Lecture 2: Communication complexity and Circuits

Anup Rao

January 5, 2023

Communication Complexity

Communication complexity has been a very useful model for proving lower bounds. In this model, there are two parties Alice and Bob. Alice is given an n-bit string x, and Bob is given an n-bit string y. In order to compute a function $f(x, y)$, they exchange messages about their inputs.

So, Alice sends Bob a message $m_1(x)$. Bob responds with a message $m_2(y, m_1)$. In this way, they exchange messages until someone knows the value of f. The communication complexity of f is the minimum number of bits that must be communicated before the players know the value of f.

To formally define a communication protocol, we use a protocol tree. See Figure 1. This is a rooted binary tree where every node is associated with one of the players, and associated with a boolean function. To execute the protocol, Alice and Bob start at the root of the tree. If the root is owen by Alice, she evaluates the function associated with the root on her input and sends the result to Bob. The result of the evaluation determines which of the two children the protocol moves to next. In this way, the players reach a leaf of the tree, which is labeled with the output of the computation. The cost of the tree is simply its depth, which determines the maximum number of bits that may be exchanged during any execution of the protocol.

The main drawback of this model is that it is not very practical: just because a function has an efficient communication protocol doesn't mean that we can compute it efficiently in practice. For example, in this model, one can compute any boolean function of x with 1 bit of communication. However most functions cannot be computed efficiently in practice.

The main advantage of this model is that almost every other computational model seems to involve communication. So lower bounds on the communication complexity of functions are extremely useful to prove lower bounds on the complexity of computing functions in other models of computation.

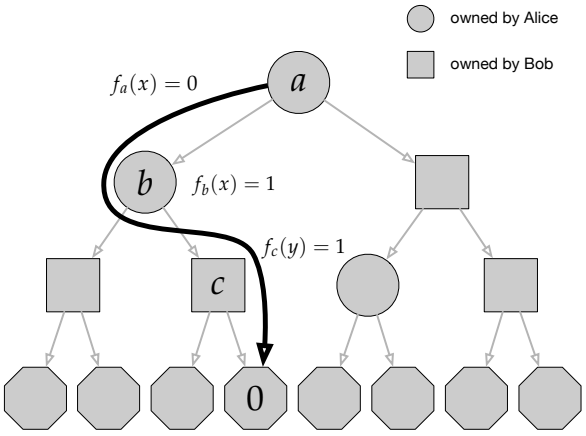

Figure 1: An execution of a protocol.

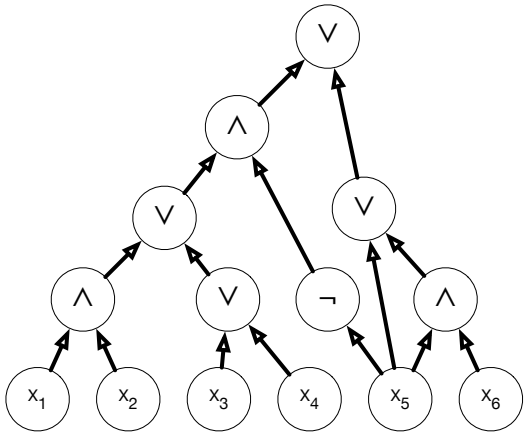

Figure 2: An example of a boolean circuit.

Boolean Circuits

A *boolean circuit* computing a function $f : \{0,1\}^n \to \{0,1\}$ is a directed acyclic graph with the following properties. Every vertex (also called a gate) has at most 2 edges coming in to it. If there are 0 edges coming in, then the vertex is labeled with an input variable x_i, or the constants 0 or 1. Otherwise, the vertex is labeled with one of the boolean operators $\wedge, \vee, \neg$, and computes the specified operation on the bits that come in along the incoming edges. One of the gates in the circuit is designated the output node. This is the node whose value is the output of the circuit.

When every gate has out-degree at most 1, the circuit is called a *formula*. In the case of a formula, the graph of the circuit looks like a tree after edges have been converted into undirected edges.

A circuit can also be viewed as a program in a simple programming language, where every line is an assignment. For example, the

circuit in Figure 2 is equivalent to this program:

1. $y_1 = x_1 \wedge x_2$
2. $y_2 = x_3 \vee x_4$
3. $y_3 = \neg x_5$
4. $y_4 = x_5 \wedge x_6$
5. $y_5 = y_1 \vee y_2$
6. $y_6 = x_5 \vee y_4$
7. $y_7 = y_5 \wedge y_3$
8. $y_8 = y_7 \vee y_6$

There are two major quantities we can measure to capture the complexity of a circuit:

Definition 1. *The* size *of the circuit is the number of gates in the circuit.*

Since every gate in the circuit has at most 2 incoming edges, the size of the circuit is proportional to the number of edges in the graph that defines the circuit:

Fact 2. *The size of the circuit is the same as the number of edges in the circuit, up to a factor of* 2*.*

We can also measure the *depth* of the circuit:

Definition 3. *The* depth *of the circuit is the length of the longest input to output path.*

The depth complexity is a measure of how much parallel time it takes to compute the function.

Just like branching programs, boolean circuits can complete *every* function:

Theorem 4. *Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size at most $O(2^n)$.*

Proof We construct the circuit recursively. When $n = 1$, there is clearly a constant sized circuit that computes f, since f must be either a constant, x_1 or $\neg x_1$.

For $n > 1$, let f_0 denote the function on $n-1$ bits given by $f_0(x) = f(x,0)$, and $f_1(x) = f(x,1)$. Then by induction we can compute f_0, f_1 recursively, and combine them using the value of the last bit to obtain f, as in Figure 3. When $x_n = 1$, the circuit outputs $f_1(x_1, \ldots, x_{n-1})$, and when $x_n = 0$, the circuit outputs $f_0(x_1, \ldots, x_{n-1})$.

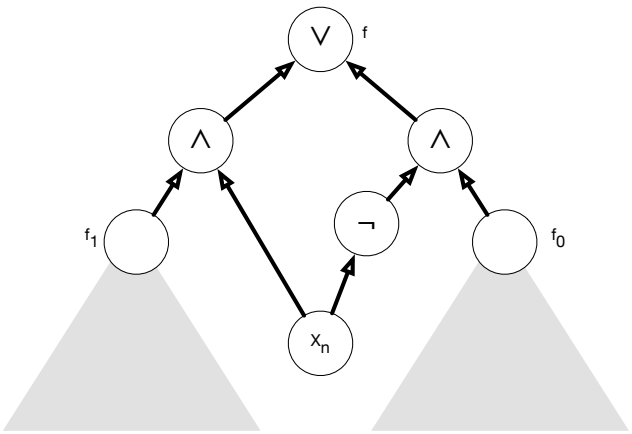

Figure 3: Recursive construction of a circuit for f.

If S_n is the size of the resulting circuit when the underlying function takes an n bit input, we have proved that

$$S_n \leq 2S_{n-1} + 5.$$

Expanding this recurrence, and using the fact that $S_1 \leq 5$, we get that

$$S_n \leq \sum_{i=0}^{n} 2^i 5 = 5 \cdot (2^{n+1} - 1) < 10 \cdot 2^n,$$

where here we used the formula for computing the sum of a geometric series. ■

The above theorem is not the best result we know about this subject. In fact, we know:

Theorem 5. *Every function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size at most $O(2^n/n)$.*

Proof We will use a recursive construction, but stop the recursion at a certain point. For a parameter t, we start by computing *every* function of the first t bits of the input. There are 2^{2^t} such functions, and each one can be computed by a circuit of size $O(2^t)$ using Theorem 4, so we can compute every function using at most $O(2^t \cdot 2^{2^t}) \leq O(2^{t+2^t})$ gates.

To compute the function f, we use the recursive construction defined in the proof of Theorem 4 for $n - t$ steps. After $n - t$ steps, we have put down $O(2^{n-t})$ gates, and need to compute functions on the first t bits, but since we have already computed every such function, we are done. The size of the final circuit is thus $O\left(2^{t+2^t} + 2^{n-t}\right)$. Setting $t = \log n - 1$, we get a circuit of size

$$O(2^{\log n - 1 + n/2} + 2^{n - \log n + 1}) \leq O(2^n/n).$$

■