9

Circuits and Proofs

ALTHOUGH COMMUNICATION COMPLEXITY ostensibly studies the amount of communication needed between parties that are far apart, it is deeply involved in our understanding of many other concrete computational models and discrete systems. In this chapter, we discuss applications of communication complexity to Boolean circuits and proof systems.

Boolean Circuits

BOOLEAN CIRCUITS ARE THE MOST NATURAL model for computing Boolean functions. A *Boolean circuit* is a directed acyclic graph whose vertices, often called *gates*, are associated with either Boolean operators or input variables. Every gate with in-degree 0 corresponds to an input variable, the negation of an input variable, or a constant bit. All other gates compute either the logical AND (denoted by the symbol $\wedge$) or the OR (denoted by the symbol $\vee$) of the inputs that feed into them. Usually, the fan-in of the gates is restricted to being 2. We adopt this convention, unless we explicitly state otherwise. See Figure 9.1 for an illustration.

Every gate v in a circuit naturally computes a Boolean function f_v of the inputs to the circuit. We say that a circuit computes a function f if $f = f_v$ for some gate v in it.

Every circuit is associated with two standard complexity measures. The *size* of the circuit is the number of gates. It corresponds to the number of operations the circuit performs. The *depth* of the circuit is the length of the longest directed path in the underlying graph. The depth corresponds to the parallel time it takes the computation to end, using many processors.

We thus get a measure of computational complexity – circuits have costs and functions have complexities. Every Boolean function f can be computed by a Boolean circuit. The circuit complexity of f is the size of the smallest circuit that computes it. Understanding the circuit complexity of interesting functions is a fundamental problem in computer science.

Boolean circuits can efficiently simulate algorithms. Any function that can be computed by an algorithm in $T(n)$ steps can also be

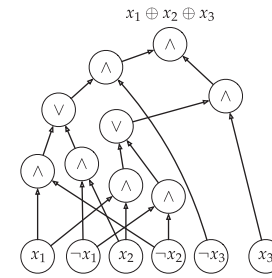

Figure 9.1 A circuit computing the parity $x_1 \oplus x_2 \oplus x_3$. This circuit has size 15 and depth 4.

One may consider circuits where every gate has fan-in 2 and computes an arbitrary function of its inputs. This only changes the size and depth of the circuit by a constant factor.

computed by circuits of size approximately $T(n)$. So, to prove lower bounds on time complexity, it suffices to prove that there are no small circuits that can carry out the computation.

For example, a super-polynomial lower bound on the circuit size of an NP problem would imply that P ≠ NP, resolving the most important open problem in computer science.

Every Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit[1] of depth n and size at most $O(2^n/n)$. Counting arguments imply that almost every function requires circuits of exponential size.[2] However, we do not know of any emphexplicit function for which we can prove even a super-linear lower bound.

[1] Lupanov, 1958.

[2] Shannon, 1949.

The number of circuits of size s is at most $2^{O(s \log s)}$. The number of functions f is 2^{2^n}. So, if $s \ll 2^n/n$, one cannot hope to compute every function with a circuit of size s.

We do not yet understand in circuit complexity is the power of depth:

Open Problem 9.1 *Can every function that is computable using circuits of size polynomial in n be computed by circuits of depth $O(\log n)$?*

We now describe a general connection between circuit complexity and communication complexity. We focus on two restricted families of circuits. A *formula* is a circuit whose underlying graph is a tree. Equivalently, the fan-out of every gate is 1. Every circuit of depth d can always be turned into a formula whose size is at most 2^d, and depth is at most d. A *monotone* circuit is a circuit that does not use negated variables. A monotone circuit computes a *monotone* function; $f(y) \geq f(x)$ whenever $y_i \geq x_i$ for all i.

Counting arguments imply that there is a constant ϵ such that the set of functions computable by size $s \log s$ circuits is strictly larger than the set of functions computable by size ϵs circuits. Similarly, counting arguments show that circuits of depth d compute a bigger set of functions than those computable in depth ϵd.

Several restricted classes of circuits are not discussed in this book. We focus on methods related to communication complexity.

Karchmer-Wigderson Games

EVERY BOOLEAN FUNCTION defines a communication problem via its *Karchmer-Wigderson* game.[3] In the game defined by the function $f : \{0,1\}^n \to \{0,1\}$, Alice gets $x \in f^{-1}(0)$, Bob gets $y \in f^{-1}(1)$, and they seek to find $i \in [n]$ such that $x_i \neq y_i$. When f is monotone, we define the *monotone* Karchmer-Wigderson game as follows: Alice and Bob want to find an index i such that $x_i < y_i$.

[3] Karchmer and Wigderson, 1990.

In the Karchmer-Wigderson game, Alice and Bob are computing a relation rather than a function – there may be many indices i with the property they seek.

The basic observation is that circuit-depth is equivalent to communication complexity, as the following two lemmas show.

Lemma 9.2 *A circuit of depth d computing f yields a length d deterministic protocol for the associated game. If the circuit is monotone, the protocol solves the monotone game.*

Proof The construction of the protocol is by induction on the depth of the circuit. If the top gate in the circuit is an AND gate ($f = g \wedge h$), then either $g(x) = 0$ or $h(x) = 0$, while $g(y) = h(y) = 1$. Alice can announce whether $g(x)$ or $h(x)$ is 0, and the parties can continue the protocol using g or h. Similarly if $f = g \vee h$, Bob can announce whether $g(y) = 1$ or $h(y) = 1$, and the parties then continue with either g or h. If f is the negation of g, then the parties can continue the protocol using g, without communicating at all. If f is the ith input variable, the parties identify an index i for which $x_i \neq y_i$.

When the circuit is monotone, the same simulation finds an index i such that $x_i = 0$ and $y_i = 1$ because there are no negated variables.

The topology of the circuit determines the topology of the protocol tree. Every AND gate corresponds to a node in the protocol tree where Alice speaks, every OR gate corresponds to a node where Bob speaks, and every input gate corresponds to a leaf in the protocol tree. Thus, a circuit of depth d gives a protocol of length at most d. □

If the function is constant, the Karchmer-Wigderson game is not well defined. The circuit-size is 1, and the depth is 0.

Lemma 9.3 *If the Karchmer-Wigderson game for a function f can be solved with d bits of communication, then there is a circuit of depth d computing f. If f is monotone, and the monotone game can be solved with d bits of communication, then there is a monotone circuit of depth d computing f.*

Proof We shall prove, by induction on d, that for any nonempty sets $A \subseteq f^{-1}(0)$ and $B \subseteq f^{-1}(1)$, the following holds. If there is a protocol such that whenever $x \in A$ is given to Alice and $y \in B$ is given to Bob, they can exchange d bits to find i such that $x_i \neq y_i$, then there is a circuit of depth d computing a Boolean function g with $g(A) = 0$ and $g(B) = 1$. When $A = f^{-1}(0)$ and $B = f^{-1}(1)$, this implies the lemma.

When $d = 0$, the protocol must have a fixed output i, and so we must have that $x_i \neq y_i$ for every $x \in A$ and $y \in B$. Thus, setting g to be the ith variable or its negation works.

Suppose $d > 0$ and Alice speaks first. Her message partitions the set A into two disjoint sets $A = A_0 \cup A_1$, where A_0 is the set of inputs that lead her to send 0 as the first message, and A_1 is the set of inputs that lead her to send 1. If one of A_0 or A_1 is empty, then we can ignore the first message, and the proof is complete. So, both A_0 and A_1 are nonempty. By induction, the two children of the root correspond to Boolean functions g_0 and g_1, with $g_0(A_0) = g_1(A_1) = 0$ and $g_0(B) = g_1(B) = 1$. Consider the circuit that takes the AND of the two gates obtained inductively and denotes the function it computes by g. For all $y \in B$, we have $g(y) = g_0(y) \wedge g_1(y) = 1$. For all $x \in A$, either $x \in A_0$ or $x \in A_1$. In either case $g(x) = g_0(x) \wedge g_1(x) = 0$. If the first bit of the protocol is sent by Bob, the proof is similar, except we take the OR of the gates obtained by induction.

If we are working with the monotone game, the resulting circuit is monotone. □

The Karchmer-Wigderson connection between circuit complexity and communication complexity is a powerful tool for proving lower bounds on circuit complexity.

Monotone Circuit-Depth Lower Bounds

A matching in a graph is a collection of disjoint edges. One of the most studied combinatorial problems is finding the largest matching in a graph. Today, we know of several efficient algorithms for solving this problem.[4]

[4] Kleinberg and Tardos, 2006.

We focus on the following decision problem. Given a graph G on n vertices, define

$$\mathsf{Match}(G) = \begin{cases} 1 & \text{if } G \text{ has a matching of size at least } n/3+1, \\ 0 & \text{otherwise.} \end{cases}$$

Because there are polynomial time algorithms for finding matchings, one can obtain polynomial sized circuits that compute Match. However, we do not know of any logarithmic depth circuits that compute Match. Here we show that there are no monotone circuits of depth $o(n)$ computing Match.[5]

[5] Raz and Wigderson, 1992.

Match is a monotone function.

Theorem 9.4 *Every monotone circuit computing* Match *has depth* $\Omega(n)$.

By Lemma 9.3, it is enough to prove a lower bound on the communication complexity of the corresponding monotone Karchmer-Wigderson game. In the monotone matching game, Alice gets a graph G with $\mathsf{Match}(G) = 1$ and Bob gets a graph H with $\mathsf{Match}(H) = 0$. Their goal is to find an edge that is in G, but not in H.

Theorem 9.5 *Any randomized protocol solving the matching game must communicate* $\Omega(n)$ *bits.*

Proof The theorem is proved by reduction to the disjointness lower bound proved in Theorem 6.19. We shall show that if there is a protocol for the monotone matching game with length c, then there is a randomized protocol with length $O(c)$ solving the disjointness problem on a universe of size $m = \Omega(n)$. By Theorem 6.19, this implies that $c \geq \Omega(n)$.

Suppose Alice and Bob get inputs $X \subseteq [m]$ and $Y \subseteq [m]$. They encode X and Y as two graphs G_X and H_Y on the vertex set $[3m+2]$. They use public randomness to permute the vertices of the graphs and feed them into the protocol for the monotone matching game. Figure 9.2 shows an example for G_X and H_Y. These graphs are constructed as follows:

Alice builds G_X***:*** For each $i \in [m]$, the graph G_X contains the edge $\{3i, 3i-1\}$ if $i \in X$ and has the edge $\{3i, 3i-2\}$ if $i \notin X$. In addition, G_X contains the edge $\{3m+1, 3m+2\}$.

The construction ensures that G_X contains a matching of size $m+1$.

Bob builds H_Y***:*** For each $i \in [m]$, if $i \in Y$ then Bob connects $3i-2$ to *all* the other $3m+1$ vertices of the graph, and if $i \notin Y$ then Bob connects $3i$ to all the other vertices.

By construction, there are m vertices so that every edge of H_Y touches one of these vertices (the gray vertices in Figure 9.2). So, H_Y does not contain a matching of size $m+1$.

If X and Y are disjoint, the outcome of the protocol must be the edge corresponding to $\{3m+1, 3m+2\}$. On the other hand, if X and Y intersect in $k > 0$ elements, then there are exactly $k+1$ edges in G_X that are not in H_Y.

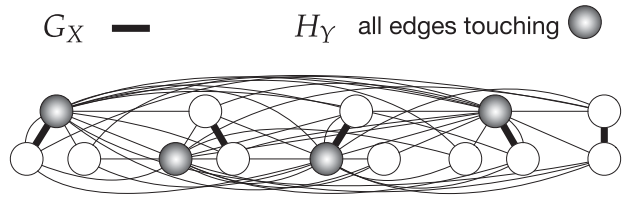

Figure 9.2 A schematic description of G_X and H_Y.

Because the graph is permuted uniformly at random before the protocol is executed, and the protocol for the game does not know the permutation, the outcome of the protocol is equally likely to be one of these $k+1$ edges. Indeed, let e and e' be two of these $k+1$ edges, and let σ be a permutation of the vertices such that σ maps the edge e to the edge e'. For every permutation τ, if the protocol outputs the edge $\tau(e)$ when it samples τ, then it outputs $\tau(e')$ when it samples $\tau \circ \sigma$.

When the sets are disjoint, the protocol outputs the edge corresponding to $\{3m+1, 3m+2\}$. When the sets are not disjoint, the output corresponds to $\{3m+1, 3m+2\}$ with probability at most $1/2$. Repeating this experiment a constant number of times, the parties are able to solve disjointness with probability of error at most $1/3$. □

Monotone Circuit-Depth Hierarchy

BOOLEAN CIRCUITS CAN BE GRADED by their depth. It is natural to conjecture that for constant k, circuits of depth $k+1$ are strictly more powerful than circuits of depth k. Communication complexity allows us to prove this in the monotone setting.

Throughout this section, we work with circuits of arbitrarily large fan-in.

Let $F = F_{n,k}$ be the full AND-OR formula with fan-in n and depth k. All noninput gates in F have fan-in exactly n. The gates of odd depth are OR gates, and the gates of even depth are AND gates. Every input gate is labeled by a distinct unnegated variable. The size of F is $O(n^k)$.

We do not know how to prove a similar statement for general circuits.

We prove that any monotone circuit of smaller depth computing F must have exponential size.

We think of F both as a formula and as a function.

Theorem 9.6 *Any monotone circuit of depth $k-1$ that computes F must have size at least $2^{\Omega(n/k)}$.*

Proof Assume that there is a monotone circuit of size s and depth $k-1$ computing F. The circuit yields a protocol for the monotone Karchmer-Wigderson game with $k-1$ rounds and length at most $O(k \log s)$.

It thus suffices to prove that the monotone game requires communication at least $n/16 - k$. We prove this by reduction to the pointer-chasing problem, that we studied in Chapter 6. In pointer-chasing, Alice and Bob are given $x, y \in [n]^n$ and want to compute $z = z(k)$, where $z(0) = 1$, and $z(1), z(2), \ldots$ are inductively defined using the rule

$$z(i) = \begin{cases} x_{z(i-1)} & \text{if } i \text{ is odd,} \\ y_{z(i-1)} & \text{if } i \text{ is even.} \end{cases}$$

Given inputs x, y to the pointer-chasing problem, the inputs x', y' in $\{0,1\}^{[n]^k}$ to F are constructed as follows. Every variable in the formula can be described by a string in $v \in [n]^k$. We say that v is consistent with x if

$$v_i = \begin{cases} x_1 & \text{when } i = 1, \\ x_{v_{i-1}} & \text{when } i \text{ is odd and not 1.} \end{cases}$$

We say that v is consistent with y if $v_i = y_{v_{i-1}}$ when i is even. Alice sets all the coordinates of x' that are consistent with her input to be 0, and all other coordinates to be 1. Bob sets all the coordinates of y' that are consistent with his input to be 1, and all other coordinates to be 0.

We now prove that $F(x') = 0$ and $F(y') = 1$. We focus on $F(x')$; a similar argument works for $F(y')$. Every a gate of depth d in the formula corresponds to a vector in $[n]^d$. We claim that every gate that corresponds to a vector that is consistent with x evaluates to 0 on x'. This is true for the input gates at depth k because that is how we set the variables in x'. For gates at depth $d < k$, if the gate is an AND gate then one of its children is consistent with x and so evaluate to 0, and if the gate is an OR gate then all of its children are consistent with x and so evaluate to 0.

For every x, y, there is a unique input gate v that is consistent with both x and y. This gate is the output $v = z(k)$ of the pointer-chasing problem. The only place where x' is 0 and y' is 1 is the vth entry.

Any protocol for the monotone Karchmer-Wigderson game, therefore, gives a protocol solving the pointer-chasing problem. By Theorem 6.18, the communication of the game must be at least $n/16 - k$. □

Boolean Formulas

FORMULAS CORRESPOND TO COMPUTATIONS that use each sub-computation exactly once. One immediate consequence of the Karchmer-Wigderson connection is a sharp lower bound on the formula-size of parity. In Chapter 1, we proved that solving the Karchmer-Wigderson games for parity requires at least $2\log n - O(1)$ bits of communication. This shows that its circuit-depth is at least $2\log n - O(1)$. The formula complexity of parity is therefore $\Theta(n^2)$.

A similar lower bound holds for the circuit-depth of majority. See Exercise 1.4.

For example, parity has linear-size formulas using ⊕ gates, but requires quadratic-size formulas using AND, OR and NOT gates.

When it comes to formulas, the choice of basis can affect the formula size by more than a constant factor. Nevertheless, one can prove super-linear lower bounds even when allowing each gate to compute an arbitrary function of two bits.

Consider the function $\mathsf{Distinct} : [2n]^{n+1} \to \{0,1\}$, defined as

$$\mathsf{Distinct}(x_1, \ldots, x_{n+1}) = \begin{cases} 1 & \text{if } x_1, \ldots, x_{n+1} \text{ are distinct,} \\ 0 & \text{else.} \end{cases}$$

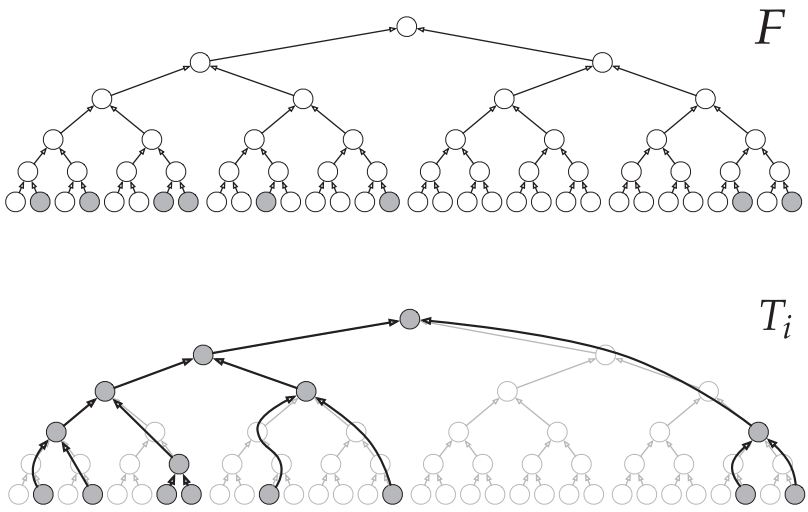

Figure 9.3 A formula F and the tree T_i that corresponds to the input gates of x_i. Shaded input gates correspond to x_i.

Distinct is a Boolean function that depends on $O(n \log(n))$ bits. We shall prove:[6]

[6] Neciporuk, 1966; and Klauck, 2007.

Theorem 9.7 *Any formula computing* Distinct *must have at least* $n^2 - O(n \log n)$ *input gates.*

We start by proving a simple communication complexity lower bound. Suppose Alice is given n numbers $y_1, \ldots, y_n \in [2n]$, and Bob is given $z \in [2n]$. They want to compute $\mathsf{Distinct}(y_1, \ldots, y_n, z)$.

Lemma 9.8 *If there is a* 1*-round protocol where Alice sends Bob* t *bits and Bob outputs* $\mathsf{Distinct}(y_1, \ldots, y_n, z)$*, then* $t \geq \log \binom{2n}{n} \geq 2n - O(\log n)$.

Proof It is enough to consider the case when $y_1, \ldots, y_n$ are distinct elements. In this case, Alice's message must determine $S = \{y_1, \ldots, y_n\}$, or else Bob will not be able to compute Distinct. This is because if $S \neq S'$ are two sets of size n that are consistent with Alice's message, then there must be $z \in S$ such that $z \notin S'$. The element z is distinct from S', but not from S.

The number of bits transmitted by Alice must, therefore, be at least $\log \binom{2n}{n} \geq 2n - O(\log n)$. □

The middle binomial coefficient is maximal, so $\binom{2n}{n} \geq \frac{2^{2n}}{n+1}$. A more accurate bound using Stirling's approximation gives $\binom{2n}{n} = \Theta(\frac{2^{2n}}{\sqrt{n}})$.

We are ready to prove the formula lower bound:

Proof of Theorem 9.7 Suppose there is a formula F computing Distinct using s gates. Each input gate in the formula reads a bit of one of the numbers x_i. For each $i \in [n+1]$ we define the tree T_i as follows (see Figure 9.3). Every vertex of T_i corresponds to a gate in F. Start by discarding all the gates in F that do not depend on x_i. In the graph that remains, iteratively replace every gate that has only one input feeding into it with an edge connecting its input to its output.

Now, suppose Alice knows all of the input numbers except x_i, Bob knows x_i, and Alice and Bob want to compute $\mathsf{Distinct}(x_1, \ldots, x_{n+1})$.

They can use the tree T_i to carry out the computation efficiently. Bob already knows the values at the leaves of T_i. Every gate v in T_i computes a Boolean function f_v, which depends on gates in T_i and some number of Alice's inputs. There are $2^{2^2} = 2^4$ Boolean functions that depend on two variables, so Alice can send 4 bits to Bob to indicate which of these functions he should use to compute $f_v(x_1, \ldots, x_{n+1})$ using the two inputs that correspond to gates of T_i. Using this information, Bob can compute Distinct. The overall communication is at most four times the number of vertices in T_i.

Because F has only s gates, there must be some i for which T_i has at most $\ell = s/n$ leaves. If m denotes the number of vertices of T_i, and e the number of edges in T_i, then we must have $e = m - 1$ because T_i is a tree. Counting the number of edges by adding up the degrees of the vertices, we have

$$2(m-1) = 2e \geq 3(m - \ell - 1) + \ell.$$

So, $m \leq 2\ell + 1 \leq 2s/n + 1$.

By Lemma 9.8, we get $2s/n + 1 \geq 2n - O(\log n)$, proving the theorem. □

Formulas with Arbitrary Gates

Communication complexity allows us to prove nontrivial lower bounds even when gates are allowed to compute arbitrary functions of a linear number of variables.[7] Suppose we want to express a function $f : \{0,1\}^n \to \{0,1\}$ as

$$f = g(g_1, \ldots, g_k),$$

where each of the functions $g_1, \ldots, g_k$ depends on at most $2n/3$ input bits. What is the minimum k required?

We can represent Distinct in this form with $k = O(\log n)$.[8] Nevertheless, the closely related scrambled distinctness function requires $k \geq n^{\Omega(1)}$. Assume n is a power of 2. For a subset S of $[n \log(2n)]$ of size $\log(2n)$, and $b \in \{0,1\}^{n \log(2n)}$, define SDistinct$(S, b)$ as follows. Use the coordinates of S in b to define a number $z \in [2n]$. Use the remaining bits of b to define $y_1, \ldots, y_{n-1} \subseteq [2n]$. Output Distinct$(y_1, \ldots, y_{n-1}, z)$.

Theorem 9.9 SDistinct(S, b) *requires* $k \geq n^{\Omega(1)}$.

Proof As in the formula lower bound, we shall appeal to Lemma 9.8. Suppose we can write SDistinct as $g(g_1, \ldots, g_k)$, where each of the gates g_i depends on at most 2/3rds of the input variables.

We claim that if k is small, there must be some S for which every gate g_i reads at most $\frac{4}{5}\log(2n)$ of inputs that correspond to S. Indeed, suppose we pick the elements of S independently and uniformly at random. For each $i \in [k]$, the expected number of coordinates of S read by g_i is at most $\frac{2}{3}\log(2n)$. By the Chernoff-Hoeffding bound, the probability that more than $\frac{4}{5}\log(2n)$ of the coordinates are read in g_i is at most $e^{-\Omega(\log(2n))} = n^{-\gamma}$, for some constant $\gamma > 0$. The probability that the $\log(2n)$ coordinates sampled are not all distinct is at most $\log^2(2n)/n$.

[7] Hrubes and Rao, 2015.

Counting arguments show that most functions f require $k = \Omega(n)$.

[8] To see this, let $S_1, \ldots, S_k \subset [n]$ be sets of size $n/2$ so that that for every $i, j \in [n]$, there is some set of the sequence that contains both i, j. One can show that a random choice of $O(\log n)$ sets satisfies this property with positive probability. Use these sets to construct a formula. For each i, let g_i be the function that reads the numbers x_ℓ for $\ell \in S_i$, and outputs 1 if and only if these numbers are distinct. Let g be the OR function.

It remains an open problem to find an explicit function for which $k = \Omega(n)$.

The input to SDistinct can be encoded using $n \log(2n) + \log^2 n$ bits.

Overall, if $k < n^{\gamma}/2$, then $k \cdot n^{-\gamma} + \log^2(2n)/n < 1$, and there is a set S satisfying the properties we want.

Given such a set S, Alice and Bob can use the circuit to obtain a protocol solving the distinctness problem. Bob sets the coordinates of b in S according to his input, and Alice sets the remaining coordinates according to her input. Each gate g_i depends on at most $\frac{4}{5}\log(2n)$ of Bob's bits. There are $2^{O(n^{4/5})}$ Boolean functions that depend on $\frac{4}{5}\log(2n)$ bits, so Alice can send Bob $k \cdot O(n^{4/5})$ bits to describe the function Bob should evaluate to compute each g_i.

Finally, by Lemma 9.8, we must have that

$$k \geq \Omega(\min\{n^{\gamma}, n^{1/5}\}).$$

□

Boolean Depth Conjecture

CAN WE EFFICIENTLY balance circuits? Can every polynomial sized circuit be simulated by a circuit of logarithmic depth? Recall that we showed how to balance a protocol tree in Chapter 1.

See Exercise 9.3.

This seemingly simple problem remains open, despite much effort to resolve it. Here we discuss an approach[9] for proving a negative answer. The approach is based on direct-sum in communication complexity. Its goal is to prove that there are functions that can be computed using polynomial sized circuits but *cannot* be computed by a circuit of logarithmic depth.

[9] Håstad and Wigderson, 1990; Karchmer et al., 1995; Edmonds et al., 2001; Gavinsky et al., 2014; and Dinur and Meir, 2016.

The idea is to start with a function $f : \{0,1\}^t \to \{0,1\}$ that requires circuits of depth $\Omega(t)$. A random function requires such depth with high probability. The function we are interested in is obtained from f by composition. Given functions $h : \{0,1\}^t \to \{0,1\}$ and $g : \{0,1\}^k \to \{0,1\}$, define their composition $h \circ g : \{0,1\}^{tk} \to \{0,1\}$ by

$$h \circ g(x_1, \ldots, x_t) = h(g(x_1), \ldots, g(x_t)),$$

where each x_i is a k-bit string. Define $f^{\circ t}$ as the t-fold composition of f with itself. The function $f^{\circ t}$ can be computed naively by a circuit of size $O(n^2)$. Indeed, f can be computed using a circuit of size $O(2^t)$. To compute $f^{\circ t}$ we need to evaluate f at most $O(t^t)$ times. We obtain a circuit computing $f^{\circ t}$ with $O(t^t \cdot 2^t) \leq O(n^2)$ gates.

It is natural to conjecture that this naive circuit has essentially smallest possible depth. Namely, that the depth complexity of $f^{\circ t}$ is at least $\Omega(t^2)$. This is much larger than $O(\log n)$.

If there is a function f as above for which for all $k < t$, the circuit-depth of $f \circ f^{\circ k-1}$ must be at least ϵt more than the circuit depth of $f^{\circ k-1}$, then the circuit-depth of $f^{\circ t}$ is at least $\epsilon t^2 \gg t \log t = \log n$. In other words, if there is such an f, then there is a Boolean function depending on n variables that can be computed using $O(n^2)$ gates, but cannot be computed with a circuit of depth $O(\log n)$.

In terms of communication complexity, all that is needed is an example of a function f for which the communication complexity of the

Karchmer-Wigderson game of $f^{\circ k}$ is at least ϵt larger than the communication complexity of the game of $f^{\circ(k-1)}$. This looks quite similar to understanding the direct-sum question in communication that we studied in Chapters 1 and 7. The ideas we discussed there, unfortunately, do not seem to apply in this situation.

Proof Systems

PROOF SYSTEMS PROVIDE A FORMAL FRAMEWORK for proving theorems and for studying the complexity of proofs. A proof system is a specific language for expressing proofs. It consists of a set of rules that allow one to logically derive theorems from axioms. The study of proof systems has led to many interesting results, including Gödel's famous incompleteness theorem.[10]

[10] Gödel, 1931.

Resolution Refutations

THE SIMPLEST proof system is *resolution*.[11] It allows us to refute Boolean formulas expressed in conjunctive normal form (CNF). A proof in resolution shows that a CNF formula cannot possibly be satisfied.

[11] Robinson, 1965.

Some terminology: A literal is a variable or its negation. A clause is an expression of the form $C = \bigvee_j \ell_j$ where each ℓ_j is a literal. We assume that each variable occurs at most once in a clause. A CNF formula is an expression of the form $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where C_i is a clause.

Let us start with an example. Consider the formula

$$F = (x_2 \vee x_1) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_3).$$

The formula F cannot be satisfied by any Boolean assignment. To prove that the formula is unsatisfiable, we repeatedly use the *resolution* rule. The rule derives a clause that must be true if two other clauses are both true:

$$\left.\begin{array}{l} a \vee B \\ \neg a \vee C \end{array}\right\} \Rightarrow B \vee C,$$

where a is a variable, B, C are clauses and $B \vee C$ is the derived clause obtained by including all the literals in B and C. This rule is *sound*. Namely, if both $a \vee B$ and $\neg a \vee C$ are true then at least one of B or C must be true. The resolution refutation shown in Figure 9.4 repeatedly applies this rule to prove that F cannot be satisfied.

A *resolution refutation* is a sequence of clauses. The sequence starts with the clauses of the CNF formula. Each new clause is derived from two previously derived clauses using the resolution rule. The proof ends when the empty clause is derived. The empty clause represents a contradiction.

One can think of a resolution refutation as a directed acyclic graph, like a circuit. The initial clauses correspond to input gates, and the intermediate clauses obtained from the resolution rule correspond to the

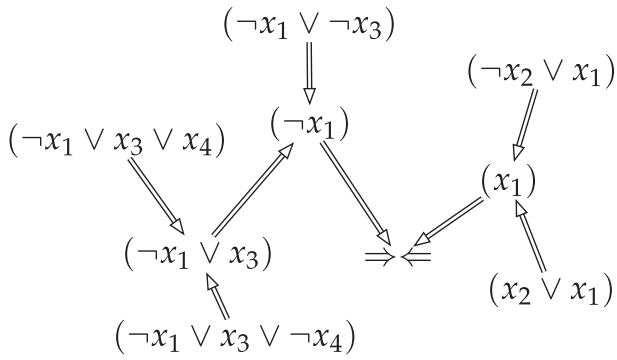

Figure 9.4 A refutation of F. In each step, two clauses are combined to give a new clause that must be true. The final step produces an empty clause, which represents a contradiction.

noninput gates. A refutation is said to be *tree-like* if every derived clause is used only once. Tree-like proofs are the analog of Boolean formulas in proof complexity.

The problem of understanding whether a Boolean formula is satisfiable is a central problem because of its connection[12] to the complexity classes NP and coNP. The best solvers known today try to find satisfying assignments while simultaneously trying to refute formulas obtained after partial assignments. It is important, therefore, to classify the kinds of formulas can be efficiently refuted.

[12] Wikipedia, 2016b.

To study the power of a given proof system, like resolution, we need to a family of formulas of growing complexity. A basic example of such a sequence is the pigeonhole principle.

P = coNP if and only if there is a proof system in which every unsatisfiable Boolean formula can be refuted in a polynomial number of steps.

The Pigeonhole Principle

The pigeonhole principle states that if n pigeons are placed in $n-1$ holes, then some hole must contain at least two pigeons. One can use the principle to construct a sequence of unsatisfiable Boolean formulas. For $i \in [n]$ and $j \in [n-1]$, we have the variable $x_{i,j}$ which indicates that the ith pigeon is in the jth hole. Define the following $n+1$ formulas:

$$P_i = \bigvee_{j \in [n-1]} x_{i,j} \qquad \text{for all } i \in [n]$$

Pigeon i must be in some hole.

$$H = \bigwedge_{\substack{i < i' \in [n] \\ j \in [n-1]}} (\neg x_{i,j} \vee \neg x_{i',j}).$$

Each hole contains at most one pigeon.

The pigeonhole principle states that

$$P = H \wedge \bigwedge_i P_i$$

is not satisfiable.

How hard is it to prove that P is unsatisfiable? If one uses resolution, it is very hard.[13]

[13] Haken, 1985; and Beame and Pitassi, 1996.

The proof actually shows that an exponential number of steps are required in any proof system where each step derives a clause using *any* derivation rule from two clauses!

Theorem 9.10 *Any resolution refutation of the pigeonhole principle must involve* $2^{\Omega(n)}$ *derivation steps.*

We give the proof of this theorem, even though it is not directly related to communication complexity. It will help us get a feel for the basic notions in proof complexity. Later, we discuss connections of proof complexity to communication complexity.

A key idea in the proof is to give the proof system even more power. We allow the proof to assume the following axiom for free.

Axiom 9.11 *Each hole contains exactly one pigeon, and the* $n-1$ *pigeons that are in the holes are distinct.*

H is implied by Axiom 9.11.

In other words, we only consider assignments to the variables that satisfy this axiom. Adding an axiom can only make it easier to derive a contradiction. Axiom 9.11 implies that for each i, j,

$$\neg x_{i,j} \Leftrightarrow \bigvee_{i' \neq i} x_{i',j}.$$

This allows us to replace every negated variable in the proof with a disjunction of unnegated variables.

It is no loss of generality to assume that 4 divides n. If this is not the case, replace n with a nearby multiple of 4. Consider any refutation of P that derives s clauses. Let C be one of the clauses derived in the proof. We say that C is *big* if there is a set $S \subset [n]$ of size $|S| \geq n/4$ such that for each $i \in S$ the number of j's so that C contains $x_{i,j}$ is at least $n/4$.

Let us see how a random assignment affects the refutation. Pick $n/4$ of the pigeons uniformly at random, and randomly assign them to $n/4$ different holes. If pigeon i is assigned to hole j in this process, then we set $x_{i,j} = 1$, we set $x_{i',j} = 0$ for all $i' \neq i$, and $x_{i,j'} = 0$ for all $j' \neq j$. This makes sure that the relevant pigeons and holes are not involved with any of the remaining holes and pigeons.

After this assignment to the variables, $n/4$ of the pigeon clauses become true. Moreover, several variables disappear, and the formula becomes equivalent to the corresponding formula for $3n/4$ pigeons and $3n/4 - 1$ holes. The resolution refutation must still derive a contradiction.

Claim 9.12 *One of the big clauses must survive the assignment.*

Proof Consider the refutation of P after the random assignment. Say that a clause has pigeon complexity w if there is a set $S \subset [n]$ of size w such that

The implication is allowed to use Axiom 9.11.

$$\bigwedge_{i \in S} P_i \Rightarrow C,$$

yet no smaller set S has this property.

The contradiction can only be derived from all $3n/4$ pigeon clauses that remain because one can satisfy any strict subset of those clauses

with some assignment to the variables. So, the empty clause in the proof has pigeon complexity at least $3n/4$. Because the empty clause is derived from two clauses, one of the clauses used to derive the contradiction must have pigeon complexity at least $3n/8$. Continuing in this way, we obtain a sequence of clauses in the proof, where each clause requires at least half as many pigeon clauses as the previous one. Because the clauses of P have pigeon complexity at most 1, there must be a clause C in this sequence that has pigeon complexity at least $n/4$ and at most $n/2 - 1$.

Because n is a multiple of 4.

Let $S \subset [n]$ be the minimal set of pigeon clauses that imply C, and let $i \in S$. Because S is minimal, there must be an assignment x' to all the variables where $\bigwedge_{i' \in S-\{i\}} P_{i'}$ is true, yet C is false. This assignment places all of the pigeons of S into holes, except for the ith pigeon. Suppose j is a hole that did not receive a pigeon during the random assignment, and does not receive a pigeon from S in x'. We claim that C contains the variable $x_{i,j}$. By Axiom 9.11, every hole gets a pigeon in all the assignments under consideration. So, there is a pigeon $i_0 \notin S$ that gets mapped to hole j in this assignment—$x'_{i_0,j} = 1$. Consider what happens when we change the assignment by setting $x'_{i_0,j} = 0$ and $x'_{i,j} = 1$, and leave the rest of the variables as they are. Doing so *must* make C true because $\bigwedge_{i' \in S} P_{i'} = 1$ in the assignment. Because C is a disjunction of unnegated variables, this can only happen if C contains $x_{i,j}$.

We remove the pigeon i_0 from hole j and put i into hole j.

Thus, for each $i \in S$, there must be at least

$$n - 1 - n/4 - (n/2 - 1) = n/4$$

values of j for which $x_{i,j}$ is in the clause C. So, not only is C big, it is big even after the random assignment. □

Claim 9.13 *If a clause C is big, then the probability that C survives the random assignment is at most $\left(\frac{63}{64}\right)^{n/8}$.*

Proof Consider what happens when the first pigeon is assigned to a hole. The probability that the pigeon is one of the $n/4$ pigeons relevant to C is at least $1/4$. The probability that it is assigned to one of the $n/4$ holes that would imply C is at least $1/4$. So the probability that C becomes true after the first pigeon is assigned to a hole is at least $1/16$. Continuing in this way, we see that for each of the first $n/8$ pigeons that we assign to a hole in the random assignment, there are at least $n/4 - n/8 = n/8$ pigeons, which if assigned to $n/4 - n/8 = n/8$ holes would lead to the clause becoming true. Thus, the probability that C survives the first $n/8$ assignments of pigeons to holes is at most

$$\left(1 - \frac{(n/8) \cdot (n/8)}{n^2}\right)^{n/8} = \left(\frac{63}{64}\right)^{n/8}.$$

□

We are ready to prove the theorem:

Proof of Theorem 9.10 Suppose toward a contradiction that the refutation of P has less than $(64/63)^{n/8}$ clauses. By Claim 9.13, there is a partial assignment of the pigeons to holes such that every big clause does not survive. On the other hand, by Claim 9.12, at least one big clause must survive. □

Cutting Planes

A STRONGER PROOF SYSTEM than resolution can be obtained by reasoning about linear inequalities. Clauses are converted into linear inequalities, and the rules allow to combine two linear inequalities to get a new one.

The clause $a \vee \neg b \vee c$ can be viewed as asserting that the Boolean variables a, b, c satisfy the linear inequality

$$-a + b - c \leq 0.$$

The proof system operates on linear inequalities of the form

$$\langle c, x \rangle \leq t$$

where x is an n-bit vector, c is an n-dimensional vector with integer coefficients, and t is a rational number. Because the variables are Boolean, we allow the proof to use the inequalities $x_i \leq 1$ and $-x_i \leq 0$ for free. There are two type of rules in the proof system. For any nonnegative rationals α, α', we can take a linear combination of two inequalities to derive

$$\left.\begin{array}{l} \langle c, x \rangle \leq t \\ \langle c', x \rangle \leq t' \end{array}\right\} \Rightarrow \langle \alpha c + \alpha' c', x \rangle \leq \alpha t + \alpha' t',$$

as long as $\alpha c + \alpha' c'$ is a vector of integers. We also allow the *rounding rule*:

$$\langle c, x \rangle \leq t \Rightarrow \langle c, x \rangle \leq \lfloor t \rfloor.$$

Namely, we can replace t with the largest integer that is at most t. This rule is sound because the left-hand side is always an integer.

The proof system allows us to refute a collection of linear inequalities by deducing the contradiction $1 \leq 0$. Cutting planes can efficiently simulate resolution, line-by-line:

Cutting planes is complete in the sense that every collection of inequalities that has a solution over $\mathbb{R}^n$ but does not have a solution over $\mathbb{Z}^n$ can be refuted in it.

Lemma 9.14 *If a formula can be refuted in s steps using resolution, then it can be refuted in $O(ns)$ steps using cutting planes.*

We do not prove the lemma here, but provide an illustrative example. Consider the resolution derivation

$$\left.\begin{array}{l} \neg x \vee y \vee z \\ x \vee y \vee \neg w \end{array}\right\} \Rightarrow \; y \vee z \vee \neg w.$$

Viewing the clauses as inequalities, this corresponds to

$$\left.\begin{array}{l} x - y - z \leq 0 \\ -x - y + w \leq 0 \end{array}\right\} \Rightarrow -y - z + w \leq 0.$$

This derivation does not directly follow by taking linear combinations. If we add the first two inequalities, we get $-2y - z + w \leq 0$, which is not quite what we want. However, we can derive the inequality we seek using the rounding rule:

$$\left.\begin{aligned} x - y - z &\leq 0 \\ w &\leq 1 \end{aligned}\right\} \Rightarrow x - y - z + w \leq 1,$$

$$\left.\begin{aligned} -x - y + w &\leq 0 \\ -z &\leq 0 \end{aligned}\right\} \Rightarrow -x - y - z + w \leq 0,$$

$$\left.\begin{aligned} \tfrac{1}{2} \cdot (x - y - z + w - 1 \leq 1) \\ \tfrac{1}{2} \cdot (-x - y - z + w \leq 0) \end{aligned}\right\} \Rightarrow -y - z + w \leq \lfloor 1/2 \rfloor = 0.$$

In fact, cutting planes is strictly stronger than resolution. For example, cutting planes allows us to refute the pigeonhole principle using just $O(n^2)$ steps.[14] Rewriting the clauses of the pigeonhole principle as linear inequalities, we get

[14] Cook et al., 1987; and Jukna, 2012.

$$P_i \equiv -\sum_{j=1}^{n-1} x_{i,j} \leq -1,$$

Pigeon i must be in some hole.

$$H_{i,i',j} \equiv x_{i,j} + x_{i',j} \leq 1.$$

Hole j cannot contain both i, i'.

We claim that for each j, we can derive the inequality

$$L_{k,j} \equiv \sum_{i=1}^{k} x_{i,j} \leq 1$$

in $O(k)$ steps. The inequality $L_{2,j}$ is $H_{1,2,j}$. To derive $L_{k,j}$ from previously derived inequalities, use the derivation rules $O(k)$ times to get

$$\begin{aligned} &(k-1) \cdot L_{k-1,j} + \sum_{i=1}^{k-1} H_{i,k,j} \\ &\equiv k(x_{1,j} + x_{2,j} + \cdots + x_{k,j}) \leq 2k - 1. \end{aligned}$$

Now, divide by k and round to get $L_{k,j}$. To complete the proof, observe

$$\sum_{j=1}^{n-1} L_{n,j} \equiv \sum_{j=1}^{n-1} \sum_{i=1}^{n} x_{i,j} \leq n - 1,$$

while

$$\sum_{i=1}^{n} P_i \equiv -\sum_{i=1}^{n} \sum_{j=1}^{n-1} x_{i,j} \leq -n.$$

Adding these last two inequalities gives $1 \leq 0$.

To summarize, cutting planes can efficiently prove the pigeonhole principle, although resolution cannot. Can we find a formula that is difficult to refute using cutting planes? Communication complexity provides such an example.

Lower Bounds on Cutting Planes

Here we give an example of an unsatisfiable formula that requires an exponential number of steps to refute in the cutting planes proof system. The formula is based on properties of graphs.

Given a graph, a *vertex cover* is a set of vertices U such that every edge of the graph contains at least one vertex from U. A *matching* is a set of disjoint edges. We design the formula to encode the fact that the size of every vertex cover must be larger than the size of every matching.

Every edge in the matching must be covered by one vertex from any vertex cover, and the edges are disjoint.

We construct a formula that asserts that the input graph has a vertex cover of size $k-1$, as well as a matching of size k. This ensures that the formula is unsatisfiable. For each possible edge $e = \{v, u\} \subset [n]$, we have the variable x_e which is 1 if and only if the edge e is present in the graph. For $i \in [k]$ and e, the variable $y_{i,e}$ encodes whether e is the ith edges in the matching. For $j \in [k-1]$ and a vertex $v \in [n]$, the variable $z_{j,v}$ encodes whether v is the jth vertex in a cover. Now, define the following formulas:

Every edge is covered.

$$C = \bigwedge_{e \in \binom{[n]}{2}} \left(\neg x_e \vee \bigvee_{v \in e, j \in [k-1]} z_{j,v} \right),$$

The jth vertex in the cover is unique.

$$C_j = \left(\bigvee_{v \in [n]} z_{j,v} \right) \wedge \bigwedge_{v \neq v' \in [n]} (\neg z_{j,v} \vee \neg z_{j,v'}) \quad \text{for all } j \in [k-1],$$

Edges in matching are disjoint.

$$M = \bigwedge_{e,e' \in \binom{[n]}{2} : |e \cap e'| = 1} \bigwedge_{i \neq i' \in [k]} (\neg y_{i,e} \vee \neg y_{i',e'}),$$

The ith edge of the matching is a unique edge in the graph.

$$M_i = \left(\bigvee_{e \in \binom{[n]}{2}} y_{i,e} \right) \wedge \left(\bigwedge_{e \neq e' \in \binom{[n]}{2}} (\neg y_{i,e} \vee \neg y_{i,e'}) \right) \quad \text{for all } i \in [k],$$

Edges in the matching are in the graph.

$$K = \bigwedge_{e \in \binom{[n]}{2}, i \in [k]} (x_e \vee \neg y_{i,e}) .$$

Finally, define the formula:

$$F = C \wedge \left(\bigwedge_{j=1}^{k-1} C_j \right) \wedge \left(\bigwedge_{i=1}^{k} M_i \right) \wedge M \wedge K.$$

The formula F has at most $O(n^4)$ clauses. However, an exponential number of inequalities are needed to refute it, at least with a tree-like proof.[15]

[15] Impagliazzo et al., 1994; Krajíček, 1997; Pudlák, 1997; and Hrubeš, 2013.

Theorem 9.15 *Any tree-like cutting planes refutation of F with $n/4 \leq k \leq n/2$ must derive $2^{\Omega(n/\log n)}$ inequalities.*

The proof is by reduction to the communication complexity of the matching game, for which we already proved a lower bound in Theorem 9.5. In the matching game, Alice gets a graph G that has a matching of size $k \approx n/3$, and Bob gets a graph H that does not have a matching of size k. Their goal is to find an edge in G that is not in H.

Lemma 9.16 *If there is a tree-like cutting plane proof of size s refuting F, then there is a randomized protocol for the matching game with communication $O(\log(s)(\log(n) + \log\log(s)))$.*

By the lemma and Theorem 9.5,

$$\log(s)(\log(n) + \log\log(s)) \geq \Omega(n),$$

which proves Theorem 9.15.

The proof of the lemma shows how to efficiently convert a cutting planes refutation to a communication protocol. The proof extends to many other formulas that have similar structure. For simplicity, we limit the discussion to this particular formula.

Proof of Lemma 9.16 Alice sets the variables $y_{i,e}$ to be consistent with her matching, and Bob sets the variables $z_{j,v}$ and x_e to be consistent with the graph H. Under this setting of variables, all of the clauses in M_i, M, C_j, C are true, but one of the clauses in K must be false. This false clause specifies an edge that is in Alice's graph G but not in Bob's graph H. Our goal is to find this clause using the refutation of F.

By Lemma 1.8, the proof must derive an inequality L that depends on at most $2s/3$ of the clauses, and on at least $s/3$ of the clauses. Our aim is to check whether L is satisfied under the assignment to the variables described above. The inequality L can be written as

$$\kappa + \sum_{i,e} \alpha_{i,e} \cdot y_{i,e} \leq \sum_{j,v} \beta_{j,v} \cdot z_{j,v} + \sum_{e} \gamma_e \cdot x_e.$$

All of the variables on the left-hand side are known to Alice, and all the variables on the right-hand side are known to Bob. Because the variables are Boolean, there are at most 2^{n^3} possible values for the left-hand side, and at most 2^{2n^3} possible values for the right-hand side. Alice and Bob know L, so they also know all these $\leq 2^{1+2n^3}$ possible values. The parties can use the randomized protocol for solving the greater-than problem to compute whether or not this inequality is satisfied by their variables, as in Exercise 3.1. They expend $O(\log(n/\epsilon))$ bits of communication in order to make sure that output of their computation is correct with error ϵ.

If the inequality L is not satisfied, Alice and Bob can safely discard the clauses that are not used to derive L and continue to find a false clause. Otherwise, all of the clauses used to derive L can safely be discarded, and Alice and Bob can start their search again after discarding all the inequalities used to derive L. In either case, they discard at least $s/3$ clauses.

This process can repeat at most $O(\log s)$ times, so the probability that they make an error is at most $O(\epsilon \log s)$ by the union bound. Setting ϵ to be small enough so that this number is at most $1/3$, we obtain a protocol whose length is at most $O(\log(s)(\log n + \log\log s))$. □

Exercises

Ex 9.1 – Prove that every circuit of size s can be simulated by a formula of size 2^s.

Ex 9.2 – We know that every function can be computed by a circuit of depth n, and that most functions require depth $\Omega(n)$. Show that there is a function that can be computed in depth $O(\sqrt{n})$, but cannot be computed in depth $O(n^{1/4})$.

Ex 9.3 – Show that any Boolean formula can be *balanced*. Prove that if a Boolean function can be computed by a formula of size s, then it can be computed using a formula of depth $O(\log s)$. *Hint: This is similar to how we balanced protocol trees in Chapter 1.*

Ex 9.4 – We showed that any formula that computes whether or not $x \in [2n]^n$ corresponds to n distinct numbers requires a formula of size $\Omega(n^2)$. This gives a Boolean function that depends on m bits but requires $\Omega(m^2/\log^2 m)$ size formulas. Show how you can improve the lower bound to get a Boolean function depending on m bits that requires formulas of size $\Omega(m^2/\log m)$. *Hint: Consider the element distinctness function with $x \in [n]^k$ for the appropriate n and k.*

Ex 9.5 – Suppose $\phi(x_1, \ldots, x_n, y_1, \ldots, y_n)$ is a Boolean formula in conjunctive normal form that is unsatisfiable. Suppose Alice and Bob are given $x, y \in \{0,1\}^n$, respectively, and want to find a specific clause in ϕ that is not satisfied.

1. Show that if the formula has a resolution refutation of depth d, then Alice and Bob can find this clause with d bits of communication using a deterministic protocol.
2. Show that if the formula has a cutting-planes refutation of depth d, then Alice and Bob can find this clause with $d \log d$ bits of communication using a randomized protocol.

Ex 9.6 – In this exercise we give another lower bound for the cutting planes proof system. Consider the game where Alice is given a permutation $\pi : [n] \to [n]$, and Bob is given a subset $Y \subseteq [n]$, with the promise that $\pi(1) \in Y, \pi(n) \notin Y$. Their goal is to compute i such that $\pi(i) \in Y, \pi(i+1) \notin Y$. One can show that $\Omega(\log^2 n)$ communication is required.[16]

[16] Karchmer et al., 1995.

1. Give a protocol that solves the game with $O(\log^2 n)$ communication.
2. Consider the monotone function that takes a graph G as input, and outputs whether or not the vertex 1 is connected to 2. Use the lower bound described above to prove that any monotone formula for this function must be of size at least $n^{\Omega(\log n)}$.