Lecture 8: More on Communication Complexity, Start up Turing Machines
Announcements:
- Dylan’s office hours? Vote!
- Midterm: March 19
Communication Complexity

A theoretical model of distributed computing

- **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

- **Two computers**: Alice and Bob
  - 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.
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 $b_r = A(x, b_1 \ldots b_{r-1})$

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

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, $cc(f)$, is *minimum* cost over *all protocols* for $f$ on $n$-bit strings
\[
= \text{ the minimum number of rounds used by any protocol computing } f(x, y), \text{ over all } n\text{-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 whatever bit he wants in even rounds
After $2n - 1$ rounds, Bob knows $x$ and can send $f(x, y)$

Proposition: For every $f$, $\text{cc}(f) \leq 2n$
Example. $\text{EQUALS}(x, y) = 1 \iff x = y$

What’s a good protocol for computing $\text{EQUALS}$?

Communication complexity of $\text{EQUALS}$ 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. $f_L : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ 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 $cc(f_L) \leq 4s + 4$.

Proof Idea: Alice runs streaming algorithm $A$ on $x$, reaches a memory state $m$. She sends $m$ to Bob in $4s+3$ rounds. Then Bob starts up $A$ from state $m$, runs $A$ on $y$. Gets an output bit, sends bit to Alice.

(...why $4s+3$ rounds?)
Connection to Streaming and DFAs

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

Theorem: If \( L \) has a streaming algorithm using \( \leq s \) space,  
then \( \text{cc}(f_L) \leq 4s + 4 \).

Corollary: For every regular \( L \), \( \text{cc}(f_L) \leq O(1) \).

Example: \( \text{cc}(\text{PARITY}) = 2 \)

Corollary: \( \text{cc}(\text{MAJORITY}) \leq O(\log n) \),  
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 \text{EQUALS}?
Communication Complexity of EQUALS

Theorem: \( cc(\text{EQUALS}) = \Theta(n) \).
In particular, every communication protocol for EQUALS needs to send \( \geq n \) bits.

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

Corollary: \( L = \{xx \mid x \in \{0,1\}^*\} \) is not regular.

Moreover, every streaming algorithm for \( L \) needs \( c \ n \) bits of memory, for some constant \( c > 0 \)!
Communication Complexity of EQUALS

Theorem: $cc(EQUALS) = \Theta(n)$. 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 Alice & Bob send.

Pattern: 0110
Key Lemma: If \((x, y)\) and \((x', y')\) have the same pattern \(P\) in a protocol, then \((x, y')\) and \((x', y)\) also have pattern \(P\)
Key Lemma: If \((x, y)\) and \((x', y')\) have the same pattern \(P\) in a protocol, then \((x, y')\) and \((x', y)\) also have pattern \(P\)
Key Lemma: If \((x, y)\) and \((x', y')\) have the same pattern \(P\) in a protocol, then \((x, y')\) and \((x', y)\) also have pattern \(P\).
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.

Proof: By contradiction. Suppose $cc($EQUALS$) \leq n - 1$. 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 $P$.

By the Key Lemma, $(x, y)$ and $(y, x)$ also use pattern $P$.

So Alice & Bob output the same bit on $(x, y)$ and $(x, x)$. But $EQUALS(x, y) = 0$ and $EQUALS(x, x) = 1$. Contradiction!
Randomized Protocols Help!

EQUALS needs $\geq n$ bits of communication, but...

Theorem: For all $(x, y)$ of $n$ bits each, there is a randomized protocol for $\text{EQUALS}(x, y)$ using only $O(\log n)$ bits of communication, which is correct with probability 99.9%!
Turing Machines
Turing Machine (1936)

In each step:
- Reads a symbol
- Writes a symbol
- Changes state
- Moves Left or Right

FINITE STATE CONTROL

INFINITE REWRITABLE TAPE
ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The “computable” numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbersome technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth in another. This will include a development...
GREAT. A WAREHOUSE FILLED WITH MILES AND MILES OF REWRITABLE TAPE! WHAT ARE WE EVER GOING TO DO WITH THIS, ALAN?

...ALAN?

And thus the Turing Machine was born.

Turing Machines versus DFAs

The input is written on an infinite tape
  with “blank” symbols after the input

The “tape head” can move right and left

The TM can both write to and read from the tape,
  and can write symbols that aren’t part of input

Accept and Reject take immediate effect
Deciding the language $L = \{ w#w \mid w \in \{0,1\}^* \}$

1. If there’s no # on the tape (or more than one #), reject.
2. While there is a bit to the left of #,
   Replace the first bit with X, and check if the first bit $b$ to the right of the # is identical. (If not, reject.)
   Replace that bit $b$ with an X too.
3. If there’s a bit to the right of #, then reject else accept
Definition: A Turing Machine is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$
This Turing machine decides the language $\{0\}$.
This Turing machine recognizes the language \{0\}

\(\Sigma = \{0\}\)

\(\Box = \text{“blank”}\)
Turing Machine Configurations

corresponds to the configuration:

\[ q_011010001100 \in (Q \cup \Gamma)^* \]
Turing Machine Configurations

\[ 0q_110100011001110 \in (Q \cup \Gamma)^* \]
Turing Machine Configurations

 corresponds to the configuration:

\[0000011110q_7 \square \in (Q \cup \Gamma)^*\]
Defining Acceptance and Rejection for TMs

Let $C_1$ and $C_2$ be configurations of a TM $M$.

Definition. $C_1$ yields $C_2$ if $M$ is in configuration $C_2$ after running $M$ in configuration $C_1$ for one step.

Example. Suppose $\delta(q_1, b) = (q_2, c, L)$.
Then $aq_1bb$ yields $q_2acb$.
Suppose $\delta(q_1, a) = (q_2, c, R)$.
Then $abq_1a$ yields $abcq_2\square$.

Let $w \in \Sigma^*$ and $M$ be a Turing machine.
$M$ accepts $w$ if there are configs $C_0, C_1, \ldots, C_k$, s.t.
- $C_0 = q_0w$ [the initial configuration]
- $C_i$ yields $C_{i+1}$ for $i = 0, \ldots, k-1$, and
- $C_k$ contains the accept state $q_{\text{accept}}$. 
A TM $M$ recognizes a language $L$ if $M$ accepts exactly those strings in $L$.

A language $L$ is recognizable (a.k.a. recursively enumerable) if some TM recognizes $L$.

A TM $M$ decides a language $L$ if $M$ accepts all strings in $L$ and rejects all strings not in $L$.

A language $L$ is decidable (a.k.a. recursive) if some TM decides $L$. 