CMSC 651 Advanced Algorithms	Lecture 14
Lecturer: Samir Khuller	Th. Mar. 4, 1993

Notes by Marios Leventopoulos

17 An $O(n^3)$ Max-Flow Algorithm

The max-flow algorithm operates in phases. At each phase we construct the residual network G_f, and from it we find the layered network L_{G_f}.

In order to construct L_{G_f} we have to keep in mind that the shortest augmenting paths in G with respect to f correspond to the shortest paths from s to t in G_f. With a breadth-first search we can partition the vertices of G_f into *disjoint* layers according to their distance (the number of arcs in the shortest path) from s. Further more, since we are interested only in *shortest* $s-t$ paths in G_f we can discard any vertices in layers greater than that of t, and all other than t in the same layer as t, because they cannot be in any shortest $s-t$ path. Additionally a shortest path can only contain arcs that go from layer j to layer $j+1$, for any j. So we can also discard arcs that go from a layer to a lower layer, or any arc that joins two nodes of the same layer.

An augmenting path in a layered network with respect to some flow g is called *forward* if it uses no backward arcs. A flow g is called *maximal* (not *maximum*) if there are no **F**orward **A**ugmenting **P**aths (FAP).

We then find a maximal flow g in L_{G_f}, add g to f and repeat. Each time at least one arc becomes saturated, and leaves the net. (the backward arc created is discarded). Eventually s and t become disconnected, and that signals the end of the current phase. The following algorithm summarizes the above:

Step 1: $f = 0$

Step 2: Construct G_f

Step 3: Find *maximal* flow g in L_{G_f}

Step 4: $f \leftarrow f + g$

Step 5: goto step 2 (next phase)

In L_{G_f} there are no forward augmenting paths with respect to g, because g is maximal. Thus all augmenting paths have length greater than $s-t$ distance. We can conclude that the $s-t$ distance in the layered network is increasing from one phase to the other. Since it can not be greater than n, the number of phases is $O(n)$.

Throughput(v) of a vertex v is defined as the sum of the outgoing capacities, or the incoming capacities, whichever is smaller, i.e it is the maximum amount of flow that can be pushed through vertex v.

The question is, given a layered net G_f (with a source and sink node), how can we efficiently find a maximal flow g?

Step 1: Pick a vertex v with minimum throughput,

Step 2: Pull that much flow from s to v and push it from v to t

Step 3: Repeat until t is disconnected from s

By picking the vertex with the smallest throughput, no node will ever have to handle an amount of flow larger than its throughput, and hence no backtracking is required. In order to push flow from v to t we process nodes layer by layer in breadth-first way. Each vertex sends flow away by completely saturating each of its outgoing edges, one by one, so there is at most one outgoing edge that had flow sent on it but did not get saturated.

Each edge that gets saturated is not further considered in the current phase. We charge each such edge when it leaves the net, and we charge the node for the partially saturated edge.

The operation required to bring flow from s to v is completely symmetrical to pushing flow from v to t. It can be carried out by traversing the arcs backwards, processing layers in the same breadth-first way and processing the incoming arcs in the same organized manner.

To summarize we have used the following techniques:

1. Consider nodes in non-decreasing order of throughput
2. Process nodes in layers (i.e. in a breadth-first manner)
3. While processing a vertex we have at most one unsaturated edge (consider only edges we have sent flow on)

After a node is processed it is removed from the net, for the current phase.

Work done: We have $O(n)$ phases. At each phase we have to consider how many times different arcs are processed. Processing an arc can be either *saturating* or *partial*. Once an arc is saturated we can ignore it for the current phase. Thus saturated pushes take $O(m)$ time. However one arc may have more than one unsaturated push per phase, but note that the total number of pulls and pushes at each stage is at most n because every such operation results to the deletion of the node with the lowest throughput; the node where the this operation was started. Furthermore, for each push or pull operation, we have at most n partial steps, for each node processed. Therefore the work done for partial saturation at each phase is $O(n^2)$.

So we have $O(n^3)$ total work done for all phases.