1 Introduction

1.1 From Last Lecture

- Presented a BP (branching program) of size $O(n \text{ polylog } n)$ for

  $$\text{EXACT}(x) = \begin{cases} 
1 & \text{if } \sum_{i=1}^{n} x_i = \frac{n}{2} \\
0 & \text{otherwise.}
\end{cases}$$

- Did most of the proof of Alon-Maass theorem: Any $O(1)$-width BP for MAJ
  must have size $\Omega(n \log \log n)$:
    - Communication complexity argument
    - Reduce to oblivious BPs
    - $S, T$ alternations
    - **Key Lemma**: suppose string $z \in [n]^m$ is s.t. (i) $\forall i \in [n], i \in z$, (ii) $\forall S, T \subset [n]$ disjoint, $|S| = |T| = n^{1/3}$, $z$ has $\Omega(\log n)$ alternations w.r.t $S, T$, then $m = \Omega(n \log \log n)$.

1.2 Today’s Topic

- Prove the “Intermediate Lemma” and finish Alon-Maass theorem.

- Prove the **Barrington’s Theorem**: if $f$ can be computed by a Boolean formula of size $s$, then it has a width-5 (we only prove width-8) BP of size poly($s$). (And it’s converse.)
2 Alon-Maass Theorem

Recall the statement of “intermediate lemma”:

**Lemma 1 (“Intermediate Lemma”).** Let \( z \in [n]^{kn} \). Divide \( z \) into \( r \) segments of equal length, each \( z^{(i)} \) has length \( \frac{kn}{r} \).

\[
\begin{array}{cccc}
128379 & 789172 & 238732 & 772382 \\
\rightarrow B_1 & \rightarrow B_1 & \rightarrow B_2 & \rightarrow B_1 \\
\end{array}
\]

Independently uniformly assign \( z^{(i)} \) to \( B_1 \) or \( B_2 \) with probability \( \frac{1}{2} \) each.

Define

- \( \text{unseen}(B_1) = \{ i \in [n] : i \text{ doesn’t occur in any } z^{(i)} \text{ in } B_1 \} \),
- \( X_1 = |\text{unseen}(B_1)| \) and \( \mu = E[|X_i|] \).

Then we have (1) \( \mu \geq \frac{n}{2k} \), (2) \( \Pr[|X_i - \mu| \geq \frac{\mu}{2}] \leq \frac{2^{k+2}k^2}{r} \).

**Proof.** For each \( i \in [n] \), let \( G_i \) be the event that \( i \) is in \( \text{unseen}(B_1) \). Then \( X_i = \sum_{i=1}^n 1[G_i] \) and \( \mu = E[\sum_{i=1}^n 1[G_i]] = \sum_{i=1}^n \Pr[G_i] \).

Let \( s_i \) be the number of segments that \( i \) appears in. \( \Pr[G_i] = \frac{1}{2s_i} \). The total occurrences of \( i \) is at least \( s_i \), we have \( kn \geq \sum_{i=1}^n s_i \). So

\[
\mu = \sum_{i=1}^n \Pr[G_i] = \sum_{i=1}^n \frac{1}{2s_i} \geq n \cdot \frac{1}{2\sum_{i=1}^n s_i/n} \geq \frac{n}{2k}.
\]

The second to last inequality is because of the AM-GM inequality \( x_1 + x_2 + \cdots + x_n \geq (x_1 x_2 \cdots x_n)^{1/n} \).

For (2), we’ll show \( \text{Var}[X_i] \leq \frac{k^2n\mu}{r} \). Given this, by Chebyshev’s inequality we have

\[
\Pr[|X_1 - \mu| \geq \frac{\mu}{2}] \leq \frac{\text{Var}[X_1]}{\frac{\mu^2}{4}} = \frac{4k^2n}{r\mu} \leq \frac{2^{k+2}k^2}{r}
\]
Consider
\[
\begin{align*}
\text{Var}[X_1] &= \mathbb{E}[X_1^2] - \mathbb{E}[X_1]^2 \\
&= \mathbb{E} \left[ \left( \sum_{i=1}^n \mathbb{1}[G_i] \right)^2 \right] - \mathbb{E} \left[ \sum_{i=1}^n \mathbb{1}[G_i] \right]^2 \\
&= \sum_{i,i'} \mathbb{E}[\mathbb{1}[G_i] \cdot \mathbb{1}[G_{i'}]] - \left( \sum_{i=1}^n \mathbb{E}[\mathbb{1}[G_i]] \right)^2 \\
&= \sum_{i,i'} (\mathbb{P}[G_i \land G_{i'}] - \mathbb{E}[\mathbb{1}[G_i]] \cdot \mathbb{E}[\mathbb{1}[G_{i'}]]) \\
&= \sum_{i,i'} (\mathbb{P}[G_i \land G_{i'}] - \mathbb{P}[G_i] \cdot \mathbb{P}[G_{i'}])
\end{align*}
\]

Let’s say \( i \sim i' \) holds if \( i \) and \( i' \) appear together in at least one segment.

Suppose \( i \not\sim i' \) then \( G_i \) and \( G_{i'} \) are independent because they never appear in the same segment, and the contribution of \( (i,i') \) to the sum is 0 since \( \mathbb{P}[G_i \land G_{i'}] = \mathbb{P}[G_i] \cdot \mathbb{P}[G_{i'}] \).

In the case where \( i \sim i' \), \( G_i \) and \( G_{i'} \) may not be independent. But we still have
\[
\mathbb{P}[G_i \land G_{i'}] - \mathbb{P}[G_i] \cdot \mathbb{P}[G_{i'}] \leq \mathbb{P}[G_i]
\]

Plus the fact that, the number of \( i' \) that occurs in at least one segment with \( i \) is no more than \( s_i \cdot \left(\frac{kn}{r}\right) \), we have
\[
\begin{align*}
\text{Var}[X_1] &\leq \sum_{i \sim i'} \mathbb{P}[G_i] \leq \sum_{i=1}^n \mathbb{P}[G_i] \cdot s_i \cdot \left(\frac{kn}{r}\right) \\
&= \frac{kn}{r} \cdot \sum_{i=1}^n s_i \cdot \frac{1}{2^{s_i}} \leq \frac{k}{r} \left( \sum_{i=1}^n s_i \right) \cdot \frac{1}{2^{s_i}} = \frac{k2^n \cdot \mu}{r}
\end{align*}
\]

The (?) inequality follows from the Chebyshev’s sum inequality after we carefully rearrange the sequence of \( s_i \)’s and \( \frac{1}{2^{s_i}} \)’s. ■

**Proposition 2** (Chebyshev’s sum inequality). If we have 2 sequences such that \( a_1 \leq a_2 \leq \cdots \leq a_n \) and \( b_1 \geq b_2 \geq \cdots \geq b_n \), then
\[
\sum_{i=1}^n a_i b_i \leq \left( \sum_{i=1}^n a_i \right) \left( \sum_{i=1}^n b_i \right)
\]
Proof. Consider \( \sum_{i=1}^{n} \sum_{j=1}^{n} (a_i - a_j)(b_i - b_j) \).

For any \( i, j \), \((a_i - a_j)(b_i - b_j) \leq 0\), so
\[
\sum_{i=1}^{n} \sum_{j=1}^{n} (a_i - a_j)(b_i - b_j) \leq 0
\]

\[
\sum_{i=1}^{n} \sum_{j=1}^{n} (a_ib_i - a_ib_j - a_jb_i + a_jb_j) \leq 0
\]

\[
2n \cdot \sum_{i=1}^{n} a_i b_i \leq 2 \sum_{i=1}^{n} \sum_{j=1}^{n} a_ib_j = 2 \left( \sum_{i=1}^{n} a_i \right) \left( \sum_{i=1}^{n} b_i \right)
\]

\[\blacksquare\]

3 Barrington’s Theorem

In this section we’ll show the surprising “equivalence” between constant width branching program and small size Boolean formula. We start by proving the easy direction of Barrington’s Theorem.

**Theorem 3.** If \( f \) has a width-\( w = O(1) \), length-\( s \) branching program, then \( f \) is computed by an \( O(\log s) \)-depth poly(s)-size fan-in-2 Boolean circuit/formula.

**Proof.** The idea behind is “divide and conquer”.

Consider a width-\( w \), length-\( s \) branching program, we can assume WLOG that there’s only one 1-leaf in the last level. Otherwise we can always modify the branching program to be so which only increase the width \( w \) by 1, as the example in the following figure:
Let $d_w(s) = d(s)$ be the maximum depth of Boolean formula needed to compute width-$w$ length-$s$ branching program, i.e. to check that the input $x$ reaches the unique 1-node in the last layer.

Given length-$s$, width-$w$ branching program computing $f$, let $v_1, \ldots, v_w$ be the nodes in layer $s/2$.

We have $f(x) = 1$ if and only if there exists a $v_i$ such that $x$ reaches $v_i$ first and starting from $v_i$, $x$ reaches the 1-sink. That is:

Note that each leaf is a function that is computed by a width-$w$ length-$s/2$ branching program. By induction each of them has a circuit of $O(\log w) \cdot (\log \frac{s}{2}) = O(\log w) \cdot (\log s - 1)$ depth.

Therefore the whole circuit computing $f$ has depth $O(\log w) \cdot \log s = O(\log s)$.

**Theorem 4** (Barrington’s Theorem). If $f$ is computed by a size-$s$ Boolean formula, then $f$ is computed by a poly($s$)-length, width-5 branching program. (We only prove a width of 8.)

**Plan of the proof** we are going to transform a size-$s$ Boolean formula into a equivalent branching program step by step in the following way:

-size-$s$ Boolean formula

\[\Downarrow\text{(Lemma 0)}\]
\(O(\log s)\)-depth Boolean formula 
\[\Downarrow (\text{Lemma 1})\]
\(O(\log s)\)-depth algebraic formula 
\[\Downarrow (\text{Lemma 2})\]
\(\text{poly}(s)\)-length, 3-bit linear bijection straight-line program (LBSLP) 
\[\Downarrow (\text{Lemma 3})\]
\(\text{poly}(s)\)-length, width-8 branching program

Remind that lemma 0 is already obtained in previous classes, that any size-\(s\) Boolean formula can be rebalanced to a \(O(\log s)\)-depth one.

We start the rest of proof by defining algebraic formula: view \(\{0, 1\}\) as in GF\([2]\], “+” gate as the PAR-gate in Boolean formula, “\(\cdot\)” gate as the AND-gate, “1 + \(x\)” as the NOT-gate.

**Definition 5.** An algebraic formula over \(x_1, \ldots, x_n\) is a rooted binary tree. Each internal node has 2 children and labelled with \(\cdot\) or \(+\). Each leaf is labelled with \(x_0\) or 0 or 1.

- *Size of algebraic formula is the number of \(x_i\)-leaves.*
- *Depth of algebraic formula is the depth of the binary tree.*

**Lemma 6** (Lemma 1 in the figure). If \(f\) is computed by size-\(s\), depth-\(d\) Boolean formulas, then \(f\) is computed by size-\(s\), depth-\(O(d)\) algebraic formulas.

**Proof.** Given Boolean formula \(F\) with \(\wedge, \vee, \neg\) gates. We do the following transformation:

\[
(G \vee H) \equiv \overline{G \wedge H} 
\]

Easy to verify that the size remains the same and the depth blows up by a multiplicative factor of at most 3. \(\blacksquare\)
Note that the converse doesn’t hold. One quick example is the parity function. \( \text{PAR}(x_1, \ldots, x_n) \) has a size-\( n \) algebraic formula. But from Khrapchenko lower bound we know that any Boolean formula for PAR has size \( \Omega(n^2) \).

Now we give the definition for linear bijection straight-line program (LBSLP). More precisely, a \( k \)-bit LBSLP is

- Register \( R_1, \ldots, R_k \) each holds 0 or 1.
- A \( k \)-bit LBSLP is a sequence of register offset instructions of the form \( R_j \leftarrow R_j + (R_i \cdot c) \) or \( R_j \leftarrow (R_i \cdot x_u) \) where \( i \neq j, c \in \{0, 1\} \) and \( u \in [n] \).
- Initially \( R_2 = 1, R_i = 0 \) for all \( i \neq 2 \).
- Length of LBSLP is the number of instructions.
- Output of LBSLP is the value ultimately in \( R_1 \) at the end of the program.

The reason to call it “linear bijection straight-line program” is: first it’s clearly a straight-line program. For linear bijection part, at any step, we can view the “state” of the program as the vector \((R_1, \ldots, R_k) \in \text{GF}[2]^k\). Then each instruction is a linear bijection from \( \text{GF}[2]^k \) to itself.

As an example, suppose \( k = 3 \). For instruction \( R_1 \leftarrow R_1 + R_2 \cdot x_1 \), we have

\[
(R_1, R_2, R_3) \begin{pmatrix} 1 & 0 & 0 \\ x_1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (R_1 + R_2 \cdot x_1, R_3, R_3)
\]

where the 3 by 3 matrix is invertible.

**Lemma 7** (Lemma 3 in the figure). If \( f \) is computed by a length-\( l \) \( k \)-bit LBSLP, then \( f \) is computed by a length-\( l \) width-\( 2^k \) branching program.

**Proof.** We construct the branching program as follow: \( j \)-th layer of branching program corresponds to the state of \( k \) registers at step \( j \) of the computation of LBSLP.

For example, say \( k = 3 \) and there are 2 instructions:
Then in the last layer, we label each terminal node according to the value of $R_1$. That is, we accept all the states whose $R_1$ is 1, and reject all others.

**Lemma 8** (Lemma 2 in the figure). If $f$ is computed by a depth-$d$ algebraic formula, then $f$ is computed by a 3-bit LBSLP of length $4^d$.

We’ll prove this final step by induction on $d$ in the next lecture.