Graphs and Networks Lecture 11

Cutting Graphs, Personal PageRank and Spilling Paint

Daniel A. Spielman October 3, 2013

11.1 Disclaimer

These notes are not necessarily an accurate representation of what happened in class. They are a combination of what I intended to say with what I think I said. They have not been carefully edited.

11.2 Cutting Graphs

You may have figured out by now that I like cutting graphs into pieces. It is how I discover the structure of a graph. When I encounter a new graph and want to understand what it looks like I first try drawing it. If the drawing looks good, I feel like I understand it. If the drawing looks good, I try chopping the graph into pieces without cutting too many edges. That is, I look for cuts of low conductance.

The first way I try to find cuts of low conductance is by examining the eigenvector of the second-largest eigenvalue of the walk matrix. In lecture 9, we observed that lazy walks converge to the stable distribution, and that they do so at a rate that depends on the magnitude of the second-largest eigenvalue. This was because

$$\boldsymbol{p}_t - \boldsymbol{\pi} = \boldsymbol{D}^{1/2} \sum_{i \geq 2} \lambda_i^t \alpha_i \boldsymbol{v}_i,$$

where

$$\alpha_i = \boldsymbol{v}_i^T \boldsymbol{D}^{-1/2} \boldsymbol{p}_0.$$

After a sufficient amount of time, the dominant term in the expression for $\boldsymbol{p}_t - \boldsymbol{\pi}$ will be

$$\boldsymbol{D}^{1/2} \lambda_2^t \alpha_2 \boldsymbol{v}_2,$$

at least if $\lambda_2 > \lambda_i$ for all $i > 2$. This is why you will often find the same cut when you try the experimental part of problem set 3, regardless of at which vertex you begin your walk.

I should mention now that we have just described an algorithm for computing $\boldsymbol{v}_2$: start with an arbitrary distribution, apply the matrix $\widehat{\boldsymbol{W}}$ many times, and subtract off $\boldsymbol{\pi}$ when you are done. You will get better results if you subtract off $\boldsymbol{\pi}$ at every step. The resulting algorithm is called the power method.

Cheeger's inequality tells us that if a graph has a set of low conductance, then we can use $\boldsymbol{v}_2$ to find a set of low conductance. We know from Theorem 9.4.1 that if a graph has a set of low conductance, then some random walks will not mix quickly, and so λ_2 must be close to 1. The following theorem quantifies this nicely.

Theorem 11.2.1. *Let $1 = \lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ be the eigenvalues of the lazy random walk on a graph, and let $\mu = 1 - \lambda_2$. Then,*

$$\phi_G \geq \mu.$$

Cheeger's inequality is really the converse: if μ is small, then the graph must have a set of low conductance, and one can find one by scanning along the values of $\boldsymbol{v}_2$.

Theorem 11.2.2. *[Cheeger's inequality for lazy random walks]*

$$\mu \geq \phi_G^2/4.$$

Moreover, if $\boldsymbol{w}_2$ is a right-eigenvector of $\boldsymbol{W}$ corresponding to λ_2, then there is a number x for which the set

$$S_x = \{a \in V : \boldsymbol{w}_2(a)/d(a) \geq x\}$$

satisfies

$$\phi(S_x)^2/4 \leq \mu.$$

The drawback of using $\boldsymbol{v}_2$ to find a set of low conductance is that it is a global process: the computation of $\boldsymbol{v}_2$ requires examining the whole graph. And, it might yield a small set.

One of the advantages of considering random walks starting at a given vertex is that they are more likely to find a set of low conductance near that given vertex. This allows us to focus on a vertex of interest. In fact, we can improve this procedure to find a set of low conductance in time proportional to the size of that set! I will now explain a particularly nice way of doing this using Personal PageRank vectors.

11.3 PageRank

We have all encountered the PageRank algorithm: it is how Google got started ranking web pages. It involves random walks in directed graphs. We are going to talk more about random walks in directed graphs in a later lecture. But, I need to give you the introduction now.

PageRank considers a process that with probability α jumps to a uniformly random vertex of a graph, and with probability $1 - \alpha$ follows a random edge out of the present node. The PageRank vector is the steady-state distributuion of this process. That is, if we let $\boldsymbol{W}$ be the walk matrix of the directed graph (you can figure out how to define it), the PageRank vector $\boldsymbol{p}$ will satisfy

$$\boldsymbol{p} = \alpha \frac{1}{n}\mathbf{1} + (1 - \alpha)W\boldsymbol{p}.$$

We are going to consider a variation of the PageRank vector called the *personal PageRank vector*. Where PageRank measures the importance of nodes overall, the personal PageRank vector measures

the importance of nodes with respect to a give node u. It does this by modifying the walk so that with probability α it jumps back to u, rather than to a random node. We denote the vector that satisfies this equation by $\boldsymbol{p}_u$, and note that it must satisfy the equation

$$\boldsymbol{p}_u = \alpha\chi_u + (1-\alpha)\,\boldsymbol{W}\boldsymbol{p},$$

where χ_u is the elementary unit vector in the direction of vertex u.

For today, we will just consider these vectors in *undirected* graphs.

Let's begin by showing that the vectors $\boldsymbol{p}_u$ actuall exist. By maniuplating the equation for $\boldsymbol{p}_u$, we derive

$$\begin{aligned}\boldsymbol{p}_u - (1-\alpha)\,\boldsymbol{W}\boldsymbol{p}_u &= \alpha\chi_u\\ [\boldsymbol{I} - (1-\alpha)\,\boldsymbol{W}]\,\boldsymbol{p}_u &= \alpha\chi_u\\ \boldsymbol{p}_u &= [\boldsymbol{I} - (1-\alpha)\,\boldsymbol{W}]^{-1}\,\alpha\chi_u.\end{aligned}$$

Before we write this, we should be sure that that the inverse exists. We know that it does because all eigenvalues of $\boldsymbol{W}$ lie between -1 and 1, so all eigenvalues of $\boldsymbol{I} - (1-\alpha)\,\boldsymbol{W}$ are at least α.

11.4 Spilling Paint in a Graph

Just as ordinary random walks were related to a diffusion process, the PageRank random walks are as well. However, we should think of this diffusion process as diffusing paint in a graph. The main characterisic of paint is that it dries.

In our model, we will say that at every time step an α fraction of the paint at each vertex dries in place. As for the wet paint, we assume that half stays where it is and the other half is distributed equally among its neighbors. So, we will need to keep track of two quantities, the amount of wet paint and the amount of dried paint. We will let $\boldsymbol{s} : V \to \mathbb{R}^{\geq 0}$ be the vector the records how much paint has become *stuck* at each vertex, and we will let $\boldsymbol{r} : V \to \mathbb{R}^{\geq 0}$ indicate how much wet paint *remains* at each vertex. At time zero, we set $\boldsymbol{r}^0 = \chi_u$. These vectors now evolve according to the equations

$$\begin{aligned}\boldsymbol{s}^{t+1} &= \boldsymbol{s}^t + \alpha\boldsymbol{r}^t\\ \boldsymbol{r}^{t+1} &= (1-\alpha)\,\widehat{\boldsymbol{W}}\,\boldsymbol{r}^t.\end{aligned}$$

We will be interested in where the paint is stuck in the end. We could denote this by $\boldsymbol{s}^\infty$. We derive the following equation for $\boldsymbol{s}^\infty$:

$$\boldsymbol{s}^\infty = \alpha\sum_{t\geq 0}\boldsymbol{r}^t = \alpha\sum_{t\geq 0}(1-\alpha)^t\,\widehat{\boldsymbol{W}}^t\,\boldsymbol{r}^0 = \alpha\sum_{t\geq 0}(1-\alpha)^t\,\widehat{\boldsymbol{W}}^t\chi_u.$$

We will now see that this is the same equation that the personal PageRank vector satisfies, up to scaling and with a slightly different α. The reason for the different α is that here we used the lazy

walk matrix, whereas in personal PageRank we used the ordinary walk matrix. We will use a fact that is just as true of diagonalizable matrices as it is of real numbers:

$$(\boldsymbol{I} - \boldsymbol{X})^{-1} = \boldsymbol{I} + \boldsymbol{X} + \boldsymbol{X}^2 + \boldsymbol{X}^3 + \cdots,$$

provided that all eigenvalues of $\boldsymbol{X}$ have absolute value less than 1.

So,

$$\boldsymbol{p}_u = \alpha \sum_{t \geq 0} (1-\alpha)^t \boldsymbol{W}^t \chi_u.$$

To see that the difference between using $\boldsymbol{W}$ and $\widehat{\boldsymbol{W}}$ is just a change in α, consider the equation which we now know $\boldsymbol{s}^\infty$ satisfies:

$$\begin{aligned}
\boldsymbol{s}^\infty &= \alpha \left(\boldsymbol{I} - (1-\alpha)\widehat{\boldsymbol{W}}\right)^{-1} \chi_u \\
&= \alpha \left(\frac{1+\alpha}{2}\boldsymbol{I} - \frac{1-\alpha}{2}\boldsymbol{W}\right)^{-1} \chi_u \\
&= \frac{2\alpha}{1+\alpha} \left(\boldsymbol{I} - \frac{1-\alpha}{1+\alpha}\boldsymbol{W}\right)^{-1} \chi_u \\
&= \beta \left(\boldsymbol{I} - (1-\beta)\boldsymbol{W}\right)^{-1} \chi_u,
\end{aligned}$$

where

$$\beta = \frac{2\alpha}{1+\alpha}.$$

I should point out that this gives an algorithm for quickly approximating personal PageRank vectors: just take the first couple terms of the power series. This works as the terms for which $t >> 1/(1-\alpha)$ contribute vanishingly little to the sum.

11.5 Local Updates

From the discussion so far, we can see two obvious ways of computing the vectors $\boldsymbol{p}_u$: either by solving a linear system or by simulating the paint diffusion process. It turns out that there is a very nice way of simulating the paint diffusion process. Berkhin [Ber06], building on work of Jeh and Widom [JW03], observes that it does not need to be done globally through the equations we derived. Rather, we can arbitrarily pick a vertex of the graph, proclaim an α fraction of the wet paint at that vertex dry, and then push the wet paint from that vertex to its neighbors as appropriate. This won't change the answer that we eventually get!

That is, we can ignore time, and do this by a completely asynchronous process. Since we will ignore time, let $\boldsymbol{s}$ be the vector of dried paint and let $\boldsymbol{r}$ be the vector of wet paint. Let's denote by $\boldsymbol{p}_{\boldsymbol{s},\boldsymbol{r}}$ the vector of dried paint that we will eventually compute:

$$\boldsymbol{p}_{\boldsymbol{s},\boldsymbol{r}} = \boldsymbol{s} + \alpha \sum_{t \geq 0} (1-\alpha)^t \boldsymbol{W}^t \boldsymbol{r} = \boldsymbol{s} + \alpha \left(\boldsymbol{I} - (1-\alpha)\boldsymbol{W}\right)^{-1} \boldsymbol{r}.$$

Remark I'm changing from lazy to ordinary walk here as we know it won't make any difference, and it saves ink.

I am claiming that we can now update $\boldsymbol{s}$ and $\boldsymbol{r}$ as follows. Pick an arbitrary vertex u. Now, create the new vectors $\boldsymbol{s}'$ and $\boldsymbol{r}'$ by the rules

$$\begin{aligned}\boldsymbol{s}'(u) &= \boldsymbol{s}(u) + \alpha \boldsymbol{r}(u)\\ \boldsymbol{r}'(u) &= 0\\ \boldsymbol{r}'(v) &= \boldsymbol{r}(v) + \frac{1-\alpha}{d(u)}\boldsymbol{r}(u), \text{for every neighbor } v \text{ of } u.\end{aligned}$$

Lemma 11.5.1.

$$\boldsymbol{p}_{\boldsymbol{s}',\boldsymbol{r}'} = \boldsymbol{p}_{\boldsymbol{s},\boldsymbol{r}}.$$

Proof. In vector notation,

$$\begin{aligned}\boldsymbol{s}' &= \boldsymbol{s} + \alpha \boldsymbol{p}(u)\chi_u, \quad \text{and,}\\ \boldsymbol{r}' &= \boldsymbol{r} - \boldsymbol{p}(u)\chi(u) + (1-\alpha)\boldsymbol{p}(u)\,\boldsymbol{W}\chi_u.\end{aligned}$$

So,

$$\begin{aligned}\boldsymbol{p}_{\boldsymbol{s}',\boldsymbol{r}'} &= \boldsymbol{s}' + \alpha \sum_{t\geq 0}(1-\alpha)^t\,\boldsymbol{W}^t\boldsymbol{r}'\\ &= \boldsymbol{s} + \alpha\boldsymbol{p}(u)\chi_u + \alpha\sum_{t\geq 0}(1-\alpha)^t\,\boldsymbol{W}^t\boldsymbol{r} - \alpha\sum_{t\geq 0}(1-\alpha)^t\,\boldsymbol{W}^t\boldsymbol{p}(u)\chi_u + \alpha\sum_{t\geq 0}(1-\alpha)^t\,\boldsymbol{W}^t(1-\alpha)\,\boldsymbol{W}\chi_u\\ &= \boldsymbol{p}_{\boldsymbol{s},\boldsymbol{r}} + \alpha\boldsymbol{p}(u)\chi_u - \alpha\sum_{t\geq 0}(1-\alpha)^t\,\boldsymbol{W}^t\boldsymbol{p}(u)\chi_u + \alpha\sum_{t\geq 0}(1-\alpha)^t\,\boldsymbol{W}^t(1-\alpha)\,\boldsymbol{W}\chi_u\\ &= \boldsymbol{p}_{\boldsymbol{s},\boldsymbol{r}} + \alpha\boldsymbol{p}(u)\left(\chi_u + \sum_{t\geq 1}(1-\alpha)^t\,\boldsymbol{W}^t\chi_u - \sum_{t\geq 0}(1-\alpha)^t\,\boldsymbol{W}^t\chi_u\right)\\ &= \boldsymbol{p}_{\boldsymbol{s},\boldsymbol{r}}.\end{aligned}$$

□

This led to the idea of computing *approximate PageRank vectors*. The idea behind these is to always pick the vertex for which $\boldsymbol{r}(u)$ is largest, and then distribute the paint from this vertex. Once there is very little paint at every vertex, we stop the process. In particular, we choose some threshold ϵ, and don't bother to process a vertex if it satisfies

$$\boldsymbol{r}(u) \leq \epsilon d(u).$$

Under this situation, one can show that the process will stop within $1/\epsilon\alpha$ iterations.

This leads to a very interesting notion of how one should explore a graph. If you asked me 20 years ago how one should explore a graph from a vertex u, I would have given the obvious answer:

"Breadth First Search". But, for graphs with low diameter, as we now know many are, this is not so useful. I prefer this way of exploring a graph. We only explore nodes when we process them for the first time. Still, this can be improved. Unlike in breadth first search, this process could involve a lot of computation that does not lead to the exploration of new vertices. For example, you can see this if you simulate it on a path graph.

Question Can we improve on this exploration process in a reasonable way?

One improvement has been provided by Andersen and Peres [AP09]. But, I believe we can do better.

11.6 Personal PageRank and Conductance

Andersen, Chung and Lang [ACL06] show that we can use personal PageRank vectors to find sets of low conductance, if we start from a random vector in such a set. Actually, they do this for approximate personal PageRank vectors. This is particularly nice because the number of vertices the algorithm touches is actually proportional to the size of the set that it outputs. So, if there is a small set of low conductance then the algorithm will run very quickly. This is desirable in very large graphs.

Their analysis also uses the holistic approach of Lovàsz and Simonovits that we saw in the last lecture. For a simpler analysis, I recommend that one in the paper of Andersen and Chung [AC07].

As in the last lecture, they find sets by looking for the vertices with the most dried paint, divided by their degree. If we think of the surface area of a vertex as being its degree, then this will involve looking at the darkest vertices.

From the vector $\boldsymbol{p}_v$, we derive the vector $\boldsymbol{q}_v$:

$$\boldsymbol{q}_v(u) = \frac{\boldsymbol{p}_v(u)}{d(u)}.$$

We now assume without loss of generality that the vertices are numbered so that

$$\boldsymbol{q}_v(1) \geq \boldsymbol{q}_v(2) \geq \cdots \geq \boldsymbol{q}_v(n).$$

Let S_k then be the set of vertices $\{1, \ldots, k\}$.

They prove that if we start the process from a random vertex in a set of low conductance, then one of these sets will have low conductance.

Theorem 11.6.1. *Let C be a set with $d(C) \leq d(V)/2$ and conductance less than ϕ. Let $\alpha = \phi/\texttt{const}\log n$. If u is chosen according to $\boldsymbol{\pi}_C$, then with probability at least one half one of the sets S_k constructed from $\boldsymbol{p}_u$ satisfies*

$$\phi(S_k) \leq O(\sqrt{\phi(S)\log m}).$$

Moreover, at least $(2/3)$ of the volume of S_k lies inside C.

So, we don't find a set of the same conductance as C, or even the same set as C, but we do find a small set that substantially overlaps C. They also prove that the constants change very little if we use approximate personal PageRank vectors with $\epsilon = 1/\texttt{const}d(C)$. As the total number of steps of the algorithm is $O(1/\alpha\epsilon)$, the total work done is proportional to $d(C)$, at least when ϕ is big.

Warning: I have written things like $O(\cdot)$ and `const` to hide embarassingly large constants (between 8 and 1000). These are an artifact of theoretical analysis. We find in practice that this algorithm works very well.

There have been many improvements on this algorithm. You can find some on a web page that I maintain: `http://www.cs.yale.edu/homes/spielman/precon/precon.html` .

References

[AC07] Reid Andersen and Fan Chung. Detecting sharp drops in pagerank and a simplified local partitioning algorithm. In Jin-Yi Cai, S. Cooper, and Hong Zhu, editors, *Theory and Applications of Models of Computation*, volume 4484 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2007.

[ACL06] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 475–486, Washington, DC, USA, 2006. IEEE Computer Society.

[AP09] Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 235–244, New York, NY, USA, 2009. ACM.

[Ber06] Pavel Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics*, 3(1):41–62, 2006.

[JW03] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279. ACM, 2003.