EECS 290S: Network Information Flow **Fall 2008**

Lecture 5 — September 16

Lecturer: Anant Sahai *Scribe: Nebojsa Milosavljevic*

5.1 Cyclic graphs

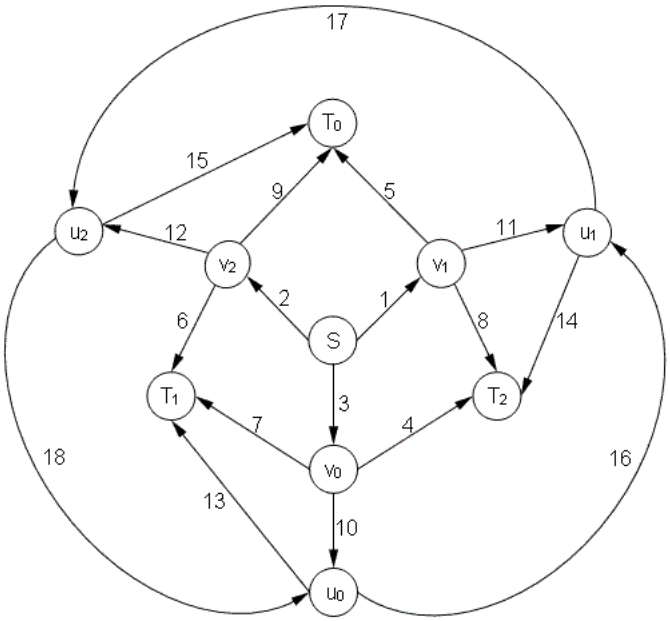

Figure 5.1. Cyclic graph.

Figure 5.1 represents a cyclic graph. Note that $R = \min \text{mincut} = 3$ because there are 3 disjoint paths reaching from source to destination (See Table 5.1). Now, we will show why a simple routing solution does not work in this case. From Table 5.1 we see that edge 17 appears in two distinct routes (S to T_0 and S to T_1), and moreover, must be carrying two distinct bits — therefore, there is no routing solution.

In previous lectures, we saw how to topologically sort an acyclic graph. For graphs with cycles this is impossible. For the above example, it is not even possible to prune the graph to eliminate the cycle since cutting even a single edge would reduce the mincut for some destination.

The only way to sort such a cyclic graph is by tracking time. Here, we assume a unit delay at each edge. Consider what happens if each node calculates a sum of all incoming edges to that node and forwards that information along all outgoing edges. The flow across

Routes	Edges
S to T_0:	1, 5
	2, 9
	3, 10, 16, **17**, 15
S to T_1:	2, 6
	3, 7
	1, 11, **17**, 18, 13
S to T_3:	1, 8
	3, 4
	2, 12, 18, 16, 14

Table 5.1. Paths from source to destinations.

the edges which are connected to nodes that have only one incoming edge are presented in Table 5.2. Figure 5.1 shows that edges (13,16), (14,17), (15,18) carry the summation of incoming edges connected to nodes u_0, u_1, u_2 respectively (See Table 5.3).

Edges		
1	at time t carries	$x_1(t)$
2	at time t carries	$x_2(t)$
3	at time t carries	$x_1(t)$
5 8 11	at time t carry	$x_1(t-1)$
6 9 13	at time t carry	$x_2(t-1)$
4 7 10	at time t carry	$x_3(t-1)$

Table 5.2. Flow across the edges.

Time	**13** and **16**	**17** and **14**	**15** and **18**
1	0	0	0
2	0	0	0
3	$x_3(1)$	$x_1(1)$	$x_2(1)$
4	$x_3(2) + x_2(1)$	$x_1(2) + x_3(1)$	$x_2(2) + x_1(1)$
5	$x_3(3) + x_2(2) + x_1(1)$	$x_1(3) + x_3(2) + x_2(1)$	$x_2(3) + x_1(2) + x_3(1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 5.3. Flow across the edges.

5.1.1 Decoding

Table 5.4 shows what is available at the destinations T_0, T_1, T_2 at each time, where + denotes all the additional information that was already available from the previous times. For example, observe destination T_0. For the first 4 time units, the results are obvious. At

time=5, T_0 receives $x_1(4)$, $x_2(4)$, $x_3(1)+x_1(2)+x_2(3)$. The last sum can be solved for $x_3(1)$ since all the other terms in the summation are known to T_0 from information received at previous time steps. From this result and by symmetry, we can deduce that with some delay, all information is eventually available to all the destinations.

Time	T_0	T_1	T_2
1	—	—	—
2	$x_1(1)$,$x_2(1)$	$x_2(1)$,$x_3(1)$	$x_1(1)$,$x_3(1)$
3	$+x_1(2)$,$x_2(2)$	$+x_2(2)$,$x_3(2)$	$+x_1(2)$,$x_3(2)$
4	$+x_1(3)$,$x_2(3)$	$+x_2(3)$,$x_3(3)$	$+x_1(3)$,$x_3(3)$
5	$+x_1(4)$,$x_2(4)$,$x_3(1)$	$+x_2(4)$,$x_3(4)$,$x_1(1)$	$+x_1(4)$,$x_3(4)$,$x_2(1)$
⋮	⋮	⋮	⋮

Table 5.4. Decoding at destinations.

5.1.2 Trellis Network

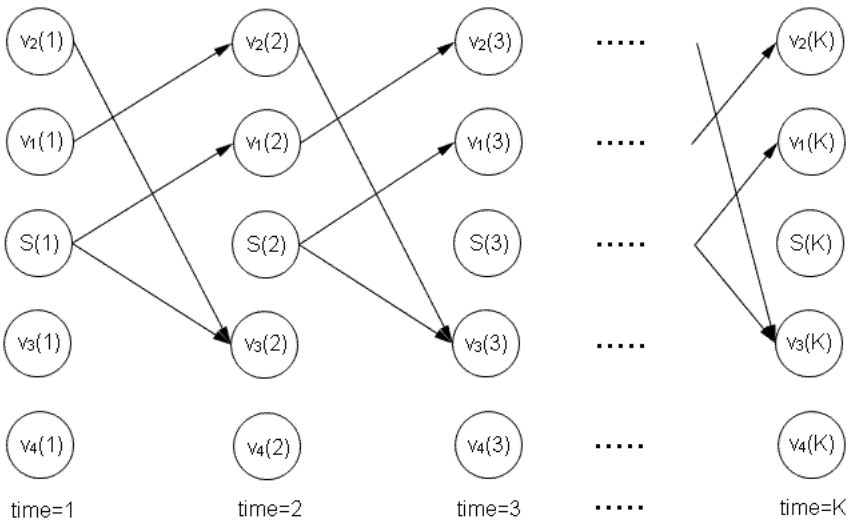

Figure 5.2. Trellis network.

Any general unit delay network can be unrolled thus with respect to the time into a special acyclic network called a trellis (Figure 5.2). Note that the trellis network is acyclic because all edges are moving forward in time, and therefore a directed cycle cannot be formed.

The only problem is that the network in Figure 5.2 consists of K sources. But, in order to apply min mincut we need to have a single source in the network. Because of that, we introduce a virtual source with R edges to each source node (RK total edges) (Figure 5.3),

where R is min mincut of the original network, here assumed to be 2 just for simplicity. Similarly, assume that all the destinations are pure sinks and have no outgoing edges. (If they do have outgoing edges, clone each such destination into one purely internal node and one pure sink. This can be accomplished by inserting fake nodes into each of the incoming edges and giving each fake node exactly two output edges: one to the internal node and another to the pure sink.) Now merge all the time-replicas of a single pure sink together into a single node. Since these have no outgoing edges, no cycles have been created. Now, we are back to having a single source and the otiginal number of destinations.

Notice that some of the new edges that were introduced are not a part of the communication, and this is why min mincut is not exactly R. However, this "boundary effect" becomes small as K gets large. We can bound the min mincut by using this intuition.

$$\frac{RK - R(n-1)}{K} \leq \frac{\text{min mincut}}{K} \leq \frac{RK}{K}, \tag{5.1}$$

where n is the number of nodes, implying that the max delay for routing is n-1. Therefore, in the trellis network the maximum number of routes that could not make it to the destination in time is R(n-1). These routes are not part of the communication, but since they do not scale with the unrolling time K, this does not asymptotically cost us any rate.

To finish the story, suppose that we apply Jaggi's algorithm to the trellis network in Figure 5.3. An encoding matrix for every node for every time step is obtained. This is a time-varying memoryless linear network code. Notice that the minimal field size is no different than it would have been in an acyclic network since all that matters for Jaggi's algorithm is the number of destinations.

Next time, we will see how it is possible to get a time-invariant code solution.

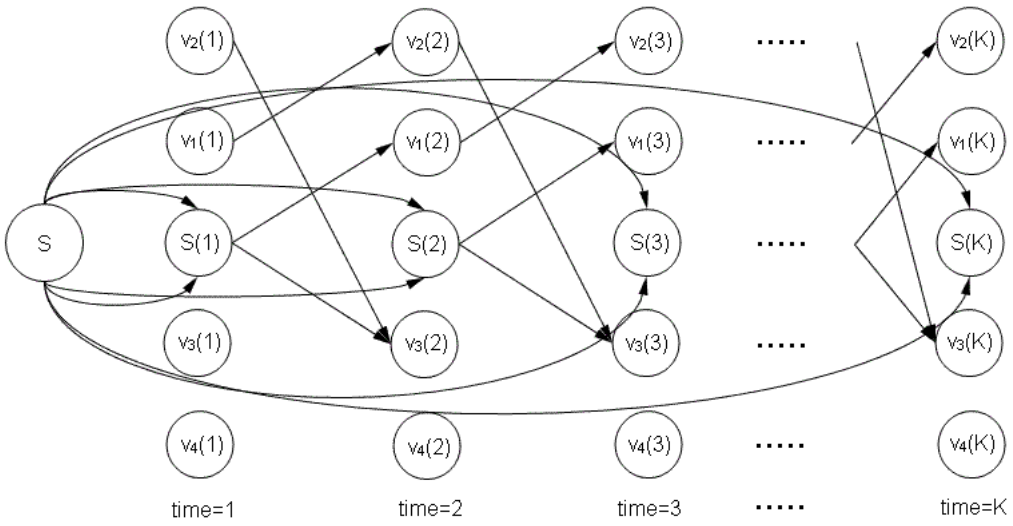

Figure 5.3. Trellis network with virtual source.