CSE 347 Lecture 12

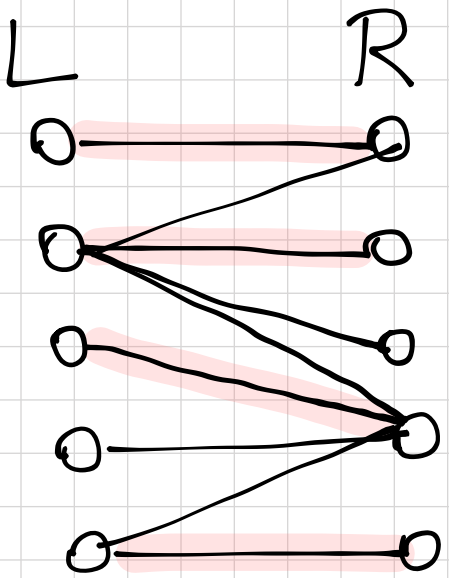
Reductions I: Bipartite Matching and More

1. Bipartite Matchings from Max. Flow
2. Image Segmentation from Min. Cut

Over the last two lectures, we invested quite some work in developing an algorithm for the Maximum Flow problem. We already saw that this algorithm solves the Minimum s-t Cut problem, and indeed, that these problems are closely related to one another, in spite of their rather different appearance. Today, we'll see how algorithms for these problems can be used as subroutines to solve still further problems that, on the surface, seem quite hard. In this sense, algorithms for the Max. Flow problem are thought of as being "powerful."

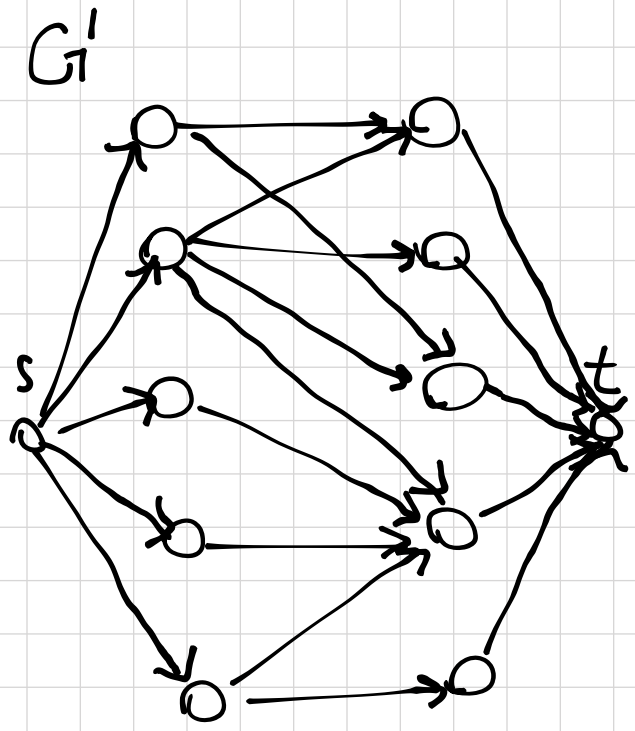
The Bipartite Matching Problem

A graph $G=(V,E)$ is said to be bipartite if the vertex set V can be partitioned into disjoint sets $L \& R$ such that every edge has one endpoint in L and the other endpoint in R.



A matching M is a subset of the edges such that no vertex is an endpoint of more than one edge in M. The Maximum Bipartite Matching

problem is to find a matching M of the largest possible size. This problem arises naturally when we wish to assign, for example, jobs to candidates, with the restriction that each candidate can perform at most one job, and is qualified for a subset of the jobs given by the edges in the graph. (The assignment of doctors to residency programs is actually done by a weighted variant of the matching problem that accounts for preferences.) What is a maximum matching in this example graph? (See highlighted edges) Note that we can only match four out of five since the bottom three left vertices can only be matched with the bottom two right vertices, so one must go unmatched.



To use Max-Flow to solve this problem, we must convert it to a directed graph: we orient the edges to point from L to R, and add a "source" vertex s with an edge to every vertex in L, and a "sink" vertex with an edge from every vertex in R. We assign every edge in this new graph G' a capacity of 1. Our algorithm is simply: given a bipartite graph G, construct G', and find an integer

maximum flow f in G' (e.g., using Ford-Fulkerson). Then, return the set of edges $e \in L \times R$ such that $f(e) = 1$.

Lemma 1: If G has a matching of size k, then G' has an (integer) flow of value k.

Proof: Let M be a matching of size k, and consider the following flow f: ① for each $u \in L$ such that some $(u,v) \in M$, put $f(s,u) = 1$, and all other $f(s,u') = 0$. ② for each $(u,v) \in M$, put $f(u,v) = 1$ and for each $(u',v') \in L \times R - M$, put $f(u',v') = 0$ ③ for each v s.t. $(u,v) \in M$, put $f(v,t) = 1$, and put each other $f(v',t) = 0$. Notice, since M has size k, there are exactly k vertices of L with endpoints in M, so this has value k. Each $u \in L$ and $v \in R$ is the endpoint of at most one edge in M, and these vertices have, respectively, one unit of flow arriving on (s,u) by ① and one unit exiting on (v,t) by ③, with one unit crossing (u,v) by ②. All other $(u',v') \notin M$ have $f(u',v') = 0$ by ②, zero entering from s by ①, and zero exiting to t by ③, so the conservation and capacity constraints are satisfied. Thus, we have a flow of value k. ▨

Moreover, we can show that these "intended" flows are the only (integer) flows: Suppose G' has a flow f' of value k, and let M' be the set of edges we return

Lemma 2 M' is a matching.

Proof? First notice that since f' is integer-valued, each edge has $f(e) = 0$ or $f'(e) = 1$. Now, at most one unit of flow enters each $u \in L$ from (s, u), so at most one unit crosses from u to any $v' \in R$. Thus, each $u \in L$ is the tail of at most one edge in M'. Similarly, at most one unit of flow exits each $v \in R$ to t along (v, t), so each $v \in R$ can be the head of at most one edge $(u', v) \in M'$. Thus, M' is a matching ☒

Lemma 3: M' contains k edges.

Proof: Consider the cut $\{s\} \cup L$ in G'. We know that the value of the flow f' $(= k)$ is also equal to the total flow crossing from $\{s\} \cup L$ to $R \cup \{t\}$, minus the total flow crossing back from $R \cup \{t\}$ to $\{s\} \cup L$. But this latter quantity is 0, since the only edges between these cross from L to R by construction. Moreover, since f' is integer-valued, each of these edges carries either 0 or 1 units of flow; hence, the number of edges with one unit of flow (= size of M') is precisely the value of the flow $(= k)$ ☒☒

In summary:

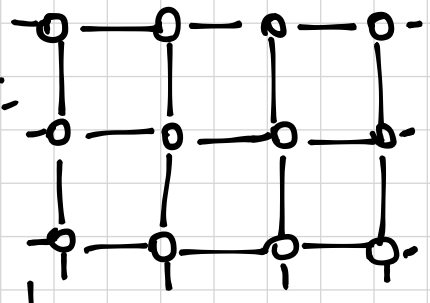
Theorem Our algorithm solves the Maximum Bipartite Matching problem, and can be implemented in time $O(|V||E|)$ (assuming each vertex has an incident edge).

Proof: Lemma 1 guarantees that the value of the maximum flow in G' is at least the size of the maximum matching, Lemma 2 guarantees that the algorithm returns a matching, and Lemma 3

guarantees that the size of this matching is at least the value of the flow, which we just established was at least the size of the maximum matching - since we are finding a matching of this size, our matching must indeed be of maximum size.

As for the running time, recall that our assumption that each vertex has at least one edge incident to it allowed us to guarantee that the Ford-Fulkerson algorithm runs in time $O(|E| \cdot C)$ where C was an upper bound on the maximum value of the flow, e.g., the sum of the capacities of all edges leaving s. Here, this gives a bound of $C = |L| \leq |V|$. Thus, it runs in time $O(|E| \cdot |V|)$. We observe that constructing G' and reading off M' can each be easily performed in $O(|E|)$ time, so the $O(|E| \cdot |V|)$ time for Ford-Fulkerson dominates. ▨

Image Segmentation

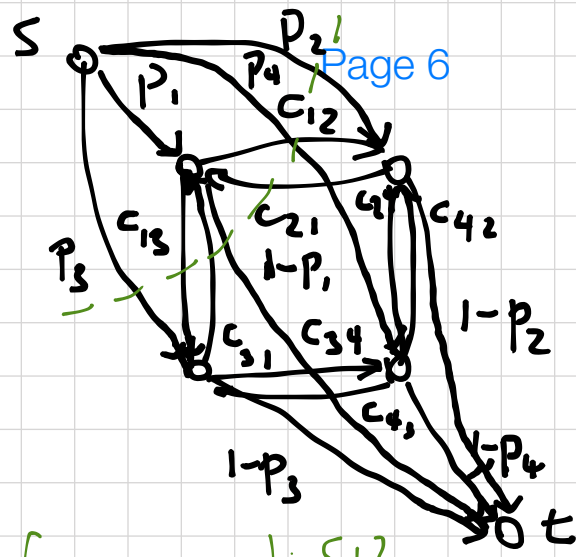


A basic task in computer vision is to separate an object from the rest of an image. We can model this task as follows: for each pixel, we suppose we have some likelihood that it is in the object p_i and there is some cost c_{ij} to labeling neighbors i & j differently, to help encourage a smooth boundary. We wish to find a partition of the pixels into two sets, A (object) and B (background) to maximize $\sum_{i \in A} p_i + \sum_{i \in B} (1 - p_i) - \sum_{\substack{\text{neighbors } i,j: \\ \text{exactly one in } A}} c_{ij}$

We'll show how to convert this into a Min-Cut problem, that we can then solve using, eg, Ford-Fulkerson. First, we note that the objective can be rewritten as:

maximize $(\#\text{pixels}) - \sum_{i\in A}(1-p_i) - \sum_{i\in B} p_i - \sum_{\text{nbrs } i,j \text{ one in } A} c_{ij}$

Why? $\sum_{i\in A} 1 + \sum_{i\in B} 1 \to \sum_{i\in A}(1-(1-p_i)) + \cdots$



foreground: {1}
background: {2,3,4}

So we are, equivalently, minimizing $\sum_{i\in A}(1-p_i) + \sum_{i\in B} p_i + \sum_{\text{cut nbrs } i,j} c_{ij}$.

We generate a directed version of the pixel grid, with c_{ij} capacity on the edge from pixel i to pixel j. We add new source & sink vertices s & t, such that s has an edge (s,i) with capacity p_i for each i and t has an edge (i,t) with capacity $1-p_i$ for each i. Suppose now that we find an s-t cut in this graph, and declare the pixels on the s-side of the cut to be in the object (A), and the pixels on the t-side to be in the background (B). Then, observe: for each pixel in the object, the cut (i,t) edge contributes $1-p_i$ to the cut; each pixel in the background contributes p_i to the cut via the (s,i) edge; and, each neighboring pair (i,j) that is separated contributes c_{ij}. So, the total cost of the cut is $\sum_{i\in A}(1-p_i) + \sum_{i\in B} p_i + \sum_{\substack{\text{neighbors } i,j \\ \text{cut}}} c_{ij}$. This is exactly what we want: a cut A, B that minimizes this quantity maximizes $\#\text{pixels} - \sum_{i\in A}(1-p_i) - \sum_{i\in B} p_i - \sum_{\text{nbrs } i,j \text{ cut}} c_{ij}$ which we argued previously was our objective. So, we can read the desired segmentation off directly from a Min-Cut in the directed graph we construct. Since the capacity C

can be bounded by $\sum_i p_i \leq \#\text{pixels}$, Ford-Fulkerson will find such a cut in time $O(\#\text{pixels}^2)$, thus giving a quadratic-time algorithm overall.