CS 154, Lecture 7: Communication Complexity

http://a2ru.org/
Communication Complexity

A model capturing one aspect of distributed computing.
Here focus on two parties: Alice and Bob

Function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$

Two inputs, $x \in \{0,1\}^*$ and $y \in \{0,1\}^*$

We assume $|x| = |y| = n$, Think of $n$ as HUGE

Alice only knows $x$, Bob only knows $y$

Goal: Compute $f(x, y)$ by communicating as few bits as possible between Alice and Bob

We do not count computation cost. We only care about the number of bits communicated.
Alice and Bob Have a Conversation

In every step: A bit is sent, which is a function of the party’s input and all the bits communicated so far.

Communication cost = number of bits communicated = 4 (in the example)
We assume Alice and Bob alternate in communicating, and the last bit sent is $f(x,y)$

More sophisticated models: separate number of rounds from number of bits
Def. A *protocol* for a function $f$ is a pair of functions $A, B : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1, \text{STOP}\}$ with the semantics:

On input $(x, y)$, let $r := 0$, $b_0 = \varepsilon$

While ($b_r \neq \text{STOP}$),

$r++$

If $r$ is odd, Alice sends $b_r = A(x, b_1 \cdots b_{r-1})$

else Bob sends $b_r = B(y, b_1 \cdots b_{r-1})$

Output $b_{r-1}$.

Number of *rounds* $= r - 1$
Def. The cost of a protocol \( P \) for \( f \) on \( n \)-bit strings is
\[
\max_{x, y \in \{0,1\}^n} \text{number of rounds in } P \text{ to compute } f(x, y)
\]

The communication complexity of \( f \) on \( n \)-bit strings is the minimum cost over all protocols for \( f \) on \( n \)-bit strings = the minimum number of rounds used by any protocol that computes \( f(x, y) \), over all \( n \)-bit \( x, y \).
Example. Let \( f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\} \) be arbitrary.

There is always a “trivial” protocol:
- Alice sends the bits of her \( x \) in odd rounds
- Bob sends the bits of his \( y \) in even rounds
After \( 2n \) rounds, they both know each other’s input!

The communication complexity of every \( f \) is at most \( 2n \)
Example: \[ \text{PARITY}(x, y) = \sum_i x_i + \sum_i y_i \mod 2. \]

What’s a good protocol for computing PARITY?

Alice sends \( b_1 = (\sum_i x_i \mod 2) \)
Bob sends \( b_2 = (b_1 + \sum_i y_i \mod 2) \). Alice stops.

The communication complexity of PARITY is 2
Example: \( \text{MAJORITY}(x, y) = \) most frequent bit in \( xy \)

What’s a good protocol for computing MAJORITY?

Alice sends \( N_x = \) number of 1s in \( x \)
Bob computes \( N_y = \) number of 1s in \( y \),
  sends 1 iff \( N_x + N_y \) is greater than \( (|x| + |y|)/2 = n \)

Communication complexity of MAJORITY is \( O(\log n) \)
Example: $EQUALS(x, y) = 1 \iff x = y$

What’s a good protocol for computing $EQUALS$ ???

Communication complexity of $EQUALS$ is at most $2n$
Connection to Streaming and DFAs

Let $L \subseteq \{0,1\}^*$
Def. $f_L: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$
for $x, y$ with $|x| = |y|$ as:

$$f_L(x, y) = 1 \iff xy \in L$$

Examples:

$L = \{ x \mid x \text{ has an odd number of 1s} \}$

$\Rightarrow f_L(x, y) = \text{PARITY}(x,y) = \sum_i x_i + \sum_i y_i \mod 2$

$L = \{ x \mid x \text{ has more 1s than 0s} \}$

$\Rightarrow f_L(x, y) = \text{MAJORITY}(x,y)$

$L = \{ xx \mid x \in \{0,1\}^* \}$

$\Rightarrow f_L(x, y) = \text{EQUALS}(x,y)$
Connection to Streaming and DFAs

Let $L \subseteq \{0,1\}^*$
Def. $f_L: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$
for $x, y$ with $|x| = |y|$ as:
\[ f_L(x, y) = 1 \iff xy \in L \]

Theorem: If $L$ has a streaming algorithm using $\leq s$ space, then the comm. complexity of $f_L$ is at most $O(s)$.

Proof: Alice runs streaming algorithm $A$ on $x$.
Sends the memory content of $A$: this is $s$ bits of space
Bob starts up $A$ with that memory content, runs $A$ on $y$. Gets an output bit, sends to Alice.
Connection to Streaming and DFAs

Let \( L \subseteq \{0,1\}^\ast \)  
Def. \( f_L(x,y) = 1 \iff xy \in L \)

Theorem: If \( L \) has a streaming algorithm using \( \leq s \) space, then the comm. complexity of \( f_L \) is at most \( O(s) \).

Corollary: For every regular language \( L \), the comm. complexity of \( f_L \) is \( O(1) \).

Example: Comm. Complexity of PARITY is \( O(1) \)

Corollary: Comm. Complexity of MAJORITY is \( O(\log n) \), because there’s a streaming algorithm for \( \{ x : x \text{ has more 1's than 0's} \} \) with \( O(\log n) \) space

What about the Comm. Complexity of EQUALS?
Communication Complexity of EQUALS

Theorem: The comm. complexity of EQUALS is $\Theta(n)$. In particular, every protocol for EQUALS needs $\geq n$ bits of communication.

No communication protocol can do much better than “send your whole input”!

Corollary: $L = \{ww \mid w \in \{0,1\}^*\}$ is not regular.

Moreover, every streaming algorithm for $L$ needs $c \cdot n$ bits of memory, for some constant $c > 0$. 
Theorem: The comm. complexity of EQUALS is $\Theta(n)$. In particular, every protocol for EQUALS needs $\geq n$ bits of communication.

Idea: Consider all possible ways $A$ & $B$ can communicate.

Definition: The communication pattern of a protocol on inputs $(x, y)$ is the sequence of bits that Alice & Bob send.

Pattern: 0110
Communication Complexity of EQUALS

Theorem: The communication complexity of EQUALS is $\Theta(n)$. In particular, every protocol for EQUALS needs $\geq n$ bits of communication.

Proof: By contradiction. Suppose CC of EQUALS is $\leq n - 1$. Then there are $\leq 2^{n-1}$ possible communication patterns of that protocol, over all pairs of inputs $(x, y)$ with $n$ bits each.

Claim: There are $x \neq y$ such that on $(x, x)$ and on $(y, y)$, the protocol uses the same pattern $P$.

Now, what is the communication pattern on $(x, y)$? This pattern is also $P$ (WHY?)
So Alice & Bob output the same bit on $(x, y)$ and $(x, x)$.
But $\text{EQUALS}(x, y) = 0$ and $\text{EQUALS}(x, x) = 1$. Contradiction!
Randomized Protocols Help!

EQUALS needs $cn$ bits of communication, but...

**Theorem:** For all $(x, y)$ of $n$ bits each, there is a *randomized* protocol for EQUALS$(x, y)$ using only $O(\log n)$ bits of communication, which works with probability 99.9%!

Use Error Correcting Codes … E.g:

- Alice picks a random prime number $p$ between 2 and $n^2$.
- She sends $p$ and her string $x$ modulo $p$.
- This is a number between 0 and $n^2$, takes $O(\log n)$ bits to send.
- Bob checks whether $y = x$ modulo $p$. Sends output bit.

Why does it work (with high probability)?
Communication Complexity: Powerful Tool (we seen just a tiny demonstration).

Communication Complexity, Streaming Algorithms and Regular languages – connected.

Randomness – could be a useful resource of computation (II)
Questions?