Stat260/CS294: Spectral Graph Methods **Lecture 19 - 04/02/2015**

Lecture: Local Spectral Methods (2 of 4)

Lecturer: Michael Mahoney *Scribe: Michael Mahoney*

Warning: these notes are still very rough. They provide more details on what we discussed in class, but there may still be some errors, incomplete/imprecise statements, etc. in them.

19 Computing spectral ranking with the push procedure

Last time, we talked about spectral ranking methods, and we observed that they can be computed as eigenvectors of certain matrices or as the solution to systems of linear equations of certain constraint matrices. These computations can be performed with a black box solver, but they can also be done with specialized solvers that take advantage of the special structure of these matrices. (For example, a vanilla spectral ranking method, e.g., one with a preference vector $\vec{v}$ that is an all-ones vector $\vec{1}$, has a large eigenvalue gap, and this means that one can obtain a good solution with just a few steps of a simple iterative method.) As you can imagine, this is a large topic. Here, we will focus in particular on how to solve for these spectral rankings in the particular case when the preference vector $\vec{v}$ is has small support, i.e., when it has its mass localized on a small seed set of nodes. This is a particularly important use case, and the methods developed for it are useful much more generally.

In particular, in this class, we will describe how to compute personalized/local spectral rankings with a procedure called the *push procedure*. This has several interesting properties, in general, but in particular when computing locally-biased or personalized spectral rankings in a large graph. As with the last class, we will follow the Boldi-Vigna's notes on "The Push Algorithm for Spectral Ranking" on the topic.

19.1 Background on the method

To start, let $M \in \mathbb{R}^{n \times n}$ be a nonnegative matrix; and WLOG assume that $\|M\|_1 = 1$, i.e., assume that M is stochastic. (Actually, this assumes that M is sub-stochastic and that at least one row of M sums to 1; this distinction won't matter for what we do, but it could matter in other cases, e.g., when dealing with directed graphs, etc.) Then, let v be a nonnegative vector s.t. $\|v\|_1 = 1$, i.e., assume that v is a distribution, and let $\alpha \in [0, 1)$.

Then, recall that we defined the spectral ranking of M with preference vector v and damping factor α to be

$$\begin{aligned} r &= (1-\alpha)\, v\, (I - \alpha M)^{-1} \\ &= (1-\alpha)\, v \sum_{k=0}^{\infty} \alpha^k M^k. \end{aligned}$$

For this, note that r need not be a distribution unless M is stochastic (although it is usually applied

in that case). Note also that the linear operator is defined for all $\alpha \in \left[0, \frac{1}{\rho(M)}\right)$, but estimating $\rho(M)$ can be difficult. (We'll see an interesting example next week of what arises when we push this parameter to the limit.)

From this perspective, while it is difficult to "guess" what is the spectral ranking r associated with a preference vector v, it is "easy" to solve the inverse problem: given r, solve

$$v = \frac{1}{1-\alpha} r \left(I - \alpha M\right).$$

(While this equation is always true, the resulting preference vector might not be a distribution; otherwise, one could obtain any spectral ranking using a suitable preference vector.)

While this is obvious, it has important consequences for computing spectral rankings and approximate spectral rankings that we will now describe.

Consider the indicator vector $\mathcal{X}_x(z) = [x = z]$. If we want to obtain the vector $(1-\alpha)\,\mathcal{X}_x$ as a spectral ranking, then the associated preference vector v has the simple form:

$$\begin{aligned} v &= \frac{1}{1-\alpha}(1-\alpha)\,\mathcal{X}_x\,(I - \alpha M) \\ &= \mathcal{X}_x - \alpha \sum_{x \to y} M_{xy} \mathcal{X}_y \\ &= \mathcal{X}_x - \alpha \frac{1}{d(x)} \sum_{x \to y} \mathcal{X}_y, \end{aligned} \tag{1}$$

where note that if $M_{xy} = \frac{1}{d(x)}$ is the natural random walk, then we can take it out of the summation.

This is true in general. The important point for us here is that if v is highly concentrated, e.g., if it is an indicator vector of s small set of nodes, and if α is not too close to 1, then most of the updates done by the linear solver or by an iterative method either don't update much or update below a threshold level. Motivated by this, we will discuss the so-called *push algorithm*—this uses a particular form of updates to reduce computational burden. This was used in work of Jeh and Widom and also of Berkhin on personalized page rank, and it was used with this name by ACL. Although they all apply it to PageRank, it applies for the steady state of Markov chains with restart, and it is basically an algorithm for spectral ranking with damping (see the Boldi, Lonati, Santini, Vigna paper).

19.2 The basic push procedure

The basic idea of this approach is that, rather that computing an *exact* PPR vector by iterating the corresponding Markov chain (e.g., with vanilla matrix-vector multiplies) until it converges, it is also possible to consider computing an *approximate* PPR vector much more efficiently. Recall that the PPR vector is the unique solution to

$$\pi_\alpha(s) = (1-\alpha)\,s + \alpha \pi_\alpha(s) W,$$

and it can be written as an infinite sum

$$\pi_\alpha(s) = (1-\alpha)\,s + (1-\alpha) \sum_{t=1}^{\infty} \alpha^t \left(sW\right)^t.$$

With this notation, we can define the following notion of approximation of a PPR vector.

Definition 1. *An* ϵ-approximate PageRank vector $\pi_\alpha(s)$ *is any PageRank vector* $\pi_\alpha(s-r)$*, where* r *is nonnegative and* $r(v) \leq \epsilon d_v$*, for all* $v \in V$*.*

Fact. The approximation error of an ϵ-approximate PageRank vector on any set of nodes S can be bounded in terms of the Vol(S) and ϵ. Here is a basic lemma from ACL.

Lemma 1. *For all* ϵ*-approximate PageRank vectors* $\pi_\alpha(s-r)$*, and for all* $S \subset V$*, we have*

$$\pi_\alpha(s)1_S^T \geq \pi_\alpha(s-r)1_S^T \geq \pi_\alpha(s)1_S^T - \epsilon \mathit{Vol}(S).$$

Here is an algorithm to compute an ϵ-approximate PageRank vector; let's call this algorithm ApproxPR(s, α, ϵ).

1. Let $p = \vec{0}$ and $r = \vec{s}$.
2. While $r_u \geq \epsilon d_u$ for some vertex u,
   - Pick and u such that $r_u \geq \epsilon d_n$
   - Apply Push(u)
3. Return the vectors p and r

And here is the Push(u) algorithm that is called by the ApproxPR(s, α, ϵ) algorithm.

1. Let $p' = p$ and $r' = r$, except for the following updates:
   - $p'_u = p_u + \alpha r_u$
   - $r'_u = (1-\alpha)\frac{r_u}{2}$
   - $r'_v = r_v + (1-\alpha)\frac{r_u}{2d_u}$, for all vertices v such that $(u, v) \in E$.

Note that we haven't specified the order in which the pushes are executed, i.e., in which the Push(u) algorithm is called by the ApproxPR(s, α, ϵ) algorithm, and so they can be done in different ways, leading to slightly different algorithms.

Here is the theorem that ACL establishes about this procedure.

Theorem 1. *Algorithm* ApproxPR(s, α, ϵ) *has the following properties.*

- *For all starting vertices with* $\|s\|_1 \leq 1$ *and for all* $\epsilon \in (0, 1]$*, the algorithm returns an* ϵ*-approximate* p *for* $p_\alpha(s)$*.*
- *The support of* p *satisfies*
$$\mathit{Vol}(\mathit{Supp}(p)) \leq \frac{2}{(1+\alpha)\epsilon}.$$
- *The running time of the algorithm is* $O\left(\frac{1}{\alpha\epsilon}\right)$*.*

The idea of the proof—outlined roughly above—is that Algorithm Push(u) preserves the approximate PageRank condition $p = \pi(s-r)$; and the stopping condition ensures that it is an ϵ approximation. The running time follows since $\|r\|_1 = 1$ and it decreases by $\alpha\epsilon d_n$ at each time Push(u) is called.

19.3 More discussion of the basic push procedure

The basic idea of the method is to keep track of two vectors, p which is the vector of current approximations and r which is the vector of residuals, such that the following global invariant is satisfied at every step of the algorithm.

$$p + (1-\alpha)\, r\, (I - \alpha M)^{-1} = (1-\alpha)\, v\, (I - \alpha M)^{-1}. \tag{2}$$

Initially, $p = 0$ and $r = v$, and so this invariant is trivially satisfied. Subsequently, at each step the push method increases p and decreases r to keep the invariant satisfied.

To do so, it iteratively "pushes" some node x. A *push on* x adds $(1-\alpha)\, r_x \mathcal{X}_x$ to the vector p. To keep the invariant try, the method must update r. To do so, think of r as a preference vector, in which we are just trying to solve Eqn. (1). By linearity, if we subtract

$$r_x \left(\mathcal{X}_x - \alpha \sum_{x \to y} M_{xy} \mathcal{X}_y \right)$$

from r, then the value $(1-\alpha)\, r\, (I - \alpha M)^{-1}$ will decrease by $(1-\alpha)\, r_x \mathcal{X}_x$, thus preserving the invariant.

This is a good choice since

- We zero out the x^{th} entry of r.
- We add a small positive quantities to a small set of entries (small set, if the graph is sparse, which is the use case we are considering). This increases the ℓ_1 norm of p by $(1-\alpha)\, r_x$, and it decreases r by at least the same amount.

Since we don't create negative entries in this process, it always holds that

$$\|p\|_1 + \|r\|_1 \leq 1,$$

and thus we can keep track of two norms at each update. The error in the estimate is then given by the following.

$$\begin{aligned} \| (1-\alpha)\, r\, (I - \alpha M)^{-1} \|_1 &= (1-\alpha)\, \| r \sum_{k \geq 0} \alpha^k M^k \|_1 \\ &\leq (1-\alpha)\, \|r\|_1 \sum_{k \geq 0} \alpha^k \|M^k\|_1 \\ &\leq \|r\|_1 \end{aligned}$$

So, in particular, we can control the absolute error of the algorithm by controlling the ℓ_1 error of the residual.

19.4 A different interpretation of the same process

Here is a different interpretation of the push method. Some might find useful—if so, good, and if not, then just ignore the following.

Just as various random walks can be related to diffusion of heat/mass, one can think of PageRank as related to the diffusion of a substance where some fraction of that substance gets stuck in place

at each time step—e.g., diffusing paint, where the point of paint is that part of the paint dries in place and then stops diffusing/flowing. (Think here of doing the updates to push asynchronously.) At each time step, $1-\alpha$ fraction of the paint dries in place, and α fraction of the paint does a lazy random walk. So, we need to keep track of two quantities: the amount of wet paint (that still moves at the next step) and the amount of dry paint (that is probability mass that won't move again). If we let

- $p : V \to \mathbb{R}^n$ be a vector that says how much pain is stuck at each vertex.
- $r : V \to \mathbb{R}^n$ be a vector saying how much wet paint remains at each vertex.

Then, at $t = 0$, $r^0 = \mathcal{X}_u$, and these vectors evolve as

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

(Note that I have swapped sides where the vector is multiplying, so I think I have inconsistencies here to fix. Also, I think I refer to α and $1-\alpha$ inconsistently, so that must be fixed too)

Given this, if we let p^∞ be where paint is dried at the end of the process, then

$$\begin{aligned} p^\infty &= (1-\alpha) \sum_{t \geq 0} r^t \\ &= (1-\alpha) \sum_{t \geq 0} \alpha^t r^0 \hat{W}^t \\ &= \sum_{t \geq 0} \alpha^t \mathcal{X}_u \hat{W}^t \end{aligned}$$

This is simply PPR, with a different scaling and α. Recall here that $\hat{W} = \frac{1}{2} I + \frac{1}{2} W$. Since $(I+X)^{-1} = \sum_{i=0}^{\infty} X^i$, if the spectral radius of X is less than 1, it follows that

$$\begin{aligned} p^\infty &= (1-\alpha)\, \mathcal{X}_u \left(I - \alpha \hat{W} \right)^{-1} \\ &= (1-\alpha)\, \mathcal{X}_u \left(\left(1 - \frac{\alpha}{2} \right) I - \frac{\alpha}{2} W \right)^{-1} \\ &= \gamma \mathcal{X}_u \left(I - (1-\gamma) W \right)^{-1}, \end{aligned}$$

where the parameter γ is defined in terms of the parameter α.

Note that in this we don't need to update the above time process but we can ignore time and do it in an asynchronous manner. That is, we can compute this by solving a linear equation or running a random walk in which one keeps track of two vectors that has this interpretation in terms of diffusing paint. But, Jeh-Widom and Berkhin note that rather than doing it with these equations, one can instead choose a vertex, say that a fraction α of paint at that vertex is dry, and then push the wet paint to the neighbors according to the above rule. (That is, we can ignore time and do it asynchronously.) But this just gives us the push process. To see this, let $\pi_{p,r}$ be the vector of "dried paint" that we eventually compute. Then

$$\begin{aligned} \pi_{p,r} &= p + (1-\alpha) \sum_{t \geq 0} r \alpha^t W^t \\ &= p + (1-\alpha)\, r \left(I - \alpha W \right)^{-1} \end{aligned}$$

In this case, the updates we wrote above, written another way, are the following: pick a vertex u and create p' and r' as follows.

- $p'(u) = p(u) + (1-\alpha)\, r(u)$
- $r'(u) = 0$
- $r'(v) = r(v) + \frac{\alpha}{d(u)} r(u)$, for all neighbors v of u.

Then, it can be shown that

$$\pi_{p',r'} = \pi_{p,r},$$

which is the invariant that we noted before.

So, the idea of computing the approximate PageRank vectors is the following.

- Pick a vertex where $r(u)$ is large or largest and distribute the paint according to that rule.
- Choose a threshold ϵ and don't bother to process a vertex if $r(u) \leq \epsilon d(u)$.

Then, it can be shown that

Lemma 2. *The process will stop within $\frac{1}{\epsilon(1-\alpha)}$ iterations.*

(Note there is an inconsistency with α and $1-\alpha$, i.e., whether α is the teleporting or non-teleporting probability, that needs to be fixed.)

19.5 Using this to find sets of low-conductance

This provides a way to "explore" a large graph. Two things to note about this.

- This way is very different that DFS or BFS, which are not so good if the diameter is very small, as it is in many real-world graphs.
- The sets of nodes that are found will be different than with a geodesic metric. In particular, diffusions might get stuck in good conductance sets.

Following up on that second point, ACL showed how to use approximate PPR to find sets of low conductance, if they start from a random vector in a set. Importantly, since they do it with the Approximate PPR vector, the number of nodes that is touched is proportional to the size of the output set. So, in particular, if there is a set of small conductance, then the algorithm will run very quickly.

Here is the basic idea. ACL first did it, and a simpler approach/analysis can be found in AC07.

- Given π_v, construct $q_v(u) = \frac{\pi_v(u)}{d(u)}$.
- Number the vertices, WLOG, such that $q_v(1) \geq q_v(2) \geq \cdots q_v(n)$, and let $S_k = \{1, \ldots, k\}$.

Then, it can be shown that starting from a random vertex in the set of low conductance, then one of the sets has low conductance.

Computationally, these and related diffusion-based methods use only local information in the graph. If one can look at the entire graph, then one can get information about global eigenvectors, which approximate sparse cuts. But, if the graphs are huge, then one might be interested in the following question:

- Given a graph $G = (V, E)$ and a node $v \in V$, find a set of nodes V such that the Cheeger ratio h_S is small.

Recall that $h_S = \frac{|E(S,\bar{S})|}{\min\{\text{Vol}(S),\text{Vol}(\bar{S})\}}$. Before, we were interested in good global clusters, and thus we defined $h_G = \min_{S \subset V} h_S$ and tried to optimize it, e.g., by computing global eigenvectors; but here we are not interested in global eigenvectors and global clusters.

Note that it is not immediately obvious that diffusions and PageRank are related to this notion of local clustering, but we will see that they are. In particular, we are interested in the following.

- Given a seed node s, we want to find a small set of nodes that is near s and that is well connected internally but that is less well connected with the rest of the graph.

The intuition is that if a diffusion is started in such a cluster, then it is unlikely to leave the cluster since the cluster is relatively poorly-connected with the rest of the graph. This is often true, but one must be careful, since a diffusion that starts, e.g., at the boundary of a set S can leave S at the first step. So, the precise statement is going to be rather complicated.

Here is a lemma for a basic diffusion process of a graph G.

Lemma 3. *Let $C \subset V$ be a set with Cheeger ratio h_C, and let t_0 be a time parameter. Define C_{t_0} to be a subset of C such that for all $v \in C_{t_0}$ and all $t \leq t_0$, we have that $\vec{1}_v^T W^t \vec{1}_{\bar{C}}^T \leq t_0 h_C$. Then, the volume of C_{t_0} is such that*

$$Vol(C_{t_0}) \geq \frac{1}{2} Vol(C) .$$

This is a complicated lemma, but it implies that for vertex set $C \subset V$ with small Cheeger ratio, that there are many nodes $v \in C$ such that the probability of a random walk starting at v leaves C is low.

There is a similar result for the PageRank diffusion process. The statement is as follows.

Lemma 4. *Let $C \subset V$ have Cheeger ratio h_C, and let $\alpha \in (0, 1]$. Then, there is a subset $C_\alpha \subseteq C$ with volume $Vol(C_\alpha) \geq \frac{1}{2} Vol(C)$ such that for all $v \in C_\alpha$, the PageRank vector $\pi_\alpha(1_V)$ satisfies*

$$\pi_\alpha (1_V) \, \vec{1}_C^T \geq 1 - \frac{h_C}{\alpha}.$$

This is also a complicated lemma, but it implies that for any vertex set $C \subseteq V$ with small h_C, that there are many nodes $v \in C$ for which PPR, using v as a start node, is small on nodes outside of C.

These and related results show that there is a relationship or correlation between the Cheeger ratio and diffusion processes, even for "short" random walks do not reach the asymptotic limit. In

particular, for a set $C \subset V$ with small h_C, it is relatively hard for a diffusion process started within C to leave C.

We can use these ideas to get *local spectral clustering algorithms*. To do this, we need a method for creating vertex cuts from a PR or random walk vector. Here is the way.

- Say that $\pi_\alpha(s)$ is a PPR vector.
- Then, create a set C_S by doing a sweep cut over $\pi_\alpha(s)$.

That is, do the usual sweep cut, except on the vector returned by the algorithm, rather than the leading nontrivial eigenvector of the Laplacian.

Here is one such lemma that can be proved about such a local spectral algorithm.

Lemma 5. *If $\pi_\alpha(s)$ is a PPR vector with $\|s\|_1 \leq 1$ and there exists $S \subseteq V$ and a constant δ such that $\pi_\alpha(s)\vec{1}_S - \frac{Vol(S)}{Vol(G)} > \delta$, then*

$$h_{C_s} < \left(\frac{12\alpha \log\left(4\sqrt{Vol(S)}/\delta\right)}{\delta} \right)^{1/2}.$$

This too is a complicated lemma, but the point is that if there exists a set S where the PPR is much larger than the stationary distribution $\frac{d}{\mathrm{Vol}(G)}$, then a sweep cut over $\pi_\alpha(S)$ produces a set C_S with low Cheeger ratio: $O\left(\sqrt{\alpha \log\left(\mathrm{Vol}(S)\right)}\right)$.

This result is for PPR; that there is a similar result for *approximate* PPR, which has a more complicated statement still.

Two things to note.

- While the statements of these theoretical results is quite complicated, these methods do very well in practice in many applications. (We won't discuss this much.)
- These algorithms were originally used as primitives for the Laplacian solvers, a topic to which we will return in a few weeks.

While one can prove results about the output of these algorithms, one might also be interested in what these algorithms optimize. Clearly, they *approximately* optimize something related to a local version of the global spectral partitioning objective; so the question is what do they optimize *exactly*. This is the topic we will turn to next—it will turn out that there are interesting connections here to with implicit regularization ideas.