Spectral Graph Theory Lecture 19

PSRGs via Random Walks on Graphs

Daniel A. Spielman November 5, 2018

19.1 Overview

There has been a lot of work on the design of Pseudo-Random Number Generators (PSRGs) that can be proved to work for particular applications. In this class, we will see one of the foundational results in this area, namely Impagliazzo and Zuckerman's [IZ89] use of random walks on expanders to run the same algorithm many times. We are going to perform a very crude analysis that is easy to present in class. Rest assured that much tighter analyses are possible and much better PSRGs have been constructed since.

19.2 Why Study PSRGs?

Pseudo-random number generators take a seed which is presumably random (or which has a lot of randomness in it), and then generate a long string of random bits that are supposed to act random. We should first discuss why we would actually want such a thing. I can think of two reasons.

1. Random bits are scarce. This might be surprising. After all, if you look at the last few bits of the time that I last hit a key, it is pretty random. Similarly, the low-order bits of the temperature of the processor in my computer seem pretty random. While these bits are pretty random, there are not too many of them.

   Many randomized algorithms need *a lot* of random bits. Sources such as these just do not produce random bits with a frequency sufficient for many applications.

2. If you want to re-run an algorithm, say to de-bug it, it is very convenient to be able to use the same set of random bits by re-running the PSRG with the same seed. If you use truly random bits, you can't do this.

You may also wonder how good the standard pseudo-random number generators are. The first answer is that the default ones, such as `rand` in C, are usually terrible. There are many applications, such as those in my thesis, for which these generators produce behavior that is very different from what one would expect from truly random bits (yes, this is personal). On the other hand, one can use cryptographic functions to create bits that will act random for most purposes, unless one can break the underlying cryptography [HILL99]. But, the resulting generators are usually much slower than the fastest pseudo-random generators. Fundamentally, it comes down to a time-versus-quality tradeoff. The longer you are willing to wait, the better the pseudo-random bits you can get.

19.3 Expander Graphs

In today's lecture we will require an infinite family of d-regular 1/10-expander graphs. We require that d be a constant, that the graphs have 2^r vertices for all sufficiently large r, and that we can construct the neighbors of a vertex in time polynomial in r. That is, we need the graphs to have a simple explicit description. One can construct expanders families of this form using the techniques from last lecture. For today's purposes, the best expanders are the Ramanujan graphs produced by Margulis [Mar88] and Lubotzky, Phillips and Sarnak [LPS88]. Ramanujan graphs of degree $d = 400$ are 1/10-expanders. See also the work of Alon, Bruck, Naor, Naor and Roth [ABN$^+$92] for even more explicit constructions.

While the explicit Ramanujan graphs only exist in certain sizes, none of which do have exactly 2^r vertices, some of them have just a little more that 2^r vertices. It is possible to trim these to make them work, say by ignoring all steps in which the vertex does not correspond to r bits.

19.4 Today's Application : repeating an experiment

Imagine you are given a black box that takes r bits as input and then outputs either 0 or 1. Moreover, let's assume that the black box is very consistent: we know that it returns the same answer at least 99% of the time. If it almost always returns 0, we will call it a 0-box and if it almost always returns 1, we will call it a 1-box. Our job is to determine whether a given box is a 0 or 1 box. We assume that r is big, so we don't have time to test the box on all 2^r settings of r bits. Instead, we could pick r bits at random, and check what the box returns. If it says"1", then it is probably a 1-box. But, what if we want more than 99% confidence? We could check the box on many choices of r random bits, and report the majority value returned by the box.[1]. But, this seems to require a new set of random bits for each run. In this lecture, we will prove that 9 new bits per run suffice. Note that the result would be interesting for any constant other than 9.

Since we will not make any assumptions about the black box, we will use truly random bits the first time we test it. But, we will show that we only need 9 new random bits for each successive test. In particular, we will show that if we use our PSRG to generate bits for $t + 1$ test, then the probability that majority answer is wrong decreases exponentially in t.

You are probably wondering why we would want to do such a thing. The reason is to increase the accuracy of randomized algorithms. There are many randomized algorithms that provide weak guarantees, such as being correct 99% or 51% of the time. To obtain accurate answers from such algorithms, we run them many times with fresh random bits. You can view such an algorithm has having two inputs: the problem to be solved and its random bits. The black box is the behavior of the algorithm when the problem to be solved is fixed, so it is just working on the random bits.

[1] Check for yourself that running it twice doesn't help

19.5 The Random Walk Generator

Let r be the number of bits that our black box takes as input. So, the space of random bits is $\{0,1\}^r$. Let $X \subset \{0,1\}^r$ be the settings of the random bits on which the box gives the minority answer, and let Y be the settings on which it gives the majority answer.

Our pseudo-random generator will use a random walk on a 1/10-expander graph whose vertex set is $\{0,1\}^r$. Recall that we can use $d = 400$. For the first input we feed to the black box, we will require r truly random bits. We treat these bits as a vertex of our graph. For each successive test, we choose a random neighbor of the present vertex, and feed the corresponding bits to the box. That is, we choose a random i between 1 and 400, and move to the ith neighbor of the present vertex. Note that we only need $\log_2 400 \approx 9$ random bits to choose the next vertex. So, we will only need 9 new bits to generate each input we feed to the box after the first.

19.6 Formalizing the problem

Assume that we are going to test the box $t+1$ times. Our pseudo-random generator will begin at a truly random vertex v, and then take t random steps. Recall that we defined X to be the set of vertices on which the box outputs the minority answer, and we assume that $|X| \leq 2^r/100$. If we report the majority of the outcomes of the $t+1$ outputs of the box, we will return the correct answer as long as the random walk is inside X less than half the time. To analyze this, let v_0 be the initial random vertex, and let $v_1, \ldots, v_t$ be the vertices produced by the t steps of the random walk. Let $T = \{0, \ldots, t\}$ be the time steps, and let $S = \{i : v_i \in X\}$. We will prove

$$\Pr\left[|S| > t/2\right] \leq \left(\frac{2}{\sqrt{5}}\right)^{t+1}.$$

To begin our analysis, recall that the initial distribution of our random walk is $\boldsymbol{p}_0 = \mathbf{1}/n$. Let $\boldsymbol{\chi}_X$ and $\boldsymbol{\chi}_Y$ be the characteristic vectors of X and Y, respectively, and let $\boldsymbol{D}_X = \mathtt{diag}(\boldsymbol{\chi}_X)$ and $\boldsymbol{D}_Y = \mathtt{diag}(\boldsymbol{\chi}_Y)$. Let

$$\boldsymbol{W} = \frac{1}{d}\boldsymbol{M} \tag{19.1}$$

be the transition matrix of the ordinary random walk on G. We are not using the lazy random walk: it would be silly to use the lazy random walk for this problem, as there is no benfit to re-running the experiment with the same random bits as before. Let $\omega_1, \ldots, \omega_n$ be the eigenvalues of $\boldsymbol{W}$. As the graph is a 1/10-expander, $|\omega_i| \leq 1/10$ for all $i \geq 2$.

Let's see how we can use these matrices to understand the probabilities under consideration. For a probability vector $\boldsymbol{p}$ on vertices, the probability that a vertex chosen according to $\boldsymbol{p}$ is in X may be expressed

$$\boldsymbol{\chi}_X^T \boldsymbol{p} = \mathbf{1}^T \boldsymbol{D}_X \boldsymbol{p}.$$

The second form will be more useful, as

$$\boldsymbol{D}_X \boldsymbol{p}$$

is the vector obtained by zeroing out the events in which the vertices are not in X. If we then want to take a step in the graph G, we multiply by $\boldsymbol{W}$. That is, the probability that the walk starts at vertex in X, and then goes to a vertex i is $\boldsymbol{q}(i)$ where

$$\boldsymbol{q} = \boldsymbol{W}\boldsymbol{D}_X\boldsymbol{p}_0.$$

Continuing this way, we see that the probability that the walk is in X at precisely the times $i \in R$ is

$$\mathbf{1}^T\boldsymbol{D}_{Z_t}\boldsymbol{W}\boldsymbol{D}_{Z_{t-1}}\boldsymbol{W}\cdots\boldsymbol{D}_{Z_1}\boldsymbol{W}\boldsymbol{D}_{Z_0}\boldsymbol{p}_0,$$

where

$$Z_i = \begin{cases} X & \text{if } i \in R \\ Y & \text{otherwise.} \end{cases}$$

We will prove that this probability is at most $(1/5)^{|R|}$. It will then follow that

$$\begin{aligned}
\Pr\left[|S| > t/2\right] &\leq \sum_{|R|>t/2} \Pr\left[\text{the walk is in } X \text{ at precisely the times in } R\right] \\
&\leq \sum_{|R|>t/2} \left(\frac{1}{5}\right)^{|R|} \\
&\leq 2^{t+1}\left(\frac{1}{5}\right)^{(t+1)/2} \\
&= \left(\frac{2}{\sqrt{5}}\right)^{t+1}.
\end{aligned}$$

19.7 Matrix Norms

Recall that the operator norm of a matrix M (also called the 2-norm) is defined by

$$\|\boldsymbol{M}\| = \max_{\boldsymbol{v}} \frac{\|\boldsymbol{M}\boldsymbol{v}\|}{\|\boldsymbol{v}\|}.$$

The matrix norm measures how much a vector can increase in size when it is multiplied by $\boldsymbol{M}$. When $\boldsymbol{M}$ is symmetric, the 2-norm is just the largest absolute value of an eigenvalue of $\boldsymbol{M}$ (prove this for yourself). It is also immediate that

$$\|\boldsymbol{M}_1\boldsymbol{M}_2\| \leq \|\boldsymbol{M}_1\|\,\|\boldsymbol{M}_2\|.$$

You should also verify this yourself. As $\boldsymbol{D}_X$, $\boldsymbol{D}_Y$ and $\boldsymbol{W}$ are symmetric, they each have norm 1.

Warning 19.7.1. *While the largest eigenvalue of a walk matrix is* 1*, the norm of an asymmetric walk matrix can be larger than* 1*. For instance, consider the walk matrix of the path on* 3 *vertices. Verify that it has norm* $\sqrt{2}$*.*

Our analysis rests upon the following bound on the norm of $\boldsymbol{D}_X W$.

Lemma 19.7.2.

$$\|\boldsymbol{D}_X \boldsymbol{W}\| \leq 1/5.$$

Let's see why this implies the theorem. For any set R, let Z_i be as defined above. As $\boldsymbol{p}_0 = \boldsymbol{W}\boldsymbol{p}_0$, we have

$$\mathbf{1}^T \boldsymbol{D}_{Z_t} \boldsymbol{W} \boldsymbol{D}_{Z_{t-1}} \boldsymbol{W} \cdots \boldsymbol{D}_{Z_1} \boldsymbol{W} \boldsymbol{D}_{Z_0} \boldsymbol{p}_0 = \mathbf{1}^T \left(\boldsymbol{D}_{Z_t} \boldsymbol{W}\right) \left(\boldsymbol{D}_{Z_{t-1}} \boldsymbol{W}\right) \cdots \left(\boldsymbol{D}_{Z_0} \boldsymbol{W}\right) \boldsymbol{p}_0.$$

Now,

$$\left\|\boldsymbol{D}_{Z_{t-1}} \boldsymbol{W}\right\| \leq \begin{cases} 1/5 & \text{for } i \in R, \text{ and} \\ 1 & \text{for } i \notin R. \end{cases}$$

So,

$$\left\|\left(\boldsymbol{D}_{Z_t} \boldsymbol{W}\right) \left(\boldsymbol{D}_{Z_{t-1}} \boldsymbol{W}\right) \cdots \left(\boldsymbol{D}_{Z_0} \boldsymbol{W}\right)\right\| \leq (1/5)^{|R|}.$$

As $\|\boldsymbol{p}_0\| = 1/\sqrt{n}$ and $\|\mathbf{1}\| = \sqrt{n}$, we may conclude

$$\begin{aligned} \mathbf{1}^T \left(\boldsymbol{D}_{Z_t} \boldsymbol{W}\right) \left(\boldsymbol{D}_{Z_{t-1}} \boldsymbol{W}\right) \cdots \left(\boldsymbol{D}_{Z_0} \boldsymbol{W}\right) \boldsymbol{p}_0 &\leq \left\|\mathbf{1}^T\right\| \left\|\left(\boldsymbol{D}_{Z_t} \boldsymbol{W}\right) \left(\boldsymbol{D}_{Z_{t-1}} \boldsymbol{W}\right) \cdots \left(\boldsymbol{D}_{Z_0} \boldsymbol{W}\right) \boldsymbol{p}_0\right\| \\ &\leq \left\|\mathbf{1}^T\right\| (1/5)^{|R|} \|\boldsymbol{p}_0\| \\ &= (1/5)^{|R|}. \end{aligned}$$

19.8 The norm of $\boldsymbol{D}_X \boldsymbol{W}$

Proof of Lemma 19.7.2. Let $\boldsymbol{x}$ be any non-zero vector, and write

$$\boldsymbol{x} = c\mathbf{1} + \boldsymbol{y},$$

where $\mathbf{1}^T \boldsymbol{y} = 0$. We will show that $\|\boldsymbol{D}_X \boldsymbol{W} \boldsymbol{x}\| \leq \|\boldsymbol{x}\|/5$.

We know that the constant vectors are eigenvectors of $\boldsymbol{W}$. So, $\boldsymbol{W}\mathbf{1} = \mathbf{1}$ and

$$\boldsymbol{D}_X \boldsymbol{W} \mathbf{1} = \boldsymbol{\chi}_X.$$

This implies

$$\|\boldsymbol{D}_X \boldsymbol{W} c\mathbf{1}\| = c\,\|\boldsymbol{\chi}_X\| = c\sqrt{|X|} \leq c\sqrt{n}/10.$$

We will now show that $\|\boldsymbol{W}\boldsymbol{y}\| \leq \|\boldsymbol{y}\|/10$. The easiest way to see this is to consider the matrix

$$\boldsymbol{W} - \boldsymbol{J}/n,$$

where we recall that $\boldsymbol{J}$ is the all-1 matrix. This matrix is symmetric and all of its eigenvalues have absolute value at most $1/10$. So, it has norm at most $1/10$. Moreover, $(\boldsymbol{W} - \boldsymbol{J}/n)\boldsymbol{y} = \boldsymbol{W}\boldsymbol{y}$, which implies $\|\boldsymbol{W}\boldsymbol{y}\| \leq \|\boldsymbol{y}\|/10$. Another way to prove this is to expand $\boldsymbol{y}$ in the eigenbasis of $\boldsymbol{W}$, as in the proof of Lemma 2.1.3.

Finally, as $\mathbf{1}$ is orthogonal to $\boldsymbol{y}$,

$$\|\boldsymbol{x}\| = \sqrt{c^2 n + \|\boldsymbol{y}\|^2}.$$

So,

$$\|\boldsymbol{D}_X \boldsymbol{W} \boldsymbol{x}\| \leq \|\boldsymbol{D}_X \boldsymbol{W} c\mathbf{1}\| + \|\boldsymbol{D}_X \boldsymbol{W} \boldsymbol{y}\| \leq c\sqrt{n}/10 + \|\boldsymbol{y}\|/10 \leq \|\boldsymbol{x}\|/10 + \|\boldsymbol{x}\|/10 \leq \|\boldsymbol{x}\|/5.$$

□

19.9 Conclusion

We finished lecture by observing that this is a very strange proof. When considering probabilities, it seems that it would be much more natural to sum them. But, here we consider 2-norms of probability vectors.

References

[ABN+92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, March 1992.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *30th annual IEEE Symposium on Foundations of Computer Science*, pages 248–253, 1989.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[Mar88] G. A. Margulis. Explicit group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24(1):39–46, July 1988.