Introductory Notes on Communication Complexity

Evan Wang, Daniel Xie

1 Introduction

A large portion of computational theory is concerned with questions regarding the capabilities of a single party executing the algorithm: whether checking an answer or creating an answer are more difficult, how much computation can be avoided by introducing randomness, etc. While this line of inquiry has motivated a plethora of insights, it is limited in its application to analyzing and optimizing the performance of algorithms which involve interparty communication. In 1979, Andrew Yao spawned a related subject in computational theory well-suited for tackling these sorts of problems: communication complexity.

2 Definitions of communication

The standard, "classic" communication problem is the scenario where there is a function $f(x, y) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ (a function on two inputs that gives a binary yes/no output) and there are two parties that are trying to compute the function. Party A (for Alice) gets input x, and party B (for Bob) gets input y; they cannot directly see each others' inputs, but they are allowed to agree on a protocol to compute or communicate anything they want.

The Arora and Barak text describes a communication protocol formally as a sequence of function pairs $(S_1, C_1), \dots, (S_t, C_t), (f_1, f_2)$. Each pair is a "round" of communication, i.e. a period of time where one party does computation then communicates a bit of information. Each S_i is a function that says "whose turn is it to send a bit?"; on round i, one person is picked to communicate a bit (based on the agreed protocol), and that decision depends on the previous rounds. Each C_i represents "personal computation"; that is, once Alice or Bob is chosen to be the communicator for this round, they use their input string (x or y) as well as local work they have stored to do some computation, then output the bit they are going to communicate this round. The ending (f_1, f_2) represents that Alice and Bob must both output the correct answer of $f(x, y)$ at the end of the protocol; f_1 and f_2 are the functions that Alice and Bob (respectively) use to get this answer after communication is finished.

There are other ways to envision communication protocols; for example, Shraibman and Lee draws them out as binary trees, where nodes branch depends on which bit was communicated. It can also be intuitive to draw out communication protocols similar to how protocols were drawn for verifiers and provers in our discussion about ZKPs, with the two parties on two sides of a sheet of paper and arrows going between them.

There are a few key ideas that apply to any model you would like to choose. The *cost* of a communication protocol is the total number of bits communicated in the protocol, and the *communication complexity* of function f is defined as

$$C(f) = \min_{\text{protocols } \mathcal{P}} \max_{x,y} \{\# \text{ bits exchanged by } \mathcal{P} \text{ when run on } x, y\}$$

which says "choose the protocol whose worst-case uses the least bits, and use that worst-case as the communication complexity." Additionally, since we only really care about how much Alice and Bob talk to each other, we assume that they have unbounded computational power.

Just as the exact details of Turing machines don't matter as much as we discussed involved complexity results, the exact definition won't matter too much when we discuss communication complexity results; the previous key ideas are usually enough. For example, consider this simple result:

Theorem 2.1. *For all $f(x, y)$, $C(f) \leq n$.*

Proof. Alice will communicate her entire input x to Bob (or, to be pedantic, communicate one bit of her input at at time for n rounds). Bob computes $f(x, y)$ by himself and outputs the correct answer. This protocol is correct since the true value of f is computed, and this protocol uses n bits of communication. This protocol can be used for any f. $\square$

Note that this didn't rely on anything relating to function pairs or binary trees, but only on the intuitive idea that bits can be sent back and forth, and both parties can do unbounded computation. In addition, this result gives a brief picture of the kind of results we're looking for in communication complexity: generally, linear communication is considered "hard," and sublinear communication is considered "efficient."

3 The fooling set argument

The fooling set argument is a technique to prove lower bounds of communication for functions.

Definition 3.1. A fooling set for $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is some $S \subseteq \{0,1\}^n \times \{0,1\}^n$ (a set of pairs of strings) and a single bit $b \in \{0,1\}$ such that

- $\forall (x, y) \in S, f(x, y) = b$.
- For every distinct pair $(x_1, y_1), (x_2, y_2) \in S$, either $f(x_1, y_2) \neq b$ or $f(x_2, y_1) \neq b$.

The first point says that all pairs in the set evaluate to b, and the second point says that for all pairs in the set, swapping one of its inputs with that of another pair causes it to be evaluated to not b. The immediate use of fooling set comes from this following theorem:

Theorem 3.1. *For any fooling set S for f, $C(f) \geq \log_2 |S|$.*

Let's take a look at how to actually use the fooling set argument. Let $EQ(x, y)$ (for "equality") be a function that outputs 1 if $x = y$ and 0 if $x \neq y$.

Theorem 3.2. *$C(EQ) \geq n$.*

Proof. The set $S = \{(x, y) \mid x = y\}$ with $b = 1$ is a fooling set for EQ. This set is every possible n-bit string "duplicated." We verify that S is indeed a fooling set:

- For any pair in S we have $EQ(x, y) = 1 = b$ since $x = y$ by construction.
- For every distinct pair $(x_1, y_1), (x_2, y_2) \in S$, by construction $x_1 \neq y_2$ (otherwise they would be the same two pairs), so $EQ(x_1, y_2) = 0 \neq b$ as required.

Then, since x and y are both n-bit strings, there are 2^n possible values for x, and since y will match x, we have $|S| = 2^n$. Therefore, $C(EQ) \geq \log_2 2^n = n$. $\square$

4 Communication matrices

Another useful method of showing lower bounds on communication relates to matrices. Before we get to this connection between communication protocols and matrices, we need three definitions.

We know that x and y, the inputs to a function f, are n-bit strings, but they can be equivalently treated as numbers counting from $0 \ldots 2^n - 1$. If we consider them as such, we can use them as indices of a matrix and define the communication matrix:

Definition 4.1 (Communication matrix)**.** The communication matrix for f, denoted M_f, is a matrix such that $(M_f)_{xy} = f(x, y)$.

This means that every entry of the communication matrix is the correct answer to f when we take the row and column indices as the inputs to f. For example, the matrix for EQ is the identity matrix (when the row and column indices are equal, the entry is 1; this is exactly the main diagonal of the matrix).

Definition 4.2 (Combinatorial rectangle)**.** A combinatorial rectangle of a matrix is created by choosing all entries from some rows from the matrix, then from those rows, choosing all entries from some columns. They need not be contiguous. The rectangle is **monochromatic** if all the entries in the rectangle are equal (in our case, all entries are 0 or all entries are 1).

Below are various examples and non-examples using the matrix for EQ.

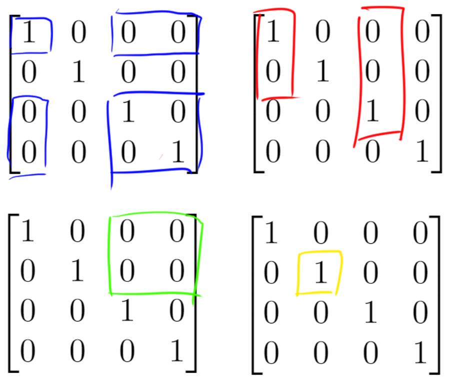

Figure 1: The top left (blue) represents a rectangle created by selecting rows 1,3,4 and columns 1,3,4. Note that it is not monochromatic since it has 0 and 1 entries. The top right (red) is a non-example of a rectangle; columns 1 and 3 are selected, but we can't choose to exclude one number from column 1. The bottom two matrices (green and yellow) are examples of monochromatic rectangles.

Definition 4.3 (Monochromatic tiling)**.** A monochromatic tiling of a matrix is a scheme that splits a matrix into non-overlapping monochromatic rectangles and covers all entries. We denote $\chi(f)$ to be the minimum number of rectangles for a monochromatic tiling of M_f.

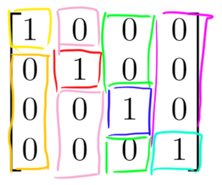

Figure 2: An example of a monochromatic tiling of the EQ matrix that uses 8 rectangles (each color is a different rectangle). There actually is no better solution, so $\chi(EQ) = 8$ when the inputs are length 2.

Now, we can finally begin exploring how communication relates to all of these definitions. Consider some protocol P in which, in the first round, Alice does some computation and sends over a bit. It might be the case that Alice outputs 0 or 1 depending on the input she gets; then, we can categorize all possible inputs into "inputs that cause Alice to send 0 on the first round" and "inputs that cause Alice to send 1 on the first round." With this sense, we can partition a communication matrix into two rectangles by choosing rows that correspond to inputs that cause Alice to send 0 or 1. Then, let's say that Bob will communicate next round. Bob will subdivide each of the rectangles that Alice created; this is because there is

- a set of inputs that cause Bob to send 0 *given that Alice sent him a 0*,
- a set of inputs that cause Bob to send 1 *given that Alice sent him a 0*,
- a set of inputs that cause Bob to send 0 *given that Alice sent him a 1*,
- a set of inputs that cause Bob to send 1 *given that Alice sent him a 1*.

At the end of the second round, the communication matrix is tiled using 4 rectangles. As the communication protocol goes on, the rectangles will be split in two again and again; in general, if the protocol lasts k steps, there is a tiling using 2^k rectangles.

What happens when the protocol ends? We know that a rectangle is generated by tracing through the bits that Alice and Bob send each other, so we actually interpret the matrix entries inside the rectangle as the answer that Alice and Bob output at the end of the protocol. For this reason, every tiling scheme that a protocol generates must only use monochromatic rectangles at the very end (it doesn't make sense, at least in a deterministic setting, for the output to be "both 0 and 1").

In this way, a valid communication protocol gives rise to a monochromatic tiling of a communication matrix. Communication complexity is related to monochromatic tiling size in this way:

Theorem 4.1. $C(f) \geq \log_2 \chi(f)$, *where* $\chi(f)$ *is the size of the smallest monochromatic tiling of* M_f.

Proof. We prove this with fooling sets. Let S be a fooling set for f. Then consider two distinct pairs $(x_1, y_1), (x_2, y_2) \in S$. These can be thought of as two "corners" of a rectangle in the communication matrix M_f. Assume that $(x_1, y_1), (x_2, y_2)$ are part of the same monochromatic rectangle. Then, $(x_1, y_2), (x_2, y_1)$ are also part of the rectangle (they are the two other "corners"). However, by the definition of a fooling set $f(x_1, y_1) = b$ but $f(x_1, y_2) \neq b$ or $f(x_2, y_1) \neq b$, so the rectangle is not monochromatic, a contradiction. Therefore, every element in S is in its own monochromatic rectangle, so a tiling must use at least $|S|$ rectangles. We have $\chi(f) \geq |S|$, so the fooling set result $C(f) \geq \log_2 |S|$ implies $C(f) \geq \log_2 \chi(f)$. $\square$

There is an even more direct connection with linear algebra:

Theorem 4.2. $\chi(f) \geq rank(M_f)$. *Consequently,* $C(f) \geq \log_2 rank(M_f)$.

Whether or not $C(f) \leq \log_2 rank(M_f)$ is true is still an open question in communication complexity and is termed the "log-rank conjecture."

5 Extensions to classic communication complexity

Much like how P has had probabilistic, approximate, and quantum complexity classes built in relation to its core definition, so has communication complexity given rise to considerations of nondeterministic, randomized, quantum, and multiparty communication complexities.

Nondeterministic communication complexity, as its name suggests, relates to communication complexity somewhat as NP relates to P. Formally, it is concerned with the case where Alice and Bob have access to a third input, z, which is sometimes called a "nondeterministic guess" or the "oracle string." In the terms of P/poly, z can be thought of as "advice" A and B receive. The nondeterministic communication complexity of a function f is defined very similarly to deterministic communication, simply with the minimizing term being (# of bytes communicated between A and B with access to z). Additionally, there is a notion of cost in nondeterministic communication complexity, which can be thought of the minimum of ($|z| + \#$ of bytes communicated by $\mathcal{P}$) over all possible z.

Moreover, communication complexity's counterpart to BPP is randomized communication complexity. In this model, A and B can be thought as having access to a public random bitstring r from which they can draw randomness. Similarly to BPP, the probability that the randomized protocol outputs the same answer as f must be $\geq \frac{2}{3}$. Furthermore, the introduction of randomness may drastically reduce the required amount of communication for a function, with the tradeoff of course being the $\leq \frac{1}{3}$ chance of error.

Quantum communication complexity departs significantly from the classic model despite, on the most basic level, being classic communication complexity with qubits instead of bits. As the Lee & Shraibman text defines it, in the quantum model, A and B are not directly communicating with each other but are instead alternating turns applying unitary transformations to a state vector. Quantum communication complexity is then characterized by the number of rounds taken in a protocol rather than the number of bits communicated between the two parties.

Lastly, multiparty communication complexity is of interest due to how it may apply to proving lower bounds of streaming algorithms, proof complexity, and circuit complexity. Multiparty situations themselves come in two flavors: number-in-hand and number-on-the-forehead. In the former, similarly to the classic model, each party is given a unique input to the function to be decided. In the latter, each party is given every input save one akin to every party having their input displayed on their forehead, such that every party may see every input except for theirs (hence the name).

As can be seen, the study of communication complexity has yielded its own set of related complexity definitions which mirror but remain relatively separated from the rest of computational theory.

6 An application to the Gap-Hamming problem

This section will discuss how reductions can be used to derive results in communication complexity and is based on the paper "The One-Way Communication Complexity of Hamming Distance" (Jayram, Kumar, Sivakumar 2008). It will explore two variants of communication: one-way and randomized. We will define the indexing problem, the GapHD problem, and reduce indexing to GapHD.

Definition 6.1 (Indexing problem)**.** The setup of the problem is as follows: Alice is given an n-bit string of any form, and Bob is given an n-bit string that only has a single 1 digit and $n-1$ 0 digits. Alice can only communicate one-way to Bob, and together the parties must determine if Alice also has a 1 digit in the same position that Bob has his only 1 digit.

This problem has a known lower bound of $\Omega(n)$, even when parties are allowed randomness. Intuitively, since Bob can't communicate with Alice, Alice has no idea where Bob's single 1 digit is, so she has to send over her entire string.

Definition 6.2 (GapHD problem)**.** In the GapHD problem, Alice and Bob are both given n-bit strings. The Hamming distance between the strings is defined as $H(x, y)$ = number of indices in which x and y differ, and in the GapHD problem the input strings are guaranteed to either satisfy $H(x, y) \geq n/2 + \sqrt{n}$ or $H(x, y) \leq n/2 - \sqrt{n}$ (the strings are guaranteed to have quite high or quite low Hamming distance), and the two parties must figure out which case occurs. We restrict the problem to one-way communication, and allow randomness.

Theorem 6.1. *The one-way, randomized communication complexity of GapHD is* $\Omega(n)$.

Proof. We reduce the indexing problem to the GapHD problem. Let x denote Alice's and y Bob's input in the indexing problem. We first transform x into a vector that has -1 or 1 entries ($u \in \{-1, 1\}^n$) by replacing all the zero entries of x with -1 instead. We also note that since y has only a single 1 entry and zeroes everywhere else, it can be realized as a standard basis vector of $\mathbb{R}^n$; we will use e_j to denote this interpretation of y, where j is the location of the single 1 entry.

For the next step, we choose N vectors from the uniform random distribution over $\{-1, 1\}^n$. Denote them $r^1 \dots r^N$ (superscripts will denote which r we are considering since subscripts will be used to denote a specific entry of some r^i).

We define Alice's input to the GapHD problem as x' such that $x'_k = \text{sign}(u \cdot r^k)$ (all dots will refer to the dot product). We define Bob's input to the GapHD problem as y' such that $y'_k = \text{sign}(e_j \cdot r^k)$. Note that the dot product of any r^i with e_j is simply going to be r^i_j (this denotes the jth entry of r^i) since e_j only has a 1 entry in the jth position and all others zero. Since r^i only has -1 and 1 entries, r^i_j is also the sign of the result, i.e. $\text{sign}(r^i \cdot e_j) = r^i_j$. So actually, we have $y'_k = r^k_j$.

Then, the Hamming distance between x' and y', by definition, is

$$\begin{aligned} H(x,y) &= \#\text{ indices } k \text{ where } x'_k \neq y'_k \\ &= |\{k : x'_k \neq y'_k\}| \\ &= |\{k : \text{sign}(u \cdot r^k) \neq \text{sign}(e^j \cdot r^k)\}| \\ &= |\{k : \text{sign}(u \cdot r^k) \neq r^k_j\}| \end{aligned}$$

It is shown in the paper that for any r drawn from the uniform random distribution over $\{-1,1\}^n$ that, for some constant $c > 0$,

$$Pr[\text{sign}(u \cdot r) \neq r_j] \geq 1/2 + c/\sqrt{n} \text{ if } u_j = -1$$
$$Pr[\text{sign}(u \cdot r) \neq r_j] \leq 1/2 - c/\sqrt{n} \text{ if } u_j = +1$$

This is saying that, even though r_j is chosen over uniform random $\{0,1\}$, the way we have constructed our reduction is such that sign$(u \cdot r)$ is quite "different" from uniform random and depends on u_j.

To finish off the proof, the authors use the Chernoff bound (specifically, an instance that is actually the Hoeffding bound we used in Homework 3). We can define $X_1, X_2, \ldots, X_N$ such that $X_i = 1$ if $x'_i \neq y'_i$ and 1 if they are unequal. These are i.i.d variables (each digit doesn't rely on another digit; the only ingredients used here are the original input strings and strings drawn from uniform random). We define $X = \sum_{i=1}^{N} X_i$, then by the bound we have

$$Pr[X - E[X] > \epsilon] \leq \exp(-2\epsilon^2/N)$$
$$Pr[X - E[X] < -\epsilon] \leq \exp(-2\epsilon^2/N)$$

Note that X counts exactly the Hamming distance between x' and y' since every time it gets incremented, that indicates an index where the two strings differ; this is to say that $X = H(x', y')$. Also, X can be thought of as a binomial variable with N trials. A trial succeeds when $x'_i \neq y'_i$, and this is equal to $Pr[\text{sign}(u \cdot r) \neq r_j]$ which depends on u_j as described above. We consider the $u_j = -1$ case first. In this case, the expected value of binomial variable X is $E[X] = N(1/2 + c/\sqrt{n})$. Choose $N = 4n/c^2$ and $\epsilon = \sqrt{N}$, and we can plug in to the Hoeffding bound to get

$$\begin{aligned} Pr[X - E[X] < -\epsilon] &\leq \exp(-2\epsilon^2/N) \\ Pr[H(x',y') - N(\frac{1}{2} + \frac{c}{\sqrt{n}}) < -\sqrt{N}] &\leq \exp(-2\sqrt{N}^2/N) \\ Pr[H(x',y') - N(\frac{1}{2} + \frac{c}{\sqrt{c^2N/4}}) < -\sqrt{N}] &\leq e^{-2} \\ Pr[H(x',y') - \frac{N}{2} - 2\sqrt{N} < -\sqrt{N}] &\leq e^{-2} \\ Pr[H(x',y') < \frac{N}{2} + \sqrt{N}] &\leq e^{-2} \\ Pr[H(x',y') > \frac{N}{2} + \sqrt{N}] &\geq 1 - e^{-2} \end{aligned}$$

Similarly, when $u_j = +1$ we will have

$$Pr[H(x', y') < \frac{N}{2} - \sqrt{N}] \geq 1 - e^{-2}$$

And note that $1 - e^{-2} \approx 0.8647$, so this is quite high probability.

So, what have we actually done? To tie things together, we began with inputs to the indexing problem, x and y, and transformed them into vectors u and e_j. From there, we used randomness to draw $N = 4n/c^2$ vectors from the uniform distribution over $\{-1, +1\}^n$. Then, we created inputs to the GapHD problem by defining $x'_k = \text{sign}(u \cdot r^k)$ and $y'_k = r^k_j$. It happens to be that, in the new inputs, the chance that bits are different is governed by u_j alone. When $u_j = -1$, the transformed inputs x' and y' have a high probability of being the "large difference" case of the GapHD problem, and when $u_j = +1$, there is a high probability of the "small difference" case of the GapHD problem occurring. Overall, our process turns an instance of the indexing problem to one of the two instances of the GapHD problem with high probability.

What this means, then, is that any one-way randomized communication protocol that solves GapHD with high probability must use $\Omega(n)$ bits of communication. If GapHD can be solved with better (i.e. sublinear) communication, then the indexing problem can be solved with high probability and sublinear communication by applying the reduction and using this sublinear scheme, but this contradicts that the communication complexity of indexing is $\Omega(n)$.

$\square$

7 An application to streaming algorithms

Communication complexity has yielded interesting insights in what functions may quickly be computed between two parties, but it has proved equally useful in its applications to proving upper and lower bounds of complexity of other problems. An example of this application is found in the 2003 Liben-Nowell et al. paper "Finding Longest Increasing and Common Subsequences in Streaming Data," in which set disjointness, a well-known communication complexity problem, is used to prove the best-case performance of the longest common subsequence (LCS) and the longest increasing subsequence (LIS) streaming algorithms. In addition, the paper provides and analyses a streaming algorithm for LIS, though that is mostly unrelated to this write up's central discussion of communication complexity.

It is helpful to define each of the above-mentioned problems.

Definition 7.1 (Set-disjointness)**.** Set-disjointness is defined as the following situation: Parties A and B each hold an n-bit string, s_A and s_B respectively. They must decide whether there exists an index in which both strings have a 1:

$$\exists i \text{ s.t. } s_A[i] = s_B[i] = 1$$

Phrased differently, A and B must decide whether $s_A \wedge s_B \neq 0$.

Definition 7.2 (Longest increasing subsequence)**.** LIS is defined as determining the longest increasing subsequence of an input sequence $S = x_1, x_2, ..., x_n$, where an increasing subsequence of arbitrary length k is defined as a sequence $x_{i_1}, x_{i_2}, ..., x_{i_k}$ such that $x_{i_1} < x_{i_2} < ... < x_{i_k}$ and $i_1 < i_2 < ... < i_k$. Note that this subsequence may not be contiguous in S.

Definition 7.3 (Longest common subsequence)**.** LCS is a streaming problem defined as determining the longest common subsequence of input sequences S and T where a common subsequence is a sequence present in both inputs may not be contiguous but is be in order.

For the sake of the relevant proofs, LIS and LCS can both be thought of as determining the length of their important subsequences rather than determining the subsequence itself. Furthermore, the following lemma will be crucial in proving the lowerbounds of LIS and LCS.

Lemma 7.1. *C(Set-Disjointness) $\geq n$.*

Proof. This result follows from the observation that $S = \{(A, \{1, 2, ..., n\} \backslash A) | A \subseteq \{1, 2, ...n\}\}$ constitutes a fooling set for Set-Disjointness. All pairs of strings in S are obviously disjoint due to being complements. Furthermore, if one were to take (x_1, y_1) and (x_2, y_2) from S and create the new pairs (x_1, y_2) and (x_2, y_1), at most one would constitute a disjoint pair; because pairs are unique, there must be at least one index in the disjoint pair where both strings lack a specific item. This is mirrored in the non-disjoint pair as both strings containing that item, thus making them jointed. S contains 2^n pairs, thus $C(\text{Set-Disjointness}) \geq n$. □

Now that all the ingredients for the proof have been prepared, we are able to begin discussing Liben-Nowell et al.'s two proofs.

Theorem 7.1. *Any streaming algorithm aiming to determine whether the length of the LIS of a sequence of n items is $\geq k$ will require $\Omega(k)$ bits of memory where $n = \Omega(k^2)$.*

Proof. The authors prove the above theorem through a reduction argument, which starts by taking an instance of $Set - Disjointness$ and reducing it to a relevant LIS instance. By doing so, it is possible to lowerbound the storage efficiency of LIS streaming algorithm by $Set - Disjointness$' communication complexity.

This write up will skip over the particular details of the reduction detailed in the paper and only focus on the specifics that are relevant for the entire proof. We will refer to the reduction of $Set - Disjointness$' n-bit inputs s_A and s_B to an input stream for LIS as $LIS - Stream$. $LIS - Stream$ has the following relevant properties.

1. $LIS(LIS - Stream(s_A, s_B)) \geq n + 1 \Leftrightarrow Set - Disjoint(s_A, s_B)$. In layman's terms, short LIS means that the strings are disjoint; long LIS mean that they intersect.

2. $LIS - Stream$ reduces s_A and s_B to a stream of items, the first part of which is only dependent on s_A and the latter half of which is only dependent on s_B. In other words, the sequence produced by $LIS - Stream(s_A, s_B)$ is equivalent to a sequence produced by some reduction $LIS - Stream_A(s_A)$ followed by some $LIS - Stream_B(s_B)$.

Now suppose that there exists some streaming algorithm for deciding whether the LIS of a sequence S is of length k, $\mathcal{A}(S)$. In order to decide a $Set - Disjointness$ instance, communication party A will produce $LIS - Stream_A(s_A)$, run the first part of $\mathcal{A}$ on it, and communicate $\mathcal{A}$'s memory state to B in some arbitrary efficient encoding; party B will likewise produce $LIS - Stream_B(s_B)$, receive A's communication, and run the second part of $\mathcal{A}$ using its received data and its part of the sequence. If the sequence is found to be longer than $n + 1$, it is known that s_A and s_B are intersect; otherwise, they are disjoint.

This bisection of $\mathcal{A}$ is possible because it is a streaming algorithm; necessarily, there must be some point in $\mathcal{A}$'s runtime where it has done computation only on the first part of its input sequence. By this logic, the ultimate output of $\mathcal{A}$ can be thought of being dependent on the second part of its input as well as its memory state immediately after reading the first part of its input. Therefore, by $LIS - Stream$'s second property, party B can finish $\mathcal{A}(LIS - Stream(s_A, s_B))$ after being given $\mathcal{A}$'s memory state after computation on $LIS - Stream_A(s_A)$.

Following this, it is apparent that the bits communicated in the above defined protocol for $Set - Disjointness$ is equal to the amount of memory that $\mathcal{A}$ uses to do computation on $LIS - Stream_A(s_A)$. From Lemma 7.1, this means that the storage efficiency of $\mathcal{A}$ must be at least linear; if this were not the case,

$Set - Disjointess$ would be able to be decided in fewer than a linear amount of communicated bits. $\square$

Theorem 7.2. *Any streaming algorithm aiming to determine whether the length of the LCS of two sequences of n items is $\geq k$ requires $\Omega(k)$ bits of memory.*

Proof. Liben-Nowell et al.'s second argument for the lower bound on determining the length of the LCS is very similar to the one presented in the proof for LIS. Their first conclusion is stronger, as it states that streaming algorithm which approximates the length of an n-item sequence's LCS within a ratio ρ requires $\Omega(\frac{n^2}{\rho})$ memory. For the sake of this write up, we will only cover their second conclusion, the lower bound for exact LCS. Interestingly, their exact LCS reduction is constructed such that the lower bound applies even when the input sequences are both permutations of the same set.

The reduction from $Set - Disjointness$ instances to an exact LCS will be called $LCS - Stream$. $LCS - Stream$ has these following properties.

1. $LCS(LCS - Stream(s_A, s_B)) \geq \frac{n}{2} + 2 \Leftrightarrow Set - Disjoint(s_A, s_B)$
2. Much like the previous proof, the first part of $LCS - Stream$'s output sequence is only dependent on s_A and the latter half of which is only dependent on s_B. In other words, the sequence produced by $LCS - Stream(s_A, s_B)$ is equivalent to a sequence produced by some reduction $LCS - Stream_A(s_A)$ followed by some $LCS - Stream_B(s_B)$.

We likewise assume that some algorithm $\mathcal{A}$ can decide whether the LCS of two sequences is longer than a given k. By similar reasoning as the last proof, we can decide $Set - Disjointness$ by having each party run their part of the reduction on their input, having one run $\mathcal{A}$ on their reduced input, having the same one communicate the state of $\mathcal{A}$ to the other, and having the other finish $\mathcal{A}$ on their reduced input. If $\mathcal{A}$ finds that the LCS of the two reduced sequences is longer than $\frac{n}{2} + 2$, then we know s_A and s_B intersect; otherwise, they are disjoint.

By similar reasoning as the last proof, $\mathcal{A}$ cannot use less than a linear amount of storage as this would imply that LCS could be decided in less than a linear amount of communication. $\square$

8 References

1 Arora, S., & Barak, B. (2016). Computational complexity: A modern approach. Cambridge University Press.

2 Jayram, T. S., Kumar, R., & Sivakumar, D. (2008). The One-Way Communication Complexity of Hamming Distance. Theory of Computing, 4, 129–135.

3 Lee, T., & Shraibman, A. (2009). Lower bounds in communication complexity. Now Publishing.

4 Liben-Nowell, D., Vee, E., & Zhu, A. (2006). Finding longest increasing and common subsequences in streaming data. Journal of Combinatorial Optimization, 11(2), 155–175.