Lecture 25: Nisan's generator for constant depth circuits

Instructor: Jin-Yi Cai Scribe: Aparna Das, Giordano Fusco

1 Overview

This lecture gives an overview of Nisan's psuedorandom generator for constant depth circuits, which is based on the fact that parity is hard to compute with constant depth circuits. We also introduce the Nisan-Wigderson generator and give the defintion of extractors.

2 A small circuit cannot compute parity

Using the results from the previous class it is possible to prove the following lemma. Intuitively it says that if the size of a circuit is bounded, then it is unlikely that it will compute the parity function (which we denote by $\oplus$). Since we can guess the value of the parity with probability 1/2, a circuit that computes parity has to give an answer that is correct with probability bigger then 1/2.

Lemma 2.1. *For any depth d circuit C_n of size $|C_n| \leq 2^{\frac{1}{d+1}}$, where n is the number of input bits, we have*

$$\left| \Pr_{x \in \{0,1\}^n} \left[C_n(x) = \bigoplus_{i=1}^{n} x_i \right] - \frac{1}{2} \right| \leq 2^{-n^{\frac{1}{d+1}}}$$

for all sufficiently large n.

Note that this result is a little bit weaker than the one given in [CW04]. In fact the bound proved in the paper is $2^{-\epsilon n^{\frac{1}{d}}}$ with a small constant ϵ. But for the purposes of this lecture it is enough.

3 Nisan's Psuedorandom generator for constant depth circuits

Now we present an overview of Nisan's Psuedorandom generator for constant depth circuits, which is based on the fact that parity is hard to compute with constant depth circuits. Consider the computation of a bunch of parities on subsets $S_i \subseteq [n]$, where $|S_i| = O(\log n)^d$ with $d = O(1)$ and $|S_i \cap S_j| \leq \log n$. The generator G is as follows:

$$G : (x_1, \ldots, x_n) \mapsto (\bigoplus_{i \in S_1} x_i, \bigoplus_{i \in S_2} x_i, \ldots, \bigoplus_{i \in S_m} x_i)$$

One can compute G using d depth, poly-sized circuits as follows. To compute $\oplus_{i \in S_i} x_i$, first divide the input into $\log n$ sized groups and then compute parity recursively as shown in figure 1. Since each $\oplus$ gate in figure 1 works on $\log n$ inputs, it can be converted into a linear circuit of depth 2 which uses only *and* and *or* gates. After converting each level as described, the total depth of the

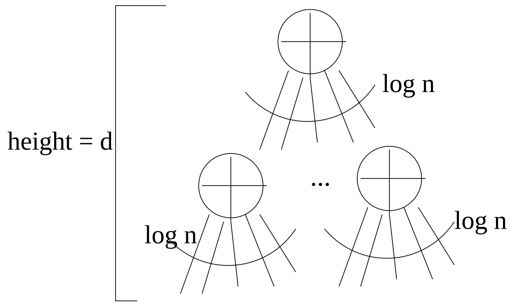

Figure 1: Parity

circuit would be about $2d$ and the size would be $2^d \cdot n = O(n)$. In fact a more careful analysis using the switching lemma shows that the circuit in figure 1 can be converted into a depth $d+1$ sized circuit. Thus G can be computed using a circuit of size $m \cdot O(n)$, and depth $d+1$. (Note that we can always ensure that m is polynomial by adjusting the size of the interestion between $|S_i \cap S_j|$).

Now we present a roungh sketch of the proof that G's output is indistinguishable from the uniform sequence $(y_1, y_2, \ldots, y_m)$ by $(d-3)$-depth circuits. We will show that if the output of G is distinguishable from $(y_1, y_2, \ldots, y_m)$, by $(d-3)$-depth circuits, then there exists a $(d-1)$-depth circuit which computes the parity on n bits. Since this is a contradiction to lemma 2.1 the indistinguishability of G by $(d-3)$-depth circuits follows.

The main steps of the proof are as follows. Assume we are given an ϵ-distinguisher for G. Yao's result which relates next bit unpredictability to indistinguishability implies the existence of a $\frac{\epsilon}{m}$ next bit predictor P for G. Specifically exists an index i such that

$$\Pr[P(\bigoplus_{j \in S_1} x_j, \ldots, \bigoplus_{j \in S_{i-1}} x_j) = \bigoplus_{j \in S_i} x_i] - \frac{1}{2} \geq \frac{\epsilon}{m} \tag{1}$$

In other words given the first $i-1$ output bits of G, P can predict the $G_i = \oplus_{j \in S_i} x_j$ with advantage $\frac{\epsilon}{m}$. Since G_i only depends on $(\log n)^d$ variables, we can evaluate equation (1), as an average over all possible choices of the remaining $[n] \setminus S_i$ variables, where only $x \in S_i$ are chosen at random. It follows that on average there exists a particular setting of the variables $[n] \setminus S_i$, such that the probability of predicting G_i is at least $\frac{1}{2} + \frac{\epsilon}{m}$.

Fix one such setting of the variable $[n] \setminus S_i$, so that only the variables in S_i can vary. Since the intersection of S_i with any other S_j is at most $O(\log n)$, each of the functions

$$\bigoplus_{j \in S_1 \cap S_i} x_j, \bigoplus_{j \in S_2 \cap S_i} x_j, \ldots, \bigoplus_{j \in S_{i-1} \cap S_i} x_j$$

depends on at most $(\log n)$ variables. Thus they can each be computed by small circuits of size $O(n)$ and depth 2. We can think of these circuits as helpers which allow us to compute the first $i-1$ bits of G. Plugging in the helpers into the predictor P, gives us a way to compute parity on n bits with advantage $\frac{\epsilon}{n}$. Since the depth of the predictor is $d-3$, the depth of the combined circuit (predictor-and-helpers) will be $d-1$. Since the predictor and the helpers are all poly-sized, the size of the combined circuit will also be polynomial. Thus we get a poly-sized, depth $d-1$ circuit which computes parity, which contradicts lemma 2.1.

4 Construction of nearly disjoints Sets

Nisan's generator is based on several great ideas. This is what we described so far:

1. The first idea is to use a collection of subsets called designs that do not intersect too much.
2. The second idea is to apply Hastad lower bound on constants depth circuits, and
3. Yao's next bit unpredictability.

What remains to do now is to show that it is actually possible to construct the *designs*. We start by giving a lemma that bounds the size of the sets that can be used as designs.

LEMMA 4.1. *For all n and m integers, such that $\log n \leq m \leq n$ there exists an $n \times l$ 0-1-matrix, where $l = O(m^2)$ and each row represents $S_i \subseteq [l] = \{0, 1, \ldots, l-1\}$, with $\forall i \neq j$ $|S_i \bigcap S_j| \leq \log n$ and $|S_i| = m$.*

We can interpret the lemma in this way. Consider a finite field $\mathbb{F}_m$ of m elements and consider the Cartesian product $\mathbb{F}_m \times \mathbb{F}_m$, and let $l = m^2$. We can think of this Cartesian product as a set of 2D points (x, y) over a finite field. We take n distinct polynomials each of degree at most $\log n - 1$ passing through these points. Two of these polynomials can intersect in at most $\log n$ points, otherwise they would be equal. This allows us to encode them in $n \times l$ matrix. Such matrix has $l = m^2$ columns representing the coordinate of the points of the 2D grid

$(0,0),\ (0,1),\ \ldots\ (0,m{-}1) \mid (1,0),\ (1,1),\ \ldots\ (1,m{-}1) \mid \ldots \mid (m{-}1,0),\ (m{-}1,1),\ \ldots\ (m{-}1,m{-}1)$

Each row of the matrix represents a polynomial. We set 1 if the column corresponds to a point that the polynomial goes through, and 0 otherwise.

5 Nisan-Wigderson Generator

Nisan-Wigderson generator presented in [NW94] is a generalization of Nisan's generator for constant depth circuits. Instead of using parity as the hard function, we can take any f which is assumed to be hard to compute by circuits of a certain size. Hard in this case refers to the inapproximability of f. Based on the hardness assumption of f, the rest of Nisan's proof carries through exactly as above. For example a result using the NW-type argument might go as follows: assume f is inapproximable by poly-sized circuits, then the Psuedorandom generator based on f and the combinatorial designs described above is hard to distinguish by poly sized circuits.

Subsequent papers after [NW94] have a trend of getting away from the assumption of hardness as the inapproximabililty and going back to assumptions about the non-existence of circuit to compute some function. One way this is done is using algebraic extentions of boolean function. Start with boolean function f which is hard to compute correctly and get a multilinear extention of it. It can be shown that if the circuit cannot compute f in a few places, it cannot compute f's extention in many places. The next step is to reinterpret the extention back to a boolean function which is inapproximiable.

6 Definition of Extractor

Now we are going to extend Nisan-Wigderson idea by using some ideas from error correcting to codes. This was presented by Zuckerman in [Z96]. We finish today's lecture giving the definition of extractors which we will need in the following classes, but first let's define the min-entropy

Definition 6.1. *Given a distribution X on $\{0,1\}^n$, the* min-entropy*, $H_\infty(X)$, is defined as*

$$H_\infty(X) = \min_x(-\log_2(\Pr[X = x]))$$

Intuitively $H_\infty(X)$ represents the number of truly random bits contained in X.

Definition 6.2. *An (n, d, k, m, ϵ)-extractor is a function $Ext : \{0,1\}^n \times \{0,1\} \to \{0,1,\}$ for which*

$$||Ext(X, U_d) - U_n|| \leq \epsilon$$

Intuitively m is "how much" of "what there was" k, using "as little as" d, to achieve the approximation ϵ.

References

[CW04] Jin-Yi Cai, Osamu Watanabe, *On Proving Circuit Lower Bounds against the Polynomial-Time Hierarchy*, SIAM J. Comput., 33(4):984–1009, 2004.

[NW94] Noam Nisan, Avi Wigderson, *Hardness vs Randomness*, J. Comput. Syst. Sci., 49(2):149–167, 1994.

[Z96] David Zuckerman, *Simulating BPP Using a General Weak Random Source*, Algorithmica, 16(4/5):367–391, 1996.