MF2, MF3: Max-Flow = Min-Cut, Image Segmentation

Notes for CS-8803-GA: Introduction to Graduate Algorithms

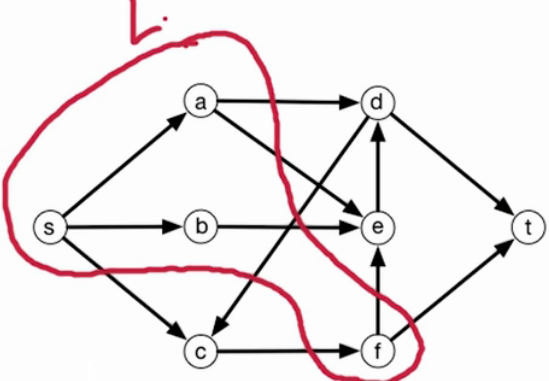
Georgia Tech (Dr. Eric Vigoda), Fall 2017

as recorded by Brent Wagenseller

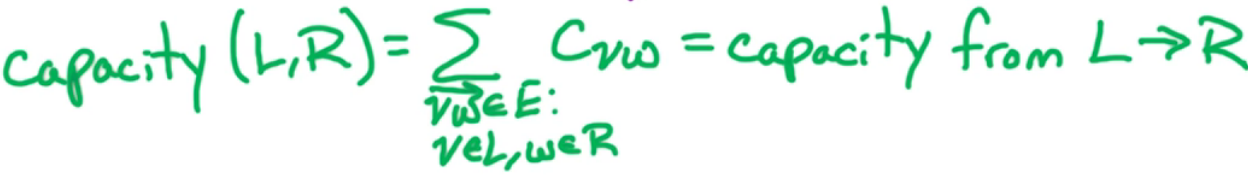
**MF2: Max-Flow Min-Cut**

**Min-Cut Problem**

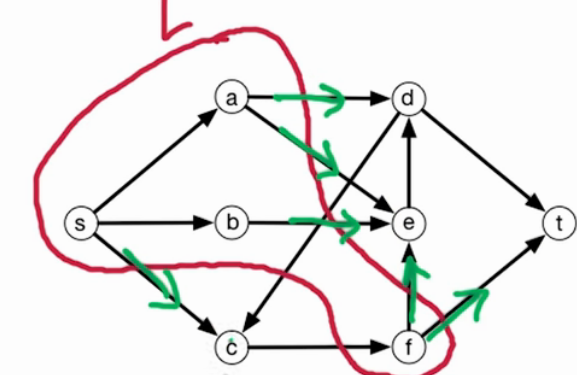
* **Max-flow** problem: given a flow network, find a flow fwith \max\text{size}(f). This is closely related to the following **min-cut** problem.
  + A *cut* is a partition of the vertices into two sets Land Rsuch that V=L\cup Rand L\cap R=\emptyset. This is an s-t cut if s\in Land t\in R. For an s-t cut, its capacity is defined as \text{capacity}(L,R)=\sum_{\overrightarrow{vw}\in E:v\in L, w\in R}c_{vw}=\text{capacity from }L\rightarrow R.
  + **Min st-cut problem**: given a flow network, find an s-t cut with minimum capacity.
* Lemma: For a flow f\* if no augmenting path in Gf\* then f\* is a max-flow
* QUIZ: It takes O(n+m) (Read: O(|V| + |E|)) time to verify whether a given flow f\* is a max-flow
* A **cut** is a partition of V = L ∪ R
  + That is, a partition of vertices into the left side and the right side
* A **st-cut** is a cut where s ∈ L, t ∈ R
  + This separates S from T, where S is in the left cut and T is in the right cut
* An Example s-t cut



* + R is the remaining set (C, D, E, T)
  + Notice that the cut does not have to be comprised of a connected set
    - For example,
      * F is not connected to A or B in set L
      * C is completely cut off from the rest of R
* The *capacity* of the st-cut is of great importance
  + In this case, the capacity is formally:



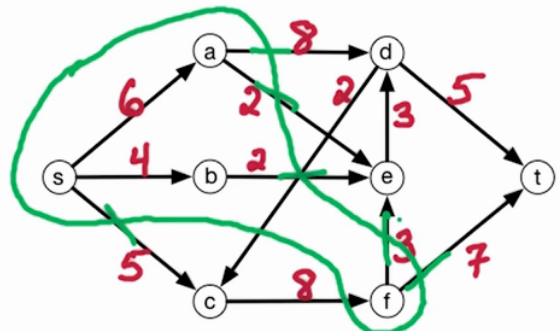
* + - Since we are looking at a directed graph, we are looking at directed edges where v is in the left side and w is in the right side, basically, the capacity for a st-cut is the capacity on the edges that start in the left side and end in the right
      * In other words, look at the capacity where the edge begins in L (when the vertex v is in the left) and ends in R (when the vertex w is in the right)
    - For the graph above, we are looking at 6 edges (in green):



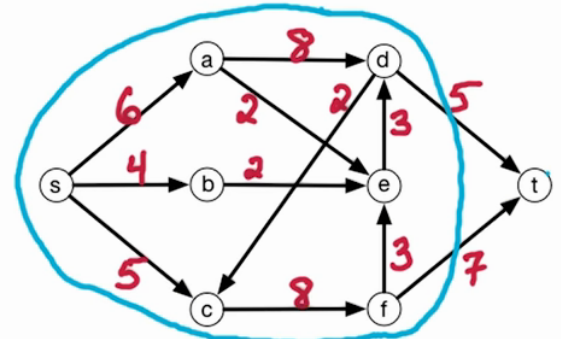
* + - * Notice the edge DF doesn’t count, as technically that’s going from Right to Left
* The min st-cut value (the value of all summed edges, where the starting vertex is in the left cluter and the ending vertex is in the right cluster) IS the max-flow ; THIS is the entire point of min-cut, it tells us the max value of max-flow
* Most of this lecture is proving the following:
  + Theorem: Size of max flow = min capacity of a st-cut
  + Proof hinges on proving that max-flow <= min st-cut while also max-flow >= min st-cut

**Formally Stating Min-Cut Theorem**

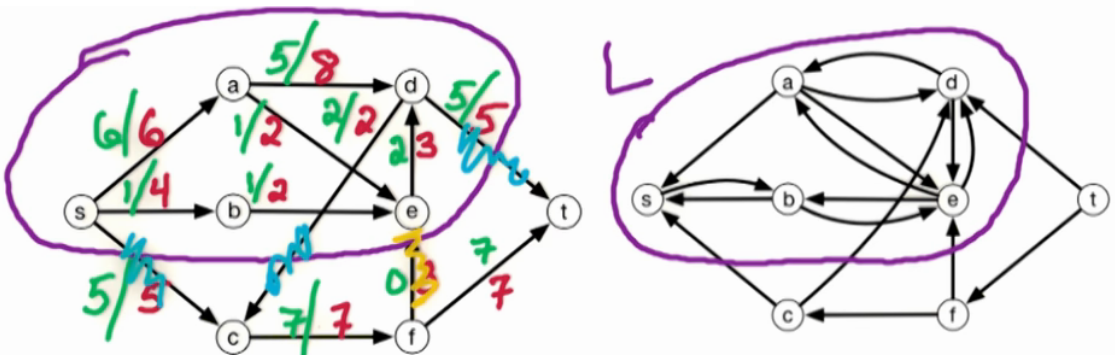
* Min st-Cut Problem:
  + Input: Flow Network
  + Output: st-cut (L, R) with min capacity
    - S has to be in L
    - T has to be in R
    - ‘Capacity’ in this sense means minimizing the edges that originate in L and terminate in R
* Note that min-cut and min st-cut are two different things; for our purposes we need to include the ‘st’ as its absolutely required that s is in the left and t is in the right!
* An Example:



* + The capacity of this cut is 8 + 2 + 2 + 5 + 3 + 7 = 27
    - We want to find the cut of minimum capacity
    - Minimum-cut is all about minimizing this; 27 isnt the actual min-cut solution for this graph; its actually….



* + - * This is the min-cut solution for this graph
        + Note that it will NOT always turn out to be only T in the R set!
      * **The min st-cut will turn out to be the max-flow; this is a critical point and is central to this lesson**
* Here is the actual st min-cut graph of our example with its residual



* + Notice that the edges in the L cut in the original graph equal the max-flow of the entire graph (max flow = 12)!
  + ALSO note that L is defined by all locations reachable by s
    - This means that to identify the L vertices, simply run the FF algorithm on the graph, run DFS on the output residual graph, ignoring the capacities and starting with vertex s; ccnum = 1 will be all vertices in L, and ccnum > 1 will be all vertices in R

**Dr. Vigoda’s Main Proof**

**Max-flow min-cut theorem**: size of max-flow = min capacity of an s-t cut.

We prove the theorem in two directions. The easy direction is that size of max-flow \lemin capacity of an s-t cut. We’ll show that for any flow fand any s-t cut (L,R), \text{size}(f)\le\text{capacity}(L,R)\ (\ast). Hence \max_f\text{size}(f)\le\min_{(L,R)}\text{capacity}(L,R).

For an s-t cut (L,R), define f^{in}(L)=\sum_{\overrightarrow{uv}\in E: u\not\in L,v\in L}f_{uv}=\text{flow into }L, f^{out}(L)=\sum_{\overrightarrow{vw}\in E: v\in L,w\not\in L}f_{vw}=\text{flow out of }L.

We show that \text{size}(f)=f^{out}(L)-f^{in}(L):

Therefore, for any flow fand any s-t cut (L,R), \text{size}(f)=f^{out}(L)-f^{in}(L)\le f^{out}(L)\le \text{capacity}(L,R), which proves (\ast).

Then we show that \max_f\text{size}(f)\ge\min_{(L,R)}\text{capacity}(L,R). Take the flow f^*produced by the Ford-Fulkerson algorithm. f^*has no augmenting path, which means that there is no s-t path in the residual network G^{f^*}. We’ll construct an s-t cut (L,R)where \text{size}(f^*)=\text{capacity}(L,R)\ (\ast\ast). Hence \max_f\text{size}(f)\ge\min_{(L,R)}\text{capacity}(L,R).

Proof of (\ast\ast):for flow f^*, let Lbe those vertices reachable from sin G^{f^*}. We know that t\not\in Lsince there is no s-t path in G^{f^*}. Let R=V\backslash Land (L,R)is a s-t cut.

Consider an edge from Lto R: for \overrightarrow{vw}\in Ewhere v\in Land w\in R, we must have f_{vw}^*=c_{vw}, otherwise \overrightarrow{vw}in G^{f^*}and wis reachable from s, contradiction.

Consider an edge from Rto L: for \overrightarrow{zy}\in Ewhere z\in Rand y\in L, we must have f_{zy}^*=0, otherwise \overrightarrow{yz}\in G^{f^*}and zis reachable from s, contradiction.

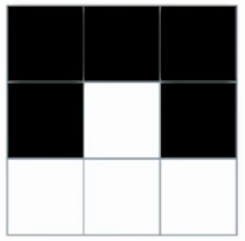
Thus, f^{*out}(L)=\text{capacity}(L,R)and f^{*in}(L)=0. Therefore, \text{size}(f^*)=f^{*out}(L)-f^{*in}(L)=\text{capacity}(L,R), which proves (\ast\ast).

We also know that any flow f^*with no augmenting paths is a max-flow, since \text{size}(f^*)=\min \text{s-t cut capacity}.

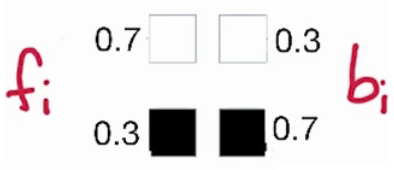
**MF3:**

**Image segmentation**

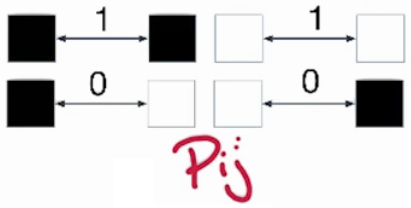
* A common application of maxflow = st min-cut is in the computer vision field of image segmentation
* Four our purposes, we will separate the image between the foreground and background:



* + We will be separating the bottom from the top
* To do this, we will view the image on a graph, where the vertices on the graph represent the pixels of the image
* The input will be an undirected graph G=(V, E) where the vertices are the pixels and the edges are neighboring pixels
* We will also have some new parameters
  + fi will be the likelihood that pixel i is in the foreground
    - this will be nonnegative
  + bi will be the likelihood that pixel i is in the background
    - this will be nonnegative
  + For each (i, j) ∈ E, Pij exists
    - For each pair of neighboring pixels, Pij = separation penalty for separating pixels i and j into different objects
    - Pij will be nonnegative
* Example of fi and bi from above picture



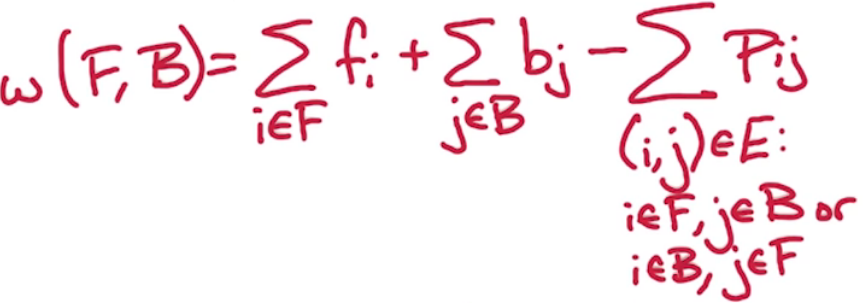
* + This is actually two examples, one white box example and one black box example
* Here are sample separation penalties:



* + When the pixels are different, we don’t pay any penalty for separating the pixels

Partitioning

* We are going to partition the vertices of pixels V into V = F ∪ B
  + Partition into foreground and background
* For some particular partiion (F, B), we need some score / weight of the likelihood of the partition
* We define the weight of the partition as the following



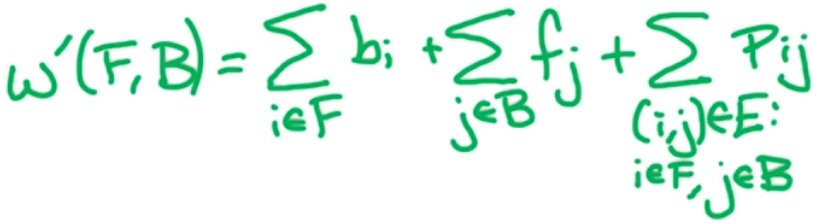
* + For each pixel assigned to the foreground we get fi
  + For each pixel assigned to the background we get bj
  + We pay a separation penalty for separated edges
    - We look at all edges where
      * the first endpoint is in F and the second endpoint is in B -OR-
      * the first endpoint is in B and the second endpoint is in F
* Ultimately, we have a weight associated with a particular assignment of the pixels to the foreground and background
* Goal: Find partition (F, B) with max w(F, B)
  + We can reduce it to min s-t cut problem!

Min-Cut

* Our goal is to reduce the problem to the min st-cut problem
* We have to change the problem from a maximization problem to a minimization problem, and we are going to have to modify the weights to do that
* We also need to change the subtraction of the penalty to an addition, since the min-cut can only work with positive numbers

Reformulation of Problem

* A new weight must be developed because we cant have any negative numbers (aka the subtracted penalty)
* We can play around with the math to get all sums:



* + This is:
    - The sums of the background probabilities for the FOREGROUND vertices
    - The sums of the foreground probabilities for the BACKGROUND vertices
    - The sums of the penalties
  + I wont list the jump from the old weight to this new weight, but if you want to know how it worked see videos 7 and 8 in ‘lesson 19: MF3: Image Segmentation’
    - Intuitively, this works though: this is a sum of the probabilities of being in the background for the foreground pixels, the probabilities of being in the foreground for the background pixels, and the sum of the penalties; this is the sum of EVERYTHING we do NOT want, so it makes sense to try to minimize this sum at all costs.
    - Proof, as worked out by Dr. Vigoda:

First, we convert it from a maximization to a minimization problem. Let Q=\sum_{i\in V}(a_i+b_i)(note that \sum_{i\in A}a_i+\sum_{j\in B}b_j=Q-\sum_{i\in A}(b_i)-\sum_{j\in B}(a_j)). Thus, w(A,B)=Q-\sum_{i\in A}(b_i)-\sum_{j\in B}(a_j)-\sum_{(i,j)\in E: i\in A, j\in B\text{ or }i\in B, j\in A}p_{ij}.

Let w'(A,B)=\sum_{i\in A}(b_i)+\sum_{j\in B}(a_j)+\sum_{(i,j)\in E: i\in A, j\in B\text{ or }i\in B, j\in A}p_{ij}. (A,B)that maximizes w(A,B)is the same as (A,B) which minimizes w'(A,B): w(A,B)=Q-w'(A,B).

To reduce to max-flow: for an edge (i,j)\in E: add edges i\rightarrow jwith capacity p_{ij}and j\rightarrow i with capacity p_{ij}; add source s, for every i\in V,add edges s\rightarrow iwith capacity a_i; add sink t, for every j\in V, add edge j\rightarrow twith capacity b_j.

In this flow network, for an s-t cut (A,B), we ask what edges cross A\rightarrow B? For j\in B, get an edge s\rightarrow jof capacity a_j, for i\in Aget an edge i\rightarrow tof capacity b_i, for (i,j)where i\in A, j\in B, get an edge i\rightarrow jof capacity p_{ij}, and if i\in B, j\in A, get an edge j\rightarrow ialso of capacity p_{ij}.

Therefore, \text{capacity}(A,B)=w'(A,B). So we run max-flow algorithm, the size of the max-flow =\min_{(A,B)}w'(A,B). Thus \max_{(A,B)}w(A,B)=Q-\min_{(A,B)}w'(A,B).

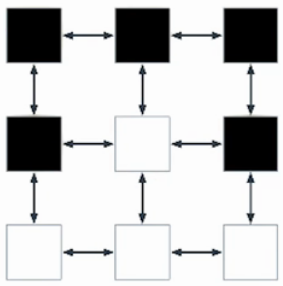
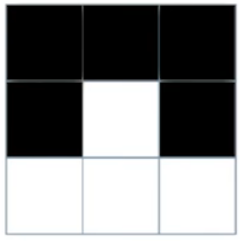
* We now have a new way to measure weights, w`
  + As it turns out, **maximizing the partition w(F, B) is the same thing as minimizing the partition w`(F, B)**
  + We can now use the min st-cut problem
  + We can define a flow network, solve for the max flow for that flow network, and we will then have the min st cut solution!

New Problem Definition

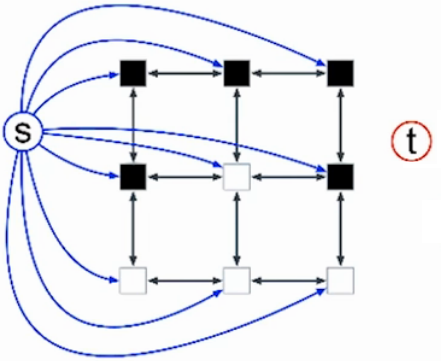
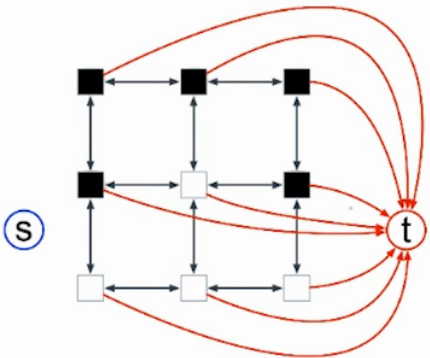
* Input: undirected G=(V, E) with weights
* For each i ∈ V: fi, bi >= 0
  + For each pixel / vertex i, we are given fi and bi and they are nonnegative
* For each (i, j) ∈ E: Pij >= 0
  + For each edge, we are given weight Pij which are also nonnegative
* Goal: find partition V = F ∪ B, and we want the partition which minimizes w`(F, B)
  + Recall this is the SAME partition that maximizes w(F, B)

Flow Network

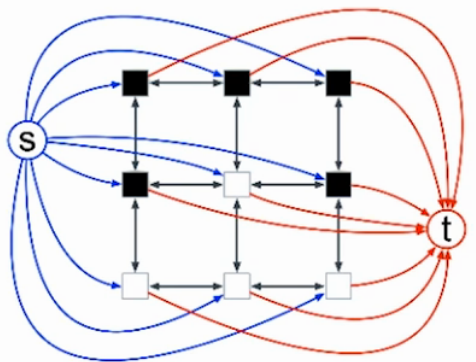
* The current graph for images is an undirected graph, but we need a directed graph for the min-cut solution
* To achieve making this a directed graph, we use bi-directional edges:



* + This is now a directed graph
  + For (i, j) ∈ E: add i → j
    - For each edge in our undirected graph, we add an edge from i to j and j to i
      * The capacity for both edges will be the separation penalty Pij
* Now we add an S and a T
  + S corresponds to the foreground
  + T corresponds to the background
  + There is an edge from S to EVERY vertex
    - The capacity is fi
  + There is an edge from EVERY vertex TO T
    - The capacity is bi
  + Example:

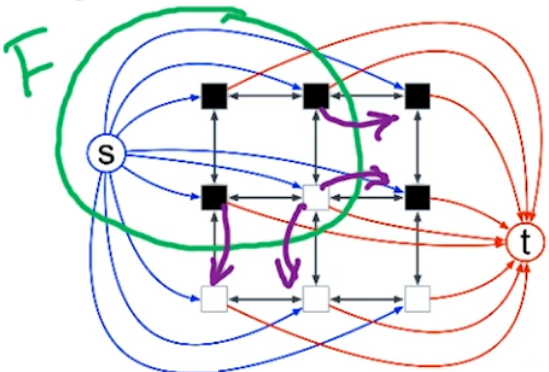
* + - Note that these two images are the SAME graph, just separated visually here so its not too busy
  + We now have the separation penalties, fi, and bi encoded into one graph:



* + - This is now a flow network, and we can use this as an input to the max flow problem

Cuts

* We will run the max-flow algorithm on the above graph
  + We will get a maximum flow, and this will also be the minimum st cut
* For a particular partition (F, B), what is the capacity of that cut?
  + Remember, we are looking for the edges that go from L to R (in our case, F to B); so we are actually looking for the edges that cross from L (aka F) into R (aka B); there are actually THREE sets that do this
    - One set is the red arrows that go directly to t
      * recall that these are the probabilities that a vertex / pixel is a background vertex bi
      * Mathematically: for i ∈ F, get i → t of capacity bi
      * Since only the edges of the foreground will be captured here, there is a low return on investment if we pick these vertices to be in the foreground if they belong in the background
    - Another set are the blue arrows from S to the individual pixels / vertexes;
      * recall this is the probability that a vertex / pixel is a foreground vertex fi
      * Mathematically: for j ∈ B, get s → j of capacity fj
      * Since only the edges of the background pixels will be captured here, there is a low return on investment if we pick these vertices to be in the background if they belong in the foreground
    - The final set are the penalties from the vertices in F to the vertices in B
      * Mathematically: for(i, j) ∈ E: i ∈ F, j ∈ B get i → j of capacity Pij
      * If we perform a min cut on a 0 – the penalty for dividing a true background from a true foreground – that is desired
  + If we sum up these terms, we get the capacity of the cut (F, B), which is w`(F,B)
  + In effect, we have found the min st cut, which is the max flow
  + An example of a (non-finalized) set of vertices F:



* + - The purple arrows are the penalties that are used
* The capacity(F, B) = w`(F, B)
  + Our goal was to minimize w`, and if we run max-flow we can find a min st-cut