CS 684: Approximation and Network Algorithms	**9/26/01**
Min-cost flows #1	**scribe: Alex Slivkins**

1 Basic definitions.

Let $G = (V, E)$ be a directed graph, possibly with parallel or opposite edges. For all edges e define *capacities* $u_e \geq 0$ and *costs* c_e. Let a *flow* be a function $f : E \to \mathbb{R}$ that satisfies the *capacity contraints*

$$0 \leq f_e \leq u_e \tag{1}$$

for each edge e. Define the cost of a flow f as[1]

$$c(f) = \sum_{e \in E} c_e f_e.$$

For all nodes v define *demands* b_v that sum up to 0. Say a flow f is *feasible* if for each node v it satisfies the *flow conservation* constraints

$$f^{\text{ in}}(v) - f^{\text{ out}}(v) = b_v, \tag{2}$$

where

$$\begin{aligned} f^{\text{ in}}(v) &= \sum_{\text{edge } e \text{ enters } v} f_e \\ f^{\text{ out}}(v) &= \sum_{\text{edge } e \text{ leaves } v} f_e \end{aligned}$$

The Min-Cost Flow problem (MCF in short) is (surprise surprise) to find a min-cost feasible flow.

If there are no parallel or opposite edges we can extend a flow f to a function $V \times V \to \mathbb{R}$

$$f_{uv} = \begin{cases} f_e, & e = (uv) \in E, \\ -f_e, & e = (vu) \in E, \\ 0, & \text{otherwise} \end{cases}$$

Then the flow conservation constraints (2) can be expressed shorter: for each node v

$$\sum_{u \in V} f_{uv} = b_v \tag{3}$$

[1] This is the most common definition for the cost of a flow. Other definitions of the form

$$c(f) = \sum_{e \in E} h(c_e, f_e)$$

where h is a function $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$ can also be meaningful and have been studied in literature.

2 Well-known special cases.

max-flow Given a directed graph G, capacities u_e, terminal pair st. Find an st flow of maximal cost. To express as MCF: set all costs and demands to 0; add an extra edge ts of infinite capacity and cost -1.

shortest-path Given a directed graph G, lenghts l_e, terminal pair st. Find an st path of minimal length. Express it as MCF as follows. Set costs to lengths: $c_e = l_e$; Set all capacities to 1.[2] Set demands $b_s = -1$, $b_t = 1$. Set all other demands to 0.

network design Given a directed graph G, costs c_e for building an edge e, a terminal pair st. Find a subgraph $G' \subset G$ of minimal cost (defined as the sum of edges of G') s.t. G' contains k (edge-)disjoint st paths. Express it as MCF as follows. Set $b_s = -k$, $b_t = k$. Set all other demands to 0. Set all capacies to 1. Keep all costs as given. Let f_e be the *integer* solution to the MCF. Then $\{e : f_e = 1\}$ is a solution to the Network Design problem.

3 Simple algorithm for MCF.

Given a flow f, define the *residue graph* $G_f = (V, E_f)$ with two kinds of edges:

- *forward* edges $e \in E_f$ s.t. $e \in E$ and $f_e < u_e$, of cost c_e.
- *backward* edges $(w, v) \in E_f$ s.t. $e = (v, w) \in E$ and $f_e > 0$, of cost $-c_e$.

Define the cost of a cycle as the sum of the costs of its edges. Say a cycle is *negative* if it has a negative sum.

Lemma 3.1 *If G_f has a negative cycle, then flow f is not min-cost.*
Proof: Say C is a negative cycle in G_f. For $\delta \geq 0$ define a function $f^\delta : E \to \mathbb{R}$ by

$$f_e^\delta = \begin{cases} f_e + \delta, & \text{if } e \in C \text{ is a forward edge,} \\ f_e - \delta, & \text{if } e = (v, w) \text{ is s.t. } (w, v) \in C \text{ is a backward edge,} \\ f_e, & \text{otherwise} \end{cases}$$

f^δ has a smaller cost than f:

$$\begin{aligned} c(f^\delta) - c(f) &= \sum_{\text{fwd } e \in C} \delta c_e - \sum_{\text{bwd } e \in C} \delta c_e \\ &= \delta \ c(C) < 0 \end{aligned}$$

[2] Will also work with all capacities are set to some other number, or to infinity.

Therefore, it remains to prove that if $\delta > 0$ is small enough then f^δ is a feasible flow. An equivalent definition of f_e^δ is

$$f_e^\delta = \begin{cases} f_e + \delta, & \text{if } f_e < u_v \text{ and } e \in C, \\ f_e - \delta, & \text{if } f_e > 0 \text{ and } e = (v,w) \text{ is s.t. } (w,v) \in C, \\ f_e, & \text{otherwise} \end{cases}$$

Therefore, if $\delta > 0$ is small enough then f^δ satisfies the capacity constraints (1).

It turns out that the flow conservation constraints (2) are satisfied by any f^δ. (2) obviously holds for any $v \notin C$. Now suppose $v \in C$. If there are no parallel or multiple edges, it is easy to check that

$$f_{uv}^\delta = \begin{cases} f_{uv} + \delta, & \text{if } (uv) \in C, \\ f_{uv} - \delta, & \text{if } (vu) \in C, \\ f_{uv}, & \text{otherwise} \end{cases}$$

So if (u, v, w) are consecutive vertices of C then

$$f_{uv}^\delta + f_{wv}^\delta = f_{uv} + f_{wv}$$

Therefore (3) holds at v. The general case is similar but a lot messier. $\square$

Now we are ready to present a simple *cycle-cancelling* algorithm for MCF.

Algorithm 3.2 Solves MCF by *cycle cancelling.*

1. Find *some* flow f by a single run of a max-flow computation (known from CS 681 or CS 482). If all capacities and demands are integer, then wlog f is an integer flow.
2. Repeatedly change f, each time decreasing its cost. Specifically, if G_f has a negative cycle, set $f \leftarrow f^\delta$ where f^δ satisfies (1) with the largest δ.
3. Stop when G_f contains no negative cycles.

In the next lecture we'll prove that if G_f contains no negative cycles then f is min-cost.

4 Running time

If all costs and capacities are integer (and finite) then throughout the algorithm the flow is integer. So at each step the flow decreases by at least 1. Since the cost of any flow is at least

$$N = \sum_{c_e < 0} u_e c_e,$$

#steps is at most $c(f) - N$, where f is the initial flow. Therefore, Alg. 3.2 terminates and runs in pseudo-polynomial time.

If there are multiple negative cycles in G_f, Alg. 3.2 does not specify which one to choose. By choosing smartly one can obtain a polynomial-time algorithm. There are two ways to do this:

- (By Francisko Barahona and Eva Tardos) Choose the cycle(s) C with the smallest $\Delta c = \delta^C c(C)$, where δ^C is the largest δ s.t. f^δ satisfies (1). Cancel several node-disjoint cycles at once.
- (By Andrew Goldberg and Robert Tarjan) Select a cycle C with minimal $\frac{c(C)}{|C|}$ (a *min mean-cost* cycle). This approach has the Edmonds-Karp algorithm as a special case.