CS 684: Approximation and Network Algorithms	**9/26/01**
Min-cost flows #2	**Scriber: Mark Sandler**

1 Correctness of simple Min-cost flow algorithm

Let us first remind the definitions we were using last time. We have graph $G = (V, E)$, for every edge $e = (v, w) \in E$ we have cost c_e and capacity u_e, for every node v we have demand b_v, so that $\sum_{v \in V} b_v = 0$. Also for every flow function f we have defined a residual graph, $G_f = (V, E_f)$, where edge $e = (v, w) \in E_f$, if and only if one of the following is true:

- $e = (v, w) \in E$ and $f_e < u_e$, then it has capacity $u_e - f_e$ and cost $c_{vw} = c_e$,
- $e = (w, v) \in E$ and $f_e > 0$, then it has capacity f_e and cost $c_{vw} = -c_e$.

Last time we discussed simple algorithm for Min-cost flows, and it was based on elimination of negative cost cycles from residual graph. On this lecture we prove that the residual graph G_f does not contain negative cost cycles if and only if the produced flow is min-cost. Clearly, "only if" direction is clear: If G_f has a negative cost cycle C then we can reduce cost of f by sending additional flow along cycle C (no constraints will be violated). And thus f could not be a min-cost flow.

Now let us prove "if" direction. First of all, if there is no negative cost cycles in G_f then we can find distances between pairs of nodes. Let us add an additional node s and connect s with all other nodes to the graph G, and edges of form (s, v) with some cost M. Let $p[v]$ denotes the distance (in terms of costs of edges) from s to v. Now, since there is no negative cycles, function p is defined and $\forall (v, w) \in E_f$, $p(w) \le p(v) + c_{vw}$. Thus if edge (v, w) is a forward edge in G_f then we have $p(w) \le p(v) + c_e$, otherwise $p(v) \le p(w) - c_e$.

Definition 1.0.1 *We say that given f and p are at equilibrium if for every edge $e = (v, w)$ the following conditions are satisfied*

$$\begin{array}{lll} f_e < u_e & \Rightarrow & p(w) \le p(v) + c_e \\ f_e > 0 & \Rightarrow & p(w) \ge p(v) + c_e \end{array}$$

By the definition of the equilibrium, one can see that if G_f has no negative cost cycles and $p[v]$ cost of the cheapest path from s to v then f and p are at equilibrium. (Indeed, the left hand sides of conditions in the definition are exactly the condition of existence backward and forward edges between v and w.)

There is a clear economic interpretation for the equilibrium. If f corresponds to the flow of goods, $p(v)$ is a price of goods in node v, and c_{vw} corresponds the cost of shipping from v to w. Then f and p are in equilibrium, if every profitable shipping is made and there is no shippings with loss of money (e.g. if $p(w) > p(v) + c_e$ then there is a maximal flow on the edge e, since one can get profit from shipping from v to w, and if $p(w) < p(v) + c_e$ there is no flow on edge e, since it is unprofitable).

Now we will prove the theorem which connects equilibrium and min-cost flows:

Theorem 1.0.2 *For every graph G and any feasible flow function* $\mathbf{f}$ *the following statements are equivavlent:*

1. $\mathbf{f}$ *is min cost flow*
2. *Graph $G_{\mathbf{f}}$ has no negative cost cycles*
3. *There exists prices p such that* $\mathbf{f}$ *and p are at equilibrium*

Proof: (1) $\Rightarrow$ (2) This is obvious (if there is a min-cost cycle, we can decrease cost of $\mathbf{f}$ and thus $\mathbf{f}$ is not min-cost flow.)

(2) $\Rightarrow$ (3) This immediately follows from the discussion above.

(3) $\Rightarrow$ (1) Consider graph G^p which is obtained from G, with cost function defined as follows:

$$c^p_{vw} = p(v) + c_{vw} - p(w)$$

We call c^p a reduced cost function. Now we shall prove that $\mathbf{f}$ is a min-cost flow in G^p. Clearly any feasible flow in G is also feasible in G^p. Since, $\mathbf{f}$ and p at equilibrium, for any edge $e \in E$, if $c^p_e > 0$ then $\mathbf{f}_e = 0$, if $c^p_e < 0$ then $\mathbf{f}_e = u_e$. So we can write cost of $\mathbf{f}$ as

$$\sum_{c_e<0, e\in E} c^p_e u_e,$$

which is a lower bound for *any* flow in G^p and therefore $\mathbf{f}$ is min-cost flow in G^p.

Now the price of any flow f in G can be expressed in terms of reduced cost function and prices:

$$\sum_{e\in E} c_e f_e = \sum_{e\in E}((p(w) - p(v) + c^p_{vw})f_e = \sum_{e\in E} c^p_e f_e - \sum_{e\in E}(p(v) - p(w))f_e = \sum_{e\in E} c^p_e f_e - \sum_{v\in V} b_v p(v)$$

The last transition is because, for every edge $e = (v, w)$, we've added to the sum term $p(v)f_e$ and substracted term $p(w)f_e$, thus for every node we've counted exactly excees of the flow in this node multiplicated by $p(v)$. Now we can rewrite this as follows:

$$\sum_{e\in E} c_e f_e = \sum_{e\in E} c^p_e f_e + const$$

Where *const* does not depend on function f, therefore min cost flow in G^p is also min cost in G and this proves that f is min-cost flow in G.

■

2 Primal-Dual algorithm

The idea of this algorithm is to find any f and p at equilibrium and then modify them in order to get a feasible flow in f, but still keeping the equilibrium. Then flow f is desired min-cost flow.

1. First part (to find f and p in equilibrium) is easy - just take $p(v) = 0$ and define f_e as follows:

$$f_e = \begin{cases} 0 & \text{if} \quad c_e \geq 0 \\ u_e & \text{if} \quad c_e < 0 \end{cases}$$

   Clearly f and p are at equilibrium.

2. We say, that node v is extra supply node iff $f^{in}(v) - f^{out}(v) > b_v$ and analogously that node v is extra demand node if $f^{in}(v) - f^{out}(v) < b_v$. Now, let V_s denotes set of extra supply nodes and V_d denote set of extra demand nodes. As in the proof above we define reduced cost function $c^p_{vw} = p(v) + c_{vw} - pw$. First of all, if we change flow on any edge which has reduced cost zero, we will remain at the equilibrium state.[1] Thus if there exists a zero reduced cost path from any node $v \in V_s$ to any node $w \in V_d$ in G_f, we can safely push some flow along it and this operation will keep the equilibrium state but will decrease total extra supply.

3. Suppose we cannot find desired zero-cost reduced path. In our ecominic interpretation this corresponds to the graph where there is no way to avoid loses if someone willing to ship from supply to demand nodes. Thus there is clear economic solution: raise prices in demand nodes. In more formal way, let S denotes set of zero-cost reachable nodes. Suppose we have raised prices in all nodes in V/S by some tiny amount ε. Clearly this operation will not affect the equilibrium for all internal edges for S and V/S for any ε.

[1]Indeed, edge which has reduced cost zero, does not have any restriction on the value of flow in the equilibrium definition.

Now, let us look at the cross edges. Any zero reduced cost edge from V/S to S, has zero amount of flow in it (because otherwise we can extend S), price raising in V/S makes reduced cost positive and thus equilibrium remains. Analogously any zero-reduced cost edge from S to V/S has maximal amount of flow and price raising in S will make the reduced price negative and thus the equilibrium conditions are also satisfied. And now we can choose ε, so that reduced cost function will not change its sign on the rest of the cross edges. In fact we should choose ε equal to:

$$\varepsilon = \min\big[(\min_{\substack{(v,w) \in E_f, \\ (v,w) \in S \times V/S, \\ c^p_{vw} > 0}} c^p_{vw}, \min_{\substack{(v,w) \in E_f, \\ (v,w) \in V/S \times S, \\ c^p_{vw} < 0}} -c^p_{vw}\big]$$

Then after this price raising we can extend S by at least one node. If there is still no demand nodes in S we can continue raising prices otherwise return to reducing excesses.

So the algorithm looks like this:

Algorithm 2.0.3

1. *Take f and p at equilibrium*
2. *Repeatedly choose zero-cost reduced path from extra supply node, to extra-demand node. If no path exists raise prices.*
3. *If no extra-supply nodes left, terminate the algorithm*

The amount of flow we have pushed on step 2 is the minimum of the following values: the residual capacity of that zero reduced-cost path and on the total excess and deficit in terminal nodes.

2.1 Correctness

- If generated f is a feasible flow, then it is a min-cost flow. (Because we keep f and p at equilibrium)
- If f is not feasible then one of two actions in step 2 applies. Thus if the algorithm ever terminates it produces the correct answer.
- However the time bounds for the algorithm are terrrible, but on the next lecture we will show how to improve them.