ORIE 633 Network Flows September 18, 2007

Lecture 5

Lecturer: David P. Williamson *Scribe: June Andrews*

1 Efficient max flow algorithms

1.1 Push/relabel algorithm

Recall that the algorithm maintains a preflow, f, that obeys capacity constraints and has at least as much flow entering a node as leaving it. The algorithm also maintains a distance labelling.

Definition 1 *We define a valid distance label d_i $\forall i \in V$ such that:*

- *d_i is a non-negative integer;*
- $d_t = 0$
- $d_s = n = |V|$
- $d_i \leq d_j + 1$ $\forall (i,j) \in A_f$, *the residual graph*

The algorithm does not maintain flow conservation.

Definition 2 *We define the excess at node i to be (flow in - flow out), or $e_i \equiv \sum_{j:(j,i)\in A} f_{ji}$.*

If a node has excess $e_i > 0$ and is not the source or the sink, then the node is considered *active*.

Push/Relabel(Goldberg, Tarjan '88) aka Preflow/Push

$f \leftarrow 0$, $e \leftarrow 0$
$f_{sj} \leftarrow u_{sj}$; $f_{js} \leftarrow -f_{sj}$; $e_j = u_{sj}$ $\forall (s,j) \in A$
$d_s \leftarrow n$; $d_t \leftarrow 0$; $d_i \leftarrow 0 \; \forall \; i \in V - \{s,t\}$
While $\exists$ active node i
  If $\exists j$, s.t. $u_{ij}^f > 0$ and $d_i = d_j + 1$
    Push $\delta \leftarrow \min(e_i, u_{ij}^f)$
      $f_{ij} \leftarrow f_{ij} + \delta$; $f_{ji} \leftarrow f_{ji} - \delta$
      $e_i \leftarrow e_i - \delta$; $e_j \leftarrow e_j + \delta$
  Else Relabel $d_i \leftarrow \min(d_j + 1, (i,j) \in A_f)$.

Theorem 1 *If algorithm terminates and all node labels, d_i, are finite, then f is a maximum flow.*

Proof: If algorithm terminates, then f is a valid flow, since there will not be any excess at any node other than the source and sink. Proof by contradiction: Suppose f is not a max flow. Then there exists an augmenting path P in G_f. By the properties of a distance labelling, this implies that $d_s \leq n-1$, which contradicts $d_s = n$. Hence there is no path from s to t in the residual graph. □

Now we recall the following lemma, which is useful in showing that the distance labels stay finite.

Lemma 2 *If f is a preflow and i is active, then s is reachable from i in G_f.*

This lemma implies $\forall$ *active* i, $d_i \leq 2n-1$ since $d_s = n$ and a path from i to s can be at most of length $n-1$.

Lemma 3 *The total number of relabel operations is at most $n(2n-1) \leq 2n^2$.*

Proof: We know that $0 \leq d_i \leq 2n-1$, d_i is integer, d_i never decreases, and each relabel operation on i increases d_i by at least 1. So each vertex can have at most $2n-1$ relabel operations and there are n vertices. Thus there are at most $n(2n-1) \leq 2n^2$ executions of relabel. □

In the algorithm, there are two types of pushes:

(i) push is saturating if (i,j) is saturated by a push; i.e. $\delta = u_{ij}^f$

(ii) otherwise the push is nonsaturating and $\delta < u_{ij}^f$; i.e. $\delta = e_i$

Lemma 4 *There are at most mn saturating pushes.*

Proof: Pick an edge $(i,j) \in A$. We need $d_i = d_j + 1$ in order to push from i to j; to do it again, need to push back from j to i and $d_j = d_i + 1$. This implies we need d_j to increase at least by 2, which implies there are at most $n-1$ saturating pushes from i to j by Lemma 3. Since there are at most m edges, there are at most $m(n-1)$ saturating pushes. □

We recall the following lemma without proof.

Lemma 5 *There are at most $4n^2m$ nonsaturating pushes.*

Theorem 6 *Push/Relabel can be implemented in $O(n^2m)$ push/relabel operations, each of which runs in constant time.*

We now turn to improving the running time of Push/Relabel. From the analysis above, it is the bound on the number of non-saturating pushes that determines the running time of the algorithm. In the algorithm above, we did not specify how pushes and relabels should be executed. We now show that if we are a little more careful, we can obtain a better bound on the non-saturating pushes.

FIFO Push/Relabel

initialize same way as Push/Relabel,
    but set $d_i \leftarrow 0 \ \ \forall i, i \neq s$
Put all active vertices in queue Q
While $Q \neq \emptyset$
    Let i be vertex at front of Q
    While $e_i > 0$ and $\exists j : u_{ij}^f > 0$ and $d_i = d_j + 1$
        Push(i, j)
        If j becomes active
            Add j to end of Q
    If $e_i > 0$
        Relabel i and add it to end of Q.

Lemma 7 *The number of passes over the queue in FIFO push-relabel is at most* $4n^2$.

Corollary 8 *The number of non-saturating pushes in FIFO push-relabel is* $O(n^3)$.

Proof: Note that each node is activated at most once per pass. Also, there is at most one non-saturating push per node per pass since the inner while loop ends with a non-saturating push. Therefore, given the lemma, the number of non-saturating pushes is $O(4n^2)$. □

We observe that the proofs of Lemma 3 and 4 remain the same, so that we get the following theorem.

Theorem 9 *The total number of push/relabel operations for FIFO Push/Relabel is* $O(n^3)$*, so that the total running time is* $O(n^3)$.

Proof of Lemma 7: We use the potential function $\Phi = \max\{d_i : i \text{ active}\}$. Divide the passes into two types:

(i) passes in which some distance label increases;

(ii) passes in which no distance label changes.

By our previous argument each $d_i \leq 2n - 1$, so that the total number of passes of type (i) is at most $n(2n - 1) \leq 2n^2$.

If a pass of type (ii) occurs, then each vertex has excess moved to lower-labeled vertices. Thus the max label of *active* node is no longer, which implies that Φ decreases by at least 1.

Let $\Delta\Phi_\ell$ be the change in Φ from the beginning of pass ℓ until the end of pass ℓ. When $\Delta\Phi_\ell > 0$, we know that some distance label has increased by at least $\Delta\Phi_\ell$. Thus $\sum_{\ell:\Delta\Phi_\ell>0} \Delta\Phi_\ell \leq 2n^2$.

Because Φ is initially zero, stays non-negative, and is zero at the end of the algorithm, the total number of passes in which Φ decreases cannot be more than $\sum_{\ell:\Delta\Phi_\ell>0} \Delta\Phi_\ell \leq 2n^2$; that is, since we know that the total increase in Φ over all passes in which it increases is at most $2n^2$, the total number number of passes in which it decreases is also at most $2n^2$.

Thus the sum of the passes of type (i) and (ii) is $\leq 2n^2 + 2n^2 \leq 4n^2$ passes. □

Theorem 10 *(Goldberg, Tarjan, 1988) Push/Relabel can be implemented in* $O(nm \log(n^2/m))$ *time.*

Push/relabel is the fastest algorithm in practice. It also uses tricks to stop early so that its performance is usually much better than what the worst-case running time indicates.

The fastest algorithm theoretically is Goldberg and Rao '98, which has a running time of $O(min(n^{2/3}, m^{1/2})m \log(n^2/m) \log(mU)))$, where $U = \max u_{ij}$ is the largest capacity value.

If the only thing we are interested in is the min s-t cut then we can calculate this faster than the max flow by stopping the push/relabel algorithm early. In particular, we change the definition of an *active* node to be a node i such that $e_i > 0$ and $d_i < n$. The intuition as to why this change will compute the min cut faster is once $d_i > n$ all excess is going to flow back to the source. Additionally, all reachable vertices from t in the residual graph are *inactive*, so the cut between reachable nodes from t and the complement is the min s-t cut.

2 Global minimum cuts

We now turn to the following problem.

Global Min-cut

- **Input:** Given:
  - directed graph G=(V,A)
  - capacities $u_{ij} \geq 0 \; \forall (i,j) \in A, \; integer^+, 0$
- **Goal:** Goal: find $S \subset V, S \neq \emptyset$ that minimizes $u(\delta^+(S)) = \sum_{(i,j) \in \delta^+(S)} u_{i,j}$

We will make use of the *minimum s-cut* when solving for the global min-cut. The minimum s-cut problem finds a cut S that minimizes $u(\delta^+ S))$ such that $s \in S$.

Given an algorithm for min s-cut we can find the global min-cut by running the min s-cut algorithm twice. We do this by running the min s-cut, then running the min s-cut in a graph with all the arcs reversed. In the latter case, if the algorithm returns S, we consider the cut $S' = V - S$ in the original graph. This will be a minimum cut such that $s \notin S'$. We then return the cut S or S' which has the smallest capacity, and this will be the global minimum cut.

We can find min s-cut in $n - 1$ max flows. For each $i \neq s$, we find a min s-i cut via a max flow, and take the minimum of all of these. We know that this gives us the min s-cut, since for the optimal min s-cut S^*, there exists some $i \notin S^*$; the min s-i cut in this case can have value no greater than the capacity of S^*.

Next time we will see how we can find a minimum s-cut in the running time of a single push/relabel algorithm.