College of Computer & Information Science
Northeastern University
CS 7880: Algorithmic Power Tools

Fall 2009
25 September 2009
Scribe: Dimitrios Kanoulas

Lecture Outline:

- Packet Routing Problem
- Application of LLL to Packet Routing

In this lecture, we study the packet routing problem introduced last class and prove using the Lovasz Local Lemma a schedule of length $(c+d)2^{O(log^* d)}$ that achieves queue size at most $(\log d)2^{O(log^* d)}$, where c and d are the congestion and dilation, respectively, of the given set of paths. This is a weaker version of the main result of Leighton-Maggs-Rao that shows the existence of a schedule of length $O(c+d)$ with constant-size queues [LMR94].

1 Packet Routing Problem

Packet routing is the process of selecting paths in a network along which to send packets of data. Routing is performed for many kinds of networks and it has been widely used for parallel computing design.

In this section we introduce the Packet Routing problem and we analyze a greedy algorithm that solve this problem. (This portion of the notes repeats a result also covered in the notes of the previous class.)

Given a network that contains a set of paths, packet routing is the problem where we need to route unit sized packets of data across each path. Each node stores a queue and at each step a packet can either move from one node to an other or wait in a queue. At each step at most one packet can traverse each edge. Upon traversing an edge (x, y) a packet is removed from node's x queue and placed in node's y queue. The size of a queue at a particular step is the number of unit-sized packets that it contains. When we talk about maximum or minimum queue size in a network we will exclude the queue size of the origin and destination nodes.

In packet routing problem we call schedule for a set of packets, the movement timing of packets along their paths.

Problem 1 (Packet Routing). Given a graph $G = (V, E)$ that contains n paths for the packets, determine the shortest total time schedule for the packets, with the smallest possible queue size needed such that all the packets are moved from their origins to their desired destination node.

For our analysis, assume that for every edge $e \in E$, the number of paths that use e is at most c and the length of each path is at most d. We call number c congestion and number d the dilation of the paths.

Naive Algorithm

Given a network that contains a set of n paths for the packets with congestion c and dilation d, route the packets to their destination nodes using queues of size c at each node. It is easy to see that the naive algorithm routes all packets to their destination in at most $O(cd)$ steps, since queues of size c at each node guarantees that each packet can be delayed for at most $O(c)$ steps at each node, for at most d nodes. So the maximum over all packets time to move a packet from its origin to its destination node is $O(cd)$ and the maximum queue size is $O(c)$.

Randomized On-line Algorithm

We introduce a simple randomized on-line algorithm that produce a much better schedule of length $O(c + d \log(nd))$ with queue size $O(\log(nd))$, where c is the congestion, d the dilation and n is the number of packets in the network.

Consider the following algorithm:

- Assign each packet randomly, independently and uniformly a delay chosen from the interval $[0, R]$, where R is a constant that we will determine later.
- In the produced schedule, each packet that is assigned a delay x waits in the queue of its origin node for x time steps and then it moves to its destination without waiting to any other node's queue.

Suppose an invalid schedule where there exists the property that the number of packets traversing each edge simultaneously are at most k. Then it is easy to convert it to a valid schedule, without capacity constraints, by assuming that each time step of the invalid schedule corresponds to k steps of a valid one. Since the length of invalid schedule will be at most $(R + d)$, the length of the valid one will be at most $k(R + d)$.

Analysis of the Greedy Algorithm

We will analyze the invalid schedule that we propose above, since it is straightforward how one can convert it to a valid one. We need to set values for R and k such that the maximum queue size is at most k, with high probability.

We know that a fixed edge f is used by at most c paths and that the probability that k packets go through this edge at a fixed time step t, is $\left(\frac{1}{R}\right)^k$. Thus:

$$\Pr[(\text{Number of packets going through a fixed edge f}) \geq k] \leq \binom{c}{k}\left(\frac{1}{R}\right)^k$$

If we set $R = c$, then:

$$\begin{aligned}\Pr[\text{(Number of packets going through a fixed edge f)} \geq k] &\leq \binom{c}{k}\left(\frac{1}{R}\right)^k \\ &\leq \left(\frac{ec}{k}\right)^k\left(\frac{1}{c^k}\right) \\ &= \left(\frac{e}{k}\right)^k\end{aligned}$$

since we know that $\binom{c}{k} \leq \left(\frac{ec}{k}\right)^k$.
Consider as a bad event, the event where more than k packets go through the fixed edge f at time step t. Then, applying the union bound we have that:

$$\Pr[\text{(No bad event happens)}] \leq \left(\frac{e}{k}\right)^k (nd)(c+d)$$

where (nd) comes from the number of possible choices for the fixed edge f and the $(c+d)$ from the number of choices for the time step t.
We can control the value of k so that the value $\left(\frac{e}{k}\right)^k$ becomes smaller than 1 by setting $k^k \gg nd(c+d)$.

If we now set $R = \frac{\alpha c}{\log(nd)}$, where α is a fixed constant and $k = \beta \log(nd)$, then

$$\begin{aligned}\Pr[\text{Number of packets through f at time step } t \geq k] &\leq \binom{c}{k}\left(\frac{1}{R}\right)^k \\ &\leq \left[\frac{ec}{\beta\log(nd)}\right]^{\beta\log(nd)}\left[\frac{\log(nd)}{\alpha c}\right]^{\beta\log(nd)} \\ &= \left[\frac{e}{\alpha\beta}\right]^{\beta\log(nd)} \\ &= \left[\frac{e}{\alpha}\right]^{\log(nd)}\end{aligned}$$

and by setting $\alpha = 8e$ and applying the union bound we have that:

$$\begin{aligned}\Pr[\text{Any edge at any time has at least k packets}] &\leq \left[\frac{e}{\alpha}\right]^{\log(nd)}(nd)(c+d) \\ &= \frac{nd(c+d)}{n^3d^3} \\ &= \frac{c+d}{n^2d^2} \\ &\leq \frac{1}{n}\end{aligned}$$

If we plug these values back in, the length of final valid schedule is at most $k(R+d) = \log(nd)[\frac{\alpha c}{\log(nd)} + d] = O(c + d\log(nd))$ and achieves this bound with probability at least $1 - \frac{1}{n}$.

The queue size of the valid schedule is at most $\log(nd)$, since no more than $k = O(\log(nd))$ packets pass through any edge at any time step.

2 A schedule of length $O((c+d)2^{O(\log^*(c+d))})$

In the previous section we proved that there exists a randomized algorithm that produce a schedule of length $O(c + d\log(nd))$ using at most $\log(nd)$ queue size with probability at most $1 - \frac{1}{n}$, where c is the congestion, d the dilation and n is the number of packets in the network. In [LMR94] Leighton et al. shows that there exists a $O(c+d)$-step schedule with maximum queue size $O(1)$. In this section we will introduce a simpler schedule of length $(c+d)2^{O(log^*d)}$ and queue size at most $(\log d)2^{O(log^*d)}$.

We first give some definitions we will use:

- A frame T is a sequence of $|T|$ time steps.
- Congestion of a frame, C, is the maximum number of packets crossing any edge in the frame.
- Relative congestion, R, of a frame is defined as the ratio: $R = \frac{\text{Congestion of T}}{|T|}$.

We give a Lemma that we use to prove the main theorem.

Lemma 1. *For every set of packets that have congestion c and dilation d, there exists a schedule of length $O(c+d)$ such that no packet ever waits in queues once it starts and the relative congestion of any frame of length at most $\log d$ is at most 1.*

Proof. For our proof we are going to use the Symmetric Lovasz Local Lemma (LLL).

First, assign an initial random delay to each packet. The delays are chosen randomly, independently and uniformly from the interval $[1, \alpha d]$, where α is a fixed constant that we will determine later when we will apply Lovasz Local Lemma. In the produced schedule, each packet that is assigned a delay x waits in the queue of its origin node for x time steps and then it moves to its destination without waiting to any other node's queue, as we did for the randomized on-line algorithm.

We define by (T, f) a bad event for an edge f in a frame T, where more than $|T|$ packets pass through the edge f, for $|T| \geq \log d$. We will use LLL to prove that there exists a set of delays such that if we choose the delays randomly, independently and uniformly from the interval $[1, \alpha d]$ then with nonzero probability no bad event happens in any frame of size $|T| \geq \log d$, which means that the relative congestion will be at most 1.

To apply LLL, we need to compute the dependences among the bad events and the probability of each individual bad event to occur. We can assume, at the expense of only a factor of two in the bounds, that the congestion c equals the dilation d.

- Computation of dependences among the bad events:
  Consider two bad events: (T_1, f_1) and (T_2, f_2). Since at most c packets are crossing a fixed edge and each of these packets are crossing d other edges, the dependence for these two events is at most $cd = d^2$.
- Computation of probability of each bad event to occur:
  For any frame T and edge f, the probability that the number of packets going through f in

T is at least k is:

$$\begin{aligned}\Pr[\text{Number of packets through } f \text{ in } T \geq k] &\leq \binom{d}{k}\left(\frac{|T|}{\alpha d}\right)^k \\ &\leq \left(\frac{ed}{k}\right)^k\left(\frac{|T|}{\alpha d}\right)^k \\ &= \left(\frac{e|T|}{\alpha k}\right)^k\end{aligned}$$

since a fixed edge f is used by at most d paths and the probability that k packets go through this edge at a fixed time t is $\left(\frac{|T|}{\alpha d}\right)^k$, since at most $|T|$ out of all possible αd delays will send a packet through edge f.

Now we need to choose such a k and α that makes the relative congestion to be at most 1. If we set $k = |T|$ and we let $|T| \geq \log(d)$, then:

$$\begin{aligned}p = \Pr[\text{Number of packets through } f \text{ in } T \geq |T|] &\leq \left(\frac{e|T|}{\alpha|T|}\right)^{|T|} \\ &= \left(\frac{e}{\alpha}\right)^{|T|}\end{aligned}$$

For fixed and sufficiently large α, probability p that more than $k = |T|$ packets go through the fixed edge f at time step t, is getting sufficiently small. In addition the product *epb* becomes less than 1, where p is the probability that each bad event occurs with dependence at most b. Thus, the LLL conditions apply to the Lemma, which means that there is some choice of delays such that the relative congestion in any frame of size at least $\log d$ is at most 1. □

Theorem 1. *For any set of packets that have dilation d and withough loss of generality assume that the congestion c equals d, there exists a schedule of length $(c+d)2^{O(log^* d)}$ and queue size at most $(\log d)2^{O(log^* d)}$, in which at each step only at most one packet can traverse each edge.*

Proof. We assume the following algorithm for producing a schedule:

- By using Lemma, produce a schedule S where the number of packets that use an edge in any frame of length $\log d$ is at most $\log d$.
- Break the schedule into frames of size $\log d$.
- Consider each frame of length $\log d$ as a new routing problem that has dilation and congestion $\log d$, and solve it recursively, until the frames have constant size and at most a constant number of packets use each edge in each frame.

The depth of the recursion is $O(\log^* d)$. Since the length of the new schedule increases by a constant factor during each recursive step, the length of the final schedule will be $(c+d)2^{O(\log^* d)}$ and the queue size will be at most the maximum amount of time steps that a packet wait, which is at most $O(\log d)2^{O(\log^* d)} = O(\log d 2^{O(\log^* d)})$.

□

References

[LMR94] F.T. Leighton, B. Maggs, and S. Rao. Packet routing and job-shop scheduling in O(congestion+dilation) steps. Combinatorica, 14(2):167-180, 1994.