CS273 Foundations of Parallel and Distributed Computation **Spring 2001**

Lectures 10,11: February 20, 22, 2001

Lecturer: Satish Rao *Scribe: Kunal Talwar*

1 Outline

We start where we left off in the last lecture, i.e. the linear program for the path selection problem. We first show how to round the linear program optimum to get a solution with a good approximation guarantee. Then we look at a simpler algorithm that does not use the power of linear programming.

2 Path Selection Problem

2.1 Recap

In the last class we started looking at the path selection problem :
Given a graph $G = (V, E)$, and pairs of vertices $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$, find paths connecting s_i to t_i for all i, such that the maximum number of paths that use any edge (i.e. the *congestion*) is minimized.
We can write this problem as a linear program, where variable $f^i(e)$ is 1 if edge e lies on the path from s_i to t_i, and 0 otherwise. In the relaxation of this integer linear program, $f^i(e)$ denotes the flow of commodity i along edge e. We minimize the maximum congestion C, and we add constraints for flow preservation at each vertex.

$$\text{minimize } C$$

subject to

$$\begin{array}{rcll}
\sum_i f^i(e) & \leq & C & \forall e \in E \\
\sum_{e=(u,v)} f^i(e) - \sum_{e=(v,w)} f^i(e) & = & \begin{cases} 0, & \text{if } v \in V \setminus \{s_i, t_i\}; \\ -1, & \text{if } v = s_i; \\ 1, & \text{if } v = t_i. \end{cases} & \forall i \\
f^i(e) & \geq & 0 & \forall i \forall e \\
f^i(e) & \leq & 1 & \forall i \forall e
\end{array}$$

We can solve the linear program in polynomial time, and get an optimal solution to the linear relaxation of the path selection problem.

2.2 Another LP

An alternate formulation of the problem as a linear program is the following. Let $f^i(p)$ be the flow of commodity i along a path p in $\mathcal{P}_i$, the set of paths from s_i to t_i. While $f^i(p)$ should be a $0-1$ variable in our solution of the path selection problem, we would relax this integrality constraint.

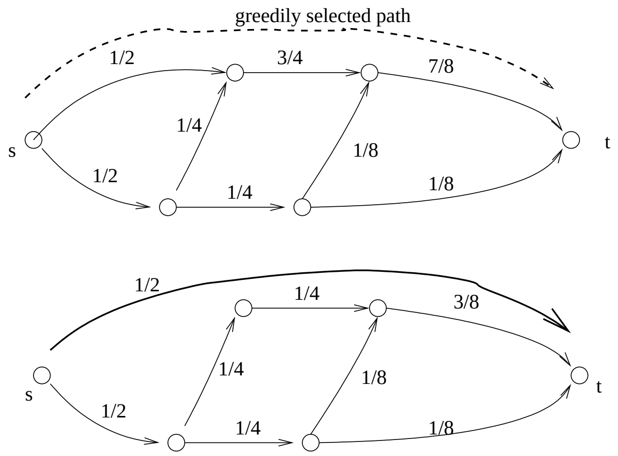

Figure 1: One step for deriving $f(p)$'s from $f(e)$'s

$$\begin{array}{lrcll} & \text{minimize} & C & & \\ \text{subject to} & & & & \\ & \sum_{i,p:e\in p,p\in\mathcal{P}_i} f^i(p) & \leq & C & \forall e \\ & \sum_{p\in\mathcal{P}_i} f^i(p) & = & 1 & \forall i \end{array}$$

Note however that number of paths between two nodes can be exponential in the number of vertices and thus the number of variables in the above linear program is exponential. Thus even writing down the linear program will take an exponential amount of time. However, there is a method due to Khachiyan, called the ellipsoid method that can solve a linear program with exponentially many variables and polynomially many constraints (under certain conditions, of course). This method works in this case, and we can actually solve the above linear program in polynomial time.

2.3 Equivalence of the LPs

On the other hand, we can also use an optimal solution of the first linear program to get an optimal solution to the second. Let $f^i(e)$ be an optimum for the first LP. Consider a particular i and look at the directed graph of edges with non-zero $f^i(e)$. While this graph can have cycles, we get rid of all cycles to get a directed acyclic graph. Now find a path p from s_i to t_i greedily. Assign $f^i(p)$ to be the minimum $f^i(e)$ along this path and deduct this amount from each of the edges on this path. (See figure 1) Recur. Since each step deletes at least one edge, we terminate in m steps. Moreover we find a solution that has only polynomially number of nonzero variables. It is clear that we find an optimal assignment of $f^i(p)$'s.

2.4 Randomized Rounding

Recall that the linear program we wrote was a relaxation of the original path selection problem, and thus the solution f_{opt} we get can be fractional. To get a solution of the path selection problem, we need to round the fractions to integers. Note that

$$\sum_{p \in \mathcal{P}_i} f^i_{opt}(p) = 1$$

For every i, we choose path p_k with probability $f^i_{opt}(p_k)$. Note that for a particular i, we choose exactly one path p_k, and the choices for different i are independent. Let $\hat{f}$ be the resultant integral solution. The congestion on an edge e is $\sum_{i,p:e \in p, p \in \mathcal{P}_i} \hat{f}^i(p)$.

Lemma 2.1. (Chernoff bounds) Let $X_1, X_2, \ldots, X_k$ be k independent 0-1 random variables. Let $X = \sum_i X_i$, and let $E[X] \leq \mu$. Then, for $\delta \leq 2e - 1$

$$Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\mu\delta^2}{4}}$$

Further, for any m,

$$Pr[X \geq m] \leq (\frac{e\mu}{m})^m$$

Using the above lemma we now show that with high probability, the congestion in $\hat{f}$ is small.

Consider an edge e. Consider all paths p that go through edge e. Congestion on this edge e is given by $\sum_{i,p:e \in p, p \in \mathcal{P}_i} \hat{f}^i(p) = \sum_i \sum_{p:e \in p} \hat{f}^i(p)$. Now for a fixed i, let $X_i = \sum_{p:e \in p \in \mathcal{P}_i} \hat{f}^i(p)$. Then X_i is a 0-1 random variable with expectation $\sum_{p:e \in p \in \mathcal{P}_i} f^j_{opt}(p)$. Here X_i's are independent and $E[\sum_i X_i] = \sum_i \sum_{p:e \in p \in \mathcal{P}_i} f^i_{opt}(p) = \sum_{i,p:e \in p; p \in \mathcal{P}_i} f^i_{opt}(p) \leq C_{opt}$.

Consider the case when $C_{opt} \gg 4c \log n$. Let $\delta = \sqrt{\frac{4c \log n}{C_{opt}}}$. Note that by assumption, $\delta \leq 1 < 2e - 1$. Chernoff bound then tell us that,

$$\begin{aligned} Pr[\sum_i X_i \geq (1+\delta)C_{opt}] &\leq e^{\frac{-4c \log n C_{opt}}{4C_{opt}}} \\ &= n^{-c} \end{aligned}$$

Thus with high probability (i.e. $> 1 - \frac{m}{n^c}$), the maximum congestion on any edge is at most $C_{opt} + \sqrt{C_{opt} 4c \log n}$, which is better than a constant factor approximation for large C_{opt}.

Using the other version of Chernoff bound, we can similarly prove that

- If $C_{opt} = \Theta(\log n)$, then with high probability, maximum congestion in $\hat{f}$ is $O(C_{opt})$.
- If $C_{opt} = O(1)$, then with high probability, maximum congestion is $O(\frac{\log n}{\log \log n})$.

3 A simpler algorithm

In the last section, we presented an LP rounding algorithm for the path selection problem. In this section, we describe a simpler algorithm that does not use the power of linear programming. Recall that the path selection problem was : Given $G = (V, E)$, and pairs of vertices (s_i, t_i), we want to find paths from s_i to t_i such that the congestion is minimized.

If we routed each commodity along its shortest path, we would be minimizing the average congestion, and thus this seems to be a good approach. However, consider the example of figure 2. In this example, the

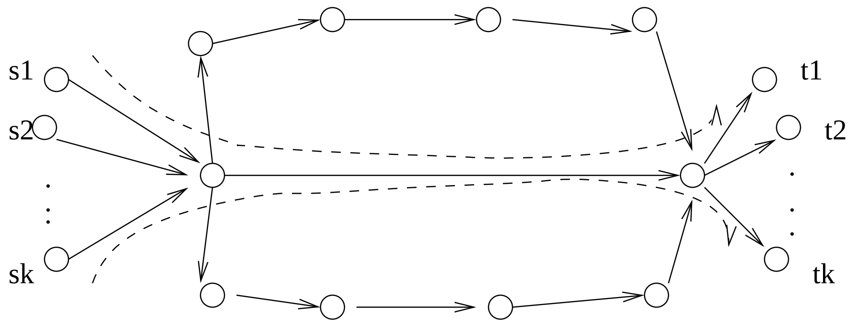

Figure 2: Example where shortest path routing can be bad

maximum congestion is large even though the average congestion is minimized for a straight forward shortest path routing.

We fix this by making an edge that is heavily congested longer. In particular, we assign lengths to edges depending on their congestion as follows.

$$d(e) = c^{f(e)}$$

Using this distance function, we can route along the shortest path. After we are done, we reconsider our routes. We use the following algorithm :

Algorithm simplepathselect;
- **1. for** every pair (s_i, t_i) **do**
  - **1.1.** find the shortest $s_i - t_i$ path and route along it
  - **1.2.** update the length function d
- **2. while** there is some (s_i, t_i) such that $d(p_i) \geq 2cd(s_i, t_i)$, where p_i is the current path connecting s_i to t_i. **do**
  - **2.1.** reroute commodity i along the new shortest path
  - **2.2.** update the length function d

end.

3.1 Convergence of the algorithm

From the algorithm as presented above, it is not clear why the second step of the algorithm should converge. We now show that the algorithm converges.

Lemma 3.1. *In each iteration of step 2. of algorithm* simplepathselect*,* $\sum_{e \in E} d(e)$ *decreases.*

Proof. Suppose in an iteration of step 2, the path p_i from s_i to t_i is replaced be the shortest path p'_i,(see figure 3) because $d(p_i) \geq 2cd(s_i, t_i) = 2cd(p'_i)$. Let the length function be updated to d' in step 2.2, due to the flow being changed to f'. Then we have,

$$f'(e) \quad = \quad \begin{cases} f(e), & \text{if } e \notin p_i \text{ and } e \notin p'_i; \\ f(e)+1, & \text{if } e \notin p_i \text{ and } e \in p'_i; \\ f(e)-1, & \text{if } e \in p_i \text{ and } e \notin p'_i; \\ f(e), & \text{if } e \in p_i \text{ and } e \in p'_i. \end{cases}$$

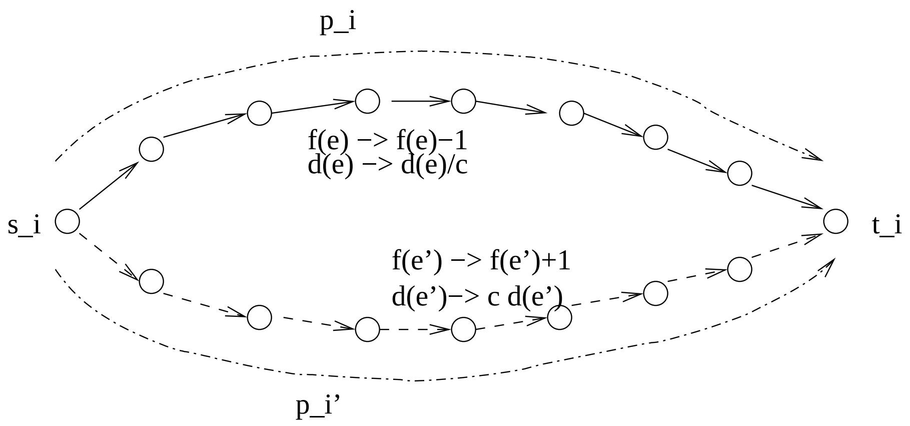

Figure 3: Convergence of the algorithm: Path p_i is replaced by the shortest path p_i' in an iteration of step 2

and therefore

$$d'(e) \quad = \quad \begin{cases} d(e), & \text{if } e \notin p_i \text{ and } e \notin p_i'; \\ cd(e), & \text{if } e \notin p_i \text{ and } e \in p_i'; \\ \frac{1}{c}d(e), & \text{if } e \in p_i \text{ and } e \notin p_i'; \\ d(e), & \text{if } e \in p_i \text{ and } e \in p_i'. \end{cases}$$

Thus

$$\begin{aligned}
& \sum_e d'(e) - \sum_e d(e) \\
= \quad & \sum_{e \in (p_i \cap p_i') \cup (p_i^c \cap p_i'^c)} (d'(e) - d(e)) + \sum_{e \in p_i^c \cap p_i'} (d'(e) - d(e)) + \sum_{e \in p_i \cap p_i' c} (d'(e) - d(e)) \\
= \quad & 0 + (c-1) \sum_{e \in p_i^c \cap p_i'} d(e) - (1 - \frac{1}{c}) \sum_{e \in p_i \cap p_i'^c} d(e) \\
\leq \quad & (c-1) \sum_{e \in p_i^c \cap p_i'} d(e) - (1 - \frac{1}{c}) \sum_{e \in p_i \cap p_i' c} d(e) + ((c-1) - (1 - \frac{1}{c})) \sum_{e \in p_i \cap p_i'} d(e) \\
= \quad & (c-1) \sum_{e \in p_i'} d(e) - (1 - \frac{1}{c}) \sum_{e \in p_i} d(e) \\
\leq \quad & (c-1) d(p_i) - (1 - \frac{1}{c}) d(p_i') \\
\leq \quad & (c-1) d(p_i) - (1 - \frac{1}{c}) 2cd(p_i) \\
= \quad & -(c-1) d(p_i) \\
< \quad & 0
\end{aligned}$$

$\square$

Thus the algorithm converges. In fact, it can be shown that $\sum_e d(e)$ decreases by at least a factor of $(1 - \frac{1}{m})$ and hence the algorithm terminates in polynomially many rounds.

3.2 How good is the solution?

Now we analyze how far from the optimal can a solution output by algorithm *simplepathselect* be.

Think of the *volume* of the network as the sum of the capacities of the edges. If each edge is assumed to have capacity C_{opt}, then the total volume is $C_{opt}|E|$. We claim that $C_{opt}|E| \geq \sum_i l(s_i, t_i)$. The intuition is that any solution would route one unit of flow from s_i to t_i and thus would use up at least $l(s_i, t_i)$ amount of volume. Thus the total volume of the network must at least be equal to $\sum_i l(s_i, t_i)$.

A formal proof follows. If there exists a flow f such that $max_e f(e) \leq C_{opt}$, then $\sum_e f(e) \leq \sum_e (max_e f(e)) \leq C_{opt}|E|$. Moreover flow f can be decomposed into paths as in section 2.3. Then for each edge e, $f(e) = \sum_{p:e \in p} f(p)$. Hence,

$$
\begin{aligned}
C_{opt}|E| &\geq \sum_e f(e) \\
&= \sum_e \sum_{p:e\in p} f(p) \\
&= \sum_p \sum_{e\in p} f(p) \\
&= \sum_p l(p) f(p) \\
&= \sum_i \sum_{p\in \mathcal{P}_i} l(p) f(p) \\
&\geq \sum_i \sum_{p\in \mathcal{P}_i} l(s_i.t_i) f(p) \\
&= \sum_i l(s_i.t_i) \sum_{p\in \mathcal{P}_i} f(p) \\
&= \sum_i l(s_i, t_i)
\end{aligned}
$$

More generally, given any length function $d : E \to \mathbf{R}$, we can define the volume as the sum over all edges of the capacity times the length of the edge $Cd(e)$. An analysis similar to the one above can still be done, and we show below that if there is a flow f of value C_{opt} then for any length function d, $C_{opt} \sum_e d(e) \geq \sum_i d(s_i, t_i)$:

$$
\begin{aligned}
C_{opt} \sum_e d(e) &\geq \sum_e f(e) d(e) \\
&= \sum_e \sum_{p:e\in p} f(p) d(e) \\
&= \sum_p \sum_{e\in p} f(p) d(e) \\
&= \sum_p d(p) f(p) \\
&= \sum_i \sum_{p\in \mathcal{P}_i} d(p) f(p)
\end{aligned}
$$

$$\begin{aligned}
&\geq \sum_i \sum_{p \in \mathcal{P}_i} d(s_i.t_i) f(p) \\
&= \sum_i d(s_i.t_i) \sum_{p \in \mathcal{P}_i} f(p) \\
&= \sum_i d(s_i, t_i)
\end{aligned}$$

Thus for any length function d,

$$C_{opt} \geq \frac{\sum_i d(s_i, t_i)}{\sum_e d(e)}$$

Now we return to the algorithm *simplepathselect*, and use the above result with the length function d at the end of the algorithm.

$$\begin{aligned}
2cC_{opt} &\geq \frac{2c \sum_i d(s_i, t_i)}{\sum_e d(e)} && (1)\\
&> \frac{\sum_i d(p_i)}{\sum_e d(e)} && (2)\\
&= \frac{\sum_i d(p_i) f(p_i)}{\sum_e d(e)} && (3)\\
&= \frac{\sum_e d(e) f(e)}{\sum_e d(e)} && (4)
\end{aligned}$$

Here the second inequality follows from the fact that the algorithm has terminated, and the last step just counts the flow along each path edge by edge. Thus we know that the average congestion is at most $2cC_{opt}$. We want to show that the worst congestion is not too high. Substituting the distance function used in the algorithm, and letting n_l count the number of edges with congestion equal to l (remember that f is an integral flow), the inequality (4) gives

$$\begin{aligned}
2cC_{opt} &\geq \frac{\sum_e c^{f(e)} f(e)}{\sum_e c^{f(e)}} && (5)\\
&= \frac{\sum_l n_l l c^l}{\sum_l n_l c^l} && (6)
\end{aligned}$$

Let C_{max} be the maximum value of l such that $n_l > 0$. Then

$$\begin{aligned}
\sum_{l \leq C_{max} - 2\log_c |E|} n_l c^l &\leq \sum_{l \leq C_{max} - 2\log_c |E|} n_l c^{C_{max} - 2\log_c |E|} \\
&= \sum_{l \leq C_{max} - 2\log_c |E|} \frac{n_l c^{C_{max}}}{|E|^2} \\
&< \frac{1}{|E|} c^{C_{max}}
\end{aligned}$$

Thus,

$$\frac{\sum_l n_l l c^l}{\sum_l n_l c^l} \geq \frac{\sum_l n_l l c^l}{(\sum_{l > C_{max} - 2\log_c |E|} n_l c^l)(1 + \frac{1}{|E|})} \qquad (7)$$

$$\geq \quad (\frac{1}{1+\frac{1}{|E|}})(\frac{\sum_{l>C_{max}-2\log_c|E|} n_l l c^l}{\sum_{l>C_{max}-2\log_c|E|} n_l c^l}) \tag{8}$$

$$\geq \quad (\frac{1}{1+\frac{1}{|E|}}) \min_{l>C_{max}-2\log_c|E|} \frac{n_l l c^l}{n_l c^l} \tag{9}$$

$$\geq \quad \frac{C_{max}-2\log_c|E|}{1+\frac{1}{|E|}} \tag{10}$$

Using (6) and (10),

$$C_{max} \leq 2c(1+\frac{1}{|E|})C_{opt} + 2\log_c|E|$$

Since $|E| \geq 1$,

$$C_{max} \leq 4cC_{opt} + 2\log_c|E|$$

Hence,

- if $C_{opt} \gg \log_c|E|$, $C_{max} \sim 4cC_{opt}$
- if $C_{opt} = O(1)$, $C_{max} \sim O(\log|E|)$. In fact, by choosing $c = \alpha\frac{\log|E|}{\log\log|E|}$ for an appropriate constant α, we can show that $C_{max} \sim O(\frac{\log|E|}{\log\log|E|})$.

Hence algorithm *simplepathselect* gives an approximation guarantee similar to that of the LP randomized rounding algorithm. Moreover this is deterministic, and directly yields integral solutions.

The same algorithm can be modified slightly, and with a tighter analysis, can be shown to give $(1+\epsilon)$ times the optimal soltion for the fractional problem. A similar approach also works for a large class of linear programs.