CAS CS 591 B: Communication Complexity

Prof. Mark Bun — Fall 2019

Lecture Notes 15:

Introduction to Lifting

Reading.

- Rao-Yehudayoff, Chapter 8

Today we'll begin our discussion of lifting theorems. Sometimes you will see these referred to as "simulation theorems" or "hardness escalation theorems." Lifting is a quite general technique which takes lower bounds against a relatively weak, restricted computational model and generically "lifts" them to apply to a stronger computational model. In the context of communication complexity, a lifting theorem has three parts:

- A query model $\mathcal{C}^{\mathbf{dt}}$ used to capture the query complexity of evaluating a boolean function $f : \{0,1\}^n \to \{0,1\}$. Examples include (deterministic, nondeterministic, randomized) decision tree complexity, decision list complexity, and (nonnegative) polynomial degree.
- A two-party "gadget" $g : X \times Y \to \{0,1\}$. The gadget should balance two key properties. On one hand, it should be easy to compute using a low communication protocol. On the other hand, it should be hard for the parties to learn the value of $g(x,y)$ without communicating. Some examples of such gadgets include:
  - The Indexing gadget $\mathrm{IND}_m : \{0,1\}^m \times [m] \to \{0,1\}$ defined by $\mathrm{IND}_m(x,y) = x_y$. Here, think of $m = \mathrm{poly}(n)$.
  - The Inner Product mod 2 gadget $\mathrm{IP}_m : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}$ defined by $\mathrm{IP}_m(x,y) = \langle x, y \rangle \mod 2$. Here, think of $m = O(\log n)$.
  - The Pattern Matrix gadget $\mathrm{PM}_m : \{0,1\}^m \times ([m] \times \{0,1\})$ defined by $\mathrm{PM}_m(x,(i,b)) = x_i \oplus b$. Here, think of $m = O(1)$.
  - The XOR gadget $\mathrm{XOR} : \{0,1\} \times \{0,1\} \to \{0,1\}$.
- A target communication model $\mathcal{C}^{\mathbf{cc}}$ in which to prove a lower bound for a composed function of the form $f \circ g^n$ defined by $(f \circ g^n)(x_1, \dots, x_n, y_1, \dots, y_n) = f(g(x_1,y_1), \dots, g(x_n,y_n))$.

Here is the template that a (good) lifting theorem follows.

Theorem 1 (Lifting Template)**.** *Fix a query complexity measure $\mathcal{C}^{\mathbf{dt}}$, a communication complexity measure $\mathcal{C}^{\mathbf{cc}}$, and a gadget $g : X \times Y \to \{0,1\}$. Then for every function $f : \{0,1\}^n \to \{0,1\}$, we have*

$$\mathcal{C}^{\mathbf{cc}}(f \circ g^n) = \mathcal{C}^{\mathbf{dt}}(f) \cdot \Theta(\mathcal{C}^{\mathbf{cc}}(g)).$$

The upper bound $\mathcal{C}^{\mathbf{cc}}(f \circ g^n) = \mathcal{C}^{\mathbf{dt}}(f) \cdot O(\mathcal{C}^{\mathbf{cc}}(g))$ is almost always easy. The parties simulate a query algorithm for computing $f(z)$ and every time they need to query a bit z_i, they evaluate the gadget $g(x_i, y_i)$. The interesting direction is the lower bound. Often the lower bound is proved by a simulation argument. Given a communication protocol Π for $f \circ G$ (where $G = g^n$), we'd like to construct a query algorithm for the function f. On input $z \in \{0,1\}^n$, the query algorithm will attempt to simulate the transcript of $\Pi(x, y)$ for a random choice of $(x, y) \sim G^{-1}(z)$ using minimal queries to z. If the simulation is accurate, then accuracy of the protocol Π implies that this query algorithm accurately computes f.

The first concrete lifting theorem we'll see is the one from deterministic decision tree complexity to deterministic communication.

1 Decision Trees

A decision tree T is a simple, concrete model for computing a boolean function $f : \{0,1\}^n \to \{0,1\}$. It is represented by a rooted binary tree. Each non-leaf vertex is labeled by a variable x_i for $i \in [n]$, and its outgoing edges are labeled by 0 and 1. Each leaf is labeled by either 0 or 1. A decision tree induces a "determinstic query algorithm" for computing a boolean function. Starting at the root, if the value of the variable $x_i = 0$ the algorithm recurses to the 0-child, and if the value of $x_i = 1$ it recurses to the 1-child. The value of the function is the label of the leaf that the algorithm reaches.

A decision tree T computes a boolean function if $T(x) = f(x)$ for every $x \in \{0,1\}^n$. The cost of a decision tree is its depth, or equivalently, the number of input bits that need to be read in a worst-case input.

Definition 2. The deterministic decision tree complexity (or deterministic query complexity) of $f : \{0,1\}^n \to \{0,1\}$ is the minimum depth of a decision tree T which computes f. We denote this by $\mathbf{P}^{\mathbf{dt}}(f)$.

Example 3. The decision tree complexity of the OR_n function is n. Every boolean function has $\mathbf{P}^{\mathbf{dt}}(f) \leq n$ (why?). To see that this is tight for the OR function, let T be a decision tree of depth $\leq n - 1$. Evaluate T on an input x for which $x_i = 0$ for every index i queried. Then when T reaches a leaf, there is still some index i^* which has not been queried, but $\mathrm{OR}(0^n) \neq \mathrm{OR}(e_{i^*})$, so the decision tree disagrees with OR on at least one input.

A randomized decision tree $\mathcal{T}$ is a distribution over deterministic decision trees T. We say that $\mathcal{T}$ computes a function f if $\Pr_{T \sim \mathcal{T}}[T(x) = f(x)] \geq 2/3$ for every $x \in \{0,1\}^n$.

There are a few equivalent ways to define nondeterministic decision tree complexity $\mathbf{NP}^{\mathbf{dt}}$. Perhaps the cleanest way is as the minimum k for which f is computed by a DNF of width k. Here, a DNF is an OR of ANDs of variables and their negations and the width of the DNF is the maximum number of literals appearing in any term.

To see what DNF width has to do with decision trees, let's start with a "natural" definition of $\mathbf{NP}^{\mathbf{dt}}$ in terms of "nondeterministic decision trees." A nondeterministic decision tree is a collection $\mathcal{T} = \{T_1, \ldots, T_m\}$ of deterministic decision trees and we say that it computes f if $f(x) = 1$ iff there exists an i such that $T_i(x) = 1$. The cost of such a nondeterministic tree is the maximum cost of any of the constituent trees.

Note that any decision tree of depth k is in particular a width-k DNF. This is because we can express it as an OR over all paths from the root of the tree to 1-leaves of the AND of the variables

(or their negations) along that path. So a nondeterministic decision tree is an OR over width-k DNF and hence still a width-k DNF.

Conversely, a width-k DNF can be expressed as a nondeterministic decision tree where each constituent tree just computes a single term of the DNF.

(Note that the definition of nondeterministic decision tree cost above does not charge for the "witness size" $\log m$. Charging for this as well wouldn't affect this equivalence by more than a log factor, however, since a width-k DNF has at most $2^k \cdot \binom{n}{k}$ terms.)

2 Deterministic Lifting and an Application

Theorem 4. *There exists a gadget $g : X \times Y \to \{0,1\}$ such that for every $f : \{0,1\}^n \to \{0,1\}$,*

$$\mathcal{C}^{\mathbf{cc}}(f \circ g^n) = \mathcal{C}^{\mathbf{dt}}(f) \cdot \Theta(\mathcal{C}^{\mathbf{cc}}(g)).$$

Here, the gadget g can be taken to be

1. $g = \mathrm{IND}_m$ *for* $m = \mathrm{poly}(n)$, *or*
2. $g = \mathrm{IP}_m$ *for* $m = O(\log n)$.

The lifting theorem actually holds for quite general f. It could be a partial function, a search problem (a relation), or a function with non-boolean codomain. We'll begin the proof of the lifting theorem in the next lecture, but now let's see an example of what it can be used for.

The 1-partition number of a function $F : X \times Y \to \{0,1\}$ is the minimum number of (non-overlapping) monochromatic rectangles needed to partition $F^{-1}(1)$. Denote this by $\chi_1(F)$. Every deterministic protocol of cost c partitions its input space into 2^c monochromatic rectangles, so $\mathbf{P}^{\mathbf{cc}}(F) \geq \log \chi_1(F)$. But we've seen that not every partition of $X \times Y$ corresponds to an actual deterministic protocol. So could there be examples where this lower bound method is not tight? It turns out that there is a (tight) quadratic separation between deterministic communication complexity and partition number.

Theorem 5. *There exists a function $F : X \times Y \to \{0,1\}$ such that $\mathbf{P}^{\mathbf{cc}}(F) \geq \tilde{\Omega}(\log^2 \chi_1(F))$.*

The known constructions of such F are obtained by lifting. As a warmup, let's show the following relaxation where we replace the 1-partition number with the 1-cover number, which we recall characterizes $\mathbf{NP}^{\mathbf{cc}}$ communication complexity.

Theorem 6. *There exists a function $F : X \times Y \to \{0,1\}$ such that $\mathbf{P}^{\mathbf{cc}}(F) \geq \tilde{\Omega}((\mathbf{NP}^{\mathbf{cc}}(F))^2)$.*

Proof. We define a function $f : \{0,1\}^n \to \{0,1\}$ which exhibits a quadratic separation between its deterministic query complexity and its non-deterministic query complexity and lift this separation to the communication world.

The function f interprets its input as a $\sqrt{n} \times \sqrt{n}$ matrix and evaluates to 1 iff there exists a column which consists of all 1s. There is an $O(\sqrt{n})$-cost nondeterminstic query algorithm for this problem. Nondeterministically guess the all 1s column and then verify it with $O(\sqrt{n})$ queries. On the other hand, every deterministic query algorithm requires $\Omega(n)$ queries. To see this, feed a determinstic query algorithm 1s up until the last query in any given column, and then feed it a 0. Then the algorithm has to query all n bits of the matrix to determine the answer.

Now consider the function $F = f \circ g^n$ for either gadget g in Theorem 4. Then by the "easy" direction of nondeterministic lifting, we have $\mathbf{NP^{cc}}(F) = O(\sqrt{n} \log n)$. On the other hand, by the "hard" direction of deterministic lifting, we have $\mathbf{P^{cc}}(F) = \Omega(n \log n)$. This gives the asserted separation. □

To prove Theorem 5, we modify the function f so that the lifted function F has a small 1-partition number.

Modified definition of f. Again, f interprets its input as a $\sqrt{n} \times \sqrt{n}$ matrix. However, instead of each entry being a bit, each entries takes the form (m_{ij}, p_{ij}) where $p_{ij} \in [\sqrt{n}]^2 \cup \{\perp\}$ is a (possibly null) pointer to another entry of the matrix. The function f evaluates to 1 iff there exists a column j for which the following holds:

All bits in column j are 1s. Consider the lexicographically first non-null pointer in column j, and follow it for $k-1$ steps. All of the pointers encountered are non-null, have bits set to 0, and visit every column of the matrix (except for j).

The natural nondeterministic query algorithm computing this function is "unambiguous" in the sense that for every 1 input, there is exactly one accepting computation path. The unambiguous nondeterministic query algorithm for this problem lifts to an $\tilde{O}(\sqrt{n})$ 1-partition of F. Meanwhile, the deterministic query complexity of f is still n. To see this, call a query "critical" if it's the last in its column. On every non-critical query, answer $(1, \perp)$. On the first critical query, answer $(0, \perp)$. On every subsequent query, answer $(0, p)$ where p points to the last critical query.