MATH 409 LECTURE 13
NETWORK FLOWS

REKHA THOMAS

A **network** (G, u, s, t) consists of the following data:

- A digraph G,
- edge capacities $u_e \in \mathbb{R}_{\geq 0}$ for all $e \in E(G)$, and
- two specified notes s (source) and t (sink) in $V(G)$.

The u in (G, u, s, t) is the vector $u = (u_{e_1}, \ldots, u_{e_m})$ of edge capacities. It is also sometimes written as $u = (u_e)_{e \in E(G)}$.

Definition 1.

- A **flow** is a vector $f = (f(e_1), \ldots, f(e_m)) = (f(e))_{e \in E(G)}$ such that $0 \leq f(e) \leq u_e$ for all $e \in E(G)$.
- A flow f satisfies **flow conservation** at vertex v if

$$\sum_{\delta^-(v)} f(e) = \sum_{\delta^+(v)} f(e)$$

  where $\delta^-(v) = \{wv \in E(G)\} = \{$ edges entering v $\}$ and $\delta^+(v) = \{vw \in E(G)\} = \{$ edges leaving v $\}$.
- Given a network (G, u, s, t), an $s - t$ flow is a flow satisfying flow conservation at all $v \in V(G) \backslash \{s, t\}$.
- The **value** of an $s - t$ flow f is

$$value(f) := \sum_{\delta^+(s)} f(e) - \sum_{\delta^-(s)} f(e)$$

  which is the total flow leaving the source node s minus the total flow entering s.

The Max Flow Problem
Input: A network (G, u, s, t).
Output: An $s - t$ flow in the network of maximum value.

The max-flow problem can be modeled as a LP as follows.

$$\begin{array}{lllll} \max & \sum_{e \in \delta^+(s)} x_e - \sum_{e \in \delta^-(s)} x_e & & & \\ s.t. & \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e & = & 0 & \forall\, v \in V(G) \backslash \{s, t\} \\ & 0 \leq x_e \leq u_e & & & \forall\, e \in E(G) \end{array}$$

Date: April 26, 2010.

Since $x_e = 0$ for all $e \in E(G)$ is a solution to this LP, the max flow problem is feasible. Further, the max value of a flow cannot exceed the sum of the capacities on the edges in $\delta^+(s)$ which is finite which means the max flow problem is bounded. Now recall that a feasible bounded LP always has an optimal solution with finite optimal value. So the max flow problem can be solved to optimality using an algorithm for linear programming. Since linear programming can be solved in polynomial time using interior point methods, we get that the max flow problem can be solved in polynomial time. However, we will see combinatorial algorithms that solve the max flow problem without using the LP formulation given above and these algorithms are also polynomial time.

Exercise 2. Write down the dual LP to the above LP formulation of the max flow problem. (The dual of an LP was written down in the previous lecture.)

Exercise 3. Write down a linear or integer programming formulation of the minimum spanning tree problem in an undirected graph G. (i.e., Write down a linear or integer program whose optimal solution gives a MST with respect to the costs on the edges of G.) Argue why your formulation is correct. Look back at Lecture 1 if needed.

Note that we are using the variable x_e instead of $f(e)$ in the above formulation as it's more typical to use x for variables instead of f. Note also that the max flow problem is a linear program. It is bounded if we assume that all the edge capacities are bounded since all solutions must lie in the box specified by the constraints $0 \leq x_e \leq u_e$ for all $e \in E(G)$. Further, since $x_e = 0$ for all $e \in E(G)$ is a feasible solution, this LP is feasible. Then the theory of linear programming implies that the max flow problem always has an optimal solution.

Definition 4.

- An (s,t)-cut in G is a set of edges

$$\delta^+(X) := \{\text{ edges of } G \text{ with tail in } X \text{ and head in } V(G)\backslash X\}.$$

  Here $X \subset V(G)$ is such that $s \in X$ and $t \in V(G)\backslash X$.
- The capacity of an (s,t)-cut $\delta^+(X)$ is the sum of the capacities on the edges of the cut.
- A minimum (s,t)-cut is an (s,t)-cut of minimum capacity in (G,u).

We now develop a combinatorial algorithm to solve this problem. Our discussion is based on [1, Chapter 8.1]. Recall that we use both f and x for flow.

Lemma 5. *For any $A \subseteq V(G)$ such that $s \in A, t \notin A$ and any (s,t)-flow f,*

(a) $value(f) = \sum_{e\in\delta^+(A)} f(e) - \sum_{e\in\delta^-(A)} f(e)$, *and*
(b) $value(f) \leq \sum_{e\in\delta^+(A)} u_e$.

Proof. We prove both facts at one shot. Recall that $value(f)$

$$
\begin{aligned}
:= & \sum_{e\in\delta^+(s)} f(e) - \sum_{e\in\delta^-(s)} f(e) \\
= & \left(\sum_{e\in\delta^+(s)} f(e) - \sum_{e\in\delta^-(s)} f(e)\right) + \sum_{v\in A\setminus\{s\}} \left(\sum_{e\in\delta^+(v)} f(e) - \sum_{e\in\delta^-(v)} f(e)\right) \\
= & \sum_{v\in A} \left(\sum_{e\in\delta^+(v)} f(e) - \sum_{e\in\delta^-(v)} f(e)\right) \\
= & \sum_{e\in\delta^+(A)} f(e) - \sum_{e\in\delta^-(A)} f(e) \leq \sum_{e\in\delta^+(A)} f(e) \leq \sum_{e\in\delta^+(A)} u_e.
\end{aligned}
$$

The first line is the definition of $value(f)$. In the second line we can add the second term to the value of f since all the components of the second term are zero due to flow conservation. The third line aggregates the sum in the second line and the first expression in the fourth line is the final outcome of this aggregate after all the cancellations. The first inequality in the fourth line comes from the fact that $\sum_{e\in\delta^-(A)} f(e)$ is a non-negative number. Lastly, since all $f(e) \leq u_e$, we get the second inequality. □

Corollary 6. *The max value of an (s,t)-flow in a network (G,u,s,t) is less than or equal to the min capacity of an (s,t)-cut in the network.*

Theorem 7. *(Ford & Fulkerson 1956)*
In a network (G,u,s,t) the max value of an (s,t)-flow **equals** *the min capacity of a (s,t)-cut in the network.*

Note that because we already established Corollary 6, what we are left with is to show the equality in the statement of the theorem. We do this after describing the algorithm for solving max flows.

References

[1] B. Korte and J. Vygen. *Combinatorial Optimization*. Springer, Berlin, 2000.