

Lecture 4 — September 11

Lecturer: Anant Sahai

Scribe: Sameer Pawar

4.1 Jaggi et.al algorithm for Direct Acyclic Graphs (DAGs)

In this lecture we will show how Jaggi's algorithm works by first executing it for the simple case of standard butterfly network. By doing this, we will also see clearly how it shows that network codes exist for any DAG as long as a sufficiently large field size (i.e. long enough packets in the case of wireline communication) is chosen.

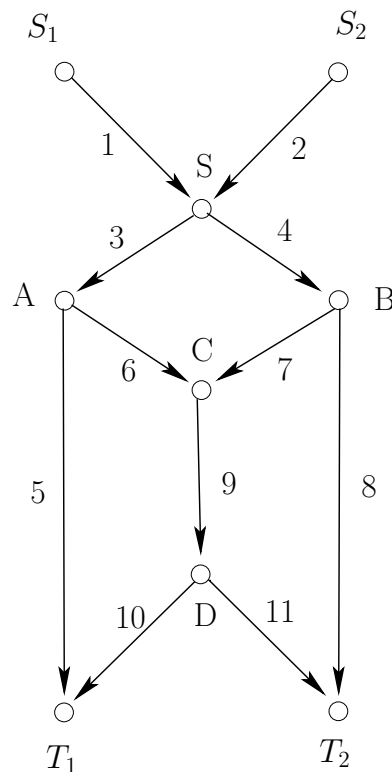


Figure 4.1. Butterfly network

Consider the butterfly network shown in Figure 4.1. For ease of description, we have labeled each edge with a number as shown. In addition, notice that we have created virtual

source nodes so that each node is only the source of a single packet. This makes it easier to think about. There are min min cut such virtual source nodes.

4.1.1 Assumptions

1. Choose field $\mathbb{F}_5 = \{0, 1, 2, 3, 4\}$ where all operations are modulo 5.
2. Each edge has a capacity of one symbol $\in \mathbb{F}_5$, i.e. $\log_2(5)$ bits per unit time.

4.1.2 Algorithm

Step 1: Start with the routing paths available by using Ford Fulkerson for example. For each destination T_i choose edge-disjoint paths from each source S_i . For example, consider the following edge disjoint paths:

S_1 to T_1 : 1, 3, 5.
 S_2 to T_1 : 2, 4, 7, 9, 10.

S_1 to T_2 : 1, 4, 8.
 S_2 to T_2 : 2, 3, 6, 9, 11.

Notice that the different paths leading to any single destination are disjoint from each other, but paths leading to different destinations can share edges. Here, all edges participate in these paths but in the general case, we eliminate any edges that do not participate in any routing path.

Let $b(e) = \begin{pmatrix} \\ \end{pmatrix}$ denote the linear combination of S_1, S_2 that flows across the edge e .

For example $b(1) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $b(2) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. The idea is to start from the source and walk down the graph in topological-sort order. Recall, this is an order (it need not be unique) in which every node occurs only after any of its ancestors. This is always possible for acyclic graphs.

On the way we maintain the running cuts C_1, C_2 for destination T_1 and T_2 respectively. At each stage, we are interested in making sure that the cut has full rank. To do this efficiently, Jaggi keeps something similar to inverse of the cut matrix around.

Lemma 4.1. (Lemma 5 in Jaggi's paper) Given a basis $\mathcal{B} = \{b_1, b_2, \dots, b_R\}$ and a set of vectors $\{a_1, a_2, \dots, a_R\}$ such that $\langle b_i, a_j \rangle = \delta_{ij}$. \bar{X} is linearly dependent on $\mathcal{B} \setminus \{b_i\}$ iff $\langle \bar{X}, a_i \rangle = 0$

Proof: a_i 's are \perp to the hyperplane spanned by $\mathcal{B} \setminus \{b_i\}$. If \bar{X} is linearly dependent on $\mathcal{B} \setminus \{b_i\}$, then $\bar{X} = \sum_{j \neq i} \bar{X}(b_j) b_j$, and we have

$$\langle \bar{X}, a_i \rangle = \sum_{j \neq i} \bar{X}(b_j) \langle b_j, a_i \rangle = 0$$

Similarly other way round. □

We begin from top of graph with the first cut $C_1 = \{1, 2\}$, $C_2 = \{1, 2\}$ for destinations T_1 and T_2 respectively. For each cut, we have an associated matrix that has $b(e)$ for its columns where e is an edge participating in the cut, e.g., for C_1 we have $\begin{bmatrix} b(1) & b(2) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ which is full-rank. As we mentioned earlier to check the full rank of these cut matrices we also construct a 's. There are a 's for each pair of edge e and destination t denoted by $a_t(e)$ such that $\langle a_t(i), b(j) \rangle = \delta_{ij} \forall b(j) \in C_t$.

For cut $C_1 = \{1, 2\}$.

matrix associated: $\begin{bmatrix} b(1) & b(2) \end{bmatrix}$

$$a_1(1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad a_1(2) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

For cut $C_2 = \{1, 2\}$.

matrix associated: $\begin{bmatrix} b(1) & b(2) \end{bmatrix}$

$$a_2(1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad a_2(2) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Step 2 Walk down the graph, pick any one edge (say 3 in this case) and update the cuts for each destination by replacing the ancestor of edge 3 with 3. Since for each destination we had edge disjoint paths from sources, each edge can only be part of one path, and hence has a unique ancestor (edge preceding it in the path). E.g. for T_1 , the ancestor of edge 3 is edge 1. So the updated values are,

$$C_1 = \{3, 2\} \\ \begin{bmatrix} b(3) & b(2) \end{bmatrix}$$

$$C_2 = \{1, 3\} \\ \begin{bmatrix} b(1) & b(3) \end{bmatrix}$$

Choose $b(3) = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ so that both the matrices are full rank.

Revise $a_1(2)$ such that inverse property i.e., $\langle a_1(2), b(3) \rangle = 0$ is maintained for newly added column $b(3)$.

$$a_1(2) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} - (\text{Projection of } a_1(2) \text{ along } b(3)) = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

similarly revise $a_2(1)$ for the inverse property,

$$a_2(1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - (\text{Projection of } a_2(1) \text{ along } b(3)) = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

Choosing $a_1(3), a_2(3)$ is easy in this case since the ancestor values for a still maintain the desired inverse property. So finally, the updated values after this stage are:

$$C_1 = \{3, 2\}$$

$$\begin{bmatrix} b(3) & b(2) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$C_2 = \{1, 3\}$$

$$\begin{bmatrix} b(1) & b(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} a_1(3) & a_1(2) \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} a_2(1) & a_2(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix}$$

Step 3 Continue down the topological sort by adding new edge 4 to the cuts C_1, C_2 and remove its ancestors for each destination. Also updating all b 's and a 's according to the rule in the previous step, we get

$$\begin{aligned} C_1 &= \{3, 4\} & C_2 &= \{3, 4\} \\ \begin{bmatrix} b(3) & b(4) \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} b(3) & b(4) \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix} \\ \begin{bmatrix} a_1(3) & a_1(4) \end{bmatrix} &= \begin{bmatrix} 3 & 3 \\ 3 & 2 \end{bmatrix} & \begin{bmatrix} a_2(3) & a_2(4) \end{bmatrix} &= \begin{bmatrix} 3 & 3 \\ 3 & 2 \end{bmatrix} \end{aligned}$$

Similarly continue walking down the graph in topological order till you reach the destinations.

In general, at every stage we remove an edge from the cut and add its descendent edge and update all the entries. Let $f_t(e)$ be the ancestor edge of e for the destination t (if for a particular destination t , edge e does not occur in the route then $f_t(e)$ for that t is not defined). We want to choose $b(e)$ in the span of $b(f_t(e)) \forall t \in T$ such that $\langle a_t(f_t(e)), b(e) \rangle \neq 0$. So natural question is, Does such a $b(e)$ always exist? The following key lemma answers this question.

Lemma 4.2. (Lemma 8 in paper) Assume that $n = |T| < |\mathbb{F}|$. Suppose we know that pairs $(\bar{X}_i, \bar{Y}_i) \in \mathbb{F}^R \times \mathbb{F}^R$ exist such that $\langle \bar{X}_i, \bar{Y}_i \rangle \neq 0$ for $1 \leq i \leq n$.

claim: There exists $\bar{U} = \sum_{i=1}^n u_i \bar{X}_i$ such that $\langle \bar{U}, \bar{Y}_i \rangle \neq 0 \forall i$

Proof: We prove it by induction on n .

For $n = 1$, the choice is trivial since we can just use $\bar{U} = \bar{X}_1$ itself.

Now assume that it is true for each $i \leq n - 1$, i.e., we have \bar{U}_i such that $\langle \bar{U}_i, \bar{Y}_j \rangle \neq 0 \forall j \leq i$. Now to add \bar{Y}_{i+1} to set of Y 's that we must have nonzero dot-product with, just try $\langle \bar{U}_i, \bar{Y}_{i+1} \rangle$. If it is non-zero, then we are done.

If it is zero, then for all scalars $\alpha \neq 0 \in \mathbb{F}$,

$$\langle \alpha \bar{U}_i + \bar{X}_{i+1}, \bar{Y}_{i+1} \rangle \neq 0$$

But this choice of α might disturb the inner products for some other $j \leq i$, i.e., $\langle \alpha \bar{U}_i + \bar{X}_{i+1}, \bar{Y}_j \rangle = 0$ for some $j \leq i$, but that happens iff

$$\alpha = \alpha_j := -\frac{\langle \bar{X}_{i+1}, \bar{Y}_j \rangle}{\langle \bar{U}_i, \bar{Y}_j \rangle}$$

So for each j there is exactly one α_j which is bad, and since $n = |T| < |\mathbb{F}|$ we are guaranteed by the pigeon-hole principle that some $\alpha \in \mathbb{F}$ must work since they can't all be bad. \square

In Jaggi's paper they also show that we can find $b(e)$ using the above approach in $\mathcal{O}(|T|^2R)$ time. The polynomial nature of the search is easy to see. Computing the dot product or computing a vector sum takes $\mathcal{O}(R)$ time. Thus all the bad α_j can be computed in $\mathcal{O}(R)$ time each. This is done $\mathcal{O}(n^2)$ times since they are n stages with at most n such computations at each stage.

How to update the a 's?

$$a_t(e) = \frac{a_t(f_t(e))}{\langle a_t(f_t(e)), b(e) \rangle}$$

And we adjust other $a_t(c)$ for $\forall c \neq e \in C_t$ by subtracting its projection along the direction of $b(e)$, so that resulting $a_t(c)$ is orthogonal to $b(e)$ i.e.,

$$a_{t,\text{new}}(c) = a_{t,\text{old}}(c) - \langle a_{t,\text{old}}(c), b(e) \rangle b(e)$$

This is just like Gram-Schmidt. So by induction $\langle a_t(i), b(j) \rangle = \delta_{ij}$ at every stage. Thus we always have a certificate of invertability of the linear combinations that flow across the cuts. Since the final linear combinations that are considered are the ones that are entering the destinations themselves (these must be the last edges considered in the topological sort), this means that each destination is able to reconstruct all R original source packets.