Chapter 17

Random and Pseudo-Random Walks on Graphs

Random walks on graphs have turned out to be a powerful tool in the design of algorithms and other applications. In particular, *expander graphs*, which are graphs on which random walks have particularly good properties, are extremely useful in complexity and other areas of computer science. In this chapter we will study random walks on general graphs, leading to a the randomized logspace algorithm for undirected connectivity alluded to in Chapter 7. We will then show the definition and constructions of expander graphs, and their application for randomness-efficient error reduction of probabilistic algorithms. Finally we will use the ideas behind that construction to show a *deterministic* logspace algorithm for undirected connectivity.

The idea of a random walk is simple: let G be a graph (in this chapter we'll restrict ourselves to *undirected* graphs) and let v be a vertex in G. A *T-step random walk from v in G* is the sequence of dependent random variables $X_0, \ldots, X_T$ defined as follows: $X_0 = v$ with probability one, and for $1 \leq i \leq t$, X_T is chosen at random from $\Gamma(X_T)$, where for any vertex u, $\Gamma(u)$ denotes the set of neighbors of u in the graph G. That is, a random walk involves starting at a vertex and then at each step going to a random neighbor of this vertex. A probabilistic process of this form, where there is a fixed distribution specifying the dependence of X_T on X_{T-1}, is also called a *Markov chain*.

17.1 Undirected connectivity in randomized logspace

Recall the language PATH of the triplets $\langle G, s, t\rangle$ where G is a (directed) graph and s and t are vertices in G with a path from s to t. In Chapter 3 we showed that PATH is **NL**-complete. Consider the problem UPATH where G is restricted to be *undirected* (or equivalently, we place the condition that there is an edge from i to j in G iff there's an edge from j to i: the adjacency matrix is symmetric). We will show that UPATH can be computed by a log space probabilistic TM.

Theorem 17.1 ([AKL+79])
UPATH $\in$ **RL**.

The algorithm for the proof of Theorem 17.1 will be simple. Given an input G, s, t to UPATH, we'll first use an implicitly computable in logspace reduction to ensure the graph is regular and of degree 4 (see Claim 17.1.1) and then take a random walk of length $T = 100n^3 \log n$ from s on the graph G. We accept if at the end of the walk we reach the vertex t. Otherwise, reject. The algorithm can be implemented in $O(\log n)$ space since it only requires space to store the current and next vertex in the walk, and a counter. Clearly, it will never accept if t is not connected to s. We will show in the next section that if t is connected to s, then the algorithm will accept with probability $\frac{1}{\Omega(n)}$ (this can of course be amplified using the standard error reduction techniques of Section 7.1.1).

Reducing to the regular constant-degree case. As mentioned above, we start by reducing to the case that every vertex in G has degree 4 (i.e., G is 4-regular) and that every vertex has a self-loop (i.e., an edge to itself). This claim not strictly necessary for this algorithm but will somewhat simplify the analysis and will be useful for us later on. We note that here and throughout this chapter all graphs may have parallel edges (i.e., more than one edge between the same pair of vertices i, j).

Claim 17.1.1
There's an implicitly computable in logspace function f that maps any triple $\langle G, s, t\rangle$ to a triple $\langle G', s', t'\rangle$ such that:

1. *G' is a 4-regular graph with a self loop at each vertex.*

2. *s is connected to t in G iff s' is connected to t' in G'.*

We sketch the proof, leaving verifying the details as an exercise. For every vertex i in G, the graph G' will have n vertices arranged in a cycle. For every two neighbors i, j in G, we'll connect in an edge the j^{th} vertex from the cycle corresponding to i, and the i^{th} vertex from the cycle corresponding to j. Thus, every vertex in G' will have degree either two (if it's only connected to its neighbors on the cycle) or three (if it also has a neighbor in a different cycle). We'll add to each vertex either one or two self loops to make the degree four. It can be seen that determining the value of the entry corresponding to a pair of vertices in the adjacency matrix of G' can be computed in log space given read-only access to the adjacency matrix of G.

17.2 Random walk on graphs

In this section we'll study random walks on (undirected regular) graphs. As a corollary we will obtain the proof of correctness for the above algorithm for UPATH. We will see that we can use elementary linear algebra to relate parameters of the graph's adjacency matrix to the behavior of the random walk on that graph. The following definitions and notations will be widely used in this and later sections of this chapter:

Distributions as vectors, adjacency matrix of graphs. Let G be a d-regular n-vertex graph. Let $\mathbf{p}$ be some probability distribution over the vertices of G. We can think of $\mathbf{p}$ as a (column) vector in $\mathbf{R}^n$ where $\mathbf{p}_i$ is the probability that vertex i is obtained by the distribution $\mathbf{p}$. Note that the one-norm of $\mathbf{p}$, defined as $|\mathbf{p}|_1 = \sum_{i=1}^{n} |\mathbf{p}_i|$ is equal to 1 (in this case the absolute value is redundant since $\mathbf{p}_i$ is always between 0 and 1). Now let $\mathbf{q}$ represent the distribution of the following random variable: choose a vertex i in G according to $\mathbf{p}$, then take a random neighbor of i in G. We can compute $\mathbf{q}$ as a function of $\mathbf{p}$: the probability $\mathbf{q}_j$ that j is chosen is equal to the sum over all the neighbors i of j of the probability that i is chosen in $\mathbf{p}$ times $1/d$ (the probability that if i is chosen, the walk moves to $\mathbf{q}$). We see that in fact $\mathbf{q} = A\mathbf{p}$, where $A = A(G)$ which is the *normalized adjacency matrix* of G. That is, for every two vertices i, j, $A_{i,j}$ is equal to the number of edges between i and j divided by d. Note that A is a symmetric matrix, where each entry is between 0 and 1, and the sum of entries in each row and column is exactly one (such a matrix is called a symmetric *stochastic* matrix). Let $\{\mathbf{e}^i\}_{i=1}^{n}$ be the *standard basis* of $\mathbf{R}^n$ (i.e. $\mathbf{e}^i$ has 1 in the i^{th} coordinate and zero everywhere else). Then, $A^T\mathbf{e}^s$ represents the distribution X_T of taking a T-step random walk from the vertex s. This already suggests that considering the adjacency matrix A could be very useful in analyzing random walks.[1]

DEFINITION 17.2 (THE PARAMETER $\lambda(G)$.) Denote by $\mathbf{1}$ the vector $(1/n, 1/n, \ldots, 1/n)$ corresponding to the uniform distribution. Denote by $\mathbf{1}^\perp$ the set of vectors perpendicular to $\mathbf{1}$. That is $\mathbf{v} \in \mathbf{1}^\perp$ if $\langle v, \mathbf{1}\rangle = 0$ or equivalently, $\sum_{i=0}^{n-1} \mathbf{v}_i = 0$. For a vector $\mathbf{v} \in \mathbf{R}^n$, we denote by $\|\mathbf{v}\|_2$ the *two norm* of $\mathbf{v}$ (i.e., $\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^n \mathbf{v}_i^2} = \sqrt{\langle \mathbf{v}, \mathbf{v}\rangle}$). We call $\mathbf{v}$ a *unit vector* if $\|\mathbf{v}\|_2 = 1$. We define $\lambda(A)$ (which we'll also denote as $\lambda(G)$) to be the maximum value of $\|A\mathbf{v}\|_2$ over all unit vectors $\mathbf{v} \in \mathbf{1}^\perp$.

REMARK 17.3 The value $\lambda(G)$ is often called the *second largest eigenvalue* of G. The reason is that since A is a symmetric matrix, we can find an orthogonal basis of eigenvectors $\mathbf{v}^1, \ldots, \mathbf{v}^n$ with corresponding eigenvalues which we can sort to ensure $|\lambda_1| \geq |\lambda_2| \ldots \geq |\lambda_n|$. Note that $A\mathbf{1} = \mathbf{1}$. Indeed, for every i, $(A\mathbf{1})_i$ is equal to the inner product of the i^{th} row of A and the vector $\mathbf{1}$ which (since the sum of entries in the row is one) is equal to $1/n$. Thus, $\mathbf{1}$ is an *eigenvector* of A with the corresponding eigenvalue equal to 1. It's not hard to show that in a symmetric stochastic matrix for all i, $\lambda_i \leq 1$ (see Exercise 1) and hence we can assume $\lambda_1 = 1$ and $\mathbf{v}^1 = \mathbf{1}$. It's also not hard to show that because $\mathbf{1}^\perp = \mathrm{Span}\{\mathbf{v}^2, \ldots, \mathbf{v}^n\}$ the value λ above will be maximized by (the normalized version of) $\mathbf{v}^2$, hence $\lambda(G) = |\lambda_2|$. The quantity $1 - \lambda(G)$ is often called the *spectral gap* of the graph. We note that some texts use *un-normalized* adjacency matrices, in which case $\lambda(G)$ will be a number between 0 and d.

One reason that $\lambda(G)$ is an important parameter is the following lemma:

LEMMA 17.4
For every regular n vertex graph $G = (V, E)$ let $\mathbf{p}$ be any probability distribution over V, then

$$\|A^T\mathbf{p} - \mathbf{1}\|_2 \leq \lambda^T$$

PROOF: By the definition of $\lambda(G)$, $\|A\mathbf{v}\|_2 \leq \lambda\|\mathbf{v}\|_2$ for every $\mathbf{v} \perp \mathbf{1}$. Note that if $\mathbf{v} \perp \mathbf{1}$ then $A\mathbf{v} \perp \mathbf{1}$ since $\langle \mathbf{1}, A\mathbf{v}\rangle = \langle A^\dagger\mathbf{1}, \mathbf{v}\rangle = \langle \mathbf{1}, \mathbf{v}\rangle = 0$ (since $A = A^\dagger$ and $A\mathbf{1} = \mathbf{1}$). Thus A maps the space $\mathbf{1}^\perp$ to itself and since it shrinks any member of this space by at least λ, we get that $\lambda(A^T) \leq \lambda(A)^T$. (In fact, using the eigenvalue definition of λ it can be shown that $\lambda(A^T) = \lambda(A)$.)

[1]Note that the observations above extend to the case that G is non-regular and even a directed graph.

Let $\mathbf{p}$ be some vector. We can break $\mathbf{p}$ into its components in the spaces parallel and orthogonal to $\mathbf{1}$ and express it as $\mathbf{p} = \alpha\mathbf{1} + \mathbf{p}'$ where $\mathbf{p}' \perp \mathbf{1}$ and α is some number. If $\mathbf{p}$ is a probability distribution then $\alpha = 1$ since the sum of coordinates in $\mathbf{p}'$ is zero. Therefore,

$$A^T\mathbf{p} = A^T(\mathbf{1} + \mathbf{p}') = \mathbf{1} + A^T\mathbf{p}'$$

We have that $\|\mathbf{p}\|_2^2 = \|\mathbf{1}\|_2^2 + \|\mathbf{p}'\|_2^2$ and in particular $\|\mathbf{p}'\|_2 \le \|\mathbf{p}\|_2$. Since $\mathbf{p}$ is a probability vector we have that $\|\mathbf{p}\|_2 \le |\mathbf{p}|_1 \le 1$. Hence $\|\mathbf{p}'\|_2 \le 1$ and

$$\|A^T\mathbf{p} - \mathbf{1}\|_2 = \|A^T\mathbf{p}'\|_2 \le \lambda^T$$

□

We'll now show that every connected graph has a noticeable spectral gap.

Lemma 17.5
For every d-regular connected G with self-loops at each vertex, $\lambda(G) \le 1 - \frac{1}{8dn^3}$.

Proof:[of Lemma 17.5] Let $\mathbf{u} \perp \mathbf{1}$ be a unit vector and let $\mathbf{v} = A\mathbf{u}$. We'll show that $1 - \|\mathbf{v}\|_2^2 \ge \frac{1}{d4n^3}$ which implies $\|\mathbf{v}\|_2^2 \le 1 - \frac{1}{d4n^3}$ and hence $\|\mathbf{v}\|_2 \le 1 - \frac{1}{d8n^3}$.

Since $\|\mathbf{u}\|_2 = 1$ we have that $1 - \|\mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 - \|\mathbf{v}\|_2^2$. We claim that this is equal to $\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2$ where i, j range from 1 to n. Indeed,

$$\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 = \sum_{i,j} A_{i,j}\mathbf{u}_i^2 - 2\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j + \sum_{i,j} A_{i,j}\mathbf{v}_j^2 = \|\mathbf{u}\|_2^2 - 2\|\mathbf{v}\|_2^2 + \|\mathbf{v}\|_2^2$$

where the latter equality is because the sum of each row and column in A equals one, and because $\|\mathbf{v}\|_2^2 = \langle \mathbf{v}, \mathbf{v}\rangle = \langle A\mathbf{u}, \mathbf{v}\rangle = \sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j$.

Thus we need to show $\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \ge \frac{1}{d4n^3}$. Note that this sum is indeed always non-negative and that it is enough to show that for *some* i, j, $A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \ge \frac{1}{d4n^3}$. Firstly, because we have all the self-loops, and hence $A_{i,i} \ge 1/d$ for all i, we can assume $|\mathbf{u}_i - \mathbf{v}_i| < \frac{1}{2n^{1.5}}$ for every $1 \le i \le n$.

Now sort the coordinates of $\mathbf{u}$ from the largest to the smallest, and hence assume that $\mathbf{u}_1 \ge \mathbf{u}_2 \ge \cdots \mathbf{u}_n$. Since $\sum_i \mathbf{u}_i = 0$ it must holds that $\mathbf{u}_1 \ge 0 \ge \mathbf{u}_n$. In fact, since $\mathbf{u}$ is a unit vector, it holds that either $\mathbf{u}_1 \ge 1/\sqrt{n}$ or $\mathbf{u}_n \le 1/\sqrt{n}$ and so $\mathbf{u}_1 - \mathbf{u}_n \ge 1/\sqrt{n}$. By looking at the $n-1$ differences between consecutive coordinates $\mathbf{u}_i - \mathbf{u}_{i+1}$, one of them must be at least $1/n^{1.5}$ and so there must be an i_0 such that if we let $S = \{1, \dots, i_0\}$ and

$\overline{S} = [n] \setminus S_i$, then for every $i \in S$ and $j \in \overline{S}$, $\mathbf{u}_i - \mathbf{u}_j \geq 1/n^{1.5}$. Since G is connected there exists an edge (i,j) between S and $\overline{S}$. Since $|\mathbf{v}_j - \mathbf{u}_j| \leq \frac{1}{2n^{1.5}}$ we get that for this choice of i,j, $|\mathbf{u}_i - \mathbf{v}_j| \geq |\mathbf{u}_i - \mathbf{u}_j| - \frac{1}{2n^{1.5}} \geq \frac{1}{2n^{1.5}}$. Thus $A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \geq \frac{1}{d}\frac{1}{4n^3}$. □

Together, Lemmas 17.4 and 17.5 imply that our algorithm for UPATH will accept with probability $1/\Omega(n)$ if s is connected to t in the graph. The reason is that by the Cauchy-Schwartz inequality $\frac{|\mathbf{v}|_1}{\sqrt{n}} \leq \|\mathbf{v}\|_2 \leq |\mathbf{v}|_1$ for every vector $\mathbf{v}$. Hence, we get that for every probability vector $\mathbf{p}$ and $T \geq 10dn^3 \log n$, $\|A^T\mathbf{p} - \mathbf{1}\|_2 < \frac{1}{2n^{1.5}}$, implying $|A^T\mathbf{p} - \mathbf{1}|_1 < \frac{1}{2n}$, leading us to the following corollary:

Corollary 17.6
Let G be a d-regular n-vertex graph with all vertices having a self-loop. Let s be a vertex in G. Let $T > 10dn^3 \log n$ and let X_T denote the distribution of the vertex of the T^{th} step in a random walk from s. Then, for every j connected to s, $\Pr[X_T = j] > \frac{1}{2n}$.

17.3 Expander graphs and their use for error reduction.

Expander graphs has played a crucial role in numerous applications in computer science, including routing networks, error correcting codes, hardness of approximation and the PCP theorem, derandomization, and more. In this chapter we will see their definition, constructions, and two applications, including a derandomized algorithm for the problem UPATH of undirected connectivity.

Expanders can be defined in several equivalent ways. One way is that these are graphs where every set of vertices has a very large boundary. That is, for every subset in the graph, the number of neighbors outside the set will be (up to a constant factor) roughly equal to the number of vertices inside the set. (Of course this condition cannot hold if the set is too big and already contains too large a fraction of the vertices in the graph.) For example, the n by n grid (where a vertex is a pair (i,j) and if it's not on the graph's boundary it is connected to the four neighbors $(i \pm 1, j \pm 1)$) is *not* an expander, as any k by k square in this graph will only have a boundary of size $O(\sqrt{k})$. Another way to define expanders is as graphs where the random walks rapidly converges to the uniform distribution. That is, unlike in the general case that (in regular graphs) the random walk may take a polynomial number of steps to converge to the uniform distribution, in an n-vertex regular expander this will only take $O(\log n)$ steps.

Given the previous section, it is not surprising that we can define expander also in an *algebraic* way, based on the parameter $\lambda(G)$ of Definition 17.7. That is, we will say that G is an expander if $\lambda(G)$ is bounded away from 1 and satisfies $\lambda(G) \leq 1 - \epsilon$. By Lemma 17.4, this does indeed imply that the random walk on G will converge to the uniform distribution (in the sense that regardless of the starting distribution, every vertex will be obtained with probability between $\frac{1}{2n}$ and $\frac{3}{2n}$) within $O(\log n)$ steps. We will also see later (Theorem 17.14) the relation between the parameter $\lambda(G)$ and the combinatorial definition of set expansion mentioned above. Formally, we define expander graphs as follows:

Definition 17.7 (Expander graphs) If G is an n-vertex d-regular G with $\lambda(G) \leq \lambda$ for some number $\lambda < 1$ then we say that G is an (n, d, λ)-graph. For every d and $\lambda < 1$ a d, λ-*expander graph family* is a sequence $\{G_n\}$ of graphs where each G_n is an (n', d, λ) and $n' = p(n)$ for some polynomial $p(\cdot)$. We'll sometimes drop the qualifier d, λ and simply call such a family an *expander graph family*, referring to a particular graph in the sequence as an expander graph.

We say that the family is *explicit* if there's a polynomial-time algorithm that on input 1^n outputs the adjacency matrix of G_n. We say that the family is *strongly explicit* if there's a polynomial-time algorithm the on inputs $\langle n, v, i \rangle$ where $1 \leq v \leq n'$ and $1 \leq i \leq d$ outputs the i^{th} neighbor of v. (Note that the algorithm runs in time polynomial in the its input length which is polylogarithmic in n.)

As we'll see below it is not hard to show that expander families exist using the probabilistic method. In fact, there are also several explicit and strongly explicit constructions of expander graphs. The smallest λ can be for a d-regular n-vertex graph is $\Omega(\frac{1}{\sqrt{d}}$ and constructions meeting this bound (specifically the bound is $(1 - o(1))\frac{2\sqrt{d-1}}{d}$ where by $o(1)$ we mean a function that tends to 0 as the number of vertices grows; graphs meeting this bound are called *Ramanujan graphs*). However, for most applications in Computer Science, any family with constant d and λ will suffice (see also Remark 17.8 below). Some of these constructions are very simple and efficient, but their analysis is highly non-trivial and uses relatively deep mathematics.[2] We

[2] An example for such an expander is the following 3-regular graph: the vertices are the numbers 1 to $p - 1$ for some prime p, and each number x is connected to $x + 1$,$x - 1$ and $x^{-1} \pmod{p}$.

will show in Section 17.4 a strongly explicit construction of expanders with elementary analysis. This construction will also introduce a tool that will be useful to derandomize the algorithm for UPATH.

Remark 17.8 One reason that the particular constants of an expander family are not extremely crucial is that we can improve the constant λ (make it arbitrarily smaller) at the expense of increasing the degree: this follows from the fact, observed above in the proof of Lemma 17.4, that $\lambda(G^T) = \lambda(G)^T$, where G^T denotes the graph obtained by taking the adjacency matrix to the T^{th} power, or equivalently, having an edge for every length-T path in G. Thus, we can transform an (n, d, λ) graph into an (n, d^T, λ^T)-graph for every $T \geq 1$. Later we will see a different transformation called the *zig-zag product* to decrease the degree at the expense of increasing λ somewhat (and also increasing the number of vertices).

17.3.1 Using expanders to reduce error in probabilistic algorithms.

Before constructing expanders, let us see one application for them in the area of probabilistic algorithms. Recall that in Section 7.1.1 we saw that we can reduce the error of a probabilistic algorithm from, say, 1/3 to $2^{-\Omega(k)}$ by executing it k times independently and taking the majority value. If the algorithm utilized m random coins, this procedure will use $m \cdot k$ random coins, and intuitively it seems hard to think of a way to save on randomness. Nonetheless, we will show that using expanders we can obtain such error reduction using only $m + O(k)$ random coins. The idea is simple: take an expander graph G from a very explicit family that is an $(N = 2^m, d, 1/10)$-graph for some constant d.[3] Choose a vertex v_1 at random, and take a length $k-1$ long random walk on G to obtain vertices $v_2, \ldots, v_k$ (note that choosing a random neighbor of a vertex requires $O(\log d) = O(1)$ random bits). Invoke the algorithm k times using $v_1, \ldots, v_k$ as random coins (we identify the set $[N]$ of vertices with the set $\{0,1\}^m$ of possible random coins for the algorithm) and output the majority answer.

[3]In our definition of an expander family, we did not require that there is an N-vertex graph in the family for every N, however typical constructions are quite dense (for example, there could be an expander for every N that is of the form c^k for some constant c) and so we if we need an expander of size N, we can use an expander of size at most cN which in this application will cost at most constantly many random bits. For simplicity of description, we will ignore this issue below and assume we have such a 2^m-vertex graph in the family. Note that we can improve the constant λ of the family to be smaller than 1/10 as indicated above in Remark 17.8.

The analysis is also not too difficult, but to make it even simpler, we will analyze only the case of algorithms with one-sided error. For example, consider an **RP** algorithm that will never output "accept" if the input is not in the language, and for inputs in the language will output "accept" with probability 2/3 (the case of a **coRP** algorithm is analogous). For such an algorithm the procedure will output "accept" if the algorithm accepts even on a single set of coins v_i. If the input is not in the language, the procedure will never accept. If the input is in the language, then let $B \subseteq [N]$ denote the "bad" set of coins on which the algorithms rejects. We know that $|B| \leq \frac{N}{3}$. To show the procedure will output reject with at most $2^{-\Omega(k)}$ probability, we prove the following lemma:

Lemma 17.9
Let G be an (N, d, λ) graph, and let $B \subseteq [N]$ be a set with $|B| \leq \beta N$. Let $X_1, \ldots, X_k$ be random variables denoting a $k-1$-long random walk from X_1, where X_1 is chosen uniformly in $[N]$. Then,

$$\underset{(*)}{\Pr[\forall_{1 \leq i \leq k} X_i \in B]} \leq ((1-\lambda)\sqrt{\beta} + \lambda)^k$$

Note that if λ and β are both constants smaller than 1 then so is the expression $(1-\lambda)\sqrt{\beta} + \lambda$.
Proof: For $1 \leq i \leq k$, let B_i be the event that $X_i \in B$. Note that the probability $(*)$ we're trying to bound is $\Pr[B_1]\Pr[B_2|B_1]\cdots\Pr[B_k|B_1,\ldots,B_{k-1}]$. Let $\mathbf{p}^i \in \mathbf{R}^N$ be the vector representing the distribution of X_i, conditioned on the events $B_1, \ldots, B_i$. Denote by $\hat{B}$ the following linear transformation from $\mathbf{R}^n$ to $\mathbf{R}^n$: for every $\mathbf{u} \in \mathbf{R}^N$, and $j \in [N]$, $\hat{B}\mathbf{u}_j = \mathbf{u}_j$ if $j \in B$ and $\hat{B}\mathbf{u}_j = 0$ otherwise. It's not hard to verify that $\mathbf{p}^1 = \frac{1}{\Pr[B_1]}\hat{B}\mathbf{1}$ (recall that $\mathbf{1} = (1/N, \ldots, 1/N)$ is the vector representing the uniform distribution over $[N]$). Similarly, $\mathbf{p}^2 = \frac{1}{\Pr[B_2|B_1]}\frac{1}{\Pr[B_1]}\hat{B}A\hat{B}\mathbf{1}$ where $A = A(G)$ is the adjacency matrix of G. Since any probability vector $\mathbf{p}$ has $|\mathbf{p}|_1 = 1$, we get that

$$(*) = |(\hat{B}A)^{k-1}\hat{B}\mathbf{1}|_1$$

We'll bound this norm by showing that

$$\|(\hat{B}A)^{k-1}\hat{B}\mathbf{1}\|_2 \leq \frac{(\sqrt{(1-\lambda)\beta}+\lambda)^k}{\sqrt{N}} \qquad (1)$$

This will be sufficient since for every vector $\mathbf{v} \in \mathbf{R}^N$, $|\mathbf{v}|_1 \leq \sqrt{N}\|\mathbf{v}\|_2$.

We'll prove Equation 1 via the following quite useful definition and lemma:

DEFINITION 17.10 (MATRIX NORM) If A is an m by n matrix, then $\|A\|$ is the maximum number α such that $\|A\mathbf{v}\|_2 \leq \alpha\|\mathbf{v}\|_2$ for every $\mathbf{v} \in \mathbf{R}^n$.

Note that if A is a normalized adjacency matrix then $\|A\| = 1$ (as $A\mathbf{1} = \mathbf{1}$ and $\|A\mathbf{v}\|_2 \leq \|\mathbf{v}\|_2$ for every $\mathbf{v}$). Also note that the matrix norm satisfies that for every two n by n matrices A, B, $\|A+B\| \leq \|A\|+\|B\|$ and $\|AB\| \leq \|A\|\|B\|$.

LEMMA 17.11 ([**?**])
Let A be a normalized adjacency matrix of an (n, d, λ)-graph G. Let J be the adjacency matrix of the n-clique with self loops (i.e., $J_{i,j} = 1/n$ for every i, j). Then

$$A = (1-\lambda)J + \lambda C \tag{2}$$

where $\|C\| \leq 1$.

Note that for every probability vector $\mathbf{p} = J\mathbf{p}$ is the uniform distribution, and so this lemma tells us that in some sense, we can think of a step on a (n, d, λ)-graph as going to the uniform distribution with probability $1-\lambda$, and to a different distribution with probability λ. This is of course not completely accurate, as a step on a d-regular graph will only go the one of the d neighbors of the current vertex, but we'll see that for the purposes of our analysis, the condition (2) will be just as good.[4]
PROOF: Indeed, simply define $C = \frac{1}{\lambda}(A-(1-\lambda)J)$. We need to prove $\|C\mathbf{v}\|_2 \leq \|\mathbf{v}\|_2$ for very $\mathbf{v}$. Indeed, let $\mathbf{v} = \mathbf{u}+\mathbf{w}$ where u is $\alpha\mathbf{1}$ for some α and $\mathbf{w} \perp \mathbf{1}$, and $\|\mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 + \|\mathbf{w}\|_2^2$. Since $A\mathbf{1} = \mathbf{1}$ and $J\mathbf{1} = \mathbf{1}$ we get that $C\mathbf{u} = \frac{1}{\lambda}(\mathbf{u} - (1-\lambda)\mathbf{u}) = \mathbf{u}$. Now, let $\mathbf{w}' = A\mathbf{w}$. We have that $\|\mathbf{w}'\|_2 \leq \lambda\|\mathbf{w}\|_2$ and, as we saw in the proof of Lemma 17.4, $\mathbf{w}' \perp \mathbf{1}$. Furthermore, since the sum of the coordinates of $\mathbf{w}$ is zero, we have that $J\mathbf{w} = \mathbf{0}$. We get that $C\mathbf{w} = \frac{1}{\lambda}\mathbf{w}'$. Since $\mathbf{w}' \perp \mathbf{u}$ we get that $\|C\mathbf{w}\|_2^2 = \|\mathbf{u} + \frac{1}{\lambda}\mathbf{w}'\|_2^2 = \|\mathbf{u}\|_2^2 + \|\frac{1}{\lambda}\mathbf{w}'\|_2^2 \leq \|\mathbf{u}\|_2^2 + \|\mathbf{w}\|_2^2 = \|\mathbf{w}\|_2^2$. $\square$

We'll prove (1) by showing $\|\hat{B}A\| \leq (1-\lambda)\sqrt{B} + \lambda$. However, since $\hat{B}A = \hat{B}\big((1-\lambda)J + \lambda C\big)$, we get that $\|\hat{B}A\| \leq (1-\lambda)\|\hat{B}J\| + \lambda\|\hat{B}C\|$. Since J's output is always a vector of the form $\alpha\mathbf{1}$, we get that $\|\hat{B}J\| \leq \sqrt{\beta}$. Also we know that $\hat{B}$ only zeros some parts of its input and hence $\|\hat{B}\| \leq 1$ implying $\|\hat{B}C\| \leq 1$. Thus, $\|\hat{B}A\| \leq (1-\lambda)\sqrt{\beta} + \lambda \leq \lambda + \sqrt{\beta}$. Since $B\mathbf{1}$ has the value $1/n$ in $|B|$ places, we get $\|B\mathbf{1}\|_2 = \frac{\sqrt{\beta}}{\sqrt{n}}$ establishing (1).

[4]Algebraically, the reason (2) is not equivalent to going to the uniform distribution in each step with probability $1-\lambda$ is that C is not necessarily a stochastic matrix, and may have negative entries.

□

As mentioned above, there is a stronger version of this lemma showing that with exponentially high probability, the number of vertices of the random walk in B will be close to the expected βk. This stronger lemma can be used to show the correctness of the error reduction procedure for algorithms with two-sided error.

17.3.2 Combinatorial expansion and existence of expanders.

We describe now a combinatorial criteria that is roughly equivalent to Definition 17.7. One advantage of this criteria is that it makes it easy to prove that a non-explicit expander family exists using the probabilistic method. It is also quite useful in several applications.

Definition 17.12 (Combinatorial expansion) An n-vertex d-regular graph $G = (V, E)$ is called an (n, d, c)-*combinatorial expander* if for every subset $S \subseteq V$ with $|S| \leq n/2$, $|E(S, \overline{S})| \geq c|S|$, where for subsets S, T of V, $E(S, T)$ denotes the set of edges (s, t) with $s \in S$ and $t \in T$.

Note that in this case the bigger c is the better the expander. We'll loosely use the term expander for any (n, d, c)-combinatorial expander with c a positive constant. Using the probabilistic method, one can prove the following theorem: (Exercise 2 asks you to prove a slightly weaker version)

Theorem 17.13 (Existence of expanders)
Let $\epsilon > 0$ be some constant. Then there exists $d = d(\epsilon)$ and $N in \mathbf{N}$ such that for every $n > N$ there exists an $(n, d, (1-\epsilon)d)$-combinatorial expander.

The following theorem related combinatorial expansion with our previous Definition 17.7

Theorem 17.14 (Combinatorial and algebraic expansion)

1. *If G is an (n, d, λ)-graph then it is an $(n, d, d(1-\lambda)/2)$-combinatorial expander.*
2. *If G is an (n, d, c)-combinatorial expander then it is an $(n, d, 1 - \frac{c^2}{2d})$-graph.*

Proof: We leave the first part as Exercise 3. Second part: to be completed.
□

17.4 The zig-zag construction of expander graphs.

We will now show a strongly explicit construction of an expander graph family. This construction will be based on the *zig-zag* graph product. The zig-zag product will enable us to reduce the degree of a large expander graph without considerably harming its expansion, by taking its product with a smaller graph. Since we can always find a constant-sized expander graph using a brute-force search, this notion naturally suggest an approach for a recursive construction of larger graphs based on smaller graphs, and this is indeed what we will do. The zig-zag product is also used in the deterministic logspace algorithm for UPATH of the next section.

Graphs as rotation maps. One can view an n-vertex degree-d graph G as a function $\hat{G}$ from $[n] \times [d]$ to $[n]$ that given a pair $\langle v, i \rangle$ outputs u where the i^{th} neighbor of v in G. In fact, it will be convenient for us to have $\hat{G}$ output an additional value $j \in [d]$ where j is the index of v as a neighbor of u. Given this definition of $\hat{G}$ it is clear that we can invert it by applying it again, and so it is a permutation on $[n] \times [d]$. We call $\hat{G}$ the *rotation map* of G. To give a very explicit construction for G is equivalent to give a polynomial (in $\log n$) time algorithm for computing $\hat{G}$. Note that given a vertex v, we can go to a random neighbor of v by choosing i at random from $[d]$ and computing $\hat{G}(v, i)$. For starters, one may think of the case that $\hat{G}(u, i) = (v, i)$ (i.e., v is the i^{th} neighbor of u iff u is the i^{th} neighbor of v). In this case we can think of $\hat{G}$ as operating only on the vertex. However, we will need the more general notion of a rotation map later on.

A graph product is an operation that takes two graphs G_1, G_2 and outputs a graph G. Typically we're interested in the relation between properties of the graphs G_1, G_2 to the properties of the resulting graph G. In this chapter we will mainly be interested in three parameters: the number of vertices (denoted n), the degree (denoted d), and the 2^{nd} largest eigenvalue of the normalized adjacency matrix (denoted λ). We will now describe three graph products:

Matrix Product We've already seen this product in the form of graph squaring. For two n vertex graphs G, G' with adjacency matrices A, A', the graph $G'G$ will be the graph described by the adjacency matrix $A'A$. That is a graph with an edge (u, v) for every length 2-path from u to v where the first step in the path is taken on en edge of G and the second is on an edge of G'. Typically we will be interested in the case $G = G'$, in which case we call this product *graph squaring.* Since both A_1 and A_2 preserve orthogonality to the uniform distribution we get that if G was an (n, d, λ) graph and G' an (n, d', λ') graph then $G'G$ is an $(n, dd', \lambda\lambda')$ graph.

Tensor product Given two graphs G, G' we denote their *tensor product* by $G \otimes G'$. If G was an (n, d, λ)-graph and G' an (n', d', λ')-graph then $G \otimes G'$ will be an $(n \cdot n', d \cdot d', \max\{\lambda, \lambda'\})$-graph (see Lemma 17.16). The simplest way to describe the tensor product is in the language of rotation maps: we have that $G \hat{\otimes} G'$ is a permutation over $([n] \times [n']) \times ([d] \times [d'])$ and the $G \hat{\otimes} G'(\langle u, v\rangle, \langle i, j\rangle) = \langle u', v'\rangle, \langle i', j'\rangle$ where $(u', i') = \hat{G}(u, i)$ and $(v', j') = \hat{G}'(v, j)$. That is, the vertex set of $G \otimes G'$ is pairs of vertices, one from G and the other from G', and taking a the step $\langle i, j\rangle$ on $G \otimes G'$ from the vertex $\langle u, v\rangle$ is akin to taking two independent steps: move to the pair $\langle u', v'\rangle$ where u' is the i^{th} neighbor of u in G and v' is the i^{th} neighbor of v in G'.

In terms of adjacency matrices, the tensor product is also quite easy to describe. If A is the $n \times n$ adjacency matrix of G and A' is the $n' \times n'$ adjacency matrix of G', then the adjacency matrix of $G \otimes G'$, denoted as $A \otimes A'$, will be an $nn' \times nn'$ matrix that in the $\langle i, i'\rangle^{th}$ row and the $\langle j, j'\rangle$ column has the value $A_{i,j} \cdot A'_{i',j'}$.

The tensor product can also be described in the language of graphs as having a cluster of n' vertices in $G \otimes G'$ for every vertex of G. Now if, u and v are two neighboring vertices in G, we will put a bipartite version of G' between the cluster corresponding to u and the cluster corresponding to v in G. That is, if (i, j) is an edge in G' then we'll put an edge between the i^{th} vertex in the cluster corresponding to u and the j^{th} vertex in the cluster corresponding to v.

Zig-zag product In both the above products, the degree of the resulting graph is larger than the degree of the input graphs. The following product helps us actually reduce the degree of one of the graphs. Given two graphs G, H satisfying the following condition: G is an $(n, D, 1-\epsilon)$-graph, and H is an $(D, d, 1-\epsilon')$-graph (i.e., the size of H is equal to the degree of G), we denote the *zig-zag product* of G, H by $G \circled{z} H$. It will be an $(n \cdot D, d^2, 1 - \epsilon\epsilon'^2)$-graph (see Lemma 17.17). Again, it will be simpler to describe the product in terms of the rotation maps: on input $(\langle u, v\rangle, \langle i', j'\rangle)$ the map will do the following:

Input:	G vertex $u \in [n]$	H vertex $v \in [D]$	H 1st index $i \in [d]$	H 2nd index $j \in [d]$
Step 1: $(v', i') = \hat{H}(v, i)$		$\downarrow \hat{H}$ v'	$\downarrow \hat{H}$ i'	
Step 2: $(u', v'') = \hat{G}(u, v')$	$\downarrow \hat{G}$ u'	$\downarrow \hat{G}$ v''		
Step 3: $(v'', j') = \hat{H}(v', j)$		$\downarrow \hat{H}$ v''		$\downarrow \hat{H}$ j'
Output:	u'	v''	i'	j'

The zig-zag product can also be described directly in terms of the resulting graph. The vertex set is the same set as in $G \otimes H$. That is, it will contain a D-sized cluster of vertices for every vertex in G. However, the degree will be significantly smaller. In the tensor product a vertex in the cluster is connected to D distinct clusters, and in each of them to d vertices in the cluster, leading to degree $D \cdot d$. In the zig-zag product, each edge corresponds to choosing one of the d edges to move within the same cluster, using the label of that vertex to deterministically move to a corresponding vertex in another cluster, and then choosing one of the d edges to make another move within that second cluster, leading to degree d^2. The"zig-zag" shape of this path lends the product its name.

Remark 17.15 (Intuition behind the zig-zag product.) We will understand the zig zag product better once we see its analysis. However, the high level intuition is the following. We think of the input graph G_1 as a good expander whose only drawback is that it has a too high degree D. This means that a k step random walk on G_1 will require $O(k \log D)$ random bits. However, as we saw in Section 17.3.1, sometimes we can use fewer random bits if we use an expander. So a natural idea is to generate the edge labels for the walk by taking a walk on an expander with D vertices and degree $d \ll D$ (e.g., $d = 3$). We will lose something in the mixing properties of the expander, but we can ensure that the drop in the degree size is more dramatic than the loss in expansion, so we can regain what we lost in expansion by taking a square of the graph (or a higher power), while still keeping the degree small. If we try to carry this intuition through, it will lead as to a somewhat simpler product than the zig-zag product, which is called the *replacement product* (denoted Ⓡ): under the same parameters, the graph G_1 Ⓡ G_2 has $n \cdot D$ vertices but degree only d, and its rotation map is that on input $\langle u, v\rangle, i$ it applies $(v', i') = \hat{G}_2(v, i)$ and $(u', v') = \hat{G}_1(u, v)$. The replacement product is tightly connected to the zig-zag product and can be used in its place in all the construction below, although with slightly more complicated analysis.[5]

We now prove that while, unlike the matrix product, the zig-zag and tensor products do not improve the expansion, they do not harm it too significantly:

Lemma 17.16 (Tensor product preserves expansion)
Let $\lambda = \lambda(G)$ *and* $\lambda' = \lambda(G')$ *then* $\lambda(G \otimes G') \leq \max\{\lambda, \lambda'\}$.

Proof: Given some basic facts about tensor products and eigenvalues this is immediate since if $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A (where A is the adjacency matrix of G) and $\lambda'_1, \ldots, \lambda'_{n'}$ are the eigenvalues of A (where A' is the adjacency matrix of G'), then the eigenvalues of $A \otimes A'$ are all numbers of the form $\lambda_i \cdot \lambda'_j$, and hence the largest ones are of the form $1 \cdot \lambda(G')$ or $\lambda(G) \cdot 1$ (see also Exercise 4).

However, we will show a different proof, that does not assume these facts, for a somewhat weaker bound that $\lambda(G \otimes G') \leq \lambda + \lambda'$. The weaker bound suffices for our applications, and the proof will provide some intuitions helpful in analyzing the zig-zag product. The adjacency matrix of $G \otimes G'$ is $A \otimes A'$, where the $\langle i, j\rangle, \langle i', j'\rangle$ position in $A \otimes A'$ is $A_{i,i'}A'_{j,j'}$. By Lemma 2, $A = (1-\lambda_1)J + \lambda_1 C$ and $A = (1-\lambda_1)J' + \lambda_1 C'$ where J is the $n \times n$ with all entries $1/n$ and J' is the $n' \times n'$ matrix with all entries $1/n'$, and C, C' are matrices with norm at most one of the corresponding dimensions. it's not hard to verify that the tensor operation is bilinear (i.e., for every three matrices A, B, C, $(A+B) \otimes C = A \otimes C + B \otimes C$), and so we get that

$$A \otimes A' = \big((1-\lambda)J + \lambda C\big) \otimes \big((1-\lambda')J' + \lambda' C'\big)$$

However, this matrix is of the form $(1-\lambda)(1-\lambda')J_1 \otimes J_2 + \big(1-(1-\lambda)(1-\lambda')\big)D$ where D is some matrix with $\|D\| \leq 1$ (one can verify that $\|A \otimes B\| \leq \|A\|\|B\|$ for example by showing $A \otimes B = (A \otimes I)(I' \otimes B)$ where I, I' are the identity matrices of the appropriate dimension). Since $J \otimes J'$ is the $nn' \times nn'$ matrix with all entries $1/(nn')$, we get that any vector $\mathbf{v} \perp \mathbf{1}$ will be zeroed out by that function and hence we will have $\|A \otimes A'\|_2 \leq (\lambda + \lambda')\mathbf{v}$ where the latter term comes from the fact that $\big(1-(1-\lambda)(1-\lambda') = \lambda + \lambda' - \lambda\lambda' \leq \lambda + \lambda'$.
$\square$

[5]Because $G_1 \text{ⓩ} G_2$ is a subgraph of $(G_1 \text{Ⓡ} G_2)^3$ the fact that $G_1 \text{ⓩ} G_2$ is an expander implies that $G_1 \text{Ⓡ} G_2$ is also an expander, albeit with worse parameters.

Lemma 17.17 (Zig-zag product preserves expansion)
Let G be an $(n, D, 1-\epsilon)$-graph, and H an $(D, d, 1-\epsilon')$-graph. Then $G \textcircled{z} H$ is an $(n \cdot D, d^2, 1-\epsilon\epsilon'^2)$-graph

Proof: Recall that the rotation map of G is a permutation on $[n] \times [D]$. This permutation can be thought of as a linear function $\hat{A}$ over $\mathbf{R}^{n \cdot D}$, where $\hat{A}\mathbf{e}^{u,v} = \mathbf{e}^{\hat{G}_1(u,v)}$ (we denote by $\mathbf{e}^{u,v}$ the $n \cdot D$-dimensional vector that is 1 in the $(u,v)^{th}$ position and zero everywhere else). It is important not to confuse the permutation $\hat{A}$, which is an $nD \times nD$ matrix with each row and column containing a single 1 and the rest are 0, with the normalized adjacency matrix A of G, which is an $n \times n$ matrix with rows and columns containing D entries with $1/D$ and the rest are 0.

Let B denote the adjacency matrix of H. Now, a vertex in $G \textcircled{z} H$ can be thought of as a pair $\langle u, v \rangle$ with $u \in n$ and $v \in [D]$. If we take a random step in $G \textcircled{z} H$ then the first operation in ignores u and takes a random step on v. In terms of adjacency this corresponds to the matrix $I \otimes B$, where I is the $n \times n$ identity matrix. This is also what happens in the last operation, while in the middle we apply the permutation $\hat{G}$ on $\langle u, v \rangle$. We get that the adjacency matrix of $G \textcircled{z} H$ is in fact

$$(I \otimes B)\hat{A}(I \otimes B) \tag{3}$$

(The reader can also verify this directly that (3) agrees with the adjacency matrix of $G \textcircled{z} H$ on the standard basis.)

We now apply Lemma 17.11 to express this matrix as

$$\left(I \otimes \left(\epsilon' J + (1-\epsilon')C\right)\right) \hat{A} \left(I \otimes \epsilon' J + (1-\epsilon')C\right) \tag{4}$$

where J is the $D \times D$ matrix with all entries $1/D$ and C has norm at most 1. We'll open up the parenthesis and get that the matrix is $\epsilon'^2(I \otimes J)\hat{A}(I \otimes J) + (1-\epsilon'^2)E$ where E is a matrix with norm at most one.

The key observation is that

$$(I \otimes J)\hat{A}(I \otimes J) = A \otimes J \tag{5}$$

where A is the actual $n \times n$ adjacency matrix of G. Both the left hand side and right hand side of this equation are stochastic matrices which are adjacency matrices of some graph. We will show that it is the same graph.

Let G_R be the graph corresponding to the matrix $A \otimes J$ in right hand side of (5). It is not hard to describe: take the graph G and transform each vertex into a full clique (with self-loops), and for every edge (u, u') in G

place the complete bipartite clique between the clique corresponding to u and the clique corresponding to v.[6]

Let G_R be the graph corresponding to the matrix $(I \otimes J)\hat{A}(I \otimes J)$ in the left hand side of (5). It is the matrix product of three graphs $H_1 H_2 H_3$. Both the graphs H_1 and H_3 correspond to the matrix $I \otimes J$, and contain n isolated cliques (with self loops) each of size D. The graphH_2 corresponding to $\hat{A}$ has in-degree and out-degree 1, and has an edge between two vertices $\langle u, v\rangle$ and $\langle u', v'\rangle$ if $\hat{G}_1(u,v) = (u', v')$. An edge in the graph G_R corresponds to a length-3 path where the first step is taken in H_3, the second in H_2, and the third in H_3.

To show that $G_R = G_L$ we need to show that any G_R edge is in G_L and vice versa (we ignore the possibility of parallel edges/edge weights here but this can be verified as well). However, in both graphs we can get from $\langle u, v\rangle$ to $\langle u', v'\rangle$ if and only if the edge (u, u') is in G.

Once we have (5) we're basically done since by (4)

$$\lambda(G \circledz H) \leq \epsilon'^2 \lambda(G \otimes J) + (1 - \epsilon'^2)\lambda(E)$$

but $\lambda(E) \leq 1$ since E has norm at most one, and $\lambda(G_1 \otimes J) \leq 1 - \epsilon$ by Lemma 17.16. □

Given the products above, it is not hard to see the construction of a strongly explicit expander family:

- Let H be a $(D = d^8, d, 0.1)$-graph, which we can find using brute force search. (We choose d to be a large enough constant that such a graph exists)

- Let G_1 be a $(4D, d^4, 1/2)$-graph, which we can find using brute force search.

- For $k > 1$, let $G_k = ((G_{k-1} \otimes G_{k-1}) \circledz H)^4$.

The following claim can be shown by induction, using the properties of the three graph products (in particular Lemmas 17.16 and 17.17):

Claim 17.17.1
For every $k \geq 1$, G_k is a $(2^{2^k} D^k, d^4, 1/2)$-graph.

[6]This discussion assumes that G is unweighted and with all the self-loops, but it generalizes to weighted graphs as well. We note that one can also verify the identity of the two matrices by considering their evaluations on all the standard basis vectors $\mathbf{e}^{u,v} \in \mathbf{R}^{n \cdot D}$.

Let $T(k)$ denote the time to compute the rotation map on G_k, then $T(k) \leq 10T(k-1) + O(1)$. We see that $T(k) = 2^{O(k)}$ which is indeed poly-logarithmic in the size of G_k. Thus we have the following theorem:

Theorem 17.18 (Explicit construction for expanders)
There exists a strongly-explicit λ, d-expander family for some constants d and $\lambda < 1$.

Remark 17.19 There is a variant of the construction supplying a *denser* family of graphs (i.e., an n-vertex graph for every n that is a power of c for some constant $c > 1$). As mentioned above, there are also constructions of expanders (typically based on number theory) that are more efficient in terms of computation time and relation between degree and the parameter λ than the zig-zag. However, the proofs for these constructions are more complicated and require deeper mathematical tools. The zig-zag product has found applications beyond the above construction of expander graphs. One such application is the deterministic logspace algorithm for undirected connectivity described in the next section. Another application is a construction of combinatorial expanders with greater expansion that what is implied by the parameter λ. (Note that even for $\lambda = 0$, Theorem 17.1.1 implies combinatorial expansion only $d/2$ while it can be shown that a random d-regular graph has combinatorial expansion close to d.)

17.5 Deterministic logspace algorithm for undirected connectivity.

The zig-zag theorem above has a surprising consequence: a *deterministic* algorithm to determine whether two vertices are connected in a graph using only logarithmic space.

Theorem 17.20 (Reingold's theorem [**?**])
$\mathsf{UPATH} \in \mathbf{L}$.

Proof:When proving Theorem 17.20, we can assume without loss of generality that the graph has *constant degree* since there's an implicitly computable in logspace function mapping to this case (see Claim 17.1.1. One underlying intuition behind the algorithm is that checking that s is connected to t in a constant-degree *expander* graph G is indeed is easy. Since the random walk from any vertex of G converges in $O(\log n)$ steps by Lemma 17.4, we get that for every two vertices s, t there's a path of length $O(\log n)$ from s to t (in other words, the graph has diameter $O(\log n)$). Thus, there's a value

$\ell = O(\log n)$ such that to check that s is connected to t we can enumerate over all ℓ-length walks from s (which takes storing ℓ indices of neighbors, each of them of constant size) and check if we hit t. Thus, the idea of the algorithm will be to transform the graph G (in an implicitly computable in logspace way) to a graph G' such that every connected component will become an expander, but two vertices that were not connected will stay unconnected. This transformation will be reminiscent of the expander construction of the previous section. Once we have this transformation, we will check connectivity of s and t in the graph G'.

Let G be a graph of n vertices and constant degree with self loops on all vertices. We will assume the graph is of degree d^8 for some constant d that is sufficiently large so that there exists a $(d^8, d, 0.1)$-graph H. Assume for starters that G is connected and consider the following sequence of graphs:

- Let H be a $(D = d^8, d, 0.1)$-graph, which we can find using brute force search.
- Let $G_0 = G$.
- For $k \geq 1$, we define $G_k = (G_{k-1} ⓩ H)^4$.

Our main claim is the following:

Claim 17.20.1
For every k, G_k is a $(nD^k, d^8, \max\{0.8, 1 - \frac{2^k}{8dn^3}\})$-graph.

Proof: Since we assume that G is connected, then G_0 is indeed a $(n, d^10, 1-\frac{1}{8dn^3})$-graph by Lemma 17.5. Using the properties of zig-zag and squaring products (in particular Lemma 17.17) it's not hard to verify inductively that the condition for G_k holds. □

Thus, after $k = O(\log n)$ iterations, G_k will be an expander graph with $\lambda(G_k) \leq 0.8$. As mentioned above, on G_k it's easy to verify connectivity since it's a constant degree graph with $O(\log n)$ diameter. Note that both the squaring and zig-zag products do not create edges between unconnected vertices. Thus, the transformation works separately on each of G's connected components. Since each of them had initially at most n vertices and hence the parameter λ at most $1 - \frac{1}{8dn^3}$, we see that they'll all become expanders within $k = O(\log n)$ steps, and that s is connected to t in $G = G_0$ iff there's an $O(\log n)$-length path from s to t in G_k.[7]

[7]Recall that when performing the zig-zag product $G_{k-1} ⓩ H$ we convert each vertex of G_{k-1} to a cluster of vertices in G_k. We will identify the original vertex with the first vertex of the cluster, and hence think of s and t as valid vertices in all the graphs G_k for $k \geq 0$.

The space required to enumerate over ℓ length walks from some vertex s in G_k is $O(\ell)$ bits to store ℓ indices and the space to compute the rotation map of G_k. To finish the proof, we'll show that we can compute this map in $O(k+\log n)$ space. This map's input length is $O(k+\log n)$ and hence we can assume it is placed on a read/write tape, and will compute the rotation map "in-place" changing the input to the output. Let s_k be the additional space (beyond the input) required to compute the rotation map of G_k. Note that $s_0 = O(\log n)$. We'll give a recursive algorithm to compute G_k, that will satisfy the equation $s_k = s_{k-1} + O(1)$. In fact, the algorithm will be a pretty straightforward implementation of the definitions of the zig-zag and matrix products.

The input to $\hat{G}_k$ is a vertex in $(G_{k-1} ⓩ H)$ and five labels of edges in the graph. If we can compute the rotation map of $G_{k-1} ⓩ H$ in $s_{k-1}+O(1)$ space then we can do so for $\hat{G}_k$, since we can simply make five consecutive calls to this procedure, each time reusing the space.[8] Now, to compute the rotation map $\hat{G'}_k$ where $G'_k = (G_{k-1} ⓩ H)$ we again simply follow the definition of the zig-zag product. Given an input of the form u, v, i, j (which we think of as read/write variables), we can transform it to $\hat{G'}_k(\langle u, v\rangle, \langle i, j\rangle)$ by transforming v, i into $\hat{H}(v, i)$ (can be done in constant space), then transforming u, v into $\hat{G}_{k-1}(u, v)$ using a recursive call (at cost of space s_{k-1}, note that u, v are conveniently located consecutively at the beginning of the input tape), and then transforming v, j into $\hat{H}(v, j)$.

□

Chapter notes and history

Problems

§1 Let A be a symmetric stochastic matrix: $A = A^\dagger$ and every row and column of A has non-negative entries summing up to one. Prove that $\|A\| \leq 1$. (Hint: first show that $\|A\|$ is at most say n^2. Then, prove that for every $k \geq 1$, A^k is also stochastic and $\|A^{2k}\mathbf{v}\|_2 \geq \|A^k\mathbf{v}\|_2^2$ using the equality $\langle \mathbf{w}, B\mathbf{z}\rangle = \langle B^\dagger \mathbf{w}, \mathbf{z}\rangle$ and the inequality $\langle \mathbf{w}, \mathbf{z}\rangle \leq \|\mathbf{w}\|_2 \|\mathbf{z}\|_2$.)

[8] One has to be slightly careful while making recursive calls, since we don't want to lose even the $O(\log\log n)$ bits of writing down k and keeping an index to the location in the input we're working on. However, this can be done by keeping k in global read/write storage and since storing the identity of the current step among the five calls we're making only requires $O(1)$ space.

§2 Let a n, d random graph be an n-vertex graph chosen as follows: choose d random permutations $\pi_1, ldots, \pi_d$ from $[n]$ to $[n]$. Let the the graph G contains an edge (u, v) for every pair u, v such that $v = \pi_i(u)$ for some $1 \leq i \leq d$. Prove that a random n, d graph is an $(n, 2d, \frac{2}{3}d)$ combinatorial expander with probability $1 - o(1)$ (i.e., tending to one with n). (Hint: for every set $S \subseteq n$ with $|S| \leq n/2$ and set $T \subseteq [n]$ with $|T| \leq (1 + \frac{2}{3}d)|S|$, try to bound the probability that $\pi_i(S) \subseteq T$ for every i).

§3 Prove the first part of Theorem 17.14: show that if S is any subset of at most half the vertices in a multigraph $G = (V, E)$ then the number of edges $\left|E(S, \overline{S})\right|$ going from S to $\overline{S}$ is at least $d(1 - \lambda(G))\,|S|/2$. (Hint: try to find a vector $\mathbf{v} \perp \mathbf{1}$ that corresponds to the set S. You can also try to prove a weaker version with 2 replaced with a bigger constant, possibly depending on d but not on n.)

§4 Let A be an $n \times n$ matrix with eigenvectors $\mathbf{u}^1, \ldots, \mathbf{u}^n$ and corresponding values $\lambda_1, \ldots, \lambda_n$. Let B be an $m \times m$ matrix with eigenvectors $\mathbf{v}^1, \ldots, \mathbf{v}^m$ and corresponding values $\alpha_1, \ldots, \alpha_m$. Prove that the matrix $A \otimes B$ has eigenvectors $\mathbf{u}^i \otimes \mathbf{v}^j$ and corresponding values $\lambda_i \cdot \alpha_j$.

§5 (*Expander Mixing Lemma*) Let S, T be any two subsets of nodes in a d-regular n-node graph G. Then the set of edges $E(S, T)$ with one endpoint in S and the other in T satisfies:

$$\left| |E(S,T)| - \frac{d\,|S|\,|T|}{n} \right| \leq \lambda(G) d \sqrt{|S|\,|T|}.$$

(Motivation: If G were a random d-regular graph then we would expect $|E(S, T)|$ to be $d\,|S|\,|T|/n$. The Lemma shows that an expander "looks like" a random graph.)