COMS W3261, Section 1, Spring 2024
Communication Complexity

Instructor : Josh Alman
Notes by : William Pires

1 Communication Complexity

We have two persons Alice and Bob. Alice has a string $x \in \{0,1\}^n$ while Bob has $y \in \{0,1\}^n$. Alice has no information about y and Bob no information about x. Their goal is to compute a function $f(x,y)$, for some function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ [1].

We will be interested in how much Alice and Bob need to *communicate* with each other to compute $f(x,y)$. Let's start with some important examples for f :

Definition 1. The equality function $\mathsf{EQUALITY} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ is defined as :

$$\mathsf{EQUALITY}(x,y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ otherwise} \end{cases}$$

Why is $\mathsf{EQUALITY}$ interesting ? Say you there is huge database (stored by Alice), and a backup copy of it (stored by Bob). The goal is to check if the content of the databases are the same. But how much information do we need to transmit to perform this check ? Here we can assume the main bottleneck is the communication from one party to the other, especially since this might be very slow, and there's a lot of data involved.

Definition 2. The majority function $\mathsf{MAJORITY} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ is defined as :

$$\mathsf{MAJORITY}(x,y) = \begin{cases} 1 \text{ if } xy \text{ has more 1s than 0s} \\ 0 \text{ otherwise} \end{cases}$$

Say we have an election with two voting stations across town. Both stations collected their votes for candidates 0 and 1. So, you can view the votes at the first station as $x \in \{0,1\}^*$, while at the second station it's $y \in \{0,1\}^*$. To know who won the elections, we can compute $\mathsf{Majority}(x,y)$. Do we have to send the list of all the votes from one location to the other ? Or can we do better ?

Definition 3. The majority function $\mathsf{PARITY} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ defined as :

$$\mathsf{PARITY}(x,y) = \begin{cases} 1 \text{ if the number of 1s in } xy \text{ is odd} \\ 0 \text{ otherwise} \end{cases}$$

Let's now introduce the formal notation needed to model this task. *Note that for us, Alice and Bob are always given strings of the same length.*

[1] Note that we don't need f's output to just be binary, but that's what we will focus on.

Definition 4 (Communication protocol)**.** A communication protocol P is specified by pair of functions:

$$A, B : \{0,1\}^* \times \{0,1\}^* \to \{0,1,\mathsf{STOP}\}.$$

The protocol keeps track of a transcript t, which start as $t := \epsilon$. Whenever Alice or Bob sends a message $b \in \{0,1,\mathsf{STOP}\}$ to the other, we set $t = t \circ b$.

We view the protocol as a series of round, starting from $r = 1$. When r is odd, Alice sends $A(x,t)$ to Bob. When r is even turns Bob sends $B(y,t)$ to Alice. When either party sends STOP, the protocol stops, and the output is whatever bit that was sent last.

In particular, the first part of the input of A is Alice's input x, while the first part of B's input is Bob's input y. The second part is the transcript t.

Since the definition might be a bit odd here's how a protocol works.

```
• Set the transcript $t = \epsilon$.
• Set the round number $r = 1$
• While $t$ doesn't end with $\mathsf{STOP}$ :
    – If $r$ is odd : Set $t = t \circ A(x,t)$.
    – Else if $r$ is even : Set $t = t \circ B(y,t)$.
    – Set $r = r + 1$. [a]
• Output the last non-$\mathsf{STOP}$ bit of $t$.
```

[a]This corresponds to Alice sending a message to Bob on odd turns and Bob to Alice on even turns. And then going to the next round.

Note that on every pair $(x,y) \in \{0,1\}^n \times \{0,1\}^n$, the protocol outputs a bit $b \in \{0,1\}$. So we can view a protocol as function !

Definition 5 (Function computed by P)**.** Any protocol P computes function $f_P : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$.

Where the functions is defined as

$$f_P(x,y) := \{\text{The output bit of } P \text{ when Alice is given } x \text{ and Bob is given } y\}$$

We also need to understand how much communication a protocol takes.

Definition 6 (Cost of a protocol)**.** Let P be a communication protocol. For a pair $(x, y) \in \{0,1\}^n \times \{0,1\}^n$, the cost of P on the pair (x, y) is the length of the transcript t when P stops [2]if Alice is given x and Bob is given y.

The cost of P, denoted $\mathsf{cost}(P)$ is a function defined as :

$$\mathsf{cost}(P)(n) := \max_{x,y \in \{0,1\}^n} \text{the cost of } P \text{ on } (x, y)$$

Note that Alice and Bob can compute any function of their inputs and the messages they received, so A, B can be *any function you want*, no matter how much space / time it would take to compute them. We only care about the number of bits they send each other.

Finally, we can consider the cost of computing a function f in the communication model. That is, we want to know what is the "best" protocol P that computes f when the inputs have length n.

Definition 7 (Communication complexity of a function)**.** We say that a protocol P computes a function f is for all $(x, y) \in \{0,1\}^n \times \{0,1\}^n$ we have $P_f(x, y) = f(x, y)$. The communication complexity of a f, denoted $\mathsf{cc}(f)$ is a function defined as :

$$\mathsf{cc}(f)(n) := \min_{P \text{ that compute } f} \mathsf{cost}(P)(n)$$

2 Examples

Let's do examples by giving protocols for the functions mentioned above. Note that when giving protocols, we don't use the formal definition as it would be cumbersome. You can give protocols like these ones for the homework and exams.

You can turn these "informal" protocols into formal one by increasing the communication complexity by a constant factor.

Example 1. A protocol to compute $\mathsf{EQUALITY}$:

[2]That is the first time one of the party sends STOP, and we output the answer. We usually don't count the STOP in the length of the transcript, but this only changes the length by ± 1, so it doesn't really matter.

- Alice sends to Bob her input x.
- Bob accepts (outputs 1) if $x = y$.
- Otherwise Bob rejects (outputs 0).

Alice sends n bits to Bob when $x \in \{0,1\}^n$. Thus $\mathsf{cc}(\mathsf{EQUALITY})(n) = \mathcal{O}(n)$. As an example, you can turn this into a "formal" protocol as follow :

- For $i = 1$ to n :
  - Alice sends to Bob the ith bit x.
  - If $i < n$ Bob replies with 1.
  - Otherwise Bob replies with 1 if $x = y$ (now that he has received all the bits of x) and 0 otherwise.
- Alice outputs STOP

You can see this only increases the communication by a constant factor. So that's why it's good enough to use "informal" protocols. You might wonder if one can do better to compute EQUALITY. We will see that for this function this is essentially the best you can do.

Example 2. A protocol to compute MAJORITY :

- Alice sends to Bob the number of 1s in x.
- Bob accepts (outputs 1) if his number of 1s + Alice's number of 1's is at least $n + 1$[a].
- Otherwise Bob rejects (outputs 0).

[a]Since the string $|xy|$ has length $2n$

It takes Alice $\lceil log_2(n+1) \rceil$ many bits to send the number of 1s in x, since this number is between 0 and n. So $\mathsf{cc}(\mathsf{MAJORITY})(n) = O(\log(n))$.

Example 3. A protocol to compute PARITY :

- Alice sends to Bob the of 1s in x mod 2.
- Bob accepts (outputs 1) if (#1s in x mod 2 + #1s in y) is odd.
- Otherwise Bob rejects (outputs 0).

Alice sends a single bit to *Bob*. So $\mathsf{cc}(\mathsf{PARITY})(n) = O(1)$.

Note that you can't get a protocol sending 0 bits to compute PARITY. Why ? If the protocol had 0 communication, Bob and Alice wouldn't communicate at all. So, Alice would know $\mathsf{PARITY}(x, y)$ without knowing anything about y. But this clearly isn't possible, since the answer depends on y [3]. And the same applies for Bob, who can't know the answer without receiving any information.

[3]If you fix x, there's some y_1 such that $\mathsf{PARITY}(x, y_1) = 1$ but there also exists y_2 with $\mathsf{PARITY}(x, y_2) = 0$

3 Lower bounds

We will now be interested in giving lower bounds for $\mathsf{cc}(f)$. To prove lower bounds, we will use something similar to length-n distinguishing sets from streaming algorithms. I promise it's the last definitions. I deviate from what was said in class here, this is a way to generalize the lower bound that was shown for EQUALITY. You don't don't have to use these definitions in the homework. However, using them can make your communication lower bound proofs much shorter.

Definition 8 (Length-n fooling inputs)**.** Fix a function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. Two different pairs (a, b), $(c, d) \in \{0,1\}^n$ are fooling if :

- $f(a, b) = f(c, d) = o$.
- $f(a, d) \neq o$ or $f(c, b) \neq o$.

The main point is that all both pairs $(a, b), (a, d)$ are mapped to same output by f but if you swap the element in a pair for an element from a different pair this changes the output of the function ! We will later show that if P sends same transcript t on (a, b) and (c, d) it must also send t on (a, d) and (c, b).

Definition 9 (Length-n fooling set)**.** A length-n fooling set set for a function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ is a set $S_n \subseteq \{0,1\}^n \times \{0,1\}^n$ such that any two distinct $(a, b), (c, d)$ in S_n are length-n fooling.

In particular there exists $o \in \{0,1\}$ such that $f(a, b) = o$ for all $(a, b) \in S_n$.

All the pairs in length-n fooling set must be mapped to the same output since $f(a, b) = f(c, d)$ for all the pairs in the set. Here's why we care about fooling sets :

Theorem 1. If f has a length-n fooling set S_n, then any protocol P that computes f must have $\mathsf{cost}(P)(n) = \Omega(\log(|S_n|))$. Which implies $\mathsf{cc}(f)(n) = \Omega(\log(|S_n|))$.

So, to show that $\mathsf{cc}(f)(n) = \Omega(k)$, it suffices to give a length-n fooling set for f of size $2^{\Omega(k)}$. Josh didn't have time to prove this last lecture, but he gave the idea in the example for EQUALITY. So let's do the formal proof for a generic function, which as you'll see is very similar.

Proof. Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be a function with a fooling set S_n of size 2^k. Assume for contradiction that there a protocol P for f and with cost $\mathsf{cost}(P)(n) < \log_2(2^k) = k$.

Let's count how many transcript are possible on inputs of length n. The cost of the protocol is $< \log_2(|S_n|)$. So the transcript t has length at most $k-1$, and there is

$$\sum_{i=0}^{k-1} 2^i = 2^{k-1+1} - 1 = 2^k - 1$$

different strings of length at most $k-1$.

There are 2^k many inputs in S_n, so by the pigeonhole principle [4], there are two *different* pairs in S_n on which the same transcript t is sent when using P.

Let $(a,b),(c,d)$ be two inputs that give the same transcript t. Since (a,b) and (c,d) are fooling, we know $f(a,b) \neq f(a,d)$ or $f(a,b) \neq f(c,a)$. Wlog, let's assume $f(a,b) \neq f(a,d)$. Our goal will be to show that Alice and Bob also send t on input (a,d).

Assuming this is true, the transcript t is sent on both (a,b) and (a,d) so the protocol outputs the same bit on both pairs ! But this can't correctly compute f for on one of them. This gives us our contradiction. So, any protocol P that computes f must have $\mathsf{cost}(P)(n) \geq \log_2(2^k) = k$.

So let's prove the claim, we will proceed by induction. We will show that at each round of the protocol, Alice and Bob send the same bit on (a,d) as they would on (a,b).

- Let t_i be the the transcript at the beginning of round i when the input is (a,b).
- Base case $r = 1$. At this point $t_1 = \epsilon$. So on input (a,b) and (a,d) Alice sends $A(a,\epsilon)$. So the same bit is sent.
- Inductive step. Assume that for round 1 to i, Alice and Bob sent the same bits (a,d) as they would on (a,b), that is the transcript so far on (a,d) is also t_i. Wlog, let's asume it's Bob's turn to speak.

  We know Alice and Bob send the same transcript t on input (a,b) and (c,d). So at round i Bob must send the same bit on both pairs, thus we must have $B(b,t_i) = B(d,t_i)$. So we know $t_{i+1} = t_i \circ B(d,t_i)$

  By the induction hypothesis we know that the transcript at the beginning of the ith round, when the input is (a,d) is also t_i. Now Bob will send $B(d,t_i)$, so the transcript becomes $t_i \circ B(d,t_i)$. So at the round $i+1$, the transcript sent on (a,d) is t_{i+1} too. This proves the inductive step.
- Conclusion : By the end of the protocol, the transcript of the communication between Alice and Bob on input (a,d) is the same as it is on input (a,b). So this proves our claim.

$\square$

In particular, we proved something a bit stronger than what the theorem says. If we have a length-n fooling set of size 2^k for a function f, we have $\mathsf{cc}(f)(n) \geq k$. Finally, we show can prove the $\mathsf{EQUALITY}$ lower bound, using length-n fooling sets this will be easy.

[4] There are 2^k pairs in S_n (pigeons) and 2^{k-1} transcripts in P (holes) for inputs of length n.

Theorem 2.

$$\mathsf{cc}(\mathsf{EQUALITY})(n) = \Theta(n).$$

Proof. We already gave a protocol for EQUALITY with cost $O(n)$ which shows $\mathsf{cc}(\mathsf{EQUALITY})(n) = O(n)$. So we need to show that $\mathsf{cc}(\mathsf{EQUALITY})(n) = \Omega(n)$.

By Theorem 1, it's enough to give a length-n fooling set of size 2^n for EQUALITY.

Consider the set $S_n := \{(a, a) \mid a \in \{0, 1\}^n \}$ which has size 2^n.

We claim this is a valid fooling set. First note that all the pairs in S_n are of the form (a, a) so we always have $\mathsf{EQUALITY}(a, a) = 1$. Now consider two distinct pairs (a, a) and (b, b) in S_n. Since $a \neq b$, we have $\mathsf{EQUALITY}(a, b) = 0$.

Since S_n is a length n-fooling set for EQUALITY, we get that $\mathsf{cc}(\mathsf{EQUALITY})(n) \geq n$ □

More generally here's how to prove a communication complexity lower bound. Again, **this wasn't mentioned in class, so you don't have to do things like this.** But this might help you have a very short proof.

Proof Template 1 (Show that $\mathsf{cc}(f)(n) = \Omega(k)$)**.**

1. Give a set $S_n \subseteq \{0, 1\}^n \times \{0, 1\}^n$, of size $2^{\Omega(k)}$ [a].
2. Show that there exists $o \in \{0, 1\}$ with $f(a, b) = o$ for all pairs in S_n. That is, all pairs are mapped to the same output.
3. For any distinct pairs (a, b) and (c, d) in S_n :
   - $f(a, d) \neq o$ or $f(c, b) \neq o$.

[a] A set of size $2^{c \cdot k}$, for any constant c is good enough.

Here's the example for MAJORITY :

Theorem 3.

$$\mathsf{cc}(\mathsf{MAJORITY})(n) = \Theta(\log(n)).$$

Proof. We already gave a protocol for MAJORITY with cost $O(\log(n))$ which shows $\mathsf{cc}(\mathsf{MAJORITY})(n) = O(\log(n))$. So, we need to show that $\mathsf{cc}(\mathsf{MAJORITY}) = \Omega(\log(n))$.

By Theorem 1, it's enough to give a length-n fooling set of size n for MAJORITY.

Consider the following set of size n

$$S_n := \{(a, b) \mid a = 1^k 0^{n-k} \text{ and } b = 1^{n-k+1} 0^{k-1} \ \text{ where } 1 \leq k \leq n\}.$$

We claim this is a valid fooling set. First, for a pair $(a, b) \in S_n$, a will have k many 1s and b will have $n - k + 1$ many 1s, so the string ab has $n + 1$ many 1s. Thus, $\mathsf{MAJORITY}(a, b) = 1$.

Now consider two distinct pairs (a, b) and (c, d) in S_n. Say that $a = 1^k 0^{n-k}$ and $c = 1^k 0^{n-\ell}$ (so $d = 1^{n-\ell+1} 0^{\ell-1}$) where $\ell \neq k$. Wlog, we can assume $k < \ell$. We have that

$$ad = 1^k 0^{n-k} 1^{n-\ell+1} 0^{\ell-1}.$$

The string ad has $k + n - \ell + 1$ many 1, but $k - \ell < 0$, so $k - \ell + n + 1 < 0$. Thus, ad doesn't have more than half its bits being 1 and we can conclude $\mathsf{MAJORITY}(a, d) = 0$.

Since S_n is a length n-fooling set for $\mathsf{MAJORITY}$ of size $\Omega(n)$, we get that $\mathsf{cc}(\mathsf{EQUALITY})(n) = \Omega(\log(n))$ $\square$