Lecture 12

Randomized Communication Complexity; Branching Programs

May 8, 2008
Lecturer: Paul Beame
Notes: Punyashloka Biswal

12.1 Randomized Communication Complexity

12.1.1 Definitions

A *randomized protocol* $\mathcal{P}$ is a probability distribution on deterministic protocols P. The *cost* of a protocol is defined by

$$cost(\mathcal{P}) = \sup_{P\sim\mathcal{P}} cost(P).$$

Let $P(x, y)$ denote the output of a deterministic protocol on inputs x and y, and let $f : X \times Y \to Z$. Then we say that $\mathcal{P}$ *computes* f *with* (*two-sided*) *error* ϵ if

$$\text{for all } x \in X \text{ and } y \in Y, \qquad \Pr_{P\sim\mathcal{P}}\big[P(x, y) \neq f(x, y)\big] \leq \epsilon.$$

In this definition, the random choice of protocol is independent of the inputs. We can equivalently consider a model where Alice and Bob follow fixed protocols but share a random string which is used to select the protocol.

We denote the *randomized communication complexity of* f *with error* ϵ, $R^{\text{cc}}_\epsilon(f)$, by

$$R^{\text{cc}}_\epsilon(f) = \inf\big\{cost(\mathcal{P}) \;\big|\; \mathcal{P} \text{ computes } f \text{ with error } \epsilon\big\}.$$

If we restrict attention to protocols where Alice and Bob only receive separate, independent random strings, we get the randomized communication complexity model with private coins, which is denoted by $R^{\text{cc,priv}}$.

12.1.2 Example: testing equality

We exhibit two randomized protocols for $\text{EQ} = \{(x, x) \mid x \in \{0, 1\}^*\}$, the language of pairs of equal strings.

Private coin protocol

We view the inputs as n-bit integers x and y. Alice chooses a random prime p from among the first $4n$ primes and sends Bob the pair $(p, x \bmod p)$. Bob computes $y \bmod p$ and accepts if and only if it equals $x \bmod p$.

By the prime number theorem, the largest prime p at most $4n \log_2 n$ and so is $O(\log n)$ bits long. It is clear then that this protocol has cost $\Theta(\log n)$ and always accepts when x and y are equal. Now $x \bmod p = y \bmod p$ if and only if p divides $x - y$. If $x, y \in [n]$ are distinct then $|x - y| < 2^n$ and so has at most n distinct prime factors. Since there $4n$ choices for p, the probability that x and y are congruent modulo p is at most $1/4$. This shows that $R^{\text{cc,priv}}_{1/4}(\text{EQ})$ is $O(\log n)$. This can be shown to be asymptotically optimal.

Shared coins

We view the inputs x and y as vectors in $\mathbb{F}_2^n$, and the random string as two vectors s and t. Alice sends Bob the pair $(s \cdot x, t \cdot x)$ where the inner products are computed over $\mathbb{F}_2$. Bob accepts if and only if the pair sent by Alice matches $(s \cdot y, t \cdot y)$.

Again, it is clear that this protocol has constant cost and always accepts when x equals y. When the inputs are unequal, the probability that $s \cdot (x - y) = 0$ is exactly $1/2$. Because s and t are chosen independently, this shows that $R^{\text{cc}}_{1/4}(\text{EQ})$ is $O(1)$.

12.1.3 The value of shared randomness

The $\Theta(\log n)$ difference between the costs of the private-coin and public-coin protocols for the last example is actually tight, because of the following result.

Theorem 12.1 (Newman) For any randomized ϵ-error protocol $\mathcal{P}$ computing f, there is an $(\epsilon + \delta)$-error protocol $\mathcal{P}'$ that uses $O(\log(1/\delta) + \log n)$ bits of shared randomness.

Proof [Proof idea] Use Chernoff bounds to show that we can choose a subset $S \subseteq supp\mathcal{P}$ of size polynomial in n and $1/\delta$, such that the error bound remains valid when P is chosen uniformly from this subset. Then only $\log_2 |S|$ bits of shared randomness are required. □

We can now have a protocol with private randomness in which Alice chooses the value for the shared random string in the protocol produced by Newman's theorem and sends this to Bob, after which both players execute the protocol with shared randomness.

12.1.4 Distributional communication complexity

Let μ be a probability distribution on $X \times Y$. Then we write

$$D^{\text{cc}}_{\mu,\epsilon}(f) := \inf\{cost(P) \mid \Pr_{\mu}[f(x,y) \neq P(x,y)] \leq \epsilon\}$$

for the cost of protocol P on input distribution μ. The Yao Minimax Lemma establishes a duality between randomized protocols and distributional adversaries, allowing us to prove tight lower bounds on the randomized complexity of a problem by exhibiting a hard input distribution.

Theorem 12.2 (Yao Minimax Lemma) Let $f : X \times Y \to Z$ be any function and let $\epsilon > 0$.

1. For any distribution μ over $X \times Y$, $R^{\text{cc}}_{\epsilon}(f) \geq D^{\text{cc}}_{\mu,\epsilon}(f)$.
2. There exists a distribution μ^* for which $R^{\text{cc}}_{\epsilon}(f) = D^{\text{cc}}_{\mu^*,\epsilon}(f)$.

Proof This theorem can be viewed as a consequence of linear programming duality, though historically it follows from Von Neumann's Minimax Lemma in game theory from 1927 which pre-dated LP duality. We will sketch a direct proof of the first part.

Let Π be the set of all deterministic protocols of cost $R^{cc}_{\epsilon}(f)$, and define the zero-one matrix M as follows. For $(x, y) \in X \times Y$ and $P \in \Pi$ define

$$M_{(x,y),P} := \begin{cases} 1 & P(x,y) \neq f(x,y) \\ 0 & \text{otherwise.} \end{cases}$$

By the definition of $R^{cc}_{\epsilon}(f)$, we know that there is a distribution $\pi = \mathcal{P}$ over columns such that for each $(x, y) \in X \times Y$,

$$\epsilon(x,y) = \sum_{P \in \Pi} M_{(x,y),P}\, \pi(P) \leq \epsilon$$

where $\epsilon(x, y)$ is the probability that the randomized protocol is incorrect on input (x, y). Now if we fix an input distribution μ over $X \times Y$, we have

$$\epsilon \geq \sum_{(x,y) \in X \times Y} \mu(x,y)\epsilon(x,y) = \sum_{\substack{(x,y) \in X \times Y \\ P \in \Pi}} \mu(x,y) M_{(x,y),P}\, \pi(P) = \sum_{P \in \Pi} \epsilon(P)\pi(P),$$

where $\epsilon(P) = \sum_{X \times Y} \mu(x, y) M_{(x,y),P}$ is the probability that protocol P is incorrect for a random input (x, y) chosen according to μ. Since π is distribution over Π it must be that $\min_{P \in \Pi} \epsilon(P) \leq \epsilon$. Fixing the protocol $P \in \Pi$ achieving this minimum shows that $D^{cc}_{\mu,\epsilon}(f) \leq R^{cc}_{\epsilon}(f)$. □

There is nothing particularly special about communication complexity in the above lemma. It generally applies to any combinatorial model of randomized computation.

12.1.5 Lower bounds

The chief technique used to prove lower bounds on randomized communication complexity is the *discrepancy method.* The discrepancy of a function measures how close it is to random by considering how balanced it is on large nicely-structured sets of inputs; in particular on combinatorial rectangles.

Recall that a combinatorial rectangle in $X \times Y$ is a set of the form $A \times B$, where $A \subseteq X$ and $B \subseteq Y$. Formally, the discrepancy of a function $f : X \times Y \to \{0, 1\}$ over a combinatorial rectangle R, under an input distribution μ, is given by

$$disc_{\mu,R}(f) := \left|\mu\big(f^{-1}(0) \cap R\big) - \mu\big(f^{-1}(1) \cap R\big)\right|,$$

and the discrepancy of f under μ is defined as

$$disc_{\mu}(f) := \max_{\text{rectangles } R} disc_{\mu,R}(f).$$

Roughly speaking, a protocol divides the input space $X \times Y$ into combinatorial rectangles over which it gives a constant answer. A function of small discrepancy must be far from constant on any large rectangle. Therefore, a function of small discrepancy forces a protocol of small error to divide the space into a large number of small rectangles, which requires many bits of communication. This intuition is made precise in the following theorem, which is easy prove given the above intuition but which we state without proof:

Theorem 12.3 $D^{cc}_{\mu,\epsilon}(f) \geq \lg\left(\frac{1-2\epsilon}{disc_\mu(f)}\right)$.

Along with the Yao minimax lemma, this lets us prove lower bounds on randomized communication complexity. In particular, the following bound can be shown on discrepancy of the Inner Product function.

Lemma 12.4 (Lindsay's Lemma) Let U be the uniform distribution on $\{0,1\}^n \times \{0,1\}^n$, and let $IP : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ map $(x,y) \mapsto x \cdot y \bmod 2$. Then $disc_U(\text{IP}) \asymp n^2/2^n$.

This immediately implies that $R^{cc}_\epsilon(\text{IP}) = \tilde{\Omega}_\epsilon(n)$.

A method called the 1-sided discrepancy method or corruption method can be used to show linear lower bounds for the randomized communication complexity of the set disjointness (set intersection) problem, which we recall was given by $DISJ(x,y) = \bigvee_i (x_i \wedge y_i)$. Unlike the inner product function, under the uniform distribution the $DISJ$ function has large discrepancy (at least $1/n$) since at least a $1/n$ fraction of all of $DISJ^{-1}(1)$ has $x_i = y_i = 1$. However, $DISJ$ has the property that under suitable distributions rectangles R such that $\mu(R \cap DISJ^{-1}(1)) \leq \epsilon\mu(R)$ must be exponentially small which yields good communication complexity bounds for a similar reason to discrepancy.

12.1.6 Multi-party communication complexity

There are circumstances in which it is necessary to understand or one can make use of communication protocols in which there are more than two players. There are two possible generalizations of the standard communication model to more than two players. In both of them, there are k players who wish to collaboratively compute a function $f : \prod_{i=1}^k X_i \to Y$.

- The *number in hand* model, where player i initially knows only input x_i. These can be used to prove lower bounds for streaming algorithms.
- The *number on forehead* model, where player i initially knows all the x_js where $j \neq i$. (Since each player can view all the other players' inputs it is as if they are written on their foreheads.) This leads to lower bounds on circuit complexity and time-space tradeoffs.

Based on the fact that ACC circuits as computable via symmetric function of quasi-polynomially many AND gates of polylog fan-in, one can show that multiparty communication complexity lower bounds in the number of forehead model for $\log^{\omega(1)} n$ players can yield circuit lower bounds on ACC. The best current lower bounds apply when the number of players is less than $\log_2 n$.

12.2 Branching Programs

12.2.1 Definitions

An *R-way branching program* on n variables $x_1, \ldots, x_n$ is a directed, acyclic graph in which all internal nodes have out-degree R, each internal node is labelled with a variable x_i, each edge is labelled with a value in $[R]$, and each leaf is labelled with a value from some set Y. Any assignment of values to variables defines a path through the branching program, chosen by starting at the root and repeatedly following the edge labelled by the value of the variable corresponding to the current

node. Therefore, the branching program defines a function $[R]^n \to Y$ given by the label of the leaf node at the end of the path. We often consider *Boolean branching programs* which edges and leaves have labels from $\{0, 1\}$.

The *size* of a branching program is the number of nodes in the graph. We write $BP(F)$ for the optimal size of a branching program for f. The *length* of a branching program is the length of the longest path; it corresponds to the running time. A branching program is said to be *levelled* if all paths are of the same length; when this is the case, we can speak of the internal nodes at a particular level. A levelled branching program is said to be *oblivious* if the label of an internal node only depends upon its level. A branching program is said to be *read-once* if any variable appears at most once as a label on a path from the root to a leaf node. Read-once oblivious branching programs are also called *(ordered) binary decision diagrams*. If a branching program is levelled, we say that it has *width* equal to the maximum number of nodes at any level. We shall see below that for many bounds, there is no loss of generality if we only look at levelled BPs.

Boolean branching programs are like Boolean decision trees with identified internal nodes. Branching programs can dramatically save on the number of nodes required relative to decision trees. For example, $Parity_n$ requires $2^n - 1$ node decision trees but can be computed the a simple read-once oblivious branching program with 2 nodes per level, one such that the parity of variables tested so far is 0 and the other that it is 1.

Theorem 12.5 If a function f can be computed by an (arbitrary) branching program of length T and size s, then it can also be computed by a *levelled* branching program of length T and size $sT + 1$.

Proof Make T copies $v_1, \ldots, v_T$ of every node v in the branching program. For each edge (u, v) in the original branching program put an edge (u_i, v_{i+1}) for $i = 1, \ldots, T - 1$. (Add similar edges from the root to level 1.) □

12.2.2 Connections to uniform complexity

Lower bounds on the branching program size and length imply corresponding lower bounds on uniform complexity, as shown in the following theorem.

Theorem 12.6 Let f be any function.

1. If $f \in \mathsf{TIME\text{-}SPACE}\big(t(n), s(n)\big)$, then there exists a branching program computing f of size $2^{O(s(n)+\lg n)}$ and length $O(t(n))$.

2. If f has a RAM algorithm with inputs in $[R]^n$ that has running time $t(n)$ and takes space $s(n)$ (where the model is unit-cost with respect to time and log-cost with respect to space), there is an R-way branching program for f of size $2^{O(s(n)+\lg n)}$ and length $O(t(n))$.

Proof We will prove the first part of the claim; the second part is similar. Let M be a TM that computes f in time $t(n)$ and space $s(n)$. As in the proof of Savitch's theorem, we shall construct a branching program whose nodes correspond to the part of the configurations of M that do not include the input. That is, each consists of the state of M, the position of the read head on the input tape, and the content and positions of the read/write heads on its storage tapes. The index of

the variable read by a node corresponds to the location of the M's input head in that configuration, and the outgoing edges go to the configurations it would read in the next step depending on the value of the input bit. □

Going in the reverse direction, we see that polynomial-sized branching programs provide a useful non-uniform version of L, deterministic log-space.

Remark: In discussing the RAM model above we use the term *log-cost with respect to space* iff the space bound is the number of memory locations multiplied by the number of bits per word of memory; i.e. the total number of bits of memory. *Unit-cost with respect to time* means that each memory access counts for one unit of time. If one allows unit cost for both time and space then by repeated multiplications one can quickly create enormous integers which allow such a model to simulate PSPACE.

12.2.3 Connections to circuit complexity

Recall that $L(f)$ is the minimum number of leaves of a formula computing f and $size(f)$ is the circuit size of f over the basis $\{\vee, \wedge, \neg\}$.

Theorem 12.7 Let $f : \{0,1\}^n \to \{0,1\}$ be any function.

1. $BP(f)$ is $O(L(f))$.
2. $size(f)$ is $O(BP(f))$.

Proof Let F be a minimal-size formula for f. We shall proceed inductively. If $F = \neg F'$, then we construct the branching program for F' and switch the labels on the leaves. If $F = F_1 \vee F_2$, then we begin by constructing the corresponding branching programs P_1 and P_2 for F_1 and F_2 respectively. Then we make the 1-output node of P_1 an output node for the whole BP, and we connect the 0-output node to the start node of P_2. A similar conversion for the case $F = F_1 \wedge F_2$ completes the proof.

On the other hand, let P be a minimal-size branching program for f. We work inductively from the output nodes which become the associated constant inputs for the circuit. Then any node with label x_i and outedges to P_0 and P_1 on 0 and 1 respectively can be replaced in the circuit by a node computing the equivalent value $(x_i \wedge P_1') \vee (\neg x_i \wedge P_0')$ where P_0' and P_1' are the circuit nodes corresponding to P_0 and P_1 respectively. Applying this conversion throughout the branching program gives us a circuit of size $O(BP(f))$. □

A simple counting argument shows that there exist functions that require exponential-size BPs: there are 2^{2^n} functions $\{0,1\}^n \to \{0,1\}$, but only $s^{O(s)}$ BPs of size s. Just as for formulas finding explicit functions for which there are size lower bounds if much harder. Nečiporuk proved an analog of his result about formulas (see Lecture 7) for branching programs. The reason that this is also possible for BPs (unlike circuits) is that we can associate each part of the BP with a particular variable.

Theorem 12.8 (Nečiporuk) Let $B_1, \ldots, B_k$ be a partition of $[n]$. Then there is a constant $c > 0$ such that for any function $f : \{0,1\}^n \to \{0,1\}$,

$$BP(f) \geq \sum_{i=1}^{k} c \log(|SUB(f, B_i)|) / \log\log(|SUB(f, B_i)|).$$

In particular the following lower bound is a consequence.

Corollary 12.9 The *element distinctness* function $\mathrm{ED_N} : [N^2]^N \to \{0,1\}$ is defined as

$$\mathrm{ED}(x_1, \ldots, x_N) := \bigwedge_{i \neq j} (x_i \neq x_j).$$

Then $BP(\mathrm{ED}_N) = \Omega(N^2)$.

Since ED_N takes $2N \log N$ bits of input, this is actually a $\Omega(n^2/\log^2 n)$ bound in our usual notation. This is the largest size lower bound known for any explicit function.

One way to try to get better size lower bounds is to consider restricted classes of branching programs to see if one can prove better size lower bounds. One of the first approaches was to consider levelled BPs and restrict their width to be constant. By applying the simulations of circuits by branching programs described above One can easily see for each that for constant m any functions in $\mathsf{AC}^0[m]$ can be computed by constant-width polynomial-size BPs. However, constant-width polynomial-size BPs turned out to be surprisingly much more powerful than that.

Recall that NC^1 is the set of all functions $f : \{0,1\}^* \to \{0,1\}$ of polynomial formula size (equivalently computable by formulas or circuits of polynomial size and logarithmic depth).

Theorem 12.10 (Barrington) A Boolean function $f : \{0,1\}^* \to \{0,1\}$ can be computed by a polynomial-sized, width-5, oblivious branching program if and only if $f \in \mathsf{NC}^1$.

Proof We begin with the easy direction, showing how to convert any width-5 oblivious BP into an equivalent NC^1 circuit. We view the edges for each level of a width-5 BP as a pair of functions ℓ_0 and ℓ_1 mapping $[5] \to [5]$. The input variable x_i queried at this stage determines which function is applied. In fact, the operation of an entire width-5 branching program can be described by a function mapping $[5] \to [5]$. In particular an oblivious width-5 branching program of length m is described by level functions ℓ_0^j and ℓ_1^j for $j = 1, \ldots, m$ and by the sequence $i_1, \ldots, i_m$ of indices of variables queried. Then the function describing the operation of the BP as a mapping from [5] to [5] is given by the composition $\ell^m_{x_{i_m}} \circ \cdots \circ \ell^1_{x_{i_1}}$. (The output is the resulting function applied to 1.) Since function composition is associative, we can write a circuit for the composition using a full binary tree of function compositions whose inputs are created using selection on the x_{i_j} to determine which of ℓ_0^j and ℓ_1^j to include in the composition. There is a circuit of constant size and depth that takes two binary strings representing functions from [5] to [5] and produces a binary string representing their composition so the resulting circuit has $O(\log m)$ depth and size $O(m)$.

For the interesting direction of the proof, we shall systematically construct a family of BPs to represent any log-depth polynomial-size circuit C which we can assume is a De Morgan formula. Let S_5 be the group of permutations on five elements, and let $G \subseteq S_5$ be the subgroup consisting of all 3-cycles. As above we can view every level of a width-5 oblivious BP as a pair of functions

$\ell_0, \ell_1 : [5] \to [5]$, one of which is chosen depending on the value of the variable read. In our construction, we will always set ℓ_0 to the identity map, and we will choose ℓ_1 from G.

The key property of G that we will use is that there are permutations $\sigma, \tau \in G$ such that $\sigma\tau\sigma^{-1}\tau^{-1}$ is not the identity. (This existence follows because G is a non-solvable group.) In particular, (1 2 3)(3 4 5)(3 2 1)(5 4 3) = (2 3 5). For every NC^1 circuit $C(x)$ and element $\pi \in G$ that is not the identity, we shall show how to construct a BP M_π such that $M_\pi(x)$ acts as the identity if $C(x) = 0$, and π if $C(x) = 1$ (equivalently, $M_\pi(x) = \pi^{C(x)}$). It is clear how to do this when $C(x)$ is a constant or a literal.

When $C(x) = \neg C'(x)$, then we inductively construct a BP $M'_{\pi^{-1}}$ for C', and then use it to construct $M_\pi(x) = \pi M'_{\pi^{-1}}(x)$ where $\pi M'$ for branching program M' amounts to renumbering its input nodes according to π.

When $C(x) = C'(x) \wedge C''(x)$, then we find $\sigma, \tau \in G$ such that $\pi = \sigma\tau\sigma^{-1}\tau^{-1}$ by renaming the points (2 3 5) to be the points of the cycle π in the construction given above. We inductively construct machines M'_σ and $M'_{\sigma^{-1}}$ for C', and M''_τ and $M''_{\tau^{-1}}$ for C''. Then we build the program

$$M_\pi = M'_\sigma M''_\tau M'_{\sigma^{-1}} M''_{\tau^{-1}}.$$

Observe that if $C'(x) = 0$ then $M'_\sigma(x)$ and $M'_\sigma(x)$ compute the identity and the resulting BP applied to x computes the permutation $\tau\tau^{-1}$ which is the identity. The analogous case applies when $M''_\tau(x)$ and $M''_{\tau^{-1}}(x)$ both produce the identity. Therefore the function on [5] computed $M_\pi(x)$ is the identity if $C(x) = 0$ and is π if $C(x) = 1$.

When $C(x) = C'(x) \vee C''(x)$, then we reduce to the previous case using De Morgan's law, i.e., $C(x) = \neg\big(\neg C'(x) \wedge \neg C''(x)\big)$. The size of the BP grows by a factor of 4 for each binary gate and remains the same for each negation $\neg$. By induction, the length of the resulting BP is $4^{depth(C)} = 4^{depth(f)}$. Since $depth(C)$ is $O(\log n)$ the BP constructed is polynomial size. □