3 Section Week 3 - Communication Complexity

This lecture assumes knowledge of the probabilistic method for only one proof.

3.1 The Computational Model

In communication complexity, we have multiple (two) players, who we'll call Alice and Bob. They are trying to compute some function $f : X \times Y \to Z$, where for the purposes of this lecture $X, Y = \{0,1\}^n$ and $Z = \{0,1\}$. Alice gets an input $x \in X$, Bob gets an input $y \in Y$, and they try to compute $f(x,y)$ by sending bits.

Alice and Bob are fully collaborative, so they can plan things out beforehand and fully trust each other and stuff.

We would like to know how Alice and Bob can compute their function while sending as few bits as possible. For now we only talk about deterministic communication complexity. We also give Alice and Bob unbounded computational power - we care about hardness due to lack of full information, not hardness due to the problem itself.

Example - computing equality

Deterministic complexity is $n + 1$

Example - computing parity

Deterministic complexity is 2

Example - computing median

Deterministic complexity is $\log^2 n$

3.2 Trees and Rectangles

Draw a communication tree

A protocol Π for a function is a tree where at each step somebody sends a bit, and depending on which bit is sent we move down the protocol tree. The protocol is correct (in the deterministic setting) if for every x, y we reach a leaf that's labelled with the correct output.

The cost $cost(x, y)$ for computing f on input (x, y) is simply the length of the path in the tree given inputs x and y. The cost of the entire tree is just the length of the longest path, corresponding to the worst case number of bits we need to send to complete the protocol on some worst case inputs.

Interpreting the communication protocol as a matrix

The rows and columns are labelled by the inputs to the function, and the values are labelled 0 or 1 corresponding to the function evaluated on the inputs.

A combinatorial rectangle on a matrix representing $X \times Y$ is just $J \times K$, where $J \subseteq X$ and $K \subseteq Y$.

Intuition: In Alice's first bit, she will either send a 0 or 1, depending on her input. Thus, we can partition X by which inputs will cause her to send a 0 or 1. Then, Bob responds with either a 0 or a 1, and this depends on Alice's first bit and his input. The partitions of Y conditioned on Alice's bit may be different, but in either case we end up with rectangles. And so on.

Draw rectangles with regions labelled by the bits sent so far

Some nice intuition - the rectangles are all unchanged up to permutations of the row labels and column labels, since subsets don't care about order.

How do we connect the protocols to the communication matrix?

Theorem. *A communication protocol that sends c bits partitions the input matrix into at most 2^c monochromatic combinatorial rectangles.*

Proof. (Informal)

Each bit that's sent at most increases the number of rectangles by a factor of 2, since at most we can cut every existing rectangle into two. Thus, we have at most 2^c rectangles.

When we color each rectangle with either a 0 or 1, we have at most 2^c monochromatic rectangles. We can have less than 2^c if some rectangles merge (for example, if for some reason we color every rectangle with all 0s, then the entire matrix is all 0s).

□

Importantly, just because a c bit protocol gives a 2^c cover doesn't mean every 2^c cover corresponds to a valid c bit protocol.

This result with the communication matrix lets us reason about some functions. For example, what does the communication matrix for equality look like? (it's just $2^n \times 2^n$ identity matrix).

Clearly we need at least 2^n rectangles just for all the 1s in the diagonal, and at least 1 more (it's actually 2^n more) for all the 0s, so we need more than 2^n rectangles.

Our result showed that sending n bits only suffices to get 2^n monochromatic rectangles, so we necessarily need to send at least one more.

Also look at example for the rectangle for parity and why it's easy.

Some result by Melhorn-Schmidt 1982: the communication complexity of a function is at least $\log(2rank(M_f) - 1)$, where M_f is the function's communication matrix.

There is also a conjecture called the log-rank conjecture, which is that the communication complexity is upper bounded by the poly log of the rank.

A canonical hard problem is disjointness: output 1 if the AND of all indices of the inputs is 0. This is like the "3SAT" of communication complexity. For those of you who know what this means, the communication matrix for disjointness on length n strings is $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{\otimes n}$, which means the matrix is full rank, and our result gives that we require $n+1$ bits. However, it's much less obvious why it's hard in the randomized setting, which we'll now talk about.

3.3 Randomized Communication

Things get a lot more interesting when Alice and Bob are allowed to toss fair coins. The protocol is considered correct if the probability of success, taken over the coin flips, is at least $\frac{2}{3}$ (in TCS we pick $\frac{2}{3}$ so we can use amplification - ask if you don't know what this means) . A natural (and common) question is do we allow public or private coins? For now let's consider private coins.

Now how can we compute equality quickly? How can randomization help?

Intuition - if we can turn one unequal bit into many unequal bits, then guessing randomly will probably catch the fact that there is an unequal bit.

Alice and Bob agree on a $[O(n), n, .1n]_2$ ECC. We use without proof the fact that such a code exists (it does). Now Alice and Bob consider $E(x)$ and $E(y)$. If $x \neq y$, then at least one tenth of the bits in $E(x)$ and $E(y)$ are different.

We can thus just have Alice choose some 100 indices i and send $(i, E(x)_i)$ for each of them. If $x \neq y$ then a uniformly random i has a one tenth chance of hitting i s.t. $E(x)_i \neq E(y)_i$, and thus the probability that Alice and Bob don't realize their inputs are different is $(\frac{9}{10})^{100}$ which is way less than $\frac{1}{3}$.

Alice only needs to send indices into $E(x)$ and the bits, so the complexity is a constant number of log length communications.

If Alice and Bob have shared randomness, Alice doesn't even need to send the indices, which is the main part of the communication cost. Instead, we can just send the constant number of $E(x)_i$ and we get constant communication in the public coins scenario.

We can even get it down to 3 bits of communication: Alice reads out some r_1, r_2 from the public randomness, computes $\langle r_1, x\rangle$ and $\langle r_2, x\rangle$, and sends these bits to Bob. Bob also computes $\langle r_1, y\rangle$ and $\langle r_2, y\rangle$.

We observe that $\langle r_1, x\rangle - \langle r_1, y\rangle = \langle r_1, x-y\rangle$. If $x \neq y$ then the inner product of a random string and a nonzero string is just a random bit. Thus, if $x \neq y$, then with probability $\frac{3}{4}$ we have that either $\langle r_1, x\rangle \neq \langle r_1, y\rangle$ or $\langle r_2, x\rangle \neq \langle r_2, y\rangle$. Bob just checks this and sends a 1 to Alice if all are equal.

This actually corresponds to using the bad Hadamard code, but we don't care that our encoding is ridiculously long since we don't have to send indices.

So clearly there's a difference between public and private randomness - the question is how can we characterize the power?

Getting private randomness from public randomness is easy - Alice and Bob can just agree to use different parts of the shared randomness (for example, even vs odd indices).

How do we get public randomness from private? Let BPP_ϵ^{priv} (and respectively BPP_ϵ^{pub}) denote the randomized communication complexity with private coins and error probability ϵ. We have a nice result called Newman's theorem:

Theorem. *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be some function. We claim that for all $\epsilon, \delta > 0$,*

$$BPP_{\epsilon+\delta}^{priv} = BPP_\epsilon^{pub} + O(\log n + \log \tfrac{1}{\delta^2})$$

Proof. Let Π Be some public coin protocol that succeeds with error ϵ and uses m random bits. First we state and prove the following lemma:

Lemma 1. *There exist strings $r_1, \ldots r_t$ for $t = O(\frac{n}{\delta^2})$ such that for all x, y,*

$$\Pr_{i \leftarrow_R [t]} [\Pi(x, y; r_i) \neq f(x,y)] \leq \epsilon + \delta$$

Proof. We use the probabilistic method by showing that if we choose the strings $r_1, \ldots r_t$ at random, then with positive probability we get the desired result.

Let the indicator I be the function where $I(x, y, r) = 1$ iff $\Pi(x, y; r) \neq f(x, y)$. We thus wish to show that

$$\frac{1}{t}\sum_{i\in[t]} I(x, y, r_i) \leq \epsilon + \delta$$

For some choice of the r_i. We know that for a fixed x and y and a single uniformly random r,

$$\mathbb{E}\left[I(x, y, r)\right] = \epsilon$$

Since Π has error ϵ. We can thus use the Chernoff bound to get that

$$\Pr_{r_1,\ldots r_t \leftarrow_R \{0,1\}^m}\left[\frac{1}{t}\sum_{i\in[t]} I(x, y, r_i) \geq \epsilon + \delta\right] \leq e^{-2\delta^2 t}$$

Union bounding over all possible x, y gives that the probability our error is higher than $\epsilon + \delta$ for ANY pair x, y is at most $2^{2n}e^{-2\delta^2 t}$, and so we just pick $t = O(\frac{n}{\delta^2})$ to get that this probability is less than 1. Thus, there exists $t = O(\frac{n}{\delta^2})$ strings $r_1, \ldots r_t$ that satisfy

$$\frac{1}{t}\sum_{i\in[t]} I(x, y, r_i) \leq \epsilon + \delta$$

for all x, y.

□

Now armed with this lemma we get a nice construction of a private coin protocol simulating a public coin protocol. If the protocol uses some m coins, Alice and Bob first get together and write a book of strings $r_1, \ldots r_t \in \{0, 1\}^m$ that satisfy the above lemma. Then when they are separated and run their protocol, Alice uses her private coins to send an index into their book so she and Bob can now use the same m bits.

Sending the index into a book of $t = O(\frac{n}{\delta^2})$ strings takes $O(\log n + \log \frac{1}{\delta^2})$ bits, which is the cost we pay to turn this into a public coin protocol.

$\square$

So one nice property of this method is it doesn't matter how large m is. Our protocol can be extremely expensive in terms of bits (thinking of randomness as a resource will come up again in pseudorandomness) but once again in communication complexity we ONLY care about how many bits are sent, so we're fine with Alice and Bob doing a ton of work to construct their books, as long as it doesn't involve sending bits.

Why can't Alice and Bob just get together and agree on the shared string before being separated, instead of going through this whole book of random strings process?

A result by Kalyanasundaram and Schnitger in 1992: disjointness still has complexity Omega(n) in the randomized setting. This result is hard to prove.