**CSE 521: Design and Analysis of Algorithms I** **Winter 2017**

# Lecture 14: Random Walks, Local Graph Clustering, Linear Programming

*Lecturer: Shayan Oveis Gharan* *3/01/17* *Scribe: Laura Vonessen*



## 14.1 Random Walks

### 14.1.1 Useful matrix comparison

The following follows from the statements that we proved in the last lecture. It shows a nice correspondence between the eigenvalues and eigenvectors of three matrices, normalized Laplacian, normalized adjacency, and the transition probability matrix. We will use this correspondence in this lecture to study mixing time of random walks.

| Matrix name | Normalized Laplacian | Normalized adjacency matrix | Random walk | "Lazy" random walk |
|---|---|---|---|---|
| Formula | $\tilde{L} = I - \tilde{A}$ | $\tilde{A} = I - \tilde{L} = D^{-1/2}AD^{-1/2}$ | $P = D^{-1}A$ | $\tilde{P} = \frac{P+I}{2}$ |
| PSD | yes | no | no | yes |
| Eigenvalue | $\lambda_i$ ($i$-th smallest) | $1-\lambda_i$ ($i$-th largest) | $1-\lambda_i$ | $1-\frac{\lambda_i}{2}$ |
| Left eigenvector | $\tilde{L}v_i = \lambda_i v_i$ | $\tilde{A}v_i = (1-\lambda_i)v_i$ | $PD^{-1/2}v_i = (1-\lambda_i)D^{-1/2}v_i$ | $\tilde{P}D^{-1/2}v_i = (1-\lambda_i/2)D^{-1/2}v_i$ |
| Right eigenvector | $v_i\tilde{L} = \lambda_i v_i$ | $v_i\tilde{A} = (1-\lambda_i)v_i$ | $v_iD^{+1/2}P = (1-\lambda_i)D^{1/2}v_i$ | $v_iD^{1/2}\tilde{P} = (1-\lambda_i/2)D^{1/2}v_i$ |
| $\lambda_1$ | $\lambda_1 = 0$ | $1-\lambda_1 = 1$ | $1-\lambda_1 = 1$ | $1-\frac{\lambda_1}{2} = 1$ |
| $v_1$ | $D^{1/2}1$ | $D^{1/2}1$ | $1D$ (right) | |

As an exercise, let us show that the largest right eigenvector $v_1$ of $P$, corresponding to eigenvalue of 1 is $\mathbf{1}D = d_w$, i.e., it is the vector which has the degree $d_w(i)$ at the $i$-th coordinate.

$$\mathbf{1}DP = \mathbf{1}DD^{-1}A = \mathbf{1}A = \mathbf{1}D.$$

### 14.1.2 Stationary distribution

Consider transition matrix $P$, where you transition from $i$ to $j$ with probability $P_{i,j}$. Suppose we start the random walk from state $X_0 = i$, then the state at time 1 is distributed according to $\mathbf{1}_iP$, i.e., $\mathbb{P}[X_1 = j | X_0 = i] = P_{i,j}$ which is just the $i$-th row of $P$. It is not hard to see that for any probability distribution $x$, if the position at time 0, $X_0$, is distributed according to $x$, then $X_1$ is distributed according

to $xP$. Having this mind, if we start a random walk from state $i$, the distribution of the walk at time $t$ is

$$\underbrace{(((\mathbf{1}_i P)P)\dots P)}_{t \text{ times}} = \mathbf{1}_i P^t.$$

**Definition 14.1** (Stationary distribution)**.** *We say a probability distribution $x$ is an* stationary distribution *of $P$ if $xP = x$. We typically denote the stationary distribution by $\pi$. So, $\pi$ is the stationary distribution if for all states $j$,*

$$\pi_j = \sum_{i=1}^{n} \pi_i P_{i,j}.$$

*Note that this means $\pi$ is a left eigenvector of $P$ with eigenvalue 1.*

In the previous section we showed that for any transition probability matrix $P$, $\mathbf{1}D$ is the left eigenvector of $P$ with eigenvalue 1. Note that $\mathbf{1}D$ have positive coordinates but the coordinates do not sum up to 1. So, we need to normalize it by summation of the degree of all vertices. The following fact follows:

**Fact 14.2.** *For any undirected graph $G$, the vector $\pi$ with coordinates*

$$\pi_i = \frac{d_w(i)}{\sum_j d_w(j)}$$

*is a stationary distribution of the walk.*

For example, for any regular graph $G$, the uniform distribution $\pi = \mathbf{1}/n$ is the stationary distribution.

The stationary distribution for directed graphs (and general Markov chains) is not predetermined based on the degrees. One can run the power method or analogs like the pagerank algorithm to estimate the stationary distribution. This is in fact the idea that Google use to rank the importance of webpages.

**Definition 14.3** (Reversible Markov Chains)**.** *We say a transition probability matrix $P$ is* reversible *if there exists a $\pi$ such that*

$$\pi_i P_{i,j} = \pi_j P_{j,i} \ \forall \ i, j.$$

*It turns out that reversible Markov chains are equivalent to random walk on undirected graph $G$. In particular, we can construct a weighted graph for for all $i, j$,*

$$w_{i,j} = \pi_i P_{i,j}$$

*and it follows that the transition probability matrix associated to $G$ is the same as $P$.*

In this lecture, we only talk about reversible Markov chains or random walks on undirected graphs.

### 14.1.3 Fundamental theorem of Markov chains

The following is one of the fundamental theorems of this area. It says that for any (non necessarily reversible) Markov chain if a certain technical conditions are satisfied, then the chain has a unique stationary distribution $\pi$ and from every starting distribution $x$, we converge to $\pi$ if we run the chain for a long enough number of steps.

**Theorem 14.4** (Fundamental Theorem of Markov Chains)**.** *For any Markov chain $P$, if $P$ is irreducible and aperiodic, there exists unique stationary distribution $\pi$ such that for all probability distributions $x$,*

$$xP^t \underset{t\to\infty}{\longrightarrow} \pi.$$

Now, let us discuss the technical terms in the above theorem for reversible Markov chains.

**Definition 14.5** (Irreducible)**.** *We say transition probability matrix $P$ corresponding to $G$ is* irreducible *if and only if $G$ is connected.*

**Definition 14.6** (Aperiodic)**.** *We say the transition probability matrix $P$ corresponding to $G$ is* aperiodic *if and only if $G$ is not bipartite.*

To see why irreducibility is required, consider the graph below on the left. It has two stationary distributions, one that stays at the node on the left, and one that stays at the node on the right, so there is no *unique* stationary distribution.


To see why aperiodicity is required, consider the graph above on the right, and suppose we start a walk from the left node. At each step, the random walker will move from one node to the other. It follows that the walker is on the left node on even time steps and is on the right node on odd time steps. Therefore, $xP^t$ will not *converge*.

Let us give an intuitive reason for the above theorem. First of all recall that the largest eigenvalue of $P$ is 1, and the corresponding eigenvector is $\pi$. So, to prove the above theorem we want to say if we run the chain long enough, then the largest eigenvalue/eigenvector will be the dominating terms in $xP^t$, i.e., the walk mixes. If $G$ is not connected, then the second largest eigenvalue is also 1. So, $xP^t$ is converges to the span of the first two eigenvectors, i.e., there are at least two stationary distributions.

Now, let's study the aperiodicity. In general one can show that any graph $G$ is bipartite if and only if the largest eigenvalue of $\tilde{L}$ is 2, or equivalently the smallest eigenvalue of $P$ is $-1$. But the latter means that for even values of $t$, the eigenvector of $-1$ and the eigenvector of 1 will be the dominating terms in $xP^t$; so again the stationary distribution is not unique.

On the other hand, if $G$ is not bipartite and connected one can show that the eigenvalues of $\tilde{L}$ are in the interval

$$0 < \lambda_2 \leq \lambda_3 \leq \cdots \leq \lambda_n < 2.$$

Therefore, for $\varepsilon = \min\{\lambda_2 - 0, 2 - \lambda_n\}$, if we run the chain for $O(\log n/\varepsilon)$ steps, the eigenvalue of 1 will be the dominating term in $xP^t$.

### 14.1.4 Mixing time

The idea of mixing time is to get an upper bound on the number of steps required to get $\varepsilon$ close to the stationary distribution, no matter which starting point you take. It is defined with the following formula:

$$\tau(\varepsilon) = \max_x \min\{t : \|xP^t - \pi\|_1 \leq \varepsilon\}$$

It can be seen that the $\ell_1$ distance is a natural measure to study the similarity of two probability distributions. In particular, if $\mu$ and $\nu$ are probability distributions, then

$$\|\mu - \nu\|_1 \leq \varepsilon \iff \forall\, \mathcal{A}, |\mathbb{P}_\mu[\mathcal{A}] - \mathbb{P}_\nu[\mathcal{A}]| \leq 2\varepsilon.$$

In the above $\mathcal{A}$ refers to the set of all possible events on the state space. For example, it could be all even numbered vertices $2, 4, \ldots,$. In other words, the $\|xP^t - \pi\| \leq \varepsilon$ it means that any probabilistic even of interest has almost the same probability of happening under $xP^t$ and $\pi$.

### 14.1.5 Lazy random walk

Clearly, every disconnected graph will have at least one stationary distribution per connected component. Can we do something slightly change bipartite graphs such that we can still talk about their stationary distribution?

Idea of *lazy* random walk: at every vertex, with probability 1/2 stay at the vertex and with probability 1/2 choose a neighbor to walk to as before. This leads to the following transition probability matrix

$$\tilde{P} = \frac{I + P}{2}$$

The above matrix $\tilde{P}$ is PSD, so we no longer need to worry about the smallest eigenvalue when we study the time it takes for the first eigenvector to be the dominating term in $x\tilde{P}^t$. See the table at the beginning of these notes for associated eigenvalues/eigenvectors of this matrix.

### 14.1.6 Spectral Upperbound on Mixing Time

It must now be clear from the above arguments that the same proof of the power method gives a natural upper bound on the mixing time. Note that we want to choose $t$ big enough such that $\pi$ will be the dominating term in $x\tilde{P}^t$. Since $\tilde{P}$ is PSD, the power method says that we need to choose $\varepsilon$ to be the difference of the largest and the second largest eigenvalue and run the chain for $O(\log n/\varepsilon)$ steps. As we discussed at the beginning of this lecture, the largest eigenvalue is 1 and the second largest eigenvalue is $1 - \lambda_2$. So, we need to choose $\varepsilon = 1 - (1 - \lambda_2) = \lambda_2$. The following theorem follows by the same proof of the power method.

**Theorem 14.7.** *For any connected graph $G$, $\forall\, \varepsilon > 0$, the mixing time of the lazy random walk on $G$ is*

$$\tau(\varepsilon) \leq \frac{4}{\lambda_2} \log\left(\frac{1}{\varepsilon \min \pi_i}\right)$$

*where as usual $\lambda_2$ is the second smallest eigenvalue of $\tilde{L}$.*

Now, let us discuss several examples.

If $G$ is a cycle, we saw that $\lambda_2 = O(1/n^2)$. On a cycle, the stationary distribution is uniform because it is a regular graph So, after putting the numbers in, we get

$$\tau(\varepsilon) \leq O(n^2 \log n \log(1/\varepsilon)).$$

Note that this bound is almost tight (up to the $\log n$ factor. We proved at early stages of this course, that a random walk on a line at time $t$ only goes $\sqrt{t}$ steps away from the origin with high probability. This implies that in order for the walk to mix on a cycle, we need to run it for at least $\Omega(n^2)$. The above theorem says that this already enough. If you run the walk for $\Omega(n^2)$ steps you are equally likely to be anywhere on the cycle.

If $G$ is a complete graph, $\lambda_2 = 1$, so $\tau(\varepsilon) = O(\log n \log(1/\varepsilon))$. So you mix really really quickly. (And actually, if $G$ is complete you just need 1 step in order to "mix").

If $G$ is an expander graph, $\lambda_2$ is at least a constant, so $\tau(\varepsilon) = O(\log n \log(1/\varepsilon))$ again—after $O(\log n)$ steps, you're equally likely to be anywhere in the network. Note that this is a very special property of expander graphs. Because as we discussed a random 3 regular graph is an expander. So, in such a sparse graph only after $O(\log n)$ steps you are equally likely to be anywhere in the graph. This the reason that expander graphs are typically used to pseudorandom generators.

### 14.1.7 Connections to Cheeger's inequality

You can combine the theorem in the previous subsection with Cheeger's inequality. Recall that Cheeger's inequality states

$$\frac{\lambda_2}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2}$$

This can be reformulated as

$$\frac{1}{\lambda_2} \leq \frac{2}{\Phi^2(G)}$$

So, for all connected graphs $G$,

$$\tau(\varepsilon) \in O\left(\frac{1}{\Phi^2(G)} \log n\right)$$

Examples:

$\Phi(G)$ of a cycle is $\approx 2/n$, and we saw already that $\tau(\varepsilon)$ is $O(n^2 \log n)$.

On a $\sqrt{n} \times \sqrt{n}$ grid, the cut with minimum conductance cuts $\sqrt{n}$ edges, so $\Phi(G)$ is $O(1/\sqrt{n})$, so $\tau(\varepsilon)$ is $O(n \log n)$.

## 14.2 Local Graph Clustering

Let us conclude the discussion of random walks by describing an application in clustering.

In the local graph clustering problem, we are given a node $i$ of a graph $G$, and we want to find a cluster around $i$. The problem is that the graph is so large that we cannot afford even to run linear time algorithms. For example suppose we want to find a community that a particular facebook user belongs to and all we are allowed to do is to look at the friends or friends of friends, etc of that user.

In general starting from the given vertex $i$ one may want to run a graph search algorithm like DFS to find a community around $i$. But it turns out that this is not the right algorithm. $i$ may have a single edge to someone outside of her community and by following the DFS tree we may go along a completely wrong path.

The idea is to run a random walk started from $i$ and look at the distribution of the walk. We choose a community around $i$ by selecting the vertices with high probability in the random walk distribution. Let us briefly give the intuition behind this idea. Consider a set $S$ of vertices it turns out that

$$\mathbb{P}\left[X_1 \notin S \mid X_0 \sim S\right] = \phi(S)$$

where $X_0 \sim S$ means that we start the walk from a vertex of $S$ chosen proportional by degree. It follows from the above inequality that

$$\mathbb{P}\left[X_1 \in S \mid X_0 \sim S\right] = 1 - \phi(S).$$

In other words, if $S$ has a very small conductance, then the walk will stay in $S$ after one step with a very high probability. Using a bit of work one can show the following lemma:

**Lemma 14.8.** $\mathbb{P}\left[X_t \in S \mid X_0 \sim S\right] \geq (1 - \phi(S)/2)^t$

This shows that if $\phi(S)$ is small then for $t = O(1/\phi(S))$ steps the walk will stay inside $S$ with probability at least $9/10$.

This is good, because it shows that members of sets with low conductance (which is what we're looking for) are more likely to be visited.

Idea: Look at *distribution* of random walk by time $t = 1/phi$, where $\phi$ is the target conductance, and choose the $k$ most probable nodes to get a cluster of size $k$. The above algorithm works indeed and it gives sets of small conductance; however, it is not fast enough because we need to compute the full distribution of the walk in order to find out the more probable vertices. There are ways to go about this, for example rounding down vertices of small probability.

But, there is a more straightforward algorithm that works perfectly and is very simple to implement. We do not discuss the details here and we refer interested students to [And+16].

**Algorithm 1** Local Graph Clustering

**procedure** FINDCLUSTER($\tilde{P}, i, \phi$) ▷ Use lazy walk matrix to find a cluster around $i$
  $x_0 \leftarrow i$
  $S \leftarrow \{i\}$
  **for** $t \in \{1, ..., \frac{\log n}{\phi}\}$ **do** ▷ $\phi$ is the target conductance
    Do 1 step of "lazy" walk—that is, given $x_t$ use $\tilde{P}$ to choose $x_{t+1}$
    Choose $Z \sim [0, \mathbb{P}[X_1 \in S_t \mid X_0 = x_{t+1}]]$, where by $X_1$ we mean the first step of the walk started at $x_{t+1}$.
    $S_{t+1} \leftarrow \{j : \mathbb{P}[X_1 \in S_t \mid X_0 = j] \geq z\}$
  **end for**
  **return** $\operatorname{argmin}_t \phi(S_t)$
**end procedure**

## 14.3 Linear Programming

Now, we switch to the last part of this course. We discuss techniques in convex optimization related to computer science.

Let us start by an example to see what type of programs are not solvable. In quadratic programming we have constraints on product of variables. For example, consider the following program:

$$\begin{aligned} \max \quad & \sum_{i \sim j} x_i(1 - x_j) \\ \text{s.t.}, \quad & x_i(1 - x_i) = 0 \qquad \forall\, i. \end{aligned} \tag{14.1}$$

Observe that the constraint $x_i(1 - x_i) = 0$ is satisfied only for $x_i = \{0, 1\}$. So, the set of feasible solutions to the above program is a discrete set. For example, suppose we sum over all neighbor pairs $i \sim j$ in a graph $G$. Then, this program exactly characterizes the optimum solution of max cut. The set of feasible solutions are all cuts, and the objective function is the size of the cut. So, this means that it is NP-hard to solve quadratic programs in general. Instead we study linear functionals that do not have product of variables.

The easiest system of linear equations are linear qualities:

$$\begin{aligned} x_1 + 2x_2 &= 1 \\ x_3 + x_1 &= 3 \end{aligned}$$

We know how to solve a system of lienar equations; This is just high school math—encode it as $Ax = b$ and solve for $x$ by finding $A^{-1}$.

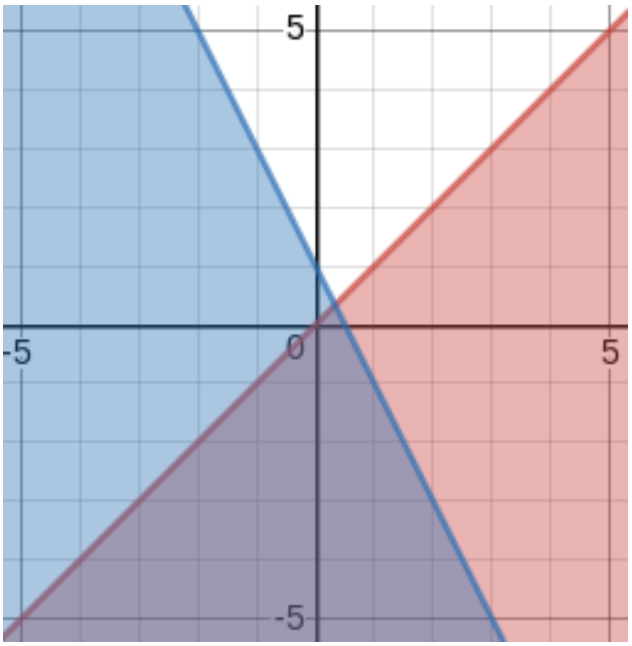


Figure 14.1: graphed using www.desmos.com

Systems to be solved by linear programming are similar to systems of linear equations, but inequalities are allowed, and there's an objective function.

$$\begin{aligned} \max \quad & 2x_1 + 3x_2 \\ \text{s.t.}, \quad & x_1 - x_2 \geq 0 \\ & 2x_1 + x_2 \leq 1. \end{aligned}$$

It turns out that the set of points that satisfy each inequality corresponds to a half-plane in two dimensions and half spaces in higher dimension. So, the set points that satisfy all of the inequalities just correspond to intersection of half spaces. In the above example, this leads to the following graph, where the first inequality is drawn in red and the second in blue. The set of feasible solutions is the bottom triangle which is colored both red and blue. It is easy to check that the optimal solution is at the intersection of the boundary lines.

All linear programming constraints can be written in the following generic way:

$$\begin{aligned} \min \quad & \langle c, x \rangle \\ \text{s.t.}, \quad & Ax \geq b \end{aligned}$$

Here, the notation $Ax \geq b$ just means that $\forall\, i,\ \langle a_i, x \rangle \geq b_i$, where $a_i$ is the $i$-th row of $A$. It is not hard to see that any type of constraints can be translated to the above generic form.

If instead we want $\langle a_i, x \rangle \leq b_i$, we can encode it as $\langle -a_i, x \rangle \geq -b_i$. If we want $\langle a_i, x \rangle = b_i$ we can encode it as $\langle a_i, x \rangle \geq b_i$ and $\langle -a_i, x \rangle \geq -b_i$. To maximize $\langle c, x \rangle$, we can minimize $\langle -c, x \rangle$.

The shape of the constrained space (formed by the intersection of finitely many half-spaces) is called a polytope. In general a linear program may be infeasible if there is no point in the intersection of half spaces. Its solution be unbounded. For example in the example of Figure 14.1, if the objective function is $\min x_2$, then the optimum solution is $-\infty$. If the optimum solution exists and it is bounded, it is always a vertex of the polytope. Note that it may also happen that all points on a face of the polytope are optimal; this happens when the vector $c$ is orthogonal to a face of the polytope.

There are various ways to solve these systems, which fall roughly into two major groups—the simplex method (idea–walk along vertices following edges) and interior point methods (start in the interior and move toward optimum iteratively).

In general, the best running time proved to solve a linear programming problem finds the optimal solution by solving $O(\sqrt{n})$ many systems of linear equations in the worst. That is, if we have an oracle that solves systems of linear equations, the algorithm would make $O(\sqrt{n})$ time calls to that oracle. Here $n$ denotes the number of variables. See the course website for pointers to relevant papers.

There exist very fast packages to solve linear programming problems, for example in CVX in Matlab or IBM's CPLEX. The last homework will involve using such a package.

## References

[And+16] R. Andersen, S. O. Gharan, Y. Peres, and L. Trevisan. "Almost Optimal Local Graph Clustering Using Evolving Sets". In: *J. ACM* 63.2 (2016), 15:1–15:31 (cit. on p. 14-6).