College of Computer Science
Northeastern University
COM 3810: Distributed Network Algorithms

Lecture 3
April 10, 2002
Scribe: Lujun Jia

Local algorithms for network flow and multicommodity flow

Lecture Outline:

- Preflow-push algorithms for network flow.
- The concurrent multicommodity flow problem and the local balancing algorithm.

1 Maximum-flow problem

1.1 Definition

In the Maximum-flow problem, we are given a flow network $G = (V, E)$ with source s and sink t, and we are required to find a flow of maximum value from s to t.

A flow network $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ is assigned a nonnegative capacity $c(u, v)$. A *flow* is define as a real-valued function $f : V \times V \rightarrow \mathcal{R}$, such that it satisfies the following three properties:

- **Capacity constraint:** For all $u, v \in V$, we require $f(u, v) \leq c(u, v)$.
- **Flow conservation:** For all $u \in V - \{s, t\}$, we require $\sum_{v \in V} f(u, v) = 0$.
- **Skew symmetry:** For all $u, v \in V$, we require $f(u, v) = -f(v, u)$.

Lemma 1 *The maximum flow is bounded by the capacity of any cut of G.*

Theorem 1 **(Max-flow min-cut theorem)** *If f is a flow in 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.*

The problem can be solved by the Ford-Fulkerson algorithm. The Ford-Fulkerson Algorithm keeps searching the augmenting paths, updating the flow on each edge, and re-computing the network.

1.2 Preflow algorithm

The augmenting path algorithm pushes flow along each augmenting path, which takes a bottom-up approach. Here we introduce a preflow-push algorithm that solves this problem in a different way. It pushes the flow out from the source at the very beginning, and then adjusts the flows by letting the "fluid" flow to the sink and any excess being returned to the source. In practice, this algorithm performs better than the augmenting path algorithms.

1.2.1 Algorithm definition

We re-define flow to be a *preflow*, since the flow conservation constraint is modified to be

- **Preflow conservation constraint:** $e(v) = \sum_u f(u,v) \geq 0$, for all $v \in V - \{s,t\}$.

Here, $e(v)$ is the *excess* at node v. Note that $e(t) \geq 0$, since no edges leave t; $e(s) \leq 0$, and s is the only node with negative excess. We assign a *height* $h(v)$ to each node v such that the following conditions are satisfied:

1. $h(t) = 0$
2. $h(u) \leq h(v) + 1$, for all directed edge (u,v) in G_f.
3. $h(s) = n$.

The goal is to push flows toward the sink. We determine the direction of *push* from the heights of neighboring nodes: flow can only be pushed from higher nodes to lower nodes. As we will see in a moment, the algorithm proceeds as long as there is an node with excess. The preflow-push algorithm proceeds by repeating the following steps:

1. For each u, if $e(u) > 0$ and $\exists v$ such that $(u,v) \in G_f$, $h(u) = h(v) + 1$;
2. **Push:** $d_f(u,v) = \min(e(u), c_f(u,v))$ from u to v;
3. Update excess: $e(u) \leftarrow (e(u) - d_f(u,v)$; $e(v) \leftarrow e(v) + d_f(u,v)$;
4. Update flow: $f(u,v) \leftarrow f(u,v) + d_f(u,v)$; $f(v,u) \leftarrow -f(u,v)$;
5. For each u, if $e(u) > 0$ and $\forall v$ such that $(u,v) \in G_f$, $h(u) \leq h(v)$;
6. **Relabel (Lift):** $h(u) \leftarrow \min(1 + h(v))$, for all $(u,v) \in G_f$.

If for any node u, $e(u) = 0$, the algorithm terminates.

1.2.2 Analysis of the preflow algorithm

We first show the correctness of the algorithm.

Lemma 2 **(Non-Stop)** *For any $u \in V$, if $e(u) > 0$, then either a push or a relabel operation applies to u.*

Proof: For each node u with $e(u) > 0$, if the push operation does not apply to it, the lift applies, since the height function comparison can have only two cases. ■

Lemma 3 **(Height never decreases)** *For any $u \in V$, $h(u)$ never decreases.*

Proof: Since the only cases we modify the height of node u is when node u is lower than all its neighbor, where the lift operation actually increase $h(u)$ by at least 1. ■

Lemma 4 *During the execution of the preflow-push algorithm, $h(u) \leq h(v) + 1$ for all $(u,v) \in G_f$.*

Proof: This can be shown by an induction on the number of basic operations performed. We can show that both lift and push do not break the rule. ■

Lemma 5 *There is no path from s to t in G_f.*

Proof: By the property of the height function, $h(u) \leq h(v)+1$ for each edge $(u,v) \in G_f$. If there is a path p from s to t, then it spans at most $n-1$ edges. Sum up the height inequalities on those edges, we have $h(s) \leq h(t)+n-1 = n-1$, but $h(s) = n$. This leads to a contradiction. ■

Theorem 2 *The preflow-push algorithm terminates with the maximum flow on G.*

Proof: Since each node has an excess of 0 and there is no path from s to t in G_f, by the max-flow min-cut theorem, the algorithm terminates with the maximum flow. ■

Now, we analyze the preflow-push algorithm.

Lemma 6 *For any $u \in V$, there is a simple path from u to s in G_f.*

Proof: We denote the part of V that u can reach as U, and let $\overline{U} = V - U$. Assume for the sake of contradiction that $s \notin U$. We can show by contradiction that $f(\overline{U}, U) \leq 0$, since each edge from $\overline{U}$ to U has non-positive flow. Consequently, we can show that the overall excess in U is non-positive. Since we know any node has non-negative excess except for s, thus, u has an excess of 0, which contradicts the fact that u has a position excess. ■

Lemma 7 *At any time during the execution of the algorithm, we have $h(u) \leq 2|V|-1$ for all $u \in V$.*

Proof: From Lemma 6 and the height function properties, we have that for each node u, $h(u) \leq h(s)+|V|-1 = 2|V|-1$. Since any simple path has at most $|V|-1$ edges. ■

Give Lemma 7, we have the following lemma:

Lemma 8 *During the execution of the preflow-push algorithm,*

- *The number of* **lift** *operations is at most $(2|V|-1)(|V|-2)$.*
- *The number of* **saturating push** *operations is at most $2|V||E|$.*
- *The number of* **non-saturating push** *operations is at most $4|V|^2(|V|+|E|)$.*

Proof: Consider the lift operation. It is clear that only $|V|-2$ vertices may be lifted, since the source and sink never overflow. From Lemma 7, $h(u) \leq 2|V|-1$ for any $u \in V$. Thus, we have the number of lift operations is at most $(2|V|-1)(|V|-2) < 2|V|^2$.

Consider the saturating push operation. The necessary condition for a saturating push along edge (u,v) is $h(u) = h(v)+1$. In order for this operation to be performed again, there must be a push from v to u, which can only happen when $h(v)$ has been increased by at least 2. Similarly, $h(u)$ must increase by at least 2 in order for another push from u to v. At the moment of the first push between u, v, $h(u)+h(v) \geq 1$. From Lemma 7, $h(u)+h(v) \leq 4|V|-2$. Since we have shown that each saturating push corresponds to an increase of at least 2 in $h(u)+h(v)$, the total number of saturating pushes is at most $2|V|-1$. This give the second part of this lemma.

Now we consider the non-saturating pushes. We define a potential function $\Phi = \sum_{v \in X} h(v)$, where X is the set of overflowing nodes. Initially, $\Phi = 0$. Consider the three possible ways that Φ can possibly be changed:

1. **Lift operation:** In this case, the set X remains the same. But from Lemma 7, for each node $v \in X$ that is lifted, $h(v)$ is increased by at most $2|V|$.

2. **Saturating push:** In this case, each node remains the same height. But after a saturating push from u to v, v can possibly be added to X, which incurs an increase of at most $2|V|$ in Φ.

3. **Non-saturating push:** Each node remains the same height. But after a non-saturating push from u to v, set $X \leftarrow X - \{u\} + \{v\}$. Thus, Φ is decreased by at least 1, since $h(u) = h(v) + 1$ before this operation.

From the first two paragraphs in this proof, we have: the total amount of increase in Φ is at most $2|V|(2|V|^2) + 2|V|(2|V||E|) = 4|V|^2(|V| + |E|)$. Since $\Phi \geq 0$ during the execution of the algorithm, the amount of decreases, and therefore the total number of non-saturating pushes is at most $4|V|^2(|V| + |E|)$. This completes the proof. ■

Given Lemma 8, we have the follow:

Theorem 3 *During the execution of the preflow-push algorithm, the number of basic operations is* $O(|V|^2|E|)$.

2 Multicommodity flow

The multicommodity flow problem consists of shipping several different commodities from their respective sources to their sinks through a common network so that the total flow going through each edge does not exceed its capacity. Associated with each commodity i is a *demand* d_i. We then would like to ask the following two questions:

1. Given the demands, is there a feasible way to satisfy all of them?

2. Given the demands and a *ratio* z, is there a way to ship at least z fraction of each demand?

The second question is actually the *concurrent flow problem*, which is what we will study. We consider a simple *edge balancing* algorithm due to Awerbuch and Leighton. In the following, we describe the algorithm. We will study it in detail in the next class.

The algorithm maintains a queue for each commodity i at the end of each edge e, more precisely, two variable $q_i(head(e))$ and $q_i(tail(e))$ is maintained for each i and e. The algorithm consists of the following four phases:

- **Phase I:** Add $(1+\epsilon)d_i\frac{1}{\delta_i}$ to each of the δ_i queues of the source, where d_i is the demand of commodity i, δ_i is the number of edges incident to the source for commodity i. Thus, a total of $(1+\epsilon)d_i$ units of commodity i is added to the source.
- **Phase II:** Push flow across each edge so as to balance the queues as much as possible. **Balance** is indicated as follows: define $\Delta_i(e) = q_i(tail(e)) - q_i(head(e))$; push f_i flow of commodity i so as to maximize $\sum_i f_i(\Delta_i(e) - f_i)d_i^{-2}$ subject to the capacity constraint.
- **Phase III:** At the sink, remove the commodity immediately.
- **Phase IV:** Balance the queues for each commodity at each node so that the queues are equal-sized.

The way that the queue are balanced between nodes can be think of as computing a potential function: the more balanced the queues are, the more decrease of the potential function.