19: Network Flow II - The Vengeance

CS 473u - Algorithms - Spring 2005

March 28, 2005

1 Previous Lecture

Theorem 1.1 (Max-flow min-cut theorem)

If $f$ is a flow in a flow network $G = (V, E)$ with source $s$ and sink $t$, then the following conditions are equivalent:

1. $f$ is a maximum flow in $G$

2. The residual network $G_f$ contains no augmenting paths.

3. $|f| = c(S, T)$ for some cut $(S, T)$ of $G$. And $(S, T)$ is a minimum cut in $G$. 
People that do not know maximum flows: essentially everybody.

Average salary on earth: $5,000

People that know maximum flow - most of them work in programming related jobs and make at least $10,000 a year.

Salary of people that learned maximum flows: > $10,000
Salary of people that did not learn maximum flows: < $5,000
Salary of people that know latin: $0 (unemployed)

Thus, by just learning maximum flows you can double your future salary!
3 Ford-Fulkerson Algorithm

**Ford-Fulkerson(G,s,t)**

- Init flow $f$ to zero

While $\exists$ a path $p$ from $s$ to $t$ in $G_f$

- $c_f(p) \leftarrow \min \{c_f(u,v) \mid uv \text{ is in } p\}$

For each edge $uv$ in $p$ do

- $f(u,v) \leftarrow f(u,v) + c_f(p)$
- $f(v,u) \leftarrow f(v,u) - c_f(p)$

**Lemma 3.1** If the capacities on the edges of $G$ are integers, then Ford-Fulkerson runs in $O(m|f^*|)$ time, where $|f^*|$ is the amount of flow in the maximum flow and $m = |E(G)|$.

*Proof:* Observe that the Ford-Fulkerson algorithm perform only subtraction/addition and min operations. Thus, if it finds an augmenting path, then $c_f(p)$ must be a positive integer number. Namely, $c_f(p) \geq 1$. Thus, $|f^*|$ must be an integer number (by induction), and each iteration of the while improves the flow by at least 1. It follows, that after $|f^*|$ iterations of the while the algorithm stops. However, each iteration of the while loop takes $O(m)$ time, as can be easily verified.

**Observation 3.2 (Integrality theorem)** If the capacity function $c$ takes on only integral values, then the maximum flow $f$ produced by the Ford-Fulkerson method has the property that $|f|$ is integer-valued. Moreover, for all vertices $u$ and $v$, the value of $f(u,v)$ is an integer.

4 Edmonds-Karp algorithm

Edmonds-Karp algorithm works by modifying the Ford-Fulkerson algorithm so that it always return the (edge) shortest augmenting path in $G_f$. This is implemented by finding $p$ using BFS.

**Definition 4.1** For a flow $f$, let $\delta_f(v)$ be the length of the shortest path from the source $s$ to $v$ in the residual graph $G_f$. Each edge is considered to be of length 1.

**Lemma 4.2** If the Edmonds-Karp algorithm is run on a flow network $G = (V,E)$ with source $s$ and sink $t$, then for all vertices $v \in V - s,t$, the shortest path distance $\delta_f(v)$ in the residual network $G_f$ increases monotonically with each flow augmentation.

We delay proving this technical lemma. Lets first prove that it is helping in our life.

**Lemma 4.3** During the execution of EK algorithm, an edge $uv$ might disappear (and thus reappear) from $G_{f_i}$ at most $n/2$ times, where $n = |V(G)|$. 

Proof: When $\overrightarrow{uv}$ disappears that it must be that $uv$ was on the augmenting path $p$. Furthermore, $c_f(p) = c_f(uv)$. We continue running EK till $uv$ magically reappear. This means that before $uv$ reappeared, we handled an augmenting path $\pi$ that contains the edge $\overrightarrow{vu}$. Let $g$ be the flow just after this. We have

$$\delta_g(u) = \delta_g(v) + 1 \geq \delta_f(v) + 1 = \delta_f(u) + 2$$

as Edmonds-Karp is always augmenting along the shortest path. Namely, the distance of $s$ to $u$ had increased by two between its disappearance and reappearance. Since $\delta_0(u) \geq 0$ and the maximum value of $\delta_f(u)$ is $n$, it follows that $uv$ can disappear and reappear at most $n/2$ times during the execution of Edmonds-Karp algorithm.

Observation 4.4 Every time we add an augmenting path during the execution of EK algorithm, at least one edge disappears from the residual graph $G_f$. The “bottleneck” edge (the one that realizes the flow along the augmenting path) along the augmenting path disappears after we apply the augmenting path.

Lemma 4.5 Edmonds-Karp algorithm handles at most $O(nm)$ augmenting paths before it stops. In particular, each augmenting path takes $O(m)$ time, and the overall running time of Edmonds-Karp algorithm is $O(nm^2)$ time, where $n = |V(G)|$ and $m = |E(G)|$.

Proof: Every edge might disappear at most $n/2$ times during EK execution. Thus, there are at most $nm/2$ edge disappearances during the execution of the EK algorithm. Each time we augment a path, an edge disappears. Augmenting a path takes $O(m)$ time, as we have to perform BFS to find the augmenting path. It follows, that the overall running time is as claimed.

We are done??
Thus,
\[ \delta_f(u) = \delta_f(v) + 1 > \delta_g(v) = \delta_g(u) + 1. \]
Thus, \( \delta_f(u) > \delta_g(u) \). A contradiction. WHY?

5 Maximum Bipartite Matching

**Definition 5.1** For an undirected graph \( G = (V, E) \) a **matching** is a subset of edges \( M \subseteq E \) such that for all vertices \( v \in V \), at most one edge of \( M \) incident on \( v \). A **maximum matching** is a matching \( M \) such that for any matching \( M' \) we have \( |M| \geq |M'| \).

A bipartite graph:

And a maximum matching in this graph:

A matching is perfect, if it involved all vertices:

**Theorem 5.2** One can compute maximum bipartite matching using network flows.

**Proof:** We create a new graph, with new source on the length and sink on the right. Direct all edges from left to right, and set their capacity to 1. See:
6 Multiple Sources and Sinks

Given several sources and sinks, how can we compute maximum flow on such a network?

Idea: Create a super source, that send all its flow to the old sources, and similarly create a super sink.

Resulting graph: