Doctoral mini-course on expander graphs
December 19, 2024 – Lecture 4

1 s, t-connectivity

Given a graph G and two vertices s and t, the *s,t-connectivity* problem consists in deciding whether s and t are in the same connected component of G (in other words, whether there is a path between s and t in G). There is a simple polynomial time algorithm for this (just run some depth-first search in the graph starting from s, for instance), but we would like to do more: solve the problem in *logarithmic space*. From the point of view of Turing machines this means that there are two tapes: the first one contains the input and is read-only, and the second one is the workable tape and has logarithmic size. Let L be the class of problems that can be solved in logarithmic space. It is known that $\mathsf{L} \subseteq \mathsf{P}$ (because there are only a polynomial number of possible states of the memory, and problems can be reduced to the reachability of some vertex in some polynomial size directed graph, which can be done in polynomial time). It is not known whether $\mathsf{L} = \mathsf{P}$, or even whether $\mathsf{L} = \mathsf{NP}$.

Note that graph problems lying in L are severely restricted: in the corresponding algorithms we can only store a constant number of names of vertices and a constant number of counters (which are polynomially bounded). Whether s,t-connectivity lies in L was a long-standing open problem, solved by Reingold in 2004 using the zig-zag product introduced in the previous lecture. Before sketching the proof, let us observe that s,t-connectivity lies in RL, the class of problems having a (Monte Carlo) randomized algorithm running in logarithmic space, and that s,t-connectivity can also be solved deterministically in $O(\log^2 n)$ space very simply.

Theorem 1. *s,t-connectivity lies in RL.*

Proof. Do a random walk starting from s, and if the walk meets t in at most $O(n^3)$ steps, report that s and t are in the same component. Otherwise report that s and t are not in the same component. Note that this can easily be implemented in logarithmic space (you just need to store the name of the current vertex of the walk, and the value of a counter for the number of steps).
Note that if s and t are not in the same component, the random walk will never meet t, so the algorithm will always be correct in this case. If s and t

are in the same component, then we can use a classical property of random walks: after $\Omega(n^3)$ steps with high probability every vertex of a connected n-vertex graph has been visited by the random walk, and thus t is very likely to have been spotted. □

Theorem 2. *s,t-connectivity can be solved deterministically in $O(\log^2 n)$ space.*

Proof. Given a constant $D \geq 1$, let $S(D)$ be the space complexity of deciding whether two vertices u and v lie at distance at most D in G. The algorithm is the following: if $D = 1$ we simply check whether u and v are adjacent. Otherwise $D \geq 2$ and for each vertex w, we test whether $d(u,w) \leq \lfloor D/2 \rfloor$ and $d(v,w) \leq \lceil D/2 \rceil$ (one after the other) and if both are true we answer yes.
The space needed is $S(D) \leq O(\log n) + S(D/2)$, and thus

$$S(D) = O((\log n)(\log D)).$$

In particular, testing whether two vertices are at distance at most n takes $O(\log^2 n)$ space. By testing whether $d(s,t) \leq n$, we obtain a $O(\log^2 n)$ space algorithm to decide whether s and t are in the same component. □

We now sketch the proof of a major result of Reingold: s,t-connectivity lies in L. Using earlier results, this directly implies that $\mathsf{RL} = \mathsf{L}$, and thus every randomized logarithmic space algorithm can be made completely deterministic.

We start by transforming our original graph (call it H) by replacing every vertex by a cycle: this produces a 3-regular graph, and we then add self-loops to all vertices to make the graph d-regular (with d constant, but sufficiently large). Note that the new graph still has polynomial size. Moreover, each connected component has $\lambda^* < d$.

We then take a small expander graph G and we perform the same kind of construction as in the previous lecture. Starting with H, we repeat the following: make the zig-zag product with G (to decrease the degree while not losing too much on the expansion), and then raise our graph to some small power (this improves the expansion but increases the degree). The point is that the value λ^* in each component decreases at each step, and after a logarithmic number of steps:

- the current graph has constant degree, and size polynomial in the size of H.
- each connected component of the current graph has λ^* bounded by a constant strictly less than 1 (say 1/2).

In particular, each component has logarithmic diameter, as we observed in the first lecture. But then telling whether s and t are in the same component can easily be done in logarithmic space, as we can simply enumerate all walks of size $t = c \log n$ in logarithmic space (recall that our current graph has constant degree, this is crucial).

The most difficult part of the proof consists in showing that all the steps of the construction can be performed in logarithmic space.

Let us be a bit more specific about how the value λ^* decreases step after step. Let us take $H_0 = H$, and for any $i \geq 0$ let us consider $H_{i+1} = (H_i \otimes G)^8$. We will need the following result, which is a variant of a result stated in the last lecture, but which is mostly interesting when λ^* is close to 1 and what we mostly care about is the gap $1 - \lambda^*$, rather than λ^* itself.

Theorem 3. *For any two graphs H and G,*

$$1 - \lambda^*(H \otimes G) \geq \tfrac{1}{2}(1 - \lambda^*(G)^2)(1 - \lambda^*(H)).$$

In particular when we choose our small expander G we can make sure that $\lambda^*(G) \leq \frac{1}{2}$, and it follows that for any graph H,

$$1 - \lambda^*(H \otimes G) \geq \tfrac{3}{8}(1 - \lambda^*(H)).$$

As a consequence, for any $i \geq 0$,

$$\lambda^*(H_{i+1}) = \lambda^*((H_i \otimes G)^8) = (\lambda^*(H_i \otimes G))^8 \leq \big(1 - \tfrac{3}{8}(1 - \lambda^*(H_i))\big)^8.$$

It can be checked that for any $x \in [0, 1]$, $(1 - \frac{3}{8}(1 - x))^8 \leq \max(\frac{1}{2}, x^2)$. This implies that $\lambda^*(H_{i+1}) \leq \max(\frac{1}{2}, \lambda^*(H_i)^2)$ and thus a simple induction shows that for any $i \geq 0$, $\lambda^*(H_i) \leq \max(\frac{1}{2}, \lambda^*(H)^{2^i})$.

Note that when $\alpha \leq 1$, $\alpha \leq \exp(\alpha - 1)$ and thus $\alpha^t \leq \exp((\alpha - 1)t)$. If the original graph H is connected (otherwise the same analysis can be done individually in each connected component of H), it can be proved that the spectral gap $1 - \lambda^*(H)$ is at least $1/n^2$, so by taking $k = \Omega(\log n)$ we have $2^k \geq n^2$ and thus $\lambda^*(H)^{2^k} \leq e^{-1} \leq \frac{1}{2}$. Hence, $\lambda^*(H_k) \leq 1/2$, as desired.

2 Error-correcting codes

Alice and Bob want to send each other messages of k bits over a noisy channel. To ensure that the message is correctly decoded despite the noise (random error on a fraction of p of the bits of the message), they encode each word of $\{0,1\}^k$ as a word of $\{0,1\}^n$ ($n > k$) (these words are called *code words*) such that the minimum Hamming distance between two code words is as large as possible. Recall that the *Hamming distance* between two binary words $(x_i)_{1\leq i\leq n}$ and $(y_i)_{1\leq i\leq n}$ is the number of entries $1 \leq i \leq n$ such that $x_i \neq y_i$. If any two code words are at Hamming distance more than $2pn$, then the "closest" code word from any n-bit word is unique (since it is at distance at most pn from a code word), so the message can be decoded.

The parameter n is called the *block length*. We also define the following parameters for the code : the *rate* k/n, and the *relative distance* which is the minimum Hamming distance between two code words, divided by n.

An infinite family of codes $\mathcal{C}$ is said to be *asymptotically good* if there are constants $c_1 > 0$ and $c_2 > 0$ such that all the rates are at least c_1 and all the relative distances are at least c_2 (in practice we also want efficient encoding and decoding procedures).

We will consider *linear codes*: the code words will form a linear subspace of $\mathrm{GF}(2)^n$ of dimension k (such a subspace contains precisely 2^k vectors, so these vectors are in bijection with the set of k-bit words). Equivalently, a code on n-bit words is linear if for any two code words, their entry-wise sum modulo 2 is also a code word.

Observe that in such a code the relative distance is precisely the minimum weight of a non-zero code word, divided by n.

Code words in linear codes can be seen as the original message (i.e. all possible words on k bits) together with $n-k$ *check bits*, each of which is a linear combination of bits of the original message (it is clear with this definition that the code words form a linear subspace of dimension k, but in fact any code words in such linear subspace can be divided into the original message and a number of check bits). In the following we really think of a linear code of block length n and rate k/n as a linear subspace of $\mathrm{GF}(2)^n$ defined by $n-k$ (linearly independent) linear equations.

Take a linear code C of block length d, rate r, and relative distance δ (such a code has dimension rd so can be defined by a set of $(1-r)d$ linear equations).

For instance we can take C to be the Hamming$(7,4)$ code, which encodes 4 bits with 7 bits using 3 additional parity bits (the code has block length 7, rate $4/7$ and relative distance $3/7$).

For any integer n, we will build a linear code C' of block length $dn/2$, rate $2r-1$, and relative distance at least $(\delta - \lambda/d)\delta$. So we produce an infinite family of asymptotically good codes.

For this we consider an (n,d,λ)-graph G, and we identify codewords of C' with (characteristic vectors of) edge-sets of G with the property that for any code word W in C', and any vertex v, the restriction of W on the edges incident to v is a code word of C. This can be done by writing $(1-r)d$ linear equations for each vertex v, so in total $(1-r)dn$ equations. It follows that the subspace we obtain has dimension at least $dn/2 - (1-r)dn = dn(r-1/2)$. So the rate is at least $dn(r-1/2)/(dn/2) = 2r-1$, as claimed.

Now what is the relative distance of C'? As we observed earlier, we only have to compute the minimum weight of a code word of C', divided by the block length, which is here the minimum number of edges of a code word, divided by $dn/2$.

Recall the *Expander mixing lemma*, which we have seen in the first lecture: For all subsets $S,T \subseteq V(G)$, $|e(S,T) - d|S||T|/n| \leq \lambda\sqrt{|S||T|}$. By taking $S = T$ we obtain that $e(S,S) \leq d|S|^2/n + \lambda|S|$. Since $e(S,S)$ is twice the number of edges in the subgraph induced by S, if $|S| < an$ we have that $E(G[S])$ has size less than $\frac{1}{2}an(ad+\lambda)$.

Assume that some code word W of C' contains $\frac{1}{2}an(ad+\lambda)$ edges, for some $a \leq 1$ (there is a real a such that W contains precisely this number of edges). Then the subgraph induced by the endpoints of the edges of W contains at least an vertices. It follows that one of these vertices sees at least one and at most $2\big(\frac{1}{2}an(ad+\lambda)\big)/an = ad+\lambda$ edges of W. This corresponds to a non-zero word of C with (relative) weight $a+\lambda/d$. This is at least δ, and so $a+\lambda/d \geq \delta$.

The number of edges of W is $\frac{1}{2}an(ad+\lambda)$, so the relative distance is at least $(\frac{1}{2}an(ad+\lambda))/(dn/2) = a(a+\lambda/d) \geq (\delta - \lambda/d)\delta$, as desired.

We haven't yet mentioned encoding and decoding. Encoding is easy, as it is usually the case for linear codes. Decoding can also be made efficient, but this is far from trivial.