Lecture 8

1 Overview

Last time, we showed that on levelled networks the random priority algorithm routes any set of packets with congestion C on a depth D levelled network in $O(C + D + \log N)$ time.

Today, we will discuss how to route on arbitrary networks and do it in better time. The solution is very "off-line" but illustrates some interesting and fundamental theoretical ideas. We only consider sets of paths where no edge is used twice by the same path.

2 Random delays.

First, we analyse the algorithm of routing by first delaying each packet by a random delay in the range of $\{0, \ldots, \alpha C\}$ and then routing without delay at all.

What? You say. This schedule is not really feasible. Two packets may use the same edge in a time step. To make it real, we could "explode" the schedule by the maximum congestion. That is, we simulate each timestep in the infeasible schedule with X timesteps, where X is the maximum "timestep congestion" on any edge in the infeasible schedule. For example, with no delays we could have simulated the infeasible schedule in time CD.

Thus, the true overhead in time of the "infeasible" schedule is the max congestion. This leads us to proving the following lemma.

Lemma 1
The maximum congestion in the infeasible is (suprise, suprise) $O(\log N)$ with high probability.

For the congestion in a particular timestep on a particular to be greater than k, k packets would have to have chosen each of their delays at exactly one of the αC possibities. That is, the probability for a particular size k set of packets is at most

$$(1/\alpha C)^k$$

. There are N choices of edges, at most $\binom{C}{k}$ choices of packets, and thus at most

$$N\binom{C}{k},$$

possible "bad events." Thus, we can upper bound the probability that any bad event happens by

$$N\binom{C}{k}\left(\frac{1}{\alpha C}\right)^k \leq N\left(\frac{e}{\alpha k}\right)^k.$$

This can be bounded by $1/N^2$, for some $k = O(\log N)$.

The random delays gives us an "off-line" scheduling algorithm that runs in $O((C + D)\log N)$ with high probability.

3 Towards an $O(C + D)$ schedule.

We will essentially use the above scheme recursively. We start with the initial "infeasible" schedule. That is, that in each timestep all the packets that wait for it cross the edge. This schedule has length D. (Exploding it, may require time as high as CD.)

Then, we will recursively place delays at points to reduce the "timestep" congestion. Actually, we will work with something called frame congestion, which is defined as follows relative to an (possibly) infeasible schedule.

DEFINITION 1 *An l-frame is an interval of l-timesteps in a schedule. The l-frame congestion of an edge is the maximum number of packets that use the edge in any l-frame. The l-frame congestion of a schedule is the maximum l-frame congestion of any edge. The relative l-frame congestion is the l-frame congestion divided by l.*

For example, the D-frame congestion of the original infeasible schedule is C and the relative D-frame congestion is D/C. The random delays schedule, for some $f = O(\log N)$ the relative f-frame congestion is constant, or even $1/\alpha+o(1)$. This is, the average congestion over a fairly small. If the relative congestion for constant length frames is constant, than the conversion from a infeasible schedule to a constant schedule has constant cost.

The recursion will proceed to ensure that the relative congestion for smaller and smaller frames is constant.

4 The recursion

We assume that we have a (probably infeasible) schedule with relative T-frame congestion of β for some small constant $\beta < 1$.

We introduce random delays in the range $\{1, \ldots, T/\beta\}$ to all the packets at the beginning of each interval.

That is, a packet waits for a randomly chosen r timesteps at the edge it is at at the beginning of an interval and then proceeds.

To get some intuition, we will first upper bound the probability that the relative t-frame congestion is higher than αt for some particular edge in some particular T-interval. This probability can be upper bounded by

$$\binom{\beta T}{\beta t}\left(\frac{\beta t}{T}\right)^{\beta t} \leq (e\beta)^{\beta t}.$$

This can be made at most $1/T^5$ for some $t = c \log N/\beta$, for some constant c.

This seems good, one needs this to happen in all the D intervals and for all the N edges. Thus, to get this bound to work for all such events one needs $t = \Omega(\log N)$.

We will use a beautiful tool from the study of the probabilistic method.

4.1 The Lovasc Local Lemma.

LEMMA 2

Given a sample space with a set of "bad" events, $A_1, \ldots, A_N$, each of which occurs with probability at most p. Moreover, each event A_i is independent of all but d other events, A_j, there is a sample point where none of the bad events occur as long as

$$4dp < 1.$$

Question 1: Use the Lovasc Local Lemma to prove that any k-regular hypergraph (every node is incident to k hyperedges) with hyperedges of size k (every hyperedge consists of a set of k nodes) has a two coloring of the nodes such that no hyperedge is monochromatic, for $k > 9$. (You may use another larger constant if that is easier.)

For our situation, the bad events are that for some edge the relative t frame congestion is more than β in some t frame. From the previous calculation, we have that

$$p < 1/T^4,$$

for some $t = O(\log T)$.

Now, an event for each edge e is only dependent on the set of random delays that are assigned to packets at that use e during a particular T frame, since each t-frame is "inside" such a T-frame. Two events depend on each other only if a packet that uses one edge during a T-frame also uses another edge. That is, the bad event for a particular edge e for a particular t-frame can only depend on the events for T^2 edges e' (i.e., edges in the paths of packets that use edge e during this T-frame) for $(T/\beta + T)/t \leq 2T/\beta t$ possible t-frames.

That is, we have that

$$d \leq 2T^3/\beta t = O(T^3),$$

for $T >> 1/\beta$.

Thus, we have $4dp < 1$, and we can apply the Lovasc Local Lemma to conclude that there is *some* set of random delays that will produce a schedule with relative t-frame congestion of β.

4.2 The induction

We start with $T_0 = \beta C$, and we have a schedule of relative T_0-frame congestion of β.

Then, at each iteration i, we start with a schedule with relative T_i-frame congestion of β and obtain a scedule with relative T_{i+1}-frame congestion of β, where $T_{i+1} = O(\log T_i)$. We can continue this process as ong $T_i << 1/\beta$. At this point the infeasible schedule can be converted to a feasible schedule at a cost of $1/\beta$.

At each iteration, the schedule is lengthened by a factor of $(1 + 1/\beta)$. Thus, the final schedule has length of $O((1 + 1/\beta)^{\log^* C}(C + D))$, where $\log^* C$ is the number of times one needs to take logs to get to 2.

For example, $\log^* 8$ is 2, $\log^* 2^8$ is 3, $\log^* 2^{2^8}$ is 4. Ok, so it doesn't get very big.

One, can extend these methods to show that there exists a schedule that runs in time $O(C + D)$.

4.3 Towards a polynomial time solution

The above only proved the existence of a good schedule. What about constructing the schedule. Well, for years the Lovasc Local Lemma was a *non constructive* tool. Recently,

Beck showed how to make it constructive. That is, to find a sample point where no bad event occurs in polynomial time.

The idea he used is essentially to analyse a random process on a random graph on the bad events. He noted that if one examines the set of events that actually turn out bad in any random setting of the underlying variables that the connected components in the associated dependency graph (actually a bigger graph based on the dependency graph) are quite small. (This analysis is similiar in some sense to the analysis we did with the power of two choices in bounding the size of the connected components of bins.) Then, he showed that by resetting a few variables one could "fix" each such connected component. He could do this separately for each connected component, he could do some by either exhaustive search (since the components were very small) or using a few levels of recursion. Indeed, this is an oversimplification, but gives a hint at how to do this.

As one might expect the conditions for which his method worked were somewhat weaker than for the Lovasc Local Lemma. For example, the condition $4dp < 1$, was weakened to a condition where $4d^3p < 1$. For our example, this condition could be met. Thus, we can construct a schedule in polynomial time which has length $O(C + D)$.

5 Job-Shop Scheduling.

Consider a set of jobs, $\{J_1, \ldots, J_n\}$ where each needs to be processed in sequence on some set of machines. For example, job J_i could need to be processed on machine l, m and then k.

If each machine is to be processed in unit time and no job is required to not use a machine more than once, this problem can be modelled as a routing problem.

Similar techniques were used for other variations of Job-Shop Scheduling.