18.409 An Algorithmist's Toolkit September 30, 2009

Lecture 7

Lecturer: Jonathan Kelner *Scribe: Alan Deckelbaum (2009)*

1 Administrivia

There were two administrative announcements made at the beginning of class today.

- The first problem set will be posted online tonight. It will be due two weeks from today, on October 15.
- Some people have asked Professor Kelner if he could post typo-free versions of the powerpoint slides online. Therefore, it is requested that the scribe email Professor Kelner a list of the known typos in the slides.

2 Outline

In today's lecture we are going to:

- Discuss nonblocking routing networks
- Start the description of a method for local and almost linear-time clustering and partitioning based on the Lovasz-Simonovits Theorem.

Nonblocking routing networks are an example application of expander graphs. Furthermore, many of the techniques that we discuss today for analyzing routing networks can also be applied to error-correcting codes. In the second half of class, we discuss local clustering and partitioning, and we will finish the analysis in Tuesday's class.

3 Nonblocking routing networks

Suppose you have a network with $2N$ terminals, consisting of N "input" terminals (drawn on the left side) and N "output" terminals (drawn on the right side). We want to design a network that connects them, in such a way that every permutation can be routed. That is, for any one-to-one function f from the input terminals to the output terminals, we would like there to be a path in the network from each input node i to the output node $f(i)$. Furthermore, we ask that for any such f, the paths can be chosen to be vertex disjoint. The motivation is that we would like for each input terminal to be able to talk to a different output terminal so that none of the intermediate routers are overloaded. This kind of networks is called a *nonblocking routing network* and it is generally useful for communication.

We also want the network to have some nice features. First, we want that each node has bounded degree (If we don't ask this, then a simple solution is to put a wire from every input to every output. Of course, this is not very plausible for large N, for example the phone network in the U.S.). Second, we want that the number of nodes in the graph is $O(n \log n)$ (so, that the network is not too big). And finally we want a fast, decentralized algorithm for finding routes.

3.1 Butterfly networks

A first attempt to find a small network in which routing is easy is to consider a *Butterfly network*. A Butterfly network for N inputs and outputs consists on $\log(N)$ layers of N nodes each. The first layer (or zeroth layer) correspond to the N inputs and the last layer corresponds to the N outputs. The i-th layer is divided into blocks of $N/2^i$ elements. Each block 'splits' into two blocks: 'up' and 'down' in the next layer. Each node has two neighbors in the next layer, one in the corresponding 'up' block and one in the corresponding 'down' block in such a way that every node in the $(i+1)$-th layer has only 2 neighbors in the i-th layer.

This can be easily done by labeling each node in the network with a pair (i, r) where i corresponds to the number of the layer and r is the position of the node in the layer written in binary. Then each node (i, r) is connected to $(i+1, r)$ and $(i+1, r^{(i+1)})$, where $r^{(i+1)}$ denotes r with the $(i+1)$-th element complemented.

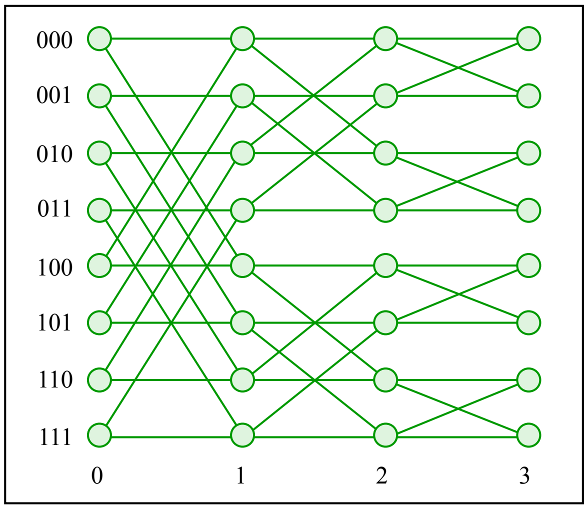

Figure by MIT OpenCourseWare.

Figure 1: Butterfly network for 8 inputs.

Although in a Butterfly network, routing is easy (just follow the binary expansion of the label of the desired output), it's not possible to avoid congestion (e.g. in Figure 1, if you try to route 000 to 000 and 110 to 001, the route will collide on the third layer). Actually, for some permutations you can get up to $\sqrt{N}$ collisions. While butterfly networks do not successfully avoid congestion, they are the base for a type of nonblocking routing network called a multibutterfly.

3.2 Multibutterfly networks

The construction of multibutterfly networks uses bipartite expanders, which we discussed last class. We recall the definition here:

Definition 1 *A d-regular bipartite graph is an (α, β)-**expander** if every set S on the left with $|S| \leq \alpha n$ has $N(S) \geq \beta|S|$.*

Thus, intuitively, a regular bipartite graph is an (α, β)-expander if any small collection of vertices on the left has a proportionately large number of neighbors.

A d-multibutterfly is a layered graph constructed similarly to the butterfly networks, but such that each node now has d edges out to the 'up' block of the next layer and d edges out to the 'down' block. These networks are carefully constructed such that the graph induced by the vertices of a block and its 'up' neighbor block and the graph induced by the block and its 'down' neighbor block are both (α, β)-expanders for $\beta > d/2$, $\beta \approx 1/2\alpha$.

A 2-multibutterfly for $N = 8$ allowing 4 pairs of inputs/outputs is shown in Figure 2.

Note that d-multibutterflies have $O(N \log N)$ vertices and that the vertices have bounded degree (at most $4d$). then the only condition needed for this to be a good nonblocking routing network is that we can route any permutation of inputs and outputs easily.

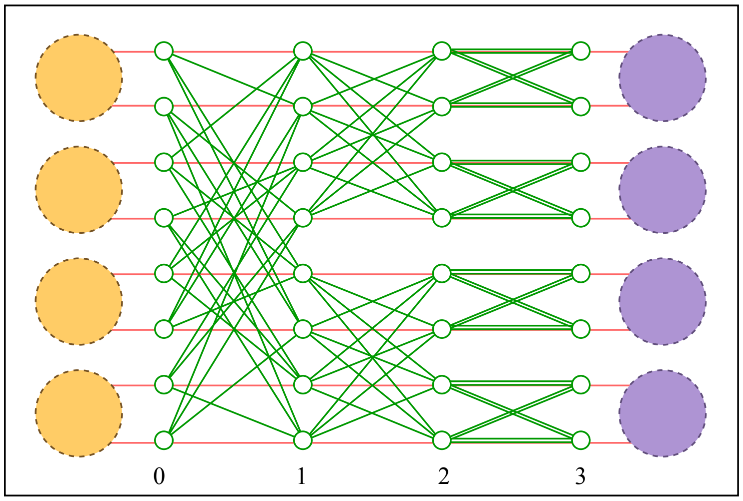

Figure by MIT OpenCourseWare.

Figure 2: 2-Multibutterfly network for 4 inputs.

Each layer of the network has N elements, but the multibutterfly network we described will only route $2N\alpha$ inputs to $2N\alpha$ outputs. In a final implementation, we would connect the actual inputs and outputs to $\alpha/2$ successive multibutterfly inputs/outputs.

Claim 2 *A d-Multibutterfly with N nodes in each layer can route any permutation of $2\alpha N$ inputs to $2\alpha N$ outputs.*

To prove this claim, we will need Hall's Theorem.

Theorem 3 (Hall) *A bipartite graph G has a perfect matching if and only if every set S on the left has at least $|S|$ neighbors on the right.*

Using Hall's Theorem, we now prove the above claim.

Proof We only need to prove that in each pair of consecutive layers, any block can successfully route its signals to both the 'up' neighbor block and the 'down' neighbor block. It suffices to prove this for the first layer, since the proof for any other layer is identical. Note that in the first layer, at most αN calls will need to go to the top half, and at most αN calls need to go to the bottom half. Let S be the set of nodes in the first layer which need to go up, and let T be the set of nodes in the top half of the second layer. We will show that it is possible to choose edges to match each vertex of S with a vertex of T.

Since $|S| \leq \alpha N$, and the graph induced by $S \cup T$ is an (α, β)-expander by construction, it follows that $|N(S) \cap T| \geq \beta|S| > |S|$, and so, by Hall's theorem, there is a perfect matching between S and T. We use that matching to route the calls that need to go up. The calls that need to go down are routed in an analogous way. ■

The previous claim guarantees that a nonblocking routing exists, and it can be found by solving a matching problem. However, in real life we don't want to use a complicated global computation for routing: We need a simple distributed algorithm. Such an algorithm exists and is simple to describe. Consider a pair of blocks S and T that we want to match (as in the proof of the claim). The algorithm is as follows:

Algorithm:

```
$S_1 \leftarrow S$.
while $S_i \neq \emptyset$
    Every node of $S_i$ sends a proposal to all neighbors in $T$.
    Every node of $T$ receiving exactly one proposal accepts it.
    Every node in $S_i$ that got at least one accepted proposal picks one arbitrarily and matches to it.
    $S_{i+1}$ is the set of unmatched remaining nodes in $S_i$.
```

Claim 4 *The previous algorithm finds a matching of S and T in $O(\log n)$ steps.*

To prove this claim, first we will need to prove the following two lemmas:

Lemma 5 *For any set S of size $|S| \leq \alpha N$, the number of vertices in T with exactly one neighbor in S is at least $(2\beta - d)|S|$*

Proof Let A be the vertices in T with exactly one neighbor in S, and let B be the remaining vertices in T which are neighbors of S. Since the graph induced by $S \cup T$ is a (α, β)-expander, $|A \cup B| \geq \beta|S|$. Also, we know that the number of edges from S to T is at most $|A| + 2|B|$. Thus, using the fact that the number of edges from S to T is exactly $d|S|$, we know:

$$|A| + |B| = |A \cup B| \geq \beta|S|$$

$$d|S| = e(S,T) \geq |A| + 2|B|$$

and hence

$$|A| \geq \beta|S| - |B| \geq \beta|S| - \frac{d|S| - |A|}{2}$$

and thus $|A| \geq 2\beta|S| - d|S|$. ■

Given the above lemma nd the fact that any node in the left side can receive at most d acceptances in any round of the protocol, we conclude:

Lemma 6 *For all i,*

$$\frac{|S_{i+1}|}{|S_i|} \leq 2(1 - \beta/d).$$

This last lemma guarantees that the algorithm converges in $O(\log n)$ steps, as desired.

4 Local and almost linear-time clustering and partitioning

4.1 Motivation

In these days, graphs are getting really big. For example, circuits layouts have 50 million transistors; scientific computing has hundreds of millions of variables; the Internet has billions of nodes, etc. So any algorithm that performs a task in these graph in time such as n^2 will be very bad in practice. Even a running time such as $n^{1.5}$ tends to not be good enough. In some cases, like in Internet, even visiting every node of the graph once is an impossible task. For that reason, we are interested in developing algorithms for certain applications that runs in almost linear time (i.e. $O(n\text{polylog}(n))$), or algorithms that are local, i.e., that do not need to visit the entire graph. (Note that log factors tend to be fairly small, even in the cases mentioned above, and oftentimes logarithmic factors depend on the specific model of computation being analyzed.)

4.2 Local Clustering

Given a vertex v in a graph we would like to know if v is contained in a cluster, where a cluster is a set that defines a cut of low conductance. We also want the running time for this algorithm to depend on cluster size and not in the size of the graph. A good example for this is finding a cluster of web pages around the mit.edu domain, where we don't want this task to depend on the number of sites recently created on the other side of the world.

The goal for this part of the lecture will be to describe an algorithm that runs in time almost linear in K that outputs a cluster of size at least $K/2$ around starting vertex, if that cluster exist.

The approach we will use will rely on what we know about cuts, eigenvalues and random walks. First observe that if v is contained in a cluster and you start running a random walk from v, then the low conductance cut will be an obstacle for the mixing time. This means that the random walk has trouble leaving the cluster. So, a good guess for the cluster is the set of vertices for which a random walk starting at v will have the highest probabilities after a given number of steps. Using this idea, a good primitive to construct an almost linear algorithm will be the following.

"Approximate, for every vertex in the graph, the probability that a random walk starting from v is in that vertex after certain time, select the k highest valued vertices and check if they define a good cut. Repeat this until you get a good cut or you reach a predetermined limit."

Note that this method is similar to the proof of Cheeger's inequality, where we ordered the entries of v_2 to obtain a cut. Here, however, use use a probability vector instead of the eigenvector v_2.

We need a bound that says that this idea works. Unfortunately, all the bounds we have proven thus far are global, involving λ_2 of the whole graph. We desire bounds on a local feature of the graph. Furthermore, we can't really compute all of the necessary probabilities, because it will take too long. We therefore need to approximate the probabilities, without knowing the size of the cluster in advance.

4.3 Lovasz-Simonovits

The Lovasz-Simonovits Theorem will give us the bound we need for the algorithm. It relies on an interesting idea: measure the progress of the walk not by one number but by a whole curve. The better the random walk is to reaching the stable distribution, the closer the curve will be to a straight line. Different points on the curve will correspond to different size partitions. Before stating the theorem, we will need some definitions.

Let G be the digraph obtained from the original graph where we first replace each undirected edge uv by two directed arcs (u, v) and (v, u), and then we add self-loops loops to each node until every node v of G has $d_v/2$ self-loops (i.e. half of the edges leaving v are self-loops).

Instead of focusing on the vertices, we will mainly study the edges of G. Suppose that we have a certain probability distribution p on vertices. Define

$$\begin{aligned}\rho(u) &= \frac{p(u)}{d_u}, \text{ for every node } u,\\ \rho(u,v) &= \rho(u), \text{ for every arc } (u,v).\end{aligned}$$

Note that $\rho(u, v)$ represents the mass about to be sent over arc (u, v) and that for every node u, $\rho(u)$ approaches $1/2m$ as the walk converges (here, $2m$ is the number of arcs in the digraph). Therefore, as the walk converges, $\rho(e)$ goes to $1/2m$ for every arc e.

We will define a curve that measures how close we are to convergence and also contains additional information.

Definition 7 (Lovasz-Simonovits curve) *For a given ρ, order the arcs such that $\rho(e_1) \geq \rho(e_2) \geq \ldots \geq \rho(e_{2m})$. We define $I : [0, 2m] \to [0, 1]$ as follows: For each $k \in \{0, \ldots, 2m\}$,*

$$I(k) = \sum_{i=1}^{k} \rho(e_i).$$

We extend I to the complete interval by interpolating it piecewise linearly.

Intuitively $I(k)$ measures how much probability is transported over the k most utilized edges. Here we describe some of the properties of the L-S curve.

- As the walk converges I should eventually converge to a straight line.
- $I(0) = 0, I(2m) = 1$.
- The slope of I between k and $k + 1$ is given by $I(k + 1) - I(k) = \rho(e_{k+1})$.
- Since ρ depends only on the start vertex, it does not matter how we order edges out of any particular node u, and therefore the slope of I is constant for all the intervals corresponding to arcs leaving u.

- The slope is nondecreasing, so I is concave.

We will prove some claims and Theorems about I, and then we will state and prove the Lovasz-Simonovits Theorem.

Claim 8 *For any $c_1, \ldots, c_{2m} \leq 1$,*

$$\sum_{i=1}^{2m} c_i \rho(e_i) \leq I\left(\sum_{i=1}^{2m} c_i\right).$$

Proof Think of the c_i's as weights for the $\rho(e_i)$. Since the $\rho(e_i)$ are non-increasing, moving some weight from some j to some $i < j$ only makes the sum bigger. So the sum is biggest when the first bunch of c_i's are 1, the next one is the remaining, and the rest of them are 0, which is exactly the value of the right hand side. ■

In what follows, let ρ^t and I^t be ρ and I at time t in the random walk.

Claim 9 *For all x and t, $I^t(x) \leq I^{t-1}(x)$.*

Proof This claim states that the value of the curve at any point never increases as t increases. Let $e_i = (u_i, v_i)$, so that $\rho(u_1, v_1) \geq \rho(u_2, v_2) \ldots \geq \rho(u_{2m}, v_{2m})$ It suffice to prove the claim in the case where $x = k$ and W is the vertex set $\{u_1, \ldots, u_k\}$ such that $(u_1, v_1), \ldots, (u_k, v_k)$ are precisely the set of edges leaving W. We then have:

$$\begin{aligned} I^t(k) &= \sum_{i=1}^{k} \rho^t(u_i, v_i)) = \sum_{i=1}^{k} \rho^t(u_i) \\ &= \sum_{i=1}^{k} \rho^{t-1}(v_i, u_i), \text{ as the mass leaving } W \text{ at } t \text{ is the amount entering } W \text{ at } t-1 \\ &\leq I^{t-1}(\sum_{i=1}^{k} 1) = I^{t-1}(k), \end{aligned}$$

where the last inequality follows from the previous claim. ■

Now we will prove something a little stronger: We will prove that the curve I^t has to lie below I^{t-1} by an amount depending on ϕ_G.

Theorem 10 *For every initial distribution p^0, all t, and every $x \in [0, m]$,*

$$I^t(x) \leq \frac{1}{2}\left(I^{t-1}(x - 2\phi_G x) + I^{t-1}(x + 2\phi_G x)\right),$$

and for every $x \in [m, 2m]$,

$$I^t(x) \leq \frac{1}{2}\left(I^{t-1}(x - 2\phi_G(2m - x)) + I^{t-1}(x + 2\phi_G(2m - x))\right).$$

Before beginning the proof, we note that the above result has a geometric interpretation. The first equation above states that the value of $I^t(x)$ lies below the midpoint of the line connecting $I^{t-1}(x - 2\phi_G x)$ and $I^{t-1}(x + 2\phi_G x)$. Recalling that the graph of I is always concave, this implies the above result that the value of $I(x)$ at time t is no greater than the value of $I(x)$ at time $t-1$. Furthermore, if I^t differs significantly from the straight line (which I converges to in the limit) and if $\phi_G(x)$ is large, then $I^t(x)$ will decrease by a significant amount in the next step (once again, by concavity). Thus, the theorem matches our intuition that low-conductance cuts around x cause the walk to mix more slowly.

Proof

This proof was not covered in its entirety in today's lecture, but it was covered in Lecture 7 of the 2007 version of the course:

We will only prove the case $x \in [0, m]$, the second case should be analogous. As in the previous claim we can assume without loss of generality that $x = k$, and that $(u_1, v_1), \dots, (u_k, v_k)$ are exactly the set of edges starting from $W = \{u_1, \dots, u_k\}$. We then have:

$$\sum_{i=1}^{k} \rho^t(u_i, v_i) = \sum_{i=1}^{k} \rho^{t-1}(v_i, u_i).$$

Break the edges of the right into two sets:

$$W_1 = \{(v_i, u_i) : u_i, v_i \in W, v_i \neq u_i\}. \quad W_2 = \{(v_i, u_i) : u_i \in W, v_i \notin W\} \cup \{\text{self loops}\}.$$

We claim that:

1. $\sum_{(v,u)\in W_1} \rho^{t-1}(v, u) \le \frac{1}{2} I^{t-1}(x - 2\phi_G x)$.
2. $\sum_{(v,u)\in W_2} \rho^{t-1}(v, u) \le \frac{1}{2} I^{t-1}(x + 2\phi_G x)$.

Note that out of the $x = k$ edges starting at W, $x/2$ are self loops, and at least $\phi_G x$ edges leave W, therefore, the number of edges in W_1 is at most $x/2 - \phi_G x$. Note that if we let c_i be 1 if $e_i \in W_i$ and 0 otherwise, we have that $\sum_{i=1}^{k} c_i \le x/2 - \phi_G x$. And then, using Claim 8, we can obtain the following weaker bound:

$$\sum_{(v,u)\in W_1} \rho^{t-1}(v, u) \le I^{t-1}(x - 2\phi_G x).$$

We need to 'move' the 1/2 factor outside of I^{t-1} somehow. Instead of doing that, we will carefully choose other values for c_i to obtain the wanted bound. For that simply let c_i be 1/2 if $e_i \in W_i$ or if e_i is a self loop in W and 0 otherwise. Then, since every vertex has the same number of self loops as edges leaving it (and they all have the same ρ value), we also obtain under this new set of weights, that $\sum_{i=1}^{k} c_i \le x/2 - \phi_G x$. Using Claim 8 and that $2c_i \le 1$, for every i, we have:

$$\sum_{(v,u)\in W_1} \rho^{t-1}(v, u) = \sum_{i=1}^{k} c_i \rho^{t-1}(v_i, u_i) = \frac{1}{2}\sum_{i=1}^{k} c_i \rho^{t-1}(v_i, u_i) \le \frac{1}{2} I^{t-1}(x - 2\phi_G x).$$

The second claim follows in a similar way, and combining both of them we obtain the result. ■

Using the previous theorem, we are ready to state and prove Lovasz-Simonovits Theorem.

Theorem 11 (Lovasz-Simonovits) *For all initial distribution p^0 and every t,*

$$I^t(x) \le \min(\sqrt{x}, \sqrt{2m - x}) \left(1 - \frac{1}{2}\phi_G^2\right)^t + \frac{x}{2m}.$$

Sketch of Proof Consider the curve

$$R^0 = \min(\sqrt{x}, \sqrt{2m - x}) + \frac{x}{2m}.$$

It is easy to show that $I^0(x) \le R^0(x)$, for all $x \in [0, 2m]$. If we set

$$R^t(x) = \frac{1}{2}\left(R^{t-1}(x - 2\phi_G x) + R^{t-1}(x + 2\phi_G x)\right),$$

for $x \in [0, m]$ and

$$R^t(x) = \frac{1}{2}\left(R^{t-1}(x - 2\phi_G(2m - x)) + R^{t-1}(x + 2\phi_G(2m - x))\right),$$

for $x \in [m, 2m]$, then it is easy to show that

$$R^t(x) \le \min(\sqrt{x}, \sqrt{2m - x})\left(1 - \frac{1}{2}\phi_G^2\right)^t + \frac{x}{2m}.$$

Using that all the curves defined so far are concave and the previous theorem, we have:

$$I^t(x) \le R^t(x),$$

for all t, which proves the theorem. ■

From here, we can derive the following Corollary:

Corollary 12 *For W a set of vertices, and $x = \sum_{w \in W} d_w$,*

$$\left|\sum_{w \in W} p^t(w) - \pi(w)\right| \le \min(\sqrt{x}, \sqrt{2m - x})\left(1 - \frac{1}{2}\phi_G^2\right)^t.$$

We can use this Corollary for local clustering in the following way. If after $O((\log m/\phi_G)^2)$ steps a set of vertices contains a constant factor more than what it would have under stationary distribution, then we can get a cut C such that $\Phi(C) \le \phi_G$. (The cut can be obtained by mapping the probabilities to real line and cut like we did with v_2 some lectures ago).

The problem with this approach is that computing all the probabilities will be too slow. In particular, after a constant number of steps we have too many nonzero values. One solution proposed by Lovasz and Simonovits is to simply zero out the smallest probabilities and prove that it doesn't hurt much. However, the analysis can get messy. Instead, in next lecture we will speak about a different approach that uses a slightly different vector, called PageRank.