CS5275 Lecture 11: Communication Complexity

Jonathan Scarlett

Acknowledgment. The first version of these notes was prepared by Yao Tong and T.-Duy Nguyen-Hien for a CS6235 assignment.

Useful references:

- Blog post: The complexity of communication (Math ∩ Programming)
- TCS toolkit style videos:
  - Basics: `https://www.youtube.com/watch?v=mQQ36cDnmR8`
  - Deterministic CC: `https://www.youtube.com/watch?v=zFHWmmThdT4`
  - Randomized CC: `https://www.youtube.com/watch?v=LRef5a88uZQ`
- Lecture notes
  - CMU CS15-859T
  - Rutgers CS514
  - UCSD CSE291
- Textbooks:
  - A. Rao and A. Yehudayoff, *Communication Complexity and Applications*. Cambridge University Press, 2020
  - A. A. Razborov, "Communication complexity," in *An Invitation to Mathematics: From Competitions to Research*. Springer, 2011, pp. 97–117
  - T. Roughgarden *et al.*, *Communication complexity (for algorithm designers)*. Now Publishers, Inc., 2016, vol. 11, no. 3–4
  - T. Lee, A. Shraibman *et al.*, "Lower bounds in communication complexity," *Foundations and Trends® in Theoretical Computer Science*, vol. 3, no. 4, pp. 263–399, 2009

Categorization of material:

- Core material: Sections 1–4
- Extra material: Section 5 (Randomized CC), Section 6 (Applications)

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Overview

Communication complexity was introduced by Yao in the late 1970s [5], and studies the minimum number of bits that two (or more) parties must exchange in order to cooperatively accomplish some task based on their inputs. This is abstracted by the problem of *computing a function* $f(x, y)$, where one party knows x and the other knows y.

Communication complexity has some conceptual similarity and even technical overlap with the communication problem we mentioned in the information theory lecture, but with some major differences:

- Most importantly, the goal is to compute a function, not to send a message.
- The communication is typically two-way rather than one-way.
- We usually consider the number of bits with respect to the "worst-case" x and y (possibly restricted to lie in certain sets), rather than putting probabilistic models on them.
- We usually consider the noiseless exchange of bits, rather than noisy communication channels.
- We usually settle on understanding scaling laws in terms of big-O notation (e.g., $O(n)$ vs. $O(\log n)$ when x, y are in $\{0,1\}^n$), rather than trying to establish the precise constants.

Having said this, information theory *does* play a major role in several proofs on communication complexity, particularly for randomized protocols (such proofs will be beyond the scope of this lecture).

Perhaps the main use of communication complexity is not to directly model the real world, but rather, to provide a useful abstraction that connects to extensive other fields throughout theoretical computer science. In particular, when a problem in communication complexity is known to be "hard" in the sense of requiring a large number of bits to be exchanged, reductions can be used to show other problems to be hard, e.g.:

- Lower bounds on the storage required in streaming algorithms;
- Lower bounds on query complexity in learning problems;
- Lower bounds on sample complexity in data-centric problems.

We will give some examples in Section 6.

Deterministic vs. randomized protocols. One of the key defining features of communication complexity problems is *determinsitic* vs. *randomized*. For deterministic protocols, both parties are required to act deterministically and produce the correct output. For randomized protocols, the parties by make random decisions, and some probability of error is allowed (e.g., output the correct answer with probability 0.9). This distinction can have a major impact on the amount of communication required, e.g. $O(n)$ deterministic vs. $O(\log n)$ randomized (as we will see later).

Generally speaking, randomized communication complexity turns out to be more useful when it comes to forming reductions to other problems in computer science. However, deterministic communication complexity is much more suitable and digestible for an introductory lecture on the topic. Thus, we will focus mainly on the deterministic case in Sections 2–4, then briefly look at the randomized case in Section 5, before giving examples of reductions (for both deterministic and randomized settings) in Section 6.

2 Problem Setup (for Deterministic Protocols)

Assume there are two parties, Alice and Bob, with Alice having access to an input $x \in \mathcal{X}$ and Bob having access to another input $y \in \mathcal{Y}$. Their collective objective is to compute the function value $f(x, y)$, where the function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is mutually known. The main obstacle is that the two parties are unaware of each other's inputs (*i.e.*, Alice knows *only* x and Bob knows *only* y). Thus, to compute the value $f(x, y)$, they will need to communicate with each other, following a fixed protocol agreed upon beforehand.

- Note: Unless stated otherwise, we will take $\mathcal{X} = \{0, 1\}^n$ and $\mathcal{Y} = \{0, 1\}^n$ (i.e., the inputs are length-n bit strings), and $\mathcal{Z} = \{0, 1\}$ (i.e., the output is binary), which is the most natural starting point.
- Note: Since we are specifically interested in communication, we allow Alice and Bob to individually have unlimited storage and computation (though keeping these low is still considered preferable).

A **deterministic communication protocol** is a protocol π that determines which player sends the next message and what they send to the other player. Each message sent may depend on the player's own knowledge (including their input x or y) and the *communication history* (all the previously communicated bits up to that point). It is common for the final message transmitted to represent the "final output" $\pi(x, y)$, which equals $f(x, y)$ if the protocol successfully computes f. However, we *do not strictly require that to be the case.*[1] Instead, we consider the following more general notion.

Definition. *We say that a protocol π computes a function f if, for all x and y, after communication ends, the value $f(x, y)$ can deterministically be computed by both Alice and Bob given the information they received during the protocol (and their own input).*

For any given function f, there will exist multiple protocols achieving this goal. As already hinted, we specifically interested in protocols that require as little communication as possible.

Definition. *The communication cost of a protocol is the total maximum number of bits exchanged by the two parties, where the maximum is taken with respect to all possible input pairs (x, y). The communication complexity of a function f is the smallest such communication cost among all protocols.*

We start with some simple examples to better understand these terms.

2.1 Example 1: OR Function

Assume Alice and Bob are two parties who want to compute the OR function of their binary inputs without revealing their entire inputs to each other. Let x be a binary string of length n held by Alice and y be another binary string of length n held by Bob. The function $f : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$ they want to compute is

$$\mathrm{OR}(x, y) = x_1 \vee y_1 \vee x_2 \vee y_2 \vee \ldots \vee x_n \vee y_n,$$

where x_i and y_i are the individual bits of x and y respectively, and $\vee$ represents the logical OR.

[1] Note, however, that requiring this would only cost at most 1 extra bit if the output of f is binary.

Protocol 1 A naive protocol would be that Alice sends her entire string x to Bob. Then Bob computes the OR function directly and sends the result back to Alice. Hence, the *cost* of this protocol is $n+1$ bits.

In fact, for arbitrary $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, there is always a "trivial" protocol for f that uses $n+1$ rounds. Alice simply sends her entire n-bit x to Bob. Since Bob knows x, he can send $f(x,y)$ in the next round. Formally, defining $\mathrm{cc}(f)$ as the communication complexity of f, we have the following.

Proposition 2.1. *For every* $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$*, we have* $\mathrm{cc}(f) \leq n+1$.

When n is very large, a protocol using $O(n)$ bits of communication is highly undesirable. Is there any better protocol?

Protocol 2 In this OR function example, a better protocol is that Alice firstly computes $g(x) = x_1 \vee x_2 \vee \ldots \vee x_n$ and sends $g(x)$ to Bob. Then, Bob computes $z = y_1 \vee y_2 \vee \ldots \vee y_n \vee g(x)$ and sends z back to Alice. Therefore, the cost of this protocol is just 2 bits.

2.2 Example 2: EQUALS Function

Another important and useful example is the function EQUALS, defined by

$$\begin{aligned}\mathrm{EQUALS}(x,y) &= \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \\ &= \mathbb{1}(x_1 = y_1) \wedge \mathbb{1}(x_2 = y_2) \wedge \ldots \wedge \mathbb{1}(x_n = y_n),\end{aligned}$$

where $\mathbb{1}$ denotes the indicator function, and $\wedge$ is the logical AND operation.

- Note: This is not just a "toy" theoretical example, but is actually very practically motivated – it corresponds to two parties each having their own version of a file, and they would like to communicate to check whether the two versions are identical. For example, they could have become out-of-date due to updates, database read/write errors, etc.

From Proposition 2.1, we know $n+1$ bits of communication suffice, achieved via Alice sending her entire string. A natural question is then: **Is there room to improve the naive $n+1$ bit protocol?** In Section 4, we will demonstrate that $n+1$ is in fact optimal for deterministic protocols, meaning EQUALS is a much "harder" function than OR. To establish this, we will study lower bounds on communication complexity.

- Note: Jumping ahead slightly, this will turn out to be an example where *randomization helps a lot.* In Section 5, we will discuss simple randomized protocols with communication cost $O(\log n)$ or even $O(1)$, depending on the model of randomness used.

With the over-arching goal of answering the question "**For a given function f, what is its communication complexity?**", we proceed to introduce some key concepts.

3 Representations and Properties of Deterministic Protocols

3.1 Protocol Trees

A natural way to represent a general protocol is using a binary tree in which each branch is based on whether a certain bit computed and sent (by Alice or Bob) is 0 or 1. An example is shown at the top of the next

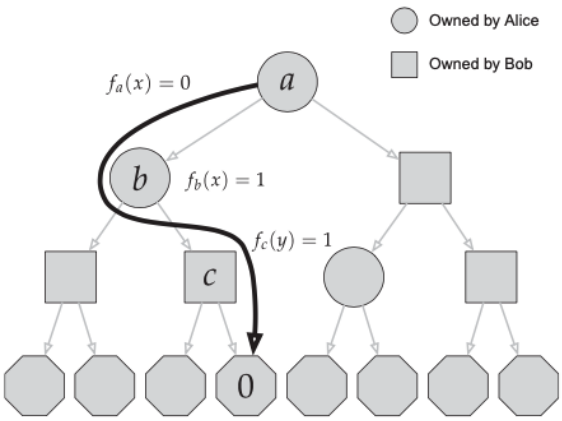


Figure 1: An execution of a protocol in a protocol tree. [1]

page. Elaborating on this example:

- The first bit $f_a(x)$ is computed by Alice and sent to Bob.
- Taking the left branch indicates that $f_a(x) = 0$, leading to the node labeled b. In this case, Alice is responsible for computing and sending another bit $f_b(x)$.
- Taking the right branch indicates that $f_b(x) = 1$, leading to the node labeled c. In this case, Bob is responsible for computing $f_c(y)$ and sending it to Alice.
- Taking the right branch indicates that $f_c(y) = 1$, and at this stage Alice and Bob declare that the protocol output is 0.

This naturally extends to larger trees (i.e., longer protocols). In the above figure, it happens that an output is always decided after sending 3 bits, but in general this could vary depending on which branches are taken.

For any tree, define the *depth* to be the distance (number of branches) of the furthest leaf from the root. With the tree representation, we can immediately deduce the following: **The deterministic communication complexity equals the smallest depth among all protocol trees that compute $f(x, y)$.**

3.2 Rectangles

It is often useful to visualize the function $f(x, y)$ as a matrix with $|\mathcal{X}|$ rows, $|\mathcal{Y}|$ columns, and (x, y)-th entry equal to $f(x, y)$. This will be done both here and when we discuss 'Rank' below.

- **Caution**: If $\mathcal{X} = \mathcal{Y} = \{0, 1\}^n$, then the matrix size is $2^n \times 2^n$, not to be confused with $n \times n$. In this case, the matrix is indexed by *length-n strings*, rather than integers like you might be more used to.

Given a subset $A \subseteq \mathcal{X}$ of the row indices and a subset $B \subseteq \mathcal{Y}$ of column indices, the set $A \times B$ is called a *combinatorial rectangle*; in the following we omit the word "combinatorial" and just call these *rectangles*. Importantly, A and B are not necessarily consecutive indices, so the "rectangle" can have gaps in it, e.g.:

The key reason for focusing on rectangles is that the sharing of bits in a protocol naturally leads to "narrowing down" the location of (x, y) to lie withing rectangles. For example, if Alice tells Bob that $x \in A$ and Bob tells Alice that $y \in B$, then both parties have a mutual understanding that (x, y) lies in the rectangle $A \times B$.

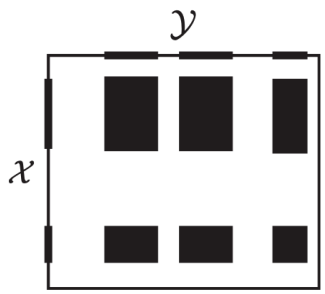


Figure 2: An example of a (combinatorial) rectangle with non-consecutive row and column indcies. [1]

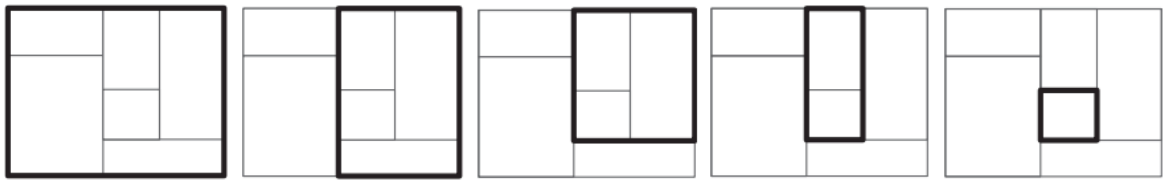


Figure 3: Example of the evolution of a rectangle (taken from [1]). When v is the root node in the protocol tree, X_v is the entire space $\mathcal{X} \times \mathcal{Y}$ (left figure). Each time a bit is communicated by Alice (resp., Bob), the rectangle is partitioned horizontally (resp., vertically), and collectively Alice and Bob both know that (x, y) lies in the rectangle shown.

Taking this idea further, it in fact holds that ***every protocol can be represented using rectangles***. To see this, consider an arbitrary protocol tree, and for each node v, define the subset $S_v \subseteq \mathcal{X} \times \mathcal{Y}$ to contain all input pairs (x, y) that would lead the protocol to node v during its execution, and let

$$X_v = \{x \in \mathcal{X} : \exists y \in \mathcal{Y} \text{ such that } (x, y) \in S_v\}$$
$$Y_v = \{y \in \mathcal{Y} : \exists x \in \mathcal{X} \text{ such that } (x, y) \in S_v\}.$$

The following lemma shows that $S_v = X_v \times Y_v$ (thus being a rectangle), and Figure 3 gives an illustration of this and how branching in the protocol tree amounts to doing a horizontal (Alice) or vertical (Bob) partitioning of the previous rectangle.

Lemma 3.1 (**Lemma 1.4 in [1]**). *For every node v in the protocol tree, the set S_v (defined above) is a rectangle R_v with $R_v = X_v \times Y_v$. Moreover, the rectangles corresponding to the leaves of the protocol tree form a partition of $\mathcal{X} \times \mathcal{Y}$.*

Proof. The lemma follows by induction. For the root node r, we have $R_r = \mathcal{X} \times \mathcal{Y}$, so indeed the lemma holds. Now consider an arbitrary node v such that $R_v = X_v \times Y_v$, and let u, w be the children of v in the protocol tree. Without loss of generality, we can suppose Alice is the one sending a bit at node v, that u is the subsequent node when 0 is sent, and that w is the subsequent node when 1 is sent. In this case, $Y_v = Y_u = Y_w$ since Bob's input is not used in this step. Define:

$$X_u = \{x \in X_v : f_v(x) = 0\}$$
$$X_w = \{x \in X_v : f_v(x) = 1\},$$

where $f_v(x)$ is Alice's sent bit corresponding to node v. It follows that X_u and X_w form a partition of X_v,

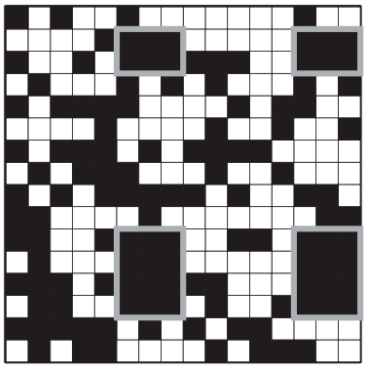
and $R_u = X_u \times Y_u$ and $R_w = X_w \times Y_w$ form a partition of R_v. Having established the base case and the induction step, the desired claim follows for all v, and we deduce that the rectangles corresponding to the leaves form a partition of $\mathcal{X} \times \mathcal{Y}$. □

3.2.1 Monochromatic Rectangles

To utilize rectangles as building blocks, we need to further introduce the concept of *monochromatic rectangles*.

Definition 3.1 (**Monochromatic Rectangles [1]**). *Given $f : \mathcal{X} \times \mathcal{Y} \to Z$ and a constant $z \in Z$, a rectangle R is z-monochromatic if $f(x, y) = z$ for all $(x, y) \in R$.*

The following figure gives an example of a Boolean function $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and one (combinatorial) rectangle that is 1-monochromatic (black):



Recalling that each leaf node in the protocol has an associated rectangle, and that reaching the leaf node amounts to outputting a specific value, we immediately deduce the following.

Fact 1. *If a protocol π computes a function $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and v is a leaf node in the protocol tree of π, then the rectangle R_v corresponding to node v is a monochromatic rectangle with respect to f.*

Since the number of leaves in a rooted binary tree of depth d is at most 2^d, we deduce the following simple but important result.

Theorem 3.2 (**Theorem 1.6 in [1]**). *If the communication complexity of $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ is c, then $\mathcal{X} \times \mathcal{Y}$ can be partitioned into at most 2^c monochromatic rectangles with respect to f.*

A particularly useful consequence of this result is the contrapositive statement: **If the function f is such that we cannot form a partition of $\mathcal{X} \times \mathcal{Y}$ using at most 2^c monochromatic rectangles, then the communication complexity must exceed c.** This will be one of the methods for deriving lower bounds on communication complexity in Section 4.

3.2.2 From Rectangles to Protocols

Theorem 3.2 shows that with communication complexity c, we will have at most 2^c monochromatic rectangles. This raises the natural question of whether we can state a similar kind of reverse relationship – if we can form a partition of f using a small number of monochromatic rectangles, can we conclude that the communication complexity is small?

A subtle issue here is that *not every partition of rectangles can be induced by a protocol*; see Figure 4 for a simple counter-example. A protocol always amounts to sequentially dividing a larger rectangle into smaller ones, whereas this "windmill-like" behavior does not fit that description. Nevertheless, it turns out that

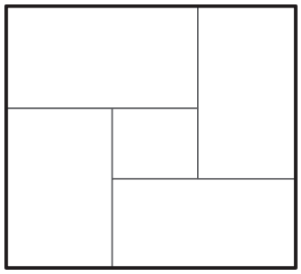


Figure 4: A partition into rectangles that cannot be realized by any protocol. [2]

when there are at most 2^c monochromatic rectangles, we can come up with a protocol of length $O(c^2)$, which is not too much higher than the "ideal" $O(c)$ that would match Theorem 3.2.[2] This is stated as follows.

Theorem 3.3. *Suppose that a function f is such that there exists a partitioning into at most 2^c monochromatic rectangles for some integer $c > 0$. Then there exists a protocol of length $O(c^2)$ that computes f.*

The (optional) proof is given in Appendix B.

3.3 Rank

The third concept we introduce is *rank*, a fundamental notion in linear algebra that can be a useful tool for bounding the communication complexity.

We continue with the notion of representing a function with inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ as an $|\mathcal{X}| \times |\mathcal{Y}|$ matrix, but we now specifically denote that matrix by M (i.e., $M_{xy} = f(x, y)$). We focus mainly on the case that $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$ and binary-valued functions (i.e., each entry of M is 0 or 1). It turns out that the *rank of this matrix* has fundamental connections to communication complexity.

As a reminder, the following fact states some equivalent definitions of matrix rank.

Fact 2. *For any matrix M (not necessarily square), the following are equivalent:*

1. $\text{rank}(M) = r$.
2. *r is the maximum size of a set of linearly independent columns in M. (Or similarly for rows.)*
3. *r is the smallest number such that M can be expressed as $M = AB$, where A is an $m \times r$ matrix, and B is an $r \times n$ matrix.*
4. *r is the smallest number such that M can be expressed as the sum of r matrices of rank 1.*

3.3.1 Communication Complexity and Rank

The following result indicates that *low rank implies low communication complexity.* Later, in Section 4.4, we will further establish that the communication complexity is *lower bounded* by $\log(\text{rank}(M) + 1)$, and discuss further results narrowing the gap between the upper and lower bounds.

[2] Recent work has shown that the square is in fact unavoidable in general [6].

Theorem 3.4 (**Theorem 2.2 in [1]**). *Consider any $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, and let M be the associated binary matrix. Then, the communication complexity of f is at most* $\text{rank}(M) + 1$.

We proceed with the proof. The idea is to use formatulion #3 of rank from Fact 2, meaning that Alice and Bob can employ a factorization $M = AB$, where A is an $m \times r$ matrix and B is an $r \times n$ matrix. Given that Alice has $x \in X$ and Bob has $y \in Y$, let e_x be the standard unit column vector with a 1 in the position corresponding to x, and similarly, let e_y be the one for y. The function can then be computed as follows:

$$f(x, y) = e_x^T M e_y = e_x^T A B e_y.$$

As a result, Alice only needs to send Bob $e_x^T A$, after which Bob multiplies $e_x^T A$ with Be_y and returns the result $M_{xy} = f(x, y)$ to Alice (requiring 1 bit).

As stated, we face the issue that A may be a non-binary matrix, making it unclear how much communication is required to send $e_x^T A$. However, it turns out that we can always ensure that A is binary, meaning we can send $e_x^T A$ using r bits, for a total of $r + 1$ (as desired). This is stated in the following claim.

Claim 1. *It is always possible to factorize a Boolean matrix M by $M = AB$, where A is a binary $m \times r$ matrix and B is a (possibly non-binary) $r \times n$ matrix.*

Proof. Construct A as an $m \times r$ matrix using r columns from M that are linearly independent. Considering that M is a binary matrix, A is a binary matrix as well. Given that r represents the rank of M, every other column in M can be expressed as a linear combination of these r columns. Hence, there exists an $r \times n$ matrix B such that $M = AB$. □

This completes the proof of Theorem 3.4.

4 Lower Bounds for Deterministic Protocols

When studying communication complexity, the natural first step is to devise various protocols, establish their communication cost, and deduce upper bounds on the communication complexity. However, unless we manage to bring the communication cost to something that clearly can't be improved (e.g., $O(1)$ scaling), we might always be left wondering whether we can do better. It turns out that the answer is often "No, we cannot do (much) better", and the way to establish this is by establishing *lower bounds on communication complexity*. In other words, we would like to establish that any protocol, no matter how "intelligent", must always have some minimum amount of communication.

- Of course, the closer the upper and lower bounds are, the better we understand the communication complexity. It's relatively uncommon to get an "exact" answer, and more common to settle on getting the correct scaling (e.g., $\Theta(n)$ or $\Theta(\log n)$).
- For communication complexity specifically, lower bounds are arguably even more important than upper bounds, because most applications of communication complexity in other areas of computer science are in getting lower bounds via reductions.

4.1 Most Functions Require High Communication

As a starting point, we show that *most functions require a high amount of communication*. Specifically, in the case that $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, the following theorem shows that almost all functions require at

least $n-1$ bits of communication to compute.

Theorem 4.1. *Consider the case that $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ (i.e., both inputs are n-bit and the output is 1-bit). Only a vanishingly small (as $n \to \infty$) fraction of such functions can be computed with $n-2$ bits (or fewer) of communication using deterministic protocols.*

Proof. In accordance with the theorem statement, we consider protocol trees of depth c, where $c \leq n-2$. In general, different leaf nodes may appear at different levels in the tree, but if that's the case, we could always modify the protocol to send additional "dummy" (non-used) bits until exactly c bits have been sent, without modifying the output. Thus, without loss of generality, we may assume that every leaf node lies at depth c (i.e., the protocol always sends exactly c bits).

We proceed to count the number of possible protocol trees. A binary tree with depth c has 2^c leaves, and the number of internal (non-leaf) nodes is $1+2+4+\ldots+2^{c-1} \leq 2^c$. Each internal node is specified by (i) a choice of whether the branch is based on the input of Alice (x) or Bob (y), and (ii) A subset $S \subseteq \{0,1\}^n$ specifying when to send '1' as the bit (for strings outside S, a '0' is sent). For a single internal node, the number of possible choices for (i) and (ii) collectively is at most $2 \cdot 2^{2^n} = 2^{2^n+1} \leq 2^{2^{n+1}}$. Thus, for all internal nodes collectively, the number of choices is at most

$$\left(2^{2^{n+1}}\right)^{2^c} = 2^{2^{n+1} \cdot 2^c} = 2^{2^{n+c+1}}.$$

We also need to consider the leaf nodes. There are 2^c of these, and each one is assigned 0 or 1 (as the output value), meaning there are 2^{2^c} possible assignments in total. Multiplying the number of choices for the internal nodes and the number of choices for the leaf nodes, we conclude that the total number of functions with communication complexity c is at most

$$2^{2^{n+c+1}} \cdot 2^{2^c} = 2^{2^c + 2^{n+c+1}}.$$

Next, we count the number of functions $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. Each function has 2^{2n} inputs, each of which can produce an output of 0 or 1, so there are $2^{2^{2n}}$ possible functions. Combining with the above calculation, the proportion of functions with communication complexity c is at most

$$\frac{2^{2^c + 2^{n+c+1}}}{2^{2^{2n}}} = 2^{2^c + 2^{n+c+1} - 2^{2n}}.$$

When $c = n-2$, this simplifies to $2^{2^{n-2} + 2^{2n-1} - 2^{2n}}$, and writing $2^{2n-1} = \frac{1}{2} \cdot 2^{2n}$ and $2^{n-2} = o(2^{2n})$, this simplifies to $2^{-\frac{1}{2}2^{2n}(1-o(1))}$. This fraction decays to zero as $n \to \infty$, and in fact does so extremely fast ("doubly exponentially fast"). □

4.2 Lower Bounds from Rectangles

We typically want to understand the communication complexity of specific functions, rather than seeking (overly) general statements as above. Theorem 3.2 gives a useful means for establishing lower bounds in such cases; for convenience, its contrapositive form is repeated as follows.

Corollary 4.2. *If the function f is such that we cannot form a partition of $\mathcal{X} \times \mathcal{Y}$ using at most 2^c monochromatic rectangles, then the communication complexity must exceed c.*

A common strategy is to show that *there are no large monochromatic rectangles*. For example, if $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$ then the size of $\mathcal{X} \times \mathcal{Y}$ is 2^{2n} (i.e., this is the number of entries in the matrix), so if each monochromatic rectangle contains at most γ entries then we require at least $\frac{2^{2n}}{\gamma}$ of them just to "cover" the entire space. We can demonstrate (a slight variation of) this idea through the EQUALS example.

Equality

Consider the equality function EQUALS : $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined as:

$$\text{EQUALS}(x,y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The trivial protocol has length $n+1$, and we would like to know whether we can do better.

Recall that we represent f as a binary matrix of size $2^n \times 2^n$, and we are interested in its 0-monochromatic and 1-monochromatic rectangles. We observe that EQUALS does in fact have large 0-monochromatic rectangles; see Figure 5 for an example. On the other hand, its 1-monochromatic rectangles are extremely small – just 1×1!

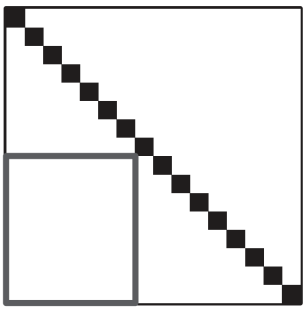


Figure 5: The EQUALS function does have large monochromatic rectangles [1].

Claim 2. *If R is a 1-monochromatic rectangle, then R has size 1×1.*

This claim is fairly straightforward to see – if we try to form a rectangle including (say) two 1s (shaded entries) in the figure shown, it will unavoidably also contain two 0s (unshaded entries).

We know there are 2^n inputs x with $\text{EQUALS}(x,x) = 1$. Thus, we need 2^n rectangles to cover these inputs. In addition, we also need at least one 0-monochromatic rectangle to cover the remaining inputs. By Theorem 3.2, if a function has a communication complexity c, then its domain can be partitioned into 2^c rectangles. Thus, deduce the lower bound $c > n$ and conclude the following.

Theorem 4.3 (**Theorem 1.16 in [1]**)**.** *The deterministic communication complexity of* EQUALS *is exactly n+1.*

Disjointness

We now turn to an example in which the above approach gives the correct scaling but a suboptimal constant (the optimal constant will be established in Section 4.3). Again consider $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, and consider the *disjointness* function defined by

$$\text{DISJ}(x,y) = \begin{cases} 1 & \text{if } x_i = y_i = 1 \text{ for some } i \in \{1,\ldots,n\}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

In other words, the function is 1 if and only if the sets $\{i : x_i = 1\}$ and $\{i : y_i = 1\}$ are disjoint. Instead of continuing with this form of the function, it will be more convenient to simply define $X = \{i : x_i = 1\}$ and

$Y = \{i : y_i = 1\}$, and proceed with the following *equivalent definition*:

$$\mathrm{DISJ}(X, Y) = \begin{cases} 1 & \text{if } X \cap Y = \varnothing, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that there is a one-to-one mapping between (x, y) and (X, Y). Trivially, Alice can send her whole input to Bob (who can then send back the answer) and achieve a protocol having $n + 1$ bits of communication.

Similar to EQUALS, the function DISJ has "very large" 0-monochromatic rectangles, e.g., $R_i = \{(X, Y) : i \in X, i \in Y\}$ for any fixed i. In fact, it also has "somewhat large" 1-monochromatic rectangles, but only containing 2^n entries which is much smaller than the total matrix size 2^{2n}. Formally:

Claim 3. *Every 1-monochromatic rectangle of* DISJ *contains at most* 2^n *entries.*

Proof. Let $R = A \times B$ be any 1-monochromatic rectangle. Let $X' = \cup_{X \in A} X$ and $Y' = \cup_{Y \in B} Y$ be the union of all the sets of A and B, respectively. By the 1-monochromatic property, we have that X' and Y' are disjoint, and thus $|X'| + |Y'| \leq n$. Since A and B consist of subsets of X' and Y' respectively, we also have $|A| \leq 2^{|X'|}$ and $|B| \leq 2^{|Y'|}$, and we conclude that the number of entries in R is $|A| \cdot |B| \leq 2^{|X'|+|Y'|} \leq 2^n$. □

Next, the number of inputs (X, Y) to DISJ that produce an output of 1 is exactly 3^n. This is because for each element in $\{1, \ldots, n\}$, there are three possibilities: It is only in X, only in Y, or in neither. Overall, at least $3^n/2^n = 2^{(\log_2 3 - 1)n}$ monochromatic rectangles are needed to cover just the 1s of DISJ. Since one further rectangle (at least) is needed for the 0s, we obtain the following.

Theorem 4.4 (**Theorem 1.14 in [1]**). *The deterministic communication complexity of* DISJ *is at least* $\lfloor (\log_2 3 - 1)n \rfloor + 1$.

This lower bound matches the upper bound of $n + 1$ to within a constant factor, but it turns out that we can get a better lower bound via a different approach, which we describe next.

4.3 Fooling Sets

A simple but powerful idea to get improved lower bounds is to *identify a "large" subset of $S \subset X \times Y$ such that each element of S must be associated with a different rectangle.* This means that the number of rectangles must also be "large", from which Corollary 4.2 gives us a lower bound. The relevant formal definition is given as follows.

Definition 4.1 (**Fooling sets**). *A set $S \subset X \times Y$ is called a* ***fooling set*** *for a function g if every monochromatic rectangle with respect to g can share at most one element with S.*

As hinted above, any partition of the inputs into rectangles must be at least as large as the size of such a fooling set. Thus, Corollary 4.2 immediately implies the following.

Corollary 4.5 (**Theorem 1.20 in [1]**). *If g has a fooling set of size s, then the communication complexity of g is at least* $\log s$.

The natural question is then *how to find a large fooling set.* This is often done case-by-case and typically requires some creativity; we give just one example by returning to the DISJ function.

Disjointness

In the following, we define $N = \{1, \ldots, n\}$ for brevity.

Claim 4. *The set* $S = \{(X, N \setminus X) : X \subseteq [n]\}$ *is a fooling set for* DISJ.

Proof. Recall the definition of a fooling set, and assume for contradiction that there exist two sets $X \neq X'$ such that $(X, N \setminus X)$ and $(X', N \setminus X')$ appear in some rectangle R. Since X and $N \setminus X$ are disjoint, it must be a 1-monochromatic rectangle. By the definition of a rectangle, R must also contain $(X, N \setminus X')$ and $(X', N \setminus X)$. However, this is impossible, since the assumption $X \neq X'$ implies that at least one of $X \setminus X'$ or $X' \setminus X$ is non-empty (and thus DISJ should return 0 instead of 1 on either $(X, N \setminus X')$ or $(X', N \setminus X)$). This establishes the desired contradiction, and we conclude that S must indeed be a fooling set. □

Since $|S| = 2^n$, we conclude from Corollary 4.5 that the deterministic communication complexity is at least n. In fact, we can do slightly better by noting that each $(X, Y) \in S$ has $\text{DISJ}(x, y) = 1$, and we need at least one more 0-monochromatic rectangle on top of the 1-monochromatic ones; this gives the following.

Theorem 4.6 (**Theorem 1.22 in [1]**). *The deterministic communication complexity of* DISJ *is exactly* $n + 1$.

4.4 The Rank Lower Bound Method

The following theorem provides a useful counterpart to the upper bound in Theorem 3.4, albeit with a sizeable gap between the two ($\text{rank} + 1$ vs. $\log_2(\text{rank} + 1)$).

Theorem 4.7 (**Theorem 2.4 in [1]**). *Consider any* $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$*, and let* M *be the associated binary matrix of size* $|\mathcal{X}| \times |\mathcal{Y}|$*. If* M *is not the all* 1*s matrix, then the communication complexity of* f *is at least* $\log_2(\text{rank}(M) + 1)$.

Proof. Let c denote the communication complexity. Theorem 3.2 shows that M can be partitioned into at most 2^c monochromatic rectangles. For each rectangle R in the partition, let z_R denote the value of the function in the rectangle, and let M_R denote the $|\mathcal{X}| \times |\mathcal{Y}|$ matrix whose (x, y)-th entry is z_R if $(x, y) \in R$, and 0 otherwise. Observe that:

- If $z_R = 0$, then M_R contains all 0s and thus has rank 0;
- If $z_R = 1$, then M_R contains a single rectangle of 1s, and thus has rank 1 (see item #2 in Fact 2 and note that every non-zero column is identical).

In both cases, we have $\text{rank}(M_R) \leq 1$. Moreover, M can be expressed as the sum of 2^c such matrices, and at least one of these matrices is 0 because M has at least one zero. Thus, by item #4 in Fact 2, we have $\text{rank}(M) \leq 2^c - 1$, and thus $c \geq \log_2(\text{rank}(M) + 1)$. □

Note that the condition that M is not all-1s is necessary, since the all-1s case gives $\log(\text{rank}(M) + 1) = 1$ but its communication complexity is trivially 0.

As an example application of Theorem 4.7, we return to the EQUALS function and give an alternative proof of its lower bound: The matrix M is simply the $2^n \times 2^n$ identity matrix, which has rank 2^n. Thus, the rank-based lower bound becomes $\lceil \log(2^n + 1) \rceil = n + 1$.

(Optional**)** Recent research has shown that the rank-based upper bound can be improved to at least to at least $\widetilde{O}(\sqrt{r})$. It remains an open problem as to whether the upper bound can be improved to

$(\log r)^{O(1)}$ – this is called the *log-rank conjecture*. If the answer is "yes", then it is known that the $O(1)$ term must be at least 2, since there exist cases where the communication complexity is $\widetilde{\Omega}((\log r)^2)$. See the end of `https://www.youtube.com/watch?v=zFHWmmThdT4` for further discussion.

5 Randomized Communication Complexity

Randomized algorithms are central to theoretical computer science, and also widely used in practice. Accordingly, it is natural to ask to what extent *randomized protocols* can help when it comes to communicating $f(x, y)$. It turns out that this can in fact change the picture quite drastically. While this is a major topic in communication complexity, we give it only a brief treatment, omitting most proofs. We first introduce some important concepts.

Randomization and error probability. By randomization, we mean that Alice and Bob are now allowed to make decisions such as "Send a 1 with probability p, and a 0 with probability $1-p$". (The source of randomness is important, and is discussed below.) Naturally, this means that multiple invocations of the protocol for fixed (x, y) might produce different outputs. We fix $\delta > 0$ and impose that for all inputs, the protocol is correct with probability at least $1 - \delta$:

$$\max_{x \in \mathcal{X}, y \in \mathcal{Y}} \mathbb{P}[\pi(x, y) \neq f(x, y)] \leq \delta, \tag{4}$$

where $\pi(x, y)$ is the (random) output of the protocol. Typically we consider δ to be a fixed constant in $\big(0, \frac{1}{2}\big)$, such as $1/3$, since a success probability of $2/3$ (say) can easily be boosted to any target $1 - \delta$ by repeating it $O\big(\log \frac{1}{\delta}\big)$ times and taking the majority output. (**Exercise:** *Try proving this using results from the concentration lecture.*)

Number of bits communicated. The notion of "number of bits communicated" can also be defined in one of several ways, since it is now a random variable. Specifically, we could consider its average, its high-probability behavior, or its worst-case behavior. It turns out that this choice is fairly inconsequential, at least in terms of scaling laws (**Exercise:** *Think about why this is the case*), and in the following we will take the requirement "at most c bits are communicated" as needing to hold *with probability one*.

Source of randomness: There are at least 3 distinct models for the source of randomness:

- **Shared randomness (the public-coin model)**: A protocol is said to use public coins if all parties have access to a common sequence of random bits.
- **Private randomness (the private-coin model)**: A protocol is said to use private coins if each party privately and independently samples a string of random bits.
- **Input randomness (the distributional model)**: Another approach is to assume that the *input* is random, and characterize how many bits need to be communicated *on average*. (Note: In this case, the above considerations, including (4), need to be suitably modified.)

Usually, when people refer to "the randomized communication complexity", they are referring to the public-coin case. This is partially justified by the following result, which shows that converting public-coin to private-coin loses at most a logarithmic factor (which is typically considered "small").

Theorem 5.1. (Newman's Theorem) *Any pubic-coin protocol can be converted to a private-coin protocol with at most a δ' increase in the error probability and at most an $O\big(\log n + \log \frac{1}{\delta'}\big)$ increase in the communication cost (for any $\delta' > 0$).*

In particular, for a constant error probability, if the public-coin communication complexity is $\Theta(\log n)$, then its private-coin communication complexity is $\Theta(\log n)$, and if its public-coin communicaton complexity is $\omega(\log n)$ (e.g., $\Theta(\sqrt{n})$), then the private-coin communication complexity is identical up to asymptotically lower-order terms. Of course, any private-coin protocol can trivially be converted to a public-coin protocol.

Equality

While EQUALS served as a "very hard" function (requiring $n+1$ bits of communciation) for determinstic protocols, it turns out that this property is lost in the randomized case. This is seen via the following upper bounds for the public-coin and private-coin cases respectively.

Theorem 5.2. *The public-coin randomized communication complexity of EQUALS is $O(1)$.*

Theorem 5.3. *The private-coin randomized communication complexity of EQUALS is $O(\log n)$.*

Public-coin case: We first consider the public coin case, and consider the following protocol:

- Alice and Bob extract $2n$ random bits, say $r = (r_1, \ldots, r_n)$ and $r' = (r'_1, \ldots, r'_n)$ (with $r_i \sim$ Bernoulli(1/2) independently and similarly for r') from the public source of randomness. Thus, the same random strings r, r' are known to both of them.
- Alice computes the inner products $r \cdot x$ and $r' \cdot x$ and sends them to Bob.
- Bob computes $r \cdot y$ and $r' \cdot y$. If $r \cdot x = r \cdot y$ and $r' \cdot x = r' \cdot y$ then Bob declares that $x = y$; otherwise he declares that $x \neq y$.

If $x = y$, then the correct decision will trivially be made with probability one. On the other hand, if $x \neq y$, it is easy to check that $\mathbb{P}[r \cdot x = r \cdot y] = \frac{1}{2}$ and similarly for r', and thus a correct decision is made with probability $1 - \left(\frac{1}{2}\right)^2 = \frac{3}{4}$. Observe that this probability can be boosted to $1 - 2^{-k}$ by using k length-n strings instead of just two.

Private-coin case: We could infer the private-coin result from the public-coin one using Theorem 5.1, but it's also interesting to give a direct proof. The above strategy cannot directly be used, since Alice's random bit string r is not known to Bob, and sending it would be too expensive.

Instead, we consider the following strategy:

- Alice and Bob agree (in advance) on using a binary error-correcting code for length-n messages, producing length-βn codewords for some $\beta > 1$, and satisfying the property that any two codewords $C(x)$ and $C(y)$ (for $x \neq y$) differ in at least a fraction α of their bits. (We know from the earlier lecture that such codes exist for suitable constants α and β.)
- Alice computes $C(x)$, and sends the pair $(i, C(x)_i)$ for k randomly-chosen indices $i \in \{1, \ldots, \beta n\}$.
- Bob computes $C(y)$, and checks if $(i, C(x)_i) = (i, C(y)_i)$ for all such i values. If so, he declares $x = y$, and otherwise he declares $x \neq y$.

We again have a probability one of being correct in the $x = y$ case, and it is a simple exercise to show that we succeed with probability at least 3/4 in the $x \neq y$ case as long as k is large enough as a function of (α, β). (**Exercise:** *Try showing this.*) Sending an index in $\{1, \ldots, \beta n\}$ requires $\lceil \log_2 \beta n \rceil$ bits (plus 1 bit for $C(x)_i$), so the communication cost is $O(\log n)$ when $k = O(1)$.

Disjointness

With EQUALS being an "easy" problem in the randomized case, we need to search for different problems that are "hard" (as is required for establishing lower bounds via reductions). The most prominent such problem in the randomized case is **disjointness**, somewhat like how SAT is the starting point for NP-hardness reductions. *(Side-note: Another prominent example is "inner product mod 2", but we'll skip that.)*

The hardness result is formally stated as follows.

Theorem 5.4 (**Theorem 6.19 in [1]**). *Any randomized (public-coin) protocol that computes the disjointness function with error probability at most* $1/2 - \epsilon$ *must have communication complexity* $\Omega(n\epsilon^2)$ *(in particular,* $\Omega(n)$ *if* ϵ *is a constant).*

This matches the trivial upper bound of $O(n)$ (via Alice sending her whole input) to within a constant factor. The proof of this theorem is non-trivial and relies heavily on tools from information theory, but interested students should now have the prerequisite background to be able to follow it; see [1, Chapter 6] for a self-contained proof.

Discussion

We have only scratched the surface of randomized communication complexity. Randomized protocols for specific functions often require some creativity (e.g., see the example 'Sparse Set Disjointness' in `https://cseweb.ucsd.edu/classes/wi19/cse291-b/3-randomized.pdf`, or the function "$\geq$" in the tutorial). There are also a number of other variations of communication complexity, some of which we briefly mention in Appendix A.

6 Applications in Theoretical Computer Science

Communication complexity has proven to be an powerful tool for proving lower bounds in other areas of theoretical computer science (TCS), including VLSI, decision tree lower bounds, cell probe model and dynamic data structures, Boolean circuit complexity, Turing machine time-space trade-offs, streaming algorithms, game theory, high-dimensional estimation, distributed computation, and proof complexity. (Don't worry if you haven't heard of most of these.)

In the following, we give just two examples of how lower bounds on communication complexity imply certain lower bounds in computer science problems of interest.

6.1 Example 1: Space Complexity Lower Bounds for Streaming Algorithms

- Loosely speaking, a streaming algorithm is one that reads in a list sequentially and outputs some property of that list, but is unable to store the entire list in memory. Thus, it has to "discard" some elements along the way.
- As a simple example, if the goal is to output the average of a list of numbers, then only two things need to be known – the sum of all elements, and the number of elements (then output the ratio of the two). When a new element of list is read, these numbers can be updated and then the list element can be forgotten.

- For other problems, even simple ones, it can be very unclear how much space is required (e.g., if the goal is median instead of mean, this is already much less obvious).
- Reductions based on communication complexity are a powerful method to prove lower bounds, showing a certain minimum amount of space is unavoidable.

Deterministic algorithm example: Number of distinct elements

- Consider the following goal: Given a list of numbers $a_1, \ldots, a_n$, each taking values in the range $\{1, 2, \ldots, m\}$, output the number of distinct values appearing in the list. For example, if the list is $(1, 3, 1, 5, 9, 5, 5, 3, 5)$, then there are 4 distinct elements: 1,3,5,9.
- **Claim:** If $m \geq n$, then any deterministic streaming algorithm for this task must use $\Omega(n)$ space.
  - The more general result is $\Theta(\min\{m, n\})$, and it is easy to get a near-matching upper bound (just store the entire list if $\min\{m, n\} = n$, or store an "indicator bit" for each value if $\min\{m, n\} = m$).
- To see this, we will perform a reduction based on the $\Omega(n)$ communication complexity of the EQUALS function. Fix any inputs (x, y) to the EQUALS problem, with length n each. Then consider the following two lists of numbers:

$$L_x = \{i : x_i = 1\}$$
$$L_y = \{i : y_i = 1\}.$$

  These lists both have size at most n.
- Consider the following protocol for EQUALS:
  - Alice runs streaming Distinct Elements on L_x, and Bob runs it on L_y (this requires no communication).
  - Alice then sends the memory contents of the Distinct Elements algorithm (after L_x has been processed) to Bob, who continues running it (with further elements from L_y) to get the number of Distinct Elements in $L_x \circ L_y$ (where $\circ$ denotes list concatenation).
  - Alice also sends the number of Distinct Elements in L_x to Bob. (This only requires $O(\log n)$ bits of communication.)
  - Bob checks whether L_x, L_y, and $L_x \cup L_y$ have the same number of distinct elements. If so, then he declares that EQUALS(x, y)=1, otherwise 0.

  It is easy to see that this produces the correct output of EQUALS. Yet if the streaming algorithm's memory were $o(n)$, then the same would be true of the communication cost.
- We know that $o(n)$ communication cost is impossible, so we conclude that the streaming algorithm must have $\Omega(n)$ memory.

Randomized algorithm example: Approximate highest count

- Consider the following goal: We are given a list of numbers $a_1, \ldots, a_n$, each taking values in the range $\{1, 2, \ldots, m\}$. Let n_i be the number of occurrences of symbol i for $i = 1, \ldots, m$, and define $n_{\max} = \max_i n_i$ to be the highest count. The goal is to output a number $\hat{n}$ such that $\hat{n} \in [0.8n_{\max}, 1.2n_{\max}]$ (i.e., find an approximate value of $n_{\max}$) with probability at least 2/3.

- The exact numbers 0.8, 1.2 and 2/3 aren't important, but they provide leniency compared to the above example – we only need to give an approximate answer, and we can use a randomized algorithm that has some probability of failing.

- **Claim:** If $m \geq n$, then any (possibly randomized) streaming algorithm achieving this goal must use $\Omega(n)$ space. (Again, the more general dependence is $\min\{m, n\}$.)

- The proof is quite similar to before, but is based on the hardness of DISJOINT rather than EQUALS (the latter is no longer suitable, since we are now allowing randomization, and EQUALS has very low randomized communication complexity).

- Specifically, we use the streaming algorithm $\mathcal{A}$ as a subroutine for computing DISJOINT(x, y):
  - Alice feeds into $\mathcal{A}$ the indices i where $x_i = 1$;
  - Alice sends the memory contents of $\mathcal{A}$ to Bob;
  - Starting with those memory contents, Bob feeds into $\mathcal{A}$ the indices i where $y_i = 1$, then reads the algorithm's output;
  - Bob declares DISJOINT(x, y)=1 if the output is at most 1.5, and 0 otherwise.

- This gives the correct answer for DISJOINT with probability at least 2/3, because $n_{\max}$ is 1 in the disjoint case (meaning the answer should be in $[0.8, 1.2]$) but is at least 2 in the non-disjoint case (meaning the answer should be 1.6 or higher).

- We know that DISJOINT has communication complexity $\Omega(n)$, so we conclude that the communication step (consisting of the memory contents) must have $\Omega(n)$ bits. That is, $\mathcal{A}$ requires $\Omega(n)$ memory.

6.2 (**Optional**) Example 2: Query Complexity of Property Testing

- *Property testing* is a broad field in computer science and statistics where we seek to determine whether a given string/function/random variable/etc. has a certain property using limited information (e.g., queries to the string/function, or samples from the distribution).

- Here we focus on the problem of *motonicity testing*:
  - There exists a function $f(x)$ with inputs in $\{0, 1\}^n$ and outputs in $\{0, 1, 2, \ldots, R\}$ for some $R > 0$.
  - The function f is not known directly, but instead the algorithm can can only perform *queries*: If input x is queried, then the value $f(x)$ is returned. We would like to use as few queries as possible. (Ideally far fewer than the trivial 2^n attained by querying every point.)
  - The function f is said to be *monotone* if changing any coordinate of the input from 0 to 1 increases the function value (or keeps it the same).
  - **Goal:** Construct a (possibly randomized) algorithm that performs Q queries and then returns:
    - 'YES' with probability at least 2/3 if f is monotone;
    - 'NO' with probability at least 2/3 is f is ϵ-far from being monotone (i.e., we would need to modify at least an ϵ-fraction of function values to get a monotone function);
    - (Either 'YES' or 'NO' may be returned in all other cases.)

- **Claim:** When $R = 2(n+1)$ (or larger) and $\epsilon = \frac{1}{8}$ (or smaller), this problem requires $Q = \Omega(n)$ queries.
  - This nearly matches a known upper bound of $O\big(\frac{n}{\epsilon} \log R\big)$, e.g., see Chapter 8 of Roughgarden.
- The proof is again based on a reduction involving (a variant of) the DISJOINT problem:
  - It will be convenient to treat binary vectors and sets interchangeably, so we can write $f(x)$ equivalently as $f(S)$, with S being the set of indices where $x_i = 1$.
  - Define $V = \{1, \ldots, n\}$, let A and B be subsets of V, and consider the following function defined on subsets of $\{0,1\}^n$:
    $$h_{AB}(S) = 2|S| + (-1)^{|S \cap A|} + (-1)^{|S \cap B|}.$$
    - Observe that if A and B are disjoint, then h_{AB} is monotone, because adding an element to S adds 2 in the first term, but only one of the other two terms can decrease from $+1$ to -1 (not both).
    - If A and B have exactly one element in common, then it can be shown that h_{AB} is $\frac{1}{8}$-far from monotone; roughly, this is because if $|A \cap B| = \{i\}$ for some i, then $\frac{1}{4}$ of the possible subsets of $\{1, \ldots, n\} \setminus \{i\}$ will give $h_{AB}(S \cup \{i\}) < h_{AB}(S)$ due to both $|S \cap A|$ and $|S \cap A|$ being even (and hence changing to odd when i is added).
    - Observe also that $h_{AB}(S)$ outputs an integer between 0 and $2n + 2$, which is consistent with the choice $R = 2(n + 1)$ above.
  - The reduction actually uses a variant of DISJOINT called UniqueDisjoint, where in the case of being non-disjoint, the strings (x, y) are guaranteed to only differ in one entry. The $\Omega(n)$ communication complexity lower bound is known to still hold for this problem.
  - The reduction is as follows:
    - Alice knows $x \in \{0,1\}^n$ and the corresponding set is called A. Bob knows $y \in \{0,1\}^n$ and the corresponding set is called B.
    - Alice and Bob both run monotonicity testing on h_{AB}; after each query S, Alice sends $(-1)^{|S \cap A|}$ to Bob, and Bob sends $(-1)^{|S \cap B|}$ to Alice, so that both of them get the query response $h_{AB}(S)$.
    - If the monotonicity tester returns 'YES' then A and B are declared to be disjoint. Otherwise they are declared to overlap in a single entry.
  - Observe that the total amount of communication is $2Q$ when Q queries are made.
  - The fact that UniqueDisjoint has (randomized) communication complexity $\Omega(n)$ thus directly implies that we need $Q = \Omega(n)$.

A (**Optional**) Other Communication Models and Settings

Beyond the settings we have covered, there are a number of other settings of interest, some of which we mention as follows:

- **Average-case complexity:** While the most common settings consider the worst-case communication cost over all x and y, there are also cases where an average-cost notion is of more interest. Specifically, we can assume that $(x, y) \sim D$ for some joint distribution D, and then study the *average* number of bits exchanged with respect to D.

- **Non-Boolean functions:** Many functions that we are interested in computing have non-binary outputs, say represented by an m-bit string for some $m > 1$. For example, the Hamming distance between $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ is represented by a number in $\{0, 1, \ldots, n\}$, which can be represented using $\lceil \log_2(n+1) \rceil$ bits. See for example [7].

- **One-way communication:** In the one-way setting, communication is only allowed from Alice to Bob, not vice versa. The goal is for Alice to communicate as little as possible while ensuring that Bob can compute $f(x, y)$. This setup moves things somewhat closer to information theory, and its associated tools like Fano's inequality are common.

- **Multi-agent settings:** Naturally, we can consider more than two agents, e.g., there could be k agents each knowing one of $x_1, \ldots, x_k$ (say with each $x_i \in \{0,1\}^n$), and the goal is to compute $f(x_1, \ldots, x_k)$. For the communication model, each agent could be allowed to broadcoast bits to all other agents, or communication could be more constrained (e.g., only allowed to occur across certain edges in a graph).

- **Nondeterministic protocols:** *(Note: The word "nondeterministic" here should be interpreted similarly to the complexity class NP; it is not synonymous with "randomized".)*

  In a nondeterministic protocol, the players are both provided an additional third input z (interpreted as a "nondeterministic guess"). Apart from this guess, the protocol is deterministic. The cost incurred on x, y is

  $$\min_z \{|z| + \text{number of bits exchanged by the protocol when the guess is } z\}.$$

  The *nondeterministic communication complexity* of f is the minimum k such that there is a nondeterministic protocol whose cost for all input pairs is at most k.

B (**Optional**) Proof of Theorem 3.3 (Rectangles to Protocols)

Recall that both Alice and Bob know f. We can similarly assume that they both know the partitioning into monochromatic rectangles, which we denote by $\mathcal{R}$. The goal is to devise a protocol that identifies the unique rectangle in $\mathcal{R}$ containing (x, y), since this clearly amounts to knowing the function value.

Let the rectangles be arbitrarily indexed as $1, 2, \ldots, 2^c$ (or fewer). We consider a protocol that operates in rounds, where in each round either Alice or Bob sends the index of one such rectangle (we will later convert from "sending indices" to "sending bits"). Specifically, this will be done in a manner such that the agents can eliminate at least half of the remaining rectangles (i.e., conclude that they do *not* contain (x, y)). This will mean that after at most $\log_2(2^c) = c$ rounds, we have narrowed down to the single (correct) rectangle containing (x, y), impying that $f(x, y)$ is known.

To achieve the above goal, we introduce the notion of *good rectangles*. Before doing so, we define the following notions (illustrated in Figure 6):

- Two rectangles $R = A \times B$ and $R' = A' \times B'$ *intersect horizontally* if A intersects A';
- Two rectangles $R = A \times B$ and $R' = A' \times B'$ *intersect vertically* if B intersects B'.

An important observation is that if x belongs to both A and A' (i.e., $x \in A \cap A'$) and y belongs to both B and B' (i.e., $y \in B \cap B'$), then (x, y) will lie in both $A \times B$ and $A' \times B'$. This implies the following:

Fact 3. *Two **disjoint** rectangles cannot intersect both horizontally and vertically.*

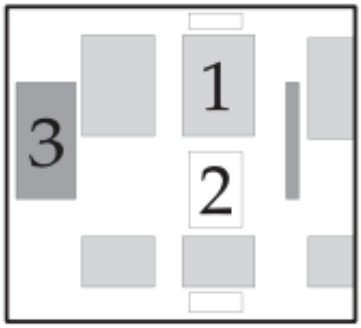


Figure 6: Rectangles 1 and 2 intersect vertically, while rectangles 1 and 3 intersect horizontally. [1]

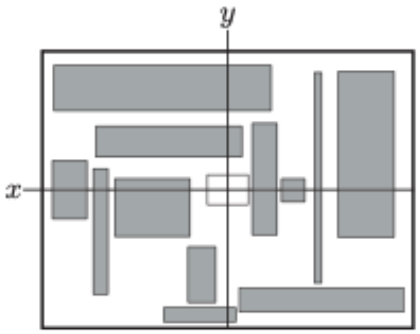


Figure 7: Either a rectangle aligned with x intersects no more than half of the other rectangles horizontally, or a rectangle aligned with y intersects at most half of them vertically, as detailed in [1].

Definition B.1 (**Good Rectangles [1]**). *For a given input (x, y), a rectangle $R = (A \times B) \in \mathcal{R}$ is **horizontally good** if $x \in A$ and R horizontally intersects at most $|\mathcal{R}|/2$ rectangles in $\mathcal{R}$. Similarly, R is **vertically good** if $y \in B$ and R vertically intersects at most $|\mathcal{R}|/2$ rectangles in $\mathcal{R}$. We say that R is **good** if it is either horizontally good or vertically good.*

Observe that if Alice can identify a horizontally good rectangle R, announcing its *name* will reduce the set of considered rectangles by at least half (since the ones that *don't* horizontally intersect R can be removed). Likewise, Bob can achieve a similar reduction by announcing a vertically good rectangle. Given that we assume all parties possess infinite computational capabilities, we don't dwell on the process by which they identify these good rectangles.

The remaining question is: *Is it always possible to find a good rectangle?* The answer is YES, as the following claim establishes.

Claim 5. *The unique rectangle in $\mathcal{R}$ that contains (x, y), denoted as $R_{x,y}$, is good.*

Proof. From Fact 3, we know that no rectangle in $\mathcal{R}$ can intersect $R_{x,y}$ both horizontally and vertically. Consequently, either at most half of the rectangles in $\mathcal{R}$ intersect $R_{x,y}$ horizontally (thus making it *horizontally good*), or at most half intersect it vertically (thus making it *vertically good*). This establishes the claim, and Figure 7 gives an illustration. □

A slight subtlety is that both Alice and Bob might know a good rectangle, and the protocol needs to specify who should send first. A simple approach is to let them take turns in giving them the opportunity to send a rectangle's index; then:

- If they know a good rectangle, they send '1' followed by the rectangle's index;
- If they don't know a good rectangle, they send '0'.

In this manner, there will be at most 2 extra bits sent for each rectangle index sent. The final communication cost is computed by noting that (i) with at most 2^c rectangles, sending a unique index requires at most $\log_2(2^c) = c$ bits; and (ii) with 2^c candidates initially and halving (or better) each round, there are at most $\log_2(2^c) = c$ rounds. Thus, the total communication cost is $O(c^2)$.

References

[1] A. Rao and A. Yehudayoff, *Communication Complexity and Applications*. Cambridge University Press, 2020.

[2] A. A. Razborov, "Communication complexity," in *An Invitation to Mathematics: From Competitions to Research*. Springer, 2011, pp. 97–117.

[3] T. Roughgarden *et al.*, *Communication complexity (for algorithm designers)*. Now Publishers, Inc., 2016, vol. 11, no. 3–4.

[4] T. Lee, A. Shraibman *et al.*, "Lower bounds in communication complexity," *Foundations and Trends® in Theoretical Computer Science*, vol. 3, no. 4, pp. 263–399, 2009.

[5] A. C.-C. Yao, "Probabilistic computations: Toward a unified measure of complexity," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE Computer Society, 1977, pp. 222–227.

[6] R. Kothari, "Nearly optimal separations between communication (or query) complexity and partitions," *arXiv preprint arXiv:1512.01210*, 2015.

[7] L. Fontes, S. Laplante, M. Lauriere, and A. Nolin, "The communication complexity of functions with large outputs," in *International Colloquium on Structural Information and Communication Complexity*. Springer, 2023, pp. 427–458.