1 Introduction

1.1 Last Time

1. Established the exact correspondence between communication protocols $P$ for the monotone relation $M_f$ and monotone formulas $F$ for a Boolean function $f$, so a lower bound on the communication complexity of $M_f$ gives lower bounds on $mdepth(f)$. Specifically looked at the connectivity function $CON : \{0, 1\}^{n \times n} \rightarrow \{0, 1\}$ which takes a 0/1 digraph adjacency matrix and outputs 1 iff $\exists$ a path $1 \sim n$.

2. Showed $D(M_{CON}) \geq D(M) \geq D(\text{FORK})$.

   - $M$ is a restricted $M_{CON}$ over a layered graph with $n$ nodes and $l + 2$ layers of width $w = l + 2 = \sqrt{n}$ such that any edges are between adjacent layers.
   - FORK is the relation defined over $X \times Y \times Z : X = Y = [w]^l, Z = \{0,...,l\}$ as $\{(x, y, i) : x = x_1...x_l, y = y_1...y_l, i$ is such that $x_i = y_i$ and $x_{i+1} \neq y_{i+1}\}$.

3. Showed $D(\text{FORK}) = \Omega(\log l \cdot \log w) = \Omega(\log^2 n)$ through the use of 3 lemmas and proved the first 2.

1.2 Today

1. Proving last/hard lemma from previous lecture’s proof of $m\text{size}(\text{CON}) = n^{\Omega(\log n)}$.

2. Full basis formula bounds.

3. Starting unit on decision trees (DTs) and branching programs.
2 CON lemma proof

The proof of the last lemma uses a combinatorial lemma that states a dense 0/1 matrix will have a sufficient number of semi-dense rows, or one very dense row.

**Lemma 1.** If a 0/1 square matrix $M$ has more than an $\alpha$ fraction of 1s, then either
a) some row has $\geq \sqrt{\frac{\alpha}{2}}$ fraction of 1s, or
b) at least $\sqrt{\frac{\alpha}{2}}$ fraction of rows each have $\geq \frac{\alpha}{2}$ fraction of 1s.

Recall the definition of an $(\alpha, l)$-protocol for FORK introduced last lecture:

**Definition 2.** For $0 \leq \alpha \leq 1$ and $l \geq 1$, a protocol $P$ is an $(\alpha, l)$-protocol for FORK if $\exists S \subseteq [w]^l, |S| \geq \alpha \cdot w^l$ such that $P$ succeeds on $(x, y) \forall x, y \in S$.

We will view each element of $[w]^l$ as a concatenation of two halves $u$ and $v$, which index an element in a $[w]^{l/2} \times [w]^{l/2}$ 0/1 matrix $M$. An entry $M_{uv} = 1$ iff $uv \in S$. At a high level, the proof consists of constructing a good protocol for suffixes $v$ in case a) of Lemma 1 and a good protocol for prefixes $u$ in case b).

**Lemma 3** (Hard Lemma). Suppose $\alpha \geq \frac{12}{w}$. If we have a $c$-bit $(\alpha, l)$-protocol for FORK, then we also have a $c$-bit $(\sqrt{\frac{\alpha}{2}}, \frac{l}{2})$-protocol.

**Proof.** Let $P$ be the $(\alpha, l)$-protocol, $S$ be the set it works on, and $|S| \geq \alpha \cdot w^l$, so the corresponding 0/1 matrix $M$ is $\alpha$-dense.

- Suppose case a) of Lemma 1 holds for $M$.
  
  $\exists$ row $u$ such that $\geq \sqrt{\frac{\alpha}{2}}$ fraction of its entries are 1. Then for $\geq \sqrt{\frac{\alpha}{2}}$ fraction of $v \in [w]^{l/2}$, $P$ works on $uv$.
  
  We can construct a protocol $P'$ with inputs $x, y \in [w]^{l/2}$. Alice and Bob follow the original $P$ on the strings $ux$ and $uy$, which outputs $i$ such that $ux_i = uy_i$ and $ux_{i+1} \neq uy_{i+1}$.
  
  Any output $i$ to this instance of FORK is necessarily in the strings $x$ and $y$ since the first half is identical. $P$ works on at least a $\sqrt{\frac{\alpha}{2}}$ fraction of length $l$ inputs with prefix $u$, so $P'$ is an $(\sqrt{\frac{\alpha}{2}}, \frac{l}{2})$-protocol.

- Suppose case b) of Lemma 1 holds for $M$.
  
  Let $S' \subseteq [w]^{l/2}$ be the set of rows (prefixes) that are $\frac{\alpha}{2}$-dense. Then $|S'| \geq \sqrt{\frac{\alpha}{2}} \cdot w^{l/2}$.
  
  Our intuitive goal is for Alice and Bob to expand each input from $S'$ so that running $P$ on the expanded string gives a valid FORK output on the original input string. For each $x \in S'$, we want to show $\exists$ two suffix completions $a(x), b(x)$ such that
1. $xa(x), xb(x) \in S$. Then running $P$ on $xa(x)$ and $xb(x)$ gives a valid output for FORK.

2. $\forall x, y \in S'$ such that $x \neq y$, $a(x)_i \neq b(y)_i$. If $a(x)$ and $b(y)$ disagree for all positions, then the FORK answer must be in $x$ and $y$.

This would give a $(\sqrt{\frac{\alpha}{2}}, \frac{1}{2})$-protocol, which is stronger than the originally stated $\sqrt{\frac{\alpha}{2}}$ by a factor of $\approx 1.4$. We can’t guarantee 1 and 2 for every element in $S'$, so instead we’ll probabilistically show that they hold for $\geq \frac{9}{10} > \frac{1}{\sqrt{2}} \approx 0.7$ fraction of elements in $S'$, which is enough for the promised $(\sqrt{\frac{\alpha}{2}}, \frac{1}{2})$-protocol. More formally, we show $\exists S'' \subseteq S', |S''| \geq \frac{9}{10}|S'|$ such that criteria 1 and 2 hold for $S''$.

We can view a suffix as a path through a layered graph with $w \cdot l/2$ nodes and edges between all nodes in adjacent layers:

![Figure 1: This path would correspond to a string 142...3 \in [w]^{l/2}](image)

To build Alice’s $a(x)$ and Bob’s $b(x)$ so that they disagree at each position, we color half of the nodes in each column red, and the other half blue using the following method:

1. Pick $w/2$ paths col 1 $\sim$ col $l/2$, uniformly and independently at random and color all the nodes red.

---

1For each $\alpha/2$-dense row $u$ of $M$, we have two columns $v_1, v_2$ such that $M_{uv_1} = M_{uv_2} = 1$

2For two distinct $\alpha/2$-dense rows $u_1, u_2$, the suffix given by $a(u_1)$ disagrees on every position with the suffix given by $b(u_2)$

3This has the same effect as directly choosing $w/2$ random red nodes per layer and making the rest blue, but picking paths is useful in the analysis.
2. If a layer has $< w/2$ red nodes, randomly color nodes red so that there are exactly $w/2$ per column.

3. Color the rest blue.

By restricting $a(x)$ to red nodes and $b(x)$ to blue nodes, they will never have the same value in the same position. Thus $\forall x, y \in S', a(x)$ will disagree with $b(y)$ and satisfy criterion 2.

Fix an $x \in S'$. Then $\geq \frac{\alpha}{2}$ fraction of all suffixes $v$ are valid, or satisfy $xv \in S$. Since each of the paths $p_1, ..., p_{w/2}$ is a random suffix, $\forall i \in [w/2]$, $\Pr[p_i \notin S] \leq (1 - \frac{\alpha}{2})$. Then the probability that none of the $w/2$ paths is a valid suffix is:

$$\prod_{i=1}^{w/2} \Pr[p_i \notin S] \leq (1 - \frac{\alpha}{2})^{w/2} \leq \frac{1}{e^3} < 0.05$$

The second inequality comes from the assumption $\alpha \geq \frac{12}{w}$. We get the probability $\Pr[x$ has no valid red suffix$] \leq 0.05$ and the same for blue. By the union bound $\Pr[x$ has no valid red or blue suffix$] \leq \frac{1}{10}$, so

$$\Pr[x$ has valid red and blue suffix$] \geq \frac{9}{10}$$

Which gives us $\mathbb{E}[\#x$ with valid red and blue suffix$] \geq \frac{9}{10} \cdot |S'|$. An $x$ with valid suffixes $a(x)$ and $b(x)$ also satisfies criterion 1, so $x \in S''$ and

$$|S''| \geq \frac{9}{10} \cdot |S'|$$

We now have a $(\frac{9}{10} \sqrt{\frac{\alpha}{2}}, \frac{1}{2})$-protocol, which is a $(\frac{\sqrt{\alpha}}{2}, \frac{1}{2})$-protocol.

\[\blacksquare\]

3 Full basis formulas

3.1 Intro

Up to now, we have been working with Boolean formulas in the deMorgan basis of the $\lor, \land, \neg$ operations. Formulas over the full binary basis allow any binary gate $\{0, 1\}^2 \rightarrow \{0, 1\}$. We’ll show some examples of lower bounds on full basis formulas for several functions.
Claim 4. Given full basis formula with \( s \) variable leaves, \( \exists \) an equivalent formula with \( s \) variable leaves using only XOR (\( \oplus \)) and AND (\( \land \)) gates.

Proof. We can get all binary functions with two variable leaves.

- Negating variables: \( \overline{x_i} \equiv x_i \oplus 1 \).
- Gates where two (out of four) of the input combinations map to 1s are either a single variable or the XOR function (or their negations).
- Gates where one (or three, by way of negation) of the input combinations map to 1 can be constructed:

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( \overline{x_1} \land \overline{x_2} )</th>
<th>( \overline{x_1} \land x_2 )</th>
<th>( x_1 \land \overline{x_2} )</th>
<th>( x_1 \land x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Allowing all binary gates breaks some lower bounds, like for PAR:

Example 5. Khrapchenko’s lower bound for deMorgan basis PAR, \( \Omega(n^2) \), does not hold for full basis formulas. Full basis size(PAR) = \( n \), since PAR = \( x_1 \oplus ... \oplus x_n \).

But the non-explicit counting argument for Shannon’s lower bound on most Boolean functions from the first lecture still applies:

Observation 6. The size of most \( n \)-variable full basis formulas is \( \Omega \left( \frac{2^n}{\log n} \right) \).

3.2 Lower bound for full basis formulas

The best known lower bound for explicit functions was found by Nečiporuk in 1966. (We’ll be proving a looser version with an Andre’ev-like method.)

Theorem 7 (Nečiporuk ’66). A full basis formula for an explicit function over \( n \) variables has size \( \Omega \left( \frac{n^2}{\log n} \right) \).

Let \( b = \log n \), and \( m = \frac{n}{b} = \frac{n}{\log n} \). Let \( f_y \) be a (\( n \))-variable function with truth table \( y_1...y_n \). Define a (\( 2n \))-variable function \( A \):

\[
A(x_1, ..., x_n, y_1, ..., y_n) = f_y(x_1 \land ... \land x_m, ..., x_{n-m+1} \land ... \land x_n)
\]

The proof will show a lower bound on the size of formulas for \( A \).
Theorem 8. Any full basis formula for $A$ must have size $\Omega\left(\frac{n^2}{\log n \log \log n}\right)$.

Proof. Fix $F$ to be a minimal size full basis formula for $A$, and let $s$ be the number of gates in $F$. Split the variables into $\log n$ blocks: $B_1 = x_1x_2...x_m$, $B_2 = x_{m+1}...x_{2m}$, up to $B_{\log n} = x_{m \log n - m + 1}...x_m \log n$.

Select the least frequently occurring variable in each block (w.l.o.g say $x_m$ in $B_1$, $x_{2m}$ in $B_2$, etc.) and let restriction $\rho$ be to fix all other variables equal to 1. This causes each block $B_i$ to collapse to $x_{im}$, so $A \upharpoonright \rho$ is equivalent to $f_y(x_m, x_{2m}, ..., x_n)$.

We can pick $y$ over $\{0, 1\}^n$ so that $f_y$ is any ($\log n$)-variable function. By Shannon’s theorem, most $f_y$ will require size $\Omega\left(\frac{n}{\log \log n}\right)$. Then $F$ will have $\geq \Omega\left(\frac{n}{\log \log n}\right)$ occurrences of the variables $x_m, x_{2m}, ..., x_{bm}$. And the total number of occurrences of $x_1, ..., x_n$ in $F$ must be $\geq m \cdot \Omega\left(\frac{n}{\log \log n}\right) = \Omega\left(\frac{n^2}{\log n \log \log n}\right)$. ■

3.3 Other bounds for full basis formulas

- Best known l.b. for MAJ: $\Omega(n \log n)$ [FMP82]
- Any explicit function: $\Omega(n \log^2 n)$ [BNS92]

4 Decision Trees

4.1 Intro

Some good reference reading for this unit is in the Jukna text Sections 14.1, 14.2, 14.9 and a survey on decision tree complexity by Buhrman and de Wolf. [BdW02]

Definition 9. A decision tree (DT) is a rooted binary tree with a variable at each internal node and a bit at each leaf.

The decision tree represents a Boolean function $f : \{0, 1\}^n \to \{-1, 1\}$.

On each node $x_i$

$$\begin{cases} \text{go left if bit } x_i = 0 \\ \text{go right if bit } x_i = 1 \end{cases}$$

For the decision tree in Figure 2, an input string 110110110 would output 1. Intuitively, they capture how much of an input you need to read to know what the output is.

We’ll use two measures of decision trees: the depth, counted as the maximum length of a root $\leadsto$ leaf path, and the size, counted as the number of leaves.

Definition 10. $\text{DTdepth}(f)$: depth of the shallowest tree for $f$. 

Definition 11. $DTsize(f)$: size of the smallest tree for $f$.

The DTdepth is more widely studied as the number of nodes on a given input’s root to leaf path corresponds to the number of variables you need to look at to get an output. The maximum path length of the shallowest tree is the worst case query complexity, or number of variables that need to be queried on any given input.

Observation 12. The parity function has $DTdepth(PAR) = n$ and $DTsize(PAR) = 2^n$, since no leaf can be shallower than depth $n$.

In fact, PAR and $\overline{PAR}$ are the only $n$ variable functions with $DTsize = 2^n$. Any other function $f$ will have some edge in the Boolean hypercube $x \leftrightarrow x^{(i)}$ such that $f(x) = b = f(x^{(i)})$, so it is possible to construct a DT for $f$ where the depth of at least one leaf is $\leq n - 1$.

Observation 13. Every $f$: $\{0, 1\}^n \rightarrow \{-1, 1\}$ has $DTdepth(f) \leq n$, $DTsize(f) \leq 2^n$.

Figure 3 shows the beginning of an optimal DT for PAR. A DT for PAR needs $2^{i-1}$ nodes in any layer $i$. The smallest DT for PAR$(x_1)$ has a single root node $x_1$ with 2 leaves (0 and 1). Since the output depends on reading every bit, no matter what value PAR$(x_1...x_i)$ outputs, PAR$(x_1,...,x_i,x_{i+1})$ depends on $x_{i+1}$, so each of the $2^i$ leaves in the DT for $i$ vars becomes an $x_{i+1}$ node with 2 leaves (0 and 1) in the DT for $i+1$ vars. By the Boolean hypercube argument, any other function can have a DT constructed with fewer leaves, and at most that many layers.
4 DECISION TREES

4.2 Evasive functions

Definition 14. If \( f: \{0, 1\}^n \rightarrow \{-1, 1\} \) has DTdepth\( (f) = n \), we say \( f \) is evasive.

The selection function \( \text{SEL}: \{0, 1\}^3 \rightarrow \{0, 1\} \) is non evasive. The first bit decides whether to output bit 2 or 3. See Fig 4.

![Figure 4: DT for SEL](image)

The and function \( \text{AND}(x_1, \ldots, x_n) \) is evasive, since all bits must be 1 (and therefore seen) in order to output 1. Then DTdepth(AND) = \( n \).

Claim 15. If \( f \) is computed by a read once Boolean formula (de Morgan basis formula of size \( n \) where each \( x_i \) occurs exactly once), then \( f \) is evasive.

Proof. By adversary and induction.

Given any read once formula \( F \), we construct an input string such that all \( n \) vars must be seen to know the output of \( f \). A 1 variable read once formula needs a DTdepth of 1. Assume an \( n - 1 \) variable formula requires a DT of depth \( n - 1 \). Let \( T \) be the DT for an \( n \) variable function \( f \) and assume it queries \( x_i \) at the root. Look at the parent gate of \( x_i \) in \( F \).
4 DECISION TREES

- Parent is $\lor$ gate: set $x_i = 0$.
- Parent is $\land$ gate: set $x_i = 1$.

Setting $x_i$ as above means the output of the parent gate depends on the other child in $F$ - i.e. the other branch can’t be eliminated, which leaves an $n - 1$ variable formula with required DTdepth $n - 1$. Thus T requires DTdepth $n$, completing the induction. ■

4.3 Non-deterministic decision trees

We have two models of non-deterministic DTs:

**Definition 16.** A Boolean formula in disjunctive normal form is an OR of terms, which are ANDs of literals. A Boolean formula in conjunctive normal form is an AND of clauses, which are ORs of literals. We’ll refer to them as DNFs and CNFs.

**Definition 17.** The size of a DNF/CNF is the number of terms/clauses. The width of a DNF/CNF is the maximum number of literals in a term/clause.

Given a decision tree for $f$, each 1-leaf corresponds to a term in a DNF for $f$, and each 0-leaf corresponds to a clause in a CNF for $f$. A DNF for $f$ will output 1 if any of the clauses are satisfied (a 1 leaf is reached). A CNF for $f$ will output 0 if any of the clauses are unsatisfied (a 0 leaf is reached). See examples in Figure 5.

\[
\text{DTsize}(f) = s \rightarrow f \text{ has an } s\text{-term DNF, } s\text{-clause CNF}
\]

\[
\text{DTdepth}(f) = w \rightarrow f \text{ has a width-}w \text{ DNF, CNF}
\]

![Figure 5: DT leaf to term/clause correspondence](image)
Observation 18. We can negate a DT $T$ by flipping all the output bits to get $\overline{T}$ for $\overline{f}$, such that the 1-leaves of $\overline{T}$ correspond to terms in a DNF for $\overline{f}$, and to clauses in a CNF for $f$.

Observation 19. A width-$w$ DNF is a non-deterministic analogue to a depth-$w$ decision tree, which is a deterministic model for a function $f$. The CNF is a co-non-deterministic model for $f$.

Guessing a path to a leaf in a DT is equivalent to picking a single term in the DNF. If $\exists$ a guess that reaches a 1, then we output 1. If $\nexists$ a guess that reaches 1, then we output 0. Thus for function $f$,

$$\text{DTdepth}(f) = w \leftrightarrow \text{deterministic complexity of } f \text{ is } w$$
$$\text{DNF width}(f) = w \leftrightarrow \text{non-deterministic complexity of } f \text{ is } w$$

References

