Lecture 10

1 Overview

Last time we began discussing path selection in arbitrary networks. Again, this is an off-line setting where the networks and routing problems is given to us a priori. Today, we continue the discussion by reviewing and

2 Direct Algorithm for Path Selection. The experts algorithm.

We present an algorithm that gets almost the same bounds but does so without (explicitly) calling linear programming.

First of all, think of shortest path routing. This minimizes the average congestion, but could have problems. E.g., think of a network with many parallel paths one being shorter than the others. We will ignore the longer paths in our routing thus unfairly congesting the shorter path. Thus, we penalize congested edges.

The following is based on this intuition.

Initialise by for each pair route along any path.

Give a cost of $d(e) = (1+\epsilon)^{f(e)}$ where $f(e)$ is the number of paths that use edge e. If any pair is routed along a path whose cost is more than $(1+2\epsilon)$ times the length of the shortest path, change the path to the shortest path.

One good thing would be that the algorithm terminates. We consider the potential function of $\sum_e d(e) = \sum_e (1+\epsilon)^{f(e)}$.

Consider one iteration. One changes a path from p to p'. The flow on all the edges in p is reduced by 1, and the flow on all edges in p' is increased by 1. (The edges on both are unchanged.) Reducing the flow on the path p reduces the potential of those edges, $d(p)$, by a factor of $\alpha = (1+\epsilon)$. Thus, the change in potential due to this is

$$d(p)(1/(1+\epsilon) - 1) \approx d(p)(-\epsilon + \epsilon^2/2),$$

using Taylor's expansion of $1 - \epsilon + \epsilon^2$ for $1/1 + \epsilon$. One should use $\epsilon < 1/4$ just to be safe from the expansion going up with more terms.

Increasing the flow on p' increases the potential, $d(p')$ by a factor of $1+\epsilon$. The difference (increase) in potential due to this is

$$d(p')(1+\epsilon-1) = \epsilon d(p').$$

Since $d(p') < d(p)(1+2\epsilon)$, this is at most

$$\frac{\epsilon d(p)}{1+2\epsilon} \approx \epsilon d(p)(1-2\epsilon).$$

Adding up the two terms, we see that the we get a decrease.

Question 1: Show that the number of iterations is polynomial in the size of the graph and $1/\epsilon$. (Nimar points that you should look at previous years notes. The algorithm has a different termination condition and ensures that one chooses the path that is currently longest in each iteration. Please state the new termination condition.)

Thus, the algorithm stops. What about the quality of the solution at the end? Well, first we need a lower bound on the quality of the optimal. One such lower bound is the average congestion. That is, the minimum congestion cannot be lower than the best average congestion.

How do we minimize the average congestion? Route along shortest paths. Thus, one lower bound is

$$\frac{\sum_i SP(s_i, t_i)}{m},$$

where m is the number of edges, and $SP(s_i, t_i)$ is the length of the shortest path from s_i to t_i.

A generalization, is the following "volume" lower bound.

For any length function $l(e)$, the congestion (max flow on any edge) is at least

$$\frac{\sum_i SP_l(s_i, t_i)}{\sum_e l(e)} \tag{1}$$

We prove this as follows. We examine the quantity consisting of the product over flowpaths of the quantity of flow times the length of the flowpath. That is, the following quantity

$$\sum_{p \in P} f(p) l(p), \tag{2}$$

This can be rewritten as

$$\sum_e f(e) l(e), \tag{3}$$

since an edge contributes $l(e)f(p)$ to a path p that uses edge e and the $\sum_{e \in p} f(p) = f(e)$.

Now for any path that satisfies commodity i, $l(p) \geq SP_l(s_i, t_i)$, and the $\sum_p f(p)$ where p is an s_i to t_i path is at least 1 in any fractional solution of the flow problem.

$$\sum_e f(e) l(e) \geq \sum_i SP_l(s_i, t_i) \tag{4}$$

To finish, we note $f_{max} \sum_e l(e) \geq \sum_e f(e) l(e)$, and we obtain the lower bound in equation 5.

Now, we use the length function $d(\cdot)$ at the algorithms termination to give us a lower bound and show that the maximum congestion at termination is close to this lower bound.

Consider that for our paths

$$\sum_e f(e) d(e) = \sum_{p_i} d(p) \leq (1 + 2\epsilon) \sum_i SP_d(s_i, t_i).$$

Thus,

$$\frac{\sum_e f(e)d(e)}{\sum_e d(e)} \leq (1+2\epsilon)C_{opt}. \tag{5}$$

Plugging in $d(e) = (1+\epsilon)^{f(e)}$ into the left hand side, we get the following expression.

$$\frac{\sum_e f(e)(1+\epsilon)^{f(e)}}{\sum_e (1+\epsilon)^{f(e)}}. \tag{6}$$

We will provide a lower bound for this expression in terms of f_{max} the congestion of the flow.

For the denominator, we note that terms where $f(e)$ is very high are the largest. Indeed, as $f(e)$ becomes very small compared to f_max the contribution to the denomicanotr decreases exponentially. Specifically for any edge e where $f(e) \leq f_{max} - 2\ln|E|/\epsilon$ the total contribution to the numerator is at most $1/|E|^2$. The total contributions of all such edges is at most $1/|E|$. Thus, we can derive a lower bound on the denominator using just heavily loaded (relative to f_{max}) edges. That is

$$\sum_e (1+\epsilon)^{f(e)} \geq (1+1/|E|) \sum_{e:f(e)\geq f_max - 2\ln|E|/\epsilon} d(e). \tag{7}$$

Furthemore, the numerator is at most

$$(f_{max} - 2\ln|E|/\epsilon) \sum_{e:f(e)>f_{max}-2\ln|E|/\epsilon} d(e). \tag{8}$$

The ratio of the above two equations provides a lower bound of $f_{max} - 2\ln|E|/\epsilon$ for equation 5. That is, we get

$$f_{max} - 2\ln|E|/\epsilon \leq (1+\epsilon)C_{opt},$$

or that

$$f_{max} \leq (1+\epsilon)C_{opt} + 2\ln|E|/\epsilon.$$

This is the desired upper bound on the congestion of our solution! Here again, as with randomized rounding, we see that if the optimal solution has large congestion we get very close to it, if it has very small congestion we get within an additive $O(\log n)$ factor. It is striking how similarly this (non random) algorithm behaves to the randomized rounding.

Question 2: Linear programming gets you a fractional answer that is optimal. How would you modify the algorithm above to get a perhaps better congestion, but allow fractional paths?

3 Packing Linear Programs

Consider the following feasibility linear program.

$$Ax \leq b$$

$$\sum_i x_i = 1.$$

$$x \geq 0$$

Here c and x denote length n vectors, b is a length m vector and A is an $m \times n$ sized matrix. We assume that A consists only of positive entries. The previous algorithm for packet routing can be used to approximately solve these programs.

For any positive valued length m vector, and any feasible x, that $\sum_i w_i(b_i - a_i x) \geq 0$, where a_i is the ith row of matrix A. Note that the vector w assigns weights to the equations. This is a possible solution to the dual linear program for those familiar with duality.

Question 3: Show how given a weight vector w, how to find a vector x such that $\sum_i w_i(b_i - a_i x)$ is maximized, where $\sum_i x_i = 1$.

The "experts" algorithm proceeds by first uniformly weighting as follows: let $d_i = 1/n$, and $w_i = d_i / \sum d_i$. Initially, $w_i = d_i$.

We then repeatedly at iteration t, find a solution with the weights w_i via question 3 above and setting

$$d_i^{t+1} = d_i^t (1-\epsilon)^{1+\frac{a_i x^t - b_i}{\delta}}.$$

We will show that $x = \sum_i x^t/T$ is a reasonable solution to the original packing program.

Why? Let

$$\Phi^t = \sum_i d_i^t.$$

Let M be the maximum value of $|b_i - a_i x|$. We have that since there is an x that satisfies all the equations that Φ^t decrease by at least around an $(1-\epsilon)$ factor at each step if ϵ is sufficiently small (much less than $\frac{a_i x^t - b_i}{\delta}$.)[1]

Furthermore,

$$\Phi^t > d_i^t = (1-\epsilon)^{T + \sum_t \frac{a_i x^t - b + i}{\delta}/N}.$$

Thus, we have that

$$(1-\epsilon)^T > (1-\epsilon)^{T + \sum_t \frac{a_i x^t - b_i}{\delta}}/N$$

and

$$N > (1-\epsilon)^{\sum_t \frac{a_i x^t - b}{\delta}}$$

which implies that for each i that

$$\sum_t \frac{a_i x^t - b}{\delta} \leq \delta \log_{1/(1+\epsilon)} N.$$

That is, the violation of an constraint is reasonably small and $\tilde{x}$ is an approximately feasible solution to the linear program.

This is simply a specific case of the multiplicative weights algorithm. Please see the survey by Arora, Hazan, and Kale for a more detailed description.

[1] This follows from the fact that $(1-\epsilon)^k$ is approximately $(1-k\epsilon)$ when $k\epsilon << 1$.