14 March 2018. Image segmentation (§7.10)

But before we begin image segmentation....

Recapping the conclusion of Bipartite Matching!

In a bipartite matching problem we're given two sets L, R and a bunch of pairs (edges) $(u,v) \in L \times R$. We want to find a matching that covers as many elements of L as possible. What could prevent us from covering <u>every</u> element of L?

E.g., scheduling appointments. $L = \{$ people who want appointments $\}$
$R = \{$ available time slots $\}$.

In general if A_L denotes any subset of L, and $\Gamma(A_L)$ denotes the set of all $v \in R$ that have a neighbor in A_L, then when $|A_L| - |\Gamma(A_L)| = k > 0$ it means that every matching must leave at least k elements of A_L uncovered. Therefore every matching must leave at least k elements of L uncovered.

Consequence of max-flow min-cut from last lecture: a converse to this principle. If for some integer $k \geq 0$, it is <u>impossible</u> to find a set A_L with $|A_L| - |\Gamma(A_L)| > k$ then it must be <u>possible</u> to find a matching that covers all but k elements of L.

<u>Image segmentation</u>. Given an image, segment its set of pixels into foreground (A) and background (B).

Model this as an optimization problem: pixels form the vertex set of a grid graph. (Vertices of graph are adjacent if their pixels have equal x-coord, y-coord differ by 1; or vice-versa.)

For every pixel u, we have two values

$a_u :=$ value of putting u in the foreground

(larger value means better to put u in foreground)

$b_u :=$ value putting u in the background.

(E.g. blue pixels might have high b_u, low a_u, because they probably represent the sky.)

For every pair of adjacent pixels, there's a penalty

$\underset{= p_{vu}}{p_{uv}} :=$ cost of putting $u \in A$, $v \in B$ or vice versa.

Partitioning objective: find a partition of pixels into A, B maximizing

$$q(A,B) \stackrel{\Delta}{=} \sum_{u \in A} a_u + \sum_{u \in B} b_u - \sum_{u \in A} \sum_{\substack{v \in B \\ v \sim u}} p_{uv}$$

"v adjacent to u"

Looks like min cut except:

1. Maximization instead of minimization,
2. a_u, b_v terms which depend on vertices, not edges.
3. Grid graph is undirected, has no source or sink.

Define $Q := \sum_{u \in V} (a_u + b_u)$ then

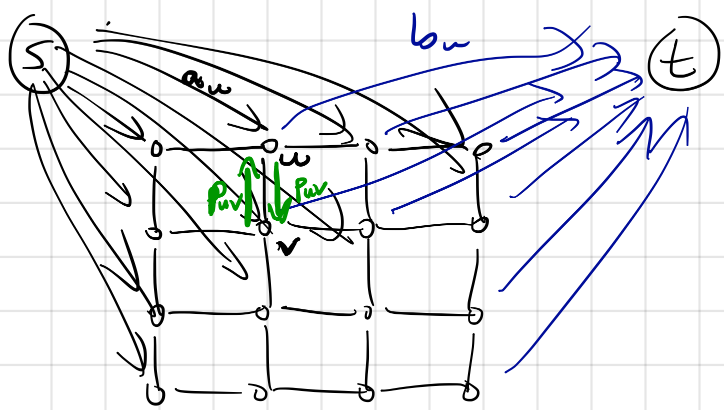
$$q(A,B) = Q - \sum_{u \in A} b_u - \sum_{u \in B} a_u - \sum_{u \in A} \sum_{\substack{v \in B \\ v \sim u}} p_{uv}$$

So maximizing $q(A,B)$ is equiv't to minimizing

$$q'(A,B) = \sum_{u \in A} b_u + \sum_{u \in B} a_u + \sum_{u \in A} \sum_{\substack{v \in B \\ v \sim u}} p_{uv}$$

$$q'(A,B) = \sum_{u \in A} b_u + \sum_{u \in B} a_u + \sum_{u \in A} \sum_{\substack{v \in B \\ v \sim u}} p_{uv}$$

Making q' into a graph cut function.
Add nodes s, t to the grid.



<u>Lemma.</u> For any partition of this flow network into sets $\{s\} \cup A$ and $\{t\} \cup B$, the cut capacity satisfies

$$c\left(\{s\} \cup A,\ \{t\} \cup B\right) = q'(A, B)$$