Augmenting Through The Auxiliary Network

Consider a network with underlying multigraph D and capacities $\{c_e | e \in E(D)\}$. Let f be an integer flow in this network. We define the auxiliary network and an associated multigraph D' as follows:

(i) For each arc $e = uv$ of D, if $x_e < c_e$ then there is an arc e^f from u to v in D' of capacity $c'_{e^f} = c_e - x_e$.

(ii) For each arc $e = uv$ of D, if $x_e > 0$ then there is an arc e^b from v to u in D' of capacity $c'_{e^b} = x_e$.

For any path P of D' from s to t, we define Δ_P to be the minimum of $\{c'_e | e \in E(P)\}$. We augment along P to obtain a new flow in the original network as follows:

For each forward arc e^f of P, we set $x_e = x_e + \Delta_P$.
For each backward arc e^b of P, we set $x_e = x_e - \Delta_P$.

As verified in class, this gives a new integer $s - t$ flow of volume $v(f) + \Delta_P$.

Our algorithms for finding maximum flows are iterative. In each iteration, given a flow f, we will find a path P of the auxiliary network to augment along, augment along this path, and then set f to be this new flow. Initially, we set f to be the flow which has $x_e = 0$ for all $e \in E(D)$.

Clearly the maximum volume of an $s - t$ flow is at most the sum of the capacities of all the edges of D, so if we let C be the maximum capacity of an edge then the maximum volume of a flow is at most $|E|C$. Thus, we perform at most $|E|C$ iterations in our iterative procedure. Since the main step in each iteration is to search for an $s - t$ path in the auxiliary network, we see that the running time per iteration is $O(|E|)$ and obtain that the overall running time of the algorithm is $O(|E|^2 C)$.

Note, that to input a capacity c_e which is at most C requires only $log(C)$ bits, so this running time may be exponentially large in comparision to the input size.

We saw that clever choices of the path P along which we augment could dramatically improve our running time bound. we considered two possibilities:

(1) Augment along an $s - t$ path P which has as few edges as possible, or

(2) Augment along an $s - t$ path which maximizes Δ_P.

We show below that taking the first approach implies we perform $O(|V||E|)$ iterations, while taking the second implies we perform $O(|E|log|E|C)$ iterations. We can find a path P as in (1) using breadth first search. We can find a path P as in (2) by modifiying Dijkstra's algorithm so that in each iteration we add the edge of maximum capacity from S to $V - S$ to the rooted tree constructed so far. This algorithm can be implemented $O(|E|log|E|)$ time, so the total

timre complexity of the second algorithm is $O(|E|^2 log(|E|C) log|E|)$. Faster algorithms for Network Flow are discussed at the end of Chapter 26 of Cormen.