**CS 229r Information Theory in Computer Science** Mar 22, 2016

# Lecture 15

*Lecturer: Madhu Sudan* *Scribe: Tiffany Wu*

## 1 Introduction

Today, we're going to talk about finding data structure problem lower bounds by reducing to communication complexity problems, for which we already have methods for establishing lower bounds. This follows the singular technique that Patrascu created for establishing lower bounds on a variety data structure problems that previously had a lot of different, more complex proofs.

We will introduce the data structure problem, and examine and begin to prove the lower bound for a particular one, called the Partial Match Retrieval problem (PMR). We show that any PMR problem can be reduced to an instance of a communication problem called Lopsided Set Disjointness (LSD) and Blocked-LSD, but won't explicitly prove today the lower bound for LSD.

## 2 Data Structure Problem

### 2.1 Definitions

**Definition 1.** *A data structure problem consists of a single function $f$ over 2 binary string inputs we want to compute:*

$$f : \{0,1\}^d \times \{0,1\}^m \to \{0,1\}$$

The first input $\in \{0,1\}^d$ is the query $q$ given to the function and the second input $\in \{0,1\}^m$ is the database $x$ that we are querying against. Often, we may write $f(q,x)$. Here, we only consider boolean functions, though one could imagine generalizing to more complex functions.

Here, we imagine that the database $x$ consists of $n$ $d$-bit strings (the same length as a query):

$$x \in \{0,1\}^m = \{\{0,1\}^d\}^n$$

and often the question is of the form "Is there an element in $x$ that's similar to our query in a particular way?"

**Example 1.** *Dictionary problem - asking if query string is in database.*

- *Data:* $x \subseteq \{0,1\}^d$
- *Query:* $q \in \{0,1\}^d$

$$f_{DICT}(q,x) = \begin{cases} 1 & \text{if } q \in x \\ 0 & \text{otherwise} \end{cases}$$

**Definition 2.** *A solution to a data structure problem consists of a data structure $D$, a query strategy $\{Q_i\}$, and a result function $g$ that outputs the answer given the queries:*

$$\begin{aligned} D &: \{0,1\}^m \to [w]^s \\ Q_i &: \{0,1\}^d \times [w]^{i-1} \to [s] \\ g &: \{0,1\}^d \times [w]^t \to \{0,1\} \end{aligned}$$

*where* $\forall x, q$,

$$f(q, x) = g(q, a_1, ...a_t)$$

*where*

$$a_i = [D(x)]_{Q(q, a_1, ..., a_{i-1})}$$

That is, we transform the database $x$ to a structure $D(x)$ with $s$ entries, each from an alphabet of $[w]$. We can think of $Q_i$ as deterimining what entry of the database we would like to have returned, given our query and all the previous responses $a_1, a_2, ..., a_{i-1} \in [w]$ to previous queries. The result is then given after $t$ queries by a function $g$, which takes as input $q, a_1, ...a_t$ and returns the correct answer, $f(q, x)$.

This is also known in the literature as **the cell probe model**.

## 2.2 What makes a good solution?

We evaluate the solution based on the parameters $(s, w, t)$. Ideally, we'd like for the size $s$ of the data structure to be as small as the input database ($O(n)$), $t$ to be almost constant ($O(1)$), and $w$ to be as small as possible.

What we don't care about in this model (but may be important for practical implementation purposes) are questions about:

- dynamic questions - updating the data structure with new data between queries
- time involved in preprocessing

Analyzing possible data structures for the dictionary problem gives the following parameters:

- Sorted Array using binary search - $w = |\{0, 1\}^d|$, $s = n$, $t = \lceil \log n \rceil$
- Trees - $w = |\{0, 1\}^d \times [s] \times [s]|$, $s = n$, $t = \lceil \log n \rceil$
- Hash Table - $w = |\{0, 1\}^{O(d)}|$, $s = O(n), t = O(1)$

One can see that the Hash Table is close to our ideal.

## 2.3 Holy Grail Problems

**Can we find problems that have really *large* lower bounds?** We want to find an explicit $f$ (e.g. any function in $P$) such that $\forall$ data structure, where $w = 2$, we have either

- $t = d^{\omega(1)}$ - number of queries is superpolynomial in length of query
- $s = m^{\omega(1)}$ - size of data structure is huge

**Can we find problems that have really *small* lower bounds?** Every problem has two obvious solutions.

- $t = m, s = m$ - simply store $x$, and ask for entire database.
- $s = 2^d, t = 1$ - preprocess this function for all inputs and store in database, so constant number of queries.

The question is, is it possible to get $s = m, t = O(1)$? As it turns out, this is an open problem.
As for these holy grail problems with large lower bounds, we consider the Partial Match Retrieval Problem as a potential candidate.

## 3 Partial Match Retrieval Problem (PMR)

**Definition 3.** *Let strings $p = p_1 p_2 ... p_d$ and $q = q_1, q_2, ... q_d$, where $p_i, q_i \in \{0,1\}$. We say $p \geq q$ ("p dominates q") iff $p_i \geq q_i$, $\forall i$.*

Informally, PMR seeks to compute whether any element in the database provided dominates the query. Formally,

$$x \subseteq \{0,1\}^d$$
$$q \in \{0,1\}^d$$
$$f_{PMR}(q,x) = \begin{cases} 1 & \text{if } \exists p \in x, p \geq q \\ 0 & \text{otherwise} \end{cases}$$

We want to prove the following theorem:

**Theorem 4.** *For every data structure, if $w \leq 2^{n^{1-\epsilon}}, \epsilon > 0$ and $s \geq 2^{\Omega(d/t)}$. That is, given that we cannot query the entire database (but we can make rather large queries), the storage must be "large."*

To prove this claim, we note that if we can reduce the communication complexity problem to a data structure problem, then a lower bound on communication complexity translates to a lower bound in our data structure problem.

Here are two possible reductions:

1. **Communication protocol $\Rightarrow$ Data structure:**
   Alice is given $q$ as input. Bob is given $x$ as input. Together, they wish to compute $f(x,q)$. There are $t$ rounds of communication, with this alternation:
   - Alice tells Bob (assuming data structure size is small) what cell she wishes to see: $\log s$ bits.
   - Bob tells Alice the contents of that cell: $\log w$ bits.
2. **Communication protocol for $f^{(k)}$ $\Rightarrow$ Data structure:**
   This is a slightly different flavor of repeated calculation, distinct from direct sum. In this case, Bob always has the same input.
   Alice is given $q_1, q_2, ... q_k$ as input. Bob is given $x$. We want to compute

   $$f^{(k)}(q_1, ..., q_k, x) = (f(q_1, x), ..., f(q_k, x))$$

   We can do this by allowing Alice and Bob to say more each round (querying in parallel), while still communication through $t$ rounds:
   - Alice tells Bob (assuming data structure size is small) what subsets of cell she wishes to see: $\log \binom{s}{k}$ bits.
   - Bob tells Alice the contents of those cells: $k \log w$ bits.

   Here, note that Alice is able to save a little by specifying the subset, which becomes $k \log \frac{s}{k}$ bits of communication per round, rather than $k \log s$, as we might expect.

We'll focus on reduction 1 in this course.

## 4 Reduce LSD to PMR

In order to establish a target communication problem to reduce from, let's examine the Lopsided Set Disjointness problem. LSD is similar to disjointness from before, but note that here Alice's input is much smaller than Bob's. (i.e. $d \leq n^{o(1)}$). Blocked-LSD is a variation on LSD has additional restrictions on Alice's input.

**Definition 5.** ***Lopsided Set Disjointness (LSD)***
*Alice gets $S \subset [N \cdot B]$ where $|S| = N$.*
*Bob gets $T \subseteq [N \cdot B]$ (no restrictions on $T$).*

*Decide if $S \cap T = \emptyset$:*

$$f(S,T) = \begin{cases} 1 & \textit{if } S \cap T \neq \emptyset \\ 0 & \textit{otherwise} \end{cases}$$

The trivial protocols where Alice or Bob sends over their entire inputs use $N \log B = \log \binom{NB}{N}$ and $NB$ bits, respectively.

**Definition 6.** ***Blocked LSD***
*Alice gets $S \subseteq [N] \times [B]$ such that $\forall i \in [N]$, $\exists!\, s_i$ s.t. $(i, s_i) \in S$.*
*Bob gets $T \subseteq [N] \times [B]$.*

*Decide if $S \cap T = \emptyset$.*

The general strategy then becomes to reduce from LSD to Block-LSD with a reasonably small amount of communication, and then reduce from Block-LSD to PMR. Then, LSD reduces to PMR, so lower bounds on LSD tell us some things about lower bounds on PMR.

## 4.1 Reduction from LSD to Block LSD

Note that the difference between LSD and Blocked-LSD is that in the latter, Alice's input is restricted to be in the form of

$$S = ((1, s_1), (2, s_2), ...(N, s_N))$$

. We can imagine that we have all possible $NB$ pairs displayed in an matrix with $B$ rows and $N$ columns, and both Alice and Bob have indicator matrices that displays which subsets they have as input.

To reduce from LSD and Blocked-LSD, we have Alice tell Bob how many points she has in each column using $O(N)$ bits of communication. Then both Alice and Bob will process their inputs. Bob duplicates each of his columns according to how many times Alice reports that column. Alice then expands the columns from left to right, deleting columns that don't have a point in them, and expanding the ones that have multiple into multiple columns with 1 point in each of them.

This costs $O(N)$ bits, which ends up getting absorbed into the overall communication complexity of Block-LSD.

## 4.2 Reduction from Block-LSD to PMR

Create an injective function $\phi : [B] \to [\binom{b}{b/2}]$ where $[\binom{b}{b/2}]$ represents the strings of length $b$ with exactly $b/2$ zeroes. We can do so by choosing a fixed $b$ large enough - as it turns out, $b = O(\log B)$. Note that

$$\phi(b) \geq \phi(c) \iff \phi(b) = \phi(c) \iff b = c$$

Alice gets input $S = \{(1, s_1), (2, s_2), ..., (N, s_N)\}$. We construct a query of length $Nb$:

$$q = \langle \phi(s_1), \phi(s_2), ....\phi(s_N) \rangle$$

Bob gets input $T$ of pairs. For every pair $(i, t)$, we create a vector of length $Nb$:

$$T = \{..., (i, t), ...\} \Rightarrow x = \{..., 1^{(i-1)b} \phi(t) 1^{(N-i)b}, ...\}$$

Here, the notation $1^{(i-1)b}\phi(t)1^{(N-i)b}$ denotes a vector with the first $(i-1)b$ bits set to 1, the next $b$ bits set to $\phi(t)$, and the last $(N-i)b$ bits set to 1. That is, if we divide up the $Nb$ vector into blocks of size $b$, the $i$th block is set to $\phi(t)$, and the vector is 1 everywhere else.

Then we claim that PMR will only return 1 (i.e. there is an element in $x$ that dominates $q$) iff $S \cap T \neq \emptyset$.

Suppose that there exists a vector $p \in x$ that dominates $q$, both of length $Nb$. Since $p$ is 1 everywhere except for the $i$th b-block, where it is $\phi(t)$ for some $t$, then we have that $p$ dominates $q$ iff $p$ dominates $q$ in the $i$th block (as it automatically dominates everywhere else). In the $i$th block, $p$ has the form $\phi(s_i)$.

From before, if $\phi(t)$ dominates $\phi(s_i)$, we must have that $t = s_i$. This is equivalent to saying that $S$ and $T$ had some overlapping element $(i, t)$. Thus, PMR returns 1 if $S, T$ not disjoint. The converse is easy to see as well.

## 5 Closing remarks

Note that in the above reduction, we have that $n = |T| = O(NB)$, and $d = Nb = O(N \log B)$.

We have a theorem that holds for LSD that we'll use to prove Lemma 8 (not this lecture though!), both of which we state now:

**Theorem 7.** *In every protocol, either Alice sends $\Omega(N \log B)$ bits or Bob sends $\Omega(NB^{1-\epsilon})$ bits.*

**Lemma 8.** *Either Alice sends $\Omega(d)$ bits to Bob, or Bob sends $\Omega(n^{1-\epsilon})$ bits to Alice.*