CS 670 Spring 2015 - Solutions to HW 7

Problem 26.2-6

Let $G(V, E)$ be the original graph on which the flow problem is defined. Introduce two vertices, s and t. For each s_i, add an edge from s of capacity p_i. For each t_j, add an edge from t_j to t of capacity q_j. Call the new graph $G'(V', E')$. Let F be the value of the max s-t flow f in G'.

Since $\{(s, s_i)\}_i$ is an s-t cut, $F \leq \sum_i p_i$.

By assigning a flow of p_i to edges (s, s_i) and q_j to (t_j, t) we have a flow of at least $\sum_i p_i = \sum_j q_j$. This is because we are assured of such a flow in G. Hence $F \geq \sum_i p_i$.

Thus $F = \sum_i p_i$ and $f((s, s_i)) = p_i$ for any max flow f. Similarly $f((t_j, t)) = q_j$. The flow conservation constraints at s_i and t_j enforce that the flow out of $s_i = p_i$ and the flow into t_j is q_j.

By computing the max s-t flow f, we can compute the flow assignments to the original problem.

Problem 26.2-11

For a cut $(S, \bar{S})$, let $c(S, \bar{S})$ denote the number of edges crossing the cut. By definition, the edge connectivity

$$k = \min_{S \subset V} c(S, \bar{S})$$

Fix a vertex $u \in V$. For every cut $(S, \bar{S})$, there is a vertex $v \in V$ such that u and v are on either side of the cut. Let $C_{u,v}$ denote the value of the min u-v cut. Thus

$$k = \min_{v \in V, v \neq u} C_{u,v}$$

For each $v \neq u$, $C_{u,v}$ can be determined by computing the max flow from u to v. Since G is undirected, we need to implement an undirected variant of the max flow algorithm. Set all edge capacities to 1. During each step of the flow computation, search for an undirected augmenting path and send a flow of 1 through this path.

The edge connectivity is the minimum over all $C_{u,v}$.

Problem 26-2

a. For every vertex $v \in V$, make a copy v' and call the resulting set of copies V'. Construct a bipartite graph $G'(V \cup V', E')$ where an edge $(v, u') \in E'$ if and only if $(v, u) \in E$.

A matching M is a subset of the edges such that no two edges in M share a vertex. A matching with the maximum number of edges is called as the maximum matching of a graph.

Given a matching M of G', the following procedure generates a vertex disjoint path cover of the graph G.

(**i**) Start a new path p with a vertex v that is not already in any path.

(**ii**) If v is unmatched in M, terminate the path.

(**iii**) Otherwise, v is matched to a vertex u'. Add u to the path p. If u had already been placed in a path (say q), then append the paths to get $p \rightarrow q$ and go to step (i). Else, go to step (ii) and process vertex u.

Since M is a matching, given a vertex at most one candidate edge leaves it and at most one candidate edge enters it. We start at a vertex or enter a vertex and leave it, building a single path. If we re-enter a vertex that was already left, then it should have been a start of a path (otherwise this would lead to a cycle in G) and we append them. Hence no vertex is in two paths.

Also, every vertex is in at least one path. Hence we have a vertex disjoint path cover (call the set of paths as P).

Given a vertex disjoint cover, we can construct a matching as follows. The edge $(v, u') \in M$ if and only if (v, u) is in a path. Hence there is a one to one correspondence between matchings and vertex disjoint path covers. We now relate their sizes.

$$\#V = \sum_{p \in P} \#Vertices(p) = \sum_{p \in P} 1 + \#Edges((p))$$

$$= \sum_{p \in P} 1 + \sum_{p \in P} \#Edges((p)) = \#P + \#M$$

Since $\#V$ is fixed, maximizing $\#M$ minimizes $\#P$.

A matching with maximum $\#M$ (maximum matching) can be found using a (integer) max-flow computation. Add a source s and a sink t. Add edges of capacity 1 from s to every vertex in V. All edges in E' are assigned capacity 1 and directed from V to V'. Add edges of capacity 1 from every vertex in V' to t. Compute the max flow from s to t using Ford-Fulkerson algorithm. The set of edges in E' with a flow of 1 constitutes a maximum matching.

b. The algorithm fails if G has cycles as the set of paths output may no longer be vertex disjoint.

Problem 26-4

a. Compute the residual network given the flow f. (Note that the residual graph contains the edge whose capacity was increased.) Search for an augmenting path and if one is found, update the flow. Informally, we perform one iteration of Ford-Fulkerson.

The flow is increased by 1 if (u, v) is in every min-cut, and is unchanged otherwise. One iteration suffices because the flow is increased by at most 1. The running time is dominated by searching for an augmenting path, which is $O(|V| + |E|)$ using BFS or DFS.

b. Let f be the originally computed flow. If $f(u, v) = 0$, there is no change. Hence assume $f(u, v) \geq 1$.

Define a new flow f' as $f'(u, v) = f(u, v) - 1$ and $f'(e) = f(e)$ for every edge $e \neq (u, v)$.

The new flow f' satisfies all capacity constraints. It violates the flow conservation constraint at u (unless $u = s$). There is one more unit of flow entering u. Likewise, it violates the flow conservation constraint at v (unless $v = t$). There is one more unit of flow leaving t.

In the residual graph with respect to f', search for an augmenting path from u to v.

If such a path exists, augment the flow along that path.

If no such path exists, we have to reduce the flows from $s \to u$ and $t \to v$ by 1 each. This can be done by finding an augmenting path from u to s and augment the flow by 1. Likewise from t to v.