Homework #10
Introduction to Algorithms/Algorithms 1
600.363/463
Spring 2014

Solutions

April 24, 2014

Problem 1 (20 points)

Suppose we have a set of m clients, $C = \{c_1, c_2, \ldots, c_m\}$ each of which must access some website. Suppose further that we have a set of n servers, $S = \{s_1, s_2, \ldots, s_n\}$, each of which hosts some set of websites. For each client c_i, there is a set of servers $S_i \subseteq S$, the set of servers that have the webpage that c_i wants to access. For each client c_i, call S_i the *acceptable server set* for client c_i. Each server s_j can serve at most ℓ_j clients, where $\ell_j \in \{0, 1, 2, \ldots\}$ for all $1 \leq j \leq n$. For each server s_j, call ℓ_j the *service limit* of server s_j. Given a set of clients $C = \{c_1, c_2, \ldots, c_m\}$ and a set of servers $S = \{s_1, s_2, \ldots, s_n\}$, and given the acceptable servers for each client and the service limit for each server, we would like to devise a plan whereby we decide which server each client is going to access to get the webpage it wants. That is, we wish to assign, for each client $c_i \in C$, a server $s_j \in S$ that client c_i is going to access. For each $1 \leq j \leq n$, at most ℓ_j clients can access server s_j. We wish to come up with a *valid assignment* of clients to servers– one that respects the servers' service limits– while also ensuring that as many clients as possible access their desired webpages. An assignment is a set of pairs of the form (c_i, s_j). Each such pair (c_i, s_j) denotes that client c_i accesses server s_j. An assignment M is valid if

(a) No client accesses more than one server (a client only needs to access one server to retrieve the webpage it wants). That is, for all i, there is at most one pair in M of the form (c_i, s_j).

(b) No server serves more clients than its service limit. That is, for all j, there are at most ℓ_j pairs in M of the form (c_i, s_j).

A valid assignment M is *maximum* if it is the largest valid assignment (i.e., $|M|$ is maximized while still being a valid assignment). Note that by definition, a valid assignment M must have $|M| \leq \min\{m, \sum_{j=1}^{n} \ell_j\}$.

Your job is to give an algorithm that takes as input

(i) a set of clients $C = \{c_1, c_2, \ldots, c_m\}$,

(ii) a set of servers $S = \{s_1, s_2, \ldots, s_n\}$,

(iii) a set of acceptable server sets $A = \{S_1, S_2, \ldots, S_m\}$ with $S_i \subseteq S$ for all $1 \leq i \leq m$ and

(iv) a set of service limits $L = \{\ell_1, \ell_2, \ldots, \ell_n\}$,

and produces as output a maximum valid assignment of clients to servers.

Your algorithm should run in $O(m^2 n)$ time. Prove the correctness and runtime of your algorithm. Hint: consider the similarities of this problem to bipartite matching.

Solution

Our solution will be to turn this problem into a maximum flow problem. We will create a flow network as follows: for each $c_i \in C$, create a corresponding vertex u_i. For each $s_j \in S$, create a corresponding vertex v_j. Create a source vertex s and a sink t. For each u_i, create edge (s, u_i) with unit capacity. For each $c_i \in C$, for each $s_j \in S_i$, create edge (u_i, v_j) also with unit capacity. Finally, for each $s_j \in S$, create edge (v_j, t) with capacity ℓ_j.

Our strategy will be to use Ford-Fulkerson on this flow network to find an integral flow. Via a similar proof to that given in CLRS for bipartite matching, we will show that the maximum flow in this graph corresponds precisely to the maximum valid assignment in our original problem.

Runtime: We require $O(m+mn+n)$ time to construct the described flow network. The maximum flow in this network $\min\{m, \sum_{i=1}^{m} |S_i|, \sum_{j=1}^{n} \ell_j\} = O(m)$ and there are $O(m + mn + n) = O(mn)$ edges in the graph, so the runtime of Ford-Fulkerson on this network is $O(m^2 n)$.

Correctness: we have already proven the Ford-Fulkerson algorithm to be correct. It remains for us to show that the maximum flow corresponds precisely to the maximum valid assignment. Firstly, let us note that the edges and capacities of our flow network precisely encode the constraints we must obey– each client is connected to at most one server, and thus at most one unit of flow can come into any client's node in the graph. Similarly, flow can move from a client vertex only to the vertex of a server in that client's set of acceptable servers. Finally, the out-flow

capacities on the server nodes correspond precisely to the service limits of those server nodes. We will show that if M is a valid assignment of cardinality $|M|$, then there is an integral flow f in the flow network G (constructed as described above) with $|f| = |M|$, and if f is an integral flow in G of value $|f|$, then there is a valid assignment M with $|M| = |f|$. It will follow that the maximum flow and the maximum valid assignment have the same cardinality, from which the correctness of our algorithm follows (we will show below that a flow encodes an assignment and vice versa, so we can retrieve one from the other).

The following proof follows almost exactly the proof in CLRS for maximum bipartite matching. We need only change a few things, namely the capacities on the edges into the sink node. Let M be a valid assignment. Define flow f so that for all $(c_i, s_j) \in M$, $f(u_i, v_j) = 1$ and $f(s, u_i) = 1$. For each $j = 1, 2, \ldots, n$, set $f(v_j, t) = \sum_{i=1}^{n} f(u_i, v_j)$. Define $f(u, v) = 0$ for all other edges. Since M is valid, our flow obeys the capacity constraints of our flow network G and conserves flow (recall that for each c_i, $f(u_i, v_j) = 1$ for some v_j if and only if $f(s, u_i) = 1$, and the fact that M is a valid assignment ensures that we do not send more flow into a node v_j) than the flow that leaves it (and ensures that this flow does not exceed the capacity, since there is one unit of flow leaving node v_j for each client assigned to s_j). Note also that f is indeed an integral flow. Define

$$S = \{s\} \cup \{u_1, u_2, \ldots, u_m\}$$
$$T = \{t\} \cup \{v_1, v_2, \ldots, v_n\}.$$

The net flow across cut (S, T) is precisely $|M|$ by construction (there is exactly one unit of flow across this cut for each $(c_i, s_j) \in M$). By CLRS lemma 26.4 (for any cut (S, T), $f(S, T) = |f|$), we have that the flow over cut (S, T) is $f(S, T) = |f| = |M|$.

Conversely, let f be an integral flow in our flow network G. define set M as follows:

$$M = \{(c_i, s_j) : f(u_i, v_j) > 0\}.$$

Define $U = \{u_1, u_2, \ldots, u_m\}$ and $V = \{v_1, v_2, \ldots, v_n\}$. By construction of G, for each $u \in U$, there is precisely one edge entering vertex u, and this edge has unit capacity. If there is one unit of flow coming into vertex u, then that flow must exit through one and only one of the edges leaving u because f is integral. So we have $f(u, v) = 1$ for some $v \in V$ if and only if $f(s, u) = 1$. Similarly, $f(v, t) = k$ if and only if there are k distinct vertices $u_{i_1}, u_{i_2}, \ldots, u_{i_k}$ for which $f(u_{i_a}, v) = 1$ (and $f(u, v) = 0$ for all other $u \in U$ not among these k vertices). It follows from these facts that M is a valid assignment. It remains for us to show that $|f| = |M|$. This fact again follows from CLRS lemma 26.4– $|f| = f(S, T) = |M|$ by our construction of M.

One small issue remains– we have only discussed integral flows here. What about non-integral flows? This is where the Ford-Fulkerson algorithm becomes crucial. CLRS theorem 26.10 states that if our flow network has integer capacities (which our graph G certainly does), then Ford-Fulkerson produces an integral flow.

Problem 2 (20 points)

In class, we discussed how Ford-Fulkerson can, in some circumstances, take quite a long time to terminate (or, in some cases, doesn't terminate at all), depending on how we choose our augmenting paths.

Part 1 (10 points)

Give an example flow network on 4 nodes with integer capacities on which Ford-Fulkerson must recompute the residual network at least 999 times (i.e., we must find an augmenting path at least 999 times), if we choose the wrong augmenting path. Note: don't overthink this– If you find yourself needing to include a picture in your writeup you're probably making things more complex than is necessary. Give a brief (2 or 3 sentences) explanation as to why Ford-Fulkerson must recompute the augmenting path so many times.

Part 2 (10 points)

The Edmonds-Karp algorithm is an improvement to Ford-Fulkerson that avoids the problem we discussed in Part 1. Instead of choosing an augmenting path arbitrarily, we do the following:

(1) Given the residual network G_f, compute yet another graph H_f, which is identical to G_f except that all of its residual capacities are 1.

(2) Find the shortest path from the source node s to the sink node t in H_f. Call the resulting path P.

(3) Use P as the augmenting path.

Put another way, Edmonds-karp chooses the augmenting path by finding the shortest path from source to sink as G_f as measured by number of edges. What augmenting path would the Edmonds-Karp algorithm choose in your example graph from Part 1 above? How many times would we have to compute an augmenting path before termination if we follow Edmonds-Karp instead of the augmenting path that you used in Part 1?

Part 3 (Optional, 10 extra credit points)

Give an example of a flow network on which Ford-Fulkerson will never terminate for some unfortunate choice of augmenting paths. Show why Ford-Fulkerson will not terminate on this graph using these augmenting paths. You need not give a rigorous proof, but at least a paragraph of explanation is probably in order. Note: the solution to this problem is all too easy to find on the internet. There is no shame in using, for example, the example given on the Wikipedia page for Ford-Fulkerson, but your solution must be in your own words.

Solutions

Solutions to this problem can be found easily in most textbooks (including CLRS), or on Wikipedia.

Optional exercises

Solve the following problems and exercises from CLRS: 26.2-7, 26.2-10, 26.2-13, 26.3-2.