CSE 417
Network Flows (pt 3)
Modeling with Min Cuts

UNIVERSITY of WASHINGTON

Reminders

- **HW6 is due on Friday**
  - start early
  - bug fixed on line 33 of `OptimalLineup.java`:
    - change “true” to “false”

Review of last two lectures

- Defined the maximum flow problem
  - find the feasible flow of maximum value
  - flow is feasible if it satisfies edge capacity and node balance constraints
- Described the Ford-Fulkerson algorithm
  - starts with a feasible flow (all zeros) and improves it (by augmentations)
  - essentially optimal if max capacity into t is O(1)
  - have not *yet* proven it correct…
- Many, many other algorithms…

Review of last two lectures

for bipartite matching:
multiple matches, group restrictions

for network flows:
node capacities, lower bounds

- Modeling as matching problems
  - covering with dominos: matches adjacent squares, two colors means bipartite
  - token placing: a token is a match between row and column
  - processor scheduling: match programs to time slots
    - number of matches to one time slot limited by number of processors
- Modeling directly with flows
  - escape problem: edge- and node-disjoint paths (latter using node capacities)
  - airline scheduling: flow problem with capacity lower bounds
    - easiest to handle as a flow feasibility problem...
- Many of those graphs have O(1) capacities

Outline for Today

- Equivalent Formulations 
- Max-Flow Min-Cut Theorem
- Network Connectivity
- Project Selection
- Image Segmentation
- Other Min-Cut Algorithms

Flow Feasibility

- **Problem**: Given a graph, two nodes s & t, and edge capacities, determine whether there is a feasible flow of value k.

- Simplified ("decision version") of max flow.
  - rather than finding the maximum value, just determine if it is at least k

Flow Feasibility with Lower Bounds

- Can generalize this to allow lower bounds…

- **Problem**: Given a graph, two nodes s & t, and for each edge e, lower and upper bounds ($0 \leq l_e \leq u_e$), determine whether there is a feasible flow of value k.
  - now feasibility means:
    - edge constraints: $l_e \leq f_e \leq u_e$ for each edge e
    - node balance constraints

Flow Feasibility with Lower Bounds and Demands

- Can generalize this to allow arbitrary flow demands...
  - i.e., multiple sources and sinks

- **Problem**: Given a graph, a demand d_u for each node u, and for each edge e, lower and upper bounds ($0 \le l_e \le u_e$), determine whether there is a feasible flow.
  - no s & t or k
  - now feasibility means:
    - edge constraints: $l_e \le f_e \le u_e$ for each edge e
    - node balance constraints: excess at u = d_u

Flow Feasibility with Lower Bounds and Demands

- **Problem**: Given a graph, a demand d_u for each node u, and for each edge e, lower and upper bounds ($0 \le l_e \le u_e$), determine whether there is a feasible flow.

- Pure generalization of previous problem:
  - choose $d_u = 0$ for all $u \neq s, t$
    - constraint that excess at $u = d_u$ matches original node balance constraints
  - choose $d_t = k$ and $d_s = -k$
    - constraint that excess at $t = d_t$ means flow value is k

Flow Feasibility with Demands but without Lower Bounds

- Lower bounds do not make the problem any harder…
  - we can easily remove them
- For any feasible flow f, consider f′ = f – l instead
  - i.e., $f'_e = f_e - l_e$ (or equivalently, $f_e = l_e + f'_e$)
  - f′ is the flow amount *above* the lower bound
- Need to reduce to a problem involving f′ only…
  - if we can do so, we have removed the lower bounds
    - f′ is only constrained by $f' \geq 0$

Flow Feasibility with Demands but without Lower Bounds

- For any feasible flow f, consider f′ = f – l instead
  - i.e., $f'_e = f_e - l_e$ (or equivalently, $f_e = l_e + f'_e$)
  - f′ is the flow amount *above* the lower bound

- What constraints on f′ tell us that f would be feasible?
  - capacity: subtract l_e from $l_e \leq f_e \leq u_e$ to get $0 \leq f'_e \leq u_e - l_e$
    - have removed the lower bound
  - node demands change…
  - for every edge e = (u,v):
    - need to decrease d_v by l_e to account for removal of l_e flow from edge
    - need to increase d_u by l_e

end up with demands even if none initially…

Flow Feasibility without Demands

- Demands also do not make the problem any harder…
  - we can easily remove them…
    (taking us back to the original feasibility problem)

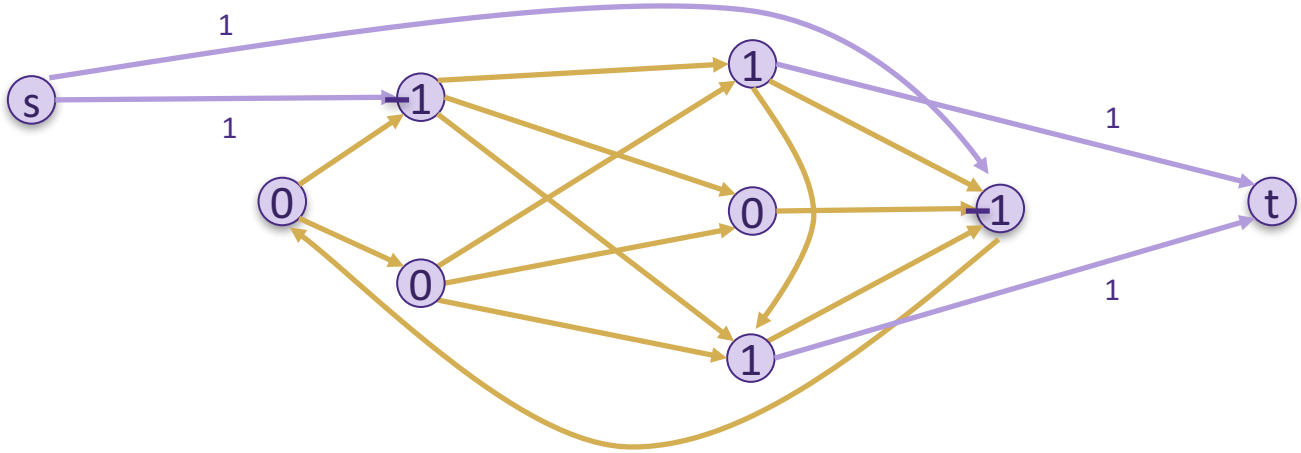
- Recall:
  - this problem has no source & sink but has demands
  - original problem has source & sink but no demands

Flow Feasibility without Demands

- Demands also do not make the problem any harder…
  - we can easily remove them…
- Introduce a new sink node t
  - edge (v,t) to each node v with $d_v > 0$
  - set capacity of this edge to d_v
  - letting excess d_v escape to t means flow is now balanced at v
- Introduce a new source node s
  - edge (s,u) to each node u with $d_u < 0$
  - set capacity of this edge to $-d_u$

Flow Feasibility without Demands

> Demands also do not make the problem any harder…
  – we can easily remove them…



Flow Feasibility without Demands

- Demands also do not make the problem any harder…
- Introduce a new sink node t
  - edge (v,t) to each node v with $d_v > 0$
  - set capacity of this edge to d_v
- Introduce a new source node s
  - edge (s,u) to each node u with $d_u < 0$
  - set capacity of this edge to $-d_u$
- Original problem is feasible iff this has flow saturating t

Maximum Flow with Lower Bounds and Demands

- We showed that flow feasibility is no harder with lower bounds
- What about about maximum flow?
  - flow feasibility (without lower bounds) is no harder than max flow
  - but is it really any easier?
- **Q**: How do we solve max flow using library for flow feasibility?
- **A**: Binary search
- Thus, we can solve max flow with lower bounds too

Maximum Flow with Lower Bounds and Demands

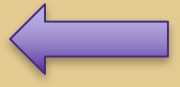
- We showed that flow feasibility is no harder with lower bounds
- Maximum flow is no harder than flow feasibility
  - just gain a log factor from the binary search
  - note: this is only truly polynomial because of binary search
    - a linear search would only be pseudo-polynomial
- This is an important general fact in theoretical CS
  - we will use it again in the theory of NP-completeness
  - there, we will only consider decision problems
    - as we just saw, those aren't any really easier than optimization

Other Transformations

- Have seen:
  - node splitting (for node capacities)
  - removal of lower bounds

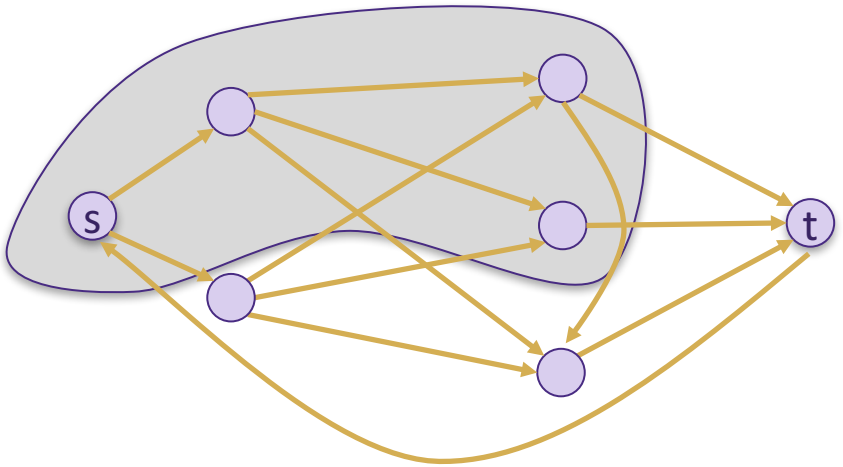
- Other possible transformations:
  - arc reversal (removes with negative capacities)
  - removing arc capacities all together
    - moves capacity constraints into demands / flow balance constraints

Outline for Today

- Equivalent Formulations
- Max-Flow Min-Cut Theorem 
- Network Connectivity
- Project Selection
- Image Segmentation
- Other Min-Cut Algorithms

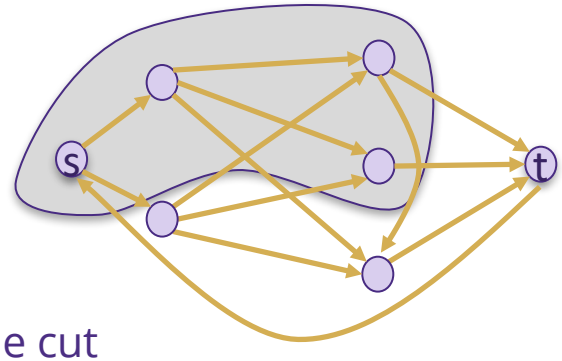
s–t Cuts

> **Definition**: An s-t cut in a graph is any subset of the nodes that includes s but not t.



s–t Cuts

- **Definition**: An s-t cut in a graph is any subset of the nodes that includes s but not t.

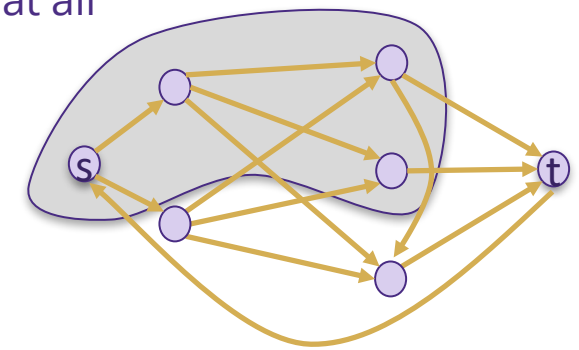


- **Definition**: The capacity of an s-t cut is the sum of all the capacities of all edges *leaving* the cut
  - i.e., edges (u,v) with u in the cut and v not in the cut

Cuts and Flows

- **Proposition**: value of any feasible flow is ≤ capacity of any cut

- **Intuition**: flow from source has to *cross the cut* to get to the sink and the capacity of the cut is the most flow that can cross it.
  - flow across would equal capacity if every outgoing edge was saturated and every incoming edge had no flow at all



Cuts and Flows

- **Proposition**: value of any feasible flow is ≤ capacity of any cut

- Proof:
  - let S = sum of –excess(u) over all u's in the cut
  - this is –excess(s) = value of flow since –excess(u) = u for u ≠ s
  - recall that excess(u) = incoming flow at u – outgoing flow at u
  - sum of –excess(u) adds outgoing flow from every node and subtracts incoming flow at every node
  - every edge (u,v) with u & v in cut cancels out
    - added once and subtracted once
    - leaves only edges with only u in cut or only v in cut…

Cuts and Flows

- **Proposition**: value of any feasible flow is ≤ capacity of any cut

- Proof:
  - let S = sum of –excess(u) over all u's in the cut
  - this is –excess(s) = value of flow since –excess(u) = u for u ≠ s
  - this is sum of outgoing flows from cut minus incoming flows to cut
    - adds flow on (u,v) if u in cut and v not
    - subtracts flow on (v,u)

Cuts and Flows

- **Proposition**: value of any feasible flow is ≤ capacity of any cut

- Proof:
  - let S = sum of –excess(u) over all u's in the cut
  - this is –excess(s) = value of flow since –excess(u) = u for u ≠ s
  - this is sum of outgoing flows from cut minus incoming flows to cut
  - outgoing flows are at most cut capacity
  - incoming flows are at least 0
  - so S = outgoing flows – incoming flows ≤ capacity of the cut

Max-Flow Min-Cut Theorem

- **Theorem**: max value of any flow = min capacity of any cut

- We just saw that we have ≤ for *any* flow and *any* cut
  - in particular, we have ≤ for the max flow and the min cut

- Remains only to show that there is *some* flow whose value equals that of *some* cut…
  - i.e. value of some flow = capacity of some cut,
    which means max value of any flow ≥ min capacity of any cut
  - if x ≤ y and x ≥ y, then x = y

Max-Flow Min-Cut Theorem

- Remains only to show that there is *some* flow whose value equals that of *some* cut…

- Get the flow and the cut from Ford-Fulkerson!
  - in particular, let the cut be all nodes reachable from s in G(f) at termination
    - this includes s and not t
  - as we saw earlier, value of the flow (S) is the sum of the flow on outgoing edges from the cut minus the flow on incoming edges to the cut
  - every outgoing edge (u,v) must be **saturated**
    - otherwise, G(f) would include (u,v) so v would be reachable

Max-Flow Min-Cut Theorem

- Remains only to show that there is *some* flow whose value equals that of *some* cut…

- Get the flow and the cut from Ford-Fulkerson!
  - as we saw earlier, value of the flow (S) is the sum of the flow on outgoing edges from the cut minus the flow on incoming edges to the cut
  - every outgoing edge (u,v) must be saturated
    - otherwise, G(f) would include (u,v) so v would be reachable
  - every incoming edge (v,u) must have **zero flow**
    - otherwise, G(f) would include (u,v) as a backward edge
  - so flow value = sum of outgoing edge capacities = cut capacity

Ford-Fulkerson Correctness

- That completes proof of the theorem, but we also get…

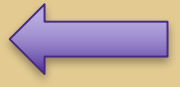
- **Corollary** (of proof): Ford-Fulkerson is correct
  - terminates with a flow whose value is equal to that of some cut
  - BUT we know that every flow has value ≤ that of min cut
  - so this must be the minimum cut
    - if the capacity of the min cut was smaller, our flow value would be bigger (impossible)
  - and since our flow value is = min cut, it is the max flow
    - if the max flow was larger,
      it would be bigger than the min cut (impossible)

Duality (out of scope)

- The fact that value of max flow = capacity of min cut is a special case of linear programming duality
  - LP is the optimization problem of maximizing a linear function subject to a collection of linear inequalities
  - value of the flow is a linear function
  - edge capacity & node balance constraints are linear inequalities
    - (that is, linear in the variables, which are the f_e's)

- (In fact, these are special case of duality in convex programming as well…
  - those have more caveats when "=" holds)

Outline for Today

- Equivalent Formulations
- Max-Flow Min-Cut Theorem
- Network Connectivity 
- Project Selection
- Image Segmentation
- Other Min-Cut Algorithms

Edge Connectivity

- **Problem**: Given a graph and nodes x & y, what is the minimum number of edges that can be removed to disconnect x from y?

- **Application**: how many fiber optic cables would have to fail to remove network connectivity between two Google data centers?

Menger's Theorem

- **Theorem** (Menger, '27): The minimum number of edges that can be removed to disconnect x from y is equal to the maximum number of *edge-disjoint paths* from x to y.

- Proof:
  - apply max-flow min-cut theorem to graph with s = x, y = t, and every edge capacity set to 1
  - max flow is a set of edge disjoint paths
  - any set of edges that disconnects x from y must include one edge from each of these
  - min cut is such a set of edges

Menger's Theorem

- **Theorem** (Menger, '27): The minimum number of edges that can be removed to disconnect x from y is equal to the maximum number of *edge-disjoint paths* from x to y.

- This was actually proven before the max-flow min-cut theorem.

- It also has a vertex connectivity version

Node Connectivity

- **Problem**: Given a graph and nodes x & y, what is the minimum number of nodes that can be removed to disconnect x from y?

- **Application**: how many network switches would have to fail to remove network connectivity between two Google data centers?

Menger's Theorem 2

- **Theorem**: The minimum number of nodes that can be removed to disconnect x from y is equal to the maximum number of *node-disjoint paths* from x to y.

- Proof:
  - apply max-flow min-cut theorem to the graph with node splitting
  - details left as an exercise…

Outline for Today

- Equivalent Formulations
- Max-Flow Min-Cut Theorem
- Network Connectivity
- Project Selection 
- Image Segmentation
- Other Min-Cut Algorithms

Project Selection

- **Problem**: Given a collection of n projects, projected revenue p_u for each project u, and a set of prerequisites {(u,v)}, where the element (u,v) says that project v must *also* be completed if u is, find the set of allowed projects with maximum total revenue.
  - "allowed" means, if u is included and (u,v) is a prerequisite, then v is included

- Note that revenues can be positive or negative
  - may want to include a negative revenue project if it allows you to then include another project with large positive revenue

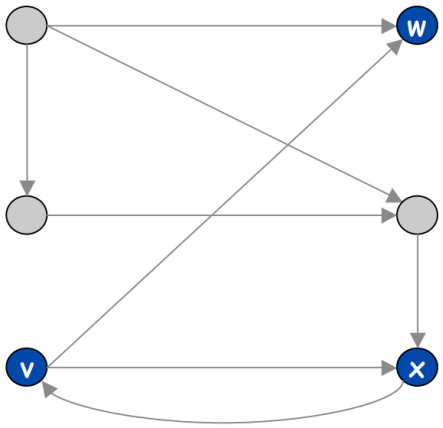
Minimization Problems over Subsets

- Search space is all subsets of the project list
  - obvious brute force will be of little use
  - (we have seen that dynamic programming often works here…)

- Given a problem described that way, see if the values associated with each subset can be created as cuts in a graph
  - if so, then we can find the minimum value by finding the min cut
  - to do so, first get a max flow, then the set of reachable nodes in G(f) gives the set of nodes in the cut
    - (can get that in O(n) time once you've solved the flow problem)
    - **note**: our analysis applies even if non F-F algorithm gives flow!

Project Selection

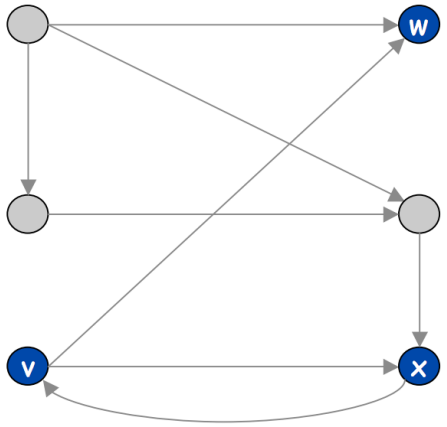
- We already have one graph structure given in the problem
  - the set of prerequisites gives edges in a graph where nodes are projects
    - textbook calls this the “prerequisite graph”



{v,w,x} is allowed

whereas {v,x} would not be

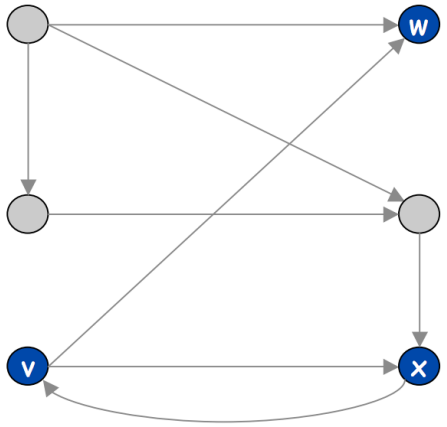
Project Selection



- If we have a prerequisite pair (u,v), we can force the cut to include v when u is in the cut by setting the capacity of the edge (u, v) to **infinity**
  - (flow will be finite as long as there exists some cut with no infinite edges)
  - (choice of infinities to force minimum cut used in node connectivity too)

- The minimum cut will now only include allowed subsets, but we still need to make the cut capacity match the value given in the problem…

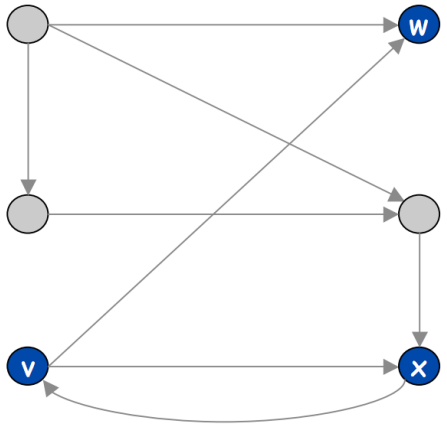
Project Selection



- Note that we are trying to minimize the cut, whereas the problem said to maximize revenue

- Choose the value of the cut to be C – (revenue of included) for some constant C
  - maximizing this is the same as minimizing revenue of included projects
  - we can pick C to be any constant we want
    - (i.e., something independent of what projects are included vs excluded)

Project Selection



- Add a source node s with edges to every project, where (s,u) has capacity max(0, p_u)
  - if u is excluded and has revenue $p_u > 0$, then (s,u) adds p_u into the cut capacity
- Add a target node t with edges from every project, where (u,t) has capacity max(0, $-p_u$)
  - if u is included and has revenue $p_u < 0$, then (u, t) adds $-p_u$ into the cut capacity
- Value of cut = sum of positive excluded + sum of negative included

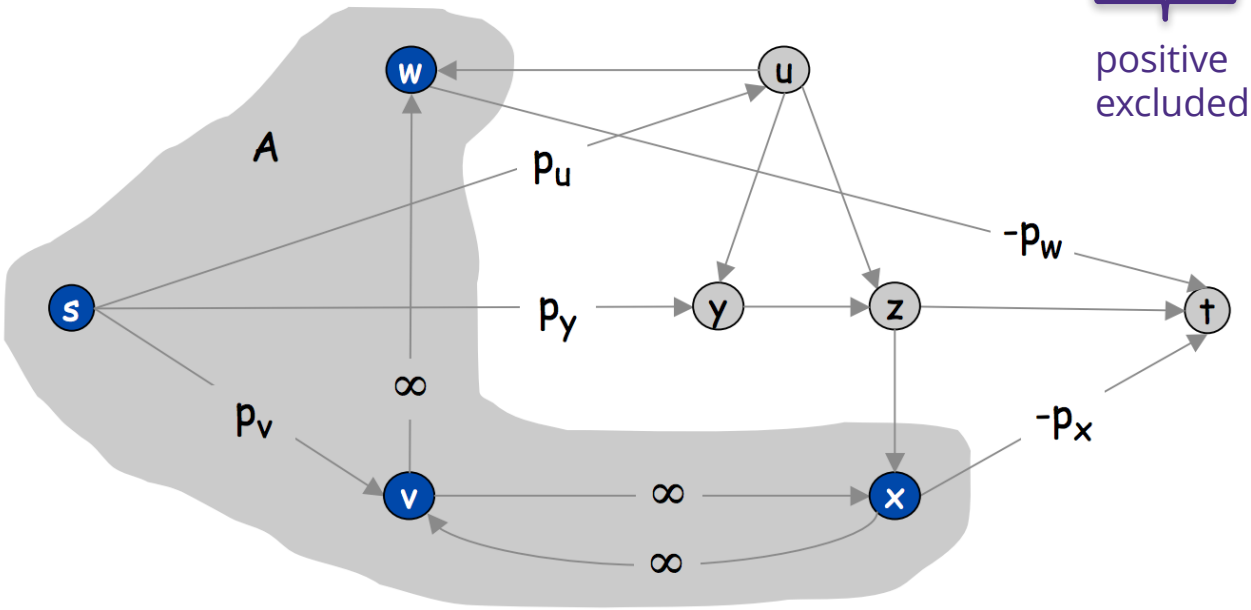
Project Selection

cut value =

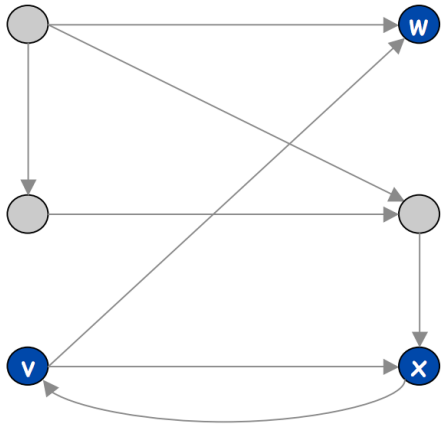
$p_u + p_y - p_w - p_x =$

positive excluded

negative included



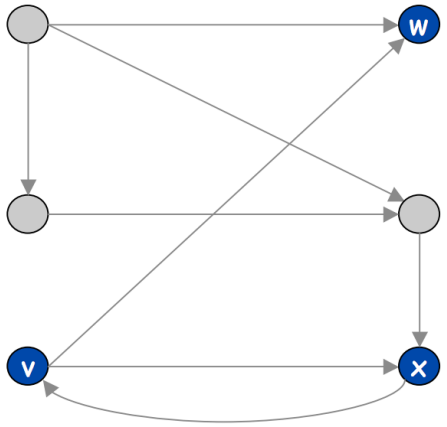
Project Selection



- Value of cut = sum of positive excluded + sum of negative included
- Add and subtract sum of positive included…
- Value of cut = sum of positive excluded + sum of positive included
  – sum of positive included + sum of negative included
  = sum of positive excluded + sum of positive included
  – sum of positive included – sum of -negative included

W

Project Selection



- Value of cut = sum of positive excluded + sum of positive included – sum of positive included – sum of -negative included
  = (sum of all positives) – (sum of included)

- Sum of all positives is a constant
- Minimizing constant – (revenue of included) is equivalent to maximizing revenue of included

Outline for Today

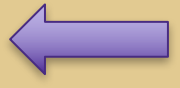
- Equivalent Formulations
- Max-Flow Min-Cut Theorem
- Network Connectivity
- Project Selection
- Image Segmentation 
- Other Min-Cut Algorithms

Image Segmentation

- **Problem**: For each pixel in an image, we are given the probability that it is foreground vs background. We are also given, for each pair of adjacent pixels, the probability that one is foreground and the other is not. Find the separation of pixels into foreground vs background that likelihood.
  - let a_i be the probability that pixel i is foreground and let b_i be the probability that it is background
  - let $p_{i,j}$ be a penalty when pixel i is foreground while j is background
  - likelihood that subset A are foreground and B are back is
    (sum a_i for i in A) + (sum of b_i for i in B) –
    (sum of $p_{i,j}$ for i in A and j in B)

Image Segmentation

from http://www.maths.lth.se/matematiklth/personal/petter/rapporter/graph.pdf

Image Segmentation

- Problem is trying to maximize likelihood, whereas we want to minimize the value of the cut....

- We need to rewrite this as constant – likelihood:
  - choose the constant to be (sum of a_i) + (sum of b_i)
  - then constant – likelihood =
    (sum of a_i for i in B) +
    (sum of b_i for i in A) +
    (sum of $p_{i,j}$ for i in A and j in B)
  - now that every term is positive, it should be easy...

Image Segmentation

> **Intuition**: source connected to those in foreground, and target connected to those in background

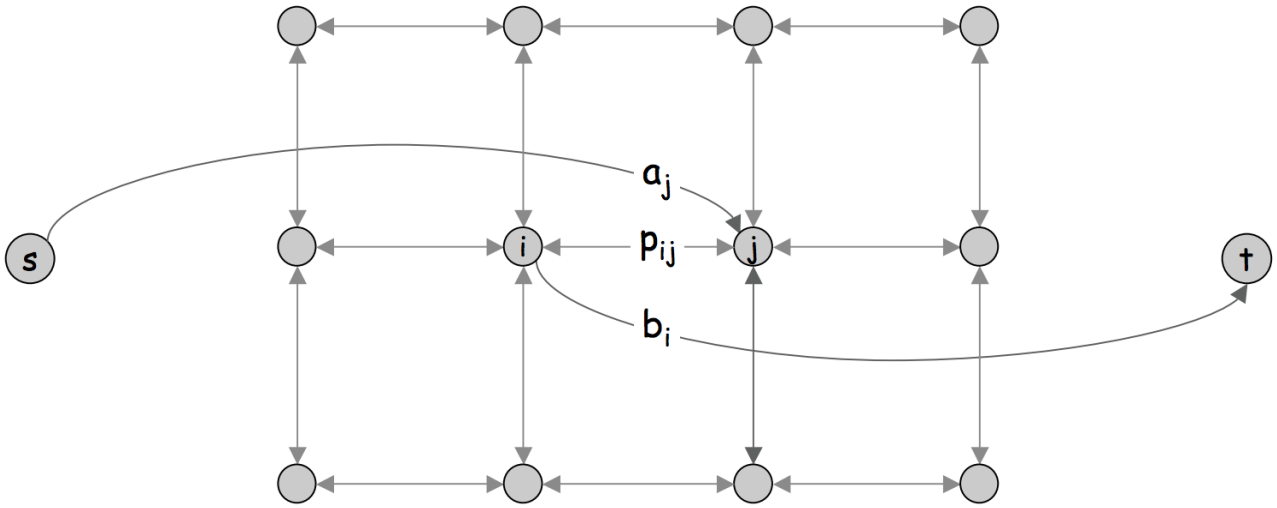


Image Segmentation

> cut capacity = (sum of a_i for i in B) + (sum of b_i for i in A) + (sum of $p_{i,j}$ for i in A and j in B)

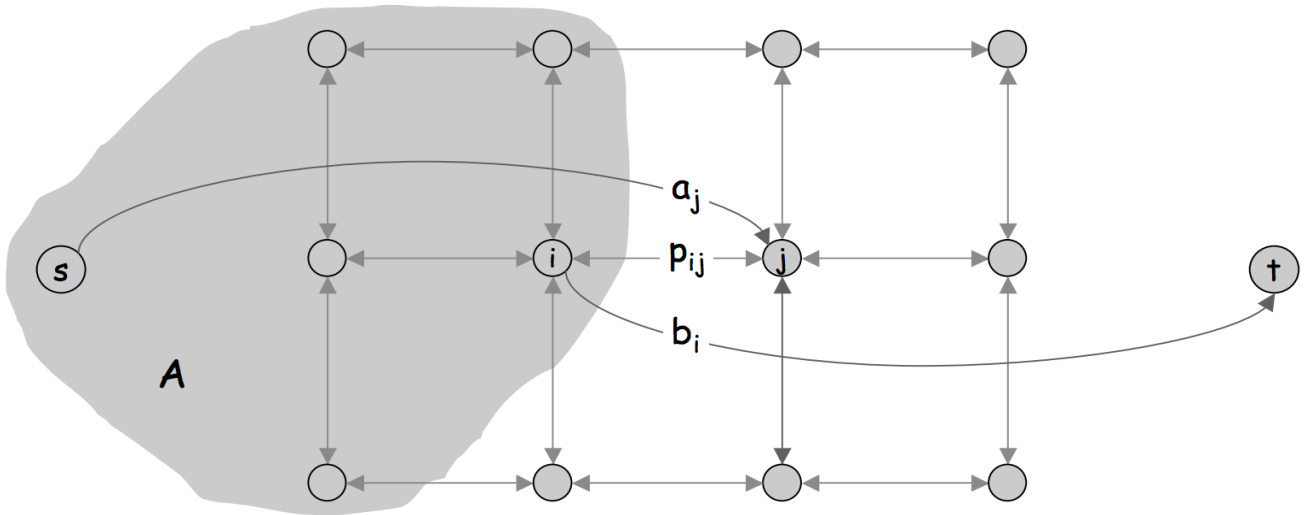
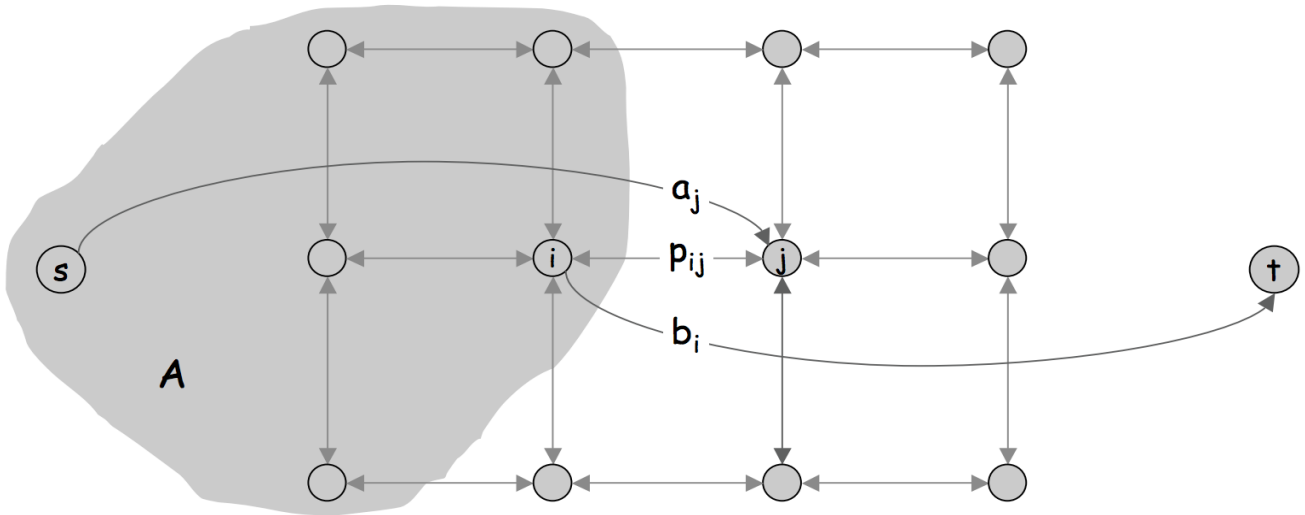


Image Segmentation

> Minimizing the capacity of this cut is maximizing the likelihood of the corresponding (A, B) split.



Outline for Today

- Equivalent Formulations
- Max-Flow Min-Cut Theorem
- Network Connectivity
- Project Selection
- Image Segmentation
- Other Min-Cut Algorithms

Other Min-Cut Algorithms (out of scope)

- The cut perspective also lead to other algorithms...
- Karger–Stein algorithm computes value of min-cut in $O^*(n^2)$ time
  - (ignoring log n factors)
  - the graph is dense, this is faster than flow algorithms by factor of n
- The algorithm is randomized...
  - basic version randomly contracts an edge until 2 vertices remain
    - two nodes at ends of the edge are joined into one
    - edges that become parallel are replaced with sum of capacities
  - the cut is just the capacity of the edge between final vertices

Other Min-Cut Algorithms (out of scope)

- The cut perspective also lead to other algorithms...

- Karger–Stein algorithm computes value of min-cut in $O^*(n^2)$ time
  - (ignoring log n factors)
  - the graph is dense, this is faster than flow algorithms by factor of n

- The algorithm is pretty simple
- Proof of correctness of the algorithm is not simple