Communication complexity

Victor Vasiliev

March 9, 2016

1 Introduction

Traditionally, complexity theory considers the setting in which a Turing machine is given some problem, and has to solve it within some limitation on the computational resources (time, space, parallelism, or some other form). In communication complexity, the computational resources are typically unlimited. The premise, however, is that instead of one machine receiving the entirety of input, there will be two (or more!) parties performing computation (we will call them Alice and Bob), and they will receive two different parts of the input, x and y. They will have to compute some function $f(x, y)$, and the resource we are interested in limiting is the amount of bits they can communicate. We will refer to the procedure by which they perform the computation as *the protocol*.

Definition 1. *For a given function $f(x, y)$, the deterministic communication complexity $D(f)$ is the number of bits used in the worst case by the smallest protocol that computes $f(x, y)$.*

Typically, we will be interested in the case when x and y are binary strings of the same size, and f returns either "yes" or "no".

Theorem 2. *For any $f(x, y)$, $D(f) \leq n + 1$.*

Examples of interesting problems we are going to discuss (here, x and y are n-bit strings):

1. $\mathrm{XOR}_n(x, y)$ – compute parity of string xy.
2. $\mathrm{EQ}_n(x, y)$ – determine whether $x = y$
3. $\mathrm{DISJ}_n(x, y)$ – determine if there exists i such that $x_i = y_i$.

DISJ is commonly known as the *disjointness problem*, because it represents the process in which both Alice and Bob have elements of some set and they are trying to figre out whether they have an element in common.

Example. $\mathrm{XOR}(x, y)$ can be easily computed in $O(1)$ bits because addition modulo two is commutative: first Alice computes parity of x, then she sends it to Bob, and Bob computes the parity of y, which he finally XORs with the pairty of x received from Alice.

2 Monochromatic partition

One way to analyze the problem is through a matrix representing the correct answers for each corresponding input.

Definition 3. *For any $f(x, y) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, a communication matrix M_f is a $2^n \times 2^n$-matrix such that each row refers to some particular value of x, each column refers to some particular value of y, and each cell is $f(x, y)$ for those x and y.*

In this setup, each protocol π partitions matrix M into some fixed number of monochromatic rectangles, that is, recatngles, containing either all 0 or all 1. Here rectangle should be interpreted not as a geometric shape, but rather as a subset of $\{0,1\}^n \times \{0,1\}^n$ that can be represented as a product of two sets in $\{0,1\}^n$.

Theorem 4. *Any deterministic protocol π that computes f correctly using only b bits partitions M_f into at most 2^c monochromatic rectangles.*

Corollary 5. $D(\mathrm{EQ}_n) = n + 1$.

Proof. M_f is an identity matrix, meaning it has 1s across the diagonal and 0s elsewhere. No rectangle with a 1 can contain another 1, since otherwise it would also contain items off diagonal, which are 0. Given that a seperate rectangle for 0s is required, there are at least 2^c rectangles, meaning the complexity is at least $n + 1$. The equality follows from the trivial protocol. □

3 Randomization

As usual, when a problem is hard to solve deterministically, it would make sense to ask if the randomness would help.

Definition 6. *A randomized protocol π_ε decides function $f(x, y)$ with probability $1 - \varepsilon$ if for any $x, y \in \{0,1\}^n$, $\Pr[f(x, y) \neq \pi_\varepsilon(x, y)] \leq \varepsilon$.*

Definition 7. *Function $f(x, y) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ has randomized complexity $R_\varepsilon(f) = c$ if the optimal randomized protocol π_ε decides it successfully with probability $1 - \varepsilon$ using at most c bits.*

An important technical detail in this is that there are two ways to construct randomized protocol. The first one is where each party has access to its own random generator (*private coins*). The second one is the one when both parties can receive randomness from the sky (*public coins*).

Before, we have demonstrated that EQ cannot be solved deterministically better than naive algorithm. If we use randomized approach, however, the outlook is more positive. Let us start with an algorithm that uses public coins.

EQ with Public Coins

The essence of this algorithm is that Alice and Bob are exchanging parities of the random subsets of bits in the input.

1. Both Alice and Bob pick n-bit random string g.
2. Alice computes $g \cdot x \pmod 2$ and sends the resulting bit to Bob.
3. Bob computes $g \cdot y \pmod 2$, and outputs 0 if they do not match.
4. Repeat the procedure above for k times, and return 1 if the inner products have always matched.

If $x = y$, the algorithm will always succeed. If $x \neq y$, then the following will happen. There will be some number m of bits which are not equal, and the number l of bits which will be the part of the parity computation is binomially distributed. The probability that l is even is at most $\frac{1}{2}$,[1] hence the probability that none of the attempts succeeds is 2^{-k}. This means we can acheieve arbitrarily low error probability ε in $O(-\log \varepsilon)$ bits. Note that the number of bits is constant in the size of input.

Equivalence of Public and Private coins

The algorithm we have described above is usign the public coins model. The private coins model is naturally more limited, but we can show that given an algorithm using public coins, we can create an algorithm that uses private coins, albeit with some overhead.

Theorem 8. *Given a protocol π^{priv}, we can construct a protocol π^{pub} that has an error probability of $\varepsilon + \delta$ and uses $O\left(\log n + \log\left(\frac{1}{\delta}\right)\right)$.*

We will further refer to $R_\varepsilon(f)$ as the public coin complexity.

4 Distributional complexity

Another approach to introducing randomness is as follows. Assume we have many inputs, which are distributed according to some probability distribution. Then we can analyze the probability that a given deterministic protocol π will succeed in deciding $f(x, y)$.

Definition 9. *Function $f(x, y)$ has distributional complexity $D^\mu_\varepsilon(f) = b$ if there exists a protocol π such that $\Pr_{x,y\sim\mu}[\pi(x, y) \neq f(x, y)] \leq \varepsilon$.*

Theorem 10. $R_\varepsilon(f) = \max_\mu D^\mu_\varepsilon(f)$.

[1] In fact, it is $\frac{1}{2}$. See `http://math.stackexchange.com/a/82949`

One of the most useful theorems of communication complexity is

Theorem 11 (Kalyanasundaram and Schintger)**.** *For some μ and a sufficiently small ε, $D^{\mu}(\text{DISJ}) = \Theta(n)$.*

Corollary 12. *$R_{\varepsilon}(\text{DISJ}) = \Theta(n)$.*

5 Streaming complexity

In many applications, the volume of the input is so large that it might not fit into the memory of an individual machine. In that case, it might be reasonable to feed the input into the machine in chunks. The streaming complexity of a problem is the smallest amount of memory a streaming algorithm could use to solve the problem.

Example. Minimum, maximum and parity are all computable in $O(1)$ space, since they all require storing exactly one element of the stream.

Now, consider the problem of finding how often does the most frequent element in the stream appear. This problem can be only solved by storing $O(n)$ of input.

Proof. Assume there exists a streaming algorithm that uses $s = o(n)$ bits to find the number of times the most frequent element appears in the input. Then consider the following protocol for solving $\text{DISJ}(x, y)$:

1. Alice runs the streaming algorithm on x, and sends its memory (s bits) to Bob.
2. Bob resumes the algorithm from the state Alice sent him, and feeds y into the algorithm.
3. If algorithm outputs 2 or more, Bob returns 1. Otherwise, Bob returns 0.

This algorithm solves $\text{DISJ}(x, y)$ in $o(n)$ bits of communication. But this is impossible from the bound we have proven on DISJ. Hence no such algorithm can exist. □

6 VLSI

A VLSI chip can be modelled as a grid of size $a \times b$, where there are n bits coming into the chip, $x_1, x_2, \ldots x_n$, and one wire coming out of the chip. Each point of the grid may have a logic gate on it, and in our model it takes every game some fixed timestep to produce the output based on the input.

The two parameters which determine the efficiency of the chip are the area $A = ab$ and the time T (in timesteps) it takes to compute the result. Assume that we seperate the input x into the chip into two parts, x' and x''. Then

$$AT^2 \geq D(f)^2$$