COMS-E6998: Information Theory in Computer Science **Fall 2017**

Lecture 6: Randomized Communication Complexity

Lecturer: Omri Weinstein *Scribes: Jiahui Liu, Andy Xu*

A central question in computation concerns the power of *randomization* – can we circumvent the hardness of solving a problem in the *worst-case*, by resorting to randomized algorithms? That is, the players are allowed to "toss coins" during the execution of the protocol and take the outcome of coin tosses into account to decide what to send to the other party.

The success of a randomized protocol can be defined in two ways. The first way is conservative, called **Las Vagas protocols**, which consider only (randomized) protocols that guarantees to output the correct value $f(x, y)$. Las Vegas protocols require 0 error but measure communication complexity in expectation. The second way is more relaxed, called **Monte Carlo protocols**, which allows protocols to error but must guarantee to compute the correct value $f(x, y)$ wit a very high probability.

We shall see that in the case of CC, randomization can sometimes be very powerful (but not always). Randomized CC also models reality in a better way in the sense that in real life protocols are very prone to errors (such UDP in computer networks).

6.1 Randomized Communication Protocols

6.1.1 Protocol Error

Suppose similar to the deterministic case, Alice gets x and Bob gets y as inputs to function $f(x, y)$ that they want to compute. They are now allowed to flip a random coin: Alice has access to a random string r_A of some arbitrary length, and Bob for r_B. Two strings are chosen independently according to some probability distribution.

We first look at the protocol tree, Alice's nodes will be labeled by some function of x and r_A, whereas Bob's nodes by some function of y and r_B. Every combination of x, y, r_A, r_B determines the leaves of the the protocol tree (the outputs of the protocol). Therefore, because we allow randomization, it's certainly possible that for a fixed (x, y), with different choices of r_A, r_B, the protocol can output different values. Some values are correct while other are error. Let's first try to formally define the error of a randomized protocol $\mathcal{P}$.

Definition 6.1.1. *Let $\mathcal{P}$ be a randomized protocol. Over random choices of r_A, r_B,*

1. *$\mathcal{P}$ computes a function f with zero error if and only if for every (x,y),*
$$\Pr[\mathcal{P}(x, y) = f(x, y)] = 1$$
2. *$\mathcal{P}$ computes a function f with ϵ-error if and only if for every (x,y),*
$$\Pr[\mathcal{P}(x, y) = f(x, y)] \geq 1 - \epsilon$$

6.1.2 Protocol Running Time and Cost

Randomized protocols not only allow Alice and Bob to output different values over different choices of random strings r_A, r_B, but also allows **number of bits exchanged in each round to vary** over different choices of random strings r_A, r_B. Therefore, naturally one would come up with two measures of protocol running time.

Definition 6.1.2. *Given a randomized protocol $\mathcal{P}$ that takes in input (x, y) and random strings r_A and r_B.*

1. ***Worst Case**: The worst case running time of $\mathcal{P}$ is defined to be the maximum number of bits communicated for **any** choices of the random strings r_A, r_B. The worst case cost of $\mathcal{P}$ is the maximum, over all inputs (x, y), of the worst case running time of $\mathcal{P}$ on (x, y). Intuitively, the worst case cost can also be seen as the deepest depth of the protocol tree over all possible input (x, y).*

2. ***Average Case**: The average case running time of $\mathcal{P}$ is defined to be the expected number of bits communicated over all choices of r_A, r_B. The average case cost of $\mathcal{P}$ is the maximum, over all inputs of (x, y), of the average case running time of $\mathcal{P}$ on (x, y).*

6.1.3 Complexity measures in Randomized CC

The two types of errors we discuss here lead to two complexity measures, for a specific function f. In each case, the complexity to compute a function f, if allowed randomization, is the cost of the "best"(cost-the-least) protocol that meets the error requirement.

Definition 6.1.3. *Let $f : X \times Y \to \{0, 1\}$ be a function that Alice and Bob tries to compute. We consider the following complexity measures for f.*

1. *$R_0(f)$ is the minimum **average case cost** of a randomized protocol that computes f with zero error.*

2. *For $0 < \epsilon < \frac{1}{2}$, $R_\epsilon(f)$ is the minimum **worst case cost** of a randomized protocol that computes f with error ϵ*

For zero error protocols, we are only interested in the average case cost for the best randomized protocol. Because otherwise using the worst case cost would just give the deterministic communication complexity. On the other hand, we are interested in the worst case cost for protocols that are allowed errors. This is because the complexity remains the same within a multiplicative constant factor whether we work with it worst case measure or average case measure. (This can be proved using reduction from average case protocol to worst case protocol). Since it's easier, as we shall see later in the public and private coin model, to work with worst case cost, we choose to use the worst case cost measurement for protocols that allow error.

6.2 Private and Public coin models

There are two natural randomized models of communication: the public coin model and the private coin model.

6.2.1 Public coin model

In the *public-coin* model, Alice and Bob both have free access to a *shared random string* R which is written in the sky and is viewable to both parties (w.l.o.g, R is an arbitrary r.v *independent* of x, y). The way the protocol work is that the players first draw $r \sim R$, and then they execute a *deterministic* protocol $\pi(x, y, r)$, like the "hashing" protocol we will see for testing Equality. Therefore, a public-coin protocol is nothing but a *distribution on deterministic protocols.* The correctness of the protocol π is now defined over the randomness R but over *worst-case* inputs (i.e., the worst-case number of bits sent in any execution of the protocol):

$$\mathsf{R}_{\varepsilon}^{pub}(f) := \min_{\pi \,:\, \Pr_r[f(x,y)=\pi(x,y,r)]\geq 1-\varepsilon \ \forall (\mathbf{x},\mathbf{y})} \|\pi\|.$$

For simplicity, we will typically set $\varepsilon = 1/3$ w.l.o.g, since we can always amplify the constant by repeating the protocol independently. The complexity of the protocol only scales by a constant factor.

Theorem 6.2.1. $\mathsf{R}_{\epsilon}^{pub}(EQ_n) = O(log(\frac{1}{\epsilon}))$.

Proof. In the *public-coin* model of equality, Alice is given a string $x \in \{0,1\}^n$ and Bob is given $y \in \{0,1\}^n$ and they have access to infinite many bits of public randomness. The protocol is through hashing, depicted as follows:

(1) Draw k random uniform $z \in \{0,1\}^n$: $z_1, z_2, ...z_k$

(2) Compare if $\langle z_i, x\rangle = \langle z_i, y\rangle \%2$ for each $i \in [k]$.

If $x = y$: all $\langle z_i, x\rangle$ and $\langle z_i, y\rangle$ will be the same.

If $x \neq y$: every test will have probability $\frac{1}{2}$ to catch it.

So if we run $k = O(log(\frac{1}{\epsilon}))$ rounds, the probability that Alice and Bob output the wrong answer, i.e. all k tests pass even though $x \neq y$ is:

$$\Pr(\text{all k succeeds}|x \ \neq y) = \frac{1}{2^k} = \epsilon$$

Since each round Alice and Bob send only constant bits to each other, the communication complexity is $O(k) = O(log(\frac{1}{\epsilon}))$. □

6.2.2 Private coin model

In the *private-coin* model, there's no shared random tape, but instead Alice and Bob have *private access* to randomness, hence they each hold R_A and R_B separately, and each message of Alice (resp. Bob) may depend on R_A (R_B), i.e., $M_i(x, m_{<i}, r_a)$ for odd i (and $M_i(y, m_{<i}, r_b)$ for even i). We define similarly:

$$\mathsf{R}_{\varepsilon}^{priv}(f) := \min_{\pi \,:\, \Pr_{r_a,r_b}[f(x,y)=\pi(x,y,r_a,r_b)]\geq 1-\varepsilon \ \forall (\mathbf{x},\mathbf{y})} \|\pi\|.$$

Claim 6.2.2. $R_{\epsilon}^{pub}(f) \leq R_{\epsilon}^{priv}(f)$

Public coin protocols are a stronger model, since they can always simulate the private coin model: we can divide the shared randomness R into R_A and R_B for Alice and Bob to use as private randomness respectively.

For example, the $O(1)$ protocol we saw for $EQ_{1/3}$ was a public-coin protocol. Can we achieve the same with private coins?

Theorem 6.2.3. $\mathsf{R}_{\epsilon}^{priv}(EQ_n) = \Theta(\lg n)$.

Proof sketch: (UB) The idea is to interpret the inputs as degree-n polynomials over a filed of size $\gtrsim n$ (and to use the property of degree-n polynomials that any 2 distinct ones cannot agree on more than n points):

Alice and Bob agree in advance on a prime number $n^2 < p < 2n^2$, and Alice uses her input to define the polynomial $P_x(y) := \sum_{j=0}^{n-1} x_j \cdot y^j$ over $\mathbb{F}_p$. Alice and Bob will now check whether $P_x(i) =^? P_y(i)$ for a random $i \in \mathbb{F}_p$ (note that the choice of p guarantees that no 2 rows are the same mod p). Indeed, Alice now draws a random number $i \in_R [n]$ and sends Bob the value of i and the evaluation of the polynomial $P_x(i)$ $\sum_{j=0}^{n-1} x_j \cdot i^j \mod p$. Bob returns "YES" iff $P_x(i) = P_y(i)$, which succeeds w.p 1 if $x = y$, and fails w.p $\leq (n-1)/|\mathbb{F}_p| \leq 1/4$ if $x \neq y$.

This technique can actually be used in error-correcting codes: even 1-bit of difference can be amplified to many differences in codewords.

(LB) The proof actually follows from a more general and simple claim:

$$\forall f \;\; \mathsf{R}_\epsilon^{priv}(f) \geq \Omega(\lg \mathsf{D(f)}).$$

The idea is that Alice and Bob can simulate the randomized protocol if they exchange the probabilities that they end up down a certain branch in the randomized protocol (which has $\leq 2^{\mathsf{R}_\epsilon}$ paths). The analysis needs to be done with some care however since we have to encode real numbers by rationals, so we need to keep track of truncation errors.

6.2.3 Newman's lemma

The equality example is somewhat disappointing, since public-coin protocols are much nicer to work with and to analyze. Luckily, a rather simple *statistical argument* shows that this is the *largest gap* b/w public and private coin protocols:

Lemma 6.2.4 (Newman's Lemma)**.** $\mathsf{R}_{\varepsilon+\delta}^{priv} \leq \mathsf{R}_\varepsilon^{pub} + O(\lg(n/\delta))$.

Proof Sketch : We need to convert a public coin protocol into a private coin protocol. The obvious idea would be for Alice to sample the maximum number of bits required to carry out the protocol from her source of randomness R_A and send this string to Bob. This way Alice and Bob would have access to the same string of randomness and could carry out the public coin protocol. We will do exactly this, but first we need to reduce the amount of randomness used by the public coin protocol since this cost will be added to the cost of communication in the private coin protocol. Suppose there existed t strings $r_1, \ldots, r_t$ where $t = poly(n)$ such that the public coin protocol δ error was at most $\delta + \varepsilon$ when we uniformly sampled a random string from among these choices. Then these strings could be fixed in advance, Alice would only need to index into this sequence and it would cost her only $O(\lg n)$ bits of communication. So we will prove that such a sequence d of random choices exists.

The key to proving this is using Hoeffding/Chernoff bounds (which we have seen in one of the first lectures of this course). For a fixed x, y, the probability that the sampled error deviates by more than δ from the real error on t samples of r is at most $\approx 2^{t\delta^2}$. If we set $t = 10n/\delta^2$, the probability of this event becomes at most 2^{-3n}. But there are only 2^{2n} possible pairs of inputs (x, y), so by a union bound the probability that a sequence of random strings $r_1, \ldots, r_t$ has mean error at least $\delta + \varepsilon$ for some pair of inputs x, y is at most 2^{-n}. By the probabilistic method, there must now exists some fixed sequence of random strings that achieves this performance. QED.

Thus, for the sake of randomized communication, we never need to worry again about the private model (so long as we don't care about additive logarithmic factors, which is usually negligible). We caution that that situation will be somewhat *reversed* when we start talking about *information cost* of a protocol (when trying to minimize *information*, the crucial resource will actually be private randomness), so we shall not forget about private coins.

Example 6.2.5 (Small-set Disjointness)**.** *One of the most surprising examples demonstrating the power of randomized protocols is $DISJ_n^k$. Naively, this requires $O(k \lg n)$ bits. The Hastad-Wigderson protocol achieves $O(k)$ bits (regardless of the universe size!).*

Exercise 6.2.6. *HW : Show that any (boolean) function on n bits strings admits a 2-bit public-coin protocol which answers correctly w.p $\geq 2^{-n}$.*

6.3 Distributional CC and the minimax principle

In order to prove lower bounds on randomized CC problems (which will be our main proxy for the future applications e.g., in streaming, data structures and interactive compression), it will be convenient to have a *distribution* over the inputs (x, y) (for example, this will let us talk about information learnt by the protocol, which is not defined without an underlying distribution). Recall that in the randomized model, inputs were worst-case and π was randomized. In the distributional CC model, all the randomness can be rolled into the inputs:

$$\mathsf{D}_\varepsilon^\mu(f) := \min_{\pi \,:\, \Pr_\mu[f(x,y)=\pi(x,y)] \geq 1-\varepsilon} \|\pi\|.$$

The inputs x, y come from a distribution μ that both ALice and Bob know of. the goal is to minimize CC for solving $f(x, y)$ with error ϵ under μ.

Note that, by the averaging principle, we may assume w.l.o.g that π is *deterministic* (indeed, there will always by a fixed r that achieves the expectation $\mathbb{E}_{r,\mu}[...] = \mathbb{E}_r[\mathbb{E}_\mu[...]] \leq \max_r \mathbb{E}_\mu[\pi_r...]$).

We stress here that the choice of distribution μ is crucial and may dramatically affect the hardness of the function:

Example 6.3.1.

- *Consider EQ_n (or $DISJ_n$) over the* uniform *distribution – if we just output* 0 *we'll be correct w.p $\geq 1 - 2^n$! (0 CC!).*
- *Consider the GT_n function over the* uniform *distribution. (binary search will terminate after $O(1)$ rounds w.h.p).*
- *Consider $DISJ_n$ over the* product *distribution that assigns $x_i, y_i = 1$ independently w.p $1/\sqrt{n}$. This ensures that the probability of intersection is $\approx 1/2$ (birthday paradox), hence the problem is nontrivial. Alice and Bob can solve this function by having Alice send Bob her input – $H(X_1, \ldots, X_n) = n \cdot H(1/\sqrt{n}) = \Theta(\sqrt{n} \lg n)$, so she can use Huffman code to compress X. It turns out that this is optimal (Babai et. al).*

The fact that randomness doesn't help in the distributional model means that

$$\forall \;\; \mu, \qquad \mathsf{R}_\varepsilon(f) \geq \mathsf{D}_\varepsilon^\mu(f).$$

This $\geq$ direction is simple: the randomized protocol is correct for every input with probability $> 1 - \epsilon$. Therefore, for each μ, the randomized protocol is correct with probability $> 1\epsilon$, where the probability is taken over both the public coin flips and the random input. It follows by a counting argument, that for some fixed choice of the public coin flips, a probability of success larger than 1ϵ is achieved, where this time the probability is taken only over the inputs.

What is surprising is that the converse of this theorem – the is always some *worst* distribution that *achieves* the randomized CC of a function:

Theorem 6.3.2 (Yao's minimax principle). $\mathsf{R}_\varepsilon(f) = \max_\mu \ \mathsf{D}^\mu_\varepsilon(f)$.

The theorem is an elegant application of the Von-Neumann minimax principle for zero-sum games. This theorem renders the problem of CC lower bounds into the "art" of designing a hard distribution, circumventing the need to deal with any extra randomness of the protocol, which is often easier to analyze.

Proof. Consider a two-player zero-sum game as follows. There is an $M \times N$ payoff matrix A. The M rows index ways in which player 1 can make a move, and the N rows index the ways in which player 2 can make a move. Both players simultaneously choose a move, say row i and column j. Then player 1 has to give player 2 the amount $A[i, j]$. Now allow the players to be randomized. A strategy for player 1 (player 2) is a distribution σ on the rows (μ on the columns). Each player chooses a move according to his/her strategy. The expected payoff is $\sigma^\top A\mu$. The goal of player 1 is to choose a σ that minimizes, over the worst choice of μ, the expected payoff $\sigma^\top A\mu$. The goal of player 2 is to choose a μ that maximizes, over the worst choice of σ, the expected payoff $\sigma^\top A\mu$. The celebrated min-max theorem says that if a player announces his/her strategy and allows the other player to make an adversarial choice,the expected payoff is the same no matter which player announces the strategy. That is,

$$\min_\sigma \max_\mu \sigma^\top A\mu = \max_\mu \min_\sigma \sigma^\top A\mu$$

In the communication setting we consider, the rows of A index deterministic protocols with cost at most c. The columns index inputs $(x, y) \in X \times Y$. Thus μ is a distribution on inputs, and σ is a distribution on deterministic cost c protocols i.e. a randomized cost c protocol. $[P, (x, y)]$ is 1 if $P(x, y) = f(x, y)$, 0 otherwise.

Since $\forall\mu, D_\mu(f) \leq c$, we see that $\min_\sigma \max_\mu \sigma^\top A\mu \leq \epsilon$. (For each μ, find the deterministic cost-c protocol P that works for it, and set σ to be 1 on P and 0 elsewhere. Then $\sigma^\top A\mu$is exactly the error of P with respect to μ.) By the min-max theorem, $\min_\sigma \max_\mu \sigma^\top A\mu \leq \epsilon$. Consider the distribution σ that achieves this minimum. Then the corresponding randomized protocol P_σ has error at most ϵ on every distribution. In particular, for any input (x, y), let $\mu_{(x,y)}$ be the distribution that is 1 at (x, y) and 0 elsewhere; P_σ has error at most ϵ on $\mu_{(x,y)}$. Thus for every input (x, y), P_σ has error at most ϵ. □