0368-4283: Space-Bounded Computation 24/4/2018 – Lecture 6

USTCONN $\in$ L via Derandomized-Squaring

Amnon Ta-Shma and Dean Doron

1 USTCONN $\in$ L via Derandomized-Squaring

We will solve USTCONN by increasing the connectivity of the graph (more accurately, of every connected component) by repeatedly using derandomized squaring. Iterating derandomized squaring highly increases connectivity, as we saw in the last section, and the degree blowup is relatively small.

Recall that had we computed G^n exactly, an algorithm for undirected st-connectivity would be just to enumerate over all neighbors of s. When $\bar{\lambda}(G)$ is small enough, the same technique can be applied.

Claim 1. *Let G be an $(N, D, \frac{1}{2N^2})$-graph. Then, G contains an edge between any pair of vertices. More generally, for a pair of vertices v, w, the probability that a random neighbor of v is w is at least $\frac{1}{N} - \frac{1}{N^2}$.*

The latter part of the above claim will be useful once we talk about pseudorandom generators.

Proof. Let M be the transition matrix of G and let $u = \frac{1}{N}\mathbf{1}$ be its stationary distribution. The initial probability distribution is e_v, and so

$$\|Me_v - u\|_\infty \;\le\; \|Me_v - u\| \;=\; \|M(e_v - u)\| \;\le\; \bar{\lambda}(G)\,\|e_v - u\|\,,$$

where the last inequality follows from the fact that $e_v - u \perp \mathbf{1}$. As $\|e_v - u\| \le 2$, we are finished. □

1.1 Phase I

We start with some 4-regular undirected graph G with a loop on each vertex. This captures the most general case (why?). We already saw that $\bar{\lambda}(G) \ge 1 - \frac{1}{4N^2}$. In phase I, we apply derandomized squaring for $m_0 = O(\log N)$ times with a constant degree expander and obtain G_{m_0} – a graph with a constant spectral gap that preserves the connectivity of the original graph.

We begin with a technical step. Take G to some constant power q. The degree of $G_1 = G^q$ is $Q = 4^q$ and q is chosen so that there exists a sequence H_m of $(Q^m, Q, \frac{1}{100})$ graphs for every m (recall that we can hope for degree $\approx \frac{1}{\lambda^2}$). Such family of graphs explicitly exists, in the sense that given a vertex $v \in [Q]^m$ and label $x \in [Q]$ we can compute $v[x]$ in space $O(m)$.

Theorem 2. *Set $m_0 = O(\log N)$ and for every $i \in [m_0 - 1]$ define $G_{i+1} = G_i \textcircled{S} H_i$. Then, $\bar{\lambda}(G_{m_0}) \le \frac{3}{4}$.*

Proof. We start with $\gamma(G_1) \ge \frac{1}{4N^2}$. By Corollary 7 of the previous lecture, it follows that

$$\gamma(G_{i+1}) \;\ge\; \gamma(G_i) \cdot \frac{3}{2} \;\ge\; \left(\frac{3}{2}\right)^m \gamma(G_1) \;\ge\; \left(\frac{3}{2}\right)^m \frac{1}{4N^2},$$

as long as $\gamma(G_i) < \frac{1}{4}$. Choosing m_0 to be a large enough multiplicity of $\log N$ will give us $\bar{\lambda}(G_{m_0}) \leq \frac{3}{4}$. □

1.2 Phase II

We need to keep increasing the spectral gap, but we cannot do that with a constant degree expander (why?). In the second stage, we each time square the degree of the H expander we use, and we keep doing that $\log\log N + O(1)$ times. We end up with a graph with polynomial degree and polynomially small second largest eigenvalue. It is then enough to just over all the neighbours of s, and we show this can be done in $O(\log N)$ space.

We define the next family of expanders,

$$H'_m \;=\; \left(H_{m_0-1+2^{m-m_0}}\right)^{2^{m-m_0}}$$

for $m \geq m_0$. So, every H'_m is a

$$\left(Q^{m_0-1+2^{m-m_0}}, Q^{2^{m-m_0}}, \mu_m = \left(\frac{1}{100}\right)^{2^{m-m_0}}\right)$$

graph. Neighbors in H'_m are again explicit, and can be computed in space $O(m + 2^{m-m_0})$. Recall that G_{m_0} is an $(N, Q^{m_0}, \frac{3}{4})$-graph.

Theorem 3. *Set $m_1 = m_0 + \log\log N + O(1)$ and for every $i \in \{m_0, \ldots, m_1 - 1\}$ define $G_{i+1} = G_i \circledS H'_i$. Then, the degree of G_{m_1} is polynomial in N and $\bar{\lambda}(G_{m_1}) \leq \frac{1}{2N^2}$.*

Proof. Observe that the degree of G_{m_0+i} is $Q^{m_0+\sum_{k=0}^{i-1} 2^k} = Q^{m_0+2^i-1}$, so the degree of G_{m_1} is $Q^{O(\log N)} = \mathrm{poly}(N)$.

For the bound on $\bar{\lambda}(G_{m_1})$, we will prove by induction that $\lambda(G_{m_0+i}) \leq \frac{64}{65} \cdot (7/8)^{2^i}$ and the theorem will follow. Indeed, for $i = 0$ the claim holds. Now, by Theorem 5 of the previous lecture,

$$\bar{\lambda}(G_{m_0+i+1}) \;\leq\; \bar{\lambda}(G_{m_0+i})^2 + \mu_{m_0+i} \;\leq\; \left(1 + \frac{1}{64}\right)\bar{\lambda}(G_{m_0+i})^2 \;\leq\; \frac{64}{65}\left(\frac{7}{8}\right)^{2^{i+1}},$$

where we used the fact that $\mu_{m_0+i} = (1/100)^{2^i} \leq \frac{1}{64}\left(\frac{64}{65}(7/8)^{2^i}\right)^2$. □

The fact that derandomized squaring preserves the connectivity of our original graph (that is, s is connected to t in G iff s is connected to t in G_i for every i) simply follows from the fact that our original graph has self loops (verify!).

1.3 The space complexity

The next theorem will finish our analysis.

Theorem 4. *For every vertex s, the neighbours of s in G_{m_1} are computable in space $O(\log N)$.*

Proof. (Sketch). Edge labels in G_i are of the form $\overline{y}_i = (y_1, a_1, \ldots, a_{i-1})$ where y_1 is an edge label in G_1 and a_i is an edge label in G_i. Given $v \in [N]$ and y_i, we want to compute $Rot_{G_i}(v, \overline{y}_i)$.

The algorithm: We follow the recursive definition of Rot_{G_i} except that we do not use recursive calls (that blow up the stack) but instead use a loop (where it is easier to see what is going on) and we keep a depth i tree of arity 3. Each node (except for a leaf) has a left "G" son, middle "H" son and right "G" son, and in the loop we remember where we are on the tree. When:

- When we are at a leaf, we apply Rot_G and return to the father.
- When we are at a "H" son we apply Rot_H at a prefix whose length corresponds to the level of the tree we are in (work out the details!) and return to the father.
- When we are at a "G" vertex, we go down first to its left son, when we return we go to the middle son, and when we return to the right son. When we return to the root we half.

Now it is easy to see that the space complexity is $O(\log N)$, needed to keep where we are on the tree, plus the space needed to do one application of Rot_G and and one application (the most expansive) of Rot_{H_i}. □

Notice that during the computation we write (again and again) on the register that initially stored $\bar{y}_i$, seemingly erasing information. However, since the *Rot* operator is reversible, this is a reversible in-place computation, and if we wish we can always recover information we had before.

Another point to notice is that if $Rot_G(v, a) = (v[a], \phi(a))$ for some function ϕ that does not depend on v (this, e.g., happens in a Cayley graph), then $Rot_{G_i}(y_1, a_1, \ldots, a_{i-1})$ open to a sequence of instructions that is independent of y_i, that is then applied on the situation where y_1 is the initial vertex. The length of that sequence is 2^i. Also, given $b = (b_0, \ldots, b_{i-1})$ the b'th symbol in that sequence can be computed in $O(\log N)$ space (do that!). We will get to that point in the next lecture when we discuss PRG.