time: 1 hour

Number	Origin	Departure	Destination	Arrival
1	Boston	6 a.m.	DC	7 a.m.
2	Philadelphia	7 a.m.	Pittsburgh	8 a.m.
3	DC	8 a.m.	LAX	11 a.m.
4	Philadelphia	11 a.m.	San Francisco	2 p.m.
5	San Francisco	2:15 p.m.	Seattle	3:15 p.m.
6	Las Vegas	5 p.m.	Seattle	6 p.m.

Yes.
Plane A: 1 → 3 → 5
Plane B: 2 → 4 → 6

Problem Statement

- A flight j is *reachable* from flight i if it is possible to use the same plane for flight j as flight i

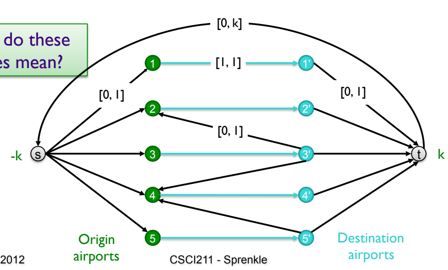
Goal: Determine if it's possible to serve all *m* flights using at most *k* planes

Ideas about our solution/approach?

Airline Scheduling Algorithm

- Flow: airplanes; Nodes: airports
- Find a feasible circulation

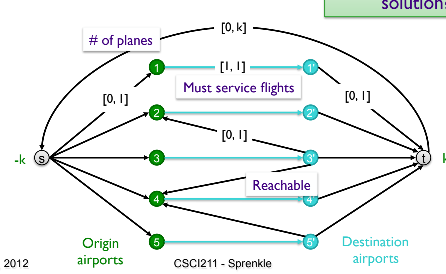
What do these edges mean?



Airline Scheduling Algorithm

- Flow: airplanes; Nodes: airports
- Find a feasible circulation

How do we know if we have a solution? How do we get the solution?



Scheduling Solution

- Model
  - Flow: airplanes
  - Nodes: airports
- Use FF algorithm to generate flow
- Construct schedules by following edges from *s* to origin airports
  - Represents the schedule of one plane

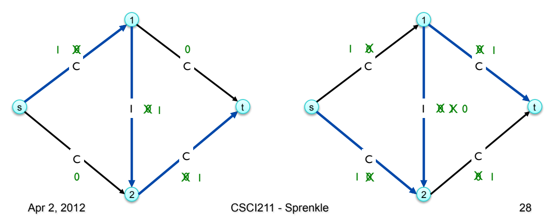
Summary: Network Flow Algorithm Overview

1. Model problem as a network flow
   - What do nodes and edges represent?
   - What do the capacity and flow represent?
   - Do supply and demand play a role? If so, what is it?
   - Why is this a correct model?
2. Solve the problem using circulation
   - Is there a feasible circulation? What does that mean? How do you get the solution you're looking for?
3. Analyze runtime

CHOOSING GOOD AUGMENTING PATHS

Ford-Fulkerson: Exponential Number of Augmentations

- Is generic Ford-Fulkerson algorithm polynomial in input size?
  - No. If max capacity is C, then algorithm can take C iterations.

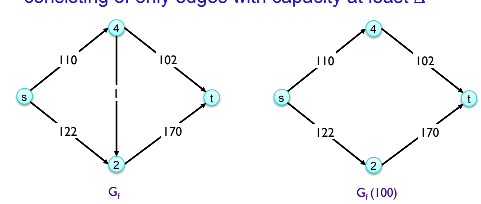


Choosing Good Augmenting Paths

- Use care when selecting augmenting paths
  - Some choices lead to exponential algorithms
  - Clever choices lead to polynomial algorithms
  - If capacities are irrational, algorithm not guaranteed to terminate!
- **Goal: choose augmenting paths so that:**
  - Can find augmenting paths efficiently
  - Few iterations
- [Edmonds-Karp 1972, Dinitz 1970]
  Choose augmenting paths with:
  - Max bottleneck capacity
  - Fewest number of edges
  - *Sufficiently large bottleneck capacity*

Intuition for Capacity Scaling

- Choosing path with highest bottleneck capacity increases flow by max possible amount.
  - Don't worry about finding *exact* highest bottleneck path
  - Maintain scaling parameter Δ
  - Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only edges with capacity at least Δ



Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c)
   foreach e ∈ E,  f(e) = 0
   Δ = greatest power of 2 less than or equal to C
   Gf = residual graph
   Gf(Δ) = Δ-residual graph

   while Δ ≥ 1:
      while there exists augmenting path P in Gf(Δ) :
         f = augment(f, c, P)
         update Gf(Δ)
      Δ = Δ / 2

   return f
```

- Why does this work?
- What is its running time?

Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c)
   foreach e ∈ E,  f(e) = 0
   Δ = greatest power of 2 less than or equal to C
   Gf = residual graph
   Gf(Δ) = Δ-residual graph

   while Δ ≥ 1:     O(log C)
      while there exists augmenting path P in Gf(Δ) :
         f = augment(f, c, P)
         update Gf(Δ)
      Δ = Δ / 2

   return f
```


Capacity Scaling: Correctness

- Assumption. All edge capacities are integers between 1 and C.
- Integrality invariant. All flow and residual capacity values are integral.
- Correctness. If the algorithm terminates, then *f* is a max flow.
- Pf.
  - By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$
  - Upon termination of $\Delta = 1$ phase, there are no augmenting paths. ▪

Capacity Scaling: Running Time

- Lemma 1. The outer while loop repeats $O(\log_2 C)$ times.
- Proof. Initially $\Delta \leq C$. Δ decreases by a factor of 2 each iteration. ▪

Capacity Scaling: Running Time

What happens to the flow's value at each iteration of the loop?

- Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $v(f) + m\,\Delta$.

Proof and further analysis in the book

This Week

- Wiki due Tuesday
  - 7.1-7.2, 7.5, 7.7
- Friday
  - Problem Set 9
- Course evaluations "open" on Wednesday in Sakai
  - Return by following Sunday evening