CSE 589 -- Lecture 3

More than any time in history mankind faces a crossroads. One path leads to despair and utter hopelessness, the other to total extinction. Let us pray that we have the wisdom to choose correctly.

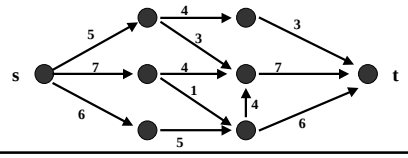
-- Woody Allen

Plan for Today

- More max-flow
- Generalizations, including min-cost max flow
- Linear programming

Maximum Flow

- Input description: a graph (network) G where each edge (v,w) has associated capacity c(v,w), and a specified source node s and sink node t
- Problem description: What is the maximum flow you can route from s to t while respecting the capacity constraint of each edge?



Max-flow outline:

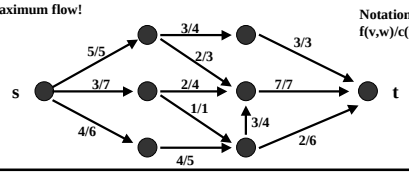
- properties of flow -- last time
- augmenting paths -- last time
- max-flow min-cut theorem -- last time
- Ford-Fulkerson method
- Edmonds-Karp method
- preflow-push methods
- applications

Properties of Flow: f(v,w) -- flow on edge (v,w)

- 0<= f(v,w) <= c(v,w) : the flow through an edge cannot exceed the capacity of an edge
- for all v except s,t $\Sigma_u f(u,v) = \Sigma_w f(v,w)$: the total flow entering a vertex is equal to total flow exiting vertex
- total flow leaving s = total flow entering t.

Not a maximum flow!

Notation on edges f(v,w)/c(v,w)

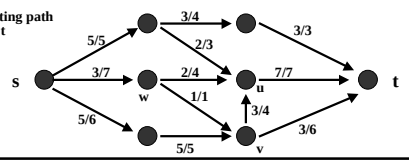


An augmenting path with respect to a given flow f is a

Directed path from s to t which consists of edges from G, but **not necessarily in same direction**. Each edge on path is either:

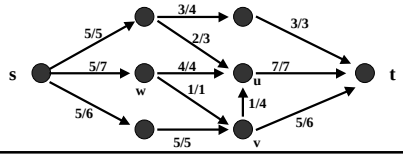
- forward edge: (u,v) in same direction as G and f(u,v) < c(u,v). (c(u,v)-f(u,v) called slack) => has room for more flow.
- backward edge: (u,v) in opposite direction in G (i.e., (v,u) in E) and f(u,v) >0 => can borrow flow from such an edge.

augmenting path s-w-u-v-t



Using an augmenting path to increase flow

- If all edges are forward => move flow through all of them. amount of flow can push = minimum slack
- Otherwise, push flow forward on forward edges, deduct flow from backward edges.



Augmenting Path Theorem:

A flow f is maximum iff it admits no augmenting path

- Already saw that if flow admits an augmenting path, then it is not maximum.
- Suppose f admits no augmenting path. Need to show f maximum.

Cut -- a set of edges that separate s from t.

Capacity of a cut = sum of capacities of edges in cut.

Prove theorem by showing that there is a cut whose capacity = f.

A = vertices s.t. for each v in A, there is augmenting path from s to v. => defines a cut.

Claim: for all edges in cut, f(v,w)=c(v,w)

=> value of flow = capacity of cut defined by A => maximum.

=> Celebrated Max-flow Min-Cut Theorem

- The value of a maximum flow in a network is equal to the minimum capacity of a cut.

- The Integral Flow Theorem

  If the capacities of all edges in the network are integers, then there is a maximum flow whose value is an integer.

Residual Graph w.r.t. flow f

- R=(V,E')

an edge (v,w) in E' if either

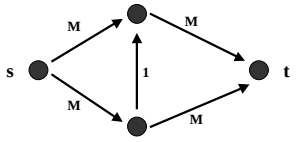
- it is forward edge
  - capacity = c(v,w)-f(v,w)
- or it is a backward edge
  - capacity = f(v,w).

- An augmenting path = a regular directed path from s to t in the residual graph.

Ford-Fulkerson Method (G,s,t)

Initialize flow on all edges to 0.
While there is a path p from s to t in residual network R
  δ = minimum capacity along p in R
  augment δ units of flow along p and update R

Running Time?



Edmonds-Karp

- If implement computation of augmenting path using breadth-first search, i.e., if augmenting path is a **shortest path** from s to t in residual network, then the number of augmentations O(|V||E|).
  - The shortest-path distance from v to t in R is non-decreasing.
  - The number of times an edge (u,v) can become **critical**, i.e., residual capacity of (u,v) = residual capacity of augmenting path, is O(|V|).

The shortest path distance from v to t in R_f is non-decreasing.

Proof by contradiction:

Consider first time SP to t decreases from some node v. Let v be the closest such node to t. SP from v to t begins with edge (v,w)

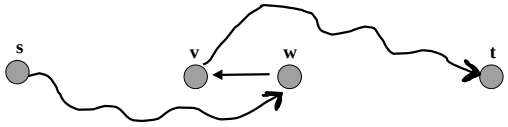
d'(v)=d'(w)+ 1.

Claim: f(v,w)=c(v,w). If not (v,w) in R_f

d(v) <= d(w)+1 <= d'(w)+1 = d'(v). Contradiction.

Shortest paths non-decreasing, cont.

Therefore (w,v) in R_f, and in order for (v,w) in R_f, augmenting path must be:



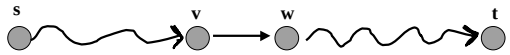
=> d(v) = d(w)-1 <= d'(w)-1 = d'(v)-2

=> d(v) <= d'(v).

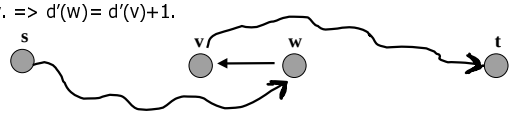
Contradiction.

Lemma: between any two consecutive saturations of (v,w), both d(v) and d(w) increase by at least 2.

In first saturation, increasing flow along augmenting path, and d(v)= d(w)+1.



Before edge can be saturated again, must send flow back from w to v. => d'(w)= d'(v)+1.



In next saturation, again have d''(v)=d''(w)+ 1.

=> d''(v)= d''(w)+1 >= d'(w)+1 >= d'(v)+2 >= d(v)+2.

=> Running time of Edmonds-Karp is $O(m^2n)$

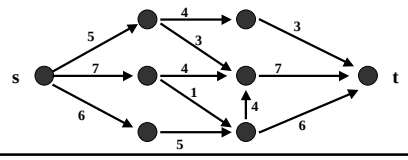
- Corollary 1: An edge (v,w) can be saturated at most O(n) times.
- Corollary 2: The total # of flow augmentations performed is O(mn)
- Since each augmentation can be performed in O(m) time => $O(m^2n)$ time.
- Can be improved to nearly O(mn) with super-fancy data structures.

Fastest max-flow algorithms: preflow-push

- work on one vertex at a time, looking only at neighbors in R.
- maintain preflow: flow except that there may be excess flow into u.
- imagine vertices as pipe junctions. each vertex has outflow pipe leading to arbitrarily large reservoir that can accumulate fluid.
- each vertex, its reservoir, and all its pipe connections are on a platform whose height increases as the algorithm progresses.
- can only push flow downhill.
- height of s fixed at |V|, height of t fixed at 0. others start at 0 and increase with time.

Preflow-push algorithms

- if only pipes leaving u that aren't saturated with flow connect to vertices that are on the same level as u or are uphill from u. => **lift** u : increase its height to one more than height of lowest neighbor to which it has unsaturated pipe.
- eventually all flow that can get through has arrived => convert preflow to "legal" flow by sending excess collected in reservoirs of overflowing vertices back to source.

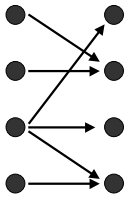


Some applications of max-flow and max-flow min-cut theorem

- Scheduling on uniform parallel machines -- last time
- Bipartite matching
- Network connectivity
- Video on demand

Bipartite Matching

- Input: a bipartite graph G=(V,E)
- Output: Largest-size set of edges M from E such that each vertex in V is incident to at most one edge of S



Network Connectivity

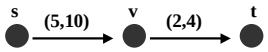
- What is the minimum number of links in the network such that if that many links go down, it is possible for nodes s and t to become disconnected?
- What is the minimum number of nodes in the network such that if that many nodes go down, it is possible for nodes s and t to become disconnected?

Video on Demand

- M storage devices (e.g., disks), each capable of supporting b simultaneous streams.
- k movies, one copy of each on 3 disks.
- Given set of R movie requests, how would you assign the requests to disks so that no disk is assigned more than b requests and the maximum number of requests is served?

Other network flow problems: 1. With lower bounds on flow.

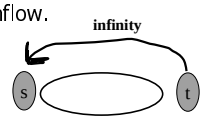
- For each (v,w): 0 <= lb(v,w) <= f(v,w) <= c(v,w)
- Not always possible:



- Most algorithms use 2-phase approach:
  - determine if problem is feasible
  - convert feasible to max-flow.

Determining feasibility

- Transform into a circulation problem <=> outflow of every node, including and t = inflow.



- **Theorem:** A circulation problem with non-negative lower bounds on arc flows is feasible iff for every set S of nodes:

$$\Sigma_{(v,w) \text{ in } (S^c, S)} lb(v,w) = \Sigma_{(v,w) \text{ in } (S, S^c)} c(v,w)$$

Other network flow problems: 2. Minimum flow

- Want to send minimum amount of flow from source to sink, while satisfying certain lower and upper bounds on flow on each edge.
- Example: shipping problem.

Shipping Problem

- A company has to deliver orders between certain origin-destination pairs. Each order has a delivery date. The company wants to determine the minimum number of shipments needed to to meet the delivery dates for the orders.
- M locations, $T_{i,j}$ = travel time between location i and j
- n orders
  - s_i = source location for order i
  - t_i = destination location for order i
  - d_i = delivery date for order i

Other network flow problems: 3. Min-cost max-flow

- Input description: a graph (network) G where each edge (v,w) has associated capacity c(v,w), and **a cost *cost*(v,w)**.
- Problem description: What is the maximum flow **of minimum cost** you can route from s to t while respecting the capacity constraint of each edge?

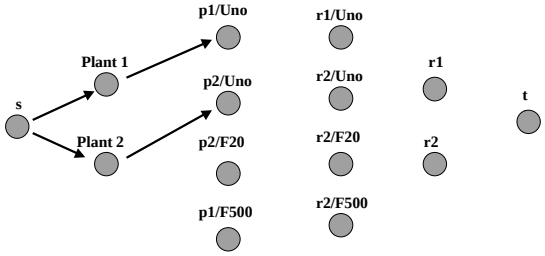
The cost of a flow :

$$\Sigma_{f(v,w)>0} \mathit{cost}(v,w) f(v,w)$$

Classical application: Transportation Problem

- Firm has p plants with known supplies, q warehouses with known demands.
- Want to identify flow that satisfies the demands at warehouses from available supplies at plants and minimizes shipping costs.
- Min cost flow yields an optimal production and shipping schedule.

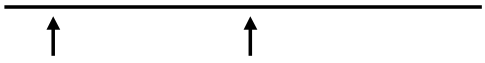
Example: Fiat makes Uno and Ferrari's.



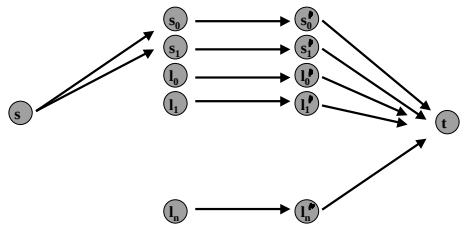
cost	0	production cost	shipping cost	0	0
cap	supply of plant	max production	capacity imposed by dist'n channel	model demand	Retailer demand

Disk head scheduling

- Have disk with 2 heads, and sequence of requests to read data on disk at locations $l_1,\ldots,l_n$.
- How to schedule movement of disk heads so as to minimize total head movement?



Set it up as min-cost max-flow.



New Topic: Linear Programming

- The process of minimizing a linear objective function subject to a finite number of linear equality and inequality constraints.
- Example applications:
  - airline crew scheduling
  - manufacturing and production planning
  - portfolio selection
  - telecommunications network design
- "Few problems studied in computer science have greater application in the real world."

Big Bucks!

"American Airlines employs more than 25,000 pilots and flight attendants to fly its fleet of over 600 aircraft. Crew cost is over 1.3 billion dollars per year is second only to fuel cost. In its effort to better utilize crew resources AA spent about 6000 hours of CPU time during 1989-90 running its crew-pairing code... Estimated savings generated by running this code during the past 5 years are in excess of 20 million dollars per year.

A crew pairing is a sequence of flights that starts and ends at a crew base and typically lasts 2-3 days. A crew member works 4-5 pairings per month. The most important part of efficient crew utilization is making up pairings that cover all flight legs and minimize excess cost. This problem is called crew-pairing optimization."

And more...

Delta Air Lines flies over 2,500 domestic flight legs every day, using about 450 aircraft from 10 different fleets. The fleet assignment problem is to match aircraft to flight legs so that seats are filled with paying passengers. Recent advances in mathematical programming algorithms and computer hardware make it possible to solve optimization problems of this scope for the first time. Delta is the first airline to solve to completion one of the largest and most difficult problems in this industry. Use of the Coldstart model is expected to save Delta Air Lines $300 million over the next three years.

An example: The diet problem

- Trying to decide on lowest cost diet that provides sufficient amount of protein, with two choices:
  - steak: 2 units of protein/pound, $3/pound
  - peanut butter: 1 unit of protein/pound, $2/pound
- In proper diet, need 4 units protein/day.

Goal: minimize 2x + 3y

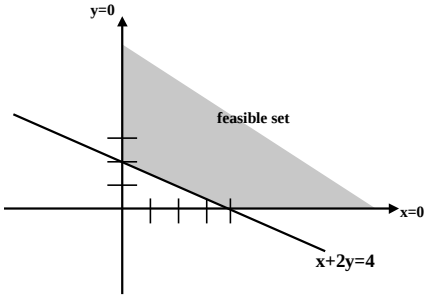
subject to constraints:

x + 2y >= 4

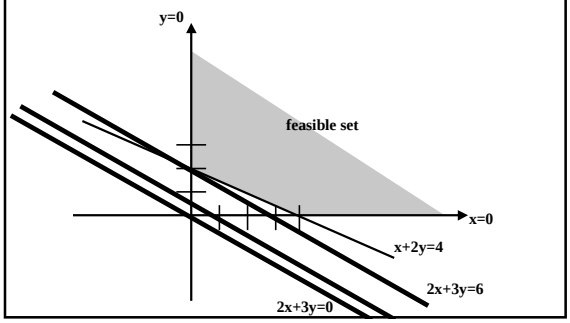
x = # pounds peanut butter/day in optimal diet >=0

y = # pounds steak/day in optimal diet >=0

Visually...
x= peanut butter, y = steak



Optimal vector occurs at corner of feasible set!



General Form of a Linear Program.

Minimize $b_1y_1 + b_2y_2 + \ldots + b_my_m$
subject to $\Sigma_{1<=i<=m} a_{ij}y_i >= c_j$ j=1..n
$y_i >= 0$ i=1..m

or
Maximize $c_1x_1 + c_2x_2 + \ldots + c_nx_n$
subject to $\Sigma_{1<=j<=n} a_{ij}x_j <= b_j$ i=1..m
$x_j >= 0$ j=1..n

The Feasible Set

- Intersection of a set of half-spaces, called a **polyhedron**.
- If it's bounded and nonempty, it's a **polytope**

3 cases:
- feasible set is empty
- cost function is unbounded on feasible set.
- cost has a minimum (or maximum) on feasible set.

First two cases very uncommon for genuine problems in economics and engineering.

The Simplex Method

- Phase I : locate a corner of the feasible set.
- Phase II: move from corner to corner along the edges of the feasible set -- always go along an edge that is guaranteed to decrease the cost.
- When reach a local minimum (maximum), you've found the optimum.

Application: Optimal Pipeline

- A piece of data of size D goes through a pipeline of n stages.
- Each stage has associated
  - o_i -- overhead of i-th stage
  - b_i -- bandwidth of i-th stage (bits/sec)
- How should data be broken up into k pieces so as to minimize time through the pipeline?

Simplex Algorithm: An Example

Maximize 5x + 4y + 3z
subject to 2x+ 3y + z <= 5
4x + y + 2z <= 11
3x + 4y + 2z <= 8
x,y,z >=0.

Step 0: convert inequalities into equalities by introducing slack variables a,b,c.

Define: a = 5-2x-3y-z => a >=0
b = 11-4x-y-2z => b >=0
c = 8-3x-4y-2z => c >=0
F = 5x+4y + 3z, objective function

Example of Simplex Method, continued.

Step 1: Find initial feasible solution:
x=0,y=0,z=0 => a=5, b=11, c=8 => F=0.
Step 2: Find feasible solution with higher value of F
For example, can increase x to get F=5x.

How much can we increase x?
a = 5-2x-3y-z >=0 => x <= 5/2 **most stringent**
b = 11-4x-y-2z >=0 => x <= 11/4
c = 8-3x-4y-2z >=0 => x <= 8/3
=> increase x to 5/2 =>**F= 25/2**, a=0, b=1, c=1/2

Example of Simplex Method, continued.

Want to keep doing this, need to get back into state where x,b,c on l.h.s. of equations.
a = 5-2x-3y-z => x= 5/2 -3/2 y -1/2 z - 1/2 a (*)
Substituting (*) into other equations:
b = 11-4x-y-2z >=0 => b = 1 + 5y + 2a
c = 8-3x-4y-2z >=0 => c = 1/2 + 1/2 y -1/2 z + 3/2 a
F = 5x+4y + 3z => F= 25/2 - 7/2 y + 1/2 z - 5/2 a

In order to increase F again, should increase

Example of Simplex Method, continued.

How much can we increase z?
x= 5/2 -3/2 y -1/2 z - 1/2 a => z <= 5
b = 1 + 5y + 2a => no restriction
c = 1/2 + 1/2 y -1/2 z + 3/2 a => z <= 1 **most stringent (^)**

Setting z = 1 yields
x=2, y=0, z=1, a=0, b = 1, c = 0.
F= 25/2 - 7/2 y + 1/2 z - 5/2 a => F= 13.

Again, construct system of equations.
From (^) z = 1 + y + 3a - 2c.

Example of Simplex Method, continued.

Substituting back into other equations:

z = 1 + y + 3a - 2c.

x= 5/2 -3/2 y -1/2 z - 1/2 a => x = 2-2y-2a + c

b = 1 + 5y + 2a => b= 1 + 5y + 2a

F = 25/2 - 7/2 y + 1/2 z - 5/2 a => F = 13 - 3y -a - c

And we're done.

What were we doing?

- Each time we had a feasible solution we were at a corner == a meeting point of 3 different planes. We chose 3 variables and set them to 0, and made sure remaining constraints were satisfied.
- Then we moved along an edge == a meeting point of 2 different planes obtained from corner by removing one equation. Moved along associated edge until arrived at a new corner.