LESSON: MF3: Image Segmentation (Optional)
***
(Slide 1) Image Segmentation

Image Segmentation

Given image: separate into objects

Simpler task: foreground & background

In this lecture, we'll look at an application of Max-flow to a problem from computer vision, known as Image Segmentation.

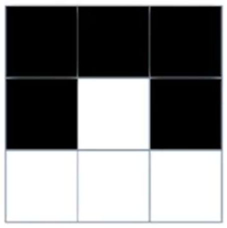
We're given an image, such as this one, and our goal is to separate the image into objects. The easier tests that we're going to look at is to simply separate the image into foreground and background. For example, in this image, we want to discern the triangle from the rest of the image.

***

(Slide 2) Formulation

Formulation

View image on a graph



Input: undirected $G=(V,E)$

V = pixels

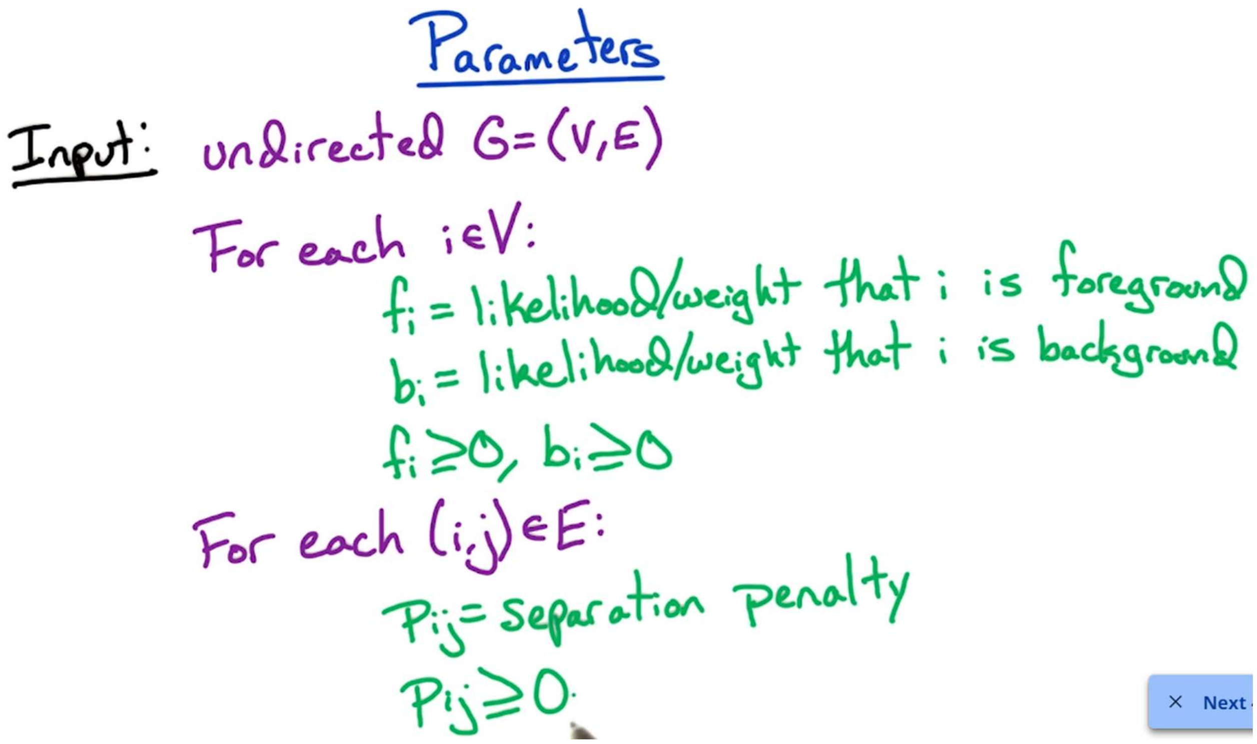
E = neighboring pixels

We are going to view the images as lying on a graph, with the vertices of a graph corresponding to the pixels of the image. So, let us look at an example of a pixelated image.

Here is an example of a small pixelated image magnified. The input is going to be specified by an undirected graph G. The vertices of the graph once again correspond to the pixels of the image. So, there are nine vertices in this example and the edges will go between neighboring pixels.

***

(Slide 3) Parameters



Parameters

Input: undirected $G=(V,E)$

For each $i \in V$:

f_i = likelihood/weight that i is foreground

b_i = likelihood/weight that i is background

$f_i \geq 0$, $b_i \geq 0$

For each $(i,j) \in E$:

P_{ij} = separation penalty

$P_{ij} \geq 0$.

The input once again is specified by an undirected graph G where the vertices correspond to the pixels. And, in addition, we have the following parameters:

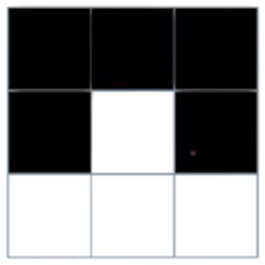
- for each pixel, we have two parameters, fi and bi:
  - fi is the likelihood or weight that vertex i, pixel i is in the foreground.
  - And bi is the weight or likelihood that pixel i is in the background.

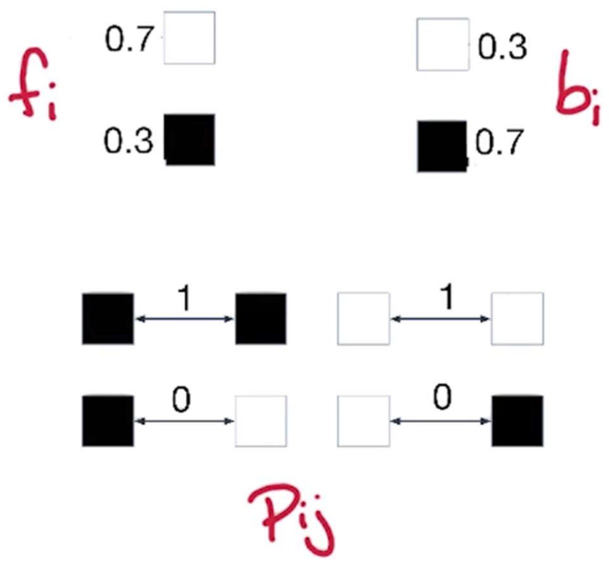
  We'll assume that both of these parameters are non-negative.
- In addition, we're given a parameter for each pair of neighboring pixels:
  - Pij is a separation penalty for this edge. It's the cost of separating i and j into different objects. And once again, we'll assume that Pij is non-negative.

***

(Slide 4) Example

Example





This is our example of an image from before. Let's look at an example setting of the parameters in this case.

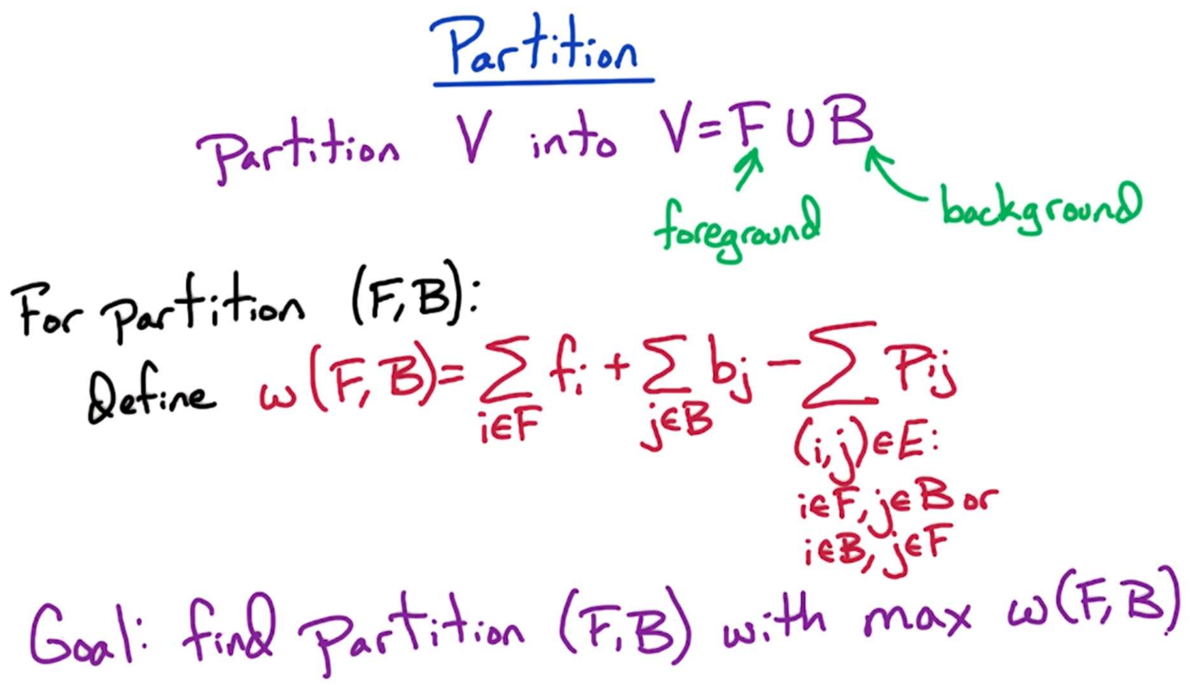
Here are example settings for the foreground and background likelihood for this image. So, we're saying that the white pixels are more likely to be in the foreground and the black pixels are more likely to be in the background. For this image, we could have just as well reversed what we called foreground and background since we've defined the matrix to be symmetric.

And here are sample separation penalties. And when the pixels are different, we don't pay any penalty for separating the pixels into different objects.

Now, keep in mind, this is very much a toy example. All the pixels are monochromatic. So we've chosen very simple weights in this example.

***

(Slide 5) Partition



Partition

Partition V into $V = F \cup B$

foreground

background

For partition (F,B):

Define $$w(F,B) = \sum_{i \in F} f_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E: \\ i \in F, j \in B \text{ or} \\ i \in B, j \in F}} P_{ij}$$

Goal: find Partition (F,B) with max w(F,B)

Our goal is to partition the vertices of pixels into two sets F and B. F will correspond to the foreground pixels and B will be the pixels that we assign to the background.

Now, for a particular partition $V = F \cup B$, we need some score or weight to measure the likelihood of this partition. Thus, we defined the weight of this partition w(F,B) as the following: $w(F,B) = \Sigma fi + \Sigma bj - \Sigma Pij$

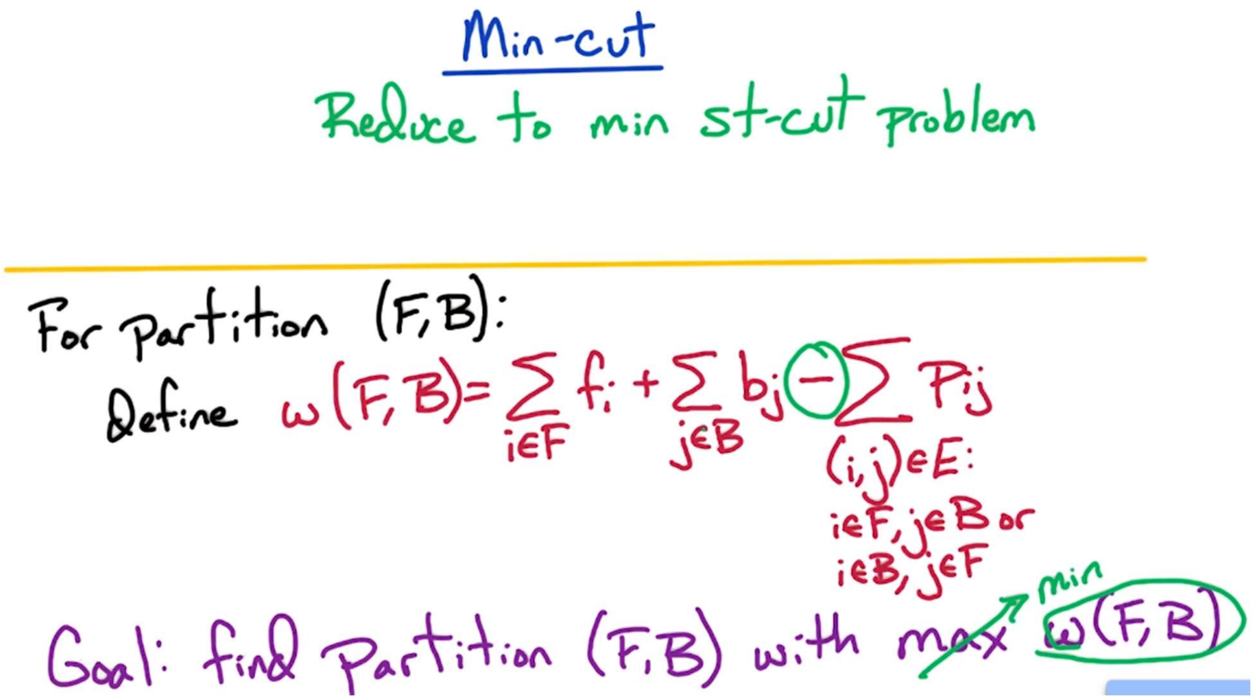
- For each pixel assigned to the foreground, we get Σ fi, $i \in F$
- For each pixel assigned to the background, we get Σ bj, $j \in B$ (… we have used different indices in these summations for these two sets to avoid some confusion).
- Finally, we pay a separation penalty for separated edges. This is a penalty so we're going to subtract it: Σ Pij, $(i,j) \in E$, $i \in F$, $j \in B$. We're going to look at all edges where the first endpoint is in F and the other end point is in B or vice versa. And, the penalty is of course Pij.

This defines the weight for a particular partition (F,B).

So, for a particular assignment of the pixels to foreground and background, we have a weight associated with it. Our goal is to find the partition or assignment of pixels to foreground and background with maximum weight.

***

(Slide 6) Min-Cut



Min-cut

Reduce to min st-cut problem

For partition (F,B):

Define $w(F,B) = \sum_{i \in F} f_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E: \\ i \in F, j \in B \text{ or} \\ i \in B, j \in F}} p_{ij}$

Goal: find partition (F,B) with ~~max~~ min $w(F,B)$

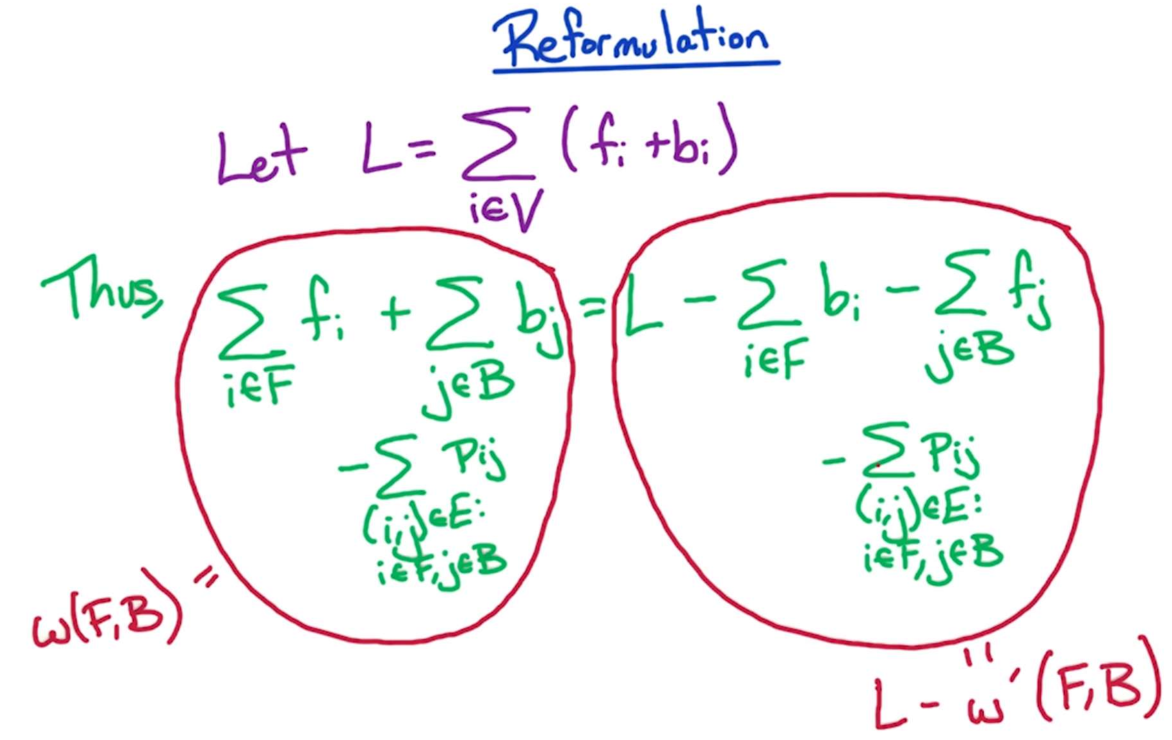
Now, our goal is to reduce this problem, this maximization problem, to the Min st-cut problem.

So we have to somehow change this maximization problem into a minimization problem. So we are going to have to modify these weights.

And, the other issue is that we are summing some terms and we are subtracting other terms (w(F,B) = Σ fi + Σ bj - Σ Pij) Now all of these parameters are non-negative. In our Min st-cut problem, all the capacities in the input flow network are all positive. So, we somehow need to change this minus of the separation penalty into an addition, so that all the terms are positive or non-negative.

***

(Slide 7) Reformulation



Reformulation

Let $L = \sum_{i\in V} (f_i + b_i)$

Thus, $$\underbrace{\sum_{i\in F} f_i + \sum_{j\in B} b_j - \sum_{\substack{(i,j)\in E:\\ i\in F, j\in B}} p_{ij}}_{\omega(F,B)} = \underbrace{L - \sum_{i\in F} b_i - \sum_{j\in B} f_j - \sum_{\substack{(i,j)\in E:\\ i\in F, j\in B}} p_{ij}}_{L-\omega'(F,B)}$$

Look at the sum of all the foreground and background likelihoods. So, let capital L denote the sum over all pixels of their foreground likelihood plus their background likelihood:
L = Σ (fi + bi)

We color our weight for a particular partition (F,B) at weight fi for each pixel assigned to the foreground, and bj for each pixel assigned to the background. Notice that this quantity is equal to L minus the sum of the pixels assigned to the foreground of their background (in other words, Σ bi, i∈F), minus the sum of all the pixels assigned to the background of their foreground (Σ fj, i∈B). Because, if we take the sum over the pixels assigned to the foreground of their foreground likelihood (Σ fi, i∈F), plus sum of the pixels assigned to the background of their foreground likelihood (Σ fj, j∈B), we get the total likelihood of all pixels for the foreground (Σ fi, i∈V). This is the first term in this sum ((Σ (fi + bi), i∈V).

Similarly, by summing this term (Σ bj: j∈B) with this term (Σ bi: i∈F), we get this total likelihood of all pixels for the background (Σ bi: i∈V).

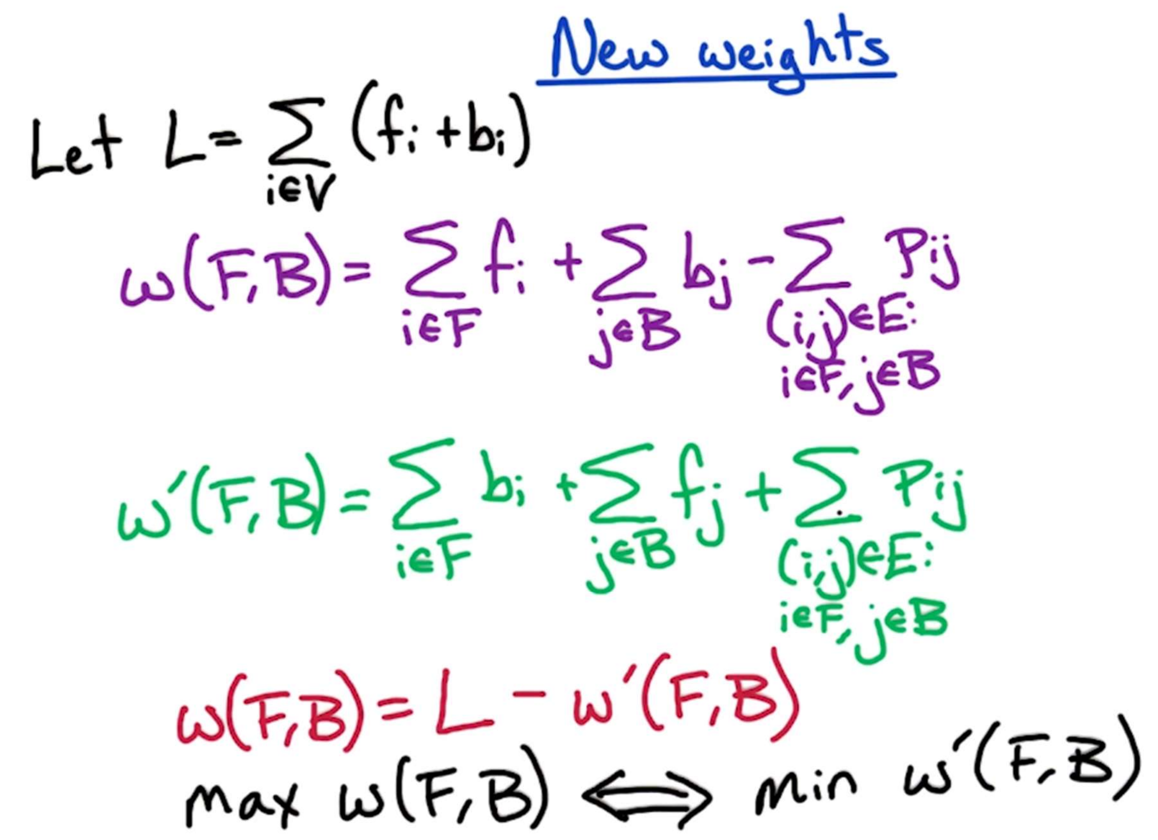
Now, we subtract a separation penalty for separated edges (Σ Pij: (i,j)∈E,i∈F,j∈B). Think of this ij as an un-ordered pair - I don't have to write both cases (i ∈ F or j ∈ B), since this un-ordered pair covers the other case (where i ∈ B and j ∈ F).

To maintain equality, we want to subtract this penalty also from the other side. Now, what do you notice about the left hand side of the equation? Well, this is exactly our definition of the weight, w(F,B).

Now, we're going to define a new weight w'(F,B), which is the sum of these three terms, and then the right hand side will be equal to L minus w'(F,B).

***

(Slide 8) New Weights



New weights

$$\text{Let } L = \sum_{i \in V} (f_i + b_i)$$

$$w(F,B) = \sum_{i \in F} f_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E: \\ i \in F, j \in B}} p_{ij}$$

$$w'(F,B) = \sum_{i \in F} b_i + \sum_{j \in B} f_j + \sum_{\substack{(i,j) \in E: \\ i \in F, j \in B}} p_{ij}$$

$$w(F,B) = L - w'(F,B)$$

$$\max w(F,B) \Longleftrightarrow \min w'(F,B)$$

Now, just to summarize, we let L be the sum of the foreground and background likelihoods for all pixels. For a partition (F, B), we define its weight to be the following quantity: it's a sum over the vertices assigned to the foreground of their foreground likelihood plus the pixels assigned to the background of their background likelihood minus the separation penalty for separated edges.

Now let's define a new weight w′. This is the sum over vertices assigned to the foreground of their background likelihood plus the sum over vertices assigned to the background of their foreground likelihood. And now we'll add instead of subtracting the separation penalty.

And, the key point is that these two quantities are related in the following manner: w(F,B) = L – w′(F,B). (The separation penalty's cancel from both sides and then when you add up these remaining terms you get L.)

Now the point is if we find the partition (F, B) which maximizes the weight w(F,B), this is equivalent to finding the partition (F,B) which minimizes m' - since, we're subtracting off w' minimizing this term is the same as maximizing the whole quantity.

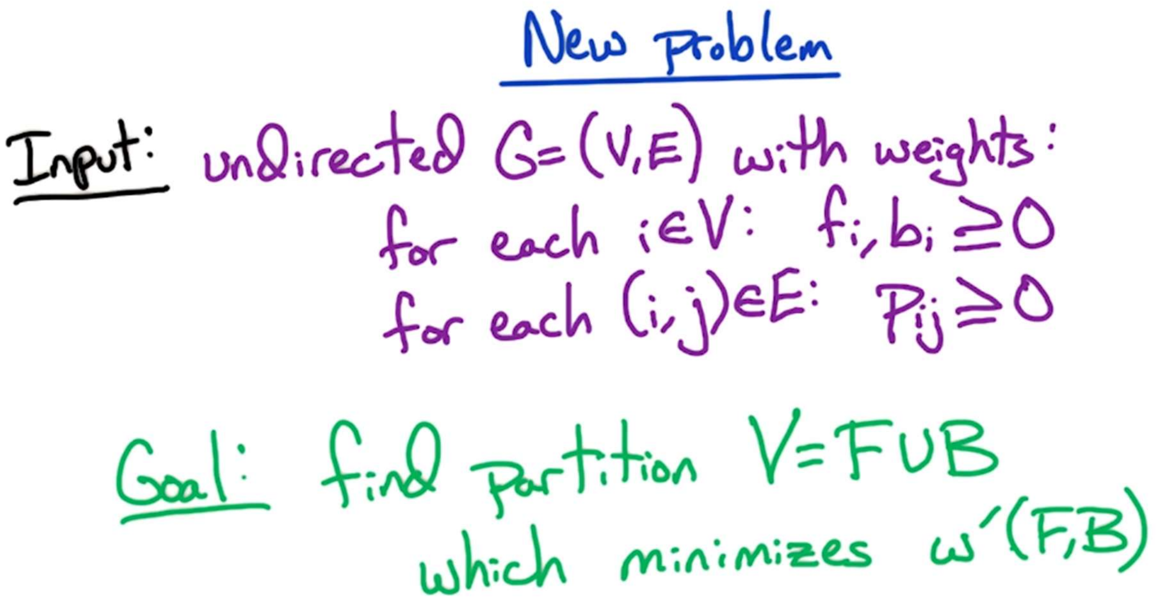
So we've changed our maximization problem into a minimization problem and all the terms and w′ are positive. So, now this is in potential form that we can convert it to a Min st-cut problem.

And, how do we solve the Min st-cut problem? We solve it by doing the Max-flow problem.

So, we're going to define a flow network, solve the Max-flow on that flow network and then that will give us the Min st-cut solution.

***

(Slide 9) New Problem



New problem

Input: undirected $G=(V,E)$ with weights:

for each $i \in V$: $f_i, b_i \geq 0$

for each $(i,j) \in E$: $P_{ij} \geq 0$

Goal: find partition $V = F \cup B$

which minimizes $w'(F,B)$

Here is our new problem formulation:

> The input is an undirected graph corresponding to the image, once again, and we're given the following weights. For each pixel, for each vertex I, we're given fi and bi and these are non-negative weights. And, for each edge, we are given the weight Pij, which is also non-negative. And, our goal is to find the partition of the vertices or pixels into two sets, F and B, and we want the partition which minimizes the weight, w'(F,B).

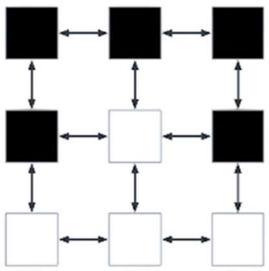
Recall that the partition (F,B), which minimizes w'(F,B), is the same partition which maximizes w(F,B) and therefore it solves the original image segmentation problem.

Now, let's see how to reduce this to the max flow problem.

***
(Slide 10)
Flow Network

Flow network

undirected $G=(V,E) \rightarrow$ directed $G'=(V',E')$



For $(i,j)\in E$: add $i\rightarrow j$, $j\rightarrow i$ with capacities p_{ij}

To convert this new image segmentation problem into a Max-flow problem, we define the flow network which is the input to our max-flow problem.

Now, the input to the image segmentation problem is undirected graph. And, a flow network is a directed graph. So we have to take this undirected graph G and define a directed graph G'.

Here's our example input to the image segmentation problem. This corresponds to a natural undirected graph which is the corresponding directed graph. Well, here are the vertices of this graph.

Instead of putting undirected edges between neighboring pixels, we'll put bi-directional edges. So, these bi-directional arrows denote edges going from this pixel to this pixel and vice versa. This pixel back to this pixel. So this is the directed graph corresponding to our undirected image.

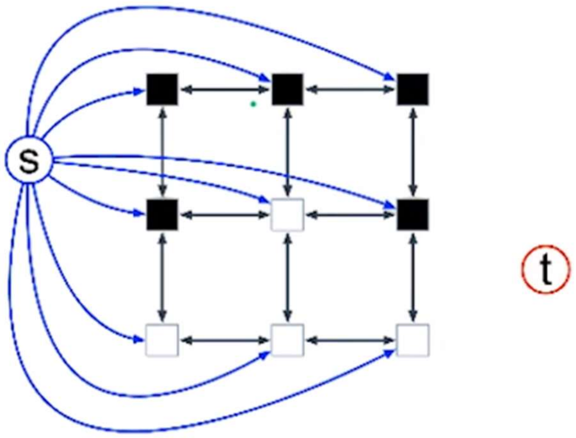
So, for each edge in our undirected graph, we add an edge from i to j and j to i. Now, what are the capacities on these edges? Well, we want to keep track of the separation penalties, so it's natural to put capacities which are the separation penalties. So both of these edges will get capacity Pij. Now, that encodes the separation penalties.

What do we do with the foreground and background likelihoods? - well, we're going to add an additional vertex corresponding to the foreground and an additional vertex corresponding to the background.

***
(Slide 11) Foreground

Foreground

For $i \in V$, add $s \to i$ of capacity f_i



So, we're going to add a vertex s, a source vertex, corresponding to the foreground pixels.

Now, here's the same directed graph - once again, with the vertices shrunk a little bit, just for illustrative purposes. Now, we're going to add two additional vertices, s and t:

- s will correspond to the foreground, t will correspond to the background, and this will be our source and our sink for our Max-flow problem.
- We're going to add an edge from s to every pixel, every vertex. So, for every pixel in the original undirected graph, we add an edge from s to i. Now, s is supposed to correspond to the foreground. So, the capacity of this edge will be fi.

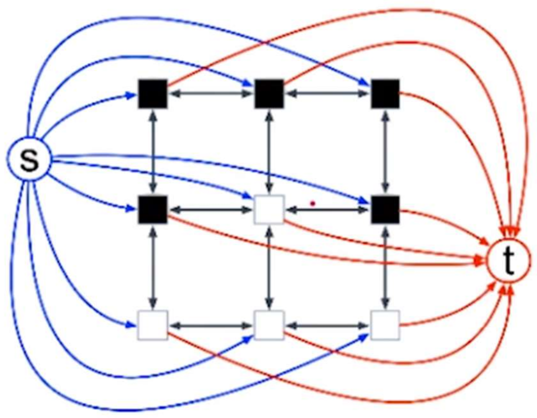
So, we have encoded the foreground likelihoods.

***

(Slide 12) Background

Background

For $i \in V$, add $s \rightarrow i$ of capacity f_i



For $i \in V$, add $i \rightarrow t$ of capacity b_i

Now for the background, we do similar, using this vertex t:

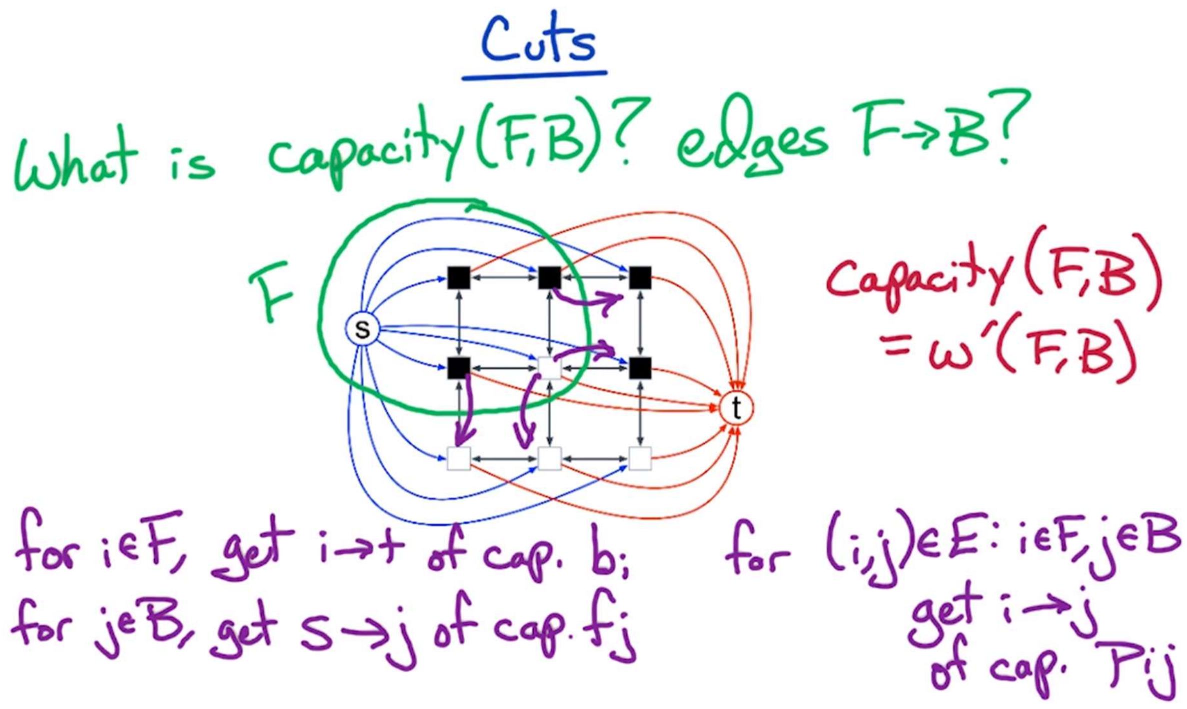
- So, we have an edge from every vertex of the original undirected graph to t. So these edges are going to t, whereas edges came out of S.   So, s is a source and T is a sink,
- and the capacity of these edges to t will be the background likelihood.

So now we've got the foreground likelihood and the background likelihood encoded, and we have the separation penalties encoded.

So here's our final graph, and we've specified the capacities along every edge. Therefore, we've defined a flow network, and we can use this as our input to the Max-flow problem.

***

(Slide 13) Cuts


Cuts
What is capacity(F,B)? edges F→B?
F
s
t
Capacity(F,B) = w′(F,B)
for i∈F, get i→t of cap. b_i
for j∈B, get s→j of cap. f_j
for (i,j)∈E: i∈F, j∈B get i→j of cap. P_{ij}

Now we're going to take this flow network, and we're going to run the Max-flow on it.

We're going to get a flow of maximum size, and we know that the size of the maximum flow equals the capacity the minimum st-cut. Now let's try to understand st-cuts.

For a particular partition (F, B), what is the capacity of that cut? Recall, the capacity of this cut is the capacity of the edges that go from F -> B … so, which edges go from F to B? - recall, we only get these edges that go from F -> B - we don't get the edges that go from B -> F.

So, let's define a partition or cut in this graph. Let's suppose that these 4 pixels get assigned to the foreground. So this is our set F, and the remaining 6 vertices are in the background B.

Now, which edges cross from F to B? Consider one of these four pixels in the foreground. Notice that its edge to t, crosses this cut. So we get this edge from i to t … and, what's the capacity of this edge? - it's bi. That was our definition of this flow network.

Similarly, look at these pixels assigned to the background. For each of these 5 pixels, we get the edge from s to that pixel, and that edge from s originates in F, and ends in B. So it crosses this cut (F, B). And what's the capacity of this edge from s to this pixel? - that was defined to be fj.

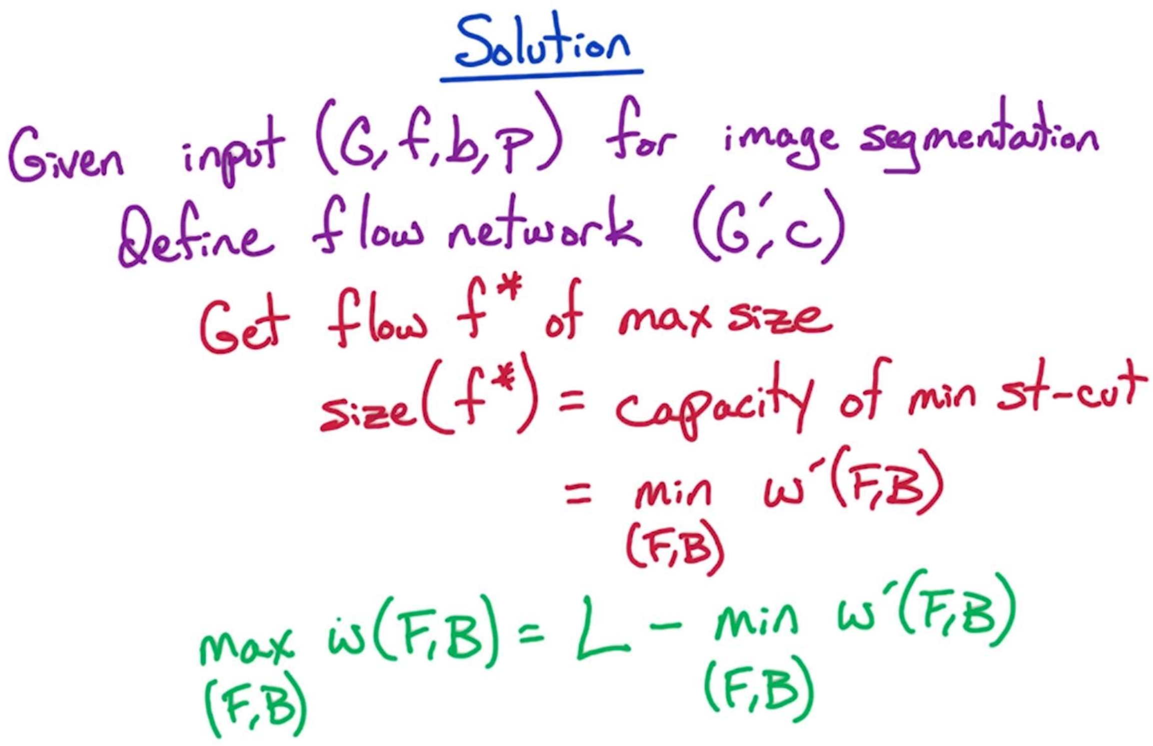
Finally, for each of these separated edges, we get the edge in one direction, but not the other direction. So, for each edge in the original undirected graph, if i is assigned to the foreground, and j is assigned to the background, then we get the edge i -> j, and the capacity is Pij.

Now if you sum up these terms, what do you get? That's the capacity of this cut, (F,B). And that's exactly equal to w'(F,B).   That was our definition of w'. We summed over pixels in the foreground of bi, pixels in the background of fj, and the separation penalty.

That's great! Our capacity is exactly equal to the quantity w', and our goal was to minimize w'. If we run Max-flow, we find a min st-cut. That's a cut with minimum capacity. So, we found the cut, the partition which minimizes w'.

***

(Slide 14) Solution



Solution

Given input (G, f, b, p) for image segmentation

Define flow network (G', c)

Get flow f^* of max size

$$\text{size}(f^*) = \text{capacity of min st-cut}$$

$$= \min_{(F,B)} w'(F,B)$$

$$\max_{(F,B)} w(F,B) = L - \min_{(F,B)} w'(F,B)$$

So, we can summarize our solution to the image segmentation problem.

Given our original input to the image segmentation problem - this consists of an undirected graph corresponding to the image, the foreground likelihoods, background likelihoods, and the separation penalties - then we define a flow network. This consists of a directed graph.

We take this undirected graph, make each edge bi-directional, and then we add s, which directs to every vertex in the original undirected graph. And, we add vertex t, which has an edge from every pixel in the original undirected graph to t. That defines the directed graph, and then we define our capacities.

Then we run Max-flow on this flow network, and we get a flow, f* of maximum size. Now, the size of this max flow equals the capacity of the min st-cut - this is the max flow min-cut theorem.

And, in fact, we can take this max flow and construct a cut of minimum capacity. And, we just saw that the capacity of a particular st-cut equals the weight, w'(F,B).

Therefore, by finding the cut of minimum capacity, it's equivalent to finding the partition of minimum weight w’.

Now, our original problem was to find the partition of maximum weight, w. Well, this is the same as L, this normalizing factor, minus the min over partitions of w’. So, if we find the partition which minimizes w’, that's the same partition which maximizes w. And, therefore, we've shown how to solve the original image segmentation problem using the Max-flow problem.