CS 880: Pseudorandomness and Derandomization 3/11/2013

Lecture 14: Pseudorandom Generators for Logarithmic Space

Instructors: Holger Dell and Dieter van Melkebeek Scribe: Adam Everspaugh

DRAFT

Last time, we discussed *derandomized squaring*, a graph operation that reduces the second-largest eigenvalue while increasing the degree only by a constant factor. This lecture uses an iterative application of derandomized squaring to construct a pseudorandom generator that runs in logarithmic space and can be used to solve the undirected connectivity problem in logarithmic space.

1 Undirected Connectivity

Given an undirected graph G and vertices $s, t \in V(G)$, the undirected connectivity problem is to decide whether there is a path from s to t in G. If G has N vertices and M edges, then the existence of a path between s and t implies the existence of a path of length at most N. If A is the adjacency matrix of G and G has, without loss of generality, self-loops on every vertex, then s and t are connected if and only if $(A^N)_{t,s} > 0$ holds. In other words, the goal of the connectivity problem is to decide if there is a non-zero probability that, starting at vertex s and after traversing N steps in G, we will arrive at vertex t.

1.1 Algorithms for undirected connectivity

Classical solutions to the connectivity problem include breadth- and depth-first search, both of which use a relatively large amount of space in their algorithms. We first discuss a few simple algorithms for undirected connectivity before we dive into the logarithmic space algorithm.

Matrix multiplication. As already indicated above, raising the adjacency matrix A to the power N corresponds to walking N steps in the graph. Therefore, perhaps the simplest way to compute the number of N-step walks is to iteratively multiply by A:

$$A \longrightarrow AA = A^2 \longrightarrow A^2A = A^3 \longrightarrow \ldots \longrightarrow A^N$$

If the algorithm is allowed to use polynomial space, it runs in time $O(N \cdot N^\omega)$, where $O(N^\omega)$ is the time it takes to multiply two $N \times N$ matrices. The algorithm for computing an entry $(A^N)_{t,s}$ can also be implemented in space $O(N \log N)$ since there are N iterations, and in each iteration we need to keep track of $O(\log N)$ bits of information when computing the inner products in the matrix multiplication. In the latter implementation, the running time blows up to $\exp(O(N \log N))$ since we have to recompute previously computed results in order to save space. The first implementation above can be thought of as following the dynamic programming paradigm since the intermediate matrices are stored in memory in order to save time.

Repeated Squaring. As is the case with computing the power of integers, we can speed up the above naïve algorithm for computing the power of a matrix by repeated squaring.

$$A \longrightarrow A^2 \longrightarrow (A^2)^2 = A^4 \longrightarrow \ldots \longrightarrow (A^2)^{\log N} = A^N$$

Here we assume for simplicity that N is a power of 2 so that $\log N$ is an integer. This algorithm requires $\log N$ iterations, and if we have polynomial space, it runs in time $O(\log N \cdot N^\omega)$. It can also optimized for low space usage: in each iteration we need to keep track of only $O(\log N)$ bits, so the total space requirement is $O(\log^2 N)$. The running time of the latter implementation is $\exp(O(\log^2 N))$.

Random walk. If we have access to randomness, then there is a simple algorithm to solve the connectivity problem: Take a random walk of length $\ell = 2^k$ in G for some integer k. This algorithm uses only $O(\log N)$ space since this is the number of bits required to store the vertex you are currently at and compare it to t. The amount of randomness required is quite large: If we are currently located at a vertex v of degree D, then we need $\log D$ bits to decide which neighbor to walk to. Without loss of generality, we can assume G to be D-regular. In this case, we need $\ell \cdot \log D$ bits to perform an ℓ-step random walk.

We will see below that the random walk started at s and stopped after a polynomial number of steps will output a vertex that is close to uniformly distributed. So let us first see what happens if we were able to take a uniform sample from the unique connected component $C(s) \subseteq V(G)$ that contains s: If v is uniformly distributed on $C(s)$, and s and t are not connected, then the probability that $v = t$ is equal to zero. Otherwise, the probability is equal to $1/|C(s)| \geq 1/N$. Thus, uniformly sampling v from $C(s)$ and checking whether $v = t$ gives rise to a randomized algorithm with one-sided error, and we can boost our confidence by repeating the process a polynomial number of times. In particular, by repeating the process $O(\log(N/\epsilon))$ times, we can reduce our one-sided error to at most ϵ.

The next step in the argument is to show that taking a random walk starting at s and selecting the vertex v after $O(N^2 \log N)$ random steps is very close to the uniform distribution on $C(s)$. For this, we use the fact that every connected D-regular directed graph G with loops on all vertices has a second-largest eigenvalue that is polynomially bounded away from 1:

Lemma 1. *Let G be a connected D-regular directed graph with loops on all vertices. Then $\lambda(G) \leq 1 - \frac{1}{2D^2N^2}$.*

After ℓ steps, we see that $\lambda(G^\ell) = \lambda(G)^\ell \leq (1 - \frac{1}{2D^2N^2})^\ell \leq \exp(-\ell/(2D^2N^2))$. Thus for $\ell = O(D^2N^2 \log N)$ steps, we have $\lambda(G^\ell) \leq \frac{1}{2N^{1.5}}$. In this situation, we can apply the following lemma, where M is the transition matrix of G^ℓ.

Lemma 2. *Let M be a random walk matrix of a graph with N vertices. If $\lambda(M) \leq \frac{1}{2N^{1.5}}$, then $(M)_{ij} \geq \frac{1}{N} - \frac{1}{N^2}$.*

This second lemma is useful because it guarantees that a uniformly random neighbor of s in the graph G^ℓ is $\frac{1}{N^2}$-close to uniformly on $C(s)$. In particular, every vertex in $C(s)$ will be reached with positive probability. In order to derandomize the random walk algorithm, we can therefore cycle through all edges incident to s in G^ℓ and check whether they lead to t. $\ell = 2^k$, this approach turns out to be equivalent to the repeated squaring idea from above. The repeated squaring brings down the second-largest eigenvalue to an inverse polynomial value in the number of vertices, but it comes at the cost of the degree, which gets squared every time.

2 Derandomized power graph

In Definition 2 of Lecture 13, we introduced the *derandomized squaring* operation: it approximates the effect that squaring has on the second-largest eigenvalue, but it increased the degree only by a constant factor instead of squaring it. We will now iteratively apply this operation, and then perform a random step in the so-obtained derandomized power graph. Intuitively, since this graph approximates a true power graph G^{ℓ}, a single random step in the derandomized power will approximate an ℓ-step random walk in G.

For simplicity, we define the derandomized power graph only for exponents $\ell = 2^i$ that are powers of two. Furthermore, we assume that G is D-regular for some D. The 2^i-th derandomized power of G is denoted as $G^{\approx 2^i}$ and defined inductively as follows:

$$\begin{aligned}
G^{\approx 2^0} &\doteq G \\
G^{\approx 2^1} &\doteq G^{\approx 2^0} \circledS H_1 = G \circledS H_1 \\
G^{\approx 2^2} &\doteq G^{\approx 2^1} \circledS H_2 = (G \circledS H_1) \circledS H_2 \\
&\vdots \\
G^{\approx 2^{i+1}} &\doteq G^{\approx 2^i} \circledS H_{i+1} \,.
\end{aligned}$$

For each i, the graph H_i is some regular directed graph, and we denote its degree by d_i. The sequence $H_1, H_2, \ldots$ is a family of expanders that may depend on G in a limited way. In our case, it will depend only on the number of vertices of G and the degree D. For the definition to be well-defined, we need the number of vertices of H_{i+1} to coincide with the degree of the graph $G^{\approx 2^i}$. It is easy to see inductively that $G^{\approx 2^i}$ has degree $D \cdot \prod_{j=1}^{i} d_j$.

2.1 Random steps in the derandomized power graph

We mentioned above that the derandomized power graph $G^{\approx 2^k}$ that approximates the graph G^{2^k} with respect to the second-largest eigenvalue if the H_i's are good expanders. For technical reasons, we have to use a different family of expanders for different values of N. The following lemma captures the eigenvalue properties of the derandomized power graph that we will need to apply Lemma 2.

Lemma 3. *For all positive integers D and N, there is a fully explicit family of graphs $H_1, H_2, \ldots$ and a number $k = O(\log D + \log N)$ such that:*

(i) for all D-regular N-vertex graphs G, we have that $\lambda(G^{\approx 2^k}) \leq \frac{1}{2N^{1.5}}$, and

(ii) we have $\log(\prod_{i=1}^{k} d_i) = \sum_{i=1}^{k} \log d_i \leq O(k)$.

Proof. The construction is in two phases. The first k' graphs in the sequence will have the same constant degree and a small, but constant, second-largest eigenvalue. This will bring down the second-largest eigenvalue of $G^{\approx 2^{k'}}$ to a constant. In the second phase, the degree increases doubly exponentially, but the phase only lasts for $k - k'$ iterations, which we will set so that $k - k' = O(\log k)$.

More precisely, we define $H_1, \ldots, H_{k'}$ to be d-regular graphs with second-largest eigenvalue at most $1/100$ and so that H_i has $D \cdot d^i$ vertices. To prove (i), let λ be the second-largest eigenvalue of G and let $\lambda_i \doteq \lambda(G^{\approx 2^i})$ be the eigenvalues of the derandomized power graphs. Initially, we have

$\lambda_0 = \lambda$ and, by Lemma 1 of Lecture 13, we have $\lambda_i \leq (1-\mu)\lambda_{i-1}^2 + \mu$ with $\mu \leq 1/100$. It is an easy exercise to verify that

$$1 - \lambda_i \geq (3/2)(1 - \lambda_{i-1})$$

holds as long as $\mu \leq 1/100$ and $\lambda_{i-1} \geq 3/4$. Thus, for some $k' = O(\log \frac{1}{1-\lambda})$, we have $\lambda_{k'} < \frac{3}{4}$. By Lemma 1, the second-largest eigenvalue of *any* directed D-regular connected graph with all self-loops is at most $1 - 1/(2D^2N^2)$; thus, the derandomized power graph $G^{\approx 2^{k'}}$ has second-largest eigenvalue $< 3/4$ after $k' \leq O(\log D + \log N)$ iterations of the derandomized squaring.

For the second phase, we define $H_{k'+i}$ for $i > 0$ to be a $d_{k'+i}$-regular graph with $d_{k'+i} = 2^{O(2^i)}$ and second-largest eigenvalue at most $1/2^{O(2^i)}$. Choosing $k = k' + \log\log N + O(1)$, it can be seen similar to above that $\lambda_k \leq 1/\operatorname{poly}(N)$ holds, which establishes (i).

For (ii), note that d is constant and thus $k' \cdot \log d \leq O(k)$ is an upper bound for the sum of the first k' terms. The remaining $k - k'$ terms in the sum are $\sum_{i=1}^{k-k'} \log 2^{O(2^i)} = O(\sum_{i=1}^{\log\log N + O(1)} 2^i) \leq O(\log N) \leq O(k)$. $\square$

After $O(\log N)$ derandomized squaring steps, we reduced the second-largest eigenvalue enough to be able to apply Lemma 2: we get that performing a single step in $G^{\approx 2^k}$ gives us a vertex that is close to uniformly distributed in the connected component of the vertex. The degree of the derandomized power graph $G^{\approx 2^k}$ is $O(k)$ by the above lemma, and the number of bits that we need to sample to perform this step is only $\log D + k \log d = O(\log N)$ as opposed to $2^k \log D = O(N)$ in the case of the true power graph G^{2^k}. By cycling over all random seeds, this method will eventually allow us to solve undirected connectivity in logspace. Before we can do this, however, we need to discuss how we can actually compute neighbors in $G^{\approx 2^k}$, and for this, we will go back to the formal definition of the derandomized square that uses the neighbor function.

2.2 Edge labels of the derandomized power graph

Recall that in Definition 2 of Lecture 13, the formal definition of the derandomized squaring $G^{\approx 2} = G \circledS H_1$, we needed to use the underlying edge labels as specified implicitly by the neighbor function $\Gamma_{G^{\approx 2}} : V(G) \times ([D] \times [d]) \to V(G)$. In order to see that we can compute a random step in $G^{\approx 2^k}$ for $k \leq O(\log N)$ in logarithmic space, we need to understand these edge labels a little bit better.

Let's start simply by looking at $G^{\approx 2} = G \circledS H_1$. Recall that its transition matrix is $M(G \circledS H_1) = P\hat{A}\hat{B}\hat{A}L$, where L is the lifting and P is the projection operation for clouds of the replacement product $G \circledR H$. Recall also that we multiply from the right, so L is the first operation and so on. Thus, every edge (v, w) in $G^{\approx 2}$ corresponds to a sequence of steps in the union of G and $G \circledR H_1$:

1. (L) randomized lifting from a vertex of G to the corresponding cloud in the replacement product $G \circledR H_1$ (picking a vertex in the cloud can described by a label $h_0 \in [D] = V(H_1)$).

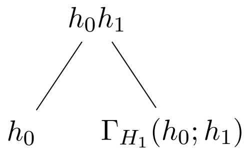
2. ($\hat{A}$) deterministic step to a new cloud.

3. ($\hat{B}$) randomized step within the cloud (which is described by an edge label $h_1 \in [d_1]$ of H_1).

4. ($\hat{A}$) deterministic step to a new cloud.

5. (P) deterministic projection back onto G.

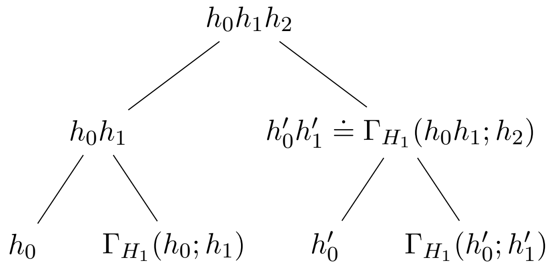
Note that only steps 1 and 3 are randomized, and thus, these are the only steps for which we need to specify the edge label that we take in order to uniquely describe a particular walk. If we call these random choices $h_0 \in [D]$ and $h_1 \in [d]$, then the edge label of (v, w) in $G^{\approx 2} = G \circledS H_1$ is $h_0 h_1$. Recall that $G^{\approx 2}$ is a subgraph of G^2, and thus, each edge describes some walk from v to w in G that has length 2. This 2-step walk in G can be described by the edges labels of G, that is, labels from the set $[D] = V(H_1)$. The lifting step together with the first $\hat{A}$-step corresponds to the first step in this 2-step walk. In particular, the first edge label we have to take in G is h_0. Then the second edge label is determined by the $\hat{B}$-step, and so, it is the h_1-neighbor of h_0 in H_1. That is, the second edge label is $\Gamma_{H_1}(h_0; h_1)$. We can depict this relationship as follows:



The edge label $h_0 h_1$ of $G^{\approx 2}$ translates to the sequence of edge labels $h_0, \Gamma_{H_1}(h_0; h_1)$ in G.

In the general case of $G^{\approx 2^i} = G^{\approx 2^{i-1}} \circledS H_i$, the lift step needs to choose a vertex in $V(H_i)$, and the $\hat{B}$-step needs to choose an edge in H_i. The latter can again be described by an edge label $h_i \in [d]$. For the former, note that $V(H_i)$ has size exactly Dd^{i-1} since that is the degree of $G^{\approx 2^{i-1}}$. Thus, an element of $V(H_i)$ is of the form $h_0 h_1 \dots h_{i-1}$, where $h_0 \in [D] = V(H_1)$ and $h_1, \dots, h_{i-1} \in [d]$. Moreover, such $h_0 \dots h_{i-1}$ is exactly an edge label of $V(G^{\approx 2^{i-1}})$. Thus, the edge labels of G_i are of the form $h_0 \dots h_i$.

Let us look at the case $i = 2$ and let us consider an edge label $h_0 h_1 h_2$ of $V(G^{\approx 4})$. We can recursively compute the walk in G that corresponds to this edge label by considering the following binary tree:


$h_0 h_1 h_2$
$h_0 h_1$
$h'_0 h'_1 \doteq \Gamma_{H_1}(h_0 h_1; h_2)$
h_0
$\Gamma_{H_1}(h_0; h_1)$
h'_0
$\Gamma_{H_1}(h'_0; h'_1)$

The root of the tree contains the "input", an edge label of $G^{\approx 4}$. The sequence of labels in the second layer from left to right is the sequence of edge labels of $G^{\approx 2}$ that the root corresponds to. The edge labels at the leaves of the tree determine a walk of length 4 in G. This process of producing the edge labels of a pseudorandom walk in G can be seen as a pseudorandom generator. This view will allow us to easily show that it can be computed in logarithmic space.

2.3 The INW-Generator

We have seen above that the process that maps an edge of the derandomized power $G^{\approx 2^i}$ to a 2^i-step walk in G has a highly recursive structure. We now define a pseudorandom generator that captures this structure, but makes no reference to the derandomized square.

Definition 1 (Impagliazzo–Nisan–Wigderson Generator [INW94]). *Let D be a positive integer and let $H_1, H_2, \ldots$ be a sequence of directed outregular graphs such that H_i has out-degree d_i, vertex set $[D] \times [d_1] \times \cdots \times [d_{i-1}]$ and neighbor function $\Gamma_{H_i} : V(H_i) \times [d_i] \to V(H_i)$. Then the INW-generator is the family of functions $G_i : [D] \times [d_1] \times \cdots \times [d_i] \to [D]^{2^i}$ such that*

$$G_0(h_0) \doteq h_0$$
$$G_{i+1}(h_0 \ldots h_{i+1}) \doteq G_i\Big(h_0 \ldots h_i\Big) \circ G_i\Big(\Gamma_{H_{i+1}}(h_0 \ldots h_i; h_{i+1})\Big),$$

where $h_0 \in [D]$ and $h_j \in [d_j]$ for all $j > 0$.

Note that different families of graphs H_i give rise to different instantiations of the INW-generator. If the family is fully explicit, that is, the neighbor function is computable in space that is linear in its input, then the INW generator has the same property.

Fact 4. *Assume that the expander family $H_1, H_2, \ldots$ is fully explicit. Given $j \in [2^i]$ and $(h_0 \ldots h_i)$, we can compute the j-th position of $G_i(h_0 \ldots h_i)$, that is, the element $(G_i(h_0 \ldots h_i))_j \in [D]$, using $O(\log |h_0 \ldots h_i|) = O(\log D + \sum_{j=1}^{i} \log d_j)$ space.*

To see that the fact is true, we view j as a binary string of length i. At each point of the algorithm, we keep track of a position in that binary string and the current label $h_0 \ldots h_{i'}$. If the highest-order bit of j that we have not read yet is 0, we do not modify the current label except by deleting the last part $h_{i'}$ and we continue the recursion in the left branch. Otherwise, we compute the neighbor in $H_{i'}$ and continue in the right branch.

The following fact connects the labels generated by the INW-generator with the labels in the derandomized power graph.

Fact 5. *Let $h_0 \in [D]$ and $h_j \in [d_j]$ for $j > 0$, let G be a consistently labeled directed d-regular graph, and let $H_1, H_2, \ldots$ be a family of consistently labeled directed d-regular graphs. The following are equivalent:*

1. *There is an edge (v, w) with label $h_0 \ldots h_i$ in the graph $G \circledS H_1 \circledS \ldots \circledS H_i$.*
2. *If we start at v and move in G by following the sequence $G_i(h_0 \ldots h_i)$ of 2^i edge labels of G, we will end up at w.*

In other words, the INW-generator provides us with an equivalent definition of the iterated derandomized square graph. The two facts above can be seen as a formalization of the discussion in the previous section.

2.4 Undirected Connectivity in logspace

We are now in position to describe the logspace-algorithm for undirected connectivity. Recall that we are given an N-vertex graph G and two vertices s and t. We instantiate the INW-generator with the strongly explicit expander graph family $H_1, H_2, \ldots$ provided by Lemma 3, and we let $k = O(\log D + \log N)$ be the number provided by the same lemma. Then the logspace algorithm works cycles over *all* seeds of the INW-generator and, for each of them, performs the walk in G that starts at s and follows the edge labels given by the output of the generator. The algorithm accepts if and only if it ever visits t. Figure 1 contains the formal description of the algorithm.

```
Input: $N$-vertex $D$-regular graph $G$; vertices $s, t \in V(G)$
Decide: Is there a path from $s$ to $t$ in $G$?

  ∘ Let $H_1, H_2, \dots, H_k$ be the sequence of strongly explicit expanders provided by Lemma 3,
    where $k = O(\log D + \log N)$ is the number provided by the same lemma.
  ∘ Instantiate the INW-generator $G_k$ with the expander graphs $H_1, \dots, H_k$.
  ∘ Cycle over all $(h_0 \dots h_k) \in [D] \times [d_1] \times \dots \times [d_k]$.
      – Set the current vertex to $v := s$.
      – For each value $j = 1, \dots, 2^i$ in increasing order:
          ∗ Compute the edge label $\ell := \big(G_i(h_0 \dots h_i)\big)_j$ and set $v := \Gamma_G(v; \ell)$.
          ∗ If $v = t$ ever holds, we halt and accept.
  ∘ If $t$ has not been found in any iteration above, we reject.
```

Figure 1: The pseudocode of the logspace algorithm for undirected s-t-connectivity.

Theorem 6. *The algorithm in Figure 1 runs in logspace and solves the undirected connectivity problem.*

Proof. By Fact 4, the edge label ℓ can be computed in space $O(|h_0 \dots h_k|)$, which is also the space required to cycle through the outer loop. By the choice of expander graphs in Lemma 3, this is bounded by $O(\log D + \sum_{i=1}^{k} \log d_i) \leq O(k) = O(\log D + \log N)$. Therefore, the algorithm runs in space $O(\log N)$.

For the correctness, observe that the algorithm performs *all* pseudorandom walks starting at s and following the edge labels that are given by the output of the INW-generator G_k. Therefore, by Fact 5, the algorithm explores all neighbors of s in the derandomized power graph $G^{\approx 2^k}$ at least once. By Lemma 3 and Lemma 2, all vertices in the same connected component as s (with respect to G) are neighbors of s in the derandomized power graph $G^{\approx 2^k}$. Therefore, the above algorithm accepts if and only if t is in the same connected component of G as s. $\square$

We remark that G_k with the given choice of the expander family is a random walk generator that is known to be pseudorandom for consistently labeled graphs G. For not necessarily consistently labeled graphs, pseudorandom generators with seed length $O(\log N)$ are not known. However, it is known that finding such a pseudorandom generator would solve the full RL vs L question. RL is the class of decision problems that that can be solved by a randomized algorithm in logarithmic space, polynomial time, and with one-sided error, and L is the class of decision problems that can be solved using deterministic logarithmic-space algorithms.

Theorem 7. *If there is an ϵ-pseudorandom walk generator that works for all directed regular graphs, $G : \{0,1\}^\ell \to [D]^t$ with $\ell = O(\log(tD))$ and $t = \mathrm{poly}(N)$, then* $\mathrm{RL} = \mathrm{L}$.

References

[INW94] Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th Annual Symposium on Theory of Computing, STOC 1994*, pages 356–364, 1994.