Lecture 10 (short summary)

Date: Tuesday, 11th February, 2016

Application: ranking web pages

Perhaps the most direct application of random walk techniques in a practical setting is in ranking web-pages. Consider the *search engine problem*: given a collection of web pages that have links to one another, come up with an "importance ordering" of the pages. This is useful because, given a query, one can then return the top-10 or so (by importance) pages that match the query.

The most natural way to measure the importance of a page is by the number of pages that link to it, or its *in-degree.* This is a natural measure, easy to compute, but it has the issue that all the incoming links are *treated equally.* In particular, suppose a restaurant's page has two incoming links -- one from a "best-of-SLC" page, and another from a "list of restaurants" page. We would want to treat the first edge higher than the second. One way to do so is to weight the edge by the inverse of the out-degree of the 'from' page, and then compute the weighted in-degree. Thus the rank is now

$$rk(u) = \sum_{v:(v,u)\in E} \frac{1}{\deg(v)}.$$

In matrix notation, the rank vector is simply

$$rk = AD_{out}^{-1}\mathbf{1},$$

where $\mathbf{1}$ is the all-one vector, as before. This is a better notion of ranking than the in-degree, but it still treats all the vertices with same outdegree similarly. Intuitively, we want the rank to be higher if there are many "important" pages that link to a given page. This suggests a recursive definition of the rank. From the above, a natural one would be so as to make

$$rk = AD_{out}^{-1}rk.$$

This is precisely the stationary distribution of the random walk corresponding to the directed graph. As we saw last class, such a distribution always exists and is unique if the graph is strongly connected. There are many issues though. For graphs of interest (e.g., the web graph), we might not have strong connectedness. Even if we do, computing the distribution could take exponential time, as we saw earlier.

The PageRank walk

The page-rank walk is an elegant way to fix both the issues above. Consider the walk in which at every step, we either move to a random out-neighbor, or with some probability, to a *random vertex* in the graph. This is called the *random surfer* model, where a surfer either follows random links, or with some probability 'gets bored' and goes to a completely random page, and continues the process.

The transition matrix for the page-rank walk is thus $\alpha\frac{J}{n} + (1-\alpha)AD_{out}^{-1}$, where J is the $n \times n$ matrix with all entries being 1. Here α is the probability of moving to a random vertex -- this is typically referred to as the *teleport probability*. Let us denote the transition matrix M_α. The definition of the walk naturally makes the underlying graph strongly connected, and thus we are guaranteed to have a unique stationary distribution.

How quickly can we compute this distribution? Here, it turns out that performing power iteration a small number of times suffices. This is essentially because for constant α, the terms involved decay very quickly. Think of $\alpha = 1/4$, for now. Then suppose we start with any distribution p and consider $M_\alpha p, M_\alpha^2 p, \dots$. Denoting $W = AD_{out}^{-1}$, we have $M_\alpha p = \alpha \mathbf{1}/n + (1-\alpha)Wp$, which implies

$$M_\alpha^2 p = \alpha\frac{\mathbf{1}}{n} + \alpha(1-\alpha)W\frac{\mathbf{1}}{n} + (1-\alpha)^2 W^2 p.$$

Continuing this, we see that

$$M_\alpha^r p = \left(\sum_{j=0}^{r-1} \alpha(1-\alpha)^j W^j \frac{\mathbf{1}}{n}\right) + (1-\alpha)^r W^r p.$$

Now, the terms $(1-\alpha)^j$ drop very quickly. After $j \approx \log n$, the contributions become negligible. This is why computing the page-rank takes only around $\log n$ iterations.

Suggested reading: the original paper of Page, Brin, Motwani and Winograd.

Recap of walks in undirected graphs

Before moving on, let us recap our discussion of lazy walks on undirected graphs. The matrix for a lazy walk (with laziness parameter 1/2) is $(1/2)I + (1/2)AD^{-1}$. We saw last time that the stationary distribution is simply the vector whose *i*th entry is $\deg(i)/2m$, where m is the number of edges in the graph. If the graph is connected, any walk converges to this distribution, and the time for convergence is related to the gap between the two largest eigenvalues of the normalized adjacency matrix $(D^{-1/2}AD^{-1/2})$.

Now, this gap is precisely equal to the gap between the two *smallest* eigenvalues of the normalized *Laplacian*, which is $I - D^{-1/2}AD^{-1/2}$. (Since the eigenvalues are just one minus the eigenvalues of the other.) This quantity, as we saw, is related to the *expansion* of the graph, $\Phi(G)$, by Cheeger's inequality. For *d*-regular graphs, recall that we defined
$\Phi(G) = \min_{|S| \le n/2} E(S, \overline{S})/d|S|.$

For general graphs, the right way of defining this is:
$\Phi(G) = \min_{S:\mathrm{vol}(S) \le m} E(S, \overline{S})/\mathrm{vol}(S),$
where the *volume* of a set S is simply the sum of the degrees of the vertices the set, and *m* is the number of edges. Verify that when the graph is *d*-regular, this becomes the same as the earlier definition.

Cheeger's inequality basically says that if the gap between the eigenvalues (described above) is δ, then
$\Phi(G) \le \sqrt{2\delta}.$

Now consider a graph in which $\Phi(G) \ge 1/4$ (or any small constant). Cheeger's inequality implies that $\delta = \Omega(1)$ for such a graph. Thus the convergence time of a lazy random walk is $O(\log n/\delta) = O(\log n)$.

This is one reason such graphs are very interesting. To generate a random vertex in the graph, one can start at any vertex, and perform a lazy random walk for $O(\log n)$ steps, and return the obtained vertex! Graphs with the property above, i.e., $\Phi(G) \ge 1/4$ are called ***expander graphs***.

Is a given graph an expander?

Given the property above (and several other nice properties) that expander graphs have, one natural question is to *prove* that some graph of interest is an expander. One way to prove is to compute the eigenvalue gap of the graph, and use the connection between the two. However, it might be difficult to reason about the eigenvalue gap as well.

A technique that works for many graphs is the so-called *canonical paths* method. Let us illustrate this technique for *d*-regular graphs. Recall that the goal is to prove that $\Phi(G)$ is large, i.e., for every set S of vertices, $E(S, \overline{S}) \ge c.d|S|$, for some *c.* The main "philosophical" difficulty with such a question is to prove it *for all* sets (there are exponentially many candidates S). Showing that $\Phi(G)$ is small is relatively easy -- we simply have to exhibit one set *S* such that $E(S, \overline{S})$ is small. (This is an instance of the "NP vs co-NP" question.)

So how does the canonical paths method work? The idea is roughly what we saw in Lecture 3. Suppose for every pair of vertices *u, v* in the graph, we give a path P_{uv}, with the additional condition that the number of paths going through *any* edge is at most *M,* for some parameter *M*. Then for any set *S* of size at most *n/2* we must have $M \cdot E(S, \overline{S}) \geq |S||\overline{S}|$, which implies, $\dfrac{E(S,\overline{S})}{|S|} \geq \dfrac{n}{2M}$.

Thus if we can come up with a collection of paths such that *M* is small, then we get a good bound for the expansion. Can this be done for a given graph?

To illustrate, we consider the *d*-dimensional hypercube. This is a graph with 2^d vertices. Every vertex corresponds to a string of length *d.* Two strings have an edge if they differ in exactly one bit. E.g., with *d=4*, the strings 0011 and 0010 have an edge, while 0011 and 1001 do not (because they differ in two bits). For d=2, the graph corresponds to a square, for d=3, it corresponds to the 3-dimensional cube, and for higher *d*, it is the natural extension of a cube (and hence the name).

Note that the *d* dimensional hypercube has degree precisely *d* (one corresponding to each bit that could be different). The question we ask is, what is the expansion of this graph? We will show that for every subset of vertices S of size at most *n/2*, $E(S, \overline{S})/|S|$ is big. From the discussion above, it suffices to come up with a path P_{uv} for every pair of vertices in the graph, such that no edge has too many paths going through (i.e., the max *congestion* is small).

For two strings $u = u_1 u_2 \dots u_d$ and $v = v_1 v_2 \dots v_d$, consider the path that moves from left to right, *correcting* one bit at a time. For instance, to go from string 0000 to 1011, it takes the path 0000 -> 1000 -> 1010 -> 1011. The number of edges in the path is exactly the number of bits that need to be flipped to go from *u* to *v*.

Now, for this collection of paths, what is the number of paths that go through a single edge? Consider the edge $a_1 a_2 \dots a_{i-1} 0 a_{i+1} \dots a_d, a_1 a_2 \dots a_{i-1} 1 a_{i+1} \dots a_d$. What paths P_{uv} use this edge? A little thought reveals that for a path to use the edge, we must have $v_1 v_2 \dots v_{i-1} v_i = a_1 a_2 \dots a_{i-1} 1$ (because the first *i-1* bits have already been "corrected"). Furthermore, we must have $u_i u_{i+1} \dots u_d = 0 a_{i+1} \dots a_d$, because the last bits have not been touched yet! Thus the total number of (*u, v*) pairs is exactly 2^{d-1}. Thus we have a set of paths with each edge being part of at most $M = 2^{d-1}$ paths.

Thus, from the reasoning earlier, we have $E(S,\overline{S})/|S| \geq 1$ for all subsets *S*. This implies that the expansion is at least $1/d$. This in fact turns out to be tight, for the hypercube, as we will see.