Part II

Lower bounds for Concrete Computational Models

Chapter 12

Decision Trees

Currently, resolving many of the basic questions on the power of Turing machines seems out of reach. Thus it makes sense to study simpler, more limited computing devices, as a way to get some insight into the elusive notion of efficient computation. Moreover, such limited computational models often arise naturally in a variety of applications, even outside Computer Science, and hence studying their properties is inherently worthwhile.

Perhaps the simplest such model is that of *decision trees*. Here the "complexity" measure for a Boolean function f is the number of bits we need to examine in an input x in order to compute $f(x)$. This chapter surveys the basic results and open questions regarding decision trees. Section 12.1 defines decision trees and decision tree complexity. We also define nondeterministic and probabilistic versions of decision trees just as we did for Turing machines; these are described in Sections 12.2 and 12.3 respectively. Section 12.4 contains some techniques for proving *lower bounds* on decision trees. We also present Yao's *Min Max Lemma* (see Note 12.8), which is useful for proving lower bounds for randomized decision tree complexity and more generally, lower bounds for randomized complexity in other computational models.

12.1 Decision trees and decision tree complexity

Let $f : \{0,1\}^n \to \{0,1\}$ be some function. A *decision tree* for f is a tree for which each internal node is labeled with some x_i, and has two outgoing edges, labeled 0 and 1. Each tree leaf is labeled with an output value 0 or 1. The computation on input $x = x_1 x_2 \ldots x_n$ proceeds at each node by inspecting the input bit x_i indicated by the node's label. If $x_i = 1$ the computation continues in the subtree reached by taking the 1-edge. The 0-edge is taken if the bit is 0. Thus input x follows a path through the tree. The output value at the leaf is $f(x)$. For example, Figure 12.1 describes a decision tree for the majority function.

Decision trees often arise in medical setting, as a compact way to describe how to reach a diagnosis from data of symptoms and test results. They are also used in operations research (to describe algorithms to make business decisions) and machine learning (where the goal is to discover such trees by looking at many examples). However, our focus is to use decision trees as a simple computational model for which we are able to prove some non-trivial lower bounds.

The decision tree complexity of a function is the number of bits examined by the most efficient decision tree on the worst case input to that tree. That is, we make the following definition:

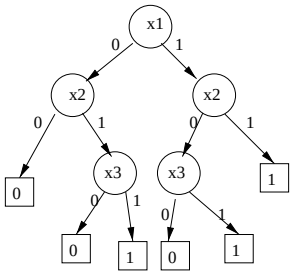


Figure 12.1 A decision tree for computing the majority function $Maj(x_1, x_2, x_3)$ on three bits. That is, the output is 1 if at least two of the input bits are 1, and otherwise the output is 0.

Definition 12.1 *(Decision tree complexity)*
The cost of tree t on input x, denoted by $cost(t, x)$, is the number of bits of x examined by t.

The *decision tree complexity* of a function f, is defined as

$$D(f) = \min_{t \in \mathcal{T}_f} \max_{x \in \{0,1\}^n} cost(t, x)\,,$$

where $\mathcal{T}_f$ denotes the set of all decision trees that compute f.

Since every Boolean function f on $\{0,1\}$ can be computed by the full binary tree of depth n (and 2^n vertices), $D(f) \leq n$ for every $f : \{0,1\}^n \to \{0,1\}$. We'll be interested in finding out for various interesting functions whether this trivial bound is the best possible or they have more efficient decision trees.

Example 12.2
Here are some examples for the decision tree complexity of some particular functions:

OR Function: Let $f(x_1, x_2, \ldots x_n) = \bigvee_{i=1}^{n} x_i$. In this case we can show that there is no decision tree with smaller depth than the trivial bound of n. To do that we use an *adversary argument*. Let t be some decision tree solving f. We think of an execution of t, where some adversary answers t's questions on the value of every input bit. The adversary will always respond that x_i equals 0 for the first $n-1$ queries. Thus the decision tree will be "in suspense" until the value of the nth bit is revealed, whose value will determine whether or not the OR of the bits is 1 or 0. Thus $D(f)$ is n. (To state this argument another way, we have shown that if there is some branch on which the tree asks at most $n-1$ questions, then there is are two inputs x, x' consistent with all the adversary's answers such that $f(x) = 0$ but $f(x') = 1$, implying that the tree will make a mistake on one of those inputs.)

Graph connectivity Suppose that we are given an m-vertex graph G as input, represented as a binary string of length $n = \binom{m}{2}$ binary string, with the e^{th} coordinate equal to 1 if the edge e is in G, and equal to 0 otherwise. We would like to know how many bits of the adjacency matrix a decision tree algorithm might have to inspect in order to determine whether or not G is connected (i.e., every two points are connected by a path of edges). Once again, we show that it's not possible to beat the trivial $\binom{m}{2}$ bound.

We again give an adversary argument. The adversary constructs a graph, edge by edge, as it responds to the algorithm's queries. At each step, the

answers to the preceding queries defines a *partial graph* such that it can be extended to both a *connected* and *disconnected* graph using the edges that have not been queried so far. Thus the algorithm (i.e., decision tree) is in "suspense" until every single possible edge has been queried.

For every query of an edge e made by the tree, our adversary will answer 0 (i.e., that the edge is not present), unless such an answer forces the current partial graph to become disconnected, in which case it answers 1. An easy induction shows that this strategy ensures that the current partial graph is a "forest" (i.e., consists of trees that are vertex-disjoint), and furthermore this forest does not turn into a connected graph until the very last edge gets queried. Thus the algorithm remains "in suspense" as long as it didn't query all possible $\binom{m}{2}$ edges.

AND-OR Function For every k, we define f_k to be the following function taking inputs of length $n = 2^k$:

$$f_k(x_1,\ldots,x_n) = \begin{cases} f_{k-1}(x_1,\ldots x_{2^{k-1}}) \wedge f_{k-1}(x_{2^{k-1}},\ldots x_{2^k+1}) & \text{if } k \text{ is even} \\ f_{k-1}(x_1,\ldots x_{2^{k-1}}) \vee f_{k-1}(x_{2^{k-1}},\ldots x_{2^k+1}) & \text{if } k>1 \text{ and is odd} \\ x_i & \text{if } k=1 \end{cases}$$

The AND-OR function f_k can be computed by a circuit of depth k (see Figure 12.2). By contrast, its decision tree complexity is 2^k (see Exercise 12.2).

Address function Suppose that $n = k + 2^k$ and let f be the function that maps $x_1,\ldots,x_k,y_1,\ldots,y_{2^k}$ to the input y_x. That is, the function treats the first $k \sim \log n$ bits as an index to an array specified by the last $n-k$ bits. Clearly, this function has a decision tree of depth $k+1$ (examine the first k bits, and use that to find out which bit to examine among the last $n-k$ bits), and hence $D(f) \leq \log n + 1$. On the other hand, Exercise 12.1 implies that $D(f) \geq \Omega(\log n)$.

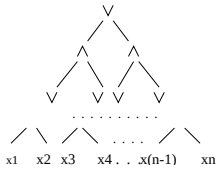


Figure 12.2 A circuit computing the AND-OR function. The circuit has k layers of alternating gates, where $n = 2^k$..

12.2 Certificate Complexity

We now introduce the notion of *certificate complexity*. This can be viewed as the nondeterministic version of decision tree complexity, analogous to the relation between nondeterministic and deterministic Turing machine (see Chapter 2).

Definition 12.3 *(Certificate complexity)*
Let $f : \{0,1\}^n \to \{0,1\}$ and $x \in \{0,1\}^n$. A 0-*certificate* for x is a subset $S \subseteq \{0,1\}^n$, such that $f(x') = 0$ for every x' such that $x'|_S = x|_S$ (where $x|_S$ denotes the substring of x in the coordinates in S). Similarly, if $f(x) = 1$ then a 1-*certificate* for x is a subset $S \subseteq \{0,1\}^n$ such that $f(x') = 1$ for every x' satisfying $x|_S = x'|_S$.

The *certificate complexity* of f is defined as the minimum k such that every string x has a $f(x)$-certificate of size at most k. (Note that a string cannot have both a 0-certificate and a 1-certificate.)

If f has a decision tree t of depth k, then $C(f) \leq k$, since the set of locations examined by t on input x serves as an $f(x)$-certificate for x. Thus, $C(f) \leq D(f)$. But sometimes $C(f)$ can be strictly smaller than $D(f)$:

Example 12.4
We show the certificate complexity of some of the functions described in Example 12.2

Graph connectivity Let f be the graph connectivity function. Recall that for an m-vertex graph, the decision tree complexity of f is $D(f) = \binom{m}{2} = \frac{m(m-1)}{2}$. A 1-certificate for a graph G is a set of edges whose existence in G implies that it is connected. Thus every connected m-vertex graph G has a 1-certificate of size $m-1$— any spanning tree for G will do. A 0-certificate for G is a set of edges whose non-existence forces G to be disconnected— a cut. Since the number of edges in a cut is maximized when its two sides are equal, every m-vertex graph has a 0-certificate of size at most $(m/2)^2 = m^2/4$. On the other hand, for some graphs (e.g., the graph consisting of two disjoint cliques of size $n/2$) do not have smaller 0-certificate. Thus $C(f) = m^2/4$.

AND-OR function Let f_k be the AND-OR function on $n = 2^k$-length inputs. Recall that $D(f) = 2^k$. We show that $C(f) \leq 2^{\lceil k/2 \rceil}$, which roughly equals $\sqrt{n}$.Recall that f_k is defined using a circuit of k layers. Each layer contains only OR-gates or only AND-gates, and the layers have alternative gate types. The bottom layer receives the bits of input x as input and the single top layer gate outputs the answer $f_k(x)$. If $f(x) = 1$, we can construct a 1-certificate as follows. For every AND-gate in the tree of gates we have to prove that both its children evaluate to 1, whereas for every OR-gate we only need to prove that *some* child evaluates to 1. Thus the 1-certificate is a subtree in which the AND-gates have two children but the OR gates only have one each. This mean that the subtree only needs to involve $2^{\lceil k/2 \rceil}$ input bits. If $f(x) = 0$, a similar argument applies, but the role of OR-gates and AND-gates, and values 1 and 0 are reversed.

Recall in Chapter 2 (Definition 2.1) we defined **NP** to be the class of functions f for which an efficient Turing machine can be convinced that $f(x) = 1$ via a short certificate. Similarly, we can think of 1-certificates as a way to convince a decision tree that $f(x) = 1$, and hence we have the following analogies

$$\begin{aligned} \text{low decision tree complexity} &\leftrightarrow \quad \mathbf{P} \\ \text{low 1-certificate complexity} &\leftrightarrow \quad \mathbf{NP} \\ \text{low 0-certificate complexity} &\leftrightarrow \quad \mathbf{coNP} \end{aligned}$$

Thus, the following result should be quite surprising, since it shows that, unlike what is believed to hold for Turing machines, in the decision tree world "$\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$".

Theorem 12.5 *For function f, $D(f) \leq C(f)^2$.* ♢

PROOF: Let $f : \{0,1\}^n \to \{0,1\}$ be some function satisfying $C(f) = k$. For every $x \in \{0,1\}^n$, denote by S_x the k-sized subset of $[n]$ that is the $f(x)$-certificate for x. The rest of the proof relies on the observation that every 1-certificate must intersect every 0-certificate, since otherwise there exists a single string that contains both certificates, which is impossible.

The following decision tree algorithm determines the value of f in at most k^2 queries. It maintains a set $\mathcal{X}$ consisting of all inputs that are consistent with the replies to queries so far.

> Initially $\mathcal{X} = \{0,1\}^n$. If there is some $b \in \{0,1\}$ such that $f(x) = b$ for every $x \in \mathcal{X}$ then halt and output b. Otherwise, choose any $x_0 \in \mathcal{X}$ such that $f(x_0) = 0$ and query all the bits in S_{x_0} that have not been queried so far. Remove from $\mathcal{X}$ every string $x' \in \{0,1\}$ that is not consistent with the answers to the queries.

Because every input x has some certificate proving the correct answer $f(x)$, this algorithm will eventually output the correct answer for every input. Furthermore, each time it queries the bits in a 0-certificate, all 1-certificates effectively shrink by at least one since, as noted, each 1-certificate must intersect each zero certificate. (Of course, the 1-certificate could be completely eliminated if the answer to some query is inconsistent with it. The same elimination could also happen to a 0-certificate.) Thus in k iterations of the above step, all 1-certificates must shrink to 0, which means that all remaining strings in $\mathcal{X}$ only have 0-certificates and hence the algorithm can answer 0. Since each iteration queries at most k bits, we conclude that the algorithm finishes after making at most k^2 queries.

■

12.3 Randomized Decision Trees

As in the case of Turing machines, we can define a *randomized* analog of decision trees. In a randomized decision tree, the choice of which input location to query is determined probabilistically. An equivalent, somewhat more convenient description is that a randomized decision tree is a probability distribution over deterministic decision trees. We will consider randomized trees that always output the right answer, but use randomization to possibly speed up their expected cost (this is analogous to the class **ZPP** of Section 7.3).

Definition 12.6 *(Randomized decision trees)*
For every function f, let $\mathcal{P}_f$ denote the set of probability distributions over decision trees that compute f. The *randomized tree complexity* of f is defined as

$$R(f) = \min_{P \in \mathcal{P}_f} \max_{x \in \{0,1\}^n} \mathsf{E}_{t \in_R P}[cost(t,x)]\,. \tag{1}$$

The randomized decision tree complexity thus expresses how well the best possible probability distribution of trees will do against the worst possible input. Obviously, $R(f) \leq D(f)$— a deterministic tree is just a special case of such a probability distribution. It's also not hard to verify that $R(f) \geq C(f)$, because for every input $x \in \{0,1\}^n$, every tree t deciding f yields an $f(x)$-certificate of x of size $cost(t,x)$. Thus the expectation in (1) is bounded below by the size of the smallest certificate for x. (This is analogous to the fact that $\mathbf{ZPP} \subseteq \mathbf{NP} \cap \mathbf{coNP}$.)

Example 12.7
Consider the majority function, $f = Maj(x_1, x_2, x_3)$. It is straightforward to see that $D(f) = 3$. We show that $R(f) \leq 8/3$. We'll use a randomized decision

tree that picks a random permutation of x_1, x_2, x_3 and makes its queries by this order, stopping when it got two identical answers. If all of x's bits are the same, then this tree will always stop after two queries. If two bits are the same and the third is different, say $x_1 = 1$ and $x_2 = x_3 = 0$, then the algorithm will make two queries if it orders x_1 last, which happens with probability 1/3. Otherwise it makes three queries. Thus the expected cost is $2 \cdot 1/3 + 3 2/3 = 8/3$. Later in Example 12.9, we'll see that in fact $R(f) = 8/3$.

12.4 Some techniques for decision tree lower bounds

We've seen the adversary method for showing lower bounds on deterministic decision tree complexity, but it does not always seem useful, especially when considering certificate complexity and randomized decision tree complexity. We now discuss some more sophisticated techniques for showing such lower bounds. These techniques have also found other uses in complexity theory and beyond.

12.5 Lower bounds on Randomized Complexity

Randomized decision trees are complicated creatures— distributions over decision trees— and hence are harder to argue about than deterministic decision trees. Fortunately, Yao had shown that we can prove lower bounds on randomized trees by reasoning about deterministic trees. Specifically, Yao's Min-Max Lemma (see Note 12.8) shows that for every function f we can lower bound $R(f)$ by k if we can find a distribution $\mathcal{D}$ over the inputs $\{0,1\}^n$ for which we prove that $\mathsf{E}_{x \in_R \mathcal{D}}[cost(t,x)] \geq k$ (in words, the *average* cost of t on an input drawn according to distribution $\mathcal{D}$ is at least k) for every deterministic decision tree for f. In other words, rather than arguing about distributions on trees and specific inputs, we can also argue about distributions on inputs and specific trees.

Example 12.9
We return to considering the function f that is the majority of three bits, and we seek to find a lower bound on $R(f)$. Consider a distribution over inputs such that inputs in which all three bits match, namely 000 and 111, occur with probability 0. All other inputs occur with probability 1/6. For any decision tree, that is, for any order in which the three bits are examined, there is exactly a 1/3 probability that the first two bits examined will be the same value, and thus there is a 1/3 probability that the cost is 2. There is then a 2/3 probability that the cost is 3. Thus for every decision tree for majority, the overall expected cost for this distribution is 8/3. This implies by Yao's Lemma that $R(f) \geq 8/3$. By Example 12.7, $R(f) = 8/3$.

12.5.1 Sensitivity

The *sensitivity* of a function is another method that allows us to prove lower bounds on decision tree complexity.

Definition 12.10 *(Sensitivity and Block Sensitivity)* If $f : \{0,1\}^n \to \{0,1\}$ is a function and $x \in \{0,1\}^n$ then the *sensitivity of f on x*, denoted $s_x(f)$, is the number of bit positions i

Note 12.8 *(Yao's Min-Max Lemma)*

Yao's min-max lemma is used in a variety of settings to prove lower bounds on randomized algorithms. Let $\mathcal{X}$ be a finite set of inputs and $\mathcal{A}$ be a finite set of deterministic algorithms that solve some computational problem f on these inputs. For $x \in \mathcal{X}$ and $A \in \mathcal{A}$, we denote by $cost(A, x)$ the *cost* incurred by algorithm A on input x (the cost could be running time, decision tree complexity, etc..). A *randomized algorithm* can be viewed as a probability distribution $\mathcal{R}$ on $\mathcal{A}$. The *cost* of $\mathcal{R}$ on input x, denoted by $cost(\mathcal{R}, x)$, is $\mathsf{E}_{A \in_R \mathcal{R}}[cost(A, x)]$. The *randomized complexity* of the problem is

$$\min_{\mathcal{R}} \max_{x \in \mathcal{X}} cost(\mathcal{R}, x)\,. \tag{2}$$

Let $\mathcal{D}$ be a distribution on inputs. For any deterministic algorithm A, the cost incurred by it on $\mathcal{D}$, denoted $cost(A, \mathcal{D})$, is $\mathsf{E}_{x \in_R \mathcal{D}}[cost(A, x)]$. The *distributional complexity* of the problem is

$$\max_{\mathcal{D}} \min_{A \in \mathcal{A}} cost(A, \mathcal{D})\,. \tag{3}$$

Yao's Lemma says that the two quantities (2) and (3) are equal. It is easily derived from von Neumann's minmax theorem for zero-sum games (see Note 19.4). The switch of quantifiers featured in Yao's lemma is typically useful for lower bounding randomized complexity. To do so, one defines (using some insight and some luck) a suitable distribution $\mathcal{D}$ on the inputs for some function f. Then one proves that *every* deterministic algorithm for f incurs high cost, say C, on this distribution. By Yao's Lemma, it follows that the randomized complexity then is at least C.

such that $f(x) \neq f(x^i)$, where x^i is x with its ith bit flipped. The *sensitivity* of f, denoted by $s(f)$, is $\max_x \{s_x(f)\}$.

The *block sensitivity of f on x*, denoted $bs_x(f)$, is the maximum number b such that there are disjoint blocks of bit positions $B_1, B_2, \ldots, B_b$ such that $f(x) \neq f(x^{B_i})$ where x^{B_i} is x with all its bits flipped in block B_i. The *block sensitivity* of f denoted $bs(f)$ is $\max_x \{bs_x(f)\}$. ♢

Clearly $s(f) \leq bs(f)$. It is conjectured that there is a constant c such that $bs(f) = O(s(f)^c)$ for all f but this is wide open (though it is known that it must holds that $c \geq 2$). It's not hard to show that both the sensitivity and block sensitivity of f lower bound its deterministic decision tree complexity:

Lemma 12.11 *For any function, $s(f) \leq bs(f) \leq D(f)$.* ♢

PROOF: Let x be such that $bs_x(f) = bs(f) = s$ for some s and $B_1, \ldots, B_s$ be the corresponding blocks. For every decision tree t for f, when given b as input t has to query at least one coordinate in each of the blocks B_i for $i \in [s]$ in order to distinguish between x and x^{B_i}. ■

On the other hand, the sensitivity squared also upper bounds the certificate complexity of f:

Theorem 12.12 $C(f) \leq s(f)bs(f)$. ♢

PROOF: For any input $x \in \{0,1\}^n$ we describe a certificate for x of size $s(f)bs(f)$. This certificate for an input x is obtained by considering the largest number of disjoint blocks of variables $B_1, B_2, \ldots, B_b$ that achieve $b = bs_x(f) \leq bs(f)$. We'll take each B_i to be of minimal size— if $f(x) \neq f(x^{B'_i})$ for some strict subset B'_i of B_i then we'll use B'_i instead. This means that setting $x' = x^{B_i}$, $f(x') \neq f(x'^j)$ for every $j \in B_i$, implying that $|B_i| \leq s(f)$ for every $i \in [b]$. Thus the certificate size is at most $s(f)bs(f)$.

We claim that setting these variables according to x constitutes a certificate for x. Suppose not, and let x' be an input such that $f(x') \neq f(x)$ but x' is consistent with the above certificate. Let B_{b+1} denote the variables that need to be flipped to make x equal x'. That is,

$D(f)$	Deterministic decision-tree complexity (corresponds to **P**)
$R(f)$	Randomized decision-tree complexity (corresponds to **ZPP**)
$C(f)$	Certificate complexity (corresponds to **NP** $\cap$ **coNP**)
$s(f)$	Sensitivity of f (maximum number of bits that flip $f(x)$)
$bs(f)$	Block sensitivity of f (maximum number of blocks that flip $f(x)$)
$deg(f)$	Degree of multilinear polynomial for f

$$C(f) \leq R(f) \leq D(f) \leq C(f)^2$$
$$s(f) \leq bs(f) \leq D(f) \leq bs(f)^3$$
$$C(f) \leq s(f)bs(f)$$
$$bs(f) \leq 2deg(f)$$
$$D(f) \leq deg(f)^2 bs(f) \leq 2deg(f)^4$$

Table 12.1 Summary of notions introduced in this chapter and relations between them. We only proved that $D(f) \leq bs(f)^4$, but the stronger relation $D(f) \leq bs(f)^3$ is known [BBC+98].

$x' = x^{B_{b+1}}$. Then B_{b+1} must be disjoint from $B_1, B_2, \ldots B_b$, which contradicts $b = bs_x(f)$. ■

12.5.2 The degree method

Recent work on decision tree lower bounds has used *polynomial representations* of Boolean functions. Recall that a multilinear polynomial is a polynomial whose degree in each variable is 1.

Definition 12.13 An n-variate polynomial $p(x_1, x_2, \ldots, x_n)$ over the reals *represents* $f : \{0,1\}^n \to \{0,1\}$ if $p(x) = f(x)$ for all $x \in \{0,1\}^n$.

The *degree* of f, denoted $deg(f)$, is the degree of the multilinear polynomial that represents f. (Exercise 12.7 asks you to show that the multilinear polynomial representation is unique, so $deg(f)$ is well-defined.)

Example 12.14
The AND of n variables $x_1, x_2, \ldots, x_n$ is represented by the multilinear polynomial $\prod_{i=1}^n x_i$ and OR is represented by $1 - \prod_{i=1}^n (1 - x_i)$.

The degree of AND and OR is n, and so is their decision tree complexity. In fact, $deg(f) \leq D(f)$ for very function f (see Exercise 12.7). A rough inequality in the other direction is also known, though we omit the proof.

Theorem 12.15 *1.* $bs(f) \leq 2 \cdot deg(f)$

2. $D(f) \leq deg(f)^2 bs(f)$ ♢

Table 12.1 contains summary of the various complexity measures introduced in this chapter.

WHAT HAVE WE LEARNED?

- The decision tree complexity of a function f is the number of input bits that need to be examined to determine f's value. There are randomized and non-deterministic variants of this notion.
- Unlike what we believe to hold in the Turing machine model, all notions of decision tree complexity (deterministic, randomized, and non-deterministic) are polynomially related to one another.
- Yao's Min-Max Lemma reduces the task of proving worst-case bounds on probabilistic algorithms to proving average-case bounds for deterministic algorithms.
- Other techniques to prove lower bounds on decision trees include the adversary method, sensitivity and block sensitivity, and the degree method.

Chapter notes and history

Decision trees have been used to encode decisions in medicine and operations research since the early days of computing. Pollack [Pol65] describes algorithm to transform decision trees into computer program, minimizing either the total program size or the the running time (i.e., decision tree complexity). Garey[Gar72] formally defined decision trees and gave some algorithms to evaluate them, while Hyafil and Rivest proved an early **NP**-completeness result for the task of finding an optimal decision tree for a classification problem [HR76].

Buhrman and de Wolf [BdW02] give a good survey of decision tree complexity. The result that the decision tree complexity of connectivity and many other problems is $\binom{n}{2}$ has motivated the following conjecture (attributed variously to Anderaa, Karp, and Rosenberg): *Every non-constant monotone graph property f on m-vertex graphs satisfies $D(f) = \binom{m}{2}$*. Here "monotone" means that adding edges to the graph cannot make it go from having the property to not having the property (e.g., connectivity). "Graph property" means that the property does not depend upon the vertex indices (e.g., conditions such as connectivity, having a k-clique etc.. are graph properties while the condition that vertex 1 and vertex 2 have an edge between them is not). This conjecture was shown to be true by Rivest and Vuillemin [RV76] if m is a prime power, but in general it's only known to hold up to a constant factor [KSS83]; the proof uses topology and is excellently described in Du and Ko [DK00]. Another conjecture is that even the randomized decision tree complexity of monotone graph properties is $\Omega(n^2)$ but here the best lower bound is close to $n^{4/3}$ [Yao87, Kin88, Haj90]. See [LY02] for a survey on these conjectures and the progress so far.

The notion of sensitivity was defined by Cook, Dwork and Reischuk [CDR86], while the notion of Block sensitivity was defined by Nisan [Nis89], who also proved Theorem 12.12. In both cases the motivation was to prove lower bounds for parallel random access machines.

Part 1 of Theorem 12.15 is due to Nisan and Szegedi [NS92]. Part 2 is due to Nisan and Smolensky (unpublished), see [BdW02] for a proof.

The polynomial method for decision tree lower bounds is surveyed in [BdW02]. The method can be used to lower bound randomized decision tree complexity (and more recently, *quantum* decision tree complexity) but then one needs to consider polynomials that *approximately* represent the function.

Exercises

12.1 Suppose f is any function that depends on all its bits; in other words, for each bit position i there is an input x such that $f(x) \neq f(x^i)$ (where x^i denotes the string obtained by flipping x's i^{th} bit). Show that $s(f) = \Omega(\log n)$. H461

12.2 For every $k \in \mathbb{N}$, let f_k be the AND-OR function on strings of length 2^k (see Example 12.2). Prove that $D(f_k) = 2^k$. H461

12.3 Let f be a function on $n = k^2$ variables that is the AND of k OR's, each of disjoint k variables. Prove that $s(f) = bs(f) = C(f) = \sqrt{n}, deg(f) = D(f) = n, R(f) \geq \Omega(n)$

12.4 Let f be a function on $n = k^2$ variables that is the OR of k applications of $g : \{0,1\}^k \to \{0,1\}$, each on a disjoint block of k variables, where $g(x_1, \ldots, x_k) = 1$ if there exists $i \in [k-1]$ such that $x_i = x_{i=1} = 1$ and $x_j = 0$ for all $j \neq i$. Prove that $s(f) = \sqrt{n}$ and $bs(f) = n/2$.

12.5 Show that for every $f : \{0,1\}^n \to \{0,1\}$, there exists a unique multilinear polynomial that represents f.

12.6 Find the multilinear representation of the PARITY of n variables.

12.7 Show that $deg(f) \leq D(f)$.

Chapter 13

Communication Complexity

Communication complexity concerns the following scenario. There are two players with unlimited computational power, each of whom holds an n bit input, say x and y. Neither knows the other's input, and they wish to collaboratively compute $f(x, y)$ where the function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is known to both. Furthermore, they had foreseen this situation (e.g., one of the parties could be a spacecraft and the other could be the base station on earth), so they had already —before they knew their inputs x, y— agreed upon a protocol for communication.[1] The *cost* of this protocol is the *number of bits communicated* by the players for the *worst-case* choice of inputs x, y.

Researchers have studied many modifications of the above basic scenario, including randomized protocols, nondeterministic protocols, and average-case protocols. Furthermore, lower bounds on communication complexity have uses in a variety of areas, including lower bounds for parallel and VLSI computation, circuit lower bounds, polyhedral theory, data structure lower bounds, and more. Communication complexity has been one of the most successful models studied in complexity, as it strikes the elusive balance of being simple enough so that we can actually prove strong lower bounds, but general enough so we can obtain important applications of these lower bounds.

In this chapter we only give a very rudimentary introduction to this area. In Section 13.1 we provide the basic definition of two-party deterministic communication complexity. In Section 13.2 we survey some of the techniques used to prove *lower bounds* for the communication complexity of various functions, using the equality function (i.e., $f(x, y) = 1$ iff $x = y$) as a running example. In Section 13.3 we define *multiparty* communication complexity and show a lower bound for the generalized inner product function. Section 13.4 contains a brief survey of other models studied, including probabilistic and non-deterministic communication complexity. The chapter notes mention some of the many applications of communication complexity.

13.1 Definition of two-party communication complexity.

Now we formalize the informal description of communication complexity given above:

[1] Do not confuse this situation with *information theory*, where an algorithm is given messages that have to be transmitted over a noisy channel, and the goal is to transmit them robustly while minimizing the amount of communication. In communication complexity the channel is not noisy and the players determine what messages to send.

Definition 13.1 *(Two party communication complexity)*
Let $f : \{0,1\}^{2n} \to \{0,1\}$ be a function. A *t-round two party protocol* Π *for computing* f is a sequence of t functions $P_1, \ldots, P_t : \{0,1\}^* \to \{0,1\}^*$. An execution of Π on inputs x, y involves the following: Player 1 computes $p_1 = P_1(x)$ and sends p_1 to Player 2, Player 2 computes $p_2 = P_2(y, p_1)$ and sends p_2 to Player 1, and so on. Generally, at the i^{th} round, if i is odd then Player 1 computes $p_i = P_i(x, p_1, \ldots, p_{i-1})$ and sends p_i to Player 2, and similarly if i is even then Player 2 computes $p_i = P_i(y, p_1, \ldots, p_{i-1})$ and sends p_i to Player 1.

The protocol Π is valid if for every pair of inputs x, y, the last message sent (i.e., the message p_t) is equal to the value $f(x, y)$. The *communication complexity* of Π is the maximum number of bits communicated (i.e., maximum of $|p_1| + \ldots + |p_t|$) over all inputs $x, y \in \{0,1\}^n$. The *communication complexity* of f, denoted by $C(f)$ is the minimum communication complexity over all valid protocols Π for f.

For every function, $C(f) \leq n + 1$ since the trivial protocol is for first player to communicate his entire input, whereupon the second player computes $f(x, y)$ and communicates that single bit to the first. Can they manage with less communication?

Example 13.2 *(Parity)*
Suppose the function $f(x, y)$ is the *parity* of all the bits in x, y. Then $C(f) = 2$. Clearly, $C(f) \geq 2$ since the function depends nontrivially on each input, so each player must transmit at least one bit. The fact that $C(f) \leq 2$ is demonstrated by the following protocol: Player 1 sends the parity a of the bits in x and Player 2 sends a XOR'd with the parity of the bits in y.

Example 13.3 *(Halting Problem)*
Consider the function $\mathrm{H}:\{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined as follows. If $x = 1^n$ and $y = \mathrm{code}(M)$ for some Turing Machine M such that M halts on x then $H(x, y) = 1$ otherwise $H(x, y) = 0$. The communication complexity of this is at most 2; first player sends a bit indicating whether or not his input is 1^n. The second player then determines the answer and sends it to the first player. This example emphasizes that the players have unbounded computational power, including ability to solve the Halting Problem.

Sometimes students ask whether a player can communicate by not saying anything? (After all, they have three options in each round: send a 0, or 1, or not send anything.) We can regard such protocols as having one additional bit of communication, and analyze them analogously.

13.2 Lower bound methods

Now we discuss methods for proving lower bounds on communication complexity. As a running example in this chapter, we will use the equality function:

$$\mathrm{EQ}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

It turns out that almost no improvement is possible over the trivial $n + 1$ bit communication protocol for this function:

Theorem 13.4 *(Equality has linear communication complexity)*
$C(EQ) \geq n$

We will prove Theorem 13.4 by several methods below.

13.2.1 The fooling set method

The first proof of Theorem 13.4 uses an idea called *fooling sets*. For any communication protocol for any function, suppose x, x' are any two different n-bit strings such that the communication pattern (i.e., sequence of bits transmitted) is the same on the input pairs (x, x) and (x', x'). Then we claim that the players' final answer must be the same on all four input-pairs $(x, x), (x, x'), (x', x), (x', x')$. This is shown by an easy induction. If player 1 communicates a bit in the first round, then by hypothesis this bit is the same whether his input is x or x'. If player 2 communicates in the 2nd round, then his bit must also be the same on both inputs x and x' since he receives the same bit from player 1. And so on. We conclude that at the end, the players' answer on (x, x) must agree with their answer on (x, x').

To show $C(EQ) \geq n$ it suffices to note that if a protocol exists whose complexity is at most $n-1$, then there are only 2^{n-1} possible communication patterns. But there are 2^n choices for input pairs of the form (x, x) and so by the pigeonhole principle, there exist two distinct pairs (x, x) and (x', x') on which the communication pattern is the same. But then the protocol must be incorrect, since $\mathrm{EQ}(x, x') = 0 \neq \mathrm{EQ}(x, x)$. This completes the proof. This argument can be easily generalized as follows (Exercise 13.1):

Lemma 13.5 *Say that a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ has a size M fooling set if there is an M-sized subset $S \subseteq \{0,1\}^n \times \{0,1\}^n$ and a value $b \in \{0,1\}$ such that **(1)** for every $\langle x, y\rangle \in S$, $f(x, y) = b$ and **(2)** for every distinct $\langle x, y\rangle, \langle x', y'\rangle \in S$, either $f(x, y') \neq b$ or $f(x', y) \neq b$.*

If f has a size-M fooling set then $C(f) \geq \log M$. ♢

Example 13.6 *(Disjointness)*
Let x, y be interpreted as characteristic vectors of subsets of $\{1, 2, \ldots, n\}$. Let $\mathrm{DISJ}(x, y) = 1$ if these two subsets are disjoint, otherwise $\mathrm{DISJ}(x, y) = 0$. As a corollary of Lemma 13.5 we obtain that $C(\mathrm{DISJ}) \geq n$ since the following 2^n pairs constitute a fooling set:

$$S = \left\{(A, \overline{A}) : A \subseteq \{1, 2, \ldots, n\}\right\}.$$

13.2.2 The tiling method

The tiling method for lower bounds takes a more global view of the function f. Consider the matrix of f, denoted $M(f)$, which is a $2^n \times 2^n$ matrix whose (x, y)'th entry is the value $f(x, y)$ (see Figure 13.1.) We visualize the communication protocol in terms of this matrix. A *combinatorial rectangle* (or just rectangle for short) in the matrix M is a submatrix of M that corresponds to entries in $A \times B$ where $A \subseteq \{0,1\}^n$, $B \subseteq \{0,1\}^n$, we say that $A \times B$ is *monochromatic* if for all x in A and y in B, $M_{x,y}$ is the same. If the protocol begins with the first player sending a bit, then $M(f)$ partitions into two rectangles of the type $A_0 \times \{0,1\}^n$, $A_1 \times \{0,1\}^n$, where A_b is the subset of the input for which the first player communicates the bit b. Notice, $A_0 \cup A_1 = \{0,1\}^n$. If the next bit is sent by the second player, then each of the two rectangles above is further partitioned into two smaller rectangles depending upon

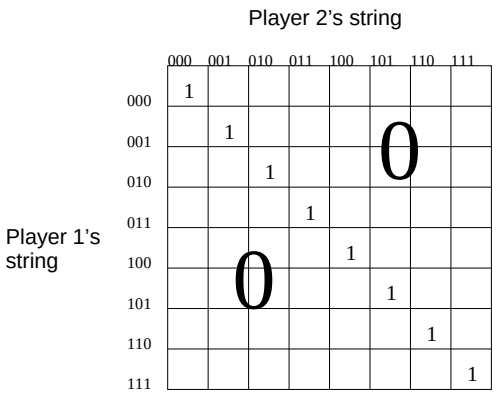


Figure 13.1 Matrix M(f) for the equality function when the inputs to the players have 3 bits. The numbers in the matrix are values of f.

what this bit was. Finally, if the total number of bits communicated is k then the matrix gets partitioned into 2^k rectangles. Note that each rectangle in the partition corresponds to a subset of input pairs for which the communication pattern thus far has been identical. (See Figure 13.2 for an example.) When the protocol stops, the value of f is determined by the sequence of bits sent by the two players, and thus must be the same for all pairs x, y in that rectangle. Thus the set of all communication patterns must lead to a partition of the matrix into *monochromatic* rectangles.

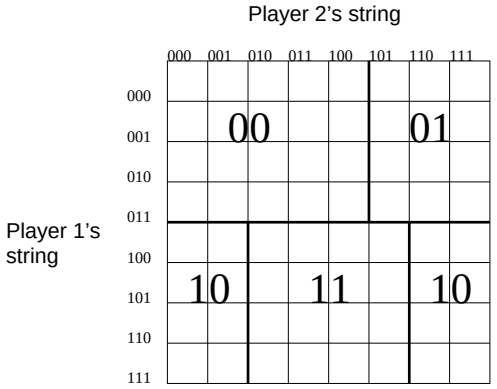


Figure 13.2 Two-way communication matrix after two steps. The large number labels are the concatenation of the bit sent by the first player with the bit sent by the second player.

Definition 13.7 A monochromatic tiling of $M(f)$ is a partition of $M(f)$ into disjoint monochromatic rectangles. We denote by $\chi(f)$ the minimum number of rectangles in any monochromatic tiling of $M(f)$. ♢

We have the following connection to communication complexity.

Theorem 13.8 *(Tiling and communication complexity* [AUY83]*)*
$\log_2 \chi(f) \leq C(f) \leq 16(\log_2 \chi(f))^2$.

PROOF: The first inequality follows from our above discussion, namely, if f has communication complexity k then it has a monochromatic tiling with at most 2^k rectangles. The second inequality is left as Exercise 13.5. ■

The following observation shows that for every function f whose communication complexity can be lower bounded using the fooling set method, the communication complexity can also be lower bounded by the tiling method. Hence the latter method subsumes the former.

Lemma 13.9 *If f has a fooling set with m pairs, then $\chi(f) \geq m$.* ♢

PROOF: If (x_1, y_1) and (x_2, y_2) are two of the pairs in the fooling set, then they cannot be in a monochromatic rectangle since not all of $(x_1, y_1), (x_2, y_2), (x_1, y_2), (x_2, y_1)$ have the same f value. ■

13.2.3 The rank method

Now we introduce an algebraic method to lower bound $\chi(f)$ (and hence the communication complexity of f). Recall the notion of *rank* of a square matrix: the size of the largest subset of rows that are linearly independent. The following lemma (left as Exercise 13.6) gives an equivalent characterization of the rank:

Lemma 13.10 *The* rank *of an $n \times n$ matrix M over a field $\mathbb{F}$, denoted by rank(M), is the minimum value of ℓ such that M can be expressed as*

$$M = \sum_{i=1}^{\ell} B_i \,,$$

where each B_i is an $n \times n$ matrix of rank 1. ♢

Note that $0, 1$ are elements of every field, so we can compute the rank of a binary matrix over any field we like. The choice of field can be crucial; see Exercise 13.10.

Observing that every monochromatic rectangle can be viewed (by filling out entries outside the rectangle with 0's) as a matrix of rank at most 1 , we obtain the following theorem:

Theorem 13.11 *For every function f, $\chi(f) \geq$ rank$(M(f))$.* ♢

Example 13.12
The matrix for the equality function is simply the identity matrix, and hence $\text{rank}(M(Eq)) = 2^n$. Thus, $C(EQ) \geq \log \chi(EQ) \geq n$, yielding another proof of Theorem 13.4.

13.2.4 The discrepancy method

For this method it is convenient to transform f into a ± 1-valued function by using the map $b \mapsto (-1)^b$ (i.e., $0 \mapsto +1, 1 \mapsto -1$. Thus $M(f)$ will also be a ± 1 matrix. We defined the *discrepancy* of a rectangle $A \times B$ in a $2^n \times 2^n$ matrix M to be

$$\frac{1}{2^{2n}} \left| \sum_{x \in A, y \in B} M_{x,y} \right| .$$

The *discrepancy* of the matrix $M(f)$, denoted by $\text{Disc}(f)$, is the maximum discrepancy among all rectangles. The following easy lemma relates it to $\chi(f)$.

Lemma 13.13 $\chi(f) \geq \dfrac{1}{Disc(f)}$. ♢

PROOF: If $\chi(f) \leq K$ then there exists a monochromatic rectangle having at least $2^{2n}/K$ entries. Such a rectangle will have discrepancy at least $1/K$. ■

Lemma 13.13 can be very loose. For the equality function, the discrepancy is at least $1-2^{-n}$ (namely, the discrepancy of the entire matrix), which would only give a lower bound of 2 for $\chi(f)$. However, $\chi(f)$ is at least 2^n, as already noted.
Now we describe a method to upper bound the discrepancy using *eigenvalues*.

Lemma 13.14 *(Eigenvalue bound) For any real matrix M, the discrepancy of a rectangle $A \times B$ is at most $\lambda_{max}(M)\sqrt{|A|\,|B|}/2^{2n}$, where $\lambda_{max}(M)$ is the magnitude of the largest eigenvalue of M.* ♢

PROOF: Let $\mathbf{1}_S \in \mathbb{R}^{2^n}$ denote the characteristic vectors of a subset $S \subseteq \{0,1\}^n$ (i.e., the x^{th} coordinate of $\mathbf{1}_S$ is equal to 1 if $x \in S$ and to 0 otherwise). Note $\|\mathbf{1}_S\|_2 = \sqrt{\sum_{x\in S} 1^2} = \sqrt{|S|}$. Note also that for every $A, B \subseteq \{0,1\}^n$, $\sum_{x\in A, y\in B} M_{x,y} = \mathbf{1}_A^\dagger M \mathbf{1}_B$.

The discrepancy of the rectangle $A \times B$ is

$$\frac{1}{2^{2n}}\mathbf{1}_A^\dagger M \mathbf{1}_B \leq \frac{1}{2^{2n}}\lambda_{max}(M)\left|\mathbf{1}_A^\dagger \mathbf{1}_B\right| \leq \frac{1}{2^{2n}}\lambda_{max}(M)\sqrt{|A|\,|B|}\,,$$

where the last inequality uses Cauchy-Schwartz. ■

Example 13.15
The *mod 2 inner product* function defined as $f(x,y) = x \odot y = \sum_i x_i y_i (\mathrm{mod} 2)$ has been encountered a few times in this book. To bound its discrepancy, let N be the $pm1$ matrix corresponding to f (i.e., $M_{x,y} = (-1)^{x\odot y}$). It is easily checked that every two distinct rows (columns) of N are orthogonal, every row has ℓ_2 norm $2^{n/2}$, and that $N^T = N$. Thus we conclude that $N^2 = 2^n I$ where I is the unit matrix. Hence every eigenvalue is either $+2^{n/2}$ or $-2^{n/2}$, and thus Lemma 13.14 implies that the discrepancy of a rectangle $A \times B$ is at most $2^{-3n/2}\sqrt{|A|\,|B|}$ and the overall discrepancy is at most $2^{-n/2}$ (since $|A|, |B| \leq 2^n$).

13.2.5 A technique for upper bounding the discrepancy

We describe an upper bound technique for the discrepancy that will later be useful also in the multiparty setting (Section 13.3). As in Section 13.2.4, we assume that f is a ± 1-valued function. We define the following quantity:

Definition 13.16 $\mathcal{E}(f) = \mathsf{E}_{a_1,a_2,b_1,b_2}\left[\prod_{i=1,2}\prod_{j=1,2} f(a_i,b_j)\right]$. ♢

Note that $\mathcal{E}(f)$ can be computed, like the rank, in time polynomial in the size of the matrix $M(f)$. By contrast, the definition of discrepancy involves a maximization over all possible subsets A, B, and a naive algorithm for computing it would take time exponential in the size of $M(f)$. (Indeed, the discrepancy is **NP**-hard to compute exactly, though it can be approximated efficiently— see the chapter notes.) The following Lemma relates these two quantities.

Lemma 13.17

$$Disc(f) \leq \mathcal{E}(f)^{1/4}.$$

PROOF: We need to show that for every rectangle $A \times B$,

$$\mathcal{E}(f) \geq \left(\tfrac{1}{2^{2n}}\left|\sum_{a\in A, b\in B} f(a,b)\right|\right)^4.$$

Let g, h the characteristic functions of A and B respectively. (That is $g(a)$ equals 1 if $a \in A$ and 0 otherwise; $h(b)$ equals 1 if $b \in B$ and 0 otherwise.) Then, the right hand side is simply $\left(\mathsf{E}_{a,b\in\{0,1\}^n}[f(a,b)g(a)h(b)]\right)^4$. But,

$$\begin{aligned}
\mathcal{E}(f) &= \mathop{\mathsf{E}}_{a_1,a_2}\left[\mathop{\mathsf{E}}_{b_1,b_2}\left[\prod_{i=1,2}\prod_{j=1,2} f(a_i,b_j)\right]\right] \\
&= \mathop{\mathsf{E}}_{a_1,a_2}\left[\left(\mathop{\mathsf{E}}_{b}[f(a_1,b)f(a_2,b)]\right)^2\right] \\
&\geq \mathop{\mathsf{E}}_{a_1,a_2}\left[g(a_1)^2 g(a_2)^2\left(\mathop{\mathsf{E}}_{b}[f(a_1,b)f(a_2,b)]\right)^2\right] \qquad \text{(using } g(a)\leq 1\text{)} \\
&= \mathop{\mathsf{E}}_{a_1,a_2}\left[\left(\mathop{\mathsf{E}}_{b}[f(a_1,b)g(a_1)f(a_2,b)g(a_2)]\right)^2\right] \\
&\geq \left(\mathop{\mathsf{E}}_{a_1,a_2}\left[\mathop{\mathsf{E}}_{b}[f(a_1,b)g(a_1)f(a_2,b)g(a_2)]\right]\right)^2 \qquad \text{(using } \mathsf{E}[X^2]\geq \mathsf{E}[X]^2\text{)} \\
&= \left(\mathop{\mathsf{E}}_{b}\left[\left(\mathop{\mathsf{E}}_{a}[f(a,b)g(a)]\right)^2\right]\right)^2 \\
&\geq \left(\mathop{\mathsf{E}}_{a,b}[f(a,b)g(a)h(b)]\right)^4. \qquad \text{(repeating previous steps)}
\end{aligned}$$

■

Exercise 13.12 asks you to derive a lower bound for the inner product function using this technique. We will see another example in Section 13.3.

13.2.6 Comparison of the lower bound methods

The tiling argument is the strongest lower bound technique, since bounds on rank, discrepancy and fooling sets imply a bound on $\chi(f)$, and hence can never prove better lower bounds than the tiling argument. Also, as Theorem 13.10, $\log \chi(f)$ fully characterizes the communication complexity of f up to a constant factor. The rank and fooling set methods are incomparable, meaning that each can be stronger than the other for some function. However, if we ignore constant factors, the rank method is always at least as strong as the fooling set method (see Exercise 13.8). Also, we can separate the power of these lower bound arguments. For instance, we know functions for which a polynomial gap exists between $\log \chi(f)$ and $\log \mathrm{rank}(M(f))$. However, the following conjecture (we only state one form of it) says that rank is in fact optimal up to a polynomial factor.

Conjecture 13.18 *(log rank conjecture)*
There is a constant $c > 1$ such that $C(f) = O(\log(\mathrm{rank}(M(f)))^c)$ for all f and all input sizes n, where rank is taken over the reals.

Of course, the difficult part of the above conjecture is to show that low rank implies a low-complexity protocol for f. Though we are still far from proving this, Nisan and Wigderson have shown that at least low rank implies low value of $1/\mathrm{Disc}(f)$.

Theorem 13.19 *([NW94])* $1/Disc(f) = O(rank(f)^{3/2})$. ♢

13.3 Multiparty communication complexity

There is more than one way to generalize communication complexity to a multiplayer setting. The most interesting model turns out to be the "number on the forehead" model: each player

has a string on his head which everybody else can see but he cannot. That is, there are k players and k strings $x_1, \ldots, x_k$, and Player i gets all the strings *except* for x_i. The players are interested in computing a value $f(x_1, x_2, \ldots, x_k)$ where $f : (\{0,1\}^n)^k \to \{0,1\}$ is some fixed function. As in the 2-player case, the k players have an agreed-upon protocol for communication (which was decided before they were given their strings), and all their communication is posted on a "public blackboard" that all of them can see (the protocol also determines the order in which the players write on the blackboard). The last message sent should contain (or at least easily determine) the value $f(x_1, \ldots, x_k)$ of the function on the inputs. By analogy with the 2-player case, we denote by $C_k(f)$ the number of bits that must be exchanged by the best protocol. Note that it is at most $n+1$, since it suffices for any $j \neq i$ to write x_i on the blackboard, at which point the ith player knows all k strings and can determine and publish $f(x_1, \ldots, x_k)$.

Example 13.20
Consider computing the function

$$f(x_1, x_2, x_3) = \bigoplus_{i=1}^{n} \mathrm{maj}(x_{1i}, x_{2i}, x_{3i})$$

in the 3-party model where x_1, x_2, x_3 are n bit strings. The communication complexity of this function is 3: each player counts the number of i's such that she can determine the majority of x_{1i}, x_{2i}, x_{3i} by examining the bits available to her. She writes the parity of this number on the blackboard, and the final answer is the parity of the players' bits. This protocol is correct because the majority for each row is known by either 1 or 3 players, and both are odd numbers.

Example 13.21 *(Generalized Inner Product)*
The *generalized inner product function* $GIP_{k,n}$ maps nk bits to 1 bit as follows

$$f(x_1, \ldots, x_k) = \bigoplus_{i=1}^{n} \bigwedge_{j=1}^{k} x_{ji}. \tag{1}$$

Notice, for $k = 2$ this reduces to the mod 2 inner product of Example 13.15.

For the 2-player model we introduced the notion of a monochromatic rectangle in order to prove lower bounds. Specifically, a communication protocol can be viewed as a way of partitioning the matrix $M(f)$: if the protocol exchanges c bits, then the matrix is partitioned into 2^c rectangles, all of which must be monochromatic if the protocol is valid.

The corresponding notion in the k-party case is a cylinder intersection. A *cylinder in dimension i* is a subset S of the inputs such that if $(x_1, \ldots, x_k) \in S$ then $(x_1, \ldots, x_{i-1}, x'_i, x_{i+1}, \ldots, x_k) \in S$ for all x'_i also. A *cylinder intersection* is $\cap_{i=1}^{k} T_i$ where T_i is a cylinder in dimension i. Since player i's communication does not depend upon x_i, it can be viewed as partitioning the set of inputs according to cylinders in dimension i. Thus we conclude that at the end of the protocol, the cube $\{0,1\}^{nk}$ is partitioned using cylinder intersections, and if the protocol communicates c bits, then the partition consists of at most 2^c monochromatic cylinder intersections. Thus we have proved:

Lemma 13.22 *If every partition of $M(f)$ into monochromatic cylinder intersections requires at least R cylinder intersections, then the k-party communication complexity is at least $\lceil \log_2 R \rceil$, where $M(f)$ is the k-dimensional table whose $(x_1, \ldots, x_k)^{th}$ entry is $f(x_1, \ldots, x_k)$.* ♢

Discrepancy-based lower bound

In this section, we will assume as in our earlier discussion of discrepancy that the range of the function f is $\{-1,1\}$. We define the *k-party discrepancy* of f by analogy to the 2-party case

$$\mathrm{Disc}(f) = \frac{1}{2^{nk}} \max_T \left| \sum_{(a_1,a_2,\ldots,a_k)\in T} f(a_1,a_2,\ldots,a_k) \right|,$$

where T ranges over all cylinder intersections.

To upper bound the discrepancy we introduce the k-party analogue of $\mathcal{E}(f)$. Let a (k,n)-*cube* be (multi) subset D of $\{0,1\}^{nk}$ of 2^k points of the form $\{a_1,a'_1\}\times\{a_2,a'_2\}\times\cdots\times\{a_k,a'_k\}$, where each $a_i, a'_i \in \{0,1\}^n$. We define

$$\mathcal{E}(f) = \mathop{\mathsf{E}}_{\substack{D \\ (k,n)\text{ cube}}} \left[\prod_{\mathbf{a}\in D} f(\mathbf{a}) \right].$$

Notice that the definition of $\mathcal{E}(f)$ for the 2-party case is recovered when $k=2$. The next lemma is also an easy generalization.

Lemma 13.23

$$\mathit{Disc}(f) \leq (\mathcal{E}(f))^{1/2^k}.$$

The proof is analogous to the proof of Lemma 13.17 and is left as Exercise 13.14. The only difference is that we need to repeat the basic step of that proof k times instead of 2 times.

Now we can prove a lower bound for the Generalized Inner Product (GIP) function. Note that since we changed the range to $\{-1,1\}$, this function is now defined as

$$GIP_{k,n}(x_1,x_2,\ldots,x_k) = (-1)^{\sum_{i\leq n}\prod_{j\leq k} x_{ji}} \tag{2}$$

(we can omit the reduction modulo 2 since $(-1)^m = (-1)^{m \pmod 2}$ for every m).

Theorem 13.24 *(Lower bound for generalized inner product)*
The function $GIP_{k,n}$ has k-party communication complexity $\Omega(n/4^k)$.

Proof: By Lemma 13.23 it suffices to upper bound $\mathcal{E}(GIP_{k,n})$. Using (2) we see that for every k,n,

$$GIP_{k,n}(x_1,\ldots,x_k) = \prod_{i=1}^{n} GIP_{k,1}(x_{1,i},\ldots,x_{k,i}).$$

where we define $GIP_{k,1}(x_1,\ldots,x_k) = (-1)^{\prod_{j\leq k} x_i}$. Thus

$$\mathcal{E}(GIP_{k,n}) = \mathop{\mathsf{E}}_{\substack{D \\ (k,n)\text{-cube}}} [\prod_{\mathbf{a}\in D}\prod_{i=1}^{n} GIP_{k,1}(\mathbf{a}_i)],$$

where for a vector $\mathbf{a} = (a_1,\ldots,a_k)$ in $(\{0,1\}^n)^k$, $\mathbf{a}_i$ denotes the k bit string $a_{1,i},\ldots,a_{k,i}$. But because each coordinate is chosen independently, the right hand side is equal to

$$\prod_{i=1}^{n} \mathop{\mathsf{E}}_{\substack{C \\ (k,1)\text{ cube}}} [\prod_{\mathbf{a}\in C} GIP_{k,1}(\mathbf{a})] = \mathcal{E}(GIP_{k,1})^n.$$

But $\mathcal{E}(GIP_{k,1}) \leq 1 - 2^{-k}$. Indeed a random $(k,1)$-cube $C = \{a_1,a'_1\}\times\cdots\times\{a_k,a'_k\}$ has probability 2^{-k} to satisfy the event E that for every i the pair (a_i,a'_i) is either $(0,1)$ or $(0,1)$. But since $GIP_{k,1}(\mathbf{a}) = -1$ if and only if $\mathbf{a}$ is the all ones vector, if E happens then there is exactly one k-bit vector $\mathbf{a}$ in C such that $GIP_{k,1}(\mathbf{a}) = -1$ and for all other vectors

$\mathbf{b} \in C$, $GIP_{k,1}(\mathbf{b}) = 1$. Hence if E happens then $\prod_{\mathbf{a}\in C} GIP_{k,1}(\mathbf{a}) = -1$, and since this product is always at most 1,

$$\mathcal{E}(GIP_{k,1}) = \mathop{\mathsf{E}}_{\substack{C \\ (k,1) \text{ cube}}} [\prod_{\mathbf{a}\in C} GIP_{k,1}(\mathbf{a})] \leq 2^{-k} \cdot -1 + (1-2^{-k}) \cdot 1 \leq 1 - 2^{-k}\,.$$

Hence $\mathcal{E}(GIP_{k,n}) \leq (1-2^{-k})^n \leq e^{-n/2^k} = 2^{-\Omega(n/2^k)}$. Thus $\mathrm{Disc}(GIP_{k,n}) \leq 2^{-\Omega(n/4^k)}$, implying that the k-party communication complexity of $GIP_{k,n}$ is $\Omega(n/4^k)$.■

At the moment, we do not know of any explicit function f for which $C_k(f) \geq n2^{-o(k)}$ and in particular have no non-trivial lower bound for computing explicit functions $f : (\{0,1\}^n)^k \to \{0,1\}$ for $k \geq \log n$. Such a result could be useful to obtain new circuit lower bounds; see Section 14.5.1.

13.4 Overview of other communication models

We outline some of the alternative settings in which communication complexity has been studied.

Randomized protocols: One can consider *randomized* protocols for jointly computing the value of a function. In such protocols, all players have access to a shared random string r, which they use in determining their actions. We define $R(f)$ to be the *expected* number of bits communicated by the players. It turns out that randomization can sometimes make a significant difference. For example, the equality function has a randomized communication protocol with $O(\log n)$ complexity (see Exercise 13.15). Nevertheless, there are techniques to prove lower bounds for such protocols as well.

Non-deterministic protocols: One can also define *non-deterministic communication complexity* analogously to the definition of the class **NP**. In a non-deterministic protocol, the players are both provided an additional third input z ("nondeterministic guess") of some length m that may depend on x, y. Apart from this guess, the protocol is deterministic. We require that $f(x,y) = 1$ iff there exists a string z that makes the players output 1, and the cost of the protocol is m plus the number of bits communicated. Once again, this can make a significant difference. For example both the *inequality* and *intersection* functions (i.e., the negations of the functions EQ and the function DISJ of Example 13.6) are easily shown to have logarithmic non-deterministic communication complexity. Analogously to the definition of **coNP**, one can define the co-non-deterministic communication complexity of f to be the non-deterministic communication complexity of the function $g(x,y) = 1 - f(x,y)$. Interestingly, it can be shown that if f has non-deterministic communication complexity k and co-non-deterministic communication complexity ℓ, then $C(f) \leq 10k\ell$, hence implying that in the communication complexity world the intersection of the classes corresponding to **NP** and **coNP** is equal to the class corresponding to **P**. In contrast, we believe that $\mathbf{P} \neq \mathbf{NP} \cap \mathbf{coNP}$.

Average case protocols: Just as we can study average-case complexity in the Turing machine model, we can study communication complexity when the inputs are chosen from a distribution $\mathcal{D}$. This is defined as

$$C_{\mathcal{D}}(f) = \min_{\mathcal{P} \text{protocol for } f} \mathop{\mathsf{E}}_{(x,y)\in_R \mathcal{D}} [\text{Number of bits exchanged by } \mathcal{P} \text{ on } x,y.]$$

Computing a non Boolean function: Here the function's output is not just $\{0,1\}$ but an m-bit number for some m. We discuss one example in the exercises.

Asymmetric communication: In this model the "cost" of communication is asymmetric: there is some B such that it costs the first player B times as much to transmit a bit than it does the second player. The goal is to minimize the total cost.

Computing a relation: One can consider protocols that aim to hit a relation rather than computing a function. That is, we have a relation $R \subseteq \{0,1\}^n \times \{0,1\}^n \times \{1,2,\ldots,m\}$ and given $x, y \in \{0,1\}^n$ the players seek to agree on any $b \in \{1,2,\ldots,m\}$ such that $(x,y,b) \in R$. See Exercise 13.16.

These and many other settings are discussed in [KN97].

WHAT HAVE WE LEARNED?

- The *communication complexity* of a two input function f is the number of bits that a player holding x and a player holding y need to exchange to compute $f(x,y)$.
- Methods to lower bound the communication complexity of specific functions include the fooling set, tiling, rank, and discrepancy methods. Using these methods we have several examples of explicit functions on two n-bit inputs whose communication complexity is at least n.
- The *multiparty communication complexity* of a k-input function f is the number of bits that k parties need to exchange to compute f where the i^{th} player has all the inputs *except* the i^{th} input. The best known lower bound of the k-party communication complexity of an explicit function is of the form $n/2^{-\Omega(k)}$.
- Other models of communication complexity studies include probabilistic, nondeterministic, and average-case communication complexity, and the communication complexity of computing relations.

Chapter notes and history

This chapter barely scratched the surface of this self-contained mini-world within complexity theory; an excellent and detailed treatment can be found in the book by Kushilevitz and Nisan [KN97] (though it does not contain some of the newer results).

Communication complexity was first defined by Yao [Yao79]. Other early papers that founded the field were Papadimitriou and Sipser [PS82], Mehlhorn and Schmidt [MS82] (who introduced the rank lower bound) and Aho, Ullman and Yannakakis [AUY83].

We briefly discussed parallel computation in Chapter 6. Yao [Yao79] invented communication complexity as a way to lower bound the running time of parallel computers for certain tasks. The idea is that the input is distributed among many processors, and if we partition these processors into two halves, we may lower bound the computation time by considering the amount of communication that must necessarily happen between the two halves. A similar idea is used to prove time/space lower bounds for VLSI circuits. For instance, in a VLSI chip that is an $m \times m$ grid, if the communication complexity for a function is greater than c, then the time required to compute it is at least c/m.

Communication complexity is also useful in time-space lower bounds for Turing machines (see Exercise 13.4), and circuit lower bounds (see Chapter 14).

Data structures such as heaps, sorted arrays, lists etc. are basic objects in algorithm design. Often, algorithm designers wish to determine if the data structure they have designed is the best possible. Communication complexity lower bounds can be used to establish such results; see [KN97]. *Streaming algorithms*, in which an algorithm can only make one pass on a very large input, is another area where communication complexity bounds imply various optimality and impossibility results. Alon, Matias and Szegedy [AMS96] were the first to use communication complexity as a tool for proving lower bounds on streaming algorithms. Ever since then, there has been extensive research on both the application of communication complexity to lower bound problems in stream algorithms, as well as in the development of new tools for communication complexity inspired by the need to tighten existing gaps in streaming problems like frequency estimation (see e.g., [CSWY01, BYJKS02]).

Yannakakis [Yan88] has shown how to use communication complexity lower bounds to prove lower bounds on the size of polytopes representing **NP**-complete problems. Solving the open

problem mentioned in Exercise 13.13 would prove a lower bound for the polytope representing vertex cover.

Theorem 13.24 is due to Babai, Nisan and Szegedy, though our proof follows Raz's simplification [Raz00] of Chung's proof [Chu90].

Computing the discrepancy (also known as the *cut norm*) of a given real matrix, and even approximating it to an arbitrarily small constant is **NP**-hard. But it can be approximated using semi-definite programming within some constant factor K_G; see Alon and Naor [AN04]. (K_G is a number between 1.5 and 1.8 that is known as *Grothendieck's constant*; determining its exact value is a major open problem.) The notion of discrepancy is known as *regularity* in the context of the famous *Szemerédi regularity lemma* [Sze76]. In that context the parameter $\mathcal{E}(f)$ is analogous to the fraction of 4-cycles in a given bipartite graph, which is again related to the regularity of the graph. Multi-party discrepancy is related to the hypergraph regularity lemma, and the parameter $\mathcal{E}(f)$ was used by Gowers [Gow07] in his proof of his lemma. A closely related group-theoretic parameter (sometimes known as Gowers's norm or Gowers's uniformity) was used by Gowers [Gow01] in his proof of the quantitatively improved version of Szemerédi's Theorem guaranteeing the existence of large arithmetic progression in dense sets. The book by Tau and Vu [TV06] contains an excellent exposition of these topics.

Lovasz and Saks [LS88] have observed that the log rank conjecture is related to a conjecture in discrete mathematics concerning chromatic number and rank of the adjacency matrix. The original log rank conjecture was that $C(f) = O(\log \mathrm{rank}(M(f)))$ but this was disproved by Raz and Spieker [RS93]. A comparison of rank and fooling set arguments appears in the paper by Dietzfelbinger, Hromkovic and Schnitger [DHS94].

In general, the complexity of computing $C(f)$ and $C_k(f)$ is not understood, and this may have some connection to why it is difficult in practice for us to prove lower bounds on these quantities. It is also intriguing that the lower bounds that we do prove involve quantities such as rank and fooling sets that are computable in polynomial time given $M(f)$. (This is an instance of the more widespread phenomenon of *natural proofs* encountered in Chapter 23.) In this regard, it is interesting to note that the *Discrepancy* parameter is **NP**-hard to compute, but can be approximated within a constant multiplicative factor in the 2-player setting by a polynomial-time algorithm [AN04]. By contrast, computing the discrepancy in the 3-player setting seems very hard (though no hardness results seem to appear anywhere); this may perhaps explain why lower bounds are so difficult in the multiplayer setting.

One relatively recent area not mentioned in this chapter is *quantum* communication complexity, where the parties may exchange quantum states with one another, see [Bra04]. Interestingly, some techniques developed in this setting [She07] were used to obtain new $\Omega(n^{1/(k-1)}/2^{2^k})$ lower bounds on the k-party communication complexity of the *disjointness* function [LS07, CA08], thus obtaining a strong separation of non-deterministic and deterministic k-party communication complexity.

Exercises

13.1 Prove Lemma 13.5.

13.2 Prove that for every set $S \subseteq \{(x,x) : x \in \{0,1\}^n\}$ and any communication protocol Π that correctly computes the equality function on n-bit inputs, there exists a pair of inputs in S on which Π uses at least $\log |S|$ bits.

13.3 Prove that a single tape TM (one whose input tape is also its read/write work tape) takes at least $O(n^2)$ to decide the language of *palindromes* $\mathsf{PAL} = \{x_n \cdots x_1 x_1 \cdots x_n : x_1, \ldots, x_n \in \{0,1\}^n, n \in \mathbb{N}\}$ of Example 1.1. H461

13.4 If $S(n) \leq n$, show that a space $S(n)$ TM takes at least $\Omega(n^2/S(n))$ steps to decide the language $\{x\#x : x \in \{0,1\}^*\}$. H461

13.5 Prove the second inequality of Theorem 13.8. That is, prove that for every $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, $C(f) = O(\log^2 \chi(f))$. H462

13.6 Prove Lemma 13.10. H462

13.7 Show that for almost all functions $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ the rank of $M(f)$ over GF(2) is n whereas the size of the largest fooling set is less than $3 \log n$. This shows that the rank lowerbound can be exponentially better than the fooling set lowerbound.

13.8 For two $n \times n$ matrices A, B, define its *tensor product* $A \otimes B$ as the $n^2 \times n^2$ matrix whose entries are indexed by 4-tuples from $[n]$. Show that the rank of $A \otimes B$ (over any field) is the product of the ranks of A and B.

Use the above fact to show that if a function f has a fooling set of size S then the rank method can be used to give a lowerbound of at least $1/2\lceil \log S \rceil$ on the communication complexity. This shows that the rank method is never much worse than the fooling set method.

13.9 Show that if M is 0/1 real matrix, and M' is the ± 1 matrix obtained by applying the transformation $a \mapsto (-1)^a$ to the entries of M, then $\text{rank}(M) - 1 \leq \text{rank}(M') \leq \text{rank}(M) + 1$. H462

13.10 Consider x, y as vectors over $GF(2)^n$ and let $f(x, y)$ be their inner product mod 2. Prove using the rank method that the communication complexity is n. H462

13.11 Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be such that all rows of $M(f)$ are distinct. Show that $C(f) \geq \log n$. H462

13.12 Prove that $\mathcal{E}(IP) \leq 2^{-n}$, where IP is the inner product function. Derive a lower bound for the communication complexity of IP. H462

13.13 For any graph G with n vertices, consider the following communication problem: Player 1 receives a clique C in G, and Player 2 receives an independent set I. They have to communicate in order to determine $|C \cap I|$. (Note that this number is either 0 or 1.) Prove an $O(\log^2 n)$ upper bound on the communication complexity.

Can you improve your upper bound or prove a lower bound better than $\Omega(\log n)$? (Open question)

13.14 Prove Lemma 13.23.

13.15 Prove that the randomized communication complexity of the equality function (i.e., $R(EQ)$) is at most $O(\log n)$. (Note that a randomized communication protocol is allowed to output the wrong answer with probability at most $1/3$.) H462

13.16 (Karchmer-Wigderson games [KW88]) Consider the following problem about computing a relation. Associate the following communication problem with any function $f : \{0,1\}^n \to \{0,1\}$. Player 1 gets any input x such that $f(x) = 0$ and player 2 gets any input y such that $f(y) = 1$. They have to communicate in order to determine a bit position i such that $x_i \neq y_i$. Show that the communication complexity of this problem is *exactly* the minimum depth of any circuit that computes f. (The maximum fan-in of each gate is 2.) H462

13.17 Use the previous question to show that computing the parity of n bits requires depth at least $2 \log n$.

13.18 Show that the following computational problem is in **EXP**: given the matrix $M(f)$ of a Boolean function, and a number K, decide if $C(f) \leq K$.

(Open since Yao [Yao79]) Can you show this problem is complete for some complexity class?

13.19 ([AMS96]) A space-$S(n)$ *streaming algorithm* is a space-$S(n)$ TM M that makes only one sweep of its input. This setup naturally occurs in many applications. Prove that there is no space-$o(n)$ streaming algorithm that solves the following problem: given a sequence $x_1, \ldots, x_m$ in $[n]$, compute the frequency of the *most frequent element*— $\max_{x \in [n]} |\{i : x_i = x\}|$. Can you show that one cannot even *approximate* this problem to within a 3/4 factor in $o(n)$? H462

Chapter 14

Circuit lower bounds

Complexity theory's Waterloo

In Chapter 2 we saw that if there is any **NP** language that cannot be computed by polynomial-sized circuits then $\mathbf{NP} \neq \mathbf{P}$. Thus proving circuit lower bounds is a potential approach for proving $\mathbf{NP} \neq \mathbf{P}$. Furthermore, there is a reason to hope that this is a viable approach, since the Karp-Lipton Theorem (Theorem 6.19) shows that if the polynomial hierarchy **PH** does not collapse then there *exists* an **NP** language that does not have polynomial size circuits.

In the 1970s and 1980s, many researchers came to believe that proving circuit lower bounds represented the best route to resolving **P** versus **NP**, since circuits seem easier to reason about than Turing machines. The success in this endeavor was mixed.

Progress on general circuits has been almost nonexistent: a lower bound of n is trivial for any function that depends on all its input bits. We are unable to prove even a superlinear circuit lower bound for any **NP** problem— the best we can do after years of effort is $5n-o(n)$.

To make life (comparatively) easier, researchers focussed on restricted circuit classes, and were successful in proving some good lower bounds. We prove some of the major results of this area, specifically, for bounded depth circuits (Section 14.1), bounded depth circuits with "counting" gates (Section 14.2), and monotone circuits (Section 14.3). In all these resuls we have a notion of "progress" of the computation. We show that small circuits simply cannot achieve the amount of progress necessary to compute the output from the inputs.

In Section 14.4 we indicate the questions at the frontier of circuit lower bound research, where we are currently stuck. A researcher starting work on this area may wish to focus on one of the open questions described there. Later in Chapter 23 we'll explain some of the inherent obstacles that need to be overcome to make further progress.

14.1 AC^0 and Håstad's Switching Lemma

As we saw in Chapter 6, $\mathbf{AC}^0$ is the class of languages computable by circuit families of constant depth, polynomial size, and whose gates have unbounded fan-in. (We need to allow the fan-in in the circuit to be unbounded since otherwise the output cannot receive information from all input bits.)

We saw in Chapter 2 (Claim 2.13) that every Boolean function can be computed by a circuit of depth 2 and exponential size— that is a CNF (or DNF) formula. When students study digital logic design they learn how to do "circuit minimization" using Karnaugh maps, and the circuits talked about in that context are depth 2 circuits. Indeed, it is easy to show (using for example the Karnaugh map technique) that the minimum DNF or CNF representing even very simple functions (such as the parity function described below) has to be of exponential size. However, those techniques do not seem to generalize to even depth 3 circuits, not to mention the class $\mathbf{AC}^0$ of (arbitrarily large) constant depth circuits of polynomial size that we encountered in Section 6.7.1.

The burning question in the late 1970s was whether problems like Clique and TSP have $\mathbf{AC}^0$ circuits. In 1981, Furst, Saxe and Sipser and independently, Ajtai, showed that they do not. In fact their lower bound applied to a much simpler function:

Theorem 14.1 *([FSS81, Ajt83])*
Let $\bigoplus$ be the parity function. That is, for every $x \in \{0,1\}^n$, $\bigoplus(x_1,\ldots,x_n) = \sum_{i=1}^n x_i \pmod 2$. Then $\bigoplus \notin \mathbf{AC}^0$.

The main tool in the proof of Theorem 14.1 is the concept of *random restrictions*. Let f be a function computable by a depth d circuit of polynomial size and suppose that we choose at random a vast majority (i.e., $n - n^\epsilon$ for some constant $\epsilon > 0$ depending on d) of the input variables and fix each such variable to be either 0 or 1 at random. We'll prove that with positive probability, the function f subject to this restriction is *constant* (i.e., it is either always zero or always one). Since the parity function cannot be made a constant by fixing values to a subset of the variables, it follows that it cannot be computed by a constant depth polynomial-sized circuit.

14.1.1 Håstad's switching lemma

As in Section 2.3, we define a k-CNF to be a Boolean formula that is an AND of OR's where each OR involves at most k variables. Similarly, a k-DNF is an OR of AND's where each AND involves at most k variables. If f is a function on n variables and ρ is a partial assignment (also known as a *restriction*) to the variables of f, then we denote by $f|_\rho$ the restriction of f under ρ. That is, $f|_\rho$ takes an assignment τ to the variables not assigned by ρ as input, and outputs f applied to ρ and τ. Now we prove the main lemma about how a circuit simplifies under a random restriction.

Lemma 14.2 *(Håstad's switching lemma* [Hås86]*)*
Suppose f is expressible as a k-DNF, and let ρ denote a random restriction that assigns random values to t randomly selected input bits. Then for every $s \geq 2$.

$$\Pr_\rho[f|_\rho \text{ is not expressible as } s\text{-CNF }] \leq \left(\frac{(n-t)k^{10}}{n}\right)^{s/2} \quad (1)$$

where $f|_\rho$ denotes the function f restricted to the partial assignment ρ.

We defer the proof of Lemma 14.2 to Section 14.1.2. We'll typically use this lemma with k, s constants and $t \approx n - \sqrt{n}$ in which case the guaranteed bound on the probability will be n^{-c} for some constant c. Note that by applying Lemma 14.2 to the function $\neg f$, we can get the same result with the terms DNF and CNF interchanged.

Proving Theorem 14.1 from Lemma 14.2. Now we show how Håstad's lemma implies that parity is not in $\mathbf{AC}^0$. We start with any $\mathbf{AC}^0$ circuit and assume that the circuit has been simplified as follows (these simplifications are straightforward to do and are left as Exercises 14.1 and 14.2): **(a)** All fanouts are 1; the circuit is a tree. **(b)** All *not* gates are at the input level of the circuit; equivalently, the circuit has $2n$ input wires, with the last n of them being the negations of the first n **(c)** The $\vee$ and $\wedge$ gates alternate (i.e., at each level of the tree there are either only $\vee$ gates or only $\wedge$ gates). **(d)** The bottom level has $\wedge$ gates of fan-in 1.

Let n^b denote an upper bound on the number of gates in the circuit with the above properties. We randomly restrict more and more variables, where each step with high

probability will reduce the depth of the circuit by 1 and will keep the bottom level at a constant fan-in. Specifically, letting n_i stand for the number of unrestricted variables after step i, we restrict $n_i - \sqrt{n_i}$ variables at step $i+1$. Since $n_0 = n$, we have $n_i = n^{1/2^i}$. Let $k_i = 10b2^i$. We'll show that with high probability, after the i^{th} restriction we're left with a depth-$(d-i)$ circuit with at most k_i fan-in in the bottom level. Indeed, suppose that the bottom level contains $\wedge$ gates and the level above it contains $\vee$ gates. The function each such $\vee$ gate computes is a k_i-DNF and hence by Lemma 14.2, with probability $1-\left(\frac{k_i^{10}}{n^{1/2^{i+1}}}\right)^{k_{i+1}/2}$, which is at least $1 - 1/(10n^b)$ for large enough n, the function such a gate computes after the $i+1^{th}$ iteration will be expressible as a k_{i+1}-CNF. Since the top gate of a CNF formula is $\wedge$, we can merge those $\wedge$ gates with the $\wedge$-gate above them, reducing the depth of the circuit by one (see Figures 14.1 and 14.2). The symmetric reasoning applies in the case the bottom level consists of $\vee$ gates— in this case we use the lemma to transform the k_i-CNF of the level above it into a k_{i+1}-DNF. Note that we apply the lemma at most once per each of the at most n^b gates of the original circuit. By the union bound, with probability 9/10, if we continue this process for $d-2$ steps, we'll get a depth two circuit with fan-in $k = k_{d-2}$ at bottom level. That is, either a k-CNF or k-DNF formula. But such a formula can be made constant by fixing at most k of the variables (e.g., in a DNF ensuring that the first clause has value 1). Since the parity function is not constant under any restriction of less than n variables, this proves Theorem 14.1. ■

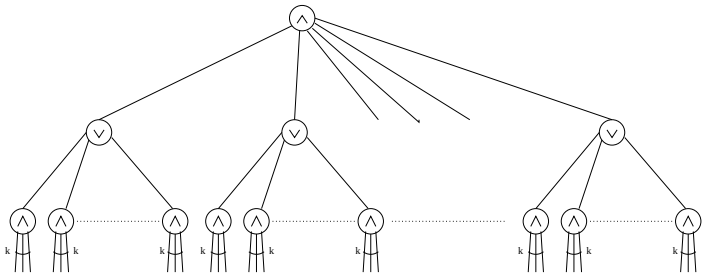


Figure 14.1 Circuit before Håstad switching transformation.

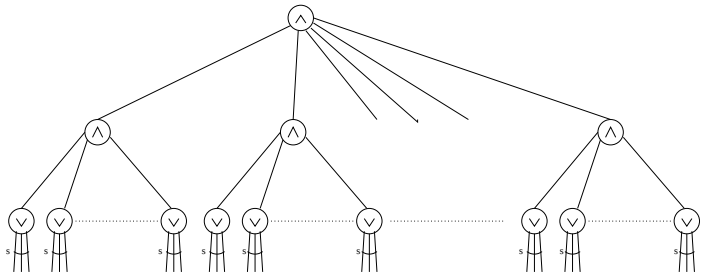


Figure 14.2 Circuit after Håstad switching transformation. Notice that the new layer of $\wedge$ gates can be collapsed with the single $\wedge$ parent gate, to reduce the number of levels by one.

14.1.2 Proof of the switching lemma (Lemma 14.2)

Now we prove the Switching Lemma. The original proof was more complicated; this one is due to Razborov.

We need a few definitions. Let a *min-term* of a function f be a partial assignment to f's variables that makes f output 1 *irrespective of the assignments to the remaining variables.* Thus every clause in a k-DNF formula for f yields a size-k min-term of f. A

max-term is a partial assignment to f's variables that makes f output 0 regardless of the other variables. Thus every clause in a k-CNF formula for f yields a size-k max-term of f. We will assume throughout that min-terms (respectively max-terms) are minimal, in the sense that no assignment to a proper subset of the term's variables would make the function 1 (respectively, 0). Thus a function that is *not* expressible as an s-CNF must have at least one max-term of length $s+1$ (see Exercise 14.3).

If π and ρ are restrictions on disjoint sets of variables, then we denote by $\pi\rho$ their union (i.e., if π assigns values to variables in S and ρ assigns values to variables in T then $\pi\rho$ assigns value to variables in $S \cup T$ according to either π or ρ respectively).

Let R_t denote the set of all restrictions of t variables, where $t \geq n/2$. Note that $|R_t| = \binom{n}{t}2^t$. Denote by B the set of *bad restrictions*— those $\rho \in R_t$ for which $f|_\rho$ is *not* expressible as an s-CNF. To prove the lemma we need to show that B is small, which we do by showing a one-to-one mapping from it to the cartesian product of the set R_{t+s} of restrictions to $(t+s)$ variables and $\{0,1\}^\ell$ for some $\ell = O(s \log k)$. This cartesian product has size $\binom{n}{t+s}2^{t+s}2^{O(s\log k)} = \binom{n}{t+s}2^t k^{O(s)}$. Hence the probability $|B|/|R_t|$ of picking a bad restriction is bounded by

$$\frac{\binom{n}{t+s}2^t k^{O(s)}}{\binom{n}{t}2^t} = \frac{\binom{n}{t+s}k^{O(s)}}{\binom{n}{t}}. \tag{2}$$

Intuitively, this ratio is small because k, s are thought of as constants and hence for t that is very close to n, it holds that $\binom{n}{t+s} \approx \binom{n}{t}/n^s$ and $n^s \gg k^{O(s)}$, meaning that (2) is bounded by $n^{-\Omega(s)}$. Formally, we can prove the bound (1) using the approximation $\binom{n}{a} \approx (ne/a)^a$; we leave it as Exercise 14.4.

Thus to prove the lemma it suffices to describe the one-to-one mapping mentioned above. Let us reason about a restriction ρ that is bad for the k-DNF formula f. None of the terms of f becomes 1 under ρ, as otherwise $f|_\rho$ would be the constant function 1. Some terms become 0, but not all, since that would also fix the overall output. In fact, since $f|_\rho$ is not expressible as an s-CNF, it has some max-term, say π, of length at least s. That is, π is a restriction of some variables not set by ρ such that $f|_{\rho\pi}$ is the sero function but $f|_{\rho\pi'}$ is non-zero for every subrestriction π' of π. The rough intuition is that the one-to-one mapping will map ρ to $\rho\sigma$, where σ is a suitably defined restriction of π's variables, and hence $\rho\sigma$ restricts at least $t+s$ variables.

Order the terms of f in some arbitrary order $t_1, t_2, \ldots,$, and within the terms, order the variables in some arbitrary order. By definition, $\rho\pi$ is a restriction that sets f to 0, and thus sets all terms of f to 0. We split π into $m \leq s$ subrestrictions $\pi_1, \pi_2, \ldots \pi_m$ as follows. Assume we have already found $\pi_1, \pi_2, \ldots, \pi_{i-1}$ such that $\pi_1\pi_2\cdots\pi_{i-1} \neq \pi$. Let t_{l_i} be the *first* term in our ordering of terms that is not set to 0 under $\rho\pi_1\pi_2\cdots\pi_{i-1}$. Such a term must exist since π is a max-term, and $\pi_1\pi_2\cdots\pi_{i-1}$ being a proper subset of it cannot be a max-term. Let Y_i be the variables of t_{l_i} that are set by π but are not set by $\rho\pi_1\cdots\pi_{i-1}$. Since π sets the term t_{l_i} to 0, Y_i cannot be empty. We define π_i to be the restriction that coincides with π on Y_i (and hence sets t_{l_i} to 0). We define σ_i to be the restriction of Y_i that keeps t_{l_i} from being 0 (such a restriction must exist since otherwise t_{l_i} would be 0). This process continues until the first time where defining π_m as above would mean that $\pi_1, \ldots, \pi_m$ (and hence also $\sigma_1, \sigma_2, \ldots, \sigma_m$) together assign at least s variables. If necessary, we trim π_m in some arbitrary way (i.e., make it assign values to fewer variables) so that these restrictions together assign exactly s variables.

Our mapping will map ρ to $(\rho\sigma_1\sigma_2\ldots\sigma_m, c)$ where c is an $O(s\log k)$ length string defined below. To show that the mapping is one-to-one we need to show how to invert it uniquely. This is harder than it looks since *a priori* there is no way to identify ρ from $\rho\sigma_1\sigma_2\ldots\sigma_m$. Indeed the purpose of the auxiliary information in c is to help us do precisely that, as described next.

Suppose we are given the assignment $\rho\sigma_1\sigma_2\ldots\sigma_m$. We can plug this assignment into f and then infer which term serves as t_{l_1}: it is the first one to that is not fixed to 0. (It is the first term not fixed to 0 by ρ, this property is maintained by σ_1, and $\sigma_2, \ldots, \sigma_m$ do not assign values to variables in the term t_{l_1}.) Let s_1 be the number of variables in π_1. The string c will contain the number s_1, the indices in t_{l_1} of the variables in π_1, and the values that π_1 assigns to these variables. Note once we know t_{l_1} this information is sufficient to reconstruct

π_1. Moreover, since each term has at most k variables, we only need $O(s_1 \log k)$ bits to store this information. Having reconstructed π_1, we can change the restriction $\rho\sigma_1\sigma_2\cdots\sigma_m$ to $\rho\pi_1\sigma_2\cdots\sigma_m$ and work out which term is t_2: it is once again the first non-zero term under the new restriction (note that π_1 sets t_{l_1} to 0). The next $O(s_2 \log k)$ bits of c will give us the assignment π_2 (where s_2 is the number of variables in π_2). We continue this process until we have processed all m terms and figured out what $\pi_1, \ldots, \pi_m$ are. Now we can "undo" them to reconstruct ρ, thus demonstrating that the mapping is one-to-one. The total length of auxiliary information required is $O((s_1 + s_2 + \ldots + s_m) \log k) = O(s \log k)$. ■

14.2 Circuits With "Counters":ACC

After the $\mathbf{AC}^0$ lower bounds of the previous section were proved, researchers were inspired to extend them to more general classes of circuits. The simplest extension seemed to be to allow gates other than $\vee$ and $\wedge$ in the circuit, while continuing to insist that the depth stays $O(1)$. A simple example of such a gate is the parity gate, which computes the parity of its input bits. Clearly, an $\mathbf{AC}^0$ circuit provided with even a single parity gate can compute the parity function. But are there still other explicit functions that it cannot compute? Razborov proved the first lower bound for such circuits using his *Method of Approximations*. Smolensky later extended this work and clarified this method for the circuit class considered here.

Definition 14.3 *(**ACC** 0)* For any integer m, the MOD_m gate outputs 0 if the sum of its inputs is 0 modulo m, and 1 otherwise.

For integers $m_1, m_2, \ldots, m_k > 1$ we say a language L is in $\mathbf{ACC}0(m_1, m_2, \ldots, m_k)$ if there exists a circuit family $\{C_n\}$ with constant depth and polynomial size (and unbounded fan-in) consisting of $\wedge$, $\vee$, $\neg$ and $MOD_{m_1}, \ldots, MOD_{m_k}$ gates accepting L.

The class $\mathbf{ACC}0$ contains every language that is in $\mathbf{ACC}0(m_1, m_2, \ldots, m_k)$ for some $k \geq 0$ and $m_1, m_2, \ldots, m_k > 1$. ♢

Good lower bounds are known only when the circuit has one kind of modular gate.

Theorem 14.4 *(Razborov-Smolensky* [Raz87, Smo87]*)*
For distinct primes p and q, the function MOD_p is not in $\mathbf{ACC}0(q)$.

We exhibit the main idea of this result by proving that the parity function cannot be computed by an $\mathbf{ACC}0(3)$ circuit.

PROOF: The proof proceeds in two steps.

Step 1. In the first step, we show (using induction on h) that for any depth h MOD_3 circuit on n inputs and size S, there is a polynomial of degree $(2l)^h$ which agrees with the circuit on $1 - S/2^l$ fraction of the inputs. If our circuit C has depth d then we set $2l = n^{1/2d}$ to obtain a degree $\sqrt{n}$ polynomial that agrees with C on $1 - S/2^{n^{1/2d}/2}$ fraction of inputs.

Step 2 We show that no polynomial of degree $\sqrt{n}$ agrees with MOD_2 on more than $49/50$ fraction of inputs.

Together, the two steps imply that $S > 2^{n^{1/2d}/2}/50$ for any depth d circuit computing MOD_2, thus proving the theorem. Now we give details.

Step 1. Consider a node v in the circuit at a depth h . (If v is an input node then we say it has depth 0.) If $g(x_1, \cdots, x_n)$ is the function computed at the node v, then we desire a polynomial $\tilde{g}(x_1, \cdots, x_n)$ over GF(3) with degree $(2l)^h$, such that $g(x_1, \ldots, x_n) = \tilde{g}(x_1, \ldots, x_n)$ for "most" $x_1, \ldots, x_n \in \{0, 1\}$. We will also ensure that on every input in $\{0, 1\}^n \subseteq GF(3)$, polynomial $\tilde{g}$ takes a value in $\{0, 1\}$. This is without loss of generality since we can just square the polynomial. (Recall that the elements of $GF(3)$ are $0, -1, 1$ and $0^2 = 0$, $1^2 = 1$ and $(-1)^2 = 1$.)

We construct the approximating polynomial by induction. When $h = 0$ the "gate" is an input wire x_i, which is exactly represented by the degree 1 polynomial x_i. Suppose we have constructed approximators for all nodes up to height $h - 1$ and g is a gate at height h.

1. If g is a NOT gate, then $g = \neg f_1$ for some other gate f_1 that is at height $h-1$ or less. The inductive hypothesis gives an approximator $\tilde{f}_1$ for f_1. Then we use $\tilde{g} = 1 - \tilde{f}_1$ as the approximator polynomial for g; this has the same degree as $\tilde{f}_1$. Whenever $\tilde{f}_1(x) = f_1(x)$ then $\tilde{g}(x) = g(x)$, so we introduced no new error.

2. If g is a MOD_3 gate with inputs $f_1, f_2, \ldots, f_k$, we use the approximation $\tilde{g} = (\sum_{i=0}^{k} \tilde{f}_i)^2$. The degree increases to at most $2 \times (2l)^{h-1} < (2l)^h$. Since $0^2 = 0$ and $(-1)^2 = 1$, we introduced no new error.

3. If g is an AND or an OR gate, we need to be more careful. We give the solution for OR; De Morgan's law allows AND gates to be handled similarly. Suppose $g = \vee_{i=0}^{k} f_i$. The naive approach would be to replace g with the polynomial $1 - \prod_{i=0}^{k}(1 - \tilde{f}_i)$. Unfortunately, this multiplies the degree by k, the fan-in of the gate, which could greatly exceed $2l$. The correct solution involves introducing some error.

   If $g = \vee_{i=0}^{k} f_i$, then on input x, $g(x) = 1$ if and only if at least one of the f_i's outputs 1 on x. Furthermore, by the *random subsum principle* (see Claim A.31) if there is some i such that $f_i(x) = 1$, then the sum (over GF(3)) of a random subset of $\{f_i(x)\}$ is nonzero with probability at least $1/2$.

   Randomly pick l subsets $T_1, \cdots, T_l$ of $\{1, \ldots, k\}$. Compute the l polynomials $(\sum_{j \in T_1} \tilde{f}_j)^2, \ldots, (\sum_{j \in T_l} \tilde{f}_j)^2$, each of which has degree at most twice than that of the largest input polynomial. Compute the OR of these l terms using the naive approach. We get a polynomial of degree at most $2l \times (2l)^{h-1} = (2l)^h$. For any x, the probability over the choice of subsets that this polynomial differs from $OR(\tilde{f}_1, \ldots, \tilde{f}_k)$ is at most $\frac{1}{2^l}$. So, by the probabilistic method, there *exists* a choice for the l subsets such that the probability over the choice of x that this polynomial differs from $OR(\tilde{f}_1, \cdots, \tilde{f}_k)$ is at most $\frac{1}{2^l}$. We use this choice of the subsets to construct the approximator.

Applying the above procedure for each gate gives an approximator for the output gate of degree $(2l)^d$ where d is depth of the entire circuit. Each operation of replacing a gate by its approximator polynomial introduces error on at most $1/2^l$ fraction of all inputs, so the overall fraction of erroneous inputs for the approximator is at most $S/2^l$. (Note that errors at different gates may affect each other. Error introduced at one gate may be canceled out by errors at another gate higher up. Thus, we are being pessimistic in applying the union bound to *upper bound* the probability that any of the approximator polynomials anywhere in the circuit miscomputes.)

Step 2. Suppose that a polynomial f agrees with the MOD_2 function for all inputs in a set $G' \subseteq \{0, 1\}^n$. If the degree of f is bounded by $\sqrt{n}$, then we show that $|G'| < (\frac{49}{50})2^n$.

Consider the change of variables $y_i = 1 + x_i \pmod 3$. (Thus $0 \to 1$ and $1 \to -1$.) This changes the input domain from $\{0, 1\}$ to $\{\pm 1\}^n$. Under this transformation f is some other polynomial, say say $g(y_1, y_2, \ldots, y_n)$, which still has degree $\sqrt{n}$. The set G' is transformed to a subset G of $\{\pm 1\}^n$ of the same size on which g and the (transformed version) of MOD_2 agree.

But it's not hard to see that MOD_2 is transformed to the function $\prod_{i=1}^{n} y_i$. Thus $g(y_1, y_2, \ldots, y_n)$, a degree $\sqrt{n}$ polynomial, agrees with $\Pi_{i=1}^{n} y_i$ on G. This is decidedly odd, and we show that any such G must be small. Specifically, let F_G be the set of all functions

$S : G \to \{0, 1, -1\}$. Clearly, $|F_G| = 3^{|G|}$, and we will show $|F_G| \leq 3^{\left(\frac{49}{50}\right)2^n}$, whence Step 2 follows.

Lemma 14.5 *For every $S \in F_G$, there exists a polynomial g_S which is a sum of monomials $a_I \prod_{i \in I} y_i$ where $|I| \leq \frac{n}{2} + \sqrt{n}$ such that $g_S(x) = S(x)$ for all $x \in G$.* ♢

PROOF: Let $\hat{S} : GF(3)^n \to GF(3)$ be any function which agrees with S on G. Then $\hat{S}$ can be written as a polynomial in the variables y_i. However, we are only interested in its values on $(y_1, y_2, \ldots, y_n) \in \{-1, 1\}^n$, when $y_i^2 = 1$ and so every monomial $\Pi_{i \in I} y_i^{r_i}$ has, without loss of generality, $r_i \leq 1$. Thus $\hat{S}$ is a polynomial of degree at most n. Now consider any of its monomial terms $\Pi_{i \in I} y_i$ of degree $|I| > n/2$. We can rewrite it as

$$\Pi_{i \in I} y_i = \Pi_{i=1}^n y_i \Pi_{i \in \bar{I}} y_i, \tag{3}$$

which takes the same values as $g(y_1, y_2, \ldots, y_n)\Pi_{i \in \bar{I}} y_i$ over G. Thus every monomial in $\hat{S}$ can be replaced with a monomial with degree at most $\frac{n}{2} + \sqrt{n}$. ■

To conclude, we bound the number of possible polynomials g_S as in Lemma 14.5. This number is at most 3 to the power of the number of possible monomials. But the number of possible such monomials is

$$\left|\left\{I \subset [n] : |I| \leq n/2 + \sqrt{n}\right\}\right| = \sum_{i=0}^{n/2+\sqrt{n}} \binom{n}{i}.$$

Using bounds on the tail of the binomial distribution (or direct calculation) it can be shown that this is less than $\frac{49}{50} 2^n$. ■

14.3 Lower bounds for monotone circuits

A Boolean circuit is *monotone* if it contains only AND and OR gates, and no NOT gates. Such a circuit can only compute monotone functions, defined as follows.

Definition 14.6 For $x, y \in \{0, 1\}^n$, we denote $x \preccurlyeq y$ if every bit that is 1 in x is also 1 in y. A function $f : \{0, 1\}^n \to \{0, 1\}$ is *monotone* if $f(x) \leq f(y)$ for every $x \preccurlyeq y$. ♢

An alternative characterization is that f is *monotone* if for every input x, changing a bit in x from 0 to 1 cannot change the value of the function from 1 to 0.

It is easy to check that every monotone circuit computes a monotone function, and every monotone function can be computed by a (sufficiently large) monotone circuit. CLIQUE is a monotone function since adding an edge to the graph cannot destroy any clique that existed in it. It is therefore natural to try to show that CLIQUE cannot be computed by polynomial-size monotone circuits. Razborov was first to prove such a result. This was soon improved by Andreev and further improved by Alon and Boppana, who proved the following theorem.

Theorem 14.7 *(Monotone-circuit lower bound for* CLIQUE [Raz85a, And85, AB87]*)*
Denote by $\mathsf{CLIQUE}_{k,n} : \{0, 1\}^{\binom{n}{2}} \to \{0, 1\}$ be the function that on input an adjacency matrix of an n-vertex graph G outputs 1 iff G contains a k-vertex clique.
There exists some constant $\epsilon > 0$ such that for every $k \leq n^{1/4}$, there's no monotone circuit of size less than $2^{\epsilon\sqrt{k}}$ that computes $\mathsf{CLIQUE}_{k,n}$.

Of course, we believe that the above theorem holds (at least roughly) for *non monotone* circuits as well (i.e., that $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$). In fact, one of the original hopes behind considering monotone circuits was that there is some connection between monotone and nonmonotone

circuit complexity. One plausible conjecture was that for *every* monotone function f, the monotone circuit complexity of f is polynomially related to its general (non-monotone) circuit complexity. Alas, this conjecture was refuted by Razborov ([Raz85b]), and in fact the gap between the two complexities is now known to be exponential [Tar88].

14.3.1 Proving Theorem 14.7

Clique indicators

To get some intuition as to why this theorem might be true, let's show that $\mathsf{CLIQUE}_{k,n}$ cannot be computed (or even approximated) by subexponential monotone circuits of a very special form. For every $S \subseteq [n]$, let C_S denote the function on $\{0,1\}^{\binom{n}{2}}$ that outputs 1 on a graph G iff the set S is a clique in G. We call C_S the *clique indicator* of S. Note that $\mathsf{CLIQUE}_{k,n} = \bigvee_{S\subseteq[n],|S|=k} \mathsf{C}_S$. We'll now prove that $\mathsf{CLIQUE}_{k,n}$ can't be computed by an OR of less than $n^{\sqrt{k}/20}$ clique indicators.

Let $\mathcal{Y}$ be the following distribution on n-vertex graphs: choose a set $K \subseteq [n]$ with $|K| = k$ at random, and output the graph that has a clique on K and no other edges. Let $\mathcal{N}$ be the following distribution on n-vertex graphs: choose a function $c : [n] \to [k-1]$ at random, and place an edge between u and v iff $c(u) \neq c(v)$. With probability one, $\mathsf{CLIQUE}_{n,k}(\mathcal{Y}) = 1$ and $\mathsf{CLIQUE}_{n,k}(\mathcal{N}) = 0$. The fact that $\mathsf{CLIQUE}_{n,k}$ requires an OR of at least $n^{\sqrt{k}/20}$ clique indicators follows immediately from the following lemma:

Lemma 14.8 *Let n be sufficiently large, $k \leq n^{1/4}$ and $S \subseteq [n]$. Then either* $\Pr[\mathsf{C}_S(\mathcal{N}) = 1] \geq 0.99$ *or* $\Pr[\mathsf{C}_S(\mathcal{Y}) = 1] \leq n^{-\sqrt{k}/20}$ ♢

Proof: Let $\ell = \sqrt{k-1}/10$. If $|S| \leq \ell$ then by the birthday bound (see Example A.4), we expect a random $f : S \to [k-1]$ to have less than 0.01 collisions and hence by Markov's inequality the probability that f is one to one is at least 0.99. This implies that $\Pr[\mathsf{C}_S(\mathcal{N}) = 1] \geq 0.99$.

If $|S| > \ell$ then $\Pr[\mathsf{C}_S(\mathcal{Y}) = 1]$ is equal to the probability that $S \subseteq K$ for a random $K \subseteq [n]$ of size k. This probability is equal to $\binom{n-\ell}{k-\ell}/\binom{n}{k}$ which, by the formula for the binomial coefficients, is less than $\left(\frac{2k}{n}\right)^{\ell} \leq n^{-0.7\ell} < n^{-\sqrt{k}/20}$ (for sufficiently large n). ■

Approximation by clique indicators.

Together with Lemma 14.8, the following lemma implies Theorem 14.7:

Lemma 14.9 *Let C be a monotone circuit of size $s < 2^{\sqrt{k}/2}$. Then, there exist sets $S_1, \ldots, S_m$ with $m \leq n^{\sqrt{k}/20}$ such that*

$$\Pr_{G\in_R\mathcal{Y}}[\bigvee_i \mathsf{C}_{S_i}(G) \geq C(G)] > 0.9 \tag{4}$$

$$\Pr_{G\in_R\mathcal{N}}[\bigvee_i \mathsf{C}_{S_i}(G) \leq C(G)] > 0.9 \tag{5}$$

$$\tag{6}$$

♢

Proof: Set $\ell = \sqrt{k}/10$, $p = 10\sqrt{k}\log n$ and $m = (p-1)^{\ell}\ell!$. Note that $m \ll n^{\sqrt{k}/20}$. We can think of the circuit C as a sequence of s monotone functions $f_1, \ldots, f_s$ from $\{0,1\}^{\binom{n}{2}}$ to $\{0,1\}$ where each function f_k is either the AND or OR of two functions $f_{k'}, f_{k''}$ for $k', k'' < k$ or is the value of an input variable $x_{u,v}$ for $u, v \in [n]$ (i.e., $f_k = \mathsf{C}_{\{u,v\}}$). The function that C computes is f_s. We'll show a sequence of functions $\tilde{f}_1, \ldots, \tilde{f}_s$ such that each function $\tilde{f}_k$ is **(1)** an OR of at most m clique indicators $\mathsf{C}_{S_1}, \ldots, \mathsf{C}_{S_m}$ with $|S_i| \leq \ell$ and **(2)** $\tilde{f}_k$ *approximates* f_k, in the sense that the two agree with good probabiity on inputs

drawn from the distributions $\mathcal{Y}$ and $\mathcal{N}$. (Thus conditions (4) and (5) are a special case of this notion of approximation.) We call any function $\tilde{f}_k$ satisfying **(1)** an (ℓ, m)-*function*.

We construct the functions $\tilde{f}_1, \ldots, \tilde{f}_s$ by induction. For $1 \leq k \leq s$, if f_k is an input variable then we let $\tilde{f}_k = f_k$. If $f_k = f_{k'} \vee f_{k''}$ then we let $\tilde{f}_{k'} \sqcup \tilde{f}_{k''}$ and if $f_k = f_{k'} \wedge f_{k''}$ then we let $\tilde{f}_{k'} \sqcap \tilde{f}_{k''}$, where the operations $\sqcup, \sqcap$ will be defined below. We'll prove that for every $f, g : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ **(a)** if f and g are (m,ℓ)-functions then so is $f \sqcup g$ (resp. $f \sqcap g$) and **(b)** $\Pr_{G \in_R \mathcal{Y}}[f \sqcup g\,(G) < f \vee g\,(G)] < 1/(10s)$ (resp. $\Pr_{G \in_R \mathcal{Y}}[f \sqcap g\,(G) < f \wedge g\,(G)] < 1/(10s)$) and $\Pr_{G \in_R \mathcal{N}}[f \sqcup g\,(G) > f \vee g\,(G)] < 1/(10s)$ (resp. $\Pr_{G \in_R \mathcal{Y}}[f \sqcap g\,(G) < f \wedge g\,(G)] < 1/(10s)$). The lemma will then follow by showing using the union bound that with probability ≥ 0.9 the equations of Condition **(b)** hold for all $\tilde{f}_1, \ldots, \tilde{f}_s$. We'll now describe the two operations $\sqcup, \sqcap$. Condition **(a)** will follow from the definition of the operations, while Condition **(b)** will require a proof.

The operation $f \sqcup g$. Let f, g be two (m,ℓ)-functions: that is $f = \bigvee_{i=1}^{m} \mathsf{C}_{S_i}$ and $g = \bigvee_{j=1}^{m} \mathsf{C}_{T_j}$ (if f or g is the OR of less than m clique indicators we can add duplicate sets to make the number m). Consider the function $h = \mathsf{C}_{Z_1} \vee \cdots \vee \mathsf{C}_{Z_{2m}}$ where $Z_i = S_i$ and $Z_{m+j} = T_j$ for $1 \leq i, j \leq m$. The function h is not an (m,ℓ)-function since it is the OR of $2m$ clique indicators. We make it into an (m,ℓ)-function in the following way: as long as there are more than m distinct sets, find p subsets $Z_{i_1}, \ldots, Z_{i_p}$ that are in a *sunflower* formation. That is, there exists a set $Z \subseteq [n]$ such that for every $1 \leq j, j' \leq p$, $Z_{i_j} \cap Z_{i_{j'}} = Z$. (The name "sunflower" comes from viewing the sets $Z_{i_1} \setminus Z$,...,$Z_{i_p} \setminus Z$ as the petals of a sunflower with center Z.) Replace the functions $\mathsf{C}_{Z_{i_1}}, \ldots, \mathsf{C}_{Z_{i_p}}$ in the function h with the function C_Z. Once we obtain an (m,ℓ)-function h' we define $f \sqcup g$ to be h'. We won't get stuck because of the following lemma (whose proof we defer):

Lemma 14.10 *(Sunflower lemma* [ER60]*) Let $\mathcal{Z}$ be a collection of distinct sets each of cardinality at most ℓ. If $|\mathcal{Z}| > (p-1)^\ell \ell!$ then there exist p sets $Z_1, \ldots, Z_p \in \mathcal{Z}$ and a set Z such that $Z_i \cap Z_j = Z$ for every $1 \leq i, j \leq p$.* ♢

The operation $f \sqcap g$. Let f, g be two (m,ℓ)-functions: that is $f = \bigvee_{i=1}^{m} \mathsf{C}_{S_i}$ and $g = \bigvee_{j=1}^{m} \mathsf{C}_{T_j}$. Let h be the function $\bigvee_{1 \leq i,j \leq m} \mathsf{C}_{S_i \cup T_j}$. We perform the following steps on h: **(1)** Discard any function C_Z for $|Z| > \ell$. **(2)** Reduce the number of functions to m by applying the sunflower lemma as above.

Proving Condition (b). To complete the proof of the lemma, we prove the following four equations:

- $\Pr_{G \in_R \mathcal{Y}}[f \sqcup g\,(G) < f \vee g\,(G)] < 1/(10s)$.

  If $Z \subseteq Z_1, \ldots, Z_p$ then for every i, $\mathsf{C}_{Z_i}(G)$ implies that $\mathsf{C}_Z(G)$ and hence the operation $f \sqcup g$ can't introduce any "false negatives".

- $\Pr_{G \in_R \mathcal{N}}[f \sqcup g\,(G) > f \vee g\,(G)] < 1/(10s)$.

  We can introduce a "false positive" on a graph G only if when we replace the clique indicators for a sunflower $Z_1, \ldots, Z_p$ with the clique indicator for the common intersection Z, it is the case that $\mathsf{C}_Z(G)$ holds even though $\mathsf{C}_{Z_i}(G)$ is false for every i. Recall that we choose $G \in_R \mathcal{N}$ by choosing a random function $c : [n] \to [k-1]$ and adding an edge for every two vertices u, v with $c(u) \neq c(v)$. Thus, we get a false positive if c is one-to-one on Z (we denote this event by B) but *not* one-to-one on Z_i for every $1 \leq i \leq p$ (we denote these events by $A_1, \ldots, A_p$). We'll show that the intersection of B and $A_1, \ldots, A_p$ happens with probability at most 2^{-p} which (by the choice of p) is less than $1/(10m^2 s)$. Since we apply the reduction step at most m times the equation will follow.

  For every i, $\Pr[A_i|B] < 1/2$. Indeed, since $|Z_i| = \ell < \sqrt{k-1}/10$, the birthday bound says that $\Pr[A_i] < 1/2$ and conditioning on having no collisions in Z only makes this event less likely. Conditioned on B, the events $A_1, \ldots, A_p$ are independent, since they depend on the values of c on disjoint sets, and hence $\Pr[A_1 \wedge \cdots \wedge A_p \wedge B] \leq \Pr[A_1 \wedge \cdots \wedge A_p|B] = \prod_{i=1}^{p} \Pr[A_p|B] \leq 2^{-p}$.

- $\Pr_{G \in_R \mathcal{Y}}[f \sqcap g\,(G) < f \wedge g\,(G)] < 1/(10s)$.

  By the distributive law $f \wedge g = \bigvee_{i,j}(\mathsf{C}_{S_i} \wedge \mathsf{C}_{T_j})$. A graph G in the support of $\mathcal{Y}$ consists of a clique over some set K. For such a graph $\mathsf{C}_{S_i} \wedge \mathsf{C}_{T_j}$ holds iff $S_i, T_j \subseteq K$ and thus $\mathsf{C}_{S_i} \wedge \mathsf{C}_{T_j}$ holds iff $\mathsf{C}_{S_i \cup T_j}$ holds. We can introduce a false negative when we discard functions of the form C_Z for $|Z| > \ell$, but by Lemma 14.8, for such sets Z, $\Pr[\mathsf{C}_Z(\mathcal{Y}) = 1] < n^{-\sqrt{k}/20} < 1/(10sm^2)$. The equation follows since we discard at most m^2 such sets.

- $\Pr_{G \in_R \mathcal{N}}[f \sqcap g\,(G) > f \wedge g\,(G)] < 1/(10s)$.

  Since $\mathsf{C}_{S \cup T}$ implies both C_S and C_T, we can't introduce false positives by moving from $f \wedge g$ to $\bigvee_{i,j} \mathsf{C}_{S_i \cup T_j}$. We can't introduce false positives by discarding functions from the OR. Thus, the only place where we can introduce false positives is where we replace the clique indicators of a sunflower with the clique indicator of the common intersection. We bound this probability in the same way as this was done for the $\sqcup$ operator.

■

Proof of the sunflower lemma (Lemma 14.10). The proof is by induction on ℓ. The case $\ell = 1$ is trivial since distinct sets of size 1 must be disjoint (hence forming a sunflower with center $Z = \emptyset$). For $\ell > 1$ let $\mathcal{M}$ be a maximal subcollection of $\mathcal{Z}$ containing only disjoint sets. We can assume that $|\mathcal{M}| < p$ since otherwise $\mathcal{M}$ is already a sufficiently large sunflower. Because of $\mathcal{M}$'s maximality for every $Z \in \mathcal{Z}$ there exists $x \in \cup\mathcal{M} = \cup_{M \in \mathcal{M}} M$ such that $x \in Z$. Since $|\cup \mathcal{M}| \leq (p-1)\ell$, by averaging there's an $x \in \cup\mathcal{M}$ that appears in at least a $\frac{1}{\ell(p-1)}$ fraction of the sets in $\mathcal{Z}$. Let $Z_1, \ldots, Z_t$ be the sets containing x, and note that $t > (p-1)^{\ell-1}(\ell-1)!$. Thus, by induction there are p sets among the $\ell - 1$-sized sets $Z_1 \setminus \{x\}, \cdots, Z_t \setminus \{x\}$ that form a sunflower, adding back x we get the desired sunflower among the original sets. Note that the statement (and proof) assume nothing about the size of the universe the sets in $\mathcal{Z}$ live in. ■

14.4 Circuit complexity: The frontier

Now we sketch the "frontier" of circuit lower bounds, namely, the dividing line between what we can prove and what we cannot. Along the way we also define multi-party communication, since it may prove useful for proving some new circuit lower bounds.

14.4.1 Circuit lower bounds using diagonalization

We already mentioned that the best lower bound on circuit size for an **NP** problem is $5n - o(n)$. For **PH** better lower bounds are known: Exercises 6.5–6.6 of Chapter 6 asked you to show using diagonalization that for every $k > 0$, some language in **PH** (in fact in $\mathbf{\Sigma}_2^p$) requires circuits of size $\Omega(n^k)$. One imagines that classes "higher up" than **PH** should have even harder languages. Thus a natural open question is:

Frontier 1: Does **NEXP** have languages that require super-polynomial size circuits?

If we go a little above **NEXP**, we can actually prove a super-polynomial lower bound: we know that $\mathbf{MA_{EXP}} \not\subseteq \mathbf{P_{/poly}}$ where $\mathbf{MA_{EXP}}$ is the set of languages accepted by a one round proof system with an all powerful prover and an exponential time *probabilistic* verifier. (This is the exponential time analog of the class **MA** defined in Section 8.2.) This follows from the fact that if $\mathbf{MA_{EXP}} \subseteq \mathbf{P_{/poly}}$ then in particular $\mathbf{PSPACE} \subseteq \mathbf{P_{/poly}}$. However, by $\mathbf{IP} = \mathbf{PSPACE}$ (Theorem 8.19) in this case $\mathbf{PSPACE} = \mathbf{MA}$ (the prover can send in

one round the circuit for computing the prover strategy in the interactive proof). By simple padding this implies that $\mathbf{MA_{EXP}}$ equals the class of languages in *exponential space*, which can be directly shown to not contain $\mathbf{P_{/poly}}$ using diagonalization. Interestingly, this lower bound does not relativize— there is an oracle under which $\mathbf{MA_{NEXP}} \subseteq \mathbf{P_{/poly}}$ [BFT98]. (The result that $\mathbf{IP} = \mathbf{PSPACE}$ used in the proof also does not relativize.)

14.4.2 Status of ACC versus P

The result that PARITY is not in $\mathbf{AC}0$ separates $\mathbf{NC}1$ from $\mathbf{AC}0$. The next logical step would be to separate $\mathbf{ACC}0$ from $\mathbf{NC}1$. Less ambitiously, we would like to show even a function in $\mathbf{P}$ or $\mathbf{NP}$ that is not in $\mathbf{ACC}0$.

The Razborov-Smolenksy method seems to fail when we allow the circuit even two types of modular gates, say MOD_2 and MOD_3. In fact if we allow the bounded depth circuit modular gates that do arithmetic mod q, when q is not a prime —a prime power, to be exact— we reach the limits of our knowledge. (The exercises ask you to figure out why the proof of Theorem 14.4 does not seem to apply when the modulus is a composite number.) To give one example, it is consistent with current knowledge that the CLIQUE function can be computed by linear size circuits of constant depth consisting entirely of MOD_6 gates. The problem seems to be that low-degree polynomials modulo m where m is composite are surprisingly expressive [BBR92].

Frontier 2: Show CLIQUE is not in $\mathbf{ACC}0(6)$.

Or even less ambitiously:

Frontier 2.1: Exhibit a language in $\mathbf{NEXP}$ that is not in $\mathbf{ACC}0(6)$.

It is worth noting that thus far we are talking about *non-uniform* circuits (to which Theorem 14.4 also applies). Stronger lower bounds are known for uniform circuits: Allender and Gore [AG94] have shown that a decision version of the Permanent (and hence the Permanent itself) requires exponential size "Dlogtime-uniform" $\mathbf{ACC}0$ circuits. (A circuit family $\{C_n\}$ is *Dlogtime uniform* if there exists a deterministic Turing machine M that given a number n and a pair of gates g, h determines in $O(\log n)$ time what types of gates g and h are and whether g is h's parent in C_n.)

But going back to non-uniform $\mathbf{ACC}0$, we wish to mention an alternative representation of $\mathbf{ACC}0$ circuits that may be useful in further lower bounds. A *symmetric* gate is a gate whose output depends only on the number of inputs that are 1. For example, majority and mod gates are symmetric. Yao has shown that $\mathbf{ACC}0$ circuits can be simplified to give an equivalent depth 2 circuits with a symmetric gate at the output (Figure 14.3). Beigel and Tarui subsequently improved Yao's result:

Theorem 14.11 *([Yao90, BT91]) If $f \in \mathbf{ACC}0$, then f can be computed by a depth 2 circuit C with a symmetric gate with quasipolynomial (i.e., $2^{\log^k n}$) fan-in at the output level and $\wedge$ gates that has polylogarithmic fan-in at the input level.* ♢

We will revisit this theorem below in Section 14.5.1.

14.4.3 Linear Circuits With Logarithmic Depth

When we restrict circuits to have bounded fan-in we necessarily need to allow them to have nonconstant (in fact, $\Omega(\log n)$) depth to allow the output to depend on all bits of the input. With this in mind, the simplest interesting circuit class seems to be the class of bounded fan-in circuits having $O(n)$ size and $O(\log n)$ depth.

Frontier 3: Find an explicit n-bit Boolean function that cannot be computed by circuits of $O(n)$ size and $O(\log n)$ depth.

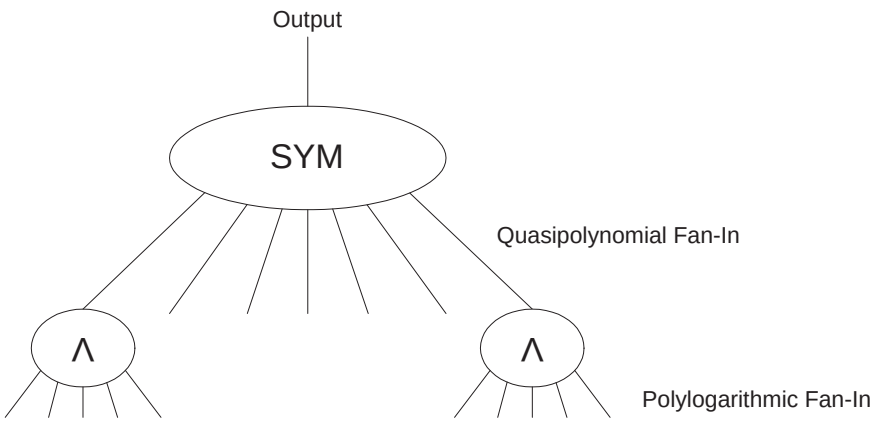


Figure 14.3 The depth 2 circuit with a symmetric output gate from Theorem 14.11.

Subfrontier: Find such a function which is not Boolean and maps $\{0,1\}^n$ to $\{0,1\}^n$.

(Note that by counting one can easily show that *some* function on n bits requires superpolynomial size circuits and hence bounded fan-in circuits with more than logarithmic depth; see the exercises in Chapter 6. Hence we want to show this for an explicit function, e.g. CLIQUE.)

Valiant thought about this problem in the '70s. His initial candidates for lower bounds boiled down to showing that a certain graph called a *superconcentrator* needed to have superlinear size. He failed to prove this and instead ended up proving that such superconcentrators do exist! However a side product of Valiant's investigations was the following important lemma concerning depth-reduction for such circuits.

Lemma 14.12 *([Val75a]) In any directed acyclic graph with m edges and depth d, there is a set S of $km/\lceil \log d \rceil$ edges whose removal leaves the graph with depth at most $d/2^{k-1}$.* $\diamondsuit$

PROOF SKETCH: Sort the graphs into d levels such that if $\overrightarrow{u\,v}$ is an edge, then u is at a lower level than v. Letting $\ell = \lceil \log d \rceil$, we can use the binary basis to represent every level as an ℓ-bit string. We label each edge $\overrightarrow{u\,v}$ with the number $i \in [\ell]$ such that i is the most significant bit in which the levels of u and v differ. We let I be the k "least popular" labels, and let S be the set of edges that are labeled with a number in I. Clearly, $|S| \leq km/\ell$. Moreover, it can be shown that every path longer than $2^{\ell-k} \leq d/2^{k-1}$ must contain more than $\ell - k$ distinct labels (and hence an edge in S). We leave completing this proof as Exercise 14.10. ■

This lemma can be applied as follows. Suppose we have an $O(n)$-size circuit C of depth $c \log n$ with n inputs $\{x_1, \ldots, x_n\}$ and n outputs $\{y_1, \ldots, y_n\}$, and suppose $2^k \sim c/\epsilon$ where $\epsilon > 0$ is arbitrarily small. One can find $O(n/\log\log n)$ edges in C whose removal results in a circuit with depth at most $\epsilon \log n$. But then, since C has *bounded* fan-in, we must have that each output y_i is connected to at most $2^{\epsilon \log n} = n^\epsilon$ inputs. So each output y_i in C is completely determined by n^ϵ inputs and the values of the removed wires. So the removed wires somehow allowed some kind of "compression" of the truth tables of $y_1, y_2, \ldots, y_n$. We do not expect this to be the case for any reasonably complex function. Surprisingly, no one has been able to exhibit an explicit function for which this is not the case.

14.4.4 Branching Programs

Just as circuits are used to investigate time requirements of Turing Machines, *branching programs* are used to as a combinatorial tool to investigate space complexity. A *branching program* on n input variables $x_1, x_2, \ldots, x_n$ is a directed acyclic graph all of whose nodes of

nonzero outdegree are labeled with a variable x_i. It has two nodes of outdegree zero that are labeled with an output value, ACCEPT or REJECT. The edges are labeled by 0 or 1. One of the nodes is designated the start node. A setting of the input variables determines a way to walk on the directed graph from the start node to an output node. At any step, if the current node has label x_i, then we take an edge going out of the node whose label agrees with the value of x_i. The branching program is *deterministic* if every nonoutput node has exactly one 0 edge and one 1 edge leaving it. Otherwise it is *nondeterministic*. The *size* of the branching program is the number of nodes in it. The branching program complexity of a language is defined analogously with circuit complexity. Sometimes one may also require the branching program to be *leveled*, whereby nodes are arranged into a sequence of levels with edges going only from one level to the next. Then the *width* is the size of the largest level.

Theorem 14.13 *If $S(n) \geq \log n$ and $L \in \mathbf{SPACE}(S(n))$ then L has branching program complexity at most $c^{S(n)}$ for some constant $c > 1$.* ♢

PROOF: Essentially mimics our proof of Theorem 4.2 that $\mathbf{SPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$. The nodes of the branching program correspond to the configurations of the space-bounded TM, and it is labeled with variable x_i if the configuration shows the TM reading the ith bit in the input. ■

A similar result holds for NDTMs and nondeterministic branching program complexity.

Frontier 4 : Describe a problem in $\mathbf{P}$ (or even $\mathbf{NP}$) that requires branching programs of size greater than $n^{1+\epsilon}$ for some constant $\epsilon > 0$.

There is some evidence that branching programs are more powerful than one may imagine. For instance, branching programs of constant width (reminiscent of a TM with $O(1)$ bits of memory) seem inherently weak. Thus the next result is unexpected.

Theorem 14.14 *(Barrington* [Bar86]*) A language has polynomial size, width 5 branching programs iff it is in* $\mathbf{NC}1$. ♢

14.5 Approaches using communication complexity

Here we outline a concrete approach (rather, a setting) in which better lower bounds may lead to a resolution of some of the questions above. It relates to generalizations of communication complexity introduced earlier. Mostly we will use *multiparty communication complexity*, (in the "number on the forehead model" defined in Section 13.3), though Section 14.5.4 will use *communication complexity of a relation*.

14.5.1 Connection to $\mathrm{ACC}0$ Circuits

Suppose $f(x_1, \ldots, x_n)$ has a depth-2 circuit with a symmetric gate with fan-in N at the output and $\wedge$ gates with fan-in $k-1$ at the input level (see Figure 2). Razborov and Wigderson [RW93] observed that in this case f's k-party communication complexity is at most $k \log N$. To see this, first partition the $\wedge$ gates amongst the players. Each bit is not known to exactly one player, so the input bits of each $\wedge$ gate are known to at least one player; assign the gate to such a player with the lowest index. Players then broadcast how many of their gates output 1. Since this number has at most $\log N$ bits, the claim follows.

Our hope is to employ this connection with communication complexity in conjunction with Theorem 14.11 to obtain lower bounds on $\mathbf{ACC}0$ circuits. For example, note that by Theorem 13.24, there is an explicit n-bit function requiring $\Omega(n/4^k)$ k-party communication complexity, and hence this function cannot by computed by a polynomial (or even quasipolynomial) depth-2 circuit as above with bottom fan-in $k - 1 < \log n/4$. However,

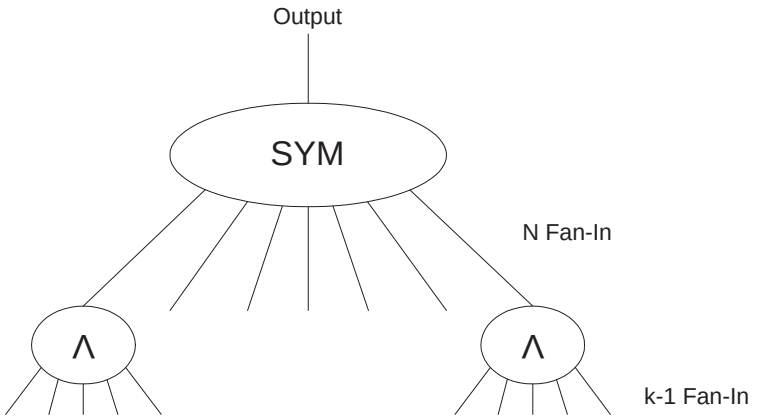


Figure 14.4 If f is computed by the above circuit, then f has a k-party protocol of complexity $k \log N$.

this is not enough to obtain a lower bound on $\mathbf{ACC}0$ circuits since we need to show that k is not polylogarithmic to employ Theorem 14.11. But a k-party communication complexity lower bound of $\Omega(n/\operatorname{poly}(k))$ for say the CLIQUE function would close Frontier 2.

14.5.2 Connection to Linear Size Logarithmic Depth Circuits

Suppose that $f : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^n$ has bounded fan-in circuits of linear size and logarithmic depth. If $f(x,j,i)$ denotes the ith bit of $f(x,j)$, then Valiant's Lemma implies that $f(x,j,i)$ has a simultaneous 3-party protocol—that is, a protocol where all parties speak only once and write simultaneously on the blackboard (i.e., non-adaptively)—where,

- (x,j) player sends $O(n/\log\log n)$ bits;
- (x,i) player sends n^ϵ bits; and
- (i,j) player sends $O(\log n)$ bits.

So, if we can show that a function does not have such a protocol, then we would have a lower bound for the function on linear size logarithmic depth circuits with bounded fan-in. For example even the simple function $f(x,j,i) = x_{j \oplus i}$, where $j \oplus i$ is the bitwise xor, not known to have such a protocol, and hence may not be computable by a bounded fan-in circuit of linear size and logarithmic depth.

14.5.3 Connection to branching programs

The notion of multiparty communication complexity (at least the "number on the forehead" model discussed here) was invented by Chandra, Furst and Lipton [CFL83] for proving lower bounds on branching programs, especially the constant-width branching programs discussed in Section 14.4.4.

14.5.4 Karchmer-Wigderson communication games and depth lower bounds

The result that PARITY is not in $\mathbf{AC}0$ separates $\mathbf{NC}1$ from $\mathbf{AC}0$. The next step would be to separate $\mathbf{NC}2$ from $\mathbf{NC}1$. (Of course, ignoring for the moment the issue of separating $\mathbf{ACC}0$ from $\mathbf{NC}1$.) Karchmer and Wigderson described how communication complexity

can be used to prove lower bounds on the minimum depth required to compute a function. They showed the following result about monotone circuits, whose proof we omit:

Theorem 14.15 *([KW88]) Detecting whether a graph has a perfect matching is impossible for monotone circuits of bounded fan-in and depth $O(\log n)$* ♢

However, we do describe the basic *Karchmer-Wigderson* game used to prove the above result, since it is relevant for nonmonotone circuits as well. For a function $f:\{0,1\}^n \to \{0,1\}$ this game is defined as follows.

There are two players, **ZERO** *and* **ONE**. *Player* **ZERO** *receives an input x such that $f(x) = 0$ and Player* **ONE** *receives an input y such that $f(y) = 1$. They communicate bits to each other, until they can agree on an $i \in \{1, 2, \ldots, n\}$ such that $x_i \neq y_i$.*

The mechanism of communication is defined similarly as in Chapter 13; there is a protocol that the players agree on in advance before receiving the input. Note that the key difference from the scenario in Chapter 13 is that the final answer is not a single bit, and furthermore, the final answer is not unique (the number of acceptable answers is equal to the number of bits that x, y differ on). Sometimes this is described as *computing a relation*. The relation in this case consists of all triples (x, y, i) such that $f(x) = 0$, $f(y) = 1$ and $x_i \neq y_i$.

We define $C_{KW}(f)$ as the communication complexity of the above game; namely, the maximum over all $x \in f^{-1}(0), y \in f^{-1}(1)$ of the number of bits exchanged in computing an answer for x, y. The next theorem shows that this parameter has a suprising alternative characterization. It assumes that circuits don't have NOT gates and instead the NOT gates are pushed down to the inputs using De Morgan's law. (In other words, the inputs may be viewed as $x_1, x_2, \ldots, x_n, \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$.) Furthermore, AND and OR gates have fan-in 2. (None of these assumptions is crucial and affects the theorem only marginally.)

Theorem 14.16 *([KW88]) $C_{KW}(f)$ is exactly the minimum depth among all circuits that compute f.* ♢

Proof: First, we show that if there is a circuit $\mathcal{C}$ of depth K that computes f then $C_{KW}(f) \leq K$. Each player has a copy of $\mathcal{C}$, and evaluates this circuit on the input given to him. Of course, it evaluates to 0 for Player **ZERO** and to 1 for Player **ONE**. Suppose that the top gate is an OR. Then for Player **ONE** at least one of the two incoming wires to this gate must be 1, and so in the first round, Player **ONE** sends one bit communicating which of these wires it was. Note that this wire is 0 for Player **ZERO**. In the next round the players focus on the gate that produced the value on this wire. (If the top gate is an AND on the other hand, then in the first round Player **ZERO** speaks, conveying which of the two incoming wires was 0. This wire will be 1 for Player **ONE**.) This goes on and the players go deeper down the circuit, always maintaining the invariant that the current gate has value 1 for Player **ONE** and 0 for Player **ZERO**. Finally, after at most K steps they arrive at an input bit. According to the invariant being maintained, this bit must be 1 for Player **ONE** and 0 for Player **ZERO**. Thus they both know an index i that is a valid answer.

For the reverse direction, we have to show that if $C_{KW}(f) = K$ then there is a circuit of depth at most K that computes f. We prove a more general result. For any two disjoint nonempty subsets $A \subseteq f^{-1}(0)$ and $B \subseteq f^{-1}(1)$, let $C_{KW}(A, B)$ be the communication complexity of the Karchmer-Wigderson game when x always lies in A and y in B. We show that there is a circuit of depth $C_{KW}(A, B)$ that outputs 0 on every input from A and 1 on every input from B. Such a circuit is called a *distinguisher* for sets A, B. The proof is by induction on $K = C_{KW}(A, B)$. The base case $K = 0$ is trivial since this means that the players do not have to communicate at all to agree on an answer, say i. Hence $x_i \neq y_i$ for all $x \in A, y \in B$, which implies that either (a) $x_i = 0$ for every $x \in A$ and $y_i = 1$ for every $y \in B$ or (b) $x_i = 1$ for every $x \in A$ and $y_i = 0$ for every $y \in B$. In case (a) we can use the depth 0 circuit x_i and in case (b) we can use the circuit $\overline{x_i}$ to distinguish A, B.

For the inductive step, suppose $C_{KW}(A, B) = K$, and at the first round Player **ZERO** speaks. Then A is the disjoint union of two sets A_0, A_1 where A_b is the set of inputs in A for which Player **ZERO** sends bit b. Then $C_{KW}(A_b, B) \leq K - 1$ for each b, and the

inductive hypothesis gives a circuit C_b of depth at most $K-1$ that distinguishes A_b, B. We claim that $C_0 \wedge C_1$ distinguishes A, B (note that it has depth at most K). The reason is that $C_0(y) = C_1(y) = 1$ for every $y \in B$ whereas for every $x \in A$, $C_0(x) \wedge C_1(x) = 0$ since if $x \in A_b$ then $C_b(x) = 0$. ■

Thus we have the following frontier.

Frontier 5: Show that some function f in $\mathbf{P}$ (or even $\mathbf{NEXP}$!) has $C_{KW}(f) = \Omega(\log n \log\log n)$.

Karchmer, Raz, and Wigderson [KRW95] describe a candidate function that may work. It uses the fact that a function on k bits has a truth table of size 2^k, and that most functions on k bits are hard (e.g., require circuit size $\Omega(2^k/k)$, circuit depth $\Omega(k)$, etc.). They define the function by assuming that part of the n-bit input encodes a very hard function, and this hard function is applied to the remaining input in a "tree" fashion.

For any function $g:\{0,1\}^k \to \{0,1\}$ and $s \geq 1$ define $g^{\circ s}:\{0,1\}^{k^s} \to \{0,1\}$ as follows. If $s = 1$ then $g^{\circ s} = g$. Otherwise express the input $x \in \{0,1\}^{k^s}$ as $x_1x_2x_3\cdots x_k$ where each $x_i \in \{0,1\}^{k^{s-1}}$ and define

$$g^{\circ s}(x_1x_2\cdots x_k) = g(g^{\circ(s-1)}(x_1)g^{\circ(s-1)}(x_2)\cdots g^{\circ(s-1)}(x_k)).$$

Clearly, if g can be computed in depth d then $g^{\circ s}$ can be computed in depth sd. But, it seems hard to reduce the depth beyond that for an arbitrary choice of the function g.

Now we describe the KRW candidate function $f:\{0,1\}^n \to \{0,1\}$. Let $k = \lfloor \log \frac{n}{2} \rfloor$ and s be the largest integer such that $k^s \leq n/2$ (thus $s = \Theta(\frac{\log n}{\log\log n})$.) For any n-bit input x, let g_x be the function whose truth table is the first 2^k bits of x. Let $x|_2$ be the string of the last k^s bits of x. Then

$$f(x) = g_x^{\circ s}(x|_2).$$

According to our earlier intuition, when the first 2^k bits of x represent a really hard function —as they must for many choices of the input— then $g_x^{\circ s}(x|_2)$ should require depth $\Omega(sk) = \Omega(\frac{\log^2 n}{\log\log n})$. Of course, proving this seems difficult.

This type of complexity questions, whereby we are asking whether s instances of a problem are s times as hard as a single instance, are called *direct sum* questions. Similar questions have been studied in a variety of computational models, and sometimes counter-intuitive results have been proven for them. One example is that by a counting argument there exists an $n \times n$ matrix A over $\{0,1\}$, such that the smallest circuit computing the linear function $v \mapsto Av$ for $v \in \{0,1\}^n$ is of size $\Omega(n^2)$. However, computing this function on n independent instances $v_1, \ldots, v_n$ can be done significantly faster than n^3 steps using fast matrix multiplication [Str69] (the current record is roughly $O(n^{2.38})$ [CW90]).

Chapter notes and history

Shannon defined circuit complexity, including monotone circuit complexity, in 1949. The topic was studied in Russia since the 1950s. (See Trakhtenbrot [Tra84] for some references.) Savage [Sav72] was the first to observe the close relationship between time required to decide a language on a TM and its circuit complexity, and to suggest circuit lower bounds as a way to separate complexity classes. A burst of results in the 1980s, such as the separation of $\mathbf{P}$ from $\mathbf{AC}0$ [FSS81, Ajt83] and Razborov's separation of monotone $\mathbf{NP}$ from monotone $\mathbf{P}_{/\mathbf{poly}}$ [Raz85a] raised hopes that a resolution of $\mathbf{P}$ versus $\mathbf{NP}$ might be near. These hopes were dashed by Razborov himself [Raz89] when he showed that his method of approximations was unlikely to apply to nonmonotone circuits. Later Razborov and Rudich [RR94] formalized what they called *natural proofs* to show that all lines of attack considered up to that point were unlikely to work. (See Chapter 23.)

Our presentation in Sections 14.2 and 14.3 closely follows that in Boppana and Sipser's excellent survey of circuit complexity [BS90], which is still useful and current 15 years later. (It omits discussion of lower bounds on algebraic circuits; see [Raz04a] for a recent result.)

Håstad's switching lemma [Hås86] is a stronger form of results from [FSS81, Ajt83, Yao85]. The Razborov-Smolensky method of using approximator polynomials is from [Raz87], strengthened in [Smo87]. Valiant's observations about superlinear circuit lower bounds are from a 1975 paper [Val75b] and an unpublished manuscript—lack of progress on this basic problem gets more embarrassing by the day!.

The $5n - o(n)$ lower bound on general circuits is by Iwama and Morizumi, improving on a previous $4.5n - o(n)$ by Lachish and Raz; the full version of both results is [ILMR05].

Barrington's theorem is a good example of how researchers' intuition about circuits can sometimes be grossly incorrect. His theorem can be seen as a surprisingly simple way to compute $\mathbf{NC}1$ functions, and has proved very influential in cryptography research (e.g., see [GMW87, Kil88, AIK04]).

Exercises

14.1 Suppose that f is computable by an $\mathbf{AC}^0$ circuit C of depth d and size S. Prove that f is computable by an $\mathbf{AC}^0$ circuit C' of size $< 10S$ and depth d that does not contain NOT gates but instead has n additional inputs that are negations of the original n inputs. H462

14.2 Suppose that f is computable by an $\mathbf{AC}^0$ circuit C of depth d and size S. Prove that f is computable by an $\mathbf{AC}^0$ circuit C' of size $< (10S)^d$ and depth d where each gate has fan-out 1.

14.3 Prove that if all the max-terms of a Boolean function f are of size at most s then f is expressible as an s-CNF. H462

14.4 Prove that for $t > n/2$, $\binom{n}{t+k} \leq \binom{n}{t}\left(\frac{e(n-t)}{n}\right)^k$. Use this to complete the proof of Lemma 14.2 (Section 14.1.2). H462

14.5 Show that $\mathbf{ACC}0 \subseteq \mathbf{NC}1$.

14.6 Identify reasons why the Razborov-Smolensky method does not work when the circuit has MOD m gates, where m is a composite number.

14.7 Show that representing the OR of n variables $x_1, x_2, \ldots, x_n$ exactly with a polynomial over $GF(q)$ where q is prime requires degree exactly n.

14.8 The Karchmer-Wigderson game can be used to prove *upper bounds*, and not just lower bounds. Show using this game that PARITY and MAJORITY are in $\mathbf{NC}1$.

14.9 Show that for every constant c, if a language is computed by a polynomial-size branching program of width c then it is in $\mathbf{NC}1$.

14.10 Complete the full proof of Valiant's Lemma (Lemma 14.12). H462

Chapter 15

Proof complexity

In defining **NP** we sought to capture the phenomenon whereby if certain statements (such as "this Boolean formula is satisfiable") are true, then there is a short certificate to this effect. Furthermore, we introduced the conjecture $\mathbf{NP} \neq \mathbf{coNP}$ according to which certain types of statements (such as "this Boolean formula is *not* satisfiable") do not have short certificates in general. In this short chapter we are interested in investigating this phenomenon more carefully, especially in settings where the existence of a short certificate is not obvious.

We start in Section 15.1 with some motivating examples. In Section 15.2 we formalize the notion of a *proof system* using a very simple example, propositional proofs. We also prove exponential lower bounds for the resolution proof system using two methods that serve as simple examples of important techniques in proof complexity. Section 15.3 surveys some other proof systems that have been studied, and lower bounds known for them. Finally, Section 15.4 presents some metamathematical ruminations about whether proof complexity can shed some light on the difficulty of resolving **P** versus **NP**. There is a related, equally interesting question of *finding* short certificates assuming they exist, which we will mostly ignore except in the chapter notes.

15.1 Some examples

We start with a few examples, many of which were studied before the notion of computational complexity arose. Consider the following computational tasks:

1. **Infeasibility of systems of linear inequalities**. You are given a system

$$\begin{aligned} \langle \mathbf{a}_1, \mathbf{x} \rangle &\leq b_1 \\ \langle \mathbf{a}_2, \mathbf{x} \rangle &\leq b_2 \\ &\vdots \\ \langle \mathbf{a}_m, \mathbf{x} \rangle &\leq b_m \end{aligned}$$

   where $\mathbf{a}_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$ for every i. Prove that there is no non-negative vector $\mathbf{x} \in \mathbb{R}^n$ satisfying this system.

2. **Infeasibility of systems of linear inequalities over the integers.** The same setting as above, but with each $\mathbf{a}_i \in \mathbb{Z}^n$ and $b_i \in \mathbb{Z}$, and the solution $\mathbf{x}$ also has to be in $\mathbb{Z}^n$.

3. **Infeasibility of systems of polynomial equations.** Given a system of polynomials $g_1(x_1, x_2, \ldots, x_n), g_2(x_1, x_2, \ldots, x_n), \ldots, g_m(x_1, x_2, \ldots, x_n)$ with real coefficients, certify that the system $g_i(x_1, \ldots, x_n) = 0 \ \ \forall i = 1, 2, \ldots, m$ has no common solution.

4. **Contradictions**. Given a Boolean formula φ in n variables, certify that it has no satisfying assignment.

5. **Nontrivial words in a finitely presented group**. We are given a group over finite set $\mathcal{S}$ (meaning every group element is a *word* of the form $s_1^{\rho_1} s_2^{\rho_2} \cdots s_n^{\rho_n}$ where n is any positive integer and each ρ_i is an integer, possibly negative). The group is implicitly described by means of a finite set of *relations* of the type $s_1^{\rho_1} s_2^{\rho_2} \cdots s_n^{\rho_n} = e$ where each $s_i \in \mathcal{S}$, $\rho_i \in \mathbb{Z}$ and $e \in \mathcal{S}$ is some designated *identity* element. These relations imply that given a word w, it is nontrivial to know whether it simplifies to e by repeatedly applying the relations. If a word can be simplified to e, then this has a finite proof, namely, the sequence of relations that need to be applied during the simplification. We are interested in the problem where, given a word w, we have to certify that it is not equal to e (i.e., is nontrivial).

In each of the above examples, there seems to be no obvious short certificate. But sometimes such intuition can lead us astray. For instance, an old result called Farkas' Lemma (see Note 19.4) implies that there is indeed a short certificate for the first problem: the system is infeasible if and only if there is a combination of the inequalities that leads to a clear contradiction, in other words a $\mathbf{y} \in \mathbb{R}^m$ such that $\sum_{i=1}^n y_i \mathbf{a}_i$ is non-negative but $\sum_i y_i b_i < 0$. The "size" of such a certificate $\mathbf{y}$ is small— it can be represented using a number of bits that is polynomial in the number of bits used to represent the inputs a_i's and b_i's.

The next three problems are **coNP**-hard (and the word problem is undecidable in general and **coNP**-hard for specific groups; see chapter notes) and therefore if **NP** $\neq$ **coNP** we do not expect short proofs for them. Nevertheless, it is interesting to study the length of the shortest proof for *specific* instances of the problem. For instance, certifying unsatisfiability (or the computationally equivalent problem of certifying *tautologyhood*, see Example 2.21) is a natural problem that arises in fields such as artificial intelligence and formal verification of computer systems and circuits, and there we are interested in the tautologyhood of a single, carefully constructed, instance. In fact, in our metamathematical musings in Section 15.4 you can read about a single formula (or family of formulae) related to the **P** versus **NP** question that we complexity theorists suspect is a tautology but whose tautologyhood seems difficult to prove. Similarly, in algebraic geometry, one may be interested in understanding the behavior of a single system of equations.

We note that there are languages / decision problems that are *unconditionally proven* not to have short certificates, namely languages outside of **coNP** (such languages can be shown to exist by diagonalization arguments a la Chapter 3). Also, a famous language that does not have any finite certificate at all is the language of true statements on the natural numbers in first-order logic (this is the famous Gödel's incompleteness theorem, see also Section 1.5.2).

15.2 Propositional calculus and resolution

Propositional logic formalizes simple modes of reasoning that have been used in philosophy for two millennia. The basic object of study is the Boolean formula, and an important task is to verify that a given formula is a tautology (i.e., evaluates to TRUE on every assignment). For convenience, we study the complement problem of verifying that the formula is a *contradiction*, namely, has no satisfying assignment. We also study this only for CNF formulae as we know how to reduce the general case to it. Specifically, to verify that a general Boolean formula ψ is a tautology, it suffices to use our reduction from Chapter 2 to transform $\neg\psi$ into an equivalent CNF formula (with additional new variables) and verify that this new formula is a contradiction.

Now we describe a simple procedure called *resolution* that tries to produce a proof that a given formula is a contradiction. Let φ be a CNF formula on the variables $x_1, x_2, \ldots, x_n$. Denote by $C_1, \ldots, C_m$ the clauses of φ. For $j = m+1, m+2, \ldots$, the resolution procedure derives a new clause C_j that is implied by the previous clauses $C_1, \ldots, C_{j-1}$ using the following rule: suppose that there is a variable x_i and clauses C, D such that both the clause

$x_i \vee C$ and $\neg x_i \vee D$ have been derived before (i.e., are in $\{C_1, \ldots, C_{j-1}\}$) then $C_j = C \vee D$. Note that the procedure may have many possible choices for C_j (the proof will be the sequence of choices). The procedure ends when it had derived an obvious contradiction: namely both the clause x_i and $\neg x_i$ for some variable x_i. The *resolution refutation* for φ is a sequence of clauses $C_1, \ldots, C_T$ containing such an obvious contradiction where $C_1, \ldots, C_m$ are φ's clauses and for $j > i$, C_j is derived from $C_1, \ldots, C_{j-1}$ using the above rule. Clearly, every clause we derive is in fact logically implied by the previous ones, and hence resolution is a *sound* proof system: there exists a resolution refutation for φ only if $\neg\varphi$ is a tautology. It is also not to hard to show that resolution is *complete*: if $\neg\varphi$ is a tautology then there exists a resolution refutation for φ of length $2^{O(n)}$ (see Exercise 15.1). The question is whether there are formulae that require such long refutations, or perhaps every unsatisfiable formula has a polynomial-sized refutation? Since Boolean unsatisfiability is **coNP**-complete and we believe that $\mathbf{NP} \neq \mathbf{coNP}$, we believe that the answer is NO. Below we prove this is indeed the case.

15.2.1 Lower bounds using the bottleneck method

We describe Haken's *bottleneck* technique [Hak85] for proving lower bounds for resolution. We will also encounter a version of the *restriction* idea used earlier in Chapter 6 in context of circuit lower bounds.

The tautology considered here is elementary yet basic to mathematics: the *pigeonhole principle*. Colloquially, it says that if you put m pigeons into n holes, where $m > n$, then at least one hole must contain more than one pigeon. Mathematically, it asserts that there is a no one-to-one and onto mapping from a set of size m to a set of size n. Though obvious, this principle underlies many nontrivial facts in mathematics such as the famous *Minkowski convex body theorem.* (See Chapter notes.) Thus it is plausible that a simple proof system like resolution would have trouble proving it succinctly, and this is what we will show.

The propositional version of the pigeonhole principle consists of the class of tautologies $\{\neg\mathrm{PHP}_n^m : m > n\}$ where $\neg\mathrm{PHP}_n^m$ is the following CNF formula. For integers $i \leq m, j \leq n$ its has a variable P_{ij} which is supposed to be assigned "true" if pigeon i is assigned to hole j. It has $m + \binom{m}{2} n \leq m^3$ clauses, which are: (i) $P_{i,1} \vee P_{i,2} \vee \cdots \vee P_{i,n}$ for each $i \leq m$; this says that the ith pigeon is assigned to some hole. (ii) $\neg P_{ik} \vee \neg P_{j,k}$ for each $i, j \leq m$, $k \leq n$; this says that the kth hole does not get both the ith pigeon and the jth pigeon. Thus the entire ensemble of this type of clauses says that no hole gets more than 1 pigeon.

Theorem 15.1 *For any $n \geq 2$, every resolution refutation of $\neg PHP_{n-1}^n$ has size at least $2^{n/20}$.* ♢

We will think of "testing" a resolution refutation by assigning values to the variables. A correct refutation proof shows that no assignment can satisfy all the given set of clauses. We will allow refutations that only show that a certain subset of assignments cannot satisfy all the given clauses. In other words, when we substitute any assignment from this subset, the refutation correctly derives a contradiction. Of course, the refutation may not correctly derive a contradiction for other assignments, so this is a *relaxation* of the notion of resolution refutation. However, any lower bound for this relaxed notion will also apply to the general notion.

The set of assignments used to test the proof will correspond to mappings that map $n-1$ pigeons to $n-1$ holes in a one-to-one manner, and leave the nth pigeon unassigned. In other words, the set of variable P_{ij}'s that are assigned true constitute a matching of size $n-1$. There are $n!$ such assignments. If the index of the sole unassigned pigeon is k we call such an assignment *k-critical*.

Restricting attention to these test assignments simplifies notation since it allows us to make all clauses in the refutation *monotone*; i.e., with no occurence of negated variables. For each clause C in the resolution proof we produce a *monotonized* clause by replacing each negated variable $\neg P_{ij}$ by $\vee_{l \neq i} P_{lj}$. It is easily checked that after this transformation the new clause is satisfied by exactly the same set of test assignments as the original clause. The

next lemma (proved a little later) shows that monotonized refutations must always have a large clause.

Lemma 15.2 *Every monotonized resolution refutation of $\neg PHP^n_{n-1}$ must contain a clause with at least $2n^2/9$ variables.* ♢

With this lemma in hand, we can prove Theorem 15.1 as follows. Say that a clause in the monotonized refutation is *large* if it has at least $n^2/10$ variables. Let L be the number of large clauses; the lemma shows that $L \geq 1$. We define a *restriction* to some of the variables that greatly reduces the number of large clauses. Averaging shows that there exists a variable P_{ij} that occurs in 1/10th of the large clauses. Define a restriction such that $P_{ij} = 1$ and $P_{i,j'} = 0$ for $j' \neq j$ and $P_{i',j} = 0$ for $i' \neq i$. This sets all monotonized clauses containing P_{ij} to true, which means they can be removed from the resolution proof, leaving at most $9/10L$ large clauses. Furthermore, one pigeon and one hole have been removed from contention by the restriction, so we now have a monotonized resolution proof for $\neg\text{PHP}^{n-1}_{n-2}$. Repeating the above step $t = \log_{10/9} L$ times, we obtain a monotonized resolution proof for $\neg\text{PHP}^{n-t}_{n-1-t}$ that has no large clauses. The proof of Theorem 15.1 follows by noticing that if $L < 2^{n/20}$ then $t < n/3$, and so we have a monotonized refutation of $\neg\text{PHP}^{n-t}_{n-t-1}$ with no clauses larger than $n^2/10$, which is less than $2(n-t)^2/9$, and hence this contradicts Lemma 15.2.

Thus to finish we prove Lemma 15.2.

PROOF: (or Lemma 15.2) For each clause C in the monotonized refutation, let

$$\mathit{witness}(C) = \{i \ : \ \text{there is an } i\text{-critical assignment } \alpha \text{ falsifying } C\}.$$

The *complexity* of a clause, $\text{comp}(C)$ is $|\mathit{witness}(C)|$. Whenever resolution is used to derive clause C from two previous clauses $C', C"$ then $\text{comp}(C) \leq \text{comp}(C') + \text{comp}(C")$ since every assignment that falsifies C must falsify at least one of $C', C"$. Thus if C is the *first* clause in the refutation whose complexity is $> n/3$ then $n/3 < \text{comp}(C) < 2n/3$. We show that such a C is large.

Specifically, we show that if $\text{comp}(C) = t$ then it contains at least $t(n-t)$ distinct literals, which finishes the proof since $t(n-t) > 2n^2/9$.

Fix any $i \in \mathit{witness}(C)$ and any i-critical assignment α that falsifies C. For each $j \notin \mathit{witness}(C)$, consider the j-critical assignment α' obtained by replacing i by j, that is, if α mapped pigeon j to hole l then α' leaves j unassigned and maps pigeon i to l. Since $j \notin \mathit{witness}(C)$, this j-critical assignment must satisfy C and so we conclude that C contains variable $P_{i,l}$. By running over all $n-t$ values of $j \notin \mathit{witness}(C)$ and using the same α, we conclude that C contains $n-t$ distinct variables of the type $P_{i,l}$. Repeating the argument for all $i \in \mathit{witness}(C)$ we conclude that C contains at least $t(n-t)$ variables. ■

15.2.2 Interpolation theorems and exponential lower bounds for resolution

This section describes a different lower bound technique for resolution that uses an interesting idea called the *Interpolation theorem*, which plays a role in several results in proof complexity. The lower bound is also interesting because it uses the lower bound for monotone circuits presented in Chapter 14.

First we state the classical (and folklore) version of the Interpolation theorem.

Theorem 15.3 *(Classical Interpolation Theorem)*
Let φ be a Boolean formula over the variables $x_1, \ldots, x_n, z_1, \ldots, z_k$ and ψ be a Boolean formula over the variables $y_1, \ldots, y_m, z_1, \ldots, z_k$ (i.e., the only shared variables are $z_1, \ldots, z_k$). Then $\varphi(\mathbf{x}, \mathbf{z}) \vee \psi(\mathbf{y}, \mathbf{z})$ is a tautology if and only if there is a Boolean function $I : \{0,1\}^k \to \{0,1\}$ such that

$$(\varphi(\mathbf{x}, \mathbf{z}) \vee I(\mathbf{z})) \wedge (\psi(\mathbf{y}, \mathbf{z}) \vee \neg I(\mathbf{z})) \tag{1}$$

is a tautology.

PROOF: It's easy to see that (1) is a tautology if and only if for every fixed assignment $\mathbf{c}$ to the $\mathbf{z}$ variables, either $\varphi(\mathbf{x},\mathbf{c})$ is a tautology or $\psi(\mathbf{y},\mathbf{c})$ is a tautology. Hence if (1) is a tautology then $\varphi(\mathbf{x},\mathbf{z}) \vee \psi(\mathbf{y},\mathbf{z})$ is true for every assignment to the $\mathbf{x},\mathbf{y},\mathbf{z}$ variables. On the other hand, suppose that there exists some $\mathbf{c}$ such that neither $\varphi(\mathbf{x},\mathbf{c})$ nor $\psi(\mathbf{y},\mathbf{c})$ are tautologies. Then this means that there are assignments $\mathbf{a}$ to the $\mathbf{x}$ variables, $\mathbf{b}$ to the $\mathbf{y}$ variables such that both $\varphi(\mathbf{a},\mathbf{c})$ and $\psi(\mathbf{b},\mathbf{c})$ are false. ■

We will be interested in a quantitative version of this interpolation theorem that upper bounds the computational complexity of $I(\cdot)$ as a function of the size of the smallest resolution refutation.

Theorem 15.4 *(Feasible Interpolation Theorem)*
In the setting of Theorem 15.3, if $\neg(\varphi(\mathbf{x},\mathbf{z}) \vee \psi(\mathbf{y},\mathbf{z}))$ has a resolution refutation of size S, then a function I satisfying the conditions of Theorem 15.3 can be computed by a circuit of size $O(S^2)$.
Furthermore, if the variables of $\mathbf{z}$ appear only positively in ψ then the above circuit is monotone *(i.e., contains no negation gates). Similarly, if the variables of $\mathbf{z}$ appear only negatively in φ then the above circuit is monotone.*

PROOF: To prove Theorem 15.4 we need to show how, given a length S resolution refutation for $\neg(\varphi(\mathbf{x},\mathbf{z}) \vee \psi(\mathbf{y},\mathbf{z}))$ and an assignment $\mathbf{c}$ to the $\mathbf{z}$ variables, we can find in $O(S^2)$ time a value $I(\mathbf{c}) \in \{0,1\}$ such that if $I(\mathbf{c}) = 0$ then $\varphi(\mathbf{x},\mathbf{c})$ is a tautology and if $I(\mathbf{c}) = 1$ then $\psi(\mathbf{y},\mathbf{c})$ is a tautology. (We know such a value $I(\mathbf{c})$ exists by Theorem 15.3.)

We show, given $\mathbf{C}$, how to compute $I(\mathbf{c})$ by transforming the size S refutation of $\neg(\varphi(\mathbf{x},\mathbf{z}) \vee \psi(\mathbf{y},\mathbf{z}))$ to a refutation of either $\neg\varphi(\mathbf{x},\mathbf{c})$ or a refutation of $\neg\psi(\mathbf{y},\mathbf{c})$ in $O(S^2)$ time. To do so, we "strip" the clauses of $\mathbf{z}$ variables. That is, we will transform the resolution refutation $C_1,\ldots,C_S$ of $\neg(\varphi(\mathbf{x},\mathbf{z}) \vee \psi(\mathbf{y},\mathbf{z}))$ into a valid resolution refutation $\tilde{C}_1,\ldots,\tilde{C}_S$ of $\neg(\varphi(\mathbf{x},\mathbf{c}) \vee \psi(\mathbf{y},\mathbf{c}))$ where each clause $\tilde{C}_i$ contains either only $\mathbf{x}$ variables (i.e., is an "$\mathbf{x}$-clause") or only $\mathbf{y}$ variables (is a $\mathbf{y}$-clause). But since at the end we derive either a contradiction of the form x_i and $\neg x_i$ or a contradiction of the form y_i and $\neg y_i$ it follows that we have proven that one of these formulae is a contradiction.

We do this transformation step by step. Suppose that clauses $C_1,\ldots,C_{j-1}$ were "stripped" of the $\mathbf{z}$ variables to obtain $\tilde{C}_1,\ldots,\tilde{C}_{j-1}$, and furthermore each clause $\tilde{C}_j$ contains either only $\mathbf{x}$-variables or only $\mathbf{y}$-variables, and we now want to "strip" the clause C_j. It is of the form $C \vee D$ where the clauses $C' = w \vee C$ and $D' = \neg w \vee D$ were derived before for some variable w. By induction we have already obtained "stripped" versions $\tilde{C}$ and $\tilde{D}$ of the clauses C', D'. If $\tilde{C}$ and $\tilde{D}$ are both $\mathbf{x}$-clauses then w must be an $\mathbf{x}$-variable contained in both[1] and we proceed with the usual resolution rule. The case that $\tilde{C}$ and $\tilde{D}$ are both $\mathbf{y}$-clauses is treated similarly. If $\tilde{C}$ is a $\mathbf{x}$-clause and $\tilde{D}$ is a $\mathbf{y}$-clause, then w must be a $\mathbf{z}$-variable, in which case we can just plug in its value according to $\mathbf{c}$ and so if $w = 0$ we simply set $\tilde{C}_j = \tilde{C}$ and if $w = 1$ we set $\tilde{C}_j = \tilde{D}$. We think of the last step in the refutation as containing the empty clause (the one obtained by using the resolution rule on two clauses containing a variable w and its negation $\neg w$). Since the clause $\tilde{C}_j$ is implied by C_j for every j, the last step in the stripped version contains the empty clause as well, implying that the new resolution proof also ends with an obvious contradiction.

We leave the "furthermore" part as Exercise 15.2. However, note that it makes sense since if the $\mathbf{z}$ variables appear only positively in ψ then changing any of them from zero to one is only more likely to make ψ a tautology and hence change $I(\mathbf{c})$ from zero to one. Similar reasoning applies if the $\mathbf{z}$ variables only appear negatively in φ. ■

We are now ready to prove a lower bound on resolution:

[1] We maintain the invariant that we never remove an $\mathbf{x}$-variable from an $\mathbf{x}$-clause or a $\mathbf{y}$-variable from a $\mathbf{y}$-clause.

Theorem 15.5 *(Exponential resolution lower bound)*
There is a constant ϵ such that if for every $n \in \mathbb{N}$ we let $\varphi_n, \psi_n : \{0,1\}^{O(n^2)} \to \{0,1\}$ be the following Boolean functions:

- *$\varphi_n(\mathbf{x}, \mathbf{z}) =$ TRUE iff the string $\mathbf{x}$ represents a clique of size $n^{1/4}$ in the graph represented by $\mathbf{z}$.*
- *$\psi_n(\mathbf{y}, \mathbf{z}) =$ TRUE iff the string $\mathbf{y}$ represents a proper $n^{1/4} - 1$ coloring for the graph represented by $\mathbf{z}$.*

Then, the smallest resolution refutation for $\varphi_n(\mathbf{x}, \mathbf{z}) \wedge \psi_n(\mathbf{y}, \mathbf{z})$ has size at least $2^{\epsilon n^{1/8}}$.

Note that because a graph with a k-clique has no $k-1$ coloring, the formula $\varphi_n(\mathbf{x}, \mathbf{z}) \wedge \psi_n(\mathbf{y}, \mathbf{z})$ is indeed unsatisfiable. Also, it is not hard to express both φ_n and ψ_n as $O(n^2)$-sized CNFs such that φ_n contains the $\mathbf{z}$ variables positively and ψ_n contains them negatively (Exercise 15.3).

Theorem 15.5 follows immediately from Theorem 15.4 and the proof of Theorem 14.7 that gave an exponential lower bound for the monotone circuit complexity of the clique function. This is because that proof actually showed that for $k < n^{1/4}$, there is no $2^{o(\sqrt{k})}$-sized monotone circuit that distinguishes between graphs having a k-clique and graphs whose chromatic number is at most $k-1$. ■

15.3 Other proof systems: a tour d'horizon

Now we briefly explain some other proof systems that have been considered. Several of these are related to the computational problems we mentioned in Section 15.1.

Cutting Planes: This proof system addresses the problem of certifying infeasibility of a set of linear inequalities with integer coefficients and variables. As mentioned in the introduction, this problem is **coNP**-complete. For instance, given any 3CNF formula φ we can represent it by such a set so that the formula is a contradiction iff this set is infeasible. To do so, for each Boolean variable x_i in φ put an integer variable X_i satisfying $0 \leq X_i \leq 1$ (in other words, $X_i \in \{0,1\}$). For a clause $x_i \vee x_j \vee x_k$ write a linear inequality $X_i + X_j + X_k \geq 1$. (If any variable x_i appears negated in the clause, use $1 - X_i$ in the corresponding inequality.)

The cutting planes proof system, given an infeasible set of linear inequalities with integer variables and coefficients, produces a proof of infeasibility by deriving the inequality $0 \geq 1$ in a finite number of steps. It produces a sequence of inequalities $l_1 \geq 0, l_2 \geq 0, \ldots l_T \geq 0$ where the rth inequality is either **(a)** an inequality appearing in the linear system, **(b)** $\alpha l_u + \beta l_v \geq 0$ where α, β are nonnegative integers and $u, v < r$, or **(c)** is derived from some l_u for $u < r$ using the following rule: if l_u has the form

$$\sum_{i=1}^{n} a_i x_i - b \geq 0,$$

where the numbers $a_1, a_2, \ldots, a_n$ have a greatest common divisor D that is at least 2 (i.e., is nontrivial) then the new inequality is

$$\sum_{i=1}^{n} \frac{a_i}{D} x_i - \lceil \frac{b}{D} \rceil \geq 0.$$

(The interesting case is when D does *not* divide b, and hence $\lceil b/D \rceil$ is different from b/D.) There is an interpolation theorem for cutting planes and it has been used to prove exponential lower bounds in [BPR97, Pud97].

Nullstellensatz and Polynomial calculus. These concern infeasibility of sets of equations defined by polynomials. Note that we can also represent infeasibility of 3SAT by such equations. For each variable x_i in the 3CNF formula, have a variable X_i and an equation $X_i^2 - X_i = 0$, thus ensuring that every solution satisfies $X_i \in \{0, 1\}$. We can then transform each clause to a degree-3 equation. For example the clause $x_i \vee x_j \vee \overline{x}_k$ is transformed to the equation $(1 - X_i)(1 - X_j)X_k = 0$.

Hilbert's Nullstellensatz is a basic result in algebra that gives an exact criterion for infeasibility: a set of equations $p_1(X_1, \ldots, X_n) = 0, p_2(X_1, X_2, \ldots, X_n) = 0, \ldots, p_m(X_1, \ldots, X_m) = 0$ in a field $\mathbb{F}$ is infeasible iff there exist polynomials $g_1, g_2, \ldots, g_m$ such that

$$\sum_i g_i(X_1, \ldots, X_n) p_i(X_1, \ldots, X_n) = 1. \tag{2}$$

Notice, these g_i's (if they exist) prove that there can be no assignment of $X_1, \ldots, X_n$ that satisfies all the p_i's, since plugging in any such assignment into (2) would lead to the contradiction $0 = 1$. Thus the nontrivial part of Hilbert's theorem is the fact that such g_i's exist for every infeasible set. (Note that in general the g_i's may have coefficients in some extension field, but in this particular case where the set of polynomials includes $X_i(X_i - 1)$ for all i the solution if any must be 0/1 and then g_i's also must have coefficients in the field.)

Now we define the Nullstellensatz proof system. The axioms are the p_i's and the proof of infeasibility is a sequence of g_i's that satisfy (2). Hilbert's theorem shows that this proof system is sound and complete. We assume that all polynomials are written out explicitly with all coefficients, and the *size* of the proof is the number of bits required to write these coefficients.

Polynomial calculus is similar, except the g_i's can be computed using a straight-line computation instead of being explicitly written out with all coefficients. (Recall that every polynomial can be computed by a straight line program.) Concretely, a refutation in Polynomial calculus is a finite sequence of polynomials $f_1, f_2, \ldots, f_T$ such that each f_r is either **(a)** one of the input polynomials p_i, **(b)** $\alpha f_u + \beta f_v$ where α, β are constants and $u, v < r$, or **(c)** $x_i f_u$ where x_i is a variable and $u < r$. The *size* of the refutation is T and the *degree* is the maximum degree of any f_u.

Exponential lower bounds for the above two proof systems are proved by proving a lower bound of $n^{\Omega(1)}$ on the *degree*; such lower bounds were first proven in [BCE+95].

Frege and Extended Frege: The *Frege* proof system is a general system of reasoning in predicate calculus using a finite set of axiom schemes and inference rules. Resolution is a special case, where all formulae used in the proof are clauses (i.e., disjunctions). An intermediate family is *bounded depth Frege*, where all formulae used in the proof have bounded depth. Ajtai [Ajt88] gave the first lower bounds for bounded depth Frege systems using a clever restriction argument inspired by the restriction argument for **AC**0 that is described in Chapter 14.

Extended Frege is a variant whereby the proof is allowed to introduce new variables $y_1, y_2, \ldots$, and at any step declare that $y_i = \psi$ for some formula ψ. The advantage of this is that now we can use y_i as a bona-fide variable in rules such as resolution, potentially saving a lot of steps. (In general, allowing a proof system to introduce new variables can greatly add to its power.)

No lower bounds are known for Frege and Extended Frege systems, and it is known that existing techniques such as interpolation theorems will likely not work (assuming reasonable complexity assumptions such as "RSA cryptosystem is secure").

15.4 Metamathematical musings

Several researchers suspect that the **P** versus **NP** question may be independent of the axioms of mathematics. Even if it is not independent, it sure seems difficult to prove for

us. Could this question can be a source of tautologies that are difficult to prove in concrete proof systems?

For instance, consider resolution and Frege-like systems for certifying tautologies. We can try to consider the minimum proof size required for a concrete propositional formula that says "SAT instances of size n cannot be solved by circuits of size n^{100}." This formula (first defined in [Raz98]) has $O(n^{100})$ variables denoted $\mathbf{Z}$ and has the form

$$\begin{aligned} &\mathbf{Z} \text{ is an encoding of an } n\text{-input circuit of size } n^{100} \\ &\Rightarrow \text{circuit } \mathbf{Z} \text{ does not compute SAT.} \end{aligned} \tag{3}$$

Note that the conclusion part of (3) is an OR over all 2^n inputs of size n and it says that the value computed by circuit $\mathbf{Z}$ on one of these inputs is not the true value of SAT. Thus such a formula has size $2^{O(n)}$ and *we think* it is a tautology for large enough n. The trivial proof of tautologyhood has size $2^{O(n^{100})}$, however, which is *superpolynomial* in the size of the formula. Can we show that the proof complexity of this formula is superpolynomial for resolution and Frege systems? Razborov [Raz98] showed a superpolynomial lower bound for Polynomial Calculus. He also proposed a different encoding of the above formula for which even resolution lower bounds seemed difficult.

Raz showed that this formula is either not a tautology or requires resolution proofs of superpolynomial size [Raz01, Raz03a, Raz04b]. But similar lower bounds for much stronger systems, say Frege, have not been obtained.

Independence from weak theories of arithmetic. Most results in mathematics can be derived using popular axiomatic systems such as Zermelo-Fraenkel (with axiom of choice) or Peano Arithmetic. But many results in combinatorics, since they have a more finitary character, do not seem to require the full power of these axiomatic systems. Instead, one can use weaker axiomatic systems such as the PV system of Cook [Coo75] or the "Bounded Arithmetic" hierarchy S_i^1 of Buss [Bus90]. Researchers who wish to prove the independence of $\mathbf{P}$ versus $\mathbf{NP}$ from say Peano Arithmetic should perhaps first try to prove independence from such weaker theories. There are deep connections between these theories and the Extended Frege proof system, and lower bounds for the "circuit lower bound formulae" for Extended Frege will imply such independence (see the survey [Raz04c]).

What have we learned?

- Proof complexity aims at proving lower bounds on proof size for tautological formulae in various proof systems.
- Assuming that $\mathbf{NP} \neq \mathbf{coNP}$ then for every complete proof and efficiently verifiable proof system there should exist tautological formulae that do not have polynomial-sized proofs.
- For some proof systems, such as Resolution, Polynomial Calculus and Cutting Plane, there are known exponential lower bounds on proof sizes of various tautologies. However, no super-polynomial lower bounds are known for the Frege and Extended Frege proof systems.

Chapter notes and history

Since proof systems are "nondeterministic", there is in general no obvious algorithm to produce a short proof (if one exists). Nevertheless, heuristic algorithms exist for producing short proofs for many of these systems, and these heuristics are extremely important in practice. In fact, in most cases, the definition of the proof system was inspired by the corresponding heuristic algorithm. Thus proof size lower bounds for all these proof systems prove lower bounds on *running times* of the corresponding heuristic algorithm.

For instance, the definition of resolution is inspired by the Davis-Putnam heuristic [DP60], which inspired a slew of other heuristics such as "resolve the two clauses that produces the smallest resolvent." Haken [Hak85] gave the first superpolynomial lower bounds on the running time of such heuristics; see also [Urq87, CS88] for extensions of this work.

Similarly, the definition of the cutting plane proof system by Chvatal [Chv73] was inspired by Gomory's cutting plane method [Gom63], an important heuristic in commercial software for integer programming.

The word problem for finitely presented groups was articulated by mathematician Dehn in the early 20th century, who gave algorithms for it in many interesting groups. Novikov showed in 1955 that the problem is undecidable in general. Recent work shows that the word problem is in **NP**-complete for some groups [SBR02], implying that the problem of deciding that a given word is *not* trivial is **coNP**-complete. See the book [BMMS00] for a survey.

The feasible interpolation theorem and its use in lower bounds was developed in the string of papers [Kra94, Raz95a, BPR97, Kra97, Pud97].

The polynomial calculus is related to algorithms for solving systems of polynomial equations by computing Groebner bases.

The pigeon hole principle is a source of hard-to-prove tautologies for several weak proof systems including resolution and the polynomial calculus. However, it has a polynomial sized proof in the Frege system.

See the book by Krajicek [Kra95] for an introduction to proof complexity and bounded arithmetic.

Exercises

15.1 Prove that if φ is an unsatisfiable CNF formula on n variables, then there exists a $2^{O(n)}$-length resolution refutation for φ. H462

15.2 Complete the proof of Theorem 15.4 by showing:

(a) If ψ contains the $\mathbf{z}$ variables only positively (without negations) then the algorithm for computing $I(\mathbf{c})$ can be implemented by an $O(S^2)$-sized monotone circuit. H462

(b) If φ contains the $\mathbf{z}$ variables only negatively (always with negations) then then the algorithm for computing $I(\mathbf{c})$ can be implemented by an $O(S^2)$-sized monotone circuit. H463

15.3 Show that both the functions φ_n and ψ_n described in the statement of Theorem 15.5 can be expressed by CNF formulae of size $O(n^2)$. Furthermore, show that the formula φ_n contains the $\mathbf{z}$ variables only positively and the formula ψ_n contains them only negatively.

15.4 Prove that the cutting plane proof system is sound and complete. H463

15.5 Write down the tautology described in words in (3).

15.6 Write down a tautology expressing the pigeonhole principle mentioned in the chapter notes. H463

Chapter 16

Algebraic computation models

"Is Horner's rule optimal for the evaluation of a polynomial?"
Ostrowski (1954)

The Turing machine model captures computations on bits (equivalently, integers), but many natural and useful algorithms are most naturally described as operating on uncountable sets such as the real numbers $\mathbb{R}$ or complex numbers $\mathbb{C}$. A simple example is *Newton's method* for finding roots of a given real-valued function f. It iteratively produces a sequence of candidate solutions $x_0, x_1, x_2, \ldots, \in \mathbb{R}$ where $x_{i+1} = x_i - f(x_i)/f'(x_i)$. Under appropriate conditions this sequence can be shown to converge to a root of f. Likewise, a wide variety of algorithms in numerical analysis, signal processing, computational geometry, robotics, and symbolic algebra typically assume that a basic computational step involves an operation $(+, \times, /)$ in some arbitrary field $\mathbb{F}$. Such algorithms are studied in a field called *computer algebra* [vzGG99].

One could defensibly argue that allowing arbitrary field operations in an algorithm is unrealistic (at least for fields such as $\mathbb{R}$) since real-life computers can only do arithmetic using finite precision. Indeed, in practice algorithms like Newton's method have to be carefully implemented within the constraints imposed by finite precision arithmetic. In this chapter though, we take a different approach and study models which do allow arithmetic operations on real numbers (or numbers from fields other than $\mathbb{R}$). Such an idealized model may not be directly implementable but it provides a useful approximation to the asymptotic behavior as computers are allowed to use more and more precision in their computations. Furthermore, from the perspective of lower bounds, one can hope that techniques from well-developed areas of mathematics such as algebraic geometry and topology may prove handy. As we've seen in Chapter 14, so far we have not been able to prove strong lower bounds for Boolean circuits.

Example 16.1 *(Pitfalls awaiting designers of such models)*
Devising a meaningful, well-behaved model of algebraic computation is not an easy task: allowing (arbitrary precision) arithmetic on real numbers as a basic step can quickly lead to unrealistically strong models. For instance, with n iterations of the basic operation $x \leftarrow x^2$ one can compute 2^{2^n}, a number with 2^n bits. In fact, Shamir has shown how to factor any integer N in $\text{poly}(\log N)$ time on any model that allows arithmetic (including the mod operation) with arbitrary precision (see Exercise 16.10) whereas factoring is a notoriously hard problem for classical TMs.
Furthermore, a real number can encode infinite amount of information. For example, a single real number is enough to encode the answer to every instance of SAT (or any other language, in general). Thus, we have to be careful in defining a model that allows even a single hardwired real number in its programs. By contrast, we can easily allow a normal Turing Machine to have any constant number of integers built into its program.

The usual way to avoid such pitfalls is to restrict the algorithm's ability to access individual bits. Alternatively, when the goal is proving non-trivial lower bounds it is OK to consider unrealistically powerful models. After all, lower bounds for unrealistically powerful models will apply to more realistic (and hence, weaker) models as well.

This chapter is a sketchy introduction to algebraic complexity. It introduces three algebraic computation models: algebraic circuits, algebraic computation trees, and algebraic Turing Machines. The algebraic TM is closely related to the standard Turing Machine model and allows us to study the issues such as decidability and complexity for inputs over arbitrary fields just we did them earlier for inputs over $\{0,1\}$. We introduce an undecidable problem (namely, deciding membership in the Mandelbrot set) and an **NP**-complete problem (decision version of Hilbert's Nullstellensatz) in this model. In general, there seems to be a close connection between algebraic complexity and complexity in the Turing machine world; see Section 16.1.4.

Throughout this chapter, we will consider algorithms that get as input a tuple of numbers over a field or a ring $\mathbb{F}$ (typically $\mathbb{R}$ or $\mathbb{C}$). The input $(x_1, x_2, \ldots, x_n) \in \mathbb{F}^n$ is said to have size n. A *language* over a field/ring $\mathbb{F}$ is a subset of $\cup_{n \geq 1} \mathbb{F}^n$.

16.1 Algebraic straight-line programs and algebraic circuits

In this section we define two simple models of algebraic computation, which turn out to be equivalent. Different authors sometimes prefer one model over the other for reasons of taste or ease of notation. We will also define analogues of **P** and **NP** for these models, and survey the known results, including notions of reductions and completeness for these classes.

16.1.1 Algebraic straight line programs

An *algebraic straight line program* over field $\mathbb{F}$ (or more generally, $\mathbb{F}$ could be a ring) is defined by analogy with Boolean straight line programs (see Note 6.4). It is reminiscent of a fragment of a standard programming language like C or C++, but it has only simple "assignment" statements; no looping or conditional (e.g., **if-then-else**) statements. The formal definition follows:

Definition 16.2 *(Algebraic straight-line program over $\mathbb{F}$)*
An *algebraic straight line program* of length T with *input variables* $x_1, x_2, \ldots, x_n \in \mathbb{F}$ and *built-in constants* $c_1, c_2, \ldots, c_k \in \mathbb{F}$ is a sequence of T statements of the form $y_i = z_{i_1}\ OP\ z_{i_2}$ for $i = 1, 2, \ldots, T$ where OP is one of the field operations $+$ or $\times$ and each of z_{i_1}, z_{i_2} is either an input variable, or a built-in constant, or y_j for $j < i$. For every setting of values to the input variables, the straight-line computation consists of executing these simple statements in order, finding values for $y_1, y_2, \ldots, y_T$. The *output* of the computation is the value of y_T. We can analogously define straight line programs with multiple outputs.

Example 16.3 *(Polynomial evaluation)*
For any $a \in \mathbb{F}$ the function $\sum_i a^i x_i$ is computable by a straight line program of length at most $3n - 2$. We provide the program with a single built-in constant, namely, a. The inputs are $x_1, x_2, \ldots, x_n$. (These inputs are being thought of as the coefficients of a degree $n-1$ polynomial, which is being evaluated at the constant a.) Then computing $a, a^2, a^3, \ldots, a^n$ takes $n - 1$ steps. Multiplying a^i

with x_i for $i = 1, 2, \ldots, n$ takes another n steps. Accumulating the sum $\sum_i a^i x_i$ takes another $n - 1$ steps.

As is clear, the model defined above is *nonuniform*, since a different straight line program could be used for each input length. As usual, we are interested in asymptotic complexity, that is, the length (as a function of n) of the shortest family of algebraic straight line programs that compute a family of functions $\{f_n\}$ where f_n is a function of n variables. Exercise 16.1 asks you to show that straight line programs over GF(2) are essentially equivalent to Boolean circuits, and the same is true for circuits over any finite field. Thus, the case when $\mathbb{F}$ is infinite is usually of greatest interest.

Recall that the *degree* of a multivariate polynomial $p(x_1, \ldots, x_n)$ is defined to be the maximum degree among all its monomials, where the degree of the monomial $c \prod_i x_i^{d_i}$ is $\sum_i d_i$. As the following lemma shows, every straight line program computes a multivariate polynomial of degree related to its length.

Lemma 16.4 *The output of a straight line program of length T with variables $x_1, x_2, \ldots, x_n$ is a polynomial $p(x_1, x_2, \ldots, x_n)$ of degree at most 2^T.* ♢

PROOF: Follows by an easy induction. Each input variable x_i is a polynomial of degree 1, and every step either adds two previous polynomials, or multiplies them. The degree of the product of two polynomials is at most the sum of their degrees. Hence the degree can at most double at each of the T steps. ■

What if we allow the division operator $\div$ as a standard operation in a straight line program? Since there is no way for the program to *test* a value for being nonzero, it could divide by 0 and then the output could be undefined. Another subtlety is that even division by a nonzero polynomial $p(x)$ could lead to undefined result if x is a root of p. Nevertheless, if we consider the formal object being computed, it is well-defined: the next lemma shows that this formal object is a *rational function*, that is, a function of the type $f(x_1, x_2, \ldots, x_n)/g(x_1, \ldots, x_n)$ where f, g are polynomials. The *degree* of the rational function f/g is the sum of degrees of f and g. We omit the (easy) proof.

Lemma 16.5 *If $\div$ (only by nonzero polynomials and scalars) is allowed as an elementary operation, then for every straight line program Π of size t there exists a rational function r of degree at most 2^T that agrees with Π on every input value on which Π is defined.* ♢

Strassen [Str73] gave a general method to transform programs that use division into programs that do not use this operator and have similar size, see also Remark 16.8 below.

16.1.2 Examples

Here are some examples for interesting functions that are computable by polynomial length algebraic straight line programs.

POLYNOMIAL MULTIPLICATION Given $(a_0, a_1, \ldots, a_n)$ and $(b_0, b_1, \ldots, b_n)$ compute the product of the polynomials $\sum_i a_i x^i$ and $\sum_j b_j x^j$, in other words the vector $(c_0, c_1, \ldots, c_{2n-1})$ where $c_k = \sum_{i+j=k} a_i b_j$. Using the ideas of Example 16.3, one obtains a trivial algorithm with straight line complexity $O(n^2)$. Using the Fast Fourier Transform (next example), this can be improved to $O(n \log n)$ for fields that have a primitive mth root of unity, where m is the smallest power of 2 greater than $2n$. The idea is to evaluate the polynomials at m points using the FFT, multiply these values, and use *interpolation* (inverse FFT) to recover the c_i's. A similar approach also works for all fields but with slightly higher $O(n \log n \log \log n)$ run time (Schoenhage and Strassen [SS71]).

FAST FOURIER TRANSFORM The *discrete fourier transform* of a vector $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}) \in \mathbb{C}^n$ is the vector $M \cdot \mathbf{x}$, where M is the $n \times n$ matrix whose (i, j)th entry is ω^{ij} where

ω is a primitive nth root of 1 (i.e., a complex number satisfying $\omega^n = 1$ and not satisfying $\omega^r \neq 1$ for any nonzero $r < n$). See Section 10.6.1.

The trivial idea for a straight line program is to do something like Polynomial Evaluation (the first example above) for each row of M where ω is a built-in constant; this would give a straight line program of length $O(n^2)$. Surprisingly, one can do much better: there is a program of length $O(n \log n)$ to compute the discrete Fourier transform using the famous *fast fourier transform* algorithm due to Cooley and Tukey [CT65] outlined in Section 10.6.1. It is not known if this algorithm is optimal, though Morgenstern [Mor73] has shown that it is optimal in a more restricted model where the only "built in" constants are $0, 1$. Some extensions of this result are also known, see [Cha94].

MATRIX MULTIPLICATION The *matrix multiplication* problem is to compute, given two $n \times n$ matrices $X = (X_{i,j})$ and $Y = (Y_{i,j})$ their product, which is the $n \times n$ matrix Z such that

$$Z_{i,j} = \sum_{k=1}^{n} X_{i,k} Y_{k,j} \tag{1}$$

The equation (1) yields an straight-line program for this problem of size $O(n^3)$. (As mentioned above, the definition of straight-line programs can be easily generalized to handle multiple outputs.) It may seem "obvious" that this is the best one can do, as each of the n^2 outputs requires n operation to compute. However, starting with the work of Strassen in 1969 [Str69], a series of surprising new algorithms have been discovered with complexity $O(n^\omega)$ for $\omega < 3$ (see Exercise 16.4). The current record is $\omega \sim 2.376..$ [CW90]. It is known that the complexity of matric multiplication is equivalent to several other linear algebra problems (see the survey [vzG88]). RAZ [Raz02] has proven that in the model where the only built-in constants are $0, 1$, straight-line programs for matrix multiplication must have size at least $\Omega(n^2 \log n)$.

DETERMINANT The *determinant* of an $n \times n$ matrix $X = (X_{i,j})$ is defined as

$$\det(X) = \sum_{\sigma \in S_n} (-1)^{sgn(\sigma)} \prod_{i=1}^{n} X_{i,\sigma(i)} \,.$$

where S_n is the set of all $n!$ permutations on $\{1, 2, \ldots, n\}$ and $sgn(\sigma)$ is the parity of the number of transposed pairs in σ (i.e., pairs $\langle i, j \rangle$ with $i > j$ but $\sigma(i) < \sigma(j)$). The determinant can be computed using the familiar Gaussian elimination algorithm, but in fact there are improved algorithms (see Exercise 16.6) that also take small *depth* (as defined below in Section 16.1.3).

The determinant function is a good illustration of how the polynomial defining a function may have exponentially many terms —in this case $n!$—but nevertheless be computable with a polynomial length straight line program. The status of lower bounds for algebraic straight line programs is very bad, as the reader probably expects by now. We do know that computing the middle symmetric polynomial requires $\Omega(n \log n)$ operations but do not know of any better bounds for any *explicit* polynomial [BCS97].

16.1.3 Algebraic circuits

An *algebraic circuit* over a field $\mathbb{F}$ is defined by analogy with a Boolean circuit (see Chapter 6). It consists of a directed acyclic graph. The leaves are called *input nodes* and labeled $x_1, x_2, \ldots, x_n$; these take values in $\mathbb{F}$ rather than being Boolean variables. We also allow the circuit to have k additional special input nodes that are labeled with arbitrary constants $c_1, \ldots, x_k$ from the field. Each internal node, called a *gate*, is labeled with one of the arithmetic operations $\{+, \times\}$ rather than with the Boolean operations $\vee, \wedge, \neg$ used in Boolean circuits. We consider only circuits with a single output node and with the in-degree of each gate being 2. The *size* of the circuit is the number of gates in it. The *depth* of the circuit is

the length of the longest path from input to output in it. One can also consider algebraic circuits that allow division ($\div$) at the gates. An *algebraic formula* is a circuit where each gate has out-degree equal to 1.

To evaluate a circuit, we perform each gate's operation by applying it on the numbers present on the incoming wires (= edges), and then passing this output to all its outgoing wires. The output of the circuit is the number present on the wire of its output node at the end of this process. The next lemma (left as an easy Exercise 16.7) shows that this model is equivalent to algebraic straight-line programs.

Lemma 16.6 *Let $f : \mathbb{F}^n \to \mathbb{F}$ be some function. If f has an algebraic circuit of size S then it has an algebraic circuit of size $3S$. If f is computable by an algebraic circuit of size S then it is computable by an algebraic straight line program of length S. Moreover, if the circuit is a formula then the equivalent straight line program is* use once *(i.e., every variable y_i that is not an input occurs on the right hand side of an assignment only once).* ♢

Note that the equivalence is only up to a small constant factor (3) because in a circuit we don't allow parallel edges and hence the operation $x \mapsto x^2$ will require first copying x by adding to it zero.

16.1.4 Analogs of P, NP for algebraic circuits

There are functions which are conjectured to require superpolynomial or even exponential algebraic circuit complexity. The *permanent* (see Sections 8.6.2 and 17.3.1) is one such function. For an $n \times n$ matrix X, the permanent of X is defined as

$$\mathsf{perm}(X) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} X_{i,\sigma(i)} \,.$$

At first sight seems the permanent seems very similar to the determinant. However, unlike the determinant that has a polynomial-time algorithm (and also a polynomial length algebraic straight-line program), the permanent is conjectured to not have such an algorithm. (As shown in Chapter 17, the permanent is **#P**-complete, which in particular means that it does not have a polynomial-time algorithm unless $\mathbf{P} = \mathbf{NP}$.)

Valiant [Val79a] defined analogs of **P** and **NP** for algebraic circuits, as well as as a notion of reducibility. The determinant and permanent functions turn out to play a vital role in this theory, since they are *complete* problems for the following important classes.

Definition 16.7 *($\mathbf{AlgP}_{/\mathbf{poly}}$, $\mathbf{AlgNP}_{/\mathbf{poly}}$)*
Let $\mathbb{F}$ be a field, we say that a family of polynomials $\{p_n\}_{n\in\mathbb{N}}$ (where p_n takes n variables over $\mathbb{F}$) has *polynomially-bounded degree* if there is a constant c such that for every n the degree of p_n is at most cn^c.
The class $\mathbf{AlgP}_{/\mathbf{poly}}$ (or $\mathbf{AlgP}^{\mathbb{F}}_{/\mathbf{poly}}$ when we wish to emphasize the underlying field) contains all polynomially-bounded degree families of polynomials that are computable by algebraic circuits (using no $\div$) of polynomial size and polynomial degree.
The class $\mathbf{AlgNP}_{/\mathbf{poly}}$ is the class of polynomially-bounded degree families $\{p_n\}$ that are definable as

$$p_n(x_1, x_2, \ldots, x_n) = \sum_{e \in \{0,1\}^{m-n}} g_m(x_1, x_2, \ldots, x_n, e_{n+1}, \ldots, e_m) \,,$$

where $g_m \in \mathbf{AlgP}_{/\mathbf{poly}}$ and m is polynomial in n.

Many texts use the names **VP** and **VNP** for the classes $\mathbf{AlgP}_{/\mathbf{poly}}$ and $\mathbf{AlgNP}_{/\mathbf{poly}}$, where V stands for Valiant, who defined these classes and proved several fundamental results

on their properties. We chose to use the notation $\mathbf{AlgP}_{/\mathbf{poly}}, \mathbf{AlgNP}_{/\mathbf{poly}}$ to emphasize the non-uniformity of these classes.

Remark 16.8
Disallowing the $\div$ operation in the definition of $\mathbf{AlgP}_{/\mathbf{poly}}$ may seem like a strong restriction, but Strassen [Str73] has shown that for infinite fields, the class $\mathbf{AlgP}_{/\mathbf{poly}}$ is unchanged whether or not $\div$ is allowed. Similarly, the class $\mathbf{AlgNP}_{/\mathbf{poly}}$ is unchanged if we require g_m to have polynomial *formula* size in addition to being in $\mathbf{AlgP}_{/\mathbf{poly}}$ [Val79a].

Example 16.9
To illustrate the definition of $\mathbf{AlgNP}_{/\mathbf{poly}}$ we show that permanent is in $\mathbf{AlgNP}_{/\mathbf{poly}}$. A permutation on $[n]$ will be represented by an $n \times n$ *permutation matrix* whose each entry is 0/1 and whose each row/column contains exactly one 1. The crux of the proof is to express the condition that values to some n^2 variables form a permutation.
For any set of n variables $c_1, c_2, \ldots, c_n$ let the polynomial Exactly-one be such that for any 0/1 assignment to the c_i's this polynomial is 1 if exactly one of c_i's is 1, and zero otherwise.

$$\mathsf{Exactly\text{-}one}(c_1, c_2, \ldots, c_n) = \sum_{i \leq n} c_i \prod_{j \neq i} (1 - c_j).$$

Now define a polynomial Is-permutation with n^2 binary variables σ_{ij} for $1 \leq i, j \leq n$ that verifies that each row and column contains exactly one 1.

$$\mathsf{Is\text{-}permutation}(\sigma) = \prod_i \mathsf{Exactly\text{-}one}(\sigma_{i1}, \sigma_{i2}, \ldots, \sigma_{in}) \mathsf{Exactly\text{-}one}(\sigma_{1i}, \sigma_{2i}, \ldots, \sigma_{ni}).$$

Finally, let Permpoly be a polynomial of n^2 variables σ_{ij} for $1 \leq i, j \leq n$ and n^2 variables X_{ij} for $1 \leq i, j \leq n$ defined as

$$\mathsf{Permpoly}(\sigma, \mathbf{X}) = \mathsf{Is\text{-}permutation}(\sigma) \prod_i (\sum_j X_{ij} \sigma_{ij}).$$

Clearly, $\mathsf{Permpoly} \in \mathbf{AlgP}_{/\mathbf{poly}}$. Finally, the permanent of $\mathbf{X}$ can be written as

$$\sum_{\sigma \in \{0,1\}^{n^2}} \mathsf{Permpoly}(\sigma, \mathbf{X}),$$

we have shown that the permanent function is in $\mathbf{AlgNP}_{/\mathbf{poly}}$.

The definition of $\mathbf{AlgNP}_{/\mathbf{poly}}$ is somewhat unexpected and merits some discussion. Valiant was motivated by the view that $+$ is the algebraic analog of the Boolean OR. Recall that a language A is in **NP** if there is a language $B \in \mathbf{P}$ such that $x \in A \Leftrightarrow \exists e$ s.t.$(x, e) \in B$. Thus the definition of **NP** involves $\exists_{e \in \{0,1\}^{m-n}}$, which is equivalent to an OR, viz, $\bigvee_{e \in \{0,1\}^{m-n}}$. Thus the algebraic analog is the operation $\sum_{e \in \{0,1\}^{m-n}}$, and this is the defining feature of $\mathbf{AlgNP}_{/\mathbf{poly}}$. Note that this makes $\mathbf{AlgNP}_{/\mathbf{poly}}$ closer to **#P** in spirit than to **NP**.

Now we arrive at a key notion: reduction between algebraic problems that preserve algebraic circuit complexity. As usual, we want a reduction f from problem A to problem B to satisfy the property that an efficient algorithm (i.e., polynomial-length straight line program or polynomial size circuit) for B should give us an efficient algorithm for A. Some thought suggests that it suffices to let the reduction be an arbitrary polynomially-bounded degree family that is computable by a polynomial-length straight-line program. The definition suggested by Valiant is much stricter: it requires the reduction to be an extremely trivial "change of variables." Obviously, the stricter the definition of reduction, the harder

it is to prove completeness results. So the fact that such a simple reduction suffices here is surprising. (Of course, if we think about most classical **NP**-completeness results from the 1970s, they also involve simple local transformations using gadgets, instead of arbitrary polynomial-time transformations.)

Definition 16.10 *(Projection reduction)* A function $f(x_1, \ldots, x_n)$ is a *projection* of a function $g(y_1, y_2, \ldots, y_m)$ if there is a mapping σ from $\{y_1, y_2, \ldots, y_m\}$ to $\{0, 1, x_1, x_2, \ldots, x_n\}$ such that $f(x_1, x_2, \ldots, x_n) = g(\sigma(y_1), \sigma(y_2), \ldots, \sigma(y_m))$.

We say that f is *projection-reducible* to g if f is a projection of g. ♢

Example 16.11
The function $f(x_1, x_2) = x_1 + x_2$ is projection reducible to $g(y_1, y_2, y_3) = y_1^2 y_3 + y_2$ since $f(x_1, x_2) = g(1, x_1, x_2)$.

One way to think of a projection reduction is that if we had a silicon chip for computing g then we could convert it to a chip for f by appropriately hardwiring its input wires, feeding either some x_i or 0 or 1 into each input wire. The next theorem shows that a chip for the Determinant or Permanent would be fairly "universal" in that it can be made to compute large families of other functions. Its proof uses clever gadget constructions, and is omitted here.

Theorem 16.12 *(Completeness of determinant and permanent* [Val79a]*)*
For every field $\mathbb{F}$ and every polynomial family on n variables that is computable by an algebraic formula of size u is projection reducible to the Determinant function (over the same field) on $u + 2$ variables.

For every field except those that have characteristic 2, every polynomial family in $\mathbf{AlgNP}_{/\mathbf{poly}}$ is projection reducible to the Permanent function (over the same field) with polynomially more variables.

Moreover, it was shown by Valiant et al [VSBR81] that every function in $\mathbf{AlgP}_{/\mathbf{poly}}$ has an algebraic formula of size $2^{O(\log^2 n)}$ (see also Exercise 16.6). Thus separating $\mathbf{AlgP}_{/\mathbf{poly}}$ and $\mathbf{AlgNP}_{/\mathbf{poly}}$ will follow from this purely mathematical conjecture that makes no mention of computation:

Conjecture 16.13 *(Valiant)*
For every field that does not have characteristic 2, the $n \times n$ permanent cannot be obtained as a projection of the $m \times m$ determinant where $m = 2^{O(\log^2 n)}$.

Conjecture 16.13 is a striking example of the close connection between computational complexity and interesting questions in pure mathematics. Another intriguing fact is that that it is necessary to show $\mathbf{AlgP}_{/\mathbf{poly}} \neq \mathbf{AlgNP}_{/\mathbf{poly}}$ before one can show $\mathbf{P} \neq \mathbf{NP}$ (see Chapter notes).

16.2 Algebraic Computation Trees

Now we move to a more powerful computational model, the algebraic computation tree. This can be defined for computations over an arbitrary ring (see the comments after Definition 16.15) but for simplicity we define it for for computations on real-valued inputs. This model augments the straight line program model (with $\div$) with the ability to do conditional branches based upon whether or not a variable y_v is greater than 0. Depending upon the

result of this comparison, the computation proceeds in one of two distinct directions. Thus the overall structure is a binary tree rather than a straight line (as the name suggests). The ability to branch based upon a variable value is somewhat reminiscent of a Boolean decision tree of Chapter 12 but here the variables (indeed, also the input) are real numbers instead of bits.

Formally, the model can be used to solve *decision problems* on real inputs; it computes a Boolean-valued function $f : \mathbb{R}^n \to \{0, 1\}$ (i.e., a language).

Example 16.14 *(Some decision problems)*
The following examples illustrate some of the languages (over real numbers) whose complexity we might want to study.

Element Distinctness Given n numbers $x_1, x_2, \ldots, x_n$, determine whether they are all distinct. This is equivalent to the question whether $\prod_{i \neq j}(x_i - x_j) \neq 0$.

Real-valued subset sum Given a list of n real numbers $x_1, \ldots, x_n$, determine whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} x_i = 1$.

As motivation for the definition of the model, consider the trivial algorithm for Element Distinctness: sort the numbers in $O(n \log n)$ steps and then check in another $O(n)$ steps if any two adjacent numbers in the sorted order are the same. Is this trivial algorithm actually optimal, or can we solve the problem in $o(n \log n)$ steps? The answer must clearly depend on the computational model we allow. The *algebraic computation tree* model studied in this section is powerful enough to implement known algorithms for the problem. As we will see in Section 16.2.1 below, it turns out that in this model the above trivial algorithm for Element Distinctness is optimal up to constant factors.

Recall that comparison-based sorting algorithms only ask questions of the type "Is $x_i > x_j$?", which is the same as asking whether $x_i - x_j > 0$. The left hand side term of this last inequality is a linear function. We can imagine other algorithms that may use more complicated functions. In *Algebraic Computation Trees*, we allow a) the ability to use any rational function and b) the introduction of new variables together with the ability to do arithmetic on them and ask questions about them. The *cost* is the number of arithmetic operations and branching steps on the worst case input.

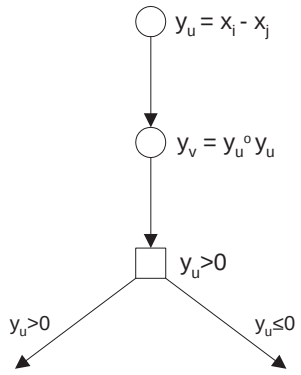


Figure 16.1 An Algebraic Computation Tree

Definition 16.15 *(Algebraic Computation Tree over $\mathbb{R}$)*
An *Algebraic Computation Tree* is a way to represent a function $f : \mathbb{R}^n \to \{0, 1\}$ by showing how to compute $f(x_1, x_2, \ldots, x_n)$ for any input vector $(x_1, x_2, \ldots, x_n)$. It is a binary tree where each of the nodes is of one of the following types:

- Leaf labeled "Accept" or "Reject".
- Computation node v labeled with y_v, where $y_v = y_u$ OP y_w and y_u, y_w are either one of $\{x_1, x_2, \ldots, x_n\}$ or the labels of ancestor nodes and the operator OP is in $\{+, -, \times, \div, \sqrt{}\}$.
- Branch node with out-degree 2. The branch that is taken depends on the evaluation of some condition of the type $y_u = 0$ or $y_u \geq 0$ or $y_u \leq 0$ where y_u is either one of $\{x_1, x_2, \ldots, x_n\}$ or the labels of an ancestor node in the tree.

The computation on any input $(x_1, x_2, \ldots, x_n)$ follows a single path from the root to a leaf, evaluating functions at internal nodes (including branch nodes) in the obvious way, until it reaches a leaf. It reaches a leaf marked "Accept" iff $f(x_1, x_2, \ldots, x_n) = 1$. The complexity of the computation on the path is measured using the following costs (which reflect real-life costs to some degree):

- $+, -$ are free.
- $\times, \div, \sqrt{}$ and branch nodes are charged unit cost.

The *depth* of the tree is the maximum cost of any path in it.

This definition allows $\sqrt{}$ as an elementary operation, which may not make sense for all fields (such as the rational numbers). The notion of algebraic computation tree extends to arbitrary ordered fields by omitting the $\sqrt{}$ as an operation. The notion also extends to fields that are not ordered by only allowing decision nodes that have a 2-way branch based upon whether or not a variable is $= 0$.

A fragment of an algebraic computation tree is shown in Figure 16.1.

Definition 16.16 *(algebraic computation tree complexity)*
Let $f : \mathbb{R}^n \to \{0, 1\}$. The *algebraic computation tree complexity* of f is

$$AC(f) = \min_{\substack{\text{computation} \\ \text{tree } T \text{ for } f}} \{\text{depth of } T\}$$

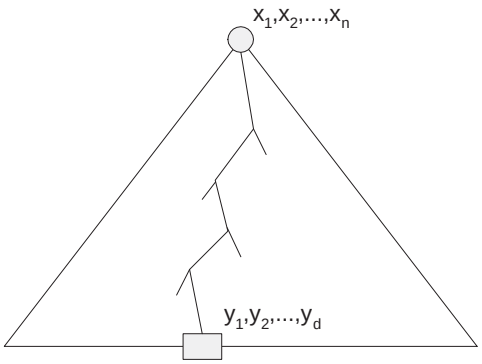


Figure 16.2 A computation path p of length d defines a set of constraints over the n input variables x_i and d additional variables y_j, which correspond to the nodes on p.

The algebraic computation tree model is much more powerful than a real-life programming language. The reason is that a tree of depth d could have 2^d nodes, so a depth d algebraic computation tree would yield (in the worst case) only a classical algorithm with a description of size 2^d. This is why the following theorem (whose proof we omit) does not imply an efficient algorithm for the **NP**-complete subset sum problem:

Theorem 16.17 *(Meyer auf der Heide* [adH88]*) The real number version of* SUBSET SUM *can be solved using an algebraic computation tree of depth* $O(n^5)$. ♢

This theorem suggests that Algebraic Computation Trees are best used to investigate lower bounds such as $n \log n$ or n^2 rather than something more ambitious like a super-polynomial lower bound for the real number version of SUBSET SUM.

16.2.1 The topological method for lower bounds

To prove lower bounds for the minimum cost of an algebraic computation tree algorithm for a function f, we will use the *topology* of the sets $f^{-1}(1)$ and $f^{-1}(0)$, specifically, the number of connected components.

Definition 16.18 *(connected components)* A set $S \subseteq \mathbb{R}^n$ is *connected* if for all $\mathbf{x}, \mathbf{y} \in S$ there is path p from $\mathbf{x}$ to $\mathbf{y}$ that lies entirely in S (in other words, a continuous function mapping $[0,1] \subseteq \mathbb{R}$ to $\mathbb{R}^n$ such that $f(0) = \mathbf{x}, f(1) = \mathbf{y}$ and $f(t) \in S$ for all $t \in [0,1]$). A *connected component* of $W \subseteq \mathbb{R}^n$ is a connected subset of W that is not a proper subset of any other connected subset of W. We let $\#(W)$ denote the number of connected components of W.♢

The following theorem relates the number of connected components to the algebraic computation tree complexity:

Theorem 16.19 *(Topological lower bound on algebraic tree complexity* [BO83]*)* *For every* $f : \mathbb{R}^n \to \{0,1\}$,

$$AC(f) = \Omega\Big(\log \left(\max \{\#(f^{-1}(1)), \#(\mathbb{R}^n \setminus f^{-1}(1))\}\right) - n \Big)$$

Before proving this theorem, let us first use it to prove the promised lower bound for Element Distinctness. This bound follow directly from the following theorem, since $\log n! = \Omega(n \log n)$.

Theorem 16.20 *Let* $W = \{(x_1, \ldots, x_n) | \prod_{i \neq j} (x_i - x_j) \neq 0\}$. *Then,*

$$\#(W) \geq n! \qquad \diamondsuit$$

PROOF: For each permutation σ let

$$W_\sigma = \{(x_1, \ldots, x_n) \mid x_{\sigma(1)} < x_{\sigma(2)} < \ldots < x_{\sigma(n)}\}.$$

That is, let W_σ be the set of n-tuples $(x_1, \ldots, x_n)$ which respect the (strict) order given by σ. Note that $W_\sigma \subseteq W$ for all σ. It suffices to prove for all $\sigma' \neq \sigma$ that the sets W_σ and $W_{\sigma'}$ are not connected.

For any two distinct permutations σ and σ', there exist two distinct i, j with $1 \leq i, j \leq n$, such that $\sigma^{-1}(i) < \sigma^{-1}(j)$ but $\sigma'^{-1}(i) > \sigma'^{-1}(j)$. Thus, in W_σ we have $X_j - X_i > 0$ while in $W_{\sigma'}$ we have $X_i - X_j > 0$. Consider any path from W_σ to $W_{\sigma'}$. Since $X_j - X_i$ has different signs at the endpoints, the intermediate value principle says that somewhere along the path this term must become 0. That point can belong in neither W_σ nor $W_{\sigma'}$, so Definition 16.18 then implies that W_σ and $W_{\sigma'}$ cannot be connected. ■

Now we turn to the proof of Theorem 16.19. This theorem is proved in two steps. First, we try to identify the property of functions with algebraic computation tree complexity: they can be defined as solution sets of a "few" systems of equations.

Lemma 16.21 *If* $f : \mathbb{R}^n \to \{0, 1\}$ *has a decision tree of depth* d *then* $f^{-1}(1)$ *(and also* $f^{-1}(0)$*) is a union of at most* 2^d *sets* $C_1, C_2, \ldots, \subseteq \mathbb{R}^n$ *where* C_i *can be described as follows: there is a system of up to* d *equations of the form*

$$p_{i_r}(y_1, \ldots, y_d, x_1, \ldots, x_n) \bowtie 0,$$

where p_{i_r} *for* $r \leq d$ *is a degree* 2 *polynomial,* $\bowtie$ *is in* $\{\leq, \geq, =, \neq\}$*, and* $y_1, \ldots, y_d$ *are new variables. Then* C_i *is the set of* $(x_1, x_2, \ldots, x_n)$ *for which there are some* $y_1, y_2, \ldots, y_d$ *such that* $(y_1, \ldots, y_d, x_1, \ldots, x_n)$ *is a solution to the above system. Additionally, we may assume without loss of generality (at the cost of doubling the number of* y_i*'s) that there are no* $\neq$ *constraints in this system of equations.* $\diamondsuit$

PROOF: The tree has 2^d leaves, so it suffices to associate a set C_i with each leaf, which is the set of $(x_1, x_2, \ldots, x_n)$ that end up at that leaf. Associate a variable $y_1, y_2, \ldots, y_d$ with the (at most) d computation or branching nodes appearing along the path from root to this leaf. For each computation node, we associate an equation with it in the obvious way (see Figure 16.2). For example, if the node computes $y_v = y_u \div y_w$ then it implies the constraint $y_v y_w - y_u = 0$. For each branch node, we associate an obvious inequality. Thus any $(x_1, x_2, \ldots, x_n)$ that end up at the leaf is a vector for which there exist values of $y_1, y_2, \ldots$, such that the combined vector is a solution to this system of d equations and inequalities.

To replace the "$\neq$" constraints with "=" constraints we take a constraint like

$$p_i(y_1, \ldots, y_m) \neq 0,$$

introduce a new variable z_i and impose the constraint

$$q_i(y_1, \ldots, y_m, z_i) \equiv 1 - z_i p_i(y_1, \ldots, y_m) = 0.$$

(This transformation, called Rabinovitch's trick, holds for all fields.) Notice, the maximum degree of the constraint remains 2, because the trick is used only for the branch $y_u \neq 0$ which is converted to $1 - z_v y_u = 0$.

Similarly, the constraint $p_i(y_1, \ldots, y_m) > 0$ is handled by introducing a new variable z_i and imposing the constraint $p_i(y_1, \ldots, y_m) = z_i^2$. ■

We find Rabinovitch's trick useful also in Section 16.3.2 where we prove a completeness result for Hilbert's Nullstellensatz.

Now to prove lower bounds on $AC(W)$ via the topological argument, we need some result about the number of connected components of the set of solutions to an algebraic system. The following is a central result in mathematics.

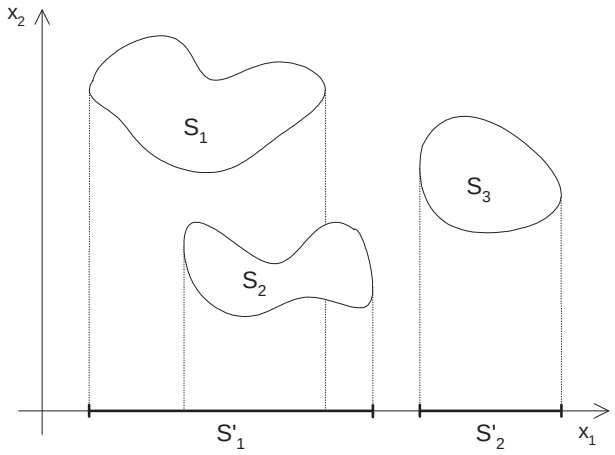


Figure 16.3 Projection can merge but not add connected components

Theorem 16.22 *(Consequence of Milnor-Thom Theorem) If $S \subseteq \mathbb{R}^n$ is defined by degree d constraints with m equalities and h inequalities then*

$$\#(S) \leq d(2d-1)^{n+h-1} \qquad \diamondsuit$$

Note that the above upper bound is independent of m. Now we can prove Ben-Or's Theorem.

PROOF OF THEOREM 16.19: Suppose that the depth of a computation tree for W is d, so that there are at most 2^d leaves. We will use the fact that if $S \subseteq \mathbb{R}^n$ and $S|_k$ is the set of points in S with their last $n-k$ coordinates removed (i.e., projection on the first k coordinates) then $\#(S|_k) \leq \#(S)$ (Figure 16.3).

For every leaf there is a set of degree 2 constraints. So, consider a leaf ℓ and the corresponding constraints $\mathcal{C}_\ell$, which are in variables $y_1, \ldots, y_d, x_1, \ldots, x_n$. Let $W_\ell \subseteq \mathbb{R}^n$ be the subset of inputs that reach ℓ and $S_\ell \subseteq \mathbb{R}^{n+d}$ the set of points that satisfy the constraints $\mathcal{C}_\ell$. Note that $W_\ell = \mathcal{C}_\ell|_n$ i.e., W_ℓ is the projection of $\mathcal{C}_\ell$ onto the first n coordinates. So, the number of connected components in W_ℓ is upperbounded by $\#(\mathcal{C}_\ell)$. By Theorem 16.22 it holds that $\#(\mathcal{C}_\ell) \leq 2 \cdot 3^{n+d-1} \leq 3^{n+d}$. Therefore the total number of connected components is at most $2^d 3^{n+d}$, so $d \geq \Omega(\log(\#(W))) - O(n)$. By repeating the same argument for $\mathbb{R}^n - W$ we have that $d \geq \Omega(\log(\#(\mathbb{R}^n - W))) - O(n)$. ■

16.3 The Blum-Shub-Smale Model

The models for algebraic complexity introduced so far were *nonuniform*. Now we introduce a *uniform* model due to Blum, Shub and Smale [BSS89]. This involves Turing Machines that compute over some arbitrary field or ring $\mathbb{F}$ (e.g., $\mathbb{F} = \mathbb{R}, \mathbb{C}, \mathrm{GF}(2)$); the input is a string in $\mathbb{F}^n$ for $n \geq 1$ and the output is Accept/Reject. Each cell can hold an element of $\mathbb{F}$, and initially, all but a finite number of cells are "blank." Thus the model is a generalization of the standard Turing Machine model with bit operations, which can be seen as operating over the field GF(2); see Exercise 16.9. The machine has a finite set of internal states. Each state belongs to one of the following three categories:

- **Shift state:** move the head to the left or to the right of the current position.
- **Branch state:** if the content of the current cell is a then goto state q_1 else goto state q_2.

- **Computation state:** This state has a hardwired function f associated with it. When the machine is in this state, it reads the contents of the current cell, say $a \in \mathbb{F} \cup \{\text{blank}\}$, and replaces it with a new value $f(a)$. If $\mathbb{F}$ is a ring, f is a polynomial over $\mathbb{F}$, while if $\mathbb{F}$ is a field then we allow f to be any rational function of the form g/h where g, h are polynomials and h is non-zero. In either case, f is represented using a constant number of elements of $\mathbb{F}$. These can be viewed as "hardwired" constants for the machine.

Note that in our standard model of the TM, the computation and branch operations can be executed in the same step, whereas here they have to be performed separately. This is purely for notational ease. But, now in order to branch, the machine has to be able to "remember" the value it just read one step ago. For this reason the machine has a single "register" onto which it can copy the contents of the cell currently under the head, and whose value can be used in the next step.

Like other models for algebraic complexity we have studied, the BSS model seems more powerful than real-world computers. For instance, by repeating the operation $x \leftarrow x^2$, the BSS machine can compute and store in one cell (without overflow) the number x^{2^n} in n steps.

However, the machine has only limited ability to benefit from such computations because it can only branch using tests like "Is the content of this cell equal to a?" The slight variant of this test, featuring an inequality test: "Is the content of the cell greater than a?" would give the machine much more power, including the ability to decide every language in $\mathbf{P}_{/\mathbf{poly}}$ (thus in particular, an undecidable language) in polynomial time. The reason is that the circuit family of a language in $\mathbf{P}_{/\mathbf{poly}}$ circuit family can be represented by a *single* real number that is a hardwired "constant" into the Turing machine (specifically, as the coefficient of some polynomial $p(x)$ belonging to a state). The individual bits of this coefficient can be accessed by dividing by 2 an appropriate number of times and then using the branch test to check if the number is greater than 0. (The details are left as Exercise 16.12.) Thus the machine can extract the polynomial length encoding of each circuit.

Similarly, if we allow rounding (computation of $\lfloor x \rfloor$) as a basic operation then it is possible to factor integers in polynomial time on the BSS model, using the ideas of Shamir mentioned earlier (see Exercise 16.10).

Note also that the BSS model is closely related to a more classical model: algebraic circuits with "branch" gates and a "uniformity" condition (so the circuits for different input sizes have to be constructible by some conventional Turing machine).

16.3.1 Complexity Classes over the Complex Numbers

It is now time to define some complexity classes related to the BSS model. For simplicity we restrict attention to machines over the field $\mathbb{C}$. As usual, the complexity of these Turing Machines is defined with respect to the input size (i.e., number of cells occupied by the input). The following complexity classes correspond to **P** and **NP** over $\mathbb{C}$:

Definition 16.23 *($\mathbf{P}_{\mathbb{C}}$,$\mathbf{NP}_{\mathbb{C}}$)* $\mathbf{P}_{\mathbb{C}}$ contains every language over $\mathbb{C}$ that can be decided by a BSS Turing Machine over $\mathbb{C}$ in polynomial time. A language L is said to be in $\mathbf{NP}_{\mathbb{C}}$ if there exists a language $L_0 \in \mathbf{P}_{\mathbb{C}}$ and a number $d > 0$, such that an input x is in L iff there exists a string $(y_1, \ldots, y_{n^d})$ in $\mathbb{C}^{n^d}$ such that (x, y) is in L_0. ♢

It is also interesting to study the complexity of standard languages (i.e., whose inputs are bit strings) with respect to this model. Thus, we make the following definition:

$$\text{0-1-}\mathbf{NP}_{\mathbb{C}} = \{L \cap \{0,1\}^* \mid L \in \mathbf{NP}_{\mathbb{C}}\}$$

Note that the input for a 0-1-$\mathbf{NP}_{\mathbb{C}}$ machine is binary but the nondeterministic "witness" may consist of complex numbers. Trivially, **NP** is a subset of 0-1-$\mathbf{NP}_{\mathbb{C}}$. The reason is that even though the "witness" for the BSS machine consists of a string of complex numbers, the machine can first check if they are all 0 or 1 using equality checks. Having verified that the

witness is actually a Boolean string, the machine continues as a normal Turing Machine to verify it.

Is 0-1-$\mathbf{NP}_{\mathbb{C}}$ much larger than **NP**? We know that that 0-1-$\mathbf{NP}_{\mathbb{C}} \subseteq$ **PSPACE**. In 1997 Koiran [Koi97] proved that if one assumes the Riemann hypothesis, then 0-1-$\mathbf{NP}_{\mathbb{C}} \subseteq$ **AM**. As shown in Chapter 20 (see Exercise 20.7), under reasonable assumptions **AM** = **NP** and so Koiran's result suggests that it's likely that 0-1-$\mathbf{NP}_{\mathbb{C}} =$ **NP**.

16.3.2 Complete problems and Hilbert's Nullstellensatz

The language $\mathbf{HN}_{\mathbb{C}}$ is defined as the decision version of Hilbert's Nullstellensatz over $\mathbb{C}$. (We encountered this principle in Section 2.7 and it also appears in Section 15.3.) The input consists of m polynomials $p_1, p_2, \ldots, p_m$ of degree d over $x_1, \ldots, x_n$. The output is "yes" iff the polynomials have a common root $a_1, \ldots, a_n$. Note that this problem is general enough to encode SAT, since we can represent each clause by a polynomial of degree 3:

$$x \vee y \vee z \leftrightarrow (1-x)(1-y)(1-z) = 0.$$

Next we use this fact to prove that the language 0-1-$\mathbf{HN}_{\mathbb{C}}$ (where the polynomials have 0-1 coefficients) is complete for 0-1-$\mathbf{NP}_{\mathbb{C}}$.

Theorem 16.24 *([BSS89]) 0-1-$\mathbf{HN}_{\mathbb{C}}$ is complete for 0-1-$\mathbf{NP}_{\mathbb{C}}$.* ♢

Proof Sketch: It is straightforward to verify that 0-1-$\mathbf{HN}_{\mathbb{C}}$ is in 0-1-$\mathbf{NP}_{\mathbb{C}}$. To prove the hardness part we imitate the proof of the Cook-Levin Theorem (Theorem 2.10). Recall that there we reduced every **NP**-computation into an AND of many *local* tests, each depending on only a constant number of variables. Here, we do the same, reasoning as in the case of algebraic computation trees (see Lemma 16.21) that we can express these local checks with polynomial constraints of bounded degree. The computation states $c \leftarrow q(a,b)/r(a,b)$ are easily handled by setting $p(c) \equiv q(a,b) - cr(a,b)$. For the branch states $p(a,b) \neq 0$ we can use Rabinovitch's trick to convert them to equality checks $q(a,b,z) = 0$. Thus the degree of our constraints depends upon the degree of the polynomials hardwired into the machine. Also, the polynomial constraints use real coefficients (involving real numbers hardwired into the machine). Converting these polynomial constraints to use only 0 and 1 as coefficients requires work. The idea is to show that the real numbers hardwired into the machine have no effect since the input is a binary string. We omit this argument here. ■

16.3.3 Decidability Questions: Mandelbrot Set

Since the Blum-Shub-Smale model is more powerful than the ordinary Turing Machine, it makes sense to also revisit decidability questions. In this section we mention an interesting undecidable problem for this model: membership problem for the *Mandelbrot set*, a famous fractal. The chapter notes mention one motivation for studying such questions, connected to Roger Penrose's claim that Artificial Intelligence is impossible.

Definition 16.25 *(Mandelbrot set decision problem)* Let $P_C(Z) = Z^2 + C$. Then, the Mandelbrot set is defined as

$$\mathcal{M} = \{C \in \mathbb{C} \mid \text{the sequence } P_C(0), P_C(P_C(0)), P_C(P_C(P_C(0))) \ldots \text{ is bounded } \}. \quad \diamondsuit$$

Note that the complement of $\mathcal{M}$ is recognizable if we allow inequality constraints. This is because the sequence is unbounded iff some number $P_C^k(0)$ has complex magnitude greater than 2 for some k (exercise!) and this can be detected in finite time. However, detecting that $P_C^k(0)$ is bounded for every k seems harder. Indeed, we have:

Theorem 16.26 *$\mathcal{M}$ is undecidable by a machine over $\mathbb{C}$.* ♢

PROOF SKETCH: The proof uses some mathematical tools that are beyond the scope of this book and hence we only give a rough sketch here. The proof uses the topology of the Mandelbrot set, and the notion of *Hausdorff* dimension. A *ball* of radius r in a metric space is a set of the form $B(x_0, r) = \{y : \text{dist}(x_0, y) < r\}$. Very roughly speaking, the Hausdorff dimension of a space is d if as $r \to 0$ then the minimum number of balls of radius r required to cover a set grows as $1/r^d$ as r goes to 0.

Let $\mathcal{N}$ be any TM over the complex numbers that supposedly decides this set. Consider T steps of the computation of this TM. Reasoning as in Theorem 16.24 and in our theorems about algebraic computation trees, we conclude (see also Exercise 16.11) that the sets of inputs accepted in T steps is a finite union of semialgebraic sets (i.e., sets defined using solutions to a system of polynomial equations). Hence the language accepted by $\mathcal{N}$ is a countable union of semi-algebraic sets, which is known to imply that its Hausdorff dimension is 1. But it is known that the Mandelbrot set has Hausdorff dimension 2, hence $\mathcal{M}$ cannot decide it. ■

WHAT HAVE WE LEARNED?

- It is possible to study computational complexity in more algebraic settings where a basic operation is over a field or ring. We saw analogs of Boolean circuits, straight line programs, decision trees, and Turing machines.
- One can define *complete* problems in some algebraic complexity classes, and even study decidability.
- Proving lower bounds for algebraic computation trees involves interesting topological methods involving the number of connected components in the set of solutions to a system of polynomial equations.
- There are interesting connections between algebraic complexity and the notions of complexity used in the rest of the book. Two examples: (a) Valiant's result that the permanent is complete for $\mathbf{AlgNP}_{/\mathbf{poly}}$; (b) complexity classes defined using the BSS model of TMs using complex-valued inputs have connections to standard complexity classes.

Chapter notes and history

It is natural to consider the minimum number of arithmetic operations required to produce a desired output from the input. The first formalization of this question appears to be by A. Scholz in 1937 [Sch37], roughly contemporaneous with Turing's work on undecidability. The notion of a straight line program goes back to Ostrowski's [Ost54] investigation of the optimality of Horner's rule for evaluating a polynomial. The formal definitions of the straight line program and algebraic computation tree models first appear in Strassen [Str72] and Rabin [Rab72] respectively, though Rabin restricted attention to linear functions instead of general polynomials. The work of Strassen in the 1960s and 1970s did much to establish algebraic complexity theory. Volume 2 of Knuth's book from 1969 [Knu69] gives a nice survey of the state of knowledge at that time. Algebraic computation trees attracted attention in computational geometry, where three-way branching on a linear function can be interpreted as the query that asks wheter a point $x \in \mathbb{R}^n$ is on the hyperplane defined by the linear function, or to the left/right of it.

The classes $\mathbf{AlgP}_{/\mathbf{poly}}$and $\mathbf{AlgNP}_{/\mathbf{poly}}$were defined by Valiant [Val79a], though he used the term "**P**-definable" for $\mathbf{AlgNP}_{/\mathbf{poly}}$and $\mathbf{AlgP}$ for $\mathbf{AlgP}_{/\mathbf{poly}}$. Later works also used the names **VNP** and **VP** for these classes. The theory was fleshed out by Skyum and Valiant [SV85], who also gave an extension of Valiant's theory of completeness via projections to the standard **NP** class. This extension relies on the observation that the Cook-Levin reduction itself is a projection reduction. One interesting consequence of this extended theory is that it shows $\mathbf{AlgP}_{/\mathbf{poly}}? = \mathbf{AlgNP}_{/\mathbf{poly}}$ must necessarily be resolved before resolving $\mathbf{P}? = \mathbf{NP}$.

The **NC** algorithm for computing the determinant mentioned in Section 16.1.4 is due to Csanky [Csa76]. It works for fields of characteristic 0. Many generalizations of this algorithm exist. The fact that determinant has algebraic formulae of size $2^{\text{poly}(\log n)}$ is due to Valiant et al. [VSBR81]. In fact, they show a general transformation of any algebraic circuit of size S computing a polynomial f to a circuit computing f of depth $O(\log S \log \deg(f))$. (The depth of a circuit is, as usual, the length of longest path from input to output in the graph).

The problem of proving lower bounds on algebraic computation trees has a long history, and Ben-Or's theorem (Theorem 16.19) falls somewhere in the middle of it. More recent work by Bjorner et al [BLY92] and Yao [Yao94] shows how to prove lower bounds for cases where the $\#(W)$ parameter is small. These rely on other topological parameters associated with the set, such as Betti numbers.

A general reference on algebraic complexity (including algorithms and lower bounds) is the 1997 book by Bürgisser et al. [BCS97]. A good modern survey of computer algebra is the book by von zur Gathen and Gerhad [vzGG99].

One important topic not covered in the chapter is Strassen's lower bound technique for algebraic circuits based upon the notion of *degree* of an algebraic variety. It leads to optimal $\Omega(n \log n)$ lower bounds for several problems. A related topic is the famous Baur-Strassen lemma which shows that one can compute the partial derivatives of f in the same resources required to compute f. See [BCS97] for details on both.

The best survey of results on the BSS model is the book by Blum et al. [BCSS97]. The question of decidability of the Mandelbrot fractal set is from Roger Penrose's controversial criticism of Artificial Intelligence [Pen89]. The full story of this debate is long, but in a nutshell, one of the issues Roger Penrose raised was that humans have an intuitive grasp of many things that seem beyond the capabilities of the Turing machine model. He mentioned computation over $\mathbb{R}$ — exemplified by our definition of the Mandelbrot set— as an example. He suggested that such mathematical objects are beyond the purview of computer science —he suggested that one cannot talk about the decidability of such sets. The BSS work shows that actually such questions can be easily studied using simple variations of the TM model. A careful evaluation of the BSS model appears in a recent survey of Braverman and Cook [BC06], who point out some of its conceptual limitations, and propose modeling real computations using a bit-based model (i.e., closer to the standard TM).

Exercises

16.1 Show for every finite field $\mathbb{F}$ there is a constant c such that for every Boolean function $f : \{0,1\}^n \to \{0,1\}$ with Boolean circuit complexity S, the size of the smallest algebraic circuit over $\mathbb{F}$ that computes S is between S/c and $c \cdot S$.

16.2 Sketch the $O(n \log n)$ size straight line program for the fast fourier transform.

16.3 Sketch a algorithm for multiplying two degree n univariate polynomials with complex coefficients in $O(n \log n)$ operations ($+$ and $\times$) over the complex numbers.

16.4 ([Str69])

(a) Prove that for every $\omega > 2$, if there *exists* $k \in \mathbb{N}$ and an algebraic straight-line program Π_k that computes the matrix multiplication of $k \times k$ matrices using at most k^ω multiplication gates, then for *every* $n \in \mathbb{N}$ there is an algebraic straight program of size $O(n^\omega)$ that computes matrix multiplication for $n \times n$ matrices. H463

(b) Prove that there exists an algebraic straight-line program using 7 multiplication gates that computes the matrix multiplication of 2×2 matrices. Conclude that there is an algebraic straight-line program of size $O(n^{2.81})$ for multiplying $n \times n$ matrices. H463

16.5 Prove that any function that can be computed by an algebraic circuit of depth d can be computed by an algebraic *formula* of size $O(2^d)$.

16.6 ([Berch]) In this exercise we show a small depth polynomial-size algebraic circuit for the determinant. Such a circuit can also be obtained by following the Gaussian elimination and using Stassen's [Str73] technique of eliminating division operations.

(a) Show that there is an $O(n^3)$-size algebraic circuit of $O(\log n)$ depth to multiply two $n \times n$ matrices.

(b) Show that for every $i \in [n]$ there is an $O(n^3)$-size algebraic circuit of depth $O(\log^2 n)$ that computes M^i for any $n \times n$ matrix M.

(c) Recall that for a matrix A, the characteristic polynomial p_A is defined as $p_A(x) = \det(A - xI)$, prove that if $A = \begin{smallmatrix} A_{1,1} & \mathbf{r} \\ \mathbf{c} & M \end{smallmatrix}$, (where M is an $(n-1)\times(n-1)$ matrix, $\mathbf{r}$ is an $n-1$-dimensional row vector, $\mathbf{c}$ is an $n-1$-dimensonal column vector) then $p_A = Cp_M$ (treating p, q as column vectors ordered from highest to lowest coefficient), where C is the following $(n-1)\times n$ matrix:

$$C_{i,j} = \begin{cases} 0 & i < j \\ 1 & i = j \\ -A_{1,1} & i = j+1 \\ -\mathbf{r}M^{i-j-2}\mathbf{c} & i \geq j+2 \end{cases}$$

H463

(d) Prove that the determinant can be computed by an algebraic circuit of size $\text{poly}(n)$ and depth $O(\log^2 n)$. (By making various optimizations the size of the circuit can be made as small as $O(n^{\omega+1+\epsilon})$ for every constant $\epsilon > 0$, where ω is the number such that there is an $O(n^\omega)$-sized $O(\log n)$ depth algebraic circuit for matrix multiplication.)

16.7 Prove Lemma 16.6

16.8 Suppose we are interested in the problem of computing the number of Hamilton cycles in graphs. Model this as an algebraic computational problem and show that this function is in $\mathbf{AlgNP}_{/\mathbf{poly}}$. H463

16.9 Show that the BSS model over the field GF(2) is equivalent to the standard TM model.

16.10 (Shamir [Sha79]) Show that any computational model that allows arithmetic (including "mod" or integer division) on arbitrarily large numbers can factor any given integer n in $\text{poly}(\log n)$ time. H463

16.11 Show that if a function $f : \mathbb{R}^n \to \{0,1\}$ can be computed in time T on algebraic TM then it has an algebraic computation tree of depth $O(T)$.

16.12 Prove that if we give the BSS model (over $\mathbb{R}$) the power to test "$a > 0$?" with arbitrary precision, then all of $\mathbf{P}_{/\mathbf{poly}}$ can be decided in polynomial time. (Hint: the machine's "program" can contain a constant number of arbitrary real numbers.)

Part III

Advanced topics

Chapter 17

Complexity of counting

> *"It is an empirical fact that for many combinatorial problems the detection of the existence of a solution is easy, yet no computationally efficient method is known for counting their number.... for a variety of problems this phenomenon can be explained."*
> L. Valiant 1979

The class **NP** captures the difficulty of finding *certificates*. However, in many contexts, one is interested not just in a single certificate, but actually in counting the *number* of certificates. This chapter studies **#P**, (pronounced "sharp p"), a complexity class that captures this notion.

Counting problems arise in diverse fields such as such as statistical estimation, statistical physics, network design, and economics, often in situations having to do with estimations of probability. Counting problems are also studied in a field of mathematics called *enumerative combinatorics*, which tries to obtain closed-form mathematical expressions for counting problems. To give an example, in 1847 Kirchoff showed how the resistance of a network can be determined by counting the number of *spanning trees* in it, for which he gave a formula involving a simple determinant computation. Results in this chapter will show that for many other natural counting problems, such efficiently computable expressions are unlikely to exist.

In Section 17.1 we give an informal introduction to counting problems and how they arise in statistical estimation. We also encounter an interesting phenomenon: a counting problem can be difficult even though the corresponding decision problem is easy.

Then in Section 17.2 we initiate a formal study of counting problems by defining the class **#P**. The quintessential problem in this class is #SAT, the problem of counting the number of satisfying assignments to a Boolean formula. We then introduce **#P**-completeness and prove the **#P**-completeness of an important problem, computing the *permanent* of a $0,1$ matrix.

We then consider whether **#P** is related to the concepts we have studied before. Section 17.4 shows a surprising result of Toda: an oracle for #SAT can be used to solve every problem in **PH** in polynomial time. The proof involves an interesting probabilistic argument, even though the statement of the Theorem involves no probabilities.

17.1 Examples of Counting Problems

In counting problems the output is a *number* rather than just $0,1$ as in a decision problem. Counting analogues of the usual decision problems are of great interest in complexity theory. We list a couple of examples.

- #CYCLE is the problem of computing, given a directed graph G, the number of simple cycles in G. (A simple cycle is one that does not visit any vertex twice.) The

corresponding decision problem of deciding if the graph has a cycle is trivial and can be solved in linear time.

- #SAT is the problem of computing, given a Boolean formula ϕ, the number of satisfying assignments for ϕ. Here of course, the corresponding decision problem is **NP**-complete, so presumably the counting problem is even harder.

17.1.1 Counting problems and probability estimation

Counting problems often arise in situations where we have to do estimations of probability.

Example 17.1
In the GraphReliability problem we are given a directed graph on n nodes. Suppose we are told that each node can fail with probability $1/2$ and want to compute the probability that node 1 has a path to n.
A moment's thought shows that under this simple node failure model, the remaining graph is uniformly chosen at random from all induced subgraphs of the original graph. Thus the correct answer is

$$\frac{1}{2^n}(\text{number of subgraphs in which node 1 has a path to } n.)$$

Again, it is trivial to determine the *existence* of a path from 1 to n.

Example 17.2 *(Maximum Likelihood Estimation in Bayes Nets)*
Suppose some data is generated by a probabilistic process but some of the data points are missing. This setting is considered in a variety of fields including machine learning and economics. In *maximum likelihood estimation* we try to come up with the most likely value of the missing data points.
A simple model of data generation is *Bayes Net*, and we restrict attention to a particularly simple example of a Bayes Net. There are n *hidden* variables $x_1, \ldots, x_n \in \{0, 1\}$, whose values are picked be nature by tossing n fair random coins independently. These values are *hidden* from us. The values actually available to us are m *visible* random variables $y_1, y_2, \ldots, y_n$, each of which is an OR of up to 3 hidden variables or their negations. We observe that all of $y_1, y_2, \ldots, y_n$ are 1. We now have to estimate the *a posteriori* probabilty that x_1 is 1.
Of course, a complexity theorist can immediately realize that an OR of 3 literals is a $3CNF$ clause, and thus recast the problem as follows: we are given a $3CNF$ Boolean formula with n variables and m clauses. What is the fraction of satisfying assignments that have $x_1 = 1$? This problem turns out to be equivalent to #SAT (see Exercise 17.1 and also the Chapter Notes).

Example 17.3 *(Estimation problems in statistical physics)*
One of the most intensively studied models in statistical physics is the *Ising model*, introduced in the 1920s by Lenz and Ising to study ferromagnetism. An instance of the model is given by a set of n *sites*, a set of *interaction energies* V_{ij} for each unordered pair of sites i, j, a *magnetic field intensity* B, and an *inverse temperature* β. A *configuration* of the system defined by these parameters is one of 2^n possible assignments σ of ± 1 spins to each site. The *energy* of a configuration σ is given by the *Hamilton* $H(\sigma)$ defined by:

$$H(\sigma) = -\sum_{\{i,j\}} V_{ij}\sigma_i\sigma_j - B\sum_k \sigma_k. \tag{1}$$

The important part of this sum is the first term, consisting of a contribution from pairs of sites. The importance of this expression comes from the *Gibbs distribution*, according to which the probability that the system is in configuration σ is proportional $\exp(-\beta H(\sigma))$. This implies that the probability of configuration σ is $1/Z \times \exp(-\beta H(\sigma))$, where the normalizing factor Z, called the *partition function* of the system, is

$$Z = \sum_{\sigma \in \{1,1\}^n} \exp(-\beta H(\sigma)).$$

Computing the exact partition function also turns out to be equivalent to #SAT (see Chapter notes).

17.1.2 Counting can be harder than decision

What is the complexity of #SAT and #CYCLE? Clearly, if #SAT has a polynomial-time algorithm then SAT $\in \mathbf{P}$ and so $\mathbf{P} = \mathbf{NP}$. How about #CYCLE? The corresponding decision problem —given a directed graph decide if it has a cycle—can be solved in linear time by breadth-first-search. The next theorem suggests that the counting problem may be much harder.

Figure 17.1 Reducing Ham to #CYCLE: by replacing every edge in G with the above gadget to obtain G', every simple cycle of length ℓ in G becomes $(2^m)^\ell$ simple cycles in G'.

Theorem 17.4 *If* #CYCLE *has a polynomial-time algorithm, then* $\mathbf{P} = \mathbf{NP}$. ♢

PROOF: We show that if #CYCLE can be computed in polynomial time, then Ham $\in \mathbf{P}$, where Ham is the **NP**-complete problem of deciding whether or not a given digraph has a Hamiltonian cycle (i.e., a simple cycle that visits all the vertices in the graph). Given a graph G with n vertices, we construct a graph G' such that G has a Hamiltonian cycle iff G' has at least n^{n^2} cycles.

To obtain G', replace each edge (u, v) in G by the gadget shown in Figure 17.1. The gadget has $m = n \log n$ levels. It is an acyclic digraph, so cycles in G' correspond to cycles in G. Furthermore, there are 2^m directed paths from u to v in the gadget, so a simple cycle of length ℓ in G yields $(2^m)^\ell$ simple cycles in G'.

Notice, if G has a Hamiltonian cycle, then G' has at least $(2^m)^n > n^{n^2}$ cycles. If G has no Hamiltonian cycle, then the longest cycle in G has length at most $n-1$. The number of cycles is bounded above by n^{n-1}. So G' can have at most $(2^m)^{n-1} \times n^{n-1} < n^{n^2}$ cycles. ■

17.2 The class #P

We now try to capture the above counting problems using the complexity class **#P**. Note that it contains functions whose output is a natural number, and not just 0/1.

Definition 17.5 *(#P)*
A function $f : \{0,1\}^* \to \mathbb{N}$ is in **#P** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$:

$$f(x) = \left|\left\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\right\}\right| .$$

Of course, the definition implies that $f(x)$ can be expressed using poly($|x|$) bits.

As in case of **NP**, we can also define **#P** using non-deterministic TMs. That is, **#P** consists of all functions f such that $f(x)$ is equal to the number of paths from the initial configuration to an accepting configuration (in brief, "accepting paths") in the *configuration graph* $G_{M,x}$ of a polynomial-time NDTM M on input x (see Section 4.1.1). Clearly, all the counting problems in Section 17.1 fall in this class.

The big open question regarding **#P** is whether all problems in this class are efficiently solvable. We define **FP** to be the set of functions from $\{0,1\}^*$ to $\{0,1\}^*$ equivalently, from $\{0,1\}^*$ to $\mathbb{N}$) computable by a deterministic polynomial-time Turing machine, is the analog of efficiently computable functions (i.e., the analog of **P** for functions with more than one bit of output). Then the question is whether **#P** = **FP**. Since computing the number of certificates is at least as hard as finding out whether a certificate exists, if **#P** = **FP** then **NP** = **P**. We do not know whether the other direction also holds: whether **NP** = **P** implies that **#P** = **FP**. We do know that if **PSPACE** = **P** then **#P** = **FP**, since counting the number of certificates can be done in polynomial space.

17.2.1 The class PP: decision-problem analog for #P.

Similar to the case of search problems, even when studying counting complexity, we can often restrict our attention to *decision problems*. The following is one such problem:

Definition 17.6 *(**PP**)* A language L is in **PP** if there exists a polynomial-time TM M and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \left|\left\{u \in \{0,1\}^{p(|x|)} : M(x,u) = 1\right\}\right| \geq \frac{1}{2} \cdot 2^{p(|x|)} \qquad \diamondsuit$$

That is, for x to be in L it does not need just one certificate (as is the case in **NP**— see Definition 2.1) but rather a *majority* of certificates.

Lemma 17.7 **PP** = **P** $\Leftrightarrow$ **#P** = **FP** $\diamondsuit$

PROOF: The non-trivial direction is that if **PP** = **P** then **#P** = **FP**. Let f be a function in **#P**. Then for every input x, $f(x)$ there is some polynomial-time TM M such that $f(x)$ the number $\#_M(x)$ of strings $u \in \{0,1\}^m$ such that $M(x,u) = 1$, where m is some polynomial in $|x|$ that is the length of certificates that M takes.

For every two TM's M_0, M_1 taking m-bit certificates denote in this proof by "$M_0 + M_1$" the TM M' that takes $n+1$ bit certificate where $M'(x, bu) = M_b(x,u)$. Then $\#_{M_0+M_1}(x) = \#_{M_0}(x) + \#_{M_1}(x)$. Also, for $N \in \{0..2^m\}$, we denote by M_N the TM that on input x, u outputs 1 iff the string u, when considered as a number, is smaller than N. Clearly $\#_{M_N}(x) = N$. If **PP** = **P** then we can determine in polynomial time if

$$\#_{M_N+M}(x) = N + \#_M(x) \geq 2^m . \qquad (2)$$

Thus, to compute $\#_M(x)$ we can use binary search to find the smallest N that makes satisfy (2), which will equal $2^m - \#_M(x)$. ■

Intuitively speaking, **PP** corresponds to computing the most significant bit of functions in **#P**: if the range is $[0, N-1]$ we have to decide whether the function value is $\geq N/2$.

One can also consider decision problems corresponding to the *least* significant bit; this is called $\oplus\mathbf{P}$ (see Definition 17.15 below).

Another related class is **BPP** (see Chapter 7), where we are guaranteed that the fraction of accepting paths of an NDTM is either $\geq 2/3$ or $\leq 1/3$ and have to determine which is true. But **PP** seems very different from **BPP** because the fraction of accepting paths in the two cases could be $\geq 1/2$ or $\leq 1/2 - \exp(-n)$, and the lack of a "gap" between the two cases means that random sampling would require $\exp(n)$ trials to distuinguish between them. (By contrast, the sampling problem for **BPP** is easy, and we even think it can be replaced by a deterministic algorithm; see Chapter 20.)

17.3 #P completeness.

Now we define **#P**-completeness. Loosely speaking, a function f is **#P**-complete if it is in **#P** and a polynomial-time algorithm for f implies that $\#\mathbf{P} = \mathbf{FP}$. To formally define **#P**-completeness, we use the notion of *oracle* TMs, as defined in Section 3.4. Recall that a TM M has *oracle access* to a language $O \subseteq \{0,1\}^*$ if it can make queries of the form "Is $q \in O$?" in one computational step. We generalize this to non-Boolean functions by saying that M has oracle access to a function $f : \{0,1\}^* \to \{0,1\}^*$, if it is given access to the language $O = \{\langle x, i\rangle : f(x)_i = 1\}$. We use the same notation for functions mapping $\{0,1\}^*$ to $\mathbb{N}$, identifying numbers with their binary representation as strings. For a function $f : \{0,1\}^* \to \{0,1\}^*$, we define $\mathbf{FP}^f$ to be the set of functions that are computable by polynomial-time TMs that have oracle access to a function f.

Definition 17.8 A function f is **#P**-*complete* if it is in **#P** and every $g \in \#\mathbf{P}$ is in $\mathbf{FP}^f$ ♢

If $f \in \mathbf{FP}$ then $\mathbf{FP}^f = \mathbf{FP}$. Thus the following is immediate.

Proposition 17.9 *If f is **#P**-complete and $f \in \mathbf{FP}$ then $\mathbf{FP} = \#\mathbf{P}$.* ♢

Counting versions of many **NP**-complete languages such as 3SAT,Ham, and CLIQUE naturally lead to **#P**-complete problems. We demonstrate this with #SAT:

Theorem 17.10 *#SAT is **#P**-complete* ♢

Proof: Consider the Cook-Levin reduction from any L in **NP** to SAT we saw in Section 2.3. This is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in L \Leftrightarrow f(x) \in \mathsf{SAT}$. However, the proof that the reduction works actually gave us more information than that. In Section 2.3.6 we saw that it provides a *Levin reduction*, by which we mean the proof showed a way to transform a *certificate* that x is in L into a certificate (i.e., satisfying assignment) showing that $f(x) \in \mathsf{SAT}$, and also vice versa (transforming a satisfying assignment for $f(x)$ into a witness that $x \in L$).

In fact, for the reduction in question, this mapping from the certificates of x to the assignments of $f(x)$ is one-to-one and onto (i.e., a bijection). Thus the number of satisfying assignments for $f(x)$ is equal to the number of certificates for x. Such reductions are called *parsimonious*. (More generally, the definition allows the witness mapping to be k-to-1 or 1-to-k, so the number of witnesses for the two problems are still the same up to scaling by k. ■

As shown below, there are **#P**-complete problems for which the corresponding decision problems are in fact in **P**.

17.3.1 Permanent and Valiant's Theorem

Now we study another problem. The *permanent* of an $n \times n$ matrix A is defined as

$$\mathsf{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)} \tag{3}$$

where S_n denotes the set of all permutations of n elements. Recall that the expression for the determinant is similar

$$\det(A) = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$$

except for an additional "sign" term.[1] This similarity does not translate into computational equivalence: the determinant can be computed in polynomial time, whereas computing the permanent seems much harder, as we see below. (For another perspective on the hardness of the permanent see Chapter 16.)

The permanent function can also be interpreted combinatorially. First, suppose the matrix A has each entry in $\{0,1\}$. Then it may be viewed as the adjacency matrix of a bipartite graph $G(X,Y,E)$, with $X = \{x_1,\ldots,x_n\}$, $Y = \{y_1,\ldots,y_n\}$ and $\{x_i,y_j\} \in E$ iff $A_{i,j} = 1$. For each permutation σ the term $\prod_{i=1}^{n} A_{i,\sigma(i)}$ is 1 iff σ is a *perfect matching* (which is a set of n edges such that every node is in exactly one edge). Thus if A is a $0,1$ matrix then $\mathsf{perm}(A)$ is simply the number of perfect matchings in the corresponding graph G. Note that the whether or not a perfect matching *exists* can be determined in polynomial time. In particular, computing $\mathsf{perm}(A)$ is in $\#\mathbf{P}$. If A is a $\{-1,0,1\}$ matrix, then $\mathsf{perm}(A) = \left|\{\sigma : \prod_{i=1}^{n} A_{i,\sigma(i)} = 1\}\right| - \left|\{\sigma : \prod_{i=1}^{n} A_{i,\sigma(i)} = -1\}\right|$. Thus one can make two calls to a $\#\mathsf{SAT}$ oracle to compute $\mathsf{perm}(A)$. Finally, if A has general integer entries (possibly negative) the combinatorial view of $\mathsf{perm}(A)$ is as follows. Consider matrix A as the the adjacency matrix of a weighted n-node complete digraph with self loops and 0 edge weights allowed. Associate with each permutation σ a *cycle cover*, which is a subgraph on the same set of vertices but only a subset of the original edges, where each node has in-degree and out-degree 1. Such a subgraph must decompose into disjoint cycles. The *weight* of the cycle cover is the product of the weights of the edges in it. Then $\mathsf{perm}(A)$ is equal to the sum of weights of all possible cycle covers. Using this observation one can show that computing the permanent is in $\mathbf{FP}^{\#\mathsf{SAT}}$ (see Exercise 17.2).

The next theorem came as a surprise to researchers in the 1970s, since it implies that if $\mathsf{perm} \in \mathbf{FP}$ then $\mathbf{P} = \mathbf{NP}$. Thus, unless $\mathbf{P} = \mathbf{NP}$, computing the permanent is much more difficult then computing the determinant.

Theorem 17.11 *(Valiant's Theorem* [Val79b]*)*
perm *for* $0,1$ *matrices is* $\#\mathbf{P}$*-complete.*

Theorem 17.11 involves very clever gadget constructions. As warmup, we introduce a simple idea.

Convention: In drawings of gadgets in the rest of the chapter, the underlying graph is a complete digraph, but edges that are missing from the figure have weight 0 and hence can be ignored while considering cycle covers. Unmarked edges have weight +1. We will also sometimes allow *parallel edges* (with possibly different weights) from a node to another node. This is not allowed per se in the definition of permanent, where there is a single edge of weight $A_{i,j}$ from i to j. But since we are describing reductions, we have the freedom to later replace each parallel edge of weight w by a path of length 2 whose two edges of weight 1 and w. This just requires adding a new node (not connected to anything else) in the middle of each parallel edge.

---

[1] It is known that every permutation $\sigma \in S_n$ can be represented as a composition of transpositions, where a transposition is a permutation that only switches between two elements in $[n]$ and leaves the other elements intact (one proof for this statement is the Bubblesort algorithm). If $\tau_1,\ldots,\tau_m$ is a sequence of transpositions such that their composition equals σ, then the *sign* of σ is equal to $(-1)^m$. It can be shown that the sign is well-defined in the sense that it does not depend on the representation of σ as a composition of transpositions. NEED TO MOVE THIS WHERE FIRST USED.

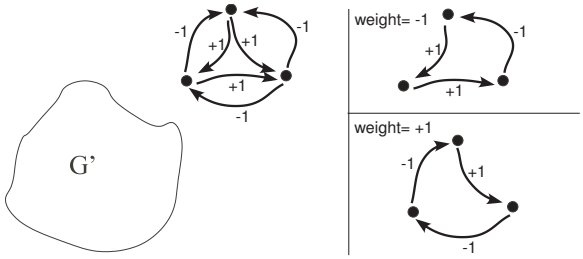


Figure 17.2 The above graph G consists of two vertex-disjoint subgraphs, one called G' and the other is the "gadget" shown. The gadget has only two cycle covers of nonzero weight, namely, weight -1 and 1. Hence the total weight of cycle covers of G is zero regardless of the choice of G', since for every cycle cover of weight w in G', there exist two covers of weight $+w$ and $-w$ in the graph G.

Example 17.12
Consider the graph in Figure 17.2. Even without knowing what the subgraph G' is, we show that the permanent of the whole graph is 0. For each cycle cover in G' of weight w there are exactly two cycle covers for the three nodes, one with weight $+w$ and one with weight $-w$. Any non-zero weight cycle cover of the whole graph is composed of a cycle cover for G' and one of these two cycle covers. Thus the sum of the weights of all cycle covers of G is 0.

Proof of Valiant's Theorem (Theorem 17.11): We reduce the **#P**-complete problem #3SAT to perm. Given a Boolean $3CNF$ formula ϕ with n variables and m clauses, first we shall show how to construct an integer matrix, or equivalently, a weighted digraph G', with some negative entries such that $\mathsf{perm}(G') = 4^{3m} \cdot (\#\phi)$, where $\#\phi$ stands for the number of satisfying assignments of ϕ. Later we shall show how to to get a digraph G with weights $0, 1$ from G' such that knowing $\mathsf{perm}(G)$ allows us to compute $\mathsf{perm}(G')$.

The main idea is that our construction will result in two kinds of cycle covers in the digraph G': those that correspond to satisfying assignments of ϕ (we will make this precise) and those that don't. Using reasoning similar to that used in Example 17.12, we will use negative weights to ensure that the contribution of the cycle covers that do not correspond to satisfying assignments cancels out. On the other hand, we will show that each satisfying assignment contributes 4^{3m} to $\mathsf{perm}(G')$, and so $\mathsf{perm}(G') = 4^{3m} \cdot (\#\phi)$.

To construct G' from ϕ, we combine three kinds of gadgets shown in Figure 17.3. There is a variable gadget for each variable, a clause gadget for each clause, and a way to connect them using gadgets called *XOR* gadgets. All are shown in Figure 17.3.

XOR gadget. Suppose we have a weighted digraph H and wish to ensure for some pair of edges $\overrightarrow{u\,u'}$ and $\overrightarrow{v\,v'}$, *exactly one* of these edges is present in any cycle cover that counts towards the final sum. To do so, we can construct a new digraph H' in which this pair of edges is replaced by the XOR gadget of Figure 17.3.

Every cycle cover of H of weight w that uses exactly one of the edges $\overrightarrow{u\,u'}$ and $\overrightarrow{v\,v'}$ is mapped to a set of cycle covers in H' whose total weight is $4w$ (i.e., the set of covers that enter the gadget at u and exit at u' or enter it at v and exit it at v'), while all the other cycle covers of H' have total weight 0 (this uses reasoning similar to Example 17.12, see Exercise 17.3).

Variable gadget. The variable gadget has both *internal edges* (that are not involved with any other part of the graph) and *external edges* (that will be connected via XOR gadgets to other edges of the graph). We partition the external edges to "true" edges and "false" edges. The variable gadget has only two possible cycle covers, corresponding to an assignment of 0 or 1 to that variable. Assigning 1 corresponds to using cycle taking all the "true" external edges, and covering all other vertices

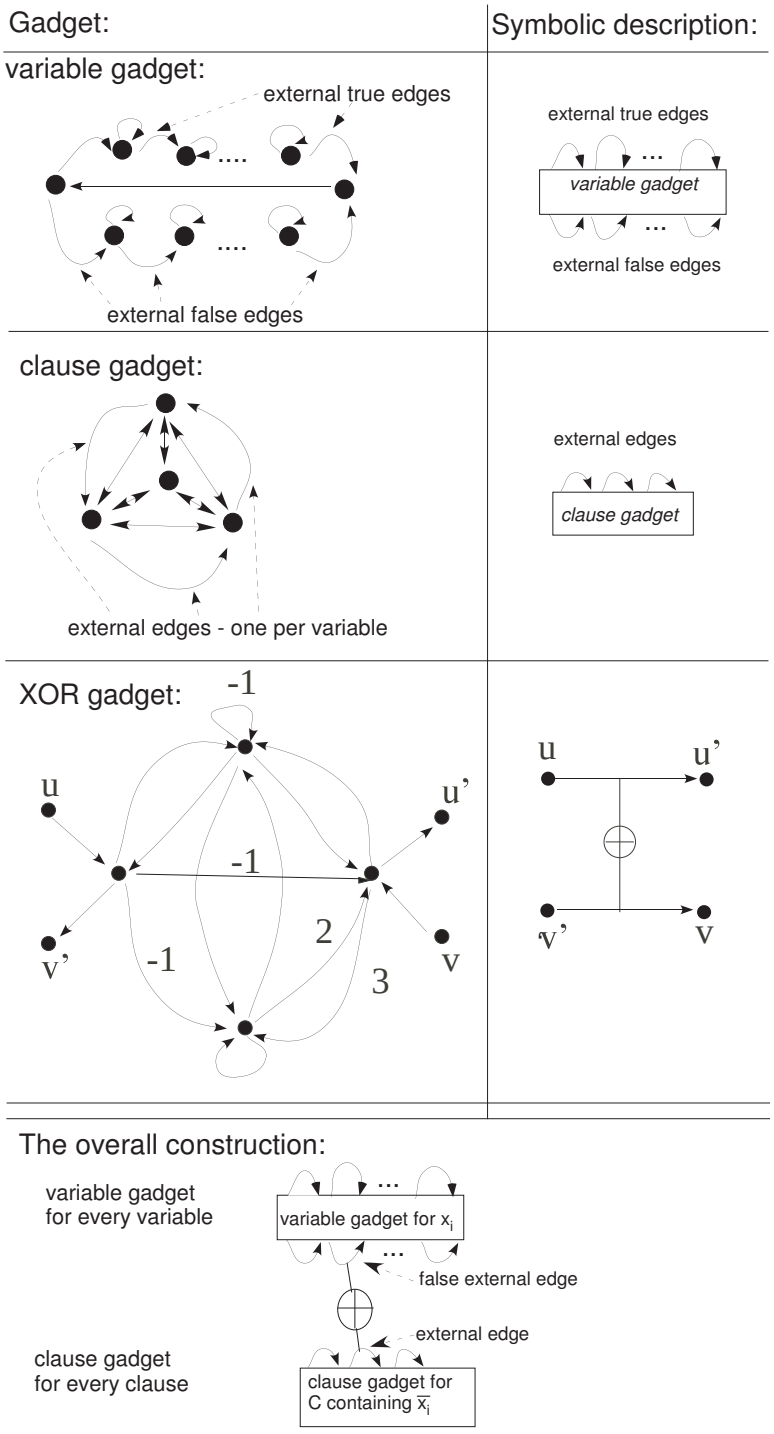


Figure 17.3 The gadgets used in the proof of Valiant's Theorem (Theorem 17.11).

with self loops. Similarly, assigning 0 correspond to taking all the "false" external edges. Each external edge of a variable gadget is associated with a clause in which the variable appears— a true edge is associated with a clause where the variables appears in a positive (non-negated) form, while a false edge is associated with a clause where the variable appears in negated form.

Clause gadget. The clause gadget consists of four vertices. The edges labeled with "external edge" are the only ones that will connect to the rest of the graph. Specifically, the external edge will connect via a XOR gadget with an external edge of a variable gadget corresponding to one of the three variables in the clause. The only possible cycle covers of the Clause gadget are those that omit at least one external edge. Also for a given proper subset of the three external edges there is a unique cycle cover of weight 1 that contains them.

The overall construction is shown at the bottom of Figure 17.3. If a clause C contains the variable x then we connect the corresponding external edge of C's gadget with the corresponding (true) external edge of x's gadget using the XOR gadget (if C contains the negation of x then the edge corresponding to X in x's gadget will be a false external edge).

To analyze this construction, note that because each variable gadget has exactly two cycle covers corresponding to the 0 and 1 assignments, there is a one-to-one correspondence between assignments to the variables $x_1, \ldots, x_n$ and the cycle covers of the variable gadgets of G'. Now for every such assignment $\mathbf{x}$, let $\mathcal{C}_{\mathbf{x}}$ denote the set of cycle covers in G' that cover the variable gadgets according to the assignment $\mathbf{x}$. That is, we cover variable x_i's gadget using the true external edges if $x_i = 1$ and cover it using the false external edges if $x_i = 0$. Let $w(\mathbf{x})$ denote the total weight of assignments in $\mathcal{C}_{\mathbf{x}}$. It suffices to show that if $\mathbf{x}$ is a satisfying assignment then $w(\mathbf{x}) = 4^{3m}$, and if $\mathbf{x}$ is not satisfying then $w(\mathbf{x}) = 0$.

Indeed, by the properties of the XOR gadget and the way we connected clauses and variables, if $x_i = 1$ then in cycle covers in $\mathcal{C}_{\mathbf{x}}$ the corresponding external edge has to be *omitted* in the gadget of every clause in which x_i appears positively, and has to be *included* for clauses containing x_i negatively, and similarly if $x_i = 0$ then the corresponding edges have to be included or omitted accordingly. (More accurately, covers without this property do not contribute toward the final sum.) But because every cover for the clause gadget has to omit at least one external edge, we see that unless every clause has a literal that evaluates to "true" in the assignment $\mathbf{x}$ (i.e., unless $\mathbf{x}$ satisfies ϕ) then the total weight of covers in $\mathcal{C}_{\mathbf{x}}$ is zero. If $\mathbf{x}$ does satisfy φ then this total weight will be 4^{3m} (since $\mathbf{x}$ determines a unique cycle cover for all clause gadgets that passes through the XOR gadget exactly $3m$ times). Thus $\mathsf{perm}(G') = 4^{3m}(\#\phi)$.

Reducing to the case of $0,1$ matrices. This transformation goes in two steps. First we create a graph G'' with edge weights in $\{-1, 0, 1\}$ and the same permanent as G'. Then we remove negative weights to create a graph G whose permanent contains enough information to compute $\mathsf{perm}(G') = \mathsf{perm}(G'')$. The transformations may blow up the number of vertices by a factor $O(nL^2 \log n)$, where L is the number of *bits* required to describe all the weights in G'.

Notice, an edge whose weight is a power of 2, say 2^k, can be replaced by a path (consisting of new nodes not connected to anything else) of k edges, each of weight 2. Similarly, an edge whose weight is $2^k + 2^{k'}$ can be replaced by two parallel paths of weight 2^k and $2^{k'}$ respectively. Combining these observations, we can replace an edge whose weight is not a power of 2 by a set of parallel paths determined by its binary expansion; the total number of nodes in these paths is quadratic in the number of bits required to represent the original weight. This gives a graph G'' with weights in $\{-1, 0, 1\}$ and at most $O(L^2)$ new vertices.

To get rid of the negative weights, we use modular arithmetic. The permanent of an n-vertex graph with edge weights in $\{\pm 1\}$ is a number x in $[-n!, +n!]$ and hence it suffices to compute this permanent modulo $2^m + 1$ where $m = n^2$. But $-1 \equiv 2^m \pmod{2^m + 1}$, so the permanent modulo $2^m + 1$ is unchanged if we replace all weight -1 edges with edges of weight 2^m. Such edges can be replaced by an unweighted subgraph of size $O(m) = O(n \log n)$ as before. Thus we obtain a graph G with all weights $0, 1$ and whose permanent can be used

to compute the original permanent (specifically, by taking the remainder modulo $2^m + 1$). The number of new vertices is at most $O(nL^2 \log n)$. ■

17.3.2 Approximate solutions to #P problems

Since computing exact solutions to **#P**-complete problems is presumably difficult, a natural question is whether we can *approximate* the number of certificates in the sense of the following definition.

Definition 17.13 Let $f:\{0,1\}^* \to \mathbb{N}$ and $\alpha < 1$. An algorithm A is an α-*approximation* for f if for every x, $\alpha f(x) \leq A(x) \leq f(x)/\alpha$. ♢

Not all **#P** problems behave identically with respect to this notion. Approximating certain problems within any constant factor $\alpha > 0$ is **NP**-hard (see Exercise 17.4). For other problems such as 0/1 permanent, there is a *Fully polynomial randomized approximation scheme* (FPRAS), which is an algorithm which, for any ϵ, δ, computes a $(1-\epsilon)$-approximation to the function with probability $1 - \delta$ (in other words, the algorithm is allowed to give an incorrect answer with probability δ) in time $\text{poly}(n, \log 1/\delta, \log 1/\epsilon)$. Such approximation of counting problems is sufficient for many applications, in particular those where counting is needed to obtain estimates for the probabilities of certain events (e.g., see our discussion of the graph reliability problem in Example 17.1). Interestingly, if $\mathbf{P} = \mathbf{NP}$ then *every* **#P** problem has an FPRAS (and in fact an FPTAS: i.e., a *deterministic* polynomial-time approximation scheme), see Exercise 17.5.

Now we explain the basic idea behind the approximation algorithm for the permanent —as well as other similar algorithms for a host of **#P**-complete problems. This is only a sketch; the Chapter notes contain additional references.

One result that underlies these algorithms is due to Jerrum, Valiant and Vazirani [JVV86]. It shows that under fairly general conditions there is a close connection (in the sense the two are interreducible in polynomial time) between

1. having an approximate formula for the size of a set S and

2. having an efficient algorithm for generating a uniform random (or approximately uniform) element of S.

The basic idea is a counting version of the *downward self-reducibility* idea we saw in Chapter 2.

Suppose we are trying to use sampling to do approximate counting and S is a subset of $\{0,1\}^n$. Let $S = S_0 \cup S_1$ where S_b is the subset of strings in S whose first bit is 1. By sampling a few random elements of S we can estimate $p_1 = |S_1| / |S|$ up to some reasonabable accuracy. Then we fix the first bit of the string, and use our algorithm recursively to estimate $|S_1|$ and multiply it by our estimate of $1/p_1$ to obtain an estimate $|S|$.

To produce a random sample from S using approximate counting, one again proceeds in a bit by bit fashion, and reverses the above argument. First we estimate $|S_1|, |S|$ and use their ratio to estimate p_1. Produce a bit b by tossing a random coin with bias p_1; i.e., if the coin comes up heads make the bit 1 else make it 0. Then make b the first bit of the sample and recursively use the same algorithm to produce a sample from S_b.

The main point is that in order to do approximate counting it suffices to draw a random sample from S. All the algorithms try to sample from S using the *Markov Chain Monte Carlo* method. One defines a connected d-uniform digraph on S, and does a random walk on this graph. Since the graph is d-regular for some d, the stationary distribution of the walk is uniform on S. Under appropriate conditions on the expansion of the graph (establishing which is usually the meat of the argument) the walk "mixes" and the sample becomes close to uniform.

17.4 Toda's Theorem: $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathsf{SAT}}$

An important open question in the 1980s concerned the relative power of the polynomial-hierarchy **PH** and the class of counting problems **#P**. Both are natural generalizations of **NP**, but it seemed that their features— alternation and the ability to count certificates, respectively — are not directly comparable to each other. Thus it came as big surprise when in 1989 Toda showed:

Theorem 17.14 *(Toda's Theorem* [Tod91]*)*
$\mathbf{PH} \subseteq \mathbf{P}^{\#\mathsf{SAT}}$.

That is, we can solve any problem in the polynomial hierarchy given an oracle to a **#P**-complete problem.

Note that we already know, even without Toda's theorem, that if $\#\mathbf{P} = \mathbf{FP}$ then $\mathbf{NP} = \mathbf{P}$ and so $\mathbf{PH} = \mathbf{P}$. However, this does not imply that any problem in **PH** can be computed in polynomial-time using an oracle to #SAT. For example, one implication of Toda's theorem is that a *subexponential* (i.e., $2^{n^{o(1)}}$-time) algorithm for #SAT will imply such an algorithm for any problem in **PH**. Such an implication is not known to hold from a $2^{n^{o(1)}}$-time algorithm for SAT.

To prove Toda's theorem, we first consider formulae with *odd* number of satisfying assignments. The following is the underlying complexity class.

Definition 17.15 A language L is in the class $\oplus\mathbf{P}$ (pronounced "parity P") iff there is a polynomial time NTM M such that $x \in L$ iff the number of accepting paths of M on input x is odd. ♢

As in the proof of Theorem 17.10, the fact that the standard **NP**-completeness reduction is parsimonious implies the following problem $\oplus$ SAT is $\oplus\mathbf{P}$-complete (under many-to-one Karp reductions):

Definition 17.16 *($\bigoplus$ quantifier and $\oplus$ SAT)*
Define the quantifier $\bigoplus$ as follows: for every Boolean formula φ on n variables. $\bigoplus_{x\in\{0,1\}^n} \varphi(x)$ is true if the number of x's such that $\varphi(x)$ is true is odd.[a] The language $\oplus$ SAT consists of all the true quantified Boolean formula of the form $\bigoplus_{x\in\{0,1\}^n} \varphi(x)$ where φ is an unquantified Boolean formula (not necessarily in CNF form).

[a] Note that if we identify true with 1 and 0 with false then $\bigoplus_{x\in\{0,1\}^n} \varphi(x) = \sum_{x\in\{0,1\}^n} \varphi(x) \pmod 2$. Also note that $\bigoplus_{x\in\{0,1\}^n} \varphi(x) = \bigoplus_{x_1\in\{0,1\}} \cdots \bigoplus_{x_n\in\{0,1\}} \varphi(x_1, \ldots, x_n)$.

$\oplus\mathbf{P}$ can be considered as the class of decision problems corresponding to the least significant bit of a **#P**-problem. One imagines that therefore it is not too powerful. For instance, it is even unclear whether we can reduce **NP** to this class. The first half of Toda's proof shows, surprisingly, a *randomized* reduction from **PH** to $\oplus$ SAT. The second half is going to be a clever "derandomization" of this reduction, and is given in Section 17.4.4.

Lemma 17.17 *(Randomized reduction from* **PH** *to* $\oplus$ SAT*)*
Let $c \in \mathbb{N}$ be some constant. There exists a probabilistic polynomial-time algorithm A that given a parameter m and any quantified Boolean formula ψ of size n with with c levels of alternations, runs in $\mathrm{poly}(n, m)$ *times and atisfies*

$$\psi \text{ is true} \Rightarrow \Pr[A(\psi) \in \oplus\,\mathsf{SAT}] \geq 1 - 2^{-m}$$
$$\psi \text{ is false} \Rightarrow \Pr[A(\psi) \in \oplus\,\mathsf{SAT}] \leq 2^{-m}.$$

Of course, to reduce **PH** to $\oplus$ SAT we better first figure out how to reduce **NP** to $\oplus$ SAT, which is already unclear. This will involve a detour into Boolean formulae with unique satisfying assignments.

17.4.1 A detour: Boolean satisfiability with unique solutions

Suppose somebody gives us a Boolean formula and promises us that it has either no satisfying assignment, or a unique satisfying assignment. Such formulae arise if we encode some classic math problems using satisfiability (see e.g. the number-theoretic DISCRETE LOG problem in Chapter 9 that is the basis of some encryption schemes). Let USAT be the language of Boolean formulae that have a unique satisfying assignment. Is it still difficult to decide satisfiability of such special instances (in other words, to answer "yes" if the formula is in USAT and "no" if the formula is in $\overline{\text{SAT}}$, and an arbitrary answer in every other case)? The next result of Valiant and Vazirani shows that if we had a polynomial-time algorithm for this problem then $\mathbf{NP} = \mathbf{RP}$. This was a surprise to most researchers in the 1980s.

Theorem 17.18 *(Valiant-Vazirani Theorem* [VV86]*)*
There exists a probabilistic polynomial-time algorithm f such that for every n-variable Boolean formula φ

$$\varphi \in \text{SAT} \Rightarrow \Pr[f(\varphi) \in \text{USAT}] \geq \tfrac{1}{8n}$$

$$\varphi \notin \text{SAT} \Rightarrow \Pr[f(\varphi) \in \text{SAT}] = 0$$

We emphasize that the conclusion in the second part is not just that $f(\varphi) \notin$ USAT but in fact $f(\varphi) \notin$ SAT.

The proof of Theorem 17.18 uses the following lemma on pairwise independent hash functions, which were introduced in Section 8.2.2:

Lemma 17.19 *(Valiant-Vazirani Lemma) Let $\mathcal{H}_{n,k}$ be a pairwise independent hash function collection from $\{0,1\}^n$ to $\{0,1\}^k$ and $S \subseteq \{0,1\}^n$ such that $2^{k-2} \leq |S| \leq 2^{k-1}$. Then,*

$$\Pr_{h \in_R \mathcal{H}_{n,k}}[\textit{there is a unique } x \in S \textit{ satisfying } h(x) = 0^k] \geq \tfrac{1}{8} \qquad \diamondsuit$$

PROOF: For every $x \in S$, let $p = 2^{-k}$ be the probability that $h(x) = 0^k$ when $h \in_R \mathcal{H}_{n,k}$. Note that for every $x \neq x'$, $\Pr[h(x)=0^k \wedge h(x')=0^k] = p^2$. Let N be the random variable denoting the number of $x \in S$ satisfying $h(x) = 0^k$. Note that $\mathsf{E}[N] = |S|p \in [\frac{1}{4}, \frac{1}{2}]$. By the inclusion-exclusion principle

$$\Pr[N \geq 1] \geq \sum_{x \in S} \Pr[h(x)=0^k] - \sum_{x<x' \in S} \Pr[h(x)=0^k \wedge h(x')=0^k] = |S|p - \binom{|S|}{2} p^2$$

and by the union bound we get that $\Pr[N \geq 2] \leq \binom{|S|}{2} p^2$. Thus

$$\Pr[N = 1] = \Pr[N \geq 1] - \Pr[N \geq 2] \geq |S|p - 2\binom{|S|}{2} p^2 \geq |S|p - |S|^2 p^2 \geq \frac{1}{8}$$

where the last inequality is obtained using the fact that $\frac{1}{4} \leq |S|p \leq \frac{1}{2}$. ■

Now we prove Theorem 17.18.

PROOF OF THEOREM 17.18: Given a formula φ on n variables, choose k at random from $\{2, \ldots, n+1\}$ and a random hash function $h \in_R \mathcal{H}_{n,k}$. Consider the statement

$$\exists_{x \in \{0,1\}^n} \varphi(x) \wedge (h(x) = 0^k)\,. \tag{4}$$

If φ is unsatisfiable then (4) is false since no x satisfies $\varphi(x)$. If φ is satisfiable, then with probability at least $1/8n$ there exists a unique assignment x satisfying (4). After all if S is the

set of satisfying assignments of φ, then with probability $1/n$, k satisfies $2^{k-2} \leq |S| \leq 2^{k-1}$, conditioned on which, with probability $1/8$, there is a unique x such that $\varphi(x) \wedge h(x) = 0^n$.

The idea of the preceding paragraph is implemented as follows. The reduction consists of using the Cook-Levin transformation to express the (deterministic) computation inside the $\exists^u$ sign in (4). Write a formula τ on variables $x \in \{0,1\}^n, y \in \{0,1\}^m$ (for $m = \mathrm{poly}(n)$) such that $h(x) = 0$ if and only if there exists a *unique* y such that $\tau(x,y) = 1$. Here the y variables come from the need to represent a TM's computation in the Cook-Levin reduction[2]. The output Boolean formula is

$$\psi = \varphi(x) \wedge \tau(x,y)\,.$$

where x, y are the variables. ■

17.4.2 Properties of $\bigoplus$ and proof of Lemma 17.17 for $\mathbf{NP}, \mathbf{coNP}$

In Lemma 17.17 the reduction is allowed to fail only with extremely low probability 2^{-m}, where m is arbitrary. If we are willing to settle for a much higher failure probability, then the Valiant-Vazirani trivially implies a reduction from **NP** to $\oplus$ SAT. Specifically, in the conclusion of Theorem 17.18 the formula has a unique satisfying assignment in the first case, and 1 is an odd number. In the second case of the conclusion the formula has no satisfying assignment, and 0 is an even number. Thus the following is a trivial corollary of Theorem 17.18

Corollary 17.20 *(Consequence of Valiant-Vazirani) There exists a probabilistic polynomial-time algorithm A such that for every n-variable Boolean formula φ*

$$\varphi \in \mathsf{SAT} \Rightarrow \Pr[A(\varphi) \in \oplus\,\mathsf{SAT}] \geq \tfrac{1}{8n}$$

$$\varphi \notin \mathsf{SAT} \Rightarrow \Pr[A(\varphi) \in \oplus\,\mathsf{SAT}] = 0 \qquad \diamondsuit$$

It is an open problem to boost the probability of $1/8n$ in the Valiant-Vazirani reduction to USAT (Theorem 17.18) to even a constant, say $1/2$. However, such a boosting is indeed possible for $\oplus$ SAT, since it turns out to be much more expressive than USAT. Let us examine some facts about the $\bigoplus$ quantifier.

For a Boolean formula φ on n variables, let $\#(\varphi)$ denote the number of satisfying assignments of φ. Given two formulae φ, ψ on variables $x \in \{0,1\}^n, y \in \{0,1\}^m$ we can construct in polynomial-time an $n+m$ variable formula $\varphi \cdot \psi$ and a $(\max\{n,m\}+1)$-variable formula $\varphi + \psi$ such that $\#(\varphi \cdot \psi) = \#(\varphi)\#(\psi)$ and $\#(\varphi + \psi) = \#(\varphi) + \#(\psi)$. Indeed, take $(\varphi \cdot \psi)(x,y) = \varphi(x) \wedge \varphi(y)$ and

$$(\varphi{+}\psi)(z) = \big((z_0 = 0) \wedge \varphi(z_1,\ldots,z_n)\big) \vee \big((z_0 = 1) \wedge (z_{m+1} = 0) \wedge \cdots \wedge (z_n = 0) \wedge \psi(z_1,\ldots,z_m)\big)\,,$$

where we are assuming $m < n$. For a formula φ, we use the notation $\varphi + 1$ to denote the formula $\varphi + \psi$ where ψ is some canonical formula with a single satisfying assignment.

Since the product of numbers is even iff one of the numbers is even, and since adding one to a number flips the parity, for every two formulae φ, ψ as above

$$\big(\bigoplus_x \varphi(x)\big) \wedge \big(\bigoplus_y \psi(y)\big) \Leftrightarrow \bigoplus_{x,y} (\varphi \cdot \psi)(x,y) \tag{5}$$

$$\neg \bigoplus_x \varphi(x) \Leftrightarrow \bigoplus_{x,z} (\varphi + 1)(x,z) \tag{6}$$

$$\big(\bigoplus_x \varphi(x)\big) \vee \big(\bigoplus_y \psi(y)\big) \Leftrightarrow \bigoplus_{x,y,z} ((\varphi+1)\cdot(\psi+1)+1)(x,y,z) \tag{7}$$

[2] For some implementations of hash functions, such as the one described in Exercise 8.4, one can construct such a formula directly without using the y variables or going through the Cook-Levin reduction.

The meaning of the observation in (6) is that $\oplus\mathbf{P}$ is closed under complementation, namely, for any φ we can write another formula ψ in polynomial time such that $\neg\bigoplus_x \varphi(x)$ is equivalent to $\bigoplus_y \psi(y)$. The meaning of observations in (5) and (7) is that a polynomial number of ANDs and ORs of $\oplus$ SAT instances can also be converted in polynomial time to a single $\oplus$ SAT instance that is equivalent.

Now we prove Lemma 17.17 for **NP** and **coNP** (i.e., when the formula φ has a single $\forall$ or $\exists$ quantifier). In fact it suffices to give a reduction from just **NP**. Since $\oplus\mathbf{P}$ is closed under complementation, that same reduction will be a probabilistic reduction of **coNP** to $\overline{\oplus\mathbf{P}}$, and hence to $\oplus\mathbf{P}$.

PROOF: (Lemma 17.17; when φ has only $\exists$ quantifier) Suppose φ is a Boolean formula, i.e., a quantified formula with one $\exists$ quantifier. The idea for reducing it to $\oplus$ SAT is the obvious one: run the reduction of Corollary 17.20 $R = O(mn)$ times, each time producing a $\bigoplus$ formula. The final formula is the OR of these formulae. If the original formula was satisfiable then this new formula is true with probability at least $1-(1-1/8n)^R = 1-2^{-m}$, and if the original formula was not satisfiable, this new formula is never true. Finally, apply the observation in (7) R times to turn this new formula into a single $\bigoplus$ formula, while possibly blowing up the formula's size by a polynomial factor. ■

17.4.3 Proof of Lemma 17.17; general case

The proof of the general case involves an induction on c, the number of quantifier alternations in φ. The base case $c = 1$ (i.e., **NP** or **coNP**) was already proved. To prove the general case, we need a more abstract version of the Valiant-Vazirani lemma, where we observe that the reduction never looks at what formula it is working with, so this formula could be an arbitrary Boolean function. (Using terminology from Chapter 3, the Valiant-Vazirani lemma *relativizes*.)

Lemma 17.21 *(Valiant-Vazirani, oblivious version) There is a probabilistic polynomial-time procedure that, given input 1^n, produces a Boolean formula $\tau(x, y)$ where x is a vector of n Boolean variables and y is also a vector of Boolean variables, such that for any Boolean function $\beta\colon\{0,1\}^n \to \{0,1\}$,*

$$\exists_{x_1}\beta(x_1) \Rightarrow \Pr[\bigoplus_{x_1,y} \tau(x_1, y) \wedge (\beta(x_1) = 1)] \geq \frac{1}{8n} \tag{8}$$

$$\neg\exists_{x_1}\beta(x_1) \Rightarrow \Pr[\bigoplus_{x_1,y} \tau(x_1, y) \wedge (\beta(x_1) = 1)] = 0\,. \tag{9}$$

♢

Now we can prove Lemma 17.17.

PROOF: (Lemma 17.17) Let φ have c quantifier alternations. As observed in Section 17.4.2, $\oplus$ SAT is closed under complementation, so we may assume wlog that the first quantifier is an $\exists$. Thus

$$\varphi = \exists_{x_1}\psi(x_1)$$

where $\psi(x_1)$ is a quantified Boolean formula with at most $c-1$ quantifier alternations, in which the variables in x_1 are free. Suppose x_1 consists of n Boolean variables. By the inductive hypothesis, there is a randomized reduction such that for each value of x_1 it produces a $\oplus$ SAT formula $\beta(x_1) = \oplus_z\psi(z, x_1)$ that is equivalent to the formula $\psi(x_1)$ with probability at least $1 - 2^{-(m+2)}$. Now imagine running the reduction of Lemma 17.21 $K = O(mn)$ times with independent random bits, and let $\tau_1(x_1, y), \tau_2(x_1, y), \ldots, \tau_K(x_1, y)$ be the Boolean formulae produced. Consider the formula

$$\alpha = \bigvee_{j=1}^{K} (\tau_j(x_1, y) \wedge \beta(x_1)).$$

If $\exists_{x_1}\beta(x_1)$ is true then according to Lemma 17.21, $\Pr[\alpha \text{ is true}] \geq 1 - (1 - 1/8n)^K = 1 - 2^{-O(m)}$. Conversely if $\exists_{x_1}\beta(x_1)$ is false then $\Pr[\alpha \text{ is true}] = 0$.

To finish, note that since the inductive hypothesis implies $\beta(x_1)$ is a $\oplus$ SAT instance, we can convert α also into a $\oplus$ SAT instance (with polynomial blowup in size) by using the transformations of Section 17.4.2. The two sources of error are: (a) the conversion of $\psi(x_1)$ into an equivalent $\oplus$ SAT instance, which by the inductive hypothesis fails with probability $2^{-(m+2)}$ and (b) the Valiant-Vazirani error when we replace $\exists_{x_1}$ to construct α as above, which also has failure probability $2^{-(m+2)}$. Thus the overall error probability is $2 \times 2^{-(m+2)}$, which is less than 2^{-m}. ■

17.4.4 Step 2: Making the reduction deterministic

Now we derandomize the randomized reduction of Lemma 17.17 to complete the proof of Toda's Theorem (Theorem 17.14). The following deterministic reduction will be the key tool.

Lemma 17.22 *There is a (deterministic) polynomial-time transformation T that, for every formula Boolean α formula $\beta = T(\alpha, 1^\ell)$ is such that*

$$\alpha \in \oplus\,\mathsf{SAT} \Rightarrow \#(\beta) = -1 \pmod{2^{\ell+1}}$$
$$\alpha \notin \oplus\,\mathsf{SAT} \Rightarrow \#(\beta) = 0 \pmod{2^{\ell+1}} \qquad \diamondsuit$$

Proof: Recall that for every pair of formulae φ,τ we defined formulas $\varphi + \tau$ and $\varphi \cdot \tau$ satisfying $\#(\varphi+\tau) = \#(\varphi) + \#(\tau)$ and $\#(\varphi \cdot \tau) = \#(\varphi)\#(\tau)$, and note that these formulae are of size at most a constant factor larger than φ, τ. Consider the formula $4\tau^3 + 3\tau^4$ (where τ^3 for example is shorthand for $\tau \cdot (\tau \cdot \tau)$). One can easily check that

$$\#(\tau) = -1 \pmod{2^{2^i}} \Rightarrow \#(4\tau^3 + 3\tau^4) = -1 \pmod{2^{2^{i+1}}} \tag{10}$$
$$\#(\tau) = 0 \pmod{2^{2^i}} \Rightarrow \#(4\tau^3 + 3\tau^4) = 0 \pmod{2)^{2^{i+1}}} \tag{11}$$

Let $\psi_0 = \alpha$ and $\psi_{i+1} = 4\psi_i^3 + 3\psi_i^4$. Let $\beta = \psi_{\lceil \log(\ell+1) \rceil}$. Repeated use of equations (10), (11) shows that if $\#(\psi)$ is odd, then $\#(\beta) = -1 \pmod{2^{\ell+1}}$ and if $\#(\psi)$ is even, then $\#(\beta) = 0 \pmod{2^{\ell+1}}$. Also, the size of β is only $\exp(O(\log \ell))$ times larger than size of α, so the reduction runs in time polynomial in the input length. ■

Proof of Theorem 17.14 using Lemmas 17.17 and 17.22.: Let f be the reduction in Lemma 17.17 obtained by setting the parameter $m = 2$. Since it is a randomized reduction, we may think of it as a deterministic function taking two inputs, the quantified formula ψ and the random string r. Let R be the number of bits in the random string. Then let T be the reduction in Lemma 17.22 obtained by setting $\ell = R + 2$, and which therefore runs in $\mathrm{poly}(R, |f(\psi)|)$ time.

Consider the combined reduction $T \circ f$ (i.e., apply f followed by T) as a deterministic function applied to the input φ, r, and focus on the value of the following sum modulo $2^{\ell+1}$:

$$\sum_{r \in \{0,1\}^R} \#(T \circ f(\psi, r)) \,. \tag{12}$$

If ψ is true then at least 3/4 of the terms are -1 modulo $2^{\ell+1}$ and the remaining terms are 0 modulo $2^{\ell+1}$. Thus the sum modulo $2^{\ell+1}$ lies between -2^R and $-\lceil 3/4 \times 2^R \rceil$ in this case.

If ψ is false on the other hand then at least 3/4 of the terms are 0 modulo $2^{\ell+1}$ and the remaining terms are -1 modulo $2^{\ell+1}$. Thus the sum modulo $2^{\ell+1}$ is between $-\lceil \frac{1}{4} \times 2^R \rceil$ and 0 in this case.

Since $2^{\ell+1} > 2^{R+2}$, these two ranges are disjoint. So if we can somehow evaluate the expression in (12) using a query to a #SAT oracle, we can tell which of the two ranges its value lies in, and hence determine if ψ is true.

But it is straightforward to come up with such a query for the #SAT oracle using the Cook-Levin construction to express the deterministic computation represented by $T \circ f$.

Specifically, denote the vector of variables of the Boolean formula $T \circ f(\varphi, r)$ by y. We write a Boolean formula $\Gamma(r, y, z)$ that is 1 for an assignment (r, y, z) iff y is a satisfying assignment $T \circ F(\varphi, r)$. Such a formula can be written by applying the Cook-Levin construction on the circuit that first computes $T \circ f(\varphi, r)$ (where φ is "hardwired" into the circuit), and then substitutes the assignment y into this formula. The variables z correspond to values of inner wires of this circuit, and since the circuit is deterministic, its value is uniquely determined given y, r.

Hence $\#(\Gamma(r, y, z)) \bmod 2^{\ell+1}$ is exactly (12), and so the query for the #SAT oracle is to ask for $\#(\Gamma)$. ■

What have we learned?

- The class **#P** consists of functions that count the number of certificates for a given instance. If $\mathbf{P} \neq \mathbf{NP}$ then it is not solvable in polynomial time.
- Counting analogs of many natural **NP**-complete problems are **#P**-complete, but there are also **#P**-complete counting problems for which the corresponding decision problem is in **P**. For example the problem perm of finding the permanent of a matrix — equivalent to counting the number of perfect matchings in a graph —is **#P**-complete, whereas deciding whether a graph has a perfect matching is in **P**.
- Surprisingly, counting is more powerful than alternating quantifiers: we can solve every problem in the polynomial hierarchy using an oracle to a **#P**-complete problem.
- The classes **PP** and $\oplus$**P** contain the decision problems that correspond to the most significant and least significant bits (respectively) of a **#P** function. The class **PP** is as powerful as **#P** itself, in the sense that if $\mathbf{PP} = \mathbf{P}$ then $\mathbf{\#P} = \mathbf{FP}$. We do not know if this holds for $\oplus$**P** but do know that every language in **PH** randomly reduces to $\oplus$**P**.

17.5 Open Problems

- What is the exact power of $\oplus$SAT and $\#SAT$?
- What is the average case complexity of $n \times n$ permanent modulo small prime, say 3 or 5 ? Note that for a prime $p > n$, random self reducibility of permanent implies that if permanent is hard to compute on the worst case for randomized algorithms, then it is hard to compute on $1 - O(n/p)$ fraction of inputs, i.e. hard to compute on average (see Theorem 8.33).

Chapter notes and history

The definition of **#P** as well as several interesting examples of **#P** problems appeared in Valiant's seminal paper [Val79c]. The **#P**-completeness of the permanent is from his other paper [Val79b]. The **#P**-completeness of computing the partition function of the Ising model (Example 17.3) is due to Jerrum and Sinclair [JS93], where an FPRAS for the problem is also given. The **#P**-completeness of bayes net max likelihood estimation (Example 17.2) first appears in Roth [Rot93]. Dagum and Luby [DL93] had showed that even approximating the probabilities is **NP**-hard. Welsh's book [Wel93] shows the rich mathematical structure of the class **#P** and the mathematical problems (involving knot theory, graph colorings, tilings etc.) it captures.

For an introduction to FPRAS's for computing approximations to many counting problems, see the relevant chapter in Vazirani [Vaz01] (an excellent resource on approximation algorithms in

general) and the survey article by Jerrum and Sinclair [Hoc97]. The FPRAS for the permanent problem is due to Jerrum, Sinclair, and Vigoda [JSV01].

Toda's Theorem is proved in [Tod91]. This result had a very beneficial effect on complexity theory, because it showed the power of using arithmetic arguments in reasoning about complexity classes (a theme developed further in Chapters 8 and 11).

In addition to classes covered in this chapter such as **#P**, $\oplus$**P**, etc., many other complexity classes also involve some notion of counting. See the survey by Fortnow [For97b].

Exercises

17.1 Show that the problem of Example 17.2 is indeed equivalent to #SAT and hence **#P**-complete.

17.2 Show that computing the permanent for matrices with integer entries is in $\mathbf{FP}^{\#\mathsf{SAT}}$.

17.3 Complete the analysis of the XOR gadget in the proof of Theorem 17.11. Let G be any weighted graph containing a pair of edges $\overrightarrow{u\,u'}$ and $\overrightarrow{v\,v'}$, and let G' be the graph obtained by replacing these edges with the XOR gadget. Prove that every cycle cover of G of weight w that uses exactly one of the edges $\overrightarrow{u\,u'}$ is mapped to a set of cycle covers in G' whose total weight is $4w$, and all the other cycle covers of G' have total weight 0.

17.4 Show that if there is a polynomial-time algorithm that approximates #CYCLE within a factor $1/2$, then $\mathbf{P} = \mathbf{NP}$.

17.5 Show that if $\mathbf{NP} = \mathbf{P}$ then for every $f \in$ **#P** there is a randomized polynomial-time algorithm that approximates f within a factor of $1/2$. Can you show the same for a factor of $1 - \epsilon$ for arbitrarily small constant $\epsilon > 0$? Can you make these algorithms *deterministic*?

Note that we do not know whether $\mathbf{P} = \mathbf{NP}$ implies that exact computation of functions in **#P** can be done in polynomial time. H463

17.6 Show that every for every language in $\mathbf{AC}^0$ there is a depth 3 circuit of $n^{\mathrm{poly}(\log n)}$ size that decides it on $1 - 1/\mathrm{poly}(n)$ fraction of inputs and looks as follows: it has a single $\oplus$ gate at the top and the other gates are $\vee, \wedge$ of fan-in at most $\mathrm{poly}(\log n)$. H463

17.7 Improve Theorem 10.23 to show that $\mathbf{BQP} \subseteq \mathbf{P}^{\#\mathbf{P}}$. H463

Chapter 18

Average Case Complexity: Levin's Theory

So far we only studied the complexity of algorithms that solve computational task on *every* possible input; that is, *worst-case* complexity. With few exceptions (such as Chapter 9) most complexity classes we defined also concerned worst-case complexity; **NP**-completeness being a canonical example.

One frequent objection to this whole framework is that practitioners are only interested in instances of the problem that arise "in practice," and the worst-case behavior of algorithms may never be encountered. Of course, it is not always easy to quantify what these real-life instances are. Algorithm designers have tried to formalize this in various ways and to design efficient algorithm that work for "many" or "most" of these instances —this body of work is known variously as *average-case analysis* or *analysis of algorithms*. It has been discovered that several **NP**-hard problems are actually quite easy on the "average" graph, depending upon how one formulates "average." One way to formalize an "average" graph is that it is generated *randomly*. The simplest model of generating an n-vertex random graph is to toss an unbiased coin for each of the $\binom{n}{2}$ potential edges to decide whether or not to include it in the graph. This method ends up generating each n-vertex graph with probability $2^{-\binom{n}{2}}$. (If each edge is picked with probability p instead of $1/2$, then the resulting distribution is called $G(n,p)$, also well-studied.) On such random graphs, many **NP**-complete problems are easy. 3-COLOR can be solved in linear time with high probability. CLIQUE and INDSET can be solved in $n^{2\log n}$ time which is only a little more than polynomial and much less than $2^{\epsilon n}$, the running time of the best algorithms on worst-case instances. At the same time, our study of one-way functions in Chapter 9 also suggests that not all **NP** problems are easy on random instances.

The question arises whether we can come up with a theory analogous to **NP**-completeness for average-case complexity, and to identify problems that are 'hardest" or "complete" with respect to some appropriate notion of reducibility. This chapter surveys such a theory due to L. Levin (the same person involved in the Cook-Levin Theorem). For simplicity we restrict our study to decision problems.

The first goal in this theory is to make precise what we mean by "average" instances of a problem. This is done by assuming that inputs are drawn from a specific *distribution*. But then the question arises: what is the class of distributions that arise "in practice"? Levin makes a daring suggestion: we allow any distribution from which we can draw samples in polynomial time (**P**-samplable distribution) . Levin's reasoning was that the "real-life" instances must be produced by the actions of the world around us. If we believe in the strong form of the Church-Turing thesis (Section 1.6.1) then the world can be simulated on a Turing machine, and it is fair to assume that the "computation" that produced our instance was not very complicated, i.e., efficient. Hence we can assume the time to produce the instance was polynomial in the instance size. See Section 18.2 for details.

Thus an "average case problem" consists of a decision problem together with a distribution on inputs that is poly-time samplable. Then the question arises how one should define

an "efficient algorithm" for such an average-case problem —in other words, the analog of the class **P**. This turns out to be slightly subtle, and we give the precise definition of the class distP in Section 18.1. In Section 18.3 we try to define an analog of **NP**-completeness for average-case complexity. This has some subtleties, especially the notion of "reduction" needed. We define the class distNP— the average-case analog of **NP**— a corresponding notion of distNP-completeness and show that there exist a few problems that are distNP complete. However, unlike the case of **NP** completeness, we do not have a rich variety of natural problems that have been proven distNP complete.

Finding out the true average case complexity of **NP** problems is one of complexity theory's most important goals. In Section 18.4 we examine our current knowledge in this area, and how it connects to the broader study of complexity.

18.1 Distributional Problems and distP

The average case complexity of a problem is only well-defined with respect to a particular *distribution* on inputs. We now make this more precise:

Definition 18.1 *(Distributional problem)*
A *distributional problem* is a pair $\langle L, \mathcal{D} \rangle$ where $L \subseteq \{0,1\}^*$ is a language, and $\mathcal{D} = \{\mathcal{D}_n\}$ is a sequence of distributions, with $\mathcal{D}_n$ being a distribution over $\{0,1\}^n$.

Example 18.2
Here are some examples for distributional problems.

Planted clique Let $G_{n,p}$ be the distribution over n-vertex graph where each edge is chosen to appear in the graph independently with probability p. This distribution is clearly **P**-computable. The most common case is $p = 2$, in which case every graph in $G_{n,p}$ has equal probability and we call a graph drawn from this distribution a "random graph".

Let $k : \mathbb{N} \to \mathbb{N}$ be some function such that $k(n) \leq n$. A naive way to give an average-case analog of the CLIQUE problem would be to decide whether a random graph has a $k(n)$-clique. However, it turns out this problem is not so difficult, since with very high probability the clique number of a random graph is equal to an easily computable value (roughly equal to $2 \log n$) [BE76, Mat76].[1]

Thus, the "right" average-case analog of the $k(n)$-clique problem uses the following distribution $\mathcal{D}_n$. With probability 1/2 output a random n-vertex graph and with probability 1/2 choose a random $k(n)$-sized subset S of the vertices and output a random graph conditioned on S being a clique in the graph. The problem is to decide whether the given graph has a clique of size at least $k(n)$. Note that for $k(n) \gg 2 \log n$, the probability that a random graph has such a clique is very small. Using spectral methods, it is known how to solve this problem efficiently for $k(n) \sim \sqrt{n}$ [Kuc95, AKS98]. But for, say $k(n) = n^{0.49}$ the problem is wide open.

Random 3SAT A random 3CNF formula on n vertices and m clauses can be obtained by choosing each clause as the OR of three random literals. Clearly, the larger the number m of clauses, the less likely the formula to

[1] For infinitely many n's, with probability $1 - o(1)$ the clique number of a random n-vertex graph will be equal to $g(n)$ where $g(n) = \lfloor 2(\log n - \log\log n + \log(e) + 1) \rfloor$. For every n with probability $1 - o(1)$ this number will be in the set $\{g(n) - 1, g(n), g(n) + 1\}$. See also Exercise 18.2.

be satisfiable. If can be easily shown that there exists constant $c_1 < c_2$ such that if the number of clauses m is less than $c_1 n$ then the formula will be satisfiable with very high probability and if $m > c_2 n$ then it will be unsatisfiable with very high probability (e.g., $c_1 = 1, c_2 = 8$ will do). In fact, it was shown by Friedgut [Fri99] that there is a function $f(n)$ (where $c_1 n \leq f(n) \leq c_2(n)$ for every n) such that for every $\epsilon > 0$, if the number of clauses m is smaller than $(1-\epsilon)f(n)$ then the formula will be satisfiable with high probability, and if his number is larger than $(1+\epsilon)f(n)$ then it will be unsatisfiable with high probability. It is believed that $f(n) = c^* n$ for some constant $c^* \sim 4.26$. For m that is very close to this value, the problem of determining satisfiability of a random n-variable m-clause 3CNF formula seems hard. In fact, because in this chapter we require average-case algorithms to *always* output the correct answer (see Definition 18.4 below), the problem remains hard for much larger value m. At the moment no expected polynomial-time algorithm is known even in the case of $m = n^{1.1}$, despite the fact that such a formula will be unsatisfiable with overwhelming probability (some partial progress was made in [GK01, FO04, FKO06]).

Decoding a random linear code Let A be an $m \times n$ matrix over GF(2), where $m > n$ (say, $m = 10n$). The *decoding* problem of A is to find, given a vector $\mathbf{z} \in \mathrm{GF}(2)^m$, the closest vector $\mathbf{y}$ to $\mathbf{z}$ such that $\mathbf{y}$ is in the image of A (i.e., $\mathbf{y} = A\mathbf{x}$ for some $\mathbf{x} \in \mathrm{GF}(2)^n$). (This is motivated by considering A as the generating matrix for an *error correcting code*; see also Section 19.2.) There are efficient algorithms to do this for matrices A of various special forms, but for a random matrix A no efficient algorithm is known. This problem is also known as the problem of *learning parity with noise*.

The decoding problem is of course a search problem. Fix $\epsilon > 0$ to some constant. The following analogous decision problem $(L, \mathcal{D}_n)$ is not known to be in distP. We let **(a)** L contain all pairs $\langle A, \mathbf{y} \rangle$ such that $\mathbf{y}$ is within Hamming distance at most ϵm to a vector in A's image and **(b)** the distribution $\mathcal{D}_n$ outputs with probability 1/2 a random $m \times n$ matrix A and a random vector $\mathbf{y} \in_R \mathrm{GF}(2)^m$, and with probability 1/2, a random $m \times n$ matrix A, and $\mathbf{y} = A\mathbf{x} + \mathbf{e}$ where $\mathbf{x}$ is chosen at random in $\mathrm{GF}(2)^n$ and $\mathbf{e}$ is a random vector in $\mathrm{GF}(2)^m$ having exactly $\lfloor \epsilon m \rfloor$ entries equal to 1.

Both these problems are related to the *subset sum* problem and problems on discrete lattices in $\mathbb{R}^n$ that have proven useful in cryptography; see the notes to Chapter 9.

Our next step is to define the class distP— the average-case analog of **P**— that aims to capture the set of distributional problems $\langle L, \mathcal{D} \rangle$ that are efficiently solvable.[2] For every algorithm A and input x, let $\mathsf{time}_A(x)$ denote the number of steps A takes on input x. A natural candidate definition is to say that $\langle L, \mathcal{D} \rangle$ is solvable in polynomial-time on the average if there is an algorithm A such that $A(x) = L(x)$ for every x and a polynomial p such that for every n, $\mathsf{E}_{x \in_R \mathcal{D}_n}[\mathsf{time}_A(x)] \leq p(n)$.

Unfortunately, it turns out that this definition is not robust in the following sense: if we change the model of computation to a different model with quadratic slow down (for example, change from multiple tape Turing machines to one-tape Turing machines) then a polynomial-time algorithm can suddenly turn into an exponential-time algorithm, as demonstrated by the following simple claim:

Claim 18.3 *There is an algorithm A such that for every n we have $\mathsf{E}_{x \in_R \{0,1\}^n}[\mathsf{time}_A(x)] \leq n+1$ but $\mathsf{E}_{x \in_R \{0,1\}^n}[\mathsf{time}_A^2(x)] \geq 2^n$.* ♢

[2] In this chapter we restrict ourselves to deterministic algorithms, although the theory extends naturally to probabilistic algorithms, yielding average-case analogs of classes such as **BPP**, **RP**, **coRP** and **ZPP**.

PROOF: Consider an algorithm A that halts in n steps on every input except for the all-zeros input, on which it runs for 2^n steps. The expected running time of A is $(1-2^{-n})n+2^{-n}2^n \leq n+1$. On the other hand, if we square the running time then the expectation becomes $(1-2^{-n})n^2+2^{-n}2^{2n} \geq 2^n$. ■

This motivates the following definition:

Definition 18.4 *(Polynomial on average and* distP*)*
A distributional problem $\langle L, \mathcal{D}\rangle$ is in distP if there is an algorithm A for L and constants C and $\epsilon > 0$ such that for every n

$$\mathop{\mathsf{E}}_{x \in_R \mathcal{D}_n}\left[\frac{\mathsf{time}_A(x)^\epsilon}{n}\right] \leq C\,. \tag{1}$$

Notice that $\mathbf{P} \subseteq$ distP: if a language can be decided deterministically by an algorithm A in time $O(|x|^c)$, then $\mathsf{time}_A(x)^{1/c} = O(|x|)$ and the expectation in (1) is bounded by a constant regardless of the distribution. Second, the definition is robust to changes in computational models: if the running times get squared, we just multiply c by 2 and the expectation in (1) is again bounded.

Another feature of this definition is that there is a high probability that the algorithm runs in polynomial time. Indeed, by Markov's inequality, (1) implies that for every $K > 1$, $\Pr[\frac{\mathsf{time}_A(x)^\epsilon}{n} \geq KC] = \Pr[\mathsf{time}_A(x) \geq (KCn)^{1/\epsilon}]$ is at most $1/K$.

Finally, we note that the definition is robust to minor changes. For instance, for every $d > 0$, the following condition is equivalent to (1): there exist ϵ, C such that

$$\mathop{\mathsf{E}}_{x \in_R \mathcal{D}_n}\left[\frac{\mathsf{time}_A(x)^\epsilon}{n^d}\right] \leq C\,, \tag{2}$$

see Exercise 18.6.

18.2 Formalization of "real-life distributions"

Real-life problem instances arise out of the world around us (images that have to be understood, a building that has to be navigated by a robot, etc.), and the world does not spend a lot of time tailoring instances to be hard for our algorithm —arguably, the world is indifferent to our algorithm. One may formalize this indifference in terms of computational ease, by hypothesizing that the instances are produced by an efficient algorithm (see also the discussion of Section 1.6.3). We can formalize this in two ways.

Polynomial time computable (or P-computable) distributions. Such distributions have an associated deterministic polynomial time machine that, given input $x \in \{0,1\}^n$, can compute the *cumulative probability* $\mu_{\mathcal{D}_n}(x)$, where

$$\mu_{\mathcal{D}_n}(x) = \sum_{y \in \{0,1\}^n : y \leq x} \Pr_{\mathcal{D}_n}[y]$$

Here $\Pr_{\mathcal{D}_n}[y]$ denotes the probability assigned to string y and $y \leq x$ means y either precedes x in lexicographic order or is equal to x.

Denoting the lexicographic predecessor of x by $x-1$, we have

$$\Pr_{\mathcal{D}_n}[x] = \mu_{\mathcal{D}_n}(x) - \mu_{\mathcal{D}_n}(x-1),$$

which shows that if $\mu_{\mathcal{D}_n}$ is computable in polynomial time, then so is $\Pr_{\mathcal{D}_n}[x]$. The converse is known to be false if $\mathbf{P} \neq \mathbf{NP}$ (Exercise 18.3). The uniform distribution is $\mathbf{P}$-computable as are many other distributions that are defined using explicit formulae.

Polynomial time samplable (or P-samplable) distributions. These distributions have an associated probabilistic polynomial time machine that can produce samples from the distribution. Specifically, we say that $\mathcal{D} = \{\mathcal{D}_n\}$ is **P**-samplable if there is a polynomial p and a probabilistic $p(n)$-time algorithm S such that for every n, the random variables $A(1^n)$ and $\mathcal{D}_n$ are identically distributed.

If a distribution is **P**-computable then it is **P**-samplable, but the converse is not true if $\mathbf{P} \neq \mathbf{P}^{\#\mathbf{P}}$ (see exercises 18.4–18.5). In this chapter we mostly restrict attention to **P**-computable distributions, but the theory can be extended to **P**-samplable distributions; see Section 18.3.2.

18.3 distNP and its complete problems

The following complexity class is at the heart of our study of average case complexity; it is the average-case analog of **NP**.

Definition 18.5 *(The class* distNP*)*
A distributional problem $\langle L, \mathcal{D}\rangle$ is in distNP if $L \in \mathbf{NP}$ and $\mathcal{D}$ is **P**-computable.

We now define reduction between distributional problems:

Definition 18.6 *(Average-case reduction)*
We say that a distributional problem $\langle L, \mathcal{D}\rangle$ *average-case reduces* to a distributional problem $\langle L', \mathcal{D}'\rangle$, denoted by $\langle L, \mathcal{D}\rangle \leq_p \langle L', \mathcal{D}'\rangle$, if there is a polynomial-time computable f and polynomials $p, q : \mathbb{N} \to \mathbb{N}$ satisfying:

1. *(Correctness)* For every $x \in \{0,1\}^*$,$x \in L \Leftrightarrow f(x) \in L'$
2. *(Length regularity)* For every $x \in \{0,1\}^*$, $|f(x)| = p(|x|)$.
3. *(Domination)* For every $n \in \mathbb{N}$ and $y \in \{0,1\}^{p(n)}$, $\Pr[y = f(\mathcal{D}_n)] \leq q(n) \Pr[y = \mathcal{D}'_{p(n)}]$.

The first condition is the standard reduction condition, ensuring that a decision algorithm for L' easily converts into a decision algorithm for L. The second condition is technical, and is used to simplify the definition and also to show that the reducibility relation is transitive (see Exercise 18.7). We now motivate the third condition, which says that $\mathcal{D}'$ "dominates" (up to a polynomial factor) the distribution $f(\mathcal{D})$ obtained by applying f on $\mathcal{D}$. Realize that the goal of the definition is to ensure that "if $\langle L, \mathcal{D}\rangle$ is hard, then so is $\langle L', \mathcal{D}'\rangle$", or equivalently, the contrapositive "if $\langle L', \mathcal{D}'\rangle$ is easy, then so is $\langle L, \mathcal{D}\rangle$." Thus if an algorithm A' is efficient for problem $(L', \mathcal{D}')$, then it would be nice if the "obvious" algorithm for the problem $(L, \mathcal{D})$ worked: namely, on input x obtained from the distribution $\mathcal{D}$, compute $y = f(x)$ and run algorithm A' on y. *A priori*, one cannot rule out the possibility that A' is very slow on some input that is unlikely to be sampled according to distribution $\mathcal{D}'$ but which has a high probability of showing up as $f(x)$ when we sample x according to $\mathcal{D}$. The domination condition rules this possibility out:

Theorem 18.7 *If $\langle L, \mathcal{D}\rangle \leq_p \langle L', \mathcal{D}'\rangle$ and $\langle L', \mathcal{D}'\rangle \in$ distP then $\langle L, \mathcal{D}\rangle \in$ distP.* ♢

PROOF: Suppose that A' is a polynomial-time algorithm for $\langle L', \mathcal{D}'\rangle$. That is, there are constants $C, \epsilon > 0$ such that for every m

$$\mathsf{E}[\frac{\mathsf{time}_{A'}(\mathcal{D}'_m)^\epsilon}{m}] \leq C\,. \tag{3}$$

Let f be the reduction from $\langle L, \mathcal{D}\rangle$ to $\langle L', \mathcal{D}'\rangle$ and let A be the "obvious" algorithm for deciding L: given input x it computes $f(x)$ and then outputs $A'(f(x))$. Since A decides L, all that is left to show is that A runs in time polynomial on the average with respect to the distribution $\mathcal{D}$.

For simplicity, assume that for every x, $|f(x)| = |x|^d$ and that computing f on length n inputs is faster than the running time of A' on length n^d inputs and hence $\mathsf{time}_A(x) \leq 2\mathsf{time}_{A'}(f(x))$. (The proof easily extends when we drop these assumptions.) We prove the Lemma by showing that

$$\mathsf{E}[\tfrac{(\frac{1}{2}\mathsf{time}_A(\mathcal{D})^\epsilon)}{q(n)n^d}] \leq C\,,$$

where q denotes the polynomial occuring in the domination condition. By Exercise 18.6, this suffices to show that $\langle L, \mathcal{D}\rangle \in \mathsf{distP}$.

Indeed, by the definition of A and our assumptions,

$$\begin{aligned}
\mathsf{E}[\frac{(\frac{1}{2}\mathsf{time}_A(\mathcal{D}_n))^\epsilon}{q(n)n^d}] &\leq \sum_{y\in\{0,1\}^{n^d}} \Pr[y = f(\mathcal{D}_n)]\tfrac{\mathsf{time}_{A'}(y)^\epsilon}{q(n)n^d} \\
&\leq \sum_{y\in\{0,1\}^{n^d}} \Pr[y = \mathcal{D}'_{n^d}]\tfrac{\mathsf{time}_{A'}(y)^\epsilon}{n^d} \text{ (by domination)} \\
&= \mathsf{E}[\frac{\mathsf{time}_{A'}(\mathcal{D}'_{n^d})^\epsilon}{n^d}] \leq C \text{ by (3). } \blacksquare
\end{aligned}$$

18.3.1 A complete problem for distNP

Of course, Theorem 18.7 is useful only if we can find reductions between interesting problems. Now we show that this is the case: we exhibit a problem (albeit an artificial one) that is complete for distNP. We say that $\langle L', \mathcal{D}'\rangle$ is distNP-*complete* if $\langle L', \mathcal{D}'\rangle$ is in distNP and $\langle L, \mathcal{D}\rangle \leq_p \langle L', \mathcal{D}'\rangle$ for every $\langle L, \mathcal{D}\rangle \in \mathsf{distNP}$. We have the following theorem:

Theorem 18.8 *(Existence of a* distNP*-complete problem* [Lev86]*)*
Let U contain all tuples $\langle M, x, 1^t\rangle$ where there exists a string $y \in \{0,1\}^\ell$ such that the non-deterministic TM M outputs 1 on input x within t steps.

For every n, we let $\mathcal{U}_n$ be the following distribution on length n tuples $\langle M, x, 1^t\rangle$: the string representing M is chosen at random from all strings of length at most $\log n$, t is chosen at random in the set $\{0, \ldots, n - |M|\}$ and x is chosen at random from $\{0,1\}^{n-t-|M|}$. This distribution is polynomial-time computable (Exercise 18.8).[3]

Then, $\langle U, \mathcal{U}\rangle$ is distNP*-complete.*

The problem U is of course $\mathbf{NP}$-complete via a trivial reduction: given a language L decidable by a $p(n)$-time NDTM M, we can reduce L to U by mapping the string x into the tuple $\langle M, x, 1^t\rangle$. However, this reduction does not necessarily work as an *average case* reduction, since it may not satisfy the domination condition. The problem is that we need to reduce *every* distributional problem $\langle L, \mathcal{D}\rangle$ to $\langle U, \mathcal{U}\rangle$ and will run into trouble if $\mathcal{D}$ has any "peaks", namely inputs x of length n that are obtained with significantly higher than 2^{-n} probability in $\mathcal{D}$, whereas the output of the reduction is a predetermined string $\langle M, x, 1^t\rangle$ whose probability in $\mathcal{U}_n$ is no more than 2^{-n}.

The obstacle is surmounted using the following lemma, which shows that for polynomial-time computable distributions, we can apply a simple transformation on the inputs such that the resulting distribution has no "peaks."

[3] Strictly speaking, the inputs might be represented by a few more than n bits to account for separators etc.. but these details can be easily taken care of and are ignored below.

Lemma 18.9 *(Peak Elimination) Let $\mathcal{D} = \{\mathcal{D}_n\}$ be a* **P***-computable distribution. Then, there is a polynomial-time computable function $g : \{0,1\}^* \to \{0,1\}^*$ such that:*

1. *g is one-to-one: $g(x) = g(z)$ iff $x = z$.*
2. *For every $x \in \{0,1\}^*$, $|g(x)| \leq |x| + 1$.*
3. *For every string $y \in \{0,1\}^m$, $\Pr[y = g(\mathcal{D}_m)] \leq 2^{-m+1}$.* ♢

PROOF: For any string $x \in \{0,1\}^n$, define $h(x)$ to be the largest common prefix of the binary representations of $\mu_{\mathcal{D}_n}(x)$ and $\mu_{\mathcal{D}_n}(x-1)$. Note that if $\Pr_{\mathcal{D}_n}[x] \geq 2^{-k}$ then since $\mu_{\mathcal{D}_n}(x) - \mu_{\mathcal{D}_n}(x-1) = \Pr_{\mathcal{D}_n}(x)$, the values $\mu_{\mathcal{D}_n}(x)$ and $\mu_{\mathcal{D}_n}(x-1)$ must differ in the somewhere in the first k bits, implying that $|h(x)| \leq k$. Note also that because $\mathcal{D}$ is **P**-samplable, the function h is computable in polynomial time. Furthermore, h is one-to-one because only two binary strings s_1 and s_2 can have the longest common prefix z; a third string s_3 sharing z as a prefix must have a longer prefix with either s_1 or s_2.

Now define for every $x \in \{0,1\}^n$

$$g(x) = \begin{cases} 0x & \text{if } \Pr_{\mathcal{D}_n}[x] \leq 2^{-n} \\ 1h(x) & \text{otherwise} \end{cases}$$

Clearly, g is one to one and satisfies $|g(x)| \leq |x| + 1$. We now show that $\Pr[y = g(\mathcal{D}_n)] \leq 2^{-n}$ for every $y \in \{0,1\}^{n+1}$. If y is not $g(x)$ for any x, this is trivially true since $\Pr_{g \circ \mathcal{D}}(y) = 0$. If $y = 0x$, where $\Pr_{\mathcal{D}}(x) \leq 2^{-|x|}$, then $\Pr_{g\circ\mathcal{D}}(y) \leq 2^{-|y|+1}$ and we also have nothing to prove. Finally, if $y = g(x) = 1h(x)$ where $\Pr_{\mathcal{D}}(x) > 2^{-|x|}$, then as already noted, $|h(x)| \leq \log 1/\Pr_{\mathcal{D}}(x)$ and so $\Pr_{g\circ\mathcal{D}}(y) = \Pr_{\mathcal{D}}(x) \leq 2^{-|y|+1}$. ■

Now we are ready to prove Theorem 18.8.

PROOF OF THEOREM 18.8: Let $\langle L, \mathcal{D}\rangle$ be in distNP and let M be the polynoimal-time non-deterministic TM M accepting L. Define the following NDTM M': on input y, guess x such that $y = g(x)$ (where g is the function obtained by Lemma 18.9) and execute $M(x)$. Let p be the polynomial running time of M'.

To reduce $\langle L, \mathcal{D}\rangle$ to $\langle U, \mathcal{U}\rangle$, we simply map every string x into the tuple $\langle M', g(x), 1^k\rangle$ where $k = p(n) + \log n + n - |M'| - |g(x)|$ (we may assume that for sufficiently large n, the description length $|M'|$ of M' is at most $\log n$). This reduction obviously satisfies the length regularity requirement. Also, because the function g is one-to-one, it satisfies the correctness condition as well. Hence, all that is left is to show the domination condition.

But indeed by Lemma 18.9, the probability that a length m tuple $\langle M', y, 1^t\rangle$ is obtained by the reduction is at most $2^{-|y|+1}$. Yet this tuple is obtained with probability at least $2^{-\log m} 2^{-|y|} \frac{1}{m}$ by $\mathcal{U}_m$, and hence the domination condition is satisfied.■

The proof relies crucially on the fact that every TM can be described by a string of constant size (i.e., independent of the input length). In fact, the proof suffers a loss exponential in this constant in the probability of hard instances. Since this constant may be quite large for typical **NP** languages, this would be a consideration in practice.

18.3.2 P-samplable distributions

Arguably some distributions arising in nature could be samplable even if they are not computable. Define sampNP to be the set of distributional problems $\langle L, \mathcal{D}\rangle$ such that $L \in \mathbf{NP}$ and $\mathcal{D}$ is **P**-samplable, and say that $\langle L', \mathcal{D}'\rangle$ is sampNP-complete if $\langle L', \mathcal{D}'\rangle \in$ sampNP and $\langle L, \mathcal{D}\rangle \leq_p \langle L', \mathcal{D}'\rangle$ for every $\langle L, \mathcal{D}\rangle \in$ sampNP. Fortunately, we can transform results such as Theorem 18.8 to sampNP-completeness via the following result:

Theorem 18.10 *([IL90]) If $\langle L, \mathcal{D}\rangle$ is* distNP*-complete then it is also* sampNP*-complete.* ♢

The (omitted) proof uses techniques from derandomization, and specifically the leftover hash lemma (Lemma 21.26).

18.4 Philosophical and practical implications

The reader has seen many complexity classes and conjectures by now, so it may be useful to consider all possible scenarios for the world of complexity. Impagliazzo [Imp95a] has partitioned these scenarios nicely under highly memorable names. At the moment, we do not know which of the scenarios is true true (i.e., which of the following worlds is the one we live in):

Algorithmica: Algorithmica is the world where $\mathbf{P} = \mathbf{NP}$ or its moral equivalent (e.g., $\mathbf{NP} \subseteq \mathbf{BPP}$). To be more concrete, let's define Algorithmica as the world where there exists a simple and magical linear time algorithm for the SAT problem. As discussed in Section 2.7.3, this world is a computational utopia. We would be able to automate various tasks that currently require significant creativity: engineering, programming, mathematics, and perhaps even writing, composing, and painting. On the other hand, this algorithm could also be used to break cryptographic schemes, and hence almost all of the cryptographic applications currently used will disappear.

Heuristica: Heuristica is the world where $\mathbf{P} \neq \mathbf{NP}$ and yet $\mathsf{distNP}, \mathsf{sampNP} \subseteq \mathsf{distP}$. That is, we have an efficient and magical algorithm that "almost" solves every **NP** problem. There may exist inputs on which it fails or runs for a long time, but it's hard to find such inputs, and we almost never encounter them in real life. In some respects, Heuristica is very similar to Algorithmica— after all, it seems hard to distinguish between the two if we can't find an input on which the algorithm magical algorithm fails! Indeed, many applications of $\mathbf{NP} = \mathbf{P}$ still hold in this world, including solving **NP**-optimization problems, coming up with short mathematical proofs, and breaking cryptographic schemes. However, some applications might not hold. In particular, even though we know that if $\mathbf{P} = \mathbf{NP}$ then the polynomial hierarchy **PH** collapses to **P** (see Theorem 5.4), we don't have an analogous result for average case complexity.

Pessiland: Pessiland is the world where distNP and sampNP are *not* in distP, but still there do not exist any one-way functions (see Chapter 9). Impagliazzo called this world Pessiland because in some sense it is the worst possible world. On the one hand, we don't have any of the exciting algorithmic wonders of Algorithmica and Heuristica, but on the other hand, we don't have most of cryptography either. (Recall from Chapter 9 that one-way functions are known to be essential to most cryptographic applications.)

Minicrypt: Minicrypt is the world where one-way functions exist (and hence $\mathsf{distNP} \not\subseteq \mathsf{distP}$, see Exercise 18.10), but all the *highly structured* problems in **NP** such as integers factoring etc are solvable in polynomial time. More formally, this is the world where although one-way functions exist, there are no public key encryption schemes or key exchange protocols. While many cryptographic applications (private key encryption, pseudorandom generators and functions, digital signatures) are achievable using only one-way functions, there are several important and exciting ones (public key encryption, secure multiparty computation) that are not known to be achievable using such functions.

Cryptomania: Cryptomania is the world where the problem of factoring large integers (or some other highly structured problem such as discrete log, shortest lattice vector etc..) is exponentially hard on the average case. Most researchers believe this is the world we live in. While we don't have general purpose algorithms in this world, and have to resort to heuristics, approximations, creativity and hard work to solve many important computational tasks, we do seem to have a host of exciting cryptographic applications. These include the ability of two parties to communicate secretly without prior sharing of keys (public key encryption, currently widely used to enable online commerce) and even more sophisticated cryptographic applications such secure online auction and voting schemes and more.

Strictly speaking, Impagliazzo has left out some intermidiate scenarios, which we lump into "Weirdland." Say, where the complexity of SAT is a not linear or quadratic but a very

large polynomial like n^{100} or a very slow growing superpolynomial function like $n^{\log n}$. Or where the complexity of problems like SAT shifts wildly for different input sizes, so that it is feasible for some input sizes and infeasible for others. But, qualitatively speaking, the above five scenarios are the main possibilities for the average-case hardness of **NP**. Narrowing this list down is in some sense *the* central task of computational complexity.

WHAT HAVE WE LEARNED?

- Average case complexity is defined with respect to a particular distribution on the inputs. The same problem might be easy with one distribution and hard with another.
- The class distP is the average case analog of the class **P**, and models distributional problems with efficient algorithms.
- The average-case analog of **NP** is either distNP or sampNP, depending on whether we pick **P**-computable or **P**-samplable distributions as our model of "real-life" distributions. The distributional problem $\langle U, \mathcal{U} \rangle$ of Theorem 18.8 is complete for both classes.
- Like the **P** vs. **NP** question, the average-case hardness of **NP** is still open. At the moment we do not even know any non-trivial relation between the two questions. For example, we do not know if $\mathbf{NP} \not\subseteq \mathbf{P}$ implies that $\mathsf{distNP} \not\subseteq \mathsf{distP}$.

Chapter notes and history

One of the most natural distributions over inputs to algorithms is the distribution of *random graphs*. Study of such graphs started with a 1959 paper of Erdos and Renyi [ER59]; a good survey of this vast area is the text by Bollobas [Bol01]. Analysis of average case behavior of algorithms is also known as *probabilistic analysis of algorithms*; see the survey by Reed [FR98]. Spielman and Teng [ST01] introduced *smoothed analysis of algorithms*— a notion that lies between worst-case analysis and probabilistic analysis, for which it would be fascinating to have an analog of the theory of **NP**-completeness.

Levin outlined his theory and Theorem 18.8 in [Lev86]. His formalization is more general than the one in this chapter. For instance, almost all the algorithms occurring in his version of the theory —e.g., the algorithm that computes a **P**-computable distribution, or the one that computes a reduction— are allowed to be randomized.

The extension of Levin's theorem to **P**-samplable distributions is from Impagliazzo and Levin [IL90]. Many basic facts about Levin's theory, such as the effect of changing the assumptions, or the interrelationship among assumptions such as **P**-samplability and **P**-computability, are discussed in Ben-David et al. [BDCGL89]; see the survey by Goldreich [Gol97]. Johnson's survey [Joh84] of average case complexity is old (it appeared around the time of Levin's original paper) but still highly readable. One of the goals in this area has been to prove the average-case completeness of "natural" **NP** problems. A recent paper of Livne [Liv06] gives the strongest such result (where the problems are "natural", though the distributions are not).

Exercises

18.1 Describe an algorithm that decides 3-colorability on the uniform distribution of graphs (each edge is chosen with probability 1/2) in expected polynomial-time. H463

18.2 Describe an algorithm that solves the CLIQUE problem on the distribution $\langle G, k \rangle$ where G is a uniformly chosen n-vertex graph and k is chosen at random from $[n]$ in $n^{2 \log n}$ expected time. H463

18.3 Show that if $\mathbf{P} \neq \mathbf{NP}$ then there is a family $\mathcal{D} = \{\mathcal{D}_n\}$ of distributions on n-bit strings such that for every $x \in \{0,1\}^n$, there is an algorithm to compute $\Pr[\mathcal{D}_n = x]$ but $\mathcal{D}$ is not **P**-computable.

18.4 Show that if a distribution is $\mathbf{P}$-computable, then it is $\mathbf{P}$-samplable.

18.5 Show that if $\mathbf{P}^{\#\mathbf{P}} \neq \mathbf{P}$ then there is a polynomial time samplable distribution that is not polynomial time computable. H463

18.6 Show that if an algorithm satisfies (2) then it satisfies (1), with possibly different constants ϵ, C. H463

18.7 Show that the notion of reducibility defined in this chapter is transitive. In other words, if $\langle L_1, \mathcal{D}_1 \rangle \leq_p \langle L_2, \mathcal{D}_2 \rangle$ and $\langle L_2, \mathcal{D}_2 \rangle \leq_p \langle L_3, \mathcal{D}_3 \rangle$ then $\langle L_1, \mathcal{D}_1 \rangle \leq_p \langle L_3, \mathcal{D}_3 \rangle$.

18.8 Show that the distribution $\mathcal{U}$ of Theorem 18.8 is $\mathbf{P}$-computable.

18.9 Show that the function g defined in Lemma 18.9 (Peak Elimination) is efficiently invertible in the following sense: if $y = g(x)$, then given y we can reconstruct x in $|x|^{O(1)}$ time.

18.10 Show that if one-way functions exist, then $\mathsf{distNP} \not\subseteq \mathsf{distP}$.

Chapter 19

Hardness Amplification and Error Correcting Codes

Complexity theory studies the *computational hardness* of functions. In this chapter we are interested in functions that are hard to compute on the "average" instance, continuing a topic that played an important role in Chapters 9 and 18, and will do so again in Chapter 20. The special focus in this chapter is on techniques for *amplifying* hardness, which is useful in a host of contexts. In *cryptography* (see Chapter 9), hard functions are necessary to achieve secure encryption schemes of non-trivial key size. Many conjectured hard functions like factoring are only hard on a few instances, not all. Thus these functions do not suffice for some cryptographic applications, but via hardness amplification we can turn them into functions that do suffice. Another powerful application will be shown in Chapter 20— derandomization of the class **BPP** under worst-case complexity theoretic assumptions. Figure 19.1 contains a schematic view of this chapter's sections and the way their results are related to that result. In addition to their applications in complexity theory, the ideas covered in this chapter have had other uses, including new constructions of error-correcting codes and new algorithms in machine learning.

For simplicity we study hardness amplification in context of Boolean functions though this notion can apply to functions that are not Boolean-valued. Section 19.1 introduces the first technique for hardness amplification, namely, *Yao's XOR Lemma*. It allows us to turn weakly hard functions into strongly hard functions. Roughly speaking, a Boolean function f is said to be *weakly hard* if every moderate-sized circuit fails to compute it on some nonnegligible fraction of inputs, say 0.01 fraction. The function is *strongly hard* if every moderate-sized circuit fails to compute it on almost *half* the inputs, say $1/2 - \epsilon$ fraction of inputs. (Note that every Boolean function can be computed correctly on at least half the inputs by a trivial circuit, namely one that always outputs 1 or always outputs 0.) The Section describes a way to transform every function using a simple "XOR" construction that does not greatly increase the complexity of computing it but has the property that if the function we started with was weakly hard then it becomes strongly hard. This construction is very useful in cryptographic applications, as mentioned in Chapter 9.

We then turn our attention to a different technique for hardness amplification that produces strongly hard functions starting with functions that are merely guaranteed to be hard in the *worst case*. This is highly non-trivial as there is often quite a difference between the worst-case and average-case complexity of computational problems. (For example, while finding the smallest factor of a given integer seems difficult in general, it's trivial to do for half the integers— namely, the even ones.) The main tool we use is *error correcting codes*. We review the basic definition and constructions in sections 19.2 and 19.3, while Section 19.4 covers *local decoding* which is the main notion needed to apply error-correcting codes in our setting. As a result we obtain a way to transform every function f that is hard in the worst case into a function $\hat{f}$ that is mildly hard in the average case.

Combining the transformation of Section 19.4 with Yao's XOR Lemma of Section 19.1, we are able to get functions that are extremely hard on the average case from functions that are

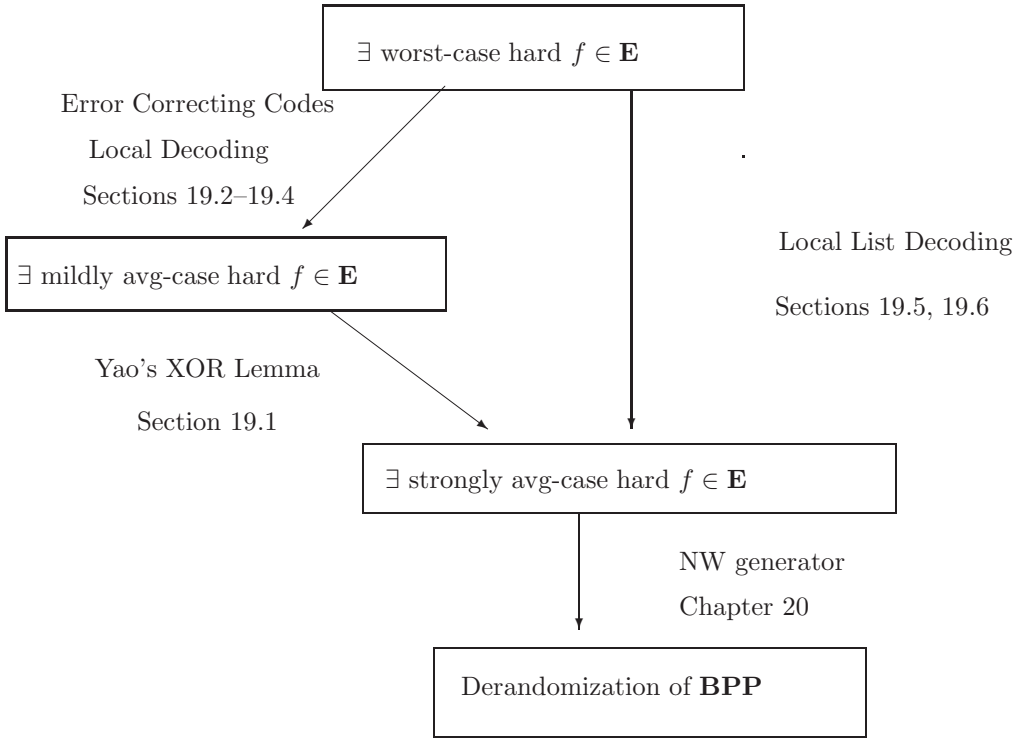


Figure 19.1 Organization of Chapter 19

only hard on the worst case. Alas, quantitatively speaking the above transformation is not optimal, in the sense that even if the original function was worst-case hard for exponential sized (i.e. size $2^{\Omega(n)}$) circuits, we are only able to guarantee that the transformed function will only be hard in the average case for sub-exponential sized (i.e., size $2^{n^{\Omega(1)}}$) circuits. In Sections 19.5 and 19.6 we show a stronger result, that transforms in one fell swoop a function f that is hard on the worst case to a function $\hat{f}$ that is *extremely hard* on the average case. This transformation uses error correcting codes in a more sophisticated way, via an independently interesting notion called *list decoding*. List decoding is covered in Section 19.5 while Section 19.6 describes *local list decoding* which is the extension of list decoding needed for our purposes.

Readers who are familiar with the theory of error-correcting codes can skim through Sections 19.2 and 19.3 in a first reading (pausing to remind themselves of the Reed-Solomon and Reed-Muller codes in Definitions 19.10 and 19.12 and their associated decoding algorithms) and go on to Section 19.4.

19.1 Mild to strong hardness: Yao's XOR Lemma.

Yao's XOR Lemma transforms a function that has "mild" average-case hardness to a function that has strong average-case hardness. The transformation is actually quite simple and natural, but its analysis is somewhat involved (yet, in our opinion, beautiful). To state the Lemma we need to define precisely the meaning of worst-case hardness and average-case hardness of a function:

Definition 19.1 *(Average-case and worst-case hardness)*
For $f : \{0,1\}^n \to \{0,1\}$ and $\rho \in [0,1]$ we define the *ρ-average case hardness of f*, denoted $\mathsf{H}^{\rho}_{\mathsf{avg}(f)}$, to be the largest S such that for every circuit C of size at most S, $\Pr_{x \in_R \{0,1\}^n}[C(x) = f(x)] < \rho$. For an infinite $f : \{0,1\}^* \to \{0,1\}$, we let $\mathsf{H}^{\rho}_{\mathsf{avg}(f)}(n)$ denote $\mathsf{H}^{\rho}_{\mathsf{avg}(f_n)}$ where f_n is the restriction of f to $\{0,1\}^n$.

We define the *worst-case hardness* of f, denoted $\mathsf{H}_{\mathsf{wrs}(f)}$, to equal $\mathsf{H}^{1}_{\mathsf{avg}(f)}$ and define the *average-case hardness of f*, denoted $\mathsf{H}_{\mathsf{avg}(f)}$, to equal $\max\{S : \mathsf{H}^{1/2+1/S}_{\mathsf{avg}}(f) \geq S\}$. That is, $\mathsf{H}_{\mathsf{avg}(f)}$ is the largest number S such that $\Pr_{x \in_R \{0,1\}^n}[C(x) = f(x)] < 1/2 + 1/S$ for every Boolean circuit C on n inputs with size at most S.

Note that for every function $f : \{0,1\}^n \to \{0,1\}$, $\mathsf{H}_{\mathsf{avg}(f)} \leq \mathsf{H}_{\mathsf{wrs}(f)} \leq O(2^n/n)$ (see Exercise 6.1). This definition of average-case hardness is tailored to the application of derandomization, and in particular only deals with the uniform distribution over the inputs. See Chapter 18 for a more general treatment of average-case complexity. We can now state Yao's lemma:

Theorem 19.2 *(Yao's XOR Lemma* [Yao82a]*)*
For every $f : \{0,1\}^n \to \{0,1\}$, $\delta > 0$ and $k \in \mathbb{N}$, if $\epsilon > 2(1-\delta)^k$ then

$$\mathsf{H}^{1/2+\epsilon}_{\mathsf{avg}}(f^{\oplus k}) \geq \tfrac{\epsilon^2}{400n}\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f)\,,$$

where $f^{\oplus k} : \{0,1\}^{nk} \to \{0,1\}$ is defined by $f^{\oplus k}(x_1,\ldots,x_k) = \sum_{i=1}^{k} f(x_i) \pmod 2$.

Yao's Lemma says that if small circuits cannot compute f with probability better than $1-\delta$ then somewhat smaller circuits cannot compute $f^{\oplus k}$ with probability better than $1/2 + 2(1-\delta)^k$. Intuitively, it makes sense that if you can only compute f on a $1-\delta$ fraction of the inputs, then given a random k tuple $x_1,\ldots,x_k$, unless all of these k inputs fall into this "good set" of inputs (which happens with probability $(1-\delta)^k$), you will have to guess the answer to $\sum_{i=1}^{k} f(x_i) \pmod 2$ at random and be successful with probability at most $1/2$; see also Exercise 19.1. But making this intuition into a proof takes some effort. The main step is the following beautiful result of Impagliazzo.

Lemma 19.3 *(Impagliazzo's Hardcore Lemma* [Imp95b]*) Say that a distribution H over $\{0,1\}^n$ has density δ if for every $x \in \{0,1\}^*$, $\Pr[H = x] \leq 1/(\delta 2^n)$. For every $\delta > 0$, $f : \{0,1\}^n \to \{0,1\}$, and $\epsilon > 0$, if $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$ then there exists a density-δ distribution H such that for every circuit C of size at most $\frac{\epsilon^2 S}{100n}$,*

$$\Pr_{x \in_R H}[C(x) = f(x)] \leq 1/2 + \epsilon\,. \qquad \diamondsuit$$

A priori, one can think that a function f that is hard to compute by small circuits with probability $1-\delta$ could have two possible forms: **(a)** the hardness is sort of "spread" all over the inputs (different circuits make mistakes on different inputs), and the function is roughly $1-\delta$-hard on every significant set of inputs or **(b)** there is a subset H of roughly a δ fraction of the inputs such that on H the function is *extremely hard* (cannot be computed better than $\frac{1}{2} + \epsilon$ for some tiny ϵ) and on the rest of the inputs the function may be even very easy. Such a set may be thought of as lying at the core of the hardness of f and is sometimes called the *hardcore set*. Impagliazzo's Lemma shows that actually *every* hard function has the form **(b)**. (While the Lemma talks about distributions and not sets, it is possible to transform it into a result on sets, see Exercise 19.2.)

19.1.1 Proof of Yao's XOR Lemma using Impagliazzo's Hardcore Lemma.

We now show how to use Impagliazzo's Hardcore Lemma (Lemma 19.3) to prove Yao's XOR Lemma (Theorem 19.2). Let $f : \{0,1\}^n \to \{0,1\}$ be a function such that $\mathsf{H}_{\mathsf{avg}}^{1-\delta}(f) \geq S$, let $k \in \mathbb{N}$ and suppose, towards a contradiction, that there is a circuit C of size $S' = \frac{\epsilon^2}{400n}S$ such that

$$\Pr_{(x_1,\ldots,x_k)\in_R U_n^k}\left[C(x_1,\ldots,x_k) = \sum_{i=1}^{k} f(x_i) \pmod 2\right] \geq 1/2 + \epsilon\,, \tag{1}$$

where $\epsilon > 2(1-\delta)^k$. We will first prove the lemma for the case $k = 2$ and then indicate how the proof can be generalized for every k.

Let H be the hardcore density-δ distribution obtained from Lemma 19.3, on which every S'-sized circuit fails to compute f with probability better than $1/2 + \epsilon/2$. We can think of the process of picking a uniform element in $\{0,1\}^n$ as follows: first toss a biased coin that comes up "Heads" with probability δ. Then, if the coin came up "Heads" then pick a random element according to H, and if it came up "Tails" pick an element according to the distribution G which is the "complement" of H. Namely, G is defined by setting $\Pr[G = x] = (2^{-n} - \delta\Pr[H = x])/(1-\delta)$. (Exercise 19.3 asks you to verify that G is indeed a distribution and that this process does indeed yield a uniform element.) We shorthand this and write

$$U_n = (1-\delta)G + \delta H\,. \tag{2}$$

If we consider the distribution $(U_n)^2$ of picking *two independent* random strings and concatenating them, then by (2) we can write

$$(U_n)^2 = (1-\delta)^2G^2 + (1-\delta)\delta GH + \delta(1-\delta)HG + \delta^2H^2\,, \tag{3}$$

where we use G^2 to denote the concatenation of two independent copies of G, GH to denote the concatenation of a string chosen from G and a string chosen independently from H, and so on.

Now for every distribution $\mathcal{D}$ over $\{0,1\}^{2n}$, let $P_{\mathcal{D}}$ be the probability of the event of the left-hand side of (1). That is, $P_{\mathcal{D}}$ is the probability that $C(x_1,x_2) = f(x_1) + f(x_2) \pmod 2$ where x_1, x_2 are chosen from $\mathcal{D}$. Combining (1) and (3) we get

$$1/2 + \epsilon \leq P_{(U_n)^2} = (1-\delta)^2P_{G^2} + (1-\delta)\delta P_{GH} + \delta(1-\delta)P_{HG} + \delta^2P_{H^2} \tag{4}$$

But since $\epsilon > 2(1-\delta)^2$ and $P_{G^2} \leq 1$, (4) implies

$$1/2 + \epsilon/2 \leq (1-\delta)\delta P_{GH} + \delta(1-\delta)P_{HG} + \delta^2P_{H^2}\,. \tag{5}$$

Since the coefficients on the right hand side of (5) sum up to less than 1, the averaging principle implies that at least one of these probabilities must be larger than the left hand side. For example, assume that $P_{HG} \geq 1/2 + \epsilon/2$ (the other cases are symmetrical). This means that

$$\Pr_{x_1\in_R H, x_2\in_R G}[C(x_1,x_2) = f(x_1) + f(x_2) \pmod 2] > 1/2 + \epsilon/2\,.$$

Thus by the averaging principle, there exists a fixed string x_2 such that

$$\Pr_{x_1\in_R H}[C(x_1,x_2) = f(x_1) + f(x_2) \pmod 2] > 1/2 + \epsilon/2\,,$$

or, equivalently,

$$\Pr_{x_1\in_R H}[C(x_1,x_2) + f(x_2) \pmod 2 = f(x_1)] > 1/2 + \epsilon/2\,.$$

But this means that we have an S'-sized circuit D (the circuit computing the mapping $x_1 \mapsto C(x_1,x_2) + f(x_2) \pmod 2$) that computes f on inputs chosen from H with probability better than $1/2 + \epsilon/2$, contradicting the fact that H is hard-core!

This completes the proof for the case $k = 2$. The proof for general k follows along the same lines, using the equation

$$(U_n)^k = (1-\delta)^k G^k + (1-\delta)^{k-1}\delta G^{k-1}H + \cdots + \delta^k H^k$$

in place of (3); we leave verifying the details to the reader as Exercise 19.4. ■

19.1.2 Proof of Impagliazzo's Lemma

We now turn to proving Impagliazzo's Hardcore Lemma (Lemma 19.3). Let f be a function with $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$ and let $\epsilon > 0$. To prove the lemma we need to show a density δ distribution H on which *every* circuit C of size $S' = \frac{\epsilon^2 S}{100n}$ cannot compute f with probability better than $1/2 + \epsilon$.

Let's think of this task as a game between two players named *Russell* and *Noam*. Noam wants to compute the function f and Russell wants Noam to fail. The game proceeds as follows: Russell first chooses a δ-density distribution H, and then Noam chooses a circuit C of size at most S'. At the game's conclusion, Russell pays Noam v dollars, where $v = \Pr_{x \in_R H}[C(x) = f(x)]$. Assume towards a contradiction that the lemma is false, and hence for every δ-density distribution H chosen by Russell, Noam can find an S'-sized circuit C on which $\Pr_{x \in_R H}[C(x) = f(x)] \geq 1/2 + \epsilon$.

Now this game is a zero-sum game, and so we can use von-Neumann's *min-max theorem* (see Note 19.4) that says that if we allow *randomized* (also known as mixed) strategies then Noam can achieve the same value even if he plays first. By randomized strategies we mean that Noam and Russell can also select arbitrary distributions over their choices. In Russell's case this makes no difference as a distribution over density-δ distributions is still a density-δ distribution.[1] However in Noam's case we need to allow him to choose a *distribution* $\mathcal{C}$ over S'-sized circuits. Our assumption, combined with the min-max theorem, means that there exists such a distribution $\mathcal{C}$ satisfying

$$\Pr_{C \in_R \mathcal{C}, x \in_R H}[C(x) = f(x)] \geq 1/2 + \epsilon \tag{6}$$

for *every* δ-density H.

Call a string $x \in \{0,1\}^n$ "bad" if $\Pr_{C \in_R \mathcal{C}}[C(x) = f(x)] < 1/2 + \epsilon$ and call x "good" otherwise. There are less than $\delta 2^n$ bad x's. Indeed, otherwise we could let H be the uniform distribution over the bad x's and it would violate (6). Now let us choose a circuit C as follows: set $t = 50n/\epsilon^2$, pick $C_1, \ldots, C_t$ independently from $\mathcal{C}$, and define $C(x)$ to equal the majority of $C_1(x), \ldots, C(x)$ for every $x \in \{0,1\}^n$. Note that the size of C is $tS' < S$. We claim that if we choose the circuit C in this way then for every good $x \in \{0,1\}^n$, $\Pr[C(x) \neq f(x)] < 2^{-n}$. Indeed, this follows by applying the Chernoff bound (see Corollary A.15). Since there are at most 2^n good x's, we can apply the union bound to deduce that there exists a size S circuit C such that $C(x) = f(x)$ for *every* good x. But since there are less than $\delta 2^n$ bad x's this implies that $\Pr_{x \in_R U_n}[C(x) = f(x)] > 1 - \delta$, contradicting the assumption that $\mathsf{H}^{1-\delta}_{\mathsf{avg}}(f) \geq S$. ■

Taken in the contrapositive, Lemma 19.3 implies that if for every significant chunk of the inputs there is some circuit that computes f with on this chunk with some advantage over $1/2$, then there is a single circuit that computes f with good probability over all inputs. In machine learning such a result (transforming a way to weakly predict some function into a way to strongly predict it) is called *Boosting* of learning methods. Although the proof we presented here is non-constructive, Impagliazzo's original proof was constructive, and was used to obtain a boosting algorithm yielding some new results in machine learning, see [KS99].

[1] In fact, the set of density δ distributions can be viewed as the set of distributions over $\delta 2^n$-flat distributions, where a distribution is K-flat if it is uniform over a set of size K (see Exercise 19.7). This fact means that we can think of the game as finite and so use the min-max theorem in the form it is stated in Note 19.4.

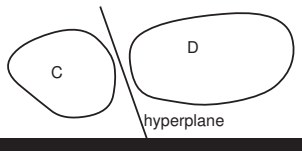
Note 19.4 *(The Min-Max Theorem)*

A *zero sum game* is, as the name implies, a game between two parties in which whatever one party loses is won by the other party. It is modeled by an $m \times n$ matrix $A = (a_{i,j})$ of real numbers. The game consists of only two moves. One party, called the *minimizer* or *column player*, chooses an index $j \in [n]$ while the other party, called the *maximizer* or *row player*, chooses an index $i \in [m]$. The *outcome* is that the column player has to pay $a_{i,j}$ units of money to the row player (if $a_{i,j}$ is negative then the row player pays the column player $|a_{i,j}|$ units). Clearly, the *order* in which players make their moves is important. The min-max theorem says that, surprisingly, if we allow the players randomized strategies, then the order of play is immaterial.

By *randomized* (also known as *mixed*) strategies we mean that the column player chooses a *distribution* over the columns; that is, a vector $\mathbf{p} \in [0,1]^n$ with $\sum_{i=1}^{n} p_i = 1$. Similarly, the row player chooses a distribution $\mathbf{q}$ over the rows. The amount paid is the expectation of $a_{i,j}$ for j chosen from $\mathbf{p}$ and i chosen from $\mathbf{q}$. If we think of $\mathbf{p}$ as a column vector and $\mathbf{q}$ as a row vector then this is equal to $\mathbf{q}A\mathbf{p}$. The min-max theorem says that

$$\min_{\substack{\mathbf{p}\in[0,1]^n \\ \sum_i p_i = 1}} \max_{\substack{\mathbf{q}\in[0,1]^m \\ \sum_i q_i = 1}} \mathbf{q}A\mathbf{p} = \max_{\substack{\mathbf{q}\in[0,1]^m \\ \sum_i q_i = 1}} \min_{\substack{\mathbf{p}\in[0,1]^n \\ \sum_i p_i = 1}} \mathbf{q}A\mathbf{p}\,, \tag{7}$$

As discussed in Exercise 19.6, the Min-Max Theorem can be proven using the following result, known as the *Separating Hyperplane Theorem*: if C and D are disjoint convex subsets of $\mathbb{R}^m$, then there is a hyperplane that separates them. (A subset $C \subseteq \mathbb{R}^m$ is *convex* if whenever it contains a pair of points $\mathbf{x}, \mathbf{y}$, it contains the line segment $\{\alpha\mathbf{x} + (1-\alpha)\mathbf{y} : 0 \leq \alpha \leq 1\}$ with $\mathbf{x}$ and $\mathbf{y}$ as its endpoints.) We ask you to prove (a relaxed variant of) the separating hyperplane theorem in Exercise 19.5 but here is a "proof by picture" for the two dimensional case:



19.2 Tool: Error correcting codes

Our next goal will be to construct average-case hard functions using functions that are only worst-case hard. Our main tool will be *error correcting codes*. An error correcting code maps strings into slightly larger strings in a way that "amplifies differences" in the sense that every two distinct strings (even if they differ by just one bit) get mapped into two strings that are "very far" from one another. The formal definition follows:

Definition 19.5 *(Error Correcting Codes)*
For $x, y \in \{0,1\}^m$, the *fractional Hamming distance* of x and y, denoted $\Delta(x,y)$, is equal to $\frac{1}{m}|\{i : x_i \neq y_i\}|$.

For every $\delta \in [0,1]$, a function $E : \{0,1\}^n \to \{0,1\}^m$ is an *error correcting code (ECC) with distance* δ, if for every $x \neq y \in \{0,1\}^n$, $\Delta(E(x), E(y)) \geq \delta$. We call the set $Im(E) = \{E(x) : x \in \{0,1\}^n\}$ the set of *codewords* of E.

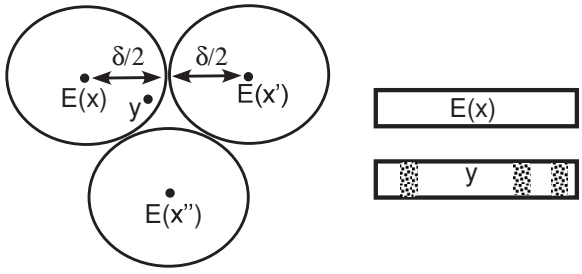


Figure 19.2 In a δ-distance error correcting code, $\Delta(E(x), E(x')) \geq \delta$ for every $x \neq x'$. We can recover x from every string y in which less than $\delta/2$ coordinates were corrupted (i.e., $\Delta(y, E(x)) < \delta/2$) since the $\delta/2$-radius balls around every codeword are disjoint. In the figure above the dotted areas represent corrupted coordinates.

Note that some texts define an error correcting code not as a function $E : \{0,1\}^n \to \{0,1\}^m$ but rather as a 2^n-sized subset of $\{0,1\}^m$ (corresponding to $Im(E)$ in our notation). Error correcting codes have had a vast number of practical and theoretical applications in Computer Science and engineering, but their motivation stems from the following simple application: suppose that Alice wants to transmit a string $x \in \{0,1\}^m$ to Bob, but her channel of communication to Bob is *noisy* and every string y she sends might be corrupted in as many as 10% of its coordinates. That is, her only guarantee is that Bob would receive a string y' satisfying $\Delta(y,y') \leq 0.1$. Alice can perform this task using an error correcting code $E : \{0,1\}^n \to \{0,1\}^m$ of distance $\delta > 0.2$. The idea is that she sends to Bob $y = E(x)$ and Bob receives a string y' satisfying $\Delta(y,y') \leq 0.1$. Since $\Delta(y, E(w)) > 0.2$ for every $w \neq x$, it follows that y is the unique codeword of E that is of distance at most 0.1 from y' and so Bob can find y and from it find x such that $E(x) = y$ (see Figure 19.2). One can see from this example that we'd want codes with as large a distance δ as possible, as small output length m as possible, and of course we'd like both Alice and Bob to be able to carry the encoding and decoding efficiently. The following lemma shows that, ignoring issues of computational efficiency, pretty good error correcting codes exist:

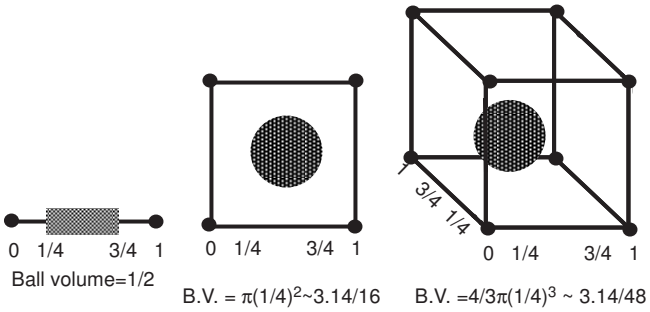
Lemma 19.6 *(Gilbert-Varshamov Bound) For every $\delta < 1/2$ and sufficiently large n, there exists a function $E : \{0,1\}^n \to \{0,1\}^{n/(1-H(\delta))}$ that is an error correcting code with distance δ, where $H(\delta) = \delta\log(1/\delta) + (1-\delta)\log(1/(1-\delta))$.*[2] ♢

Proof: We prove a slightly weaker statement: the existence of a δ-distance ECC $E : \{0,1\}^n \to \{0,1\}^m$ where $m = 2n/(1-H(\delta))$ instead of $m = n/(1-H(\delta))$. To do so, we

[2] $H(\delta)$ is called the *Shannon entropy function*. It is not hard to see that $H(1/2) = 1, H(0) = 0$, and $H(\delta) \in (0,1)$ for every $\delta \in (0, 1/2)$.

Note 19.7 *(High dimensional geometry)*

While we are normally used to geometry in two or three dimensions, we can get some intuition on error correcting codes by considering the geometry of *high dimensional spaces*. Perhaps the strongest effect of high dimension is the following: compare the volume of the cube with all sides 1 and the ball of radius 1/4. In one dimension, the ratio between these volumes is $1/(1/2) = 2$, in two dimensions it is $1/(\pi/4^2) = 16/\pi$, while in three dimensions it is $1/(4/3\pi/4^3) = 48/\pi$. As the number of dimension grows, this ratio grows exponentially in the number of dimensions. (The volume of a ball of radius r in m dimensions is roughly $\frac{\pi^{m/2}}{\lfloor m/2 \rfloor !} r^m$.) Similarly for any two radii $r_1 > r_2$, the volume of the m-dimension ball of radius r_1 is exponentially larger than the volume of the r_2-radius ball.



This intuition lies behind the existence of an error correcting code with, say, distance 1/4 mapping n bit strings into $m = 5n$ bit strings. We can have $2^{m/5}$ codewords that are all of distance at least 1/4 from one another because, also in the discrete setting the volume (i.e., number of points contained) of the radius-1/4 ball is exponentially smaller than the volume of the cube $\{0,1\}^n$. Therefore, we can "pack" $2^{m/5}$ such balls within the cube.

simply choose E at random. That is, we choose 2^n random strings $y_1, y_2, \ldots, y_{2^n} \in \{0,1\}^m$ and E maps the input $x \in \{0,1\}^n$ (which we can identify with a number in $[2^n]$) to the string y_x.

It suffices to show that the probability that for some $i < j$ with $i, j \in [2^n]$, $\Delta(y_i, y_j) < \delta$ is less than 1. But for every string y_i, the number of strings that are of distance at most δ to it is $\binom{m}{\lceil \delta m \rceil}$ which is less than $0.99 \cdot 2^{H(\delta)m}$ for m sufficiently large (see Appendix A) and so for every $j > i$, the probability that y_j falls in this ball is bounded by $0.99 \cdot 2^{H(\delta)m}/2^m$. Since there are at most 2^{2n} such pairs i, j, we only need to show that $0.99 \cdot 2^{2n} \frac{2^{H(\delta)m}}{2^m} < 1$, which is indeed the case for our choice of m. By a slightly more clever argument, we can prove the lemma as stated: see Exercise 19.9. It turns out that as δ tends to zero, there do exist codes with smaller values of m than $n/(1 - H(\delta))$, but it is not known whether or not Lemma 19.6 is optimal for δ tending to $1/2$. ■

Why half? Lemma 19.6 only provides codes of distance δ for $\delta < 1/2$ and you might wonder whether this is inherent or perhaps codes of even greater distance exist. It turns out we can have codes of distance $1/2$ but only if we allow m to be exponentially larger than n (i.e., $m \geq 2^{n-1}$). For every $\delta > 1/2$, if n is sufficiently large then there is no ECC $E : \{0,1\}^n \to \{0,1\}^m$ that has distance δ, no matter how large m is. Both these bounds are explored in Exercise 19.10.

19.2.1 Explicit codes

The mere existence of an error correcting code is not sufficient for most applications: we need to be able to actually compute them. For this we need to show an *explicit function*

$E : \{0,1\}^n \to \{0,1\}^m$ that is an error correcting satisfying the following properties:

Efficient encoding There is a poly(m) time algorithm to compute $E(x)$ from x.

Efficient decoding There is a polynomial time algorithm to compute x from every y such that $\Delta(y, E(x)) < \rho$ for some ρ. For this to be possible, the number ρ must be less than $\delta/2$, where δ is the distance of E: see Exercise 19.11.

We now describe some explicit functions that are error correcting codes.

19.2.2 Walsh-Hadamard Code.

For two strings $x, y \in \{0,1\}^n$, we define $x \odot y = \sum_{i=1}^{n} x_i y_i \pmod 2$. The *Walsh-Hadamard code* is the function $\mathsf{WH} : \{0,1\}^n \to \{0,1\}^{2^n}$ that maps every string $x \in \{0,1\}^n$ into the string $z \in \{0,1\}^{2^n}$ satisfying $z_y = x \odot y$ for every $y \in \{0,1\}^n$ (where z_y denotes the y^{th} coordinate of z, identifying $\{0,1\}^n$ with $[2^n]$ in some canonical way).

Claim 19.8 *The function* WH *is an error correcting code of distance* $1/2$. ♢

Proof: First, note that WH is a linear function. That is, $\mathsf{WH}(x+y) = \mathsf{WH}(x) + \mathsf{WH}(y)$, where $x + y$ denotes the componentwise addition of x and y modulo 2 (i.e., bitwise XOR). Thus, for every $x \neq y \in \{0,1\}^n$ the number of 1's in the string $\mathsf{WH}(x)+\mathsf{WH}(y) = \mathsf{WH}(x+y)$ is equal to the number of coordinates on which $\mathsf{WH}(x)$ and $\mathsf{WH}(y)$ differ. Thus, it suffices to show that for every $w \neq 0^n$, at least half of the coordinates in $\mathsf{WH}(w)$ are 1. Yet this follows from the random subsum principle (Claim A.31) that says that the probability that $w \odot y = 1$ for $y \in_R \{0,1\}^n$ is exactly $1/2$. ■

19.2.3 Reed-Solomon Code

The Walsh-Hadamard code has a serious drawback: its output size is exponential in the input size. By Lemma 19.6 we know that we can do much better (at least if we're willing to tolerate a distance slightly smaller than $1/2$). To get towards explicit codes with better output, we'll make a detour via codes with *non-binary* alphabet.

Definition 19.9 For every finite set Σ and $x, y \in \Sigma^m$, we define $\Delta(x,y) = \frac{1}{m}|\{i : x_i \neq y_i\}|$. We say that $E : \Sigma^n \to \Sigma^m$ is an *error correcting code with distance* δ *over alphabet* Σ if for every $x \neq y \in \Sigma^n$, $\Delta(E(x), E(y)) \geq \delta$. ♢

Allowing a larger alphabet makes the problem of constructing codes easier. For example, every ECC with distance δ over the binary ($\{0,1\}$) alphabet automatically implies an ECC with the same distance over the alphabet $\{0,1,2,3\}$: just encode strings over $\{0,1,2,3\}$ as strings over $\{0,1\}$ in the obvious way. However, the other direction does not work: if we take an ECC over $\{0,1,2,3\}$ and transform it into a code over $\{0,1\}$ in the natural way, the distance might grow from δ to 2δ (see Exercise 19.12). The Reed-Solomon code is a construction of an error correcting code that can use as its alphabet any sufficiently large field $\mathbb{F}$:

Definition 19.10 *(Reed-Solomon code)* Let $\mathbb{F}$ be a field and n, m numbers satisfying $n \leq m \leq |\mathbb{F}|$. The *Reed-Solomon code* from $\mathbb{F}^n$ to $\mathbb{F}^m$ is the function $\mathsf{RS} : \mathbb{F}^n \to \mathbb{F}^m$ that on input $a_0, \ldots, a_{n-1} \in \mathbb{F}^n$ outputs the string $z_0, \ldots, z_{m-1}$ where $z_j = \sum_{i=0}^{n-1} a_i f_j^i$, and f_j denotes the j^{th} element of $\mathbb{F}$ under some ordering. ♢

Note that an equivalent way of defining the Reed Solomon code is that it takes as input a description of the $n-1$ degree polynomial $A(x) = \sum_{i=1}^{n-1} a_i x^i$ and outputs the evaluation of A on the points $f_0, \ldots, f_{m-1}$.

Lemma 19.11 *The Reed-Solomon code* $\mathsf{RS} : \mathbb{F}^n \to \mathbb{F}^m$ *has distance* $1 - \frac{n}{m}$. ♢

PROOF: As in the case of Walsh-Hadamard code, the function RS is also linear in the sense that $\mathsf{RS}(a+b) = \mathsf{RS}(a) + \mathsf{RS}(b)$ (where addition is taken to be componentwise addition in $\mathbb{F}$). Thus, as before we only need to show that for every $a \neq 0^n$, $\mathsf{RS}(a)$ has at most n coordinates that are zero. But this immediate from the fact that a nonzero $n-1$ degree polynomial has at most n roots (see Appendix A). ■

19.2.4 Reed-Muller Codes.

Both the Walsh-Hadamard and and the Reed-Solomon code are special cases of the following family of codes known as Reed-Muller codes:

Definition 19.12 *(Reed-Muller codes)* Let $\mathbb{F}$ be a finite field, and let ℓ, d be numbers with $d < |\mathbb{F}|$. The *Reed Muller* code with parameters $\mathbb{F}, \ell, d$ is the function $\mathsf{RM} : \mathbb{F}^{\binom{\ell+d}{d}} \to \mathbb{F}^{|\mathbb{F}|^\ell}$ that maps every ℓ-variable polynomial P over $\mathbb{F}$ of total degree d to the values of P on all the inputs in $\mathbb{F}^\ell$.

That is, the input is a polynomial of the form

$$P(x_1, \ldots, x_\ell) = \sum_{i_1+i_2+\ldots+i_\ell \leq \ell} c_{i_1,\ldots,i_\ell} x_1^{i_1} x_2^{i_2} \cdots x_\ell^{i_\ell}$$

specified by the vector of $\binom{\ell+d}{d}$ coefficients $\{c_{i_1,\ldots,i_\ell}\}$ and the output is the sequence $\{P(x_1,\ldots,x_\ell)\}$ for every $x_1, \ldots, x_\ell \in \mathbb{F}$. ♢

Setting $\ell = 1$ one obtains the Reed-Solomon code (for $m = |\mathbb{F}|$), while setting $d = 1$ and $\mathbb{F} = \mathrm{GF}(2)$ one obtains a slight variant of the Walsh-Hadamard code (i.e., the code that maps every $x \in \{0,1\}^n$ into a $2 \cdot 2^n$ long string z satisfying $z_{y,a} = x \odot y + a \pmod 2$ for every $y \in \{0,1\}^n$,$a \in \{0,1\}$). The Schwartz-Zippel Lemma (Lemma A.36 in Appendix A) shows that the Reed-Muller code is an ECC with distance $1 - d/|\mathbb{F}|$. Note that this implies the previously stated bounds for the Walsh-Hadamard and Reed-Solomon codes.

19.2.5 Concatenated codes

The Walsh-Hadamard code has the drawback of exponential-sized output and the Reed-Solomon code has the drawback of a non-binary alphabet. We now show we can combine them both to obtain a code without neither of these drawbacks:

Definition 19.13 If RS is the Reed-Solomon code mapping $\mathbb{F}^n$ to $\mathbb{F}^m$ (for some $n, m, \mathbb{F}$) and WH is the Walsh-Hadamard code mapping $\{0,1\}^{\log|\mathbb{F}|}$ to $\{0,1\}^{2^{\log|\mathbb{F}|}} = \{0,1\}^{|\mathbb{F}|}$, then the code $\mathsf{WH} \circ \mathsf{RS}$ maps $\{0,1\}^{n\log|\mathbb{F}|}$ to $\{0,1\}^{m|\mathbb{F}|}$ in the following way:

1. View RS as a code from $\{0,1\}^{n\log|\mathbb{F}|}$ to $\mathbb{F}^m$ and WH as a code from $\mathbb{F}$ to $\{0,1\}^{|\mathbb{F}|}$ using the canonical representation of elements in $\mathbb{F}$ as strings in $\{0,1\}^{\log|\mathbb{F}|}$.

2. For every input $x \in \{0,1\}^{n\log|\mathbb{F}|}$, $\mathsf{WH}\circ\mathsf{RS}(x)$ is equal to $\mathsf{WH}(\mathsf{RS}(x)_1), \ldots, \mathsf{WH}(\mathsf{RS}(x)_m)$ where $\mathsf{RS}(x)_i$ denotes the i^{th} symbol of $\mathsf{RS}(x)$.

Note that the code $\mathsf{WH} \circ \mathsf{RS}$ can be computed in time polynomial in n, m and $|\mathbb{F}|$. We now analyze its distance:

Claim 19.14 *Let $\delta_1 = 1 - n/m$ be the distance of RS and $\delta_2 = 1/2$ be the distance of WH. Then $\mathsf{WH} \circ \mathsf{RS}$ is an ECC of distance $\delta_1\delta_2$.* ♢

PROOF: Let x, y be two distinct strings in $\{0,1\}^{\log|\mathbb{F}|n}$. If we set $x' = \mathsf{RS}(x')$ and $y' = \mathsf{RS}(y')$ then $\Delta(x', y') \geq \delta_1$. If we let x'' (resp. y'') to be the binary string obtained by applying WH

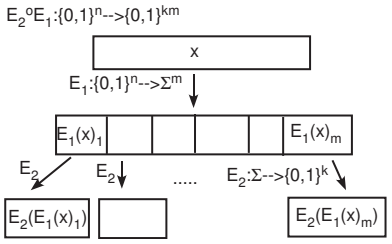


Figure 19.3 If E_1,E_2 are ECC's such that $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \sigma \to \{0,1\}^k$, then the concatenated code $E : \{0,1\}^n \to \{0,1\}^{nk}$ maps x into the sequence of blocks $E_2(E_1(x)_1), \ldots, E_2(E_1(x)_m)$.

to each of these blocks, then whenever two blocks are distinct, the corresponding encoding will have distance δ_2, and so $\delta(x'', y'') \geq \delta_1\delta_2$. ■

Because for every $k \in \mathbb{N}$, there exists a finite field $|\mathbb{F}|$ of size in $[k, 2k]$ (e.g., take a prime in $[k, 2k]$ or a power of two) we can use this construction to obtain, for every n, a polynomial-time computable ECC $E : \{0,1\}^n \to \{0,1\}^{20n^2}$ of distance 0.4.

Both Definition 19.13 and Lemma 19.14 easily generalize for codes other than Reed-Solomon and Hadamard. Thus, for every two ECC's $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ their concatenation $E_2 \circ E_1$ is a code from $\{0,1\}^n$ to $\{0,1\}^{mk}$ that has distance at least $\delta_1\delta_2$ where δ_1 (resp. δ_2) is the distance of E_1 (resp. E_2), see Figure 19.3. In particular, using a different binary code than WH, it is known how to use concatenation to obtain a polynomial-time computable ECC $E : \{0,1\}^n \to \{0,1\}^m$ of constant distance $\delta > 0$ such that $m = O(n)$, see Exercise 19.18.

19.3 Efficient decoding.

To actually use an error correcting code to store and retrieve information, we need a way to efficiently *decode* a message x from its encoding $E(x)$ even if this encoding has been corrupted in some fraction ρ of its coordinates. We now show how to do this for the Reed-Solomon code and for concatenated codes.

19.3.1 Decoding Reed-Solomon

Recall that the Reed-Solomon treats its input as describing a polynomial and outputs the values of this polynomial on m inputs. We know (see Theorem A.35 in Appendix A) that a univariate degree d polynomial can be interpolated from any $d+1$ values. Here we consider a *robust* version of this procedure, whereby we wish to recover the polynomial from m values of which ρm are "faulty" or "noisy".

Theorem 19.15 *(Unique decoding for Reed-Solomon* [BW86]*)*
There is a polynomial-time algorithm that given a list $(a_1, b_1), \ldots, (a_m, b_m)$ of pairs of elements of a finite field $\mathbb{F}$ such that there is a d-degree polynomial $G : \mathbb{F} \to \mathbb{F}$ satisfying $G(a_i) = b_i$ for t of the numbers $i \in [m]$, with $t > \frac{m}{2} + \frac{d}{2}$, recovers G.

Since Reed-Solomon is an ECC with distance $1 - \frac{d}{m}$, Theorem 19.15 means that we can efficiently recover the correct polynomial from a version corrupted in ρ places as long as ρ is smaller than half the distance. This is optimal in the sense that once the fraction of

errors is larger than half the distance we are no longer guaranteed the existence of a unique solution.

PROOF OF THEOREM 19.15.: As a warmup, we start by considering the case that the number of errors is very small. (This setting is still sufficiently strong for many applications.)

RANDOMIZED INTERPOLATION: THE CASE OF $t \geq (1 - \frac{1}{2(d+1)})m$

Assume that t is quite large: $t > (1 - \frac{1}{2(d+1)})m$. In this case, we can just select $d+1$ pairs $(x_1, y_1), \ldots, (x_{d+1}, y_{d+1})$ at random from the set $\{(a_i, b_i)\}$ and use standard polynomial interpolation to compute the unique a d-degree polynomial P such that $P(x_j) = y_j$ for all $j \in [d+1]$. We then check whether P agrees with at least t pairs of the entire sequence and if so we output P (otherwise we try again). By the union bound, the probability that $x_j \neq G(y_j)$ for one of the $d+1$ chosen pairs is at most $(d+1)\frac{m-t}{t} \leq 1/2$, and hence with probability at least $1/2$ it will be the case that $P = G$.

BERLEKAMP-WELCH PROCEDURE: THE CASE OF $t \geq \frac{m}{2} + \frac{d}{2} + 1$

We now prove Theorem 19.15 using a procedure known as the *Berlekamp-Welch decoding*. For simplicity of notations, we assume that $m = 4d$ and $t = 3d$. However, the proof generalizes to any parameters m, d, t satisfying $t > \frac{m}{2} + \frac{d}{2}$, see Exercise 19.13. Thus, we assume that there exists a d-degree polynomial G such that

$$G(a_i) = b_i \text{ for at least } 3d \text{ of } i\text{'s in } [m] = [4d]. \tag{8}$$

We will use the following decoding procedure:

1. Find a degree $2d$ polynomial $C(x)$ and a degree-d nonzero polynomial $E(x)$ such that:

$$C(a_i) = b_i E(a_i) \text{ for every } i \in [m] \tag{9}$$

   This can be done by considering (9) as a set of $4d$ linear equations with the unknowns being the $2d+1$ coefficients of $C(x)$ and the $d+1$ coefficients of E. These equations have a solution with nonzero $E(x)$ since one can define $E(x)$ to a nonzero polynomial that is equal to zero on every a_i such that $G(a_i) \neq b_i$ (under our assumption (8) there are at most d such places).[3]

2. Divide C by E: get a polynomial P such that $C(x) = E(x)P(x)$ (we will show that E divides C without remainder). Output P.

   We know by (8) and (9) that $C(x) = G(x)E(x)$ for at least $3d$ values, meaning that $C(x) - G(x)E(x)$ is a degree $2d$ polynomial with at least $3d$ roots. This means that this polynomial is identically zero (i.e., $C(x) = G(x)E(x)$ for every $x \in \mathbb{F}$). Thus it does indeed hold that $G = C/E$.

■

19.3.2 Decoding concatenated codes.

Decoding concatenated codes can be achieved through the natural algorithm. Recall that if $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ are two ECC's then $E_2 \circ E_1$ maps every string $x \in \{0,1\}^n$ to the string $E_2(E_1(x)_1) \cdots E_2(E_1(x)_n)$. Suppose that we have a decoder for E_1 (resp. E_2) that can handle ρ_1 (resp. ρ_2) errors. Then, we have a decoder for $E_2 \circ E_1$ that can handle $\rho_2\rho_1$ errors. The decoder, given a string $y \in \{0,1\}^{mk}$ composed of m blocks $y_1, \ldots, y_m \in \{0,1\}^k$, first decodes each block y_i to a symbol z_i in Σ, and then uses the decoder of E_1 to decode $z_1, \ldots, z_m$. The decoder can indeed handle $\rho_1\rho_2$ errors since if $\Delta(y, E_2 \circ E_1(x)) \leq \rho_1\rho_2$ then at most ρ_1 of the blocks of y are of distance at least ρ_2 from the corresponding block of $E_2 \circ E_1(x)$.

[3] One can efficiently find such a solution by trying to solve the equations after adding to them an equation of the form $E_j = e_j$ where E_j is the j^{th} coefficient of $E(x)$ and e_j is a nonzero element of $\mathbb{F}$. The number of such possible equations is polynomial and at least one of them will result in a satisfiable set of equations.

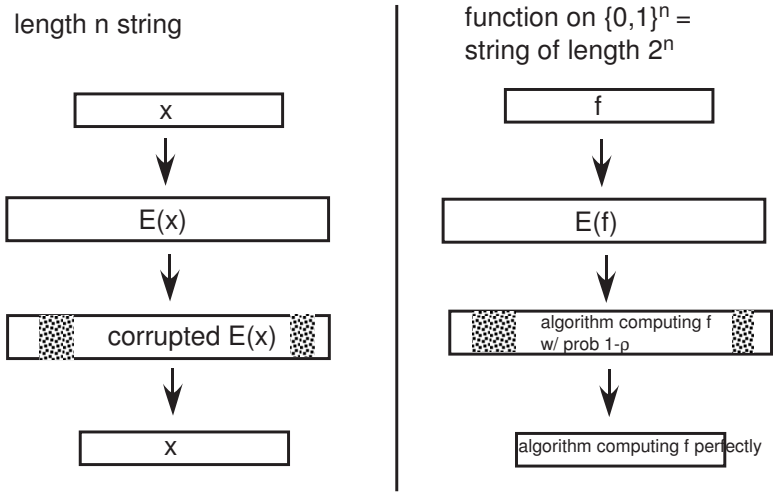


Figure 19.4 An ECC allows us to map a string x to $E(x)$ such as x can be reconstructed from a corrupted version of $E(x)$. The idea is to treat a function $f : \{0,1\}^n \to \{0,1\}$ as a string in $\{0,1\}^{2^n}$, encode it using an ECC to a function $\hat{f}$. Intuitively, $\hat{f}$ should be hard on the average case if f was hard on the worst case, since an algorithm to solve $\hat{f}$ with probability $1-\rho$ could be transformed (using the ECC's decoding algorithm) to an algorithm computing f on every input.

19.4 Local decoding and hardness amplification

We now show the connection between error correcting codes and hardness amplification. The idea is actually quite simple (see also Figure 19.4). A function $f : \{0,1\}^n \to \{0,1\}$ can be viewed as a binary string of length $N = 2^n$. Suppose we encode f to a string $\hat{f} \in \{0,1\}^M$ using an ECC mapping $\{0,1\}^N$ to $\{0,1\}^M$ with distance larger than, say, 0.2. Then we can view $\hat{f}$ as a function from $\{0,1\}^{\log M}$ to $\{0,1\}$ and at least in principle it should be possible to recover f from a corrupted version of $\hat{f}$ where, say, at most 10% of the locations have been modified. In other words, if it is possible to compute $\hat{f}$ with probability at least 0.9 then it should be possible to compute f exactly. Taking the contrapositive this means that if f is hard to compute in the worst-case then $\hat{f}$ is hard to compute in the average case!

To make this idea work we need to show we can transform every circuit that correctly computes many bits of $\hat{f}$ into a circuit that correctly computes all the bits of f. This is formalized using a *local decoder* (see Figure 19.5), which is a decoding algorithm that given *random access* to a (possibly corrupted) codeword y' close to $E(x)$ can compute any any desired bit of the original input x. Since we are interested in the circuits that could be of size as small as poly(n)— in other words, *polylogarithmic* in $N = 2^n$ —this must also be the running time of the local decoder.

Definition 19.16 *(Local decoder)*
Let $E : \{0,1\}^n \to \{0,1\}^m$ be an ECC and let ρ and q be some numbers. A *local decoder for E handling ρ errors* is an algorithm D that, given random access to a string y such that $\Delta(y, E(x)) < \rho$ for some (unknown) $x \in [n]$, and an index $j \in \mathbb{N}$, runs for polylog(m) time and outputs x_j with probability at least $2/3$.

The constant $2/3$ is arbitrary and can be replaced with any constant larger than $1/2$, since the probability of getting a correct answer can be amplified by repetition. We also note that Definition 19.16 can be easily generalized for codes with larger (i.e., non binary) alphabet. Local decoding may also be useful in applications of ECC's that have nothing to do with hardness amplification (e.g., if we use ECC's to encode a huge file, we may want to be able

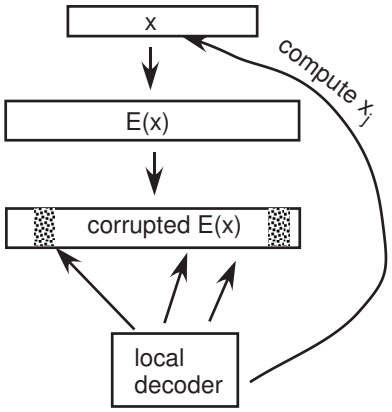


Figure 19.5 A local decoder gets access to a corrupted version of $E(x)$ and an index i and computes from it x_i (with high probability).

to efficiently recover part of the file without decoding it in its entirety). The connection between local decoders and hardness amplification is encapsulated in the following theorem:

Theorem 19.17 *(Hardness amplification from local decoding)*
Suppose that there exists an ECC with polynomial-time encoding algorithm and a local decoding algorithm handling ρ errors. Suppose also that there is $f \in \mathbf{E}$ with $\mathsf{H}_{wrs}(f)(n) \geq S(n)$ for some function $S : \mathbb{N} \to \mathbb{N}$ satisfying $S(n) \geq n$. Then, there exists $\epsilon > 0$ and $\hat{f} \in \mathbf{E}$ with $\mathsf{H}_{avg}^{1-\rho}(\hat{f})(n) \geq S(\epsilon n)^{\epsilon}$

We leave the proof of Theorem 19.17, which follows the ideas described above, as Exercise 19.14. We now show *local decoder* algorithms for several explicit codes.

19.4.1 Local decoder for Walsh-Hadamard.

The following is a two-query local decoder for the Walsh-Hadamard code that handles ρ errors for every $\rho < 1/4$. This fraction of errors we handle is best possible, as it can be easily shown that there cannot exist a local (or non-local) decoder for a binary code handling ρ errors for every $\rho \geq 1/4$.

Theorem 19.18 *For every $\rho < 1/4$, the walsh-Hadamard code has a local decoder handling ρ errors.* ♢

Proof: Theorem 19.18 is proven by the following algorithm:

Walsh-Hadamard Local Decoder for $\rho < 1/4$:

Input: $j \in [n]$, random access to a function $f : \{0,1\}^n \to \{0,1\}$ such that $\Pr_y[g(y) \neq x \odot y] \leq \rho$ for some $\rho < 1/4$ and $x \in \{0,1\}^n$.

Output: A bit $b \in \{0,1\}$. *(Our goal: $x_j = b$.)*

Operation: Let e^j be the vector in $\{0,1\}^n$ that is equal to 0 in all the coordinates except for the j^{th} and equal to 1 on the j^{th} coordinate. The algorithm chooses $y \in_R \{0,1\}^n$ and outputs $f(y) + f(y + e^j) \pmod 2$ (where $y + e^j$ denotes componentwise addition modulo 2, or equivalently, flipping the j^{th} coordinate of y).

Analysis: Since both y and $y + e^j$ are uniformly distributed (even though they are dependent), the union bound implies that with probability $1 - 2\rho$, $f(y) = x \odot y$ and

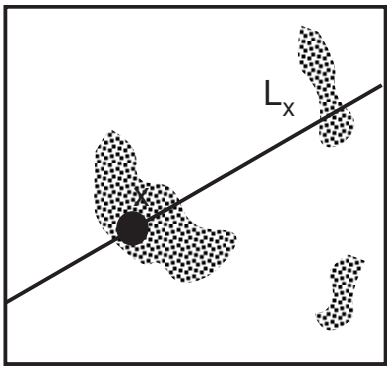


Figure 19.6 Given access to a corrupted version of a polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$, to compute $P(x)$ we pass a random line L_x through x, and use Reed-Solomon decoding to recover the restriction of P to the line L_x.

$f(y + e^j) = x \odot (y + e^j)$. But by the bilinearity of the operation $\odot$, this implies that $f(y) + f(y + e^j) = x \odot y + x \odot (y + e^j) = 2(x \odot y) + x \odot e^j = x \odot e^j \pmod 2$. Yet, $x \odot e^j = x_j$ and so with probability $1 - 2\rho$, the algorithm outputs the right value. (The success probability can be amplified by repetition.) ■

This algorithm can be modified to locally compute not just $x_i = x \odot e^j$ but in fact the value $x \odot z$ for every $z \in \{0,1\}^n$. Thus, we can use it to compute not just every bit of the original message x but also every bit of the uncorrupted codeword $\mathsf{WH}(x)$. This property is sometimes called the *self correction property* of the Walsh-Hadamard code. ■

19.4.2 Local decoder for Reed-Muller

We now show a local decoder for the Reed-Muller code. It runs in time polynomial in ℓ and d, which, for an appropriate setting of the parameters, is polylogarithmic in the output length of the code:

Theorem 19.19 *For every field $|\mathbb{F}|$, numbers d, ℓ and there is a $\mathrm{poly}(|\mathbb{F}|, \ell, d)$-time local decoder for the Reed-Muller code with parameters $\mathbb{F}, d, \ell$ handling $(1 - \frac{d}{|\mathbb{F}|})/6$ errors.*

That is, there is a $\mathrm{poly}(|\mathbb{F}|, \ell, d)$-time algorithm D that given random access to a function $f : \mathbb{F}^\ell \to \mathbb{F}$ that agrees with some degree d polynomial P on a $1 - (1 - \frac{d}{|\mathbb{F}|})/6$ fraction of the inputs and $x \in \mathbb{F}^\ell$ outputs $P(x)$ with probability at least $2/3$. ♢

Proof: Recall that the input to a Reed-Muller code is an ℓ-variable d-degree polynomial P over some field $\mathbb{F}$. When we discussed the code before, we assumed that this polynomial is represented as the list of its coefficients. However, below it will be more convenient for us to assume that the polynomial is represented by a list of its values on its first $\binom{d+\ell}{\ell}$ inputs according to some canonical ordering. Using standard interpolation, we still have a polynomial-time encoding algorithm even given this representation. Thus, it suffices to show an algorithm that, given access to a corrupted version of P, computes $P(x)$ for every $x \in \mathbb{F}^\ell$. We now show such an algorithm:

Reed-Muller Local Decoder for $\rho \leq (1 - \frac{d}{|\mathbb{F}|})/6$.

Input: A string $x \in \mathbb{F}^\ell$, random access to a function f such that $\Pr_{x \in \mathbb{F}^\ell}[P(x) \neq f(x)] < \rho$, where $P : \mathbb{F}^\ell \to \mathbb{F}$ is an ℓ-variable degree-d polynomial.

Output: $y \in \mathbb{F}$ *(Goal: $y = P(x)$.)*

Operation: 1. Let L_x be a random line passing through x. That is $L_x = \{x + tz : t \in \mathbb{F}\}$ for a random $z \in \mathbb{F}^\ell$.

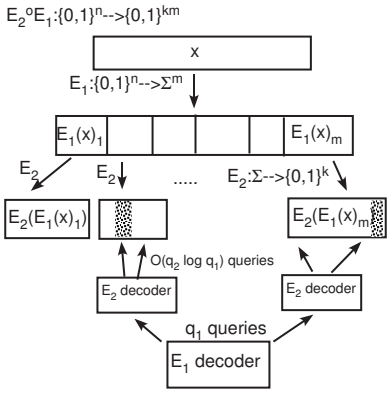


Figure 19.7 To locally decode a concatenated code $E_2 \circ E_1$ we run the decoder for E_1 using the decoder for E_2. The crucial observation is that if y is within $\rho_1\rho_2$ distance to $E_2 \circ E_1(x)$ then at most a ρ_1 fraction of the blocks in y are of distance more than ρ_2 the corresponding block in $E_2 \circ E_1(x)$.

2. Query f on all the $|\mathbb{F}|$ points of L_x to obtain a set of points $\{(t, f(x+tz))\}$ for every $t \in \mathbb{F}$.
3. Run the Reed-Solomon decoding algorithm to obtain the univariate polynomial $Q : \mathbb{F} \to \mathbb{F}$ such that $Q(t) = f(x+tz)$ for the largest number of t's (see Figure 19.6).[4]
4. Output $Q(0)$.

Analysis: For every d-degree ℓ-variable polynomial P, the univariate polynomial $Q(t) = P(x+tz)$ has degree at most d. Thus, to show that the Reed-Solomon decoding works, it suffices to show that with probability at least $2/3$, the number of points on $w \in L_x$ for which $f(w) \neq P(w)$ is less than $(1 - d/|\mathbb{F}|)/2$. Yet, for every $t \neq 0$, the point $x+tz$ where z is chosen at random in $\mathbb{F}^\ell$ is uniformly distributed (independently of x), and so the expected number of points on L_x for which f and P differ is at most $\rho|\mathbb{F}|$. By the Markov inequality, the probability that there will be more than $3\rho|\mathbb{F}| < (1 - d/|\mathbb{F}|)|\mathbb{F}|/2$ such points is at most $2/3$ and hence Reed-Solomon decoding will be successful with probability $2/3$. In this case, we obtain the correct polynomial q that is the restriction of Q to the line L_x and hence $q(0) = P(x)$.

■

19.4.3 Local decoding of concatenated codes.

As the following lemma shows, given two locally decodable ECC's E_1 and E_2, we can locally decode their concatenation $E_1 \circ E_2$:

Lemma 19.20 *Let $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ be two ECC's with local decoders of q_1 (resp. q_2) queries with respect to ρ_1 (resp. ρ_2) errors. Then there is an $O(q_1 q_2 \log q_1 \log|\Sigma|)$-query local decoder handling $\rho_1\rho_2$ errors for the concatenated code $E = E_2 \circ E_1 : \{0,1\}^n \to \{0,1\}^{mk}$.* ♢

Proof: We prove the lemma using the natural algorithm. Namely, we run the decoder for E_1, but answer its queries using the decoder for E_2 (see Figure 19.7).

Local decoder for concatenated code: $\rho < \rho_1\rho_2$

[4] If ρ is sufficiently small, (e.g., $\rho < 1/(10d)$), then we can use the simpler randomized Reed-Solomon decoding procedure described in Section 19.3.

Input: An index $i \in [n]$, random access to a string $y \in \{0,1\}^{km}$ such that $\Delta(y, E_1 \circ E_2(x)) < \rho_1\rho_2$ for some $x \in \{0,1\}^n$.

Output: $b \in \{0,1\}^n$ *(Goal: $b = x_i$)*

Operation: Simulate the actions of the decoder for E_1, whenever the decoder needs access to the j^{th} symbol of $E_1(x)$, use the decoder of E_2 with $O(q_2 \log q_1 \log |\Sigma|)$ queries applied to the j^{th} block of y to recover all the bits of this symbol with probability at least $1 - 1/(10q_1)$.

Analysis: The crucial observation is that at most a ρ_1 fraction of the length k blocks in y can be of distance more than ρ_2 from the corresponding blocks in $E_2 \circ E_1(x)$. Therefore, with probability at least 0.9, all our q_1 answers to the decoder of E_1 are consistent with the answer it would receive when accessing a string that is of distance at most ρ_1 from a codeword of E_1.

■

19.4.4 Putting it all together.

We now have the ingredients to prove our second main theorem of this chapter: transformation of a hard-on-the-worst-case function into a function that is "mildly" hard on the average case.

Theorem 19.21 *(Worst-case hardness to mild hardness)*
Let $S : \mathbb{N} \to \mathbb{N}$ and $f \in \mathbf{E}$ such that $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ for every n. Then there exists a function $g \in \mathbf{E}$ and a constant $c > 0$ such that $\mathsf{H}^{0.99}_{\mathsf{avg}}(g)(n) \geq S(n/c)/n^c$ for every sufficiently large n.

PROOF: For every n, we treat the restriction of f to $\{0,1\}^n$ as a string $f' \in \{0,1\}^N$ where $N = 2^n$. We then encode this string f' using a suitable error correcting code $E : \{0,1\}^N \to \{0,1\}^{N^C}$ for some constant $C > 1$. We will define the function g on every input $x \in \{0,1\}^{Cn}$ to output the x^{th} coordinate of $E(f')$.[5] For the function g to satisfy the conclusion of the theorem, all we need is for the code E to satisfy the following properties:

1. For every $x \in \{0,1\}^N$, $E(x)$ can be computed in $\mathrm{poly}(N)$ time.
2. There is a local decoding algorithm for E that uses $\mathrm{polylog}(N)$ running time and queries and can handle a 0.01 fraction of errors.

But this can be achieved using a concatenation of a Walsh-Hadamard code with a Reed-Muller code of appropriate parameters:

1. Let RM denote the Reed-Muller code with the following parameters:
   - The field $\mathbb{F}$ is of size $\log^5 N$.
   - The number of variables ℓ is equal to $\log N / \log\log N$.
   - The degree is equal to $\log^2 N$.

   RM takes an input of length at least $(\frac{d}{\ell})^\ell > N$ (and so using padding we can assume its input is $\{0,1\}^n$). Its output is of size $|\mathbb{F}|^\ell \leq \mathrm{poly}(n)$. Its distance is at least $1 - 1/\log N$.

[5] By padding with zeros as necessary, we can assume that all the inputs to g are of length that is a multiple of C.

2. Let WH denote the Walsh-Hadamard code from $\{0,1\}^{\log F} = \{0,1\}^{5 \log\log N}$ to $\{0,1\}^{|\mathbb{F}|} = \{0,1\}^{\log^5 N}$.

Our code will be $\mathsf{WH} \circ \mathsf{RM}$. Combining the local decoders for Walsh-Hadamard and Reed-Muller we get the desired result. ■

Combining Theorem 19.21 with Yao's XOR Lemma (Theorem 19.2), we get the following corollary:

Corollary 19.22 *Let $S : \mathbb{N} \to \mathbb{N}$ be a monotone and time-constructible function. Then there is some $\epsilon > 0$ such that if there exists $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ for every n then there exists $\hat{f} \in \mathbf{E}$ with $ACH(f)(n) \geq S(\sqrt{n})^{\epsilon}$.* ♢

Proof: By Theorem 19.21, under this assumption there exists a function $g \in \mathbf{E}$ with $\mathsf{H}^{0.99}_{\mathsf{avg}}(g)(n) \geq S'(n) = S(n)/\operatorname{poly}(n)$, where we can assume $S'(n) \geq \sqrt{S(n)}$ for sufficiently large n (otherwise S is polynomial and the theorem is trivial). Consider the function $g^{\oplus k}$ where $k = c \log S'(n)$ for a sufficiently small constant c. By Yao's XOR Lemma, on inputs of length kn, it cannot be computed with probability better than $1/2 + 2^{-cS'(n)/1000}$ by circuits of size $S'(n)$. Since $S(n) \leq 2^n$, $kn < \sqrt{n}$, and hence we get that $\mathsf{H}_{\mathsf{avg}}(g^{\oplus k}) \geq S^{c/2000}$. ■

19.5 List decoding

While Corollary 19.22 is extremely surprising in the qualitative sense (transforming worst-case hardness to average-case hardness) it is still not fully satisfying quantitatively because it loses quite a bit in the circuit size when moving from a worst-case hard to an average-case hard function. In particular, even if we start with a function f that is hard in the worst-case for $2^{\Omega(n)}$-sized circuits, we only end up with a function $\hat{f}$ that is hard on the average case for $2^{\Omega(\sqrt{n})}$-sized circuits. This can make a difference in some applications, and in particular it falls short of what we will need to fully derandomize **BPP** under worst-case assumptions in Chapter 20.

Our approach to obtain stronger worst-case to average-case reduction will be to bypass the XOR Lemma, and use error correcting codes to get directly from worst-case hardness to a function that is hard to compute with probability slightly better than $1/2$. However, this idea seems to run into a fundamental difficulty: if f is worst-case hard, then it seems hard to argue that the encoding of f, *under any error correcting code* is hard to compute with probability 0.6. The reason is that any binary error-correcting code has to have distance at most $1/2$ but the decoding algorithms work for at most half the distance and hence cannot recover a string f from $E(f)$ if the latter was corrupted in more than a $1/4$ of its locations (i.e., from a string with less than 0.75 agreement with $E(f)$).

This seems like a real obstacle, and indeed was considered as such in many contexts where ECC's were used, until the realization of the importance of the following insight: "If y is obtained by corrupting $E(x)$ in, say, a 0.4 fraction of the coordinates (where E is some ECC with good enough distance) then, while there may be more than one codeword within distance 0.4 to y, *there can not be too many such codewords*." Formally, we have the following theorem:

Theorem 19.23 *(Johnson Bound [Joh62]) If $E : \{0,1\}^n \to \{0,1\}^m$ is an ECC with distance at least $1/2 - \epsilon$, then for every $x \in \{0,1\}^m$, and $\delta \geq \sqrt{\epsilon}$, there exist at most $1/(2\delta^2)$ vectors $y_1, \ldots, y_\ell$ such that $\Delta(x, y_i) \leq 1/2 - \delta$ for every $i \in [\ell]$.* ♢

Proof: Suppose that $x, y_1, \ldots, y_\ell$ satisfy this condition, and define ℓ vectors $z_1, \ldots, z_\ell$ in $\mathbb{R}^m$ as follows: for every $i \in [\ell]$ and $k \in [m]$, set $z_{i,k}$ to equal $+1$ if $y_k = x_k$ and set it to equal -1 otherwise. Under our assumptions, for every $i \in [\ell]$,

$$\sum_{k=1}^{m} z_{i,k} \geq 2\delta m \,, \tag{10}$$

since z_i agrees with x on an $1/2 + \delta$ fraction of its coordinates. Also, for every $i \neq j \in [\ell]$,

$$\langle z_i, z_j \rangle = \sum_{k=1}^{m} z_{i,k} z_{j,k} \leq 2\epsilon m \leq 2\delta^2 m \tag{11}$$

since E is a code of distance at least $1/2 - \epsilon$.

We will show that (10) and (11) together imply that $\ell \leq 1/(2\delta^2)$. Indeed, set $w = \sum_{i=1}^{\ell} z_i$. On one hand, by (11)

$$\langle w, w \rangle = \sum_{i=1}^{\ell} \langle z_i, z_i \rangle + \sum_{i \neq j} \langle z_i, z_j \rangle \leq \ell m + \ell^2 2\delta^2 m \,.$$

On the other hand, by (10), $\sum_k w_k = \sum_{i,j} z_{i,j} \geq 2\delta m \ell$ and hence $\langle w, w \rangle \geq |\sum_k w_k|^2/m \geq 4\delta^2 m \ell^2$, since for every c, the vector $w \in \mathbb{R}^m$ with minimal two-norm satisfying $\sum_k w_k = c$ is the uniform vector $(c/m, c/m, \ldots, c/m)$. Thus $4\delta^2 m\ell^2 \leq \ell m + 2\ell^2\delta^2 m$, implying that $\ell \leq 1/(2\delta^2)$. ■

19.5.1 List decoding the Reed-Solomon code

In many contexts, obtaining a list of candidate messages from a corrupted codeword can be just as good as unique decoding. For example, we may have some outside information on which messages are likely to appear, allowing us to know which of the messages in the list is the correct one. However, to take advantage of this we need an efficient algorithm that computes this list. Such an algorithm was discovered in 1996 by Sudan for the popular and important Reed-Solomon code. It can recover a polynomial size list of candidate codewords given a length m Reed-Solomon codeword that is corrupted in up to a $1 - 2\sqrt{\frac{d}{m}}$ fraction of the coordinates. Note that this tends to 1 as m/d grows, whereas the Berlekamp-Welch algorithm of Section 19.3 (as is the case with any other unique decoding algorithm) cannot handle a fraction of errors that is more than half the distance.

Theorem 19.24 *(List decoding for the Reed-Solomon code* [Sud96]*)*
There is a polynomial-time algorithm that given a set $\{(a_i, b_i)\}_{i=1}^m$ of pairs in $\mathbb{F}^2$, returns the list of all *degree d polynomials G such that the number of i's for which $g(a_i) = b_i$ is more than $2\sqrt{dm}$.*

Proof: We prove Theorem 19.24 via the following algorithm:

Reed-Solomon List Decoding: $t > 2\sqrt{dm}$.

1. Find a nonzero bivariate polynomial $Q(x, y)$ of degree at most $\sqrt{dm}$ in x and at most $\sqrt{m/d}$ in y such that $Q(b_i, a_i) = 0$ for every $i \in [m]$.

   We can express this condition as m linear equations in the $(\sqrt{dm}+1)(\sqrt{m/d}+1) > m$ coefficients of Q. Since these equations are homogeneous (right side equalling zero) and there are more unknowns than equations, this system has a nonzero solution that can be found using gaussian elimination.

2. Factor $Q(x, y)$ using an efficient polynomial factorization algorithm (see [VG99]). For every factor of the form $(P(x) - y)$ check whether $P(x)$ has degree at most d and agrees with $\{(a_i, b_i)\}_{i=1}^m$ in at least t places. If so, output P.

   Indeed, if $G(x)$ agrees with $\{(a_i, b_i)\}_{i=1}^m$ in more than t places then $(G(x) - y)$ is a factor of $Q(x, y)$. To see this note that $Q(G(x), x)$ is a univariate polynomial of degree at most $\sqrt{dm} + d\sqrt{m/d} = 2\sqrt{dm} < t$ which has at least t zeroes and hence it is identically zero. It follows that $G(x) - y$ divides $Q(x, y)$ (see Exercise 19.16).

■

19.6 Local list decoding: getting to $\mathrm{BPP} = \mathrm{P}$.

Analogously to Section 19.4, to actually use list decoding for hardness amplification, we need to provide *local* list decoding algorithms for the codes we use. Fortunately, such algorithms are known for the Walsh-Hadamard code, the Reed-Muller code, and their concatenation. The definition of local list decoding below is somewhat subtle, and deserves a careful reading.

Definition 19.25 *(Local list decoder)* Let $E : \{0,1\}^n \to \{0,1\}^m$ be an ECC and let $\rho = 1-\epsilon$ for $\epsilon > 0$. An algorithm D is called a *local list decoder for E handling ρ errors*, if for every $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ satisfying $\Delta(E(x), y) \leq \rho$, *there exists a number* $i_0 \in [\mathrm{poly}(n/\epsilon)]$ such that for every $j \in [m]$, on inputs i_0, j and with random access to y, D runs for $\mathrm{poly}(\log(m)/\epsilon)$ time and outputs x_j with probability at least $2/3$. ♢

One can think of the number i_0 as the index of x in the list of $\mathrm{poly}(n/\epsilon)$ candidate messages output by L. As is the case for Definition 19.16, Definition 19.25 can be easily generalized to codes with non-binary alphabet.

19.6.1 Local list decoding of the Walsh-Hadamard code.

It turns out we already encountered a local list decoder for the Walsh-Hadamard code: the proof of the Goldreich-Levin Theorem (Theorem 9.12) provided an an algorithm that given access to a "black box" that computes the function $y \mapsto x \odot y$ (for $x, y \in \{0,1\}^n$) with probability $1/2 + \epsilon$, computes a list of values $x_1, \ldots, x_{\mathrm{poly}(n/\epsilon)}$ such that $x_{i_0} = x$ for some i_0. In Chapter 9 we used this algorithm to find the correct value of x from that list by checking it against the value $f(x)$ (where f is a one-way permutation). This is a good example showing how we can use outside information to narrow the list of candidates codewords obtained from a list-decoding algorithm.

19.6.2 Local list decoding of the Reed-Muller code

We now present an algorithm for local list decoding of the Reed-Muller code. Recall that the codeword of this code is the list of evaluations of a d-degree ℓ-variable polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$ and the task of the local decoder is to compute $P(x)$ on a given point $x \in \mathbb{F}^\ell$.

Theorem 19.26 *(Reed-Muller local list decoder* [BF90, Lip91, BFNW93, STV99]*) The Reed-Muller code has a local list decoder handling $1 - 10\sqrt{d/|\mathbb{F}|}$ errors.*

That is, for every $\mathbb{F}, \ell, d$ there is a $\mathrm{poly}(|\mathbb{F}|, d, \ell)$-time algorithm D that given random access to a function $f : \mathbb{F}^\ell \to \mathbb{F}$, an index $i \in \mathbb{F}^{\ell+1}$ and an input $x \in \mathbb{F}^\ell$ satisfies: if f agrees with a degree-d polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$ on $10\sqrt{d/|\mathbb{F}|}$ fraction of the inputs then there exists $i_0 \in \mathbb{F}^{\ell+1}$ such that $\Pr[D^f(i_0, x) = P(x)] \geq 2/3$ for every x. ♢

Proof: To be a valid local list decoder, given the index i_0, the algorithm should output $P(x)$ with high probability for *every* $x \in \mathbb{F}^\ell$. Below we describe a relaxed decoder that is only guaranteed to output the right value for *most* (i.e., a 0.9 fraction) of the x's in $\mathbb{F}^\ell$. One can transform this algorithm to a valid local list decoder by combining it with the Reed-Muller local decoder described in Section 19.4.2. Thus, Theorem 19.26 is proven via the following algorithm:

Reed-Muller Local List Decoder for $\rho \leq 1 - 10\sqrt{d/|\mathbb{F}|}$

Inputs:
- Random access to a function f such that $\Pr_{x \in \mathbb{F}^\ell}[P(x) = f(x)] > 10\sqrt{d/|\mathbb{F}|}$ where $P : \mathbb{F}^\ell \to \mathbb{F}$ is an ℓ-variable d-degree polynomial. We assume that $|\mathbb{F}| > d^4$ and d is sufficiently large (e.g., $d > 1000$ will do). This can always be ensured in our applications.
- An index $i_0 \in [|\mathbb{F}|^{\ell+1}]$ which we interpret as a pair (x_0, y_0) with $x_0 \in \mathbb{F}^\ell$, $y_0 \in \mathbb{F}$,

- A string $x \in \mathbb{F}^\ell$.

Output: $y \in \mathbb{F}$ *(For some pair* (x_0, y_0)*, it should hold that* $P(x) = y$ *with probability at least* 0.9 *over the algorithm's coins and* x *chosen at random from* $\mathbb{F}^\ell$*.)*

Operation:
1. Let L_{x,x_0} be a random degree 3 curve passing through x, x_0. That is, we find a random degree 3 univariate polynomial $q : \mathbb{F} \to \mathbb{F}^\ell$ such that $q(0) = x$ and $q(r) = x_0$ for some random $r \in \mathbb{F}$, and set $L_{x,x_0} = \{q(t) : t \in \mathbb{F}\}$. (See Figure 19.8.)
2. Query f on all the $|\mathbb{F}|$ points of L_{x,x_0} to obtain the set $\mathcal{S}$ of the $|\mathbb{F}|$ pairs $\{(t, f(q(t)) : t \in \mathbb{F})\}$.
3. Run Sudan's Reed-Solomon list decoding algorithm to obtain a list $g_1, \ldots, g_k$ of all degree $3d$ polynomials that have at least $8\sqrt{d|\mathbb{F}|}$ agreement with the pairs in $\mathcal{S}$.
4. If there is a unique i such that $g_i(r) = y_0$ then output $g_i(0)$. Otherwise, halt without outputting anything.

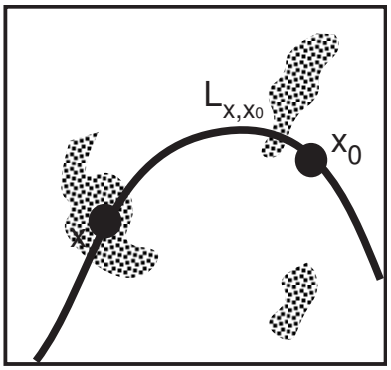


Figure 19.8 Given access to a corrupted version of a polynomial $P : \mathbb{F}^\ell \to \mathbb{F}$ and some index (x_0, y_0), to compute $P(x)$ we pass a random degree-3 curve L_{x,x_0} through x and x_0, and use Reed-Solomon list decoding to recover a list of candidates for the restriction of P to the curve L_{x,x_0}. If only one candidate satisfies that its value on x_0 is y_0, then we use this candidate to compute $P(x)$.

We will show that for every $f : \mathbb{F}^\ell \to \mathbb{F}$ that agrees with an ℓ-variable degree d polynomial on a $10\sqrt{d/|\mathbb{F}|}$ fraction of its input, and every $x \in \mathbb{F}^\ell$, if x_0 is chosen at random from $\mathbb{F}^\ell$ and $y_0 = P(x_0)$, then with probability at least 0.9 (over the choice of x_0 and the algorithm's coins) the above decoder will output $P(x)$. By a standard averaging argument, this implies that there exist a pair (x_0, y_0) such that given this pair, the algorithm outputs $P(x)$ for a 0.9 fraction of the x's in $\mathbb{F}^\ell$.

For every $x \in \mathbb{F}^\ell$, the following fictitious algorithm can be easily seen to have an identical output to the output of our decoder on the inputs x, a random $x_0 \in_R \mathbb{F}^\ell$ and $y_0 = P(x_0)$:

1. Choose a random degree 3 curve L that passes through x. That is, $L = \{q(t) : t \in \mathbb{F}\}$ where $q : \mathbb{F} \to \mathbb{F}^\ell$ is a random degree 3 polynomial satisfying $q(0) = x$.
2. Obtain the list $g_1, \ldots, g_m$ of all univariate polynomials over $\mathbb{F}$ such that for every i, there are at least $6\sqrt{d|\mathbb{F}|}$ values of t such that $g_i(t) = f(q(t))$.
3. Choose a random $r \in \mathbb{F}$. Assume that you are given the value $y_0 = P(q(r))$.
4. If there exists a unique i such that $g_i(r) = y_0$ then output $g_i(0)$. Otherwise, halt without an input.

Yet, this fictitious algorithm will output $P(x)$ with probability at least 0.9. Indeed, since all the points other than x on a random degree 3 curve passing through x are pairwise

independent, Chebyshev's inequality implies that with probability at least 0.99, the function f will agree with the polynomial P on at least $8\sqrt{d|\mathbb{F}|}$ points on this curve. Thus the list $g_1, \ldots, g_m$ we obtain in Step 2 contains the polynomial $g : \mathbb{F} \to \mathbb{F}$ defined as $g(t) = P(q(t))$. We leave it as Exercise 19.15 to show that there can not be more than $\sqrt{|F|}/4d$ polynomials in this list. Since two $3d$-degree polynomials can agree on at most $3d+1$ points, with probability at least $1 - \frac{(3d+1)\sqrt{|F|}/4d}{|\mathbb{F}|} > 0.99$, if we choose a random $r \in \mathbb{F}$, then $g(r) \neq g_i(r)$ for every $g_i \neq g$ in this list. Thus, with this probability, we will identify the polynomial g and output the value $g(0) = P(x)$. ■

19.6.3 Local list decoding of concatenated codes.

If $E_1 : \{0,1\}^n \to \Sigma^m$ and $E_2 : \Sigma \to \{0,1\}^k$ are two codes that are locally list decodable then so is the concatenated code $E_2 \circ E_1 : \{0,1\}^n \to \{0,1\}^{mk}$. As in Section 19.4.3, the idea is to simply run the local decoder for E_1 while answering its queries using the decoder of E_2. More concretely, assume that the decoder for E_1 takes an index in the set I_1 and can handle $1 - \epsilon_1$ errors, and that E_2 takes an index in I_2 and can handle $1/2 - \epsilon_2$ errors. Our decoder for $E_2 \circ E_1$ will take a pair of indices $i_1 \in I_1$ and $i_2 \in I_2$ and run the decoder for E_1 with the index i_1, and whenever this decoder makes a query answer it using the decoder E_2 with the index i_2. (See Section 19.4.3.) We claim that this decoder can handle $1/2 - \epsilon_1\epsilon_2|I_2|$ number of errors. Indeed, if y agrees with some codeword $E_2 \circ E_1(x)$ on an $\epsilon_1\epsilon_2|I_2|$ fraction of the coordinates then there are $\epsilon_1|I_2|$ blocks on which it has at least $1/2 + \epsilon_2$ agreement with the blocks this codeword. Thus, by an averaging argument, there exists an index i_2 such that given i_2, the output of the E_2 decoder agrees with $E_1(x)$ on ϵ_1 symbols, implying that there exists an index i_1 such that given (i_1, i_2) and every coordinate j, the combined decoder will output x_j with high probability.

19.6.4 Putting it all together.

As promised, we can use local list decoding to transform a function that is merely worst-case hard into a function that cannot be computed with probability significantly better than $1/2$:

Theorem 19.27 *(Worst-case hardness to strong hardness)*
Let $f \in \mathbf{E}$ be such that $\mathsf{H}_{wrs}(f)(n) \geq S(n)$ for some time-constructible non-decreasing $S : \mathbb{N} \to \mathbb{N}$. Then there exists a function $g \in \mathbf{E}$ and a constant $c > 0$ such that $\mathsf{H}_{avg}(g)(n) \geq S(n/c)^{1/c}$ for every sufficiently large n.

PROOF SKETCH: As in Section 19.4.4, for every n, we treat the restriction of f to $\{0,1\}^n$ as a string $f' \in \{0,1\}^N$ where $N = 2^n$ and encode it using the concatenation of a Reed-Muller code with the Walsh-Hadamard code. For the Reed-Muller code we use the following parameters:

- The field $\mathbb{F}$ is of size $S(n)^{1/100}$. (We may assume without loss of generality that $S(n) > n^{1000}$ as otherwise the theorem is trivial.)
- The degree d is of size $\log^2 N$.
- The number of variables ℓ is $2 \log N / \log S(n)$.

The function g is obtained by applying this encoding to f. Given a circuit of size $S(n)^{1/100}$ that computes g with probability better than $1/2 + 1/S(n)^{1/50}$, we will be able to transform it, in $S(n)^{O(1)}$ time, to a circuit computing f perfectly. We hardwire the index i_0 to this circuit as part of its description. ■

What have we learned?

- Yao's XOR Lemma allows us to amplify hardness by transforming a Boolean function with only mild hardness (cannot be computed with say 0.99 success) into a Boolean function with strong hardness (cannot be computed with 0.51 success).
- An *error correcting code* is a function that maps every two strings into a pair of strings that differ on many of their coordinates. An error correcting code with a *local decoding* algorithm can be used to transform a function hard in the worst-case into a function that is mildly hard on the average case.
- A code over the binary alphabet can have distance at most $1/2$. A code with distance δ can be uniquely decoded up to $\delta/2$ errors. *List decoding* allows to a decoder to handle almost a δ fraction of errors, at the expense of returning not a single message but a short list of candidate messages.
- We can transform a function that is merely hard in the worst case to a function that is strongly hard in the average case using the notion of *local list decoding* of error correcting codes.

Chapter notes and history

Yao's XOR Lemma was first stated and proven by Yao in oral presentations of his paper [Yao82a]. Since then several proofs have been published with the first one by Levin in [Lev87] (see the survey [GNW95]). Russell Impagliazzo's hardcore lemma was proven in [Imp95b]; the proof of Section 19.1.2 is due to Noam Nisan.

The study of error correcting codes is an extremely beautiful and useful field, and we have barely scratched its surface here. This field was initiated by two roughly concurrent seminal papers of Shannon [Sha48] and Hamming [Ham50]. The lecture notes of Madhu Sudan (available from his home page) provide a good starting point for theoretical computer scientists; see also the survey [Sud01].

Reed-Solomon codes were invented in 1960 by Irving Reed and Gustave Solomon [RS60]. The first efficient decoding algorithm for Reed-Solomon codes was by Peterson [Pet60]. (Interestingly, this algorithm is one of the first non-trivial polynomial-time algorithms invented, preceding even the formal definition of the class **P**.) The algorithm presented in Section 19.3 is a simplification due to Gemmell and Sudan [GS92] of the Berlekamp-Welch decoder [BW86].

Reed-Muller codes were invented by Muller [Mul54] with the first decoder given by Reed [Ree54]. The first Reed-Muller local decoders were given by Beaver and Feigenbaum [BF90] and Lipton [Lip91], who observed this implies a worst-case to average-case connection for the *Permanent* (see also Section 8.6.2). Babai, Fortnow, and Lund [BFL90] observed that by taking multilinear extensions, such connections also hold for **PSPACE** and **EXP**, and Babai et al [BFNW93] showed that this allows for derandomization from worst-case assumptions. The Reed-Muller local decoding algorithm of Section 19.4.2 is due to Gemmell et al [GLR+91].

The first list-decoding algorithm for Reed-Solomon codes was given by Sudan [Sud96] and was subsequently improved by Guruswami and Sudan [GS98]. Recently, Parvaresh and Vardy [PV05] showed a list-decoding algorithm handling even more errors for a variant of the Reed-Solomon code, a result that was further improved by Guruswami and Rudra [GR06], achieving an optimal tradeoff between rate and list decoding radius for large alphabets.

The quantitatively strong hardness amplification (Theorem 19.27) was first shown by Impagliazzo and Wigderson [IW97] that gave a *derandomized* version of Yao's XOR Lemma. Our presentation follows the alternative proof by Sudan, Trevisan and Vadhan [STV99] who were the first to make an explicit connection between error correcting codes and hardness amplification, and also the first to explicitly define local list decoding and use it for hardness amplification. The first local list decoding algorithm for the Walsh-Hadamard code was given by Goldreich and Levin [GL89] (although the result is not explicitly described in these terms there). The Reed-Muller local list decoding algorithm of Section 19.6 is a variant of the algorithm of [STV99].

The question raised in Problem 19.8 is treated in O'Donnell [O'D04], where a hardness amplification lemma is given for **NP**. For a sharper result, see Healy, Vadhan, and Viola [HVV04].

Exercises

19.1 Let $X_1, \ldots, X_n$ be independent random variables such that X_i is equal to 1 with probability $1-\delta$ and equal to 0 with probability δ. Let $X = \sum_{i=1}^{k} X_i \pmod 2$. Prove that $\Pr[X = 1] = 1/2 + (1-2\delta)^k$. H463

19.2 Prove that if there exists a δ-density distribution H such that $\Pr_{x \in_R H}[C(x) = f(x)] \leq 1/2 + e$ for every circuit C of size at most S with $S \leq \sqrt{\epsilon^2 \delta 2^n / 100}$, then there exists a subset $I \subseteq \{0,1\}^n$ of size at least $\frac{\delta}{2}2^n$ such that

$$\Pr_{x \in_R I}[C(x) = f(x)] \leq 1/2 + 2\epsilon$$

for every circuit C of size at most S. H464

19.3 Let H be an δ-density distribution over $\{0,1\}^n$ (i.e., $\Pr[H = x] \leq 1/(\delta 2^n)$ for every $x \in \{0,1\}^n$).

(a) Let G be the distribution defined by $\Pr[G = x] = (2^{-n} - \delta \Pr[H = x])/(1-\delta)$ for every $x \in \{0,1\}^n$. Prove that G is indeed a distribution (i.e., all probabilities are non-negative and sum up to 1).

(b) Let U be the following distribution: with probability δ pick an element from H and with probability $1-\delta$ pick an element from G. Prove that U is the uniform distribution.

H464

19.4 Complete the proof of Impagliazzo's Hardcore Lemma (Lemma 19.3) for general k.

19.5 Prove the hyperplane separation theorem in the following form: If $C, D \subseteq \mathbb{R}^m$ are two disjoint convex set with C closed and D compact (i.e., closed and bounded) then there exists a nonzero vector $\mathbf{z} \in \mathbb{R}^m$ and a number $a \in \mathbb{R}$ such that

$$\mathbf{x} \in C \Rightarrow \langle \mathbf{x}, \mathbf{z} \rangle \geq a$$
$$\mathbf{y} \in D \Rightarrow \langle \mathbf{y}, \mathbf{z} \rangle \leq a$$

H464

19.6 Prove the Min-Max Theorem (see Note 19.4) using the hyperplane separation theorem as stated in Exercise 19.5. H464

19.7 ([CG85]) We say that a distribution D over $\{0,1\}^n$ is K-flat if D is the uniform distribution over a subset of $\{0,1\}^n$ with size at least K. Prove that for every k, every 2^{-k}-density distribution H is a convex combination of 2^{n-k}-flat distributions. That is, there are N 2^{n-k}-flat distributions $D_1, \ldots, D_N$ and non-negative numbers $\alpha_1, \ldots, \alpha_N$ such that $\sum_i \alpha_i = 1$ and H is equivalent to the distribution obtained by picking i with probability α_i and then picking a random element from D_i. H464

19.8 Suppose we know that **NP** contains a function that is weakly hard for all polynomial-size circuits. Can we use the XOR Lemma to infer the existence of a strongly hard function in **NP**? Why or why not?

19.9 For every $\delta < 1/2$ and sufficiently large n, prove that there exists a function $E : \{0,1\}^n \to \{0,1\}^{n/(1-H(\delta))}$ that is an error correcting code with distance δ, where $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$. H464

19.10 Show that for every $E : \{0,1\}^n \to \{0,1\}^m$ that is an error correcting code of distance $1/2$, $2^n < 10m$. Show if E is an error correcting code of distance $\delta > 1/2$, then $2^n < 10/(\delta - 1/2)$. H464

19.11 Let $E : \{0,1\}^n \to \{0,1\}^m$ be an ECC such that there exist two distinct strings $x^1, x^2 \in \{0,1\}^m$ with $\Delta(E(x^1), E(x^2)) \leq \delta$. Prove that there's no decoder for E handling $\delta/2$ or more errors. That is, show that there is no function $D : \{0,1\}^m \to \{0,1\}^n$ such that for every $x \in \{0,1\}^m$ and y with $\Delta(y, E(x)) \leq \delta/2$, $D(y) = x$.

19.12 Let $E : \{0,1\}^n \to \{0,1\}^m$ be a δ-distance ECC. Transform E to a code $E' : \{0,1,2,3\}^{n/2} \to \{0,1,2,3\}^{m/2}$ in the obvious way. Show that E' has distance δ. Show that the opposite direction is not true: show an example of a δ-distance ECC $E' : \{0,1,2,3\}^{n/2} \to \{0,1,2,3\}^{m/2}$ such that the corresponding binary code has distance 2δ.

19.13 Prove Theorem 19.15 as stated. That is show how to recover a d-degree polynomial G from a sequence of pairs $(a_1, b_1), \ldots, (a_m, b_m)$ agreeing with G in t places whenever $t \geq \frac{m}{2} + \frac{d}{2} + 1$.

19.14 Prove Theorem 19.17. H464

19.15 Let $f : \mathbb{F} \to \mathbb{F}$ be any function. Suppose integer $d \geq 0$ and number ϵ satisfy $\epsilon > 2\sqrt{\frac{d}{|\mathbb{F}|}}$. Prove that there are at most $2/\epsilon$ degree d polynomials that agree with f on at least an ϵ fraction of its coordinates. H464

19.16 Prove that if $Q(x, y)$ is a bivariate polynomial over some field $\mathbb{F}$ and $P(x)$ is a univariate polynomial over $\mathbb{F}$ such that $Q(P(x), x)$ is the zero polynomial then $Q(x, y) = (y - P(x))A(x, y)$ for some polynomial $A(x, y)$. H464

19.17 *(Linear codes)* We say that an ECC $E : \{0,1\}^n \to \{0,1\}^m$ is *linear* if for every $x, x' \in \{0,1\}^n$, $E(x + x') = E(x) + E(x')$ where $+$ denotes componentwise addition modulo 2. A linear ECC E can be described by an $m \times n$ matrix A such that (thinking of x as a column vector) $E(x) = Ax$ for every $x \in \{0,1\}^n$.

(a) Prove that the distance of a linear ECC E is equal to the minimum over all nonzero $x \in \{0,1\}^n$ of the fraction of 1's in $E(x)$.

(b) Prove that for every $\delta > 0$, there exists a linear ECC $E : \{0,1\}^n \to \{0,1\}^{1.1n/(1-H(\delta))}$ with distance δ, where $H(\delta) = \delta \log(1/\delta) + (1-\delta)\log(1/(1-\delta))$. H464

(c) Prove that for some $\delta > 0$ there is an ECC $E : \{0,1\}^n \to \{0,1\}^{\mathrm{poly}(n)}$ of distance δ with polynomial-time encoding and decoding mechanisms. (You need to know about the field $\mathrm{GF}(2^k)$ to solve this, see Appendix A.) H464

(d) We say that a linear code $E : \{0,1\}^n \to \{0,1\}^m$ is ϵ-*biased* if for every non-zero $x \in \{0,1\}^n$, the fraction of 1's in $E(x)$ is between $1/2 - \epsilon$ and $1/2 + \epsilon$. Prove that for every $\epsilon > 0$, there exists an ϵ-biased linear code $E : \{0,1\}^n \to \{0,1\}^{\mathrm{poly}(n/\epsilon)}$ with a polynomial-time encoding algorithm.

19.18 Recall that for every m there is field $\mathbb{F} = \mathrm{GF}(2^m)$ of 2^m elements such that we can represent each element of $\mathbb{F}$ as a vector in $\mathrm{GF}(2)^m$ with addition in $\mathbb{F}$ corresponding to bitwise XOR (see Appendix A). Thus for every $a \in \mathbb{F}$, the operation $x \mapsto a \times x$ (where $\times$ denotes multiplication in $\mathbb{F}$) is a linear operation in $\mathrm{GF}(2)^m$. Moreover, this operation is efficiently computable given the description of a.

(a) Prove that for every nonzero $x \in F$, if we choose a uniformly in $\mathbb{F}$ then $a \times x$ is distributed uniformly over $\mathbb{F}$.

(b) Prove that for every nonzero $x \in F$, the probability over a random $a \in_R \mathbb{F}$ that $a \times x$ has at most $m/10$ ones in its representation as an m-bit vector is less than $2^{-m/10}$. Conclude that there exists $a \in \mathbb{F}$ such that the function that maps $x \in \{0,1\}^{m/10}$ to $a \times (x \circ 0^{0.9m})$ (where $\circ$ denotes concatenation) is an error correcting code with distance at least 0.1.

(c) Show that there exists constants $c, \delta > 0$ such that for every n there is an explicit error correcting code $E : \{0,1\}^n \to \{0,1\}^{cn}$ of distance at least δ. H464

Chapter 20

Derandomization

"God does not play dice with the universe"
Albert Einstein

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."
John von Neumann, quoted by Knuth 1981

Randomization is an exciting and powerful paradigm in computer science and, as we saw in Chapter 7, often provides the simplest or most efficient algorithms for many computational problems. In fact, in some areas of Computer Science, such as distributed algorithms and cryptography, randomization is *proven* to be necessary to achieve certain tasks or achieve them efficiently. Thus it's natural to conjecture (as many scientists initially did) that at least for some problems, randomization is *inherently* necessary: one cannot replace the probabilistic algorithm with a deterministic one without a significant loss of efficiency. One concrete version of this conjecture would be that $\mathbf{BPP} \not\subseteq \mathbf{P}$ (see Chapter 7 for definition of **BPP**). Surprisingly, recent research has provided more and more evidence that this is likely *not* to hold. As we will see in this chapter, under very reasonable complexity assumptions, there is in fact a way to *derandomize* (i.e., transform into a deterministic algorithm) *every* probabilistic algorithm of the **BPP** type with only a polynomial loss of efficiency. Thus today most researchers believe that $\mathbf{BPP} = \mathbf{P}$. We note that this need not imply that randomness is useless in every setting —we already saw in Chapter 8 its crucial role in the definition of interactive proofs.

In Section 20.1 we define *pseudorandom generators*, which will serve as our main tool for derandomizing probabilistic algorithms. Their definition is a relaxation of the definition of *secure* pseudorandom generators in Chapter 9. This relaxation will allow us to construct such generators with better parameters and under weaker assumptions than what is possible for secure pseudorandom generators. In Section 20.2 we provide a construction of such pseudorandom generators under the assumptions that there exist explicit functions with high *average-case* circuit complexity. In Chapter 19 we show how to provide a construction that merely requires high *worst-case* circuit complexity.

While the circuit lower bounds we assume are widely believed to be true, they also seem to be very difficult to prove. This raises the question of whether assuming or proving such lower bounds is *necessary* to obtain derandomization. In Section 20.3 we show that it's possible to obtain at least a partial derandomization result based only on the assumption that $\mathbf{BPP} \neq \mathbf{EXP}$. Alas, as we show in Section 20.4, full derandomization of **BPP** will require proving circuit lower bounds.

Even though we still cannot prove sufficiently strong circuit lower bounds, just as in cryptography, we can use *conjectured* hard problems for derandomization instead of *provable* hard problems, and to a certain extent end up with a win-win situation: if the conjectured hard problem is truly hard then the derandomization will be successful; and if the derandomization fails then it will lead us to an algorithm for the conjectured hard problem.

Example 20.1 *(Polynomial identity testing)*
We explain the notion of derandomization with an example. One algorithm that we would like to derandomize is the one described in Section 7.2.3 for testing if a given polynomial (represented in the form of an arithmetic circuit) is the identically zero polynomial. If P is an n-variable nonzero polynomial of total degree d over a large enough finite field $\mathbb{F}$ ($|\mathbb{F}| > 10d$ will do) then most of the vectors $\mathbf{u} \in \mathbb{F}^n$ will satisfy $P(\mathbf{u}) \neq 0$ (see Lemma 7.5). Therefore, checking whether $P \equiv 0$ can be done by simply choosing a random $\mathbf{u} \in_R \mathbb{F}^n$ and applying p on $\mathbf{u}$. In fact, it is easy to show that there exists a set of m^2-vectors $\mathbf{u}^1, \ldots, \mathbf{u}^{m^2}$ such that for *every* such nonzero polynomial P that can be computed by a size m arithmetic circuit, there exists an $i \in [m^2]$ for which $P(\mathbf{u}^i) \neq 0$.
This suggests a natural approach for a deterministic algorithm: show a deterministic algorithm that for every $m \in \mathbb{N}$, runs in poly(m) time and outputs a set $\mathbf{u}^1, \ldots, \mathbf{u}^{m^2}$ of vectors satisfying the above property. This shouldn't be too difficult— after all the vast majority of the sets of vectors have this property, so how hard can it be to find a single one? Surprisingly this turns out to be quite hard: without using complexity assumptions, we do not know how to obtain such a set, and in Section 20.4 we will see that in fact obtaining such a set (or even any other deterministic algorithm for this problem) will imply some nontrivial circuit lower bounds.

20.1 Pseudorandom Generators and Derandomization

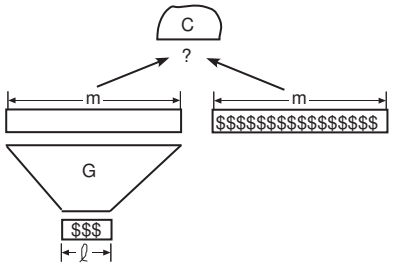
The main tool we will use for derandomization is a pseudorandom generator. This is a twist on the definition of a cryptographically *secure* pseudorandom generator we saw in Chapter 9, with the main difference that here we will allow the generator to run in *exponential* time (and in particular allow the generator more time than the distinguisher). Another difference is that we consider *non-uniform* distinguishers— in other words, circuits— rather than Turing machines, as was done in Chapter 9. (This second difference is not an essential one. As mentioned in the notes at the end of Chapter 9, we could have used circuits there as well.)

Definition 20.2 *(Pseudorandom generators)*
A distribution R over $\{0,1\}^m$ is (S,ϵ)-*pseudorandom* (for $S \in \mathbb{N}, \epsilon > 0$) if for every circuit C of size at most S,

$$\left|\Pr[C(R)=1] - \Pr[C(U_m)=1]\right| < \epsilon\,,$$

where U_m denotes the uniform distribution over $\{0,1\}^m$.

Let $S : \mathbb{N} \to \mathbb{N}$ be some function. A 2^n-time computable function $G : \{0,1\}^* \to \{0,1\}^*$ is an $S(\ell)$-*pseudorandom generator* if $|G(z)| = S(|z|)$ for every $z \in \{0,1\}^*$ and for every $\ell \in \mathbb{N}$ the distribution $G(U_\ell)$ is $(S(\ell)^3, 1/10)$-pseudorandom.



The choices of the constants 3 and 1/10 in the definition of an $S(\ell)$-pseudorandom generator are arbitrary and made for convenience. To avoid annoying cases, we will restrict our attention to $S(\ell)$-pseudorandom generators for functions $S : \mathbb{N} \to \mathbb{N}$ that are *time-constructible* and *non-decreasing* (i.e., $S(\ell') \geq S(\ell)$ for $\ell' \geq \ell$).

20.1.1 Derandomization using pseudorandom generators

The relation between pseudorandom generators and simulating probabilistic algorithms is rather straightforward:

Lemma 20.3 *Suppose that there exists an $S(\ell)$-pseudorandom generator for a time-constructible non-decreasing $S : \mathbb{N} \to \mathbb{N}$. Then for every polynomial-time computable function $\ell : \mathbb{N} \to \mathbb{N}$, $\mathbf{BPTIME}(S(\ell(n))) \subseteq \mathbf{DTIME}(2^{c\ell(n)})$ for some constant c.* ♢

Before proving Lemma 20.3 it is instructive to see what derandomization results it implies for various values of S. This is observed in the following simple corollary, left as Exercise 20.1:

Corollary 20.4

1. *If there exists a $2^{\epsilon\ell}$-pseudorandom generator for some constant $\epsilon > 0$ then $\mathbf{BPP} = \mathbf{P}$.*
2. *If there exists a 2^{ℓ^ϵ}-pseudorandom generator for some constant $\epsilon > 0$ then $\mathbf{BPP} \subseteq \mathbf{QuasiP} = \mathbf{DTIME}(2^{\mathrm{polylog}(n)})$.*
3. *If for every $c > 1$ there exists an ℓ^c-pseudorandom generator then $\mathbf{BPP} \subseteq \mathbf{SUBEXP} = \cap_{\epsilon>0}\mathbf{DTIME}(2^{n^\epsilon})$.* ♢

Proof of Lemma 20.3: A language L is in $\mathbf{BPTIME}(S(\ell(n)))$ if there is an algorithm A that on input $x \in \{0,1\}^n$ runs in time $cS(\ell(n))$ for some constant c, and satisfies

$$\Pr_{r \in_R \{0,1\}^m}[A(x,r) = L(x)] \geq 2/3\,, \tag{1}$$

where $m \leq S(\ell(n))$ and we define $L(x) = 1$ if $x \in L$ and $L(x) = 0$ otherwise.

The main idea is that if we replace the truly random string r with the string $G(z)$ produced by picking a random $z \in \{0,1\}^{\ell(n)}$, then an algorithm such as A that runs in only $S(\ell)$ time cannot detect this switch most of the time, and so the probability $2/3$ in the previous expression does not drop below $2/3 - 0.1 > 0.5$. Thus to derandomize A, we do not need to enumerate over all $r \in \{0,1\}^m$: it suffices to enumerate over all the strings $G(z)$ for $z \in \{0,1\}^{\ell(n)}$ and check whether or not the majority of these make A accept. This derandomized algorithm runs in $2^{O(\ell(n))}$ time instead of the trivial 2^m time.

Now we make this formal. On input $x \in \{0,1\}^n$, our deterministic algorithm B will go over all $z \in \{0,1\}^{\ell(n)}$, compute $A(x, G(z))$ and output the majority answer.[1] We claim that for n sufficiently large, the fraction of z's such that $A(x, G(z)) = L(x)$ is at least $2/3 - 0.1$. This suffices to prove that $L \in \mathbf{DTIME}(2^{c\ell(n)})$ as we can "hardwire" into the algorithm the correct answer for finitely many inputs.

Suppose this is false and there exists an infinite sequence of x's for which $\Pr[A(x, G(z)) = L(x)] < 2/3 - 0.1$. Then there exists a distinguisher for the pseudorandom generator: just use the Cook-Levin transformation (e.g., as in the proof of Theorem 6.6) to construct a circuit computing the function $r \mapsto A(x, r)$, where x is hardwired into the circuit. (This "hardwiring" is the place in the proof where we use non-uniformity.) This circuit has size $O(S(\ell(n)))^2$ which is smaller than $S(\ell(n))^3$ for sufficiently large n. ■

The proof of Lemma 20.3 shows why it is OK to allow the pseudorandom generator in Definition 20.2 to run in time exponential in its seed length. The reason is that the derandomized algorithm enumerates over all possible seeds of length ℓ, and thus would take exponential (in ℓ) time even if the generator itself were to run in less than $\exp(\ell)$ time.

[1] If $m < S(\ell(n))$ then $A(x, G(z))$ denotes the output of A on input x using the first m bits of $G(z)$ for its random choices.

Notice also that allowing the generator $\exp(\ell)$ time means that it has to "fool" distinguishers that run for *less* time than it does. By contrast, the definition of cryptographically *secure pseudorandom generators* (Definition 9.8 in Chapter 9) required the generator to run in some fixed polynomial time, and yet fool arbitrary polynomial-time distinguishers. This difference in these definitions stems from the intended usage. In the cryptographic setting the generator is used by honest users and the distinguisher is the adversary attacking the system — and it is reasonable to assume the attacker can invest more computational resources than those needed for normal/honest use of the system. In derandomization, the generator is used by the derandomized algorithm and the "distinguisher" is the probabilistic algorithm that is being derandomized. In this case it is reasonable to allow the derandomized algorithm more running time than the original probabilistic algorithm. Of course, allowing the generator to run in exponential time potentially makes it easier to prove their existence compared with secure pseudorandom generators, and this indeed appears to be the case. If we relaxed the definition even further and made no efficiency requirements then showing the existence of such generators becomes almost trivial (see Exercise 20.2) but they no longer seem useful for derandomization.

We will construct pseudorandom generators based on complexity assumptions, using quantitatively stronger assumptions to obtain quantitatively stronger pseudorandom generators (i.e., $S(\ell)$-pseudorandom generators for larger functions S). The strongest (though still reasonable) assumption will yield a $2^{\Omega(\ell)}$-pseudorandom generator, thus implying that $\mathbf{BPP} = \mathbf{P}$.

20.1.2 Hardness and Derandomization

We construct pseudorandom generators under the assumptions that certain explicit functions are hard. In this chapter we use assumptions about *average-case* hardness, but using the results of Chapter 19 we will be able to also construct pseudorandom generators assuming only *worst-case* hardness. Both worst-case and average-case hardness refer to the size of the minimum Boolean circuit computing the function. Recall that we define the *average-case hardness* of a function $f : \{0,1\}^n \to \{0,1\}$, denoted by $\mathsf{H}_{\mathsf{avg}}(f)$, to be the largest number S such that $\Pr_{x \in_R \{0,1\}^n}[C(x) = f(x)] < 1/2 + 1/S$ for every Boolean circuit C on n inputs with size at most S (see Definition 19.1). For $f : \{0,1\}^* \to \{0,1\}$, we let $\mathsf{H}_{\mathsf{avg}}(f)(n)$ denote the average-case hardness of the restriction of f to $\{0,1\}^n$.

Example 20.5

Here are some examples of functions and their conjectured or proven hardness:

1. If $f : \{0,1\}^* \to \{0,1\}$ is a random function (i.e., for every $x \in \{0,1\}^*$ we choose $f(x)$ using an independent unbiased coin) then with high probability, both the worst-case and average-case hardness of f are exponential (see Exercise 20.3). In particular, with probability tending to 1 as n grows, both $\mathsf{H}_{\mathsf{wrs}}(f)(n)$ and $\mathsf{H}_{\mathsf{avg}}(f)(n)$ exceed $2^{0.99n}$.
2. If $f \in \mathbf{BPP}$ then, since $\mathbf{BPP} \subseteq \mathbf{P}_{/\mathbf{poly}}$, both $\mathsf{H}_{\mathsf{wrs}}(f)$ and $\mathsf{H}_{\mathsf{avg}}(f)$ are bounded by some polynomial.
3. It seems reasonable to believe that 3SAT has exponential worst-case hardness; that is, $\mathsf{H}_{\mathsf{wrs}}(\mathsf{3SAT}) \geq 2^{\Omega(n)}$. A weaker assumption is that $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$, in which case $\mathsf{H}_{\mathsf{wrs}}(\mathsf{3SAT})$ is not bounded by any polynomial. The average case complexity of 3SAT for uniformly chosen inputs is unclear, and in any case is dependent upon the way we choose to represent formulae as strings.
4. Under widely believed cryptographic assumptions, $\mathbf{NP}$ contains functions that are hard on the average. If g is a one-way permutation (as defined in Chapter 9) that cannot be inverted with polynomial probability by polynomial-sized circuits, then by Theorem 9.12, the function f that maps the pair $x, r \in \{0,1\}^n$ to $g^{-1}(x) \odot r$ (where $x \odot r = \sum_{i=1}^{n} x_i r_i \pmod 2$) has super-polynomial *average-case* hardness: $\mathsf{H}_{\mathsf{avg}}(f) \geq n^{\omega(1)}$.

The main theorem of this section uses hard-on-the average functions to construct pseudorandom generators:

Theorem 20.6 *(PRGs from average-case hardness)*
Let $S : \mathbb{N} \to \mathbb{N}$ be time-constructible and non-decreasing. If there exists $f \in \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{avg}}(f)(n) \geq S(n)$ for every n then exists an $S(\delta\ell)^{\delta}$-pseudorandom generator for some constant $\delta > 0$.

Combining this result with Theorem 19.27 of Chapter 19 we obtain the following theorem that gives even stronger evidence (given the plethora of plausible hard functions mentioned above) for the conjecture that derandomizing probabilistic algorithms is possible:

Theorem 20.7 *(Derandomization under worst-case assumptions)*
Let $S : \mathbb{N} \to \mathbb{N}$ be time-constructible and non-decreasing. If there exists $f \in \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$ for every n then exists an $S(\delta\ell)^{\delta}$-pseudorandom generator for some constant $\delta > 0$. In particular, the following corollaries hold:

1. *If there exists $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(n)})$ and $\epsilon > 0$ such that $\mathsf{H}_{\mathsf{wrs}}(f) \geq 2^{\epsilon n}$ then $\mathbf{BPP} = \mathbf{P}$.*
2. *If there exists $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(n)})$ and $\epsilon > 0$ such that $\mathsf{H}_{\mathsf{wrs}}(f) \geq 2^{n^{\epsilon}}$ then $\mathbf{BPP} \subseteq \mathbf{QuasiP}$.*
3. *If there exists $f \in \mathbf{E} = \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{wrs}}(f) \geq n^{\omega(1)}$ then $\mathbf{BPP} \subseteq \mathbf{SUBEXP}$.*

We can replace $\mathbf{E}$ with $\mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}(n)})$ in Corollaries 2 and 3 above. Indeed, for every $f \in \mathbf{DTIME}(2^{n^c})$, let g be the function that on input $x \in \{0,1\}^*$ outputs f applied to the first $|x|^{1/c}$ bits of x. Then, g is in $\mathbf{DTIME}(2^n)$ and satisfies $\mathsf{H}_{\mathsf{avg}}(g)(n) \geq \mathsf{H}_{\mathsf{avg}}(f)(n^{1/c})$. Therefore, if there exists $f \in \mathbf{EXP}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq 2^{n^{\delta}}$ then there there exists a constant $\delta' > 0$ and a function $g \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{avg}}(g) \geq 2^{n^{\delta'}}$, and so we can replace $\mathbf{E}$ with $\mathbf{EXP}$ in Corollary 2. A similar observation holds for Corollary 3. Note that $\mathbf{EXP}$ contains many classes we believe to have hard problems, such as $\mathbf{NP}$, $\mathbf{PSPACE}$, $\oplus\mathbf{P}$ and more.

Remark 20.8
Nisan and Wigderson [NW88] were the first to show a pseudorandom generator from average-case hardness, but they did not prove Theorem 20.6 as it is stated above. Rather, Theorem 20.6 was proven by Umans [Uma03] following a long sequence of works including [BFNW93, IW97, ISW99, STV99, SU01]. Nisan and Wigderson only proved that under the same assumptions there exists an $S'(\ell)$-pseudorandom generator, where $S'(\ell) = S(n)^{\delta}$ for some constant $\delta > 0$ and n satisfying $n \geq \delta\sqrt{\ell \log S(n)}$. Note that this bound is still sufficient to derive all three corollaries above. It is this weaker version we prove in this book.

20.2 Proof of Theorem 20.6: Nisan-Wigderson Construction

How can we use a hard function to construct a pseudorandom generator? As a warmup we start with two "toy examples". We first show how to use a hard function to construct a pseudorandom generator whose output is only a single bit longer than its input. Then we show how to obtain such a generator whose output is two bits longer than the input. Of course, neither of these suffices to prove Theorem 20.6 but they do give insight into the connection between hardness and randomness.

20.2.1 Two toy examples

Extending the input by one bit using Yao's Theorem.

The following Lemma uses a hard function to construct a "toy" generator that extends its input by a single bit:

Lemma 20.9 *(One-bit generator) Suppose that there exist $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq n^4$. Then, there exists an $S(\ell)$-pseudorandom generator G for $S(\ell) = \ell + 1$.* ♢

PROOF: The generator G is very simple: for every $z \in \{0,1\}^\ell$, we set

$$G(z) = z \circ f(z)$$

(where $\circ$ denotes concatenation). G clearly satisfies the output length and efficiency requirements of an $(\ell+1)$-pseudorandom generator. To prove that its output is $((\ell+1)^3, 1/10)$-pseudorandom we use Yao's Theorem from Chapter 9 showing that pseudorandomness is implied by unpredictability:[2]

Theorem 20.10 *(Theorem 9.11, restated) Let Y be a distribution over $\{0,1\}^m$. Suppose that there exist $S > 10n$ and $\epsilon > 0$ such that for every circuit C of size at most $2S$ and $i \in [m]$,*

$$\Pr_{r \in_R Y}[C(r_1, \ldots, r_{i-1}) = r_i] \leq \frac{1}{2} + \frac{\epsilon}{m}$$

Then Y is (S, ϵ)-pseudorandom. ♢

By Theorem 20.10, to prove Lemma 20.9 it suffices to show that there does not exist a circuit C of size $2(\ell+1)^3 < \ell^4$ and a number $i \in [\ell+1]$ such that

$$\Pr_{r=G(U_\ell)}[C(r_1, \ldots, r_{i-1}) = r_i] > \tfrac{1}{2} + \tfrac{1}{20(\ell+1)}. \quad (2)$$

However, for every $i \leq \ell$, the i^{th} bit of $G(z)$ is completely uniform and independent from the first $i-1$ bits, and hence cannot be predicted with probability larger than $1/2$ by a circuit of any size. For $i = \ell+1$, Equation (2) becomes,

$$\Pr_{z \in_R \{0,1\}^\ell}[C(z) = f(z)] > \frac{1}{2} + \frac{1}{20(\ell+1)} > \frac{1}{2} + \frac{1}{\ell^4},$$

which cannot hold under the assumption that $\mathsf{H}_{\mathsf{avg}}(f) \geq n^4$. ■

Extending the input by two bits using the averaging principle.

We continue to progress in "baby steps" and consider the next natural toy problem: constructing a pseudorandom generator that extends its input by two bits. This is obtained in the following Lemma:

Lemma 20.11 *(Two-bit generator) Suppose that there exists $f \in \mathbf{E}$ with $\mathsf{H}_{\mathsf{avg}}(f) \geq n^4$. Then, there exists an $(\ell+2)$-pseudorandom generator G.* ♢

PROOF: The construction is again very natural: for every $z \in \{0,1\}^\ell$, we set

$$G(z) = z_1 \cdots z_{\ell/2} \circ f(z_1, \ldots, z_{\ell/2}) \circ z_{\ell/2+1} \cdots z_\ell \circ f(z_{\ell/2+1}, \ldots, z_\ell)\,.$$

Again, the efficiency and output length requirements are clearly satisfied. To show $G(U_\ell)$ is $((\ell+2)^3, 1/10)$-pseudorandom, we again use Theorem 20.10, and so need to prove that there does not exists a circuit C of size $2(\ell+1)^3$ and $i \in [\ell+2]$ such that

$$\Pr_{r=G(U_\ell)}[C(r_1, \ldots, r_{i-1}) = r_i] > \frac{1}{2} + \frac{1}{20(\ell+2)}. \quad (3)$$

[2] Although this theorem was stated and proved in Chapter 9 for the case of *uniform* Turing machines, the proof extends to the case of circuits; see Exercise 20.5.

Once again, (3) cannot occur for those indices i in which the i^{th} output of $G(z)$ is truly random, and so the only two cases we need to consider are $i = \ell/2 + 1$ and $i = \ell + 2$. Equation (3) cannot hold for $i = \ell/2 + 1$ for the same reason as in Lemma 20.9. For $i = \ell + 2$, Equation (3) becomes:

$$\Pr_{r,r' \in_R \{0,1\}^{\ell/2}}[C(r \circ f(r) \circ r') = f(r')] > \frac{1}{2} + \frac{1}{20(\ell+2)} \tag{4}$$

This may seem somewhat problematic to analyze since the input to C contains the bit $f(r)$, which C could not compute on its own (as f is a hard function). Couldn't it be that the input $f(r)$ helps C in predicting the bit $f(r')$? The answer is NO, and the reason is that r' and r are *independent*. Formally, we use the following principle (see Section A.2.2 in the appendix):

> The Averaging Principle: If A is some event depending on two independent random variables X, Y, then there exists some x in the range of X such that $\Pr_Y[A(x,Y)] \geq \Pr_{X,Y}[A(X,Y)]$.

Applying this principle here, if (4) holds then there exists a string $r \in \{0,1\}^{\ell/2}$ such that

$$\Pr_{r' \in_R \{0,1\}^{\ell/2}}[C(r, f(r), r') = f(r')] > \frac{1}{2} + \frac{1}{20(\ell+2)}\,.$$

(Note that this probability is now only over the choice of r'.) If this is the case, we can "hardwire" the $\ell/2 + 1$ bits $r \circ f(r)$ (as fixing r to some constant also fixes $f(r)$) to the circuit C and obtain a circuit D of size at most $2(\ell+2)^3 < (\ell/2)^4$ such that

$$\Pr_{r' \in_R \{0,1\}^{\ell/2}}[D(r') = f(r')] > \frac{1}{2} + \frac{1}{20(\ell+2)}\,,$$

contradicting the hardness of f. ■

Beyond two bits:

A generator that extends the output by two bits is still useless for our goals. We can generalize the proof of Lemma 20.11 to obtain a generator G that extends its input by k bits setting

$$G(z_1, \ldots, z_\ell) = z^1 \circ f(z^1) \circ z^2 \circ f(z^2) \cdots z^k \circ f(z^k)\,, \tag{5}$$

where z^i is the i^{th} block of ℓ/k bits in z. However, no matter how big we set k and no matter how hard the function f is, we cannot get in this way a generator that expands its input by a multiplicative factor larger than two. Note that to prove Theorem 20.6 we need a generator whose output might even be *exponentially larger* than the input! Clearly, we need a new idea.

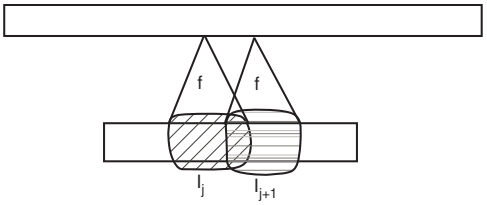
20.2.2 The NW Construction

The new idea is still inspired by the construction (5), but instead of taking $z^1, \ldots, z^k$ to be independently chosen strings (or equivalently, disjoint pieces of the input z), we take them to be *partly dependent* (non-disjoint pieces) by using *combinatorial designs*. Doing this will allow us to take k so large that we can drop the actual inputs from the generator's output and use only $f(z^1) \circ f(z^2) \cdots \circ f(z^k)$. The proof of correctness is similar to the above toy examples and uses Yao's technique, except the fixing of the input bits has to be done more carefully because of dependence among the strings.

Definition 20.12 *(NW Generator)*
Let $\mathcal{I} = \{I_1, \ldots, I_m\}$ be a family of subsets of $[\ell]$ with $|I_j| = n$ for every j, and let $f : \{0,1\}^n \to \{0,1\}$. The $(\mathcal{I}, f)$-*NW generator* is the function $\mathsf{NW}^f_{\mathcal{I}} : \{0,1\}^\ell \to \{0,1\}^m$ that maps every $z \in \{0,1\}^\ell$ to

$$\mathsf{NW}^f_{\mathcal{I}}(z) = f(z_{I_1}) \circ f(z_{I_2}) \cdots \circ f(z_{I_m}), \tag{6}$$

where for $z \in \{0,1\}^\ell$ and $I \subseteq [\ell]$, z_I denotes the restriction of z to the coordinates in I.



Conditions on the set systems and function.

We will see that in order for the generator to produce pseudorandom outputs, the function f must display some *hardness*, and the family of subsets should come from a *combinatorial design*, defined as follows:

Definition 20.13 *(Combinatorial designs)* Let d, n, ℓ satisfy $\ell > n > d$. A family $\mathcal{I} = \{I_1, \ldots, I_m\}$ of subsets of $[\ell]$ is an (ℓ, n, d)-*design* if $|I_j| = n$ for every j and $|I_j \cap I_k| \leq d$ for every $j \neq k$. ♢

The next lemma (whose proof we defer to the end of this section) yields sufficiently efficient constructions of such designs:

Lemma 20.14 *(Construction of designs) There is an algorithm A that on input $\langle \ell, d, n \rangle$ where $n > d$ and $\ell > 10n^2/d$, runs for $2^{O(\ell)}$ steps and outputs an (ℓ, n, d)-design $\mathcal{I}$ containing $2^{d/10}$ subsets of $[\ell]$.* ♢

The next lemma shows that if f is a hard function and $\mathcal{I}$ is a design with sufficiently good parameters, then $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is indeed a pseudorandom distribution:

Lemma 20.15 *(Pseudorandomness using the NW generator) If $\mathcal{I}$ is an (ℓ, n, d)-design with $|\mathcal{I}| = 2^{d/10}$ and $f : \{0,1\}^n \to \{0,1\}$ satisfies $\mathsf{H}_{\mathsf{avg}}(f) > 2^{2d}$ then the distribution $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is $(\mathsf{H}_{\mathsf{avg}}(f)/10, 1/10)$-pseudorandom.* ♢

Proof: Let $S = \mathsf{H}_{\mathsf{avg}}(f)$. By Yao's Theorem, to show that $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is $(S/10, 1/10)$-pseudorandom it suffices to prove that for every $i \in [2^{d/10}]$ there does *not* exist an $S/2$-sized circuit C such that

$$\Pr_{\substack{Z \sim U_\ell \\ R = \mathsf{NW}^f_{\mathcal{I}}(Z)}} [C(R_1, \ldots, R_{i-1}) = R_i] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}. \tag{7}$$

For contradiction's sake, assume that (7) holds for some circuit C and some i. Plugging in the definition of $\mathsf{NW}^f_{\mathcal{I}}$, (7) becomes:

$$\Pr_{Z \sim U_\ell} [C(f(Z_{I_1}), \cdots, f(Z_{I_{i-1}})) = f(Z_{I_i})] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}. \tag{8}$$

Letting Z_1 and Z_2 denote the two independent variables corresponding to the coordinates of Z in I_i and $[\ell] \setminus I_i$ respectively, (8) becomes:

$$\Pr_{\substack{Z_1 \sim U_n \\ Z_2 \sim U_{\ell-n}}} [C(f_1(Z_1, Z_2), \ldots, f_{i-1}(Z_1, Z_2)) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}, \tag{9}$$

where for every $j \in [2^{d/10}]$, f_j applies f to the coordinates of Z_1 corresponding to $I_j \cap I_i$ and the coordinates of Z_2 corresponding to $I_j \setminus I_i$. By the averaging principle, if (9) holds then there exists a string $z_2 \in \{0,1\}^{\ell-n}$ such that

$$\Pr_{Z_1 \sim U_n}[C(f_1(Z_1,z_2),\ldots,f_{i-1}(Z_1,z_2)) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}}\,. \tag{10}$$

We may now appear to be in some trouble, since all of $f_j(Z_1,z_2)$ for $j \leq i-1$ do depend upon Z_1, one might worry that they together contain enough information about Z_1 and so a circuit *could potentially* predict $f_i(Z_1)$ after seeing all of them. To prove that this fear is baseless we use the fact that $\mathcal{I}$ is a design and f is a sufficiently hard function.

Since $|I_j \cap I_i| \leq d$ for $j \neq i$, the function $Z_1 \mapsto f_j(Z_1,z_2)$ (for the fixed string z_2) depends at most d coordinates of Z_1 and hence can be trivially computed by a $d2^d$-sized circuit (or even $O(2^d/d)$ sized circuit, see Exercise 6.1). Thus if (9) holds then there exists a circuit B of size $2^{d/10} \cdot d2^d + S/2 < S$ such that

$$\Pr_{Z_1 \sim U_n}[B(Z_1) = f(Z_1)] \geq \frac{1}{2} + \frac{1}{10 \cdot 2^{d/10}} > \frac{1}{2} + \frac{1}{S}. \tag{11}$$

But this contradicts the fact that $S = \mathsf{H}_{\mathsf{avg}(\cdot)}(f)$. ■

The proof of Lemma 20.15 shows that if $\mathsf{NW}^f_{\mathcal{I}}(U_\ell)$ is distinguishable from the uniform distribution $U_{2^{d/10}}$ by some circuit D, then there exists a circuit B (of size polynomial in the size of D and in 2^d) that computes the function f with probability noticeably larger than $1/2$. The construction of this circuit B actually uses the circuit D as a *black-box*, invoking it on some chosen inputs. This property of the NW generator (and other constructions of pseudorandom generators) turned out to be useful in several settings. In particular, Exercise 20.7 uses it to show that under plausible complexity assumptions, the complexity class **AM** (containing all languages with a constant round interactive proof, see Chapter 8) is equal to **NP**. We will also use this property in Chapter 21 to construct *randomness extractors* based on pseudorandom generators.

Putting it all together: Proof of Theorem 20.6 from Lemmas 20.14 and 20.15

As noted in Remark 20.8, we do not prove here Theorem 20.6 as stated but only the weaker statement, that given $f \in \mathbf{DTIME}(2^{O(n)})$ and $S : \mathbb{N} \to \mathbb{N}$ with $\mathsf{H}_{\mathsf{avg}(\cdot)}(f) \geq S$, we can construct an $S'(\ell)$-pseudorandom generator, where $S'(\ell) = S(n)^\epsilon$ for some $\epsilon > 0$ and n satisfying $n \geq \epsilon\sqrt{\ell \log S(n)}$. On input $z \in \{0,1\}^\ell$, our generator will operate as follows:

- Set n to be the largest number such that $\ell > \frac{100n^2}{\log S(n)}$. Thus, $\ell \leq \frac{100(n+1)^2}{\log S(n+1)} \leq \frac{200n^2}{\log S(n)}$, and hence $n \geq \sqrt{\ell \log S(n)/200}$.
- Set $d = \log S(n)/10$.
- Run the algorithm of Lemma 20.14 to obtain an (ℓ,n,d)-design $\mathcal{I} = \{I_1,\ldots,I_{2^{d/5}}\}$.
- Output the first $S(n)^{1/40}$ bits of $\mathsf{NW}^f_{\mathcal{I}}(z)$.

This generator makes $2^{d/5}$ invocations of f, taking a total of $2^{O(n)+d}$ steps. By possibly reducing n by a constant factor, we can ensure the running time is bounded by 2^ℓ. Moreover, since $2^d \leq S(n)^{1/10}$, Lemma 20.15 implies that the distribution $\mathsf{NW}^f(U_\ell)$ is $(S(n)/10, 1/10)$-pseudorandom. ■

Construction of combinatorial designs.

All that is left to complete the proof is to show the construction of combinatorial designs with the required parameters:

Proof of Lemma 20.14 (construction of combinatorial designs): On inputs ℓ, d, n with $\ell > 10n^2/d$, our Algorithm A will construct an (ℓ,n,d)-design $\mathcal{I}$ with $2^{d/10}$ sets using the simple greedy strategy:

Start with $\mathcal{I} = \emptyset$ and after constructing $\mathcal{I} = \{I_1, \ldots, I_m\}$ for $m < 2^{d/10}$, search all subsets of $[\ell]$ and add to $\mathcal{I}$ the first n-sized set I satisfying the following condition **(*)**: $|I \cap I_j| \leq d$ for every $j \in [m]$.

Clearly, A runs in $\mathrm{poly}(m)2^\ell = 2^{O(\ell)}$ time and so we only need to prove it never gets stuck. In other words, it suffices to show that if $\ell = 10n^2/d$ and $\{I_1, \ldots, I_m\}$ is a collection of n-sized subsets of $[\ell]$ for $m < 2^{d/10}$, then there exists an n-sized subset $I \subseteq [\ell]$ satisfying **(*)**. We do so by showing that if we pick I at random by choosing independently every element $x \in [\ell]$ to be in I with probability $2n/\ell$ then:

$$\Pr[|I| \geq n] \geq 0.9 \tag{12}$$

$$\text{For every } j \in [m], \Pr[|I \cap I_j| \geq d] \leq 0.5 \cdot 2^{-d/10} \tag{13}$$

Because the expected size of I is $2n$, while the expected size of the intersection $I \cap I_j$ is $2n^2/\ell < d/5$, both (13) and (12) follow from the Chernoff bound. Yet, because $m \leq 2^{d/10}$, together these two conditions imply that with probability at least 0.4, the set I will simultaneously satisfy **(*)** and have size at least n. Since we can always remove elements from I without damaging **(*)**, this completes the proof. ■

20.3 Derandomization under uniform assumptions

Circuit lower bounds are notoriously hard to prove. Despite decades of effort, at the moment we do not know of a single function in **NP** requiring more than $5n$-sized circuits to compute, not to mention super-linear or super-polynomial circuits. A natural question is whether such lower bounds are *necessary* to achieve derandomization.[3] Note that pseudorandom generators as in Definition 20.2 can be easily shown to imply circuit lower bounds: see Exercise 20.4. However, there could potentially be a different way to show **BPP** = **P** without constructing pseudorandom generators.

The following result shows that to some extent this is possible: one can get a non-trivial derandomization of **BPP** under a *uniform* hardness assumption. Namely, that **BPP** $\neq$ **EXP**.

Theorem 20.16 *(Uniform derandomization* [IW98]*)*
Suppose that **BPP** $\neq$ **EXP**. *Then for every* $L \in$ **BPP** *there exists a subexponential (i.e.,* $2^{n^{o(1)}}$*) time deterministic algorithm* A *such that for infinitely many* n*'s*

$$\Pr_{x \in_R \{0,1\}^n}[A(x) = L(x)] \geq 1 - 1/n\,.$$

This means that unless randomness is a panacea, and every problem with an exponential time algorithm (including 3SAT, TQBF, the permanent, etc..) can be solved in probabilistic polynomial time, we can at least partially derandomize **BPP**: obtain a subexponential deterministic simulation that succeeds well in the average-case. In fact, the conclusion of Theorem 20.16 can be considerably strengthened: we can find an algorithm A that will solve L with probability $1 - 1/n$ not only for inputs chosen according to the uniform distribution, but on inputs chosen according to *every* distribution that can be sampled in polynomial time. Thus, while this deterministic simulation may sometimes fail, it is hard to find inputs on which it does!

Proof sketch of Theorem 20.16: We only sketch the proof of Theorem 20.16 here. We start by noting that we may assume in the proof that **EXP** $\subseteq \mathbf{P}_{/\mathbf{poly}}$, since otherwise there

[3] Note that we do not know much better lower bounds for Turing machines either. However, a-priori it seems that a result such as **BPP** $\neq$ exp may be easier to prove than circuit lower bounds, and that a natural first step to proving such a result is to get derandomization results without assuming such lower bounds.

is some problem in **EXP** with superpolynomial circuit complexity, and such a problem can be used to build a pseudorandom generator that is strong enough to imply the conclusion of the Theorem. (This follows from Theorem 20.7 and Exercise 20.8.) Next, note that if $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ then **EXP** is contained in the polynomial hierarchy (see Theorem 6.20 and also Lemma 20.18 below). But that implies $\mathbf{EXP} = \mathbf{PH}$ and hence we can conclude from Toda's and Valiant's theorems (Theorems 17.14 and 17.11) that the *permanent* function perm is **EXP**-complete under polynomial-time reductions. In addition, the Lemma hypothesis implies that perm is not in **BPP**. This is a crucial point in the proof since perm is a very special function that is downward self-reducible (see Chapter 8).

The next idea is to build a pseudorandom generator G with super-polynomial output size using the permanent as a hard function. We omit the details, but this can be done following the proofs of Theorems 19.27 and 20.6 (one needs to handle the fact that the permanent's output is not a single bit, but this can be handled for example using the Goldreich-Levin Theorem of Chapter 9). Looking at the proof of correctness for this pseudorandom generator G, it can be shown to yield an algorithm T to transform for every n a distinguisher D between G's output) and the uniform distribution into a polynomial-sized circuit C_n that computes perm_n (which this denotes the restriction of the permanent to length n inputs). This algorithm T is similar to the transformation shown in the proof of the standard NW generator (proof of Theorem 20.6): the only reason it is not efficient is that it requires computing the hard function (in this case the permanent) on several randomly chosen inputs, which are then "hardwired" into the distinguisher.[4]

Suppose for the sake of contradiction that the conclusion of Theorem 20.16 is false. This means that there is a probabilistic algorithm A whose derandomization using G fails with noticeable probability (over the choice of a random input) for all but finitely many input lengths. This implies that not only there is a sequence of polynomial-sized circuits $\{D_n\}$ distinguishing the output of G from the uniform distribution on all but finitely many input lengths, but in fact there is a probabilistic polynomial-time algorithm that on input 1^n will find such a circuit D_n with probability at least $1/n$ (Exercise 20.9). We now make the simplifying assumption that this probabilistic algorithm in facts finds such a distinguisher D_n with high probability, say at least $1 - 1/n^2$.[5] Plugging this into the proof of pseudorandomness for the generator G, this means that there exists a probabilistic polynomial-time algorithm T that can "learn" the permanent function: given oracle access to perm_n (the restriction of perm to length n inputs) the algorithm T runs in $\mathrm{poly}(n)$ time and produces a $\mathrm{poly}(n)$-sized circuit computing perm_n.

But using T we can come up with a probabilistic polynomial-time algorithm for the permanent that doesn't use any oracle! To compute the permanent on length n inputs, we compute inductively the circuits $C_1, \ldots, C_n$. Given the circuit C_{n-1} we can compute the permanent on length n inputs using the permanent's *downward self-reducibility* property (see Section 8.6.2 and the proof of Lemma 20.19 below), and so implement the oracle to T that allows us to build the circuit C_n. Since we assumed $\mathbf{BPP} \neq \mathbf{EXP}$, and under $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ the permanent is **EXP**-complete, we get a contradiction.■

20.4 Derandomization requires circuit lower bounds

Section 20.3 shows that circuit lower bounds imply derandomization. However, circuit lower bounds have proved tricky, so one can hope that derandomization could somehow be done *without* circuit lower bounds. In this section we show this is not the case: proving that $\mathbf{BPP} = \mathbf{P}$ or even that a specific problem in **BPP** (namely the problem ZEROP of testing

[4] The proof of Theorem 20.6 only showed that there exists some inputs that when these inputs and their answers are"hardwired" into the distinguisher then we get a circuit computing the hard function. However, because the proof used the probabilistic method / averaging argument, it's not hard to show that with good probability random inputs will do.

[5] This gap can be handled using the fact that the permanent is a low-degree polynomial and hence has certain self-correction / self-testing properties, see sections 8.6.2 and 19.4.2.

whether a given polynomial is identically zero) will imply super-polynomial lower bounds for either Boolean or arithmetic circuits. Depending upon whether you are an optimist or a pessimist, you can view this either as evidence that derandomizing **BPP** is difficult, or, as a reason to double our efforts to derandomize **BPP** since once we do so we'll get "two for the price of one": both derandomization and circuit lower bounds.

Recall (Definition 16.7) that we say that a function f defined over the integers is in $\mathbf{AlgP}^{\mathbb{Z}}_{/\mathbf{poly}}$ (or just $\mathbf{AlgP}_{/\mathbf{poly}}$ for short) if f can be computed by a polynomial size algebraic circuit whose gates are labeled by $+$, $-$, and $\times$.[6] We let perm denote the problem of computing the permanent of matrices over the integers. Recall also the *Polynomial Identity Testing* (ZEROP) problem in which the input consists of a polynomial represented by an arithmetic circuit computing it and we have to decide if it is the identically zero polynomial (see Example 20.1 and Section 7.2.3). The problem ZEROP is in $\mathbf{coRP} \subseteq \mathbf{BPP}$ and we will show that if it is in $\mathbf{P}$ then some super-polynomial circuit lower bounds hold:

Theorem 20.17 *(Derandomization implies lower bounds* [KI03]*)*
If $\mathsf{ZEROP} \in \mathbf{P}$ *then either* $\mathbf{NEXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$ *or* $\mathsf{perm} \notin \mathbf{AlgP}_{/\mathbf{poly}}$.

The Theorem is known to be true even if its hypothesis is relaxed to $\mathsf{ZEROP} \in \cap_{\delta>0}\mathbf{NTIME}(2^{n^\delta})$. Thus, even a derandomization of **BPP** to subexponential non-deterministic time would still imply super-polynomial circuit lower bounds. The proof of Theorem 20.17 relies on many results described earlier in the book.(This is a good example of "third generation" complexity results that use a clever combination of both "classical" results from the 60's and 70's and newer results from the 1990's.) Our first ingredient is the following lemma:

Lemma 20.18 *(*[BFL90],[BFNW93]*)* $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}} \Rightarrow \mathbf{EXP} = \mathbf{MA}$. ♢

Recall that **MA** is the class of languages that can be proven by a one round interactive proof between two players Arthur and Merlin (see Definition 8.10).

PROOF OF LEMMA 20.18: Suppose $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$. By Meyer's Theorem (Theorem 6.20), in this case **EXP** collapses to the second level $\mathbf{\Sigma}^p_2$ of the polynomial hierarchy. Hence under our assumptions $\mathbf{\Sigma}^p_2 = \mathbf{PH} = \mathbf{PSPACE} = \mathbf{IP} = \mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$. Thus every $L \in \mathbf{EXP}$ has an interactive proof, and furthermore, since our assumption implies that $\mathbf{EXP} = \mathbf{PSPACE}$, we can just use the interactive proof for TQBF, for which the prover is a **PSPACE** machine and (given that we assume $\mathbf{PSPACE} \subseteq \mathbf{P}_{/\mathbf{poly}}$) can be replaced by a polynomial size circuit family $\{C_n\}$. Now we see that the interactive proof can actually be carried out in one round: given an input x of length n, Merlin will send Arthur a polynomial size circuit C, which is supposed to be circuit C_n for the prover's strategy for L. Then Arthur simulates the interactive proof for L, using C as the prover and tossing coins to simulate the verifier. Note that if the input is not in the language, then *no* prover has a decent chance of convincing the verifier, and in particular this holds for the prover described by C. Thus we have described an **MA** protocol for L implying that $\mathbf{EXP} \subseteq \mathbf{MA}$ and hence that $\mathbf{EXP} = \mathbf{MA}$. ■

Our second lemma connects the complexity of identity testing and the permanent to the power of the class $\mathbf{P}^{\mathsf{perm}}$:

Lemma 20.19 *If* $\mathsf{ZEROP} \in \mathbf{P}$ *and* $\mathsf{perm} \in \mathbf{AlgP}_{/\mathbf{poly}}$ *then* $\mathbf{P}^{\mathsf{perm}} \subseteq \mathbf{NP}$. ♢

[6] The results below extend also to circuits that are allowed to work over the rational or real numbers and use division.

PROOF OF LEMMA 20.19: Suppose perm has algebraic circuits of size n^c, and that ZEROP has a polynomial-time algorithm. Let L be a language that is decided by an n^d-time TM M using queries to a perm-oracle. We construct an **NP** machine N for L.

Suppose x is an input of size n. Clearly, M's computation on x makes queries to perm of size at most $m = n^d$. So N will use nondeterminism as follows: it guesses a sequence of m algebraic circuits $C_1, C_2, \ldots, C_m$ where C_i has size i^c. The hope is that C_i solves perm on $i \times i$ matrices, and N will verify this in poly(m) time. The verification starts by verifying C_1, which is trivial. Inductively, having verified the correctness of $C_1, \ldots, C_{t-1}$, one can verify that C_t is correct using downward self-reducibility, namely, that for a $t \times t$ matrix A,

$$\mathsf{perm}(A) = \sum_{i=1}^{t} a_{1i}\mathsf{perm}(A_{1,i}),$$

where $A_{1,i}$ is the $(t-1)\times(t-1)$ sub-matrix of A obtained by removing the 1st row and ith column of A. Thus if circuit C_{t-1} is known to be correct, then the correctness of C_t can be checked by substituting $C_t(A)$ for $\mathsf{perm}(A)$ and $C_{t-1}(A_{1,i})$ for $\mathsf{perm}(A_{1,i})$: this yields an identity involving algebraic circuits with t^2 inputs which can be verified deterministically in poly(t) time using the algorithm for ZEROP. Proceeding this way N verifies the correctness of $C_1, \ldots, C_m$ and then simulates M^{perm} on input x using these circuits. ■

The heart of the proof of Theorem 20.17 is the following lemma, which is interesting in its own right:

Lemma 20.20 *([IKW01])* $\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}} \Rightarrow \mathbf{NEXP} = \mathbf{EXP}$. ♢

PROOF: We prove the contrapositive. That is, we assume that $\mathbf{NEXP} \neq \mathbf{EXP}$ and will prove thsat $\mathbf{NEXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$. Let $L \in \mathbf{NEXP} \setminus \mathbf{EXP}$ (such a language exists under our assumption). Since $L \in \mathbf{NEXP}$ there exists a constant $c > 0$ and a relation R such that

$$x \in L \Leftrightarrow \exists y \in \{0,1\}^{2^{|x|^c}} \text{ s.t. } R(x,y) \text{ holds},$$

where we can test whether $R(x,y)$ holds in, say, time $2^{|x|^{10c}}$.

We now consider the following approach to try to solve L in exponential deterministic time. For every constant $D > 0$, let M_D be the following machine: on input $x \in \{0,1\}^n$ enumerate over all possible Boolean circuits C of size n^{100D} that take n^c inputs and have a single output. For every such circuit let $\mathsf{tt}(C)$ be the 2^{n^c}-long string that corresponds to the truth table of the function computed by C. If $R(x, \mathsf{tt}(C))$ holds then halt and output 1. If this does not hold for any of the circuits then output 0. Since M_D runs in time $2^{n^{101D}+n^c}$, under our assumption that $L \notin \mathbf{EXP}$, M_D does not solve L and hence for every D there exists an infinite sequence of inputs $\mathcal{X}_D = \{x_i\}_{i\in\mathbb{N}}$ on which $M_D(x_i)$ outputs 0 even though $x_i \in L$ (note that M_D can only make one-sided errors). This means that for every string x in the sequence $\mathcal{X}_D$ and every y such that $R(x,y)$ holds, the string y represents the truth table of a function on n^c bits that cannot be computed by circuits of size n^{100D}, where $n = |x|$. Using the pseudorandom generator based on worst-case assumptions (Theorem 20.7), we can use such a string y to obtain an ℓ^D-pseudorandom generator. This method is called the "easy witness" method [Kab00], because it shows that unless the input x has a witness/certificate y (i.e., string satisfying $R(x,y) = 1$) that is "easy" in the sense that it can be computed by a small circuit, then any certificate for x can be used for derandomization.

Now, if $\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ then $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ and then by Lemma 20.18 $\mathbf{EXP} \subseteq \mathbf{MA}$. That is, every language in **EXP** has a proof system where Merlin proves that an n-bit string is in the language by sending a proof which Arthur then verifies using a probabilistic algorithm of at most n^D steps for some constant D. Yet, if n is the input length of some string in the sequence $\mathcal{X}_D$ and we are given $x \in \mathcal{X}_D$ with $|x| = n$, then we can replace Arthur by non-deterministic poly$(n^D)2^{n^{10c}}$ time algorithm that does not toss any coins: Arthur will

guess a string y such that $R(x,y)$ holds and then use y as a function for a pseudorandom generator to verify Merlin's proof.

This means that there is an absolute constant $c > 0$ such that *every* language in **EXP** can be decided on infinitely many inputs by an $\mathbf{NTIME}(2^{n^c})$ time algorithm using n bits of advice, and hence (since we assume $\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$) by a size $n^{c'}$ circuit family for an absolute constant c'. But using standard diagonalization we can easily come up with a language in $\mathbf{DTIME}(2^{O(n^{c'})}) \subseteq \mathbf{EXP}$ that cannot be computed by such a circuit family on almost every input. ■

It might seem that Lemma 20.20 should have an easier proof that goes along the lines of the proof of Lemma 20.18 ($\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}} \Rightarrow \mathbf{EXP} = \mathbf{MA}$) but instead of using the interactive proof for TQBF uses the *multi-prover* interactive proof system for **NEXP**. However, we do not know how to implement the provers' strategies for this latter system in **NEXP**. Intuitively, the problem arises from the fact that a **NEXP** statement may have several certificates, and it is not clear how we can ensure all provers use the same one.

We now have all the ingredients for the proof of Theorem 20.17.

Proof of Theorem 20.17: For contradiction's sake, assume that the following are all true:

$$\mathsf{ZEROP} \in \mathbf{P} \tag{14}$$

$$\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}}, \tag{15}$$

$$\mathsf{perm} \in \mathbf{AlgP}_{/\mathbf{poly}}. \tag{16}$$

Statement (15) together with Lemmas 20.18 and 20.20 imply that $\mathbf{NEXP} = \mathbf{EXP} = \mathbf{MA}$. Now recall that $\mathbf{MA} \subseteq \mathbf{PH}$, and that by Toda's Theorem (Theorem 17.14) $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$. Recall also that by Valiant's Theorem (Theorem 17.11) perm is $\#\mathbf{P}$-complete. Thus, under our assumptions

$$\mathbf{NEXP} \subseteq \mathbf{P}^{\mathsf{perm}}. \tag{17}$$

Since we assume that $\mathsf{ZEROP} \in \mathbf{P}$, Lemma 20.19 together with statements (16) and (17) implies that $\mathbf{NEXP} \subseteq \mathbf{NP}$, contradicting the Nondeterministic Time Hierarchy Theorem (Theorem 3.2). Thus the three statements (14), (15) and (14) cannot be simultaneously true. ■

What have we learned?

- Under the assumption of certain circuit lower bounds, there exist pseudorandom generator that can derandomize every probabilistic algorithm.
- In particular, if we make the reasonable assumption that there exists a function in **E** with exponentially large average-case circuit complexity, then $\mathbf{BPP} = \mathbf{P}$.
- Proving that $\mathbf{BPP} = \mathbf{P}$ will require to prove at least some type of circuit lower bounds.

Chapter notes and history

As mentioned in the notes to Chapter 9, pseudorandom generators were first studied in the context of cryptography, by Shamir [Sha81], Blum-Micali [BM82], and Yao [Yao82a]. In particular Yao was the first to point out their potential uses for derandomizing **BPP**. He showed that if secure pseudorandom generators exist then **BPP** can be partially derandomized, specifically, $\mathbf{BPP} \subseteq \cap_{\epsilon>0}\mathbf{DTIME}(2^{n^\epsilon})$. In their seminal paper [NW88], Nisan and Wigderson showed that such derandomization is possible under significantly weaker complexity assumptions, and that under some plausible assumptions it may even be possible to achieve full derandomization of **BPP**,

namely, to show $\mathbf{BPP} = \mathbf{P}$. Since then a large body of work by was devoted to improving the derandomization and weakening the assumptions (see also the notes to Chapter 19). In particular it was shown that *worst-case* hardness assumptions suffice for derandomization (see Chapter 19 and its notes). A central goal of this line of work was achieved by Impagliazzo and Wigderson [IW97], who showed that if $\mathbf{E}$ has a function with exponential circuit complexity then $\mathbf{BPP} = \mathbf{P}$.

A pseudorandom generator with optimal dependence on the hardness assumptions (Theorem 20.6) was given by Umans [Uma03] (see Remark 20.8). Interestingly, this pseudorandom generator is based directly on *worst-case* (as opposed to *average-case*) hardness (and indeed uses the local-decoding techniques originating from the works on hardness amplification). Umans' construction, which uses the Reed-Muller code described in Chapter 19, is based on a previous paper of Shaltiel and Umans [SU01] that constructed a *hitting set generator* (a relaxation of a pseudorandom generator) with the same parameters. Andreev, Clementi, and Rolim [ACR96] showed that such hitting set generators suffice for the application of derandomizing **BPP** (see [GVW00] for a simpler proof).

Impagliazzo and Wigderson [IW98] gave the first derandomization result based on the *uniform* hardness of a function in **EXP** (i.e., Theorem 20.16), a result that gave hope that perhaps the proof of $\mathbf{BPP} = \mathbf{P}$ (or at least $\mathbf{BPP} \neq \mathbf{EXP}$) will not have to wait for progress on circuit lower bounds. Alas, Impagliazzo, Kabanets and Wigderson [IKW01] showed that derandomizing **MA** (or equivalently, the promise-problem version of **BPP**) would imply lower bounds for **NEXP**, while Kabanets and Impagliazzo [KI03] proved Theorem 20.17. That is, they showed that some circuit lower bounds would follow even from derandomizing **BPP**.

Exercises

20.1 Verify Corollary 20.4.

20.2 Show that there exists a number $\epsilon > 0$ and a function $G : \{0,1\}^* \to \{0,1\}^*$ that satisfies all of the conditions of a $2^{\epsilon n}$-pseudorandom generator per Definition 20.2, save for the computational efficiency condition. H464

20.3 Show by a counting argument (i.e., probabilistic method) that for every large enough n there is a function $f : \{0,1\}^n \to \{0,1\}$, such that $\mathsf{H}_{\mathsf{avg}}(f) \geq 2^{n/10}$.

20.4 Prove that if there exists a an $S(\ell)$-pseudorandom generator then there exists a function $f \in \mathbf{DTIME}(2^{O(n)})$ such that $\mathsf{H}_{\mathsf{wrs}}(f)(n) \geq S(n)$. H464

20.5 Prove Theorem 20.10.

20.6 Prove that if there exists $f \in \mathbf{E}$ and $\epsilon > 0$ such that $\mathsf{H}_{\mathsf{avg}}(f)(n) \geq 2^{\epsilon n}$ for every $n \in \mathbb{N}$, then $\mathbf{MA} = \mathbf{NP}$. H464

20.7 We define an *oracle Boolean circuit* to be a Boolean circuit that have special gates with unbounded fan-in that are marked ORACLE. For a Boolean circuit C and language $O \subseteq \{0,1\}^*$, we define by $C^O(x)$ the output of C on x, where the operation of the oracle gates when fed input q is to output 1 iff $q \in O$.

(a) Prove that if every $f \in \mathbf{E}$ can be computed by a polynomial-size circuits with oracle to SAT, then the polynomial hierarchy collapses.

(b) For a function $f : \{0,1\}^* \to \{0,1\}$ and $O \subseteq \{0,1\}^*$, define $\mathsf{H}_{\mathsf{avg}}{}^O(f)$ to be the function that maps every $n \in \mathbb{N}$ to the largest S such that $\Pr_{x \in_R \{0,1\}^n}[C^O(x) = f(x)] \leq 1/2 + 1/S$. Prove that if there exists $f \in \mathbf{E}$ and $\epsilon > 0$ with $\mathsf{H}_{\mathsf{avg}}{}^{\mathsf{3SAT}}(f) \geq 2^{\epsilon n}$ then $\mathbf{AM} = \mathbf{NP}$.

20.8 Prove that if $\mathbf{EXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$ then the conclusions of Theorem 20.16 hold. H464

20.9 Let $G : \{0,1\}^* \to \{0,1\}^*$ be an $S(\ell)$-length candidate pseudorandom generator that *fails* to derandomize a particular **BPP** algorithm A on the average case. That is, letting $L \in \mathbf{BPP}$ be the language such that $\Pr[A(x) = L(x)] \geq 2/3$, it holds that for every sufficiently large n, with probability at least $1/n$ over the choice of $x \in_R \{0,1\}^n$, $\Pr[A(x; G(U_{\ell(n)})) = L(x)] \leq 1/2$ (we let $\ell(n)$ be such that $S(\ell(n)) = m(n)$ where $m(n)$ denotes the length of random tape used by A on inputs of length n). Prove that there exists a probabilistic polynomial-time algorithm D that on input 1^n outputs a circuit D_n such that with probability at least $1/(2n)$ over the randomness of D,

$$\| \mathsf{E}[D_n(G(U_{\ell(n)})] - \mathsf{E}[D_n(U_{m(n)})] \| > 0.1 \,.$$

H464

20.10 (van Melkebeek 2000, see [IKW01]) Prove that if $\mathbf{NEXP} = \mathbf{MA}$ then $\mathbf{NEXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$.

Chapter 21

Pseudorandom constructions: expanders and extractors

"How difficult could it be to find hay in a haystack?"
Howard Karloff

The probabilistic method is a powerful method to show the existence of objects (e.g., graphs, functions) with certain desirable properties. We have already seen it used in Chapter 6 to show the existence of functions with high circuit complexity, in Chapter 19 to show the existence of good error correcting codes, and in several other places in this book. But sometimes the mere *existence* of an object is not enough: we need an *explicit* and *efficient* construction. This chapter provides such constructions for two well-known (and related) families of pseudorandom objects, *expanders* and *extractors*. They are important in computer science because they can often be used to replace (or reduce) the amount of randomness needed in certain settings. This is reminiscent of derandomization, the topic of Chapter 20, and indeed we will see several connections to derandomization throughout the chapter. However, a big difference between Chapter 20 and this one is that all results proven here are *unconditional*, in other words do not rely on unproven assumptions. Another topic that is related to expanders is constructions of error correcting-codes and related hardness-amplification results which we saw in Chapter 19. For a brief discussion of the many deep and fascinating connections between codes, expanders, pseudorandom generators, and extractors, see the Chapter notes.

Expanders are graphs whose connectivity properties (how many edges lie between every two sets A, B of vertices) are similar to those of "random" graphs —in this sense they are "pseudorandom" or "like random." Expanders have found a vast number of applications ranging from fast sorting networks, to counterexamples in metric space theory, to proving the **PCP** Theorem. The study of expanders is closely tied to study of *eigenvalues* of adjacency matrices. In Section 21.1 we lay the groundwork for this study, showing how random walks on graphs can be analyzed in terms of the adjacency matrix's eigenvalues. Then in Section 21.2 we give two equivalent definitions for expander graphs. We also describe their use in randomness-efficient error reduction of probabilistic algorithms. In Section 21.3 we show an explicit construction of expander graphs. Finally, in Section 21.4, we use this construction to show a *deterministic* logspace algorithm for undirected graph connectivity.

Our second example of an explicit construction concerns the following issue: while randomized algorithms are modeled using a sequence of unbiased and independent coin tosses, real-life randomness sources are imperfect and have correlations and biases. Philosophically speaking, it is unclear if there is even a single source of unbiased random bits in the universe. Therefore researchers have tried to quantify ways in which a source of random bits could be imperfect and still be used to run randomized algorithms.

In Section 21.5 we define *weakly random* sources. This definition encapsulates the minimal notion of "randomness" that still allows an imperfect source to be used in randomized algorithms. We also define *randomness extractors* (or extractors for short)— algorithms

that extract unbiased random bits from such a source — and give explicit constructions for them. One philosophical consequence of these results is that the model of randomized polynomial-time Turing machines (and the associated classes like **BPP**) is realistic if and only if weakly random sources exist in the real world.

In Section 21.6 we use extractors to derandomize probabilistic logspace computations, albeit at the cost of some increase in the space requirement. We emphasize that in contrast to the results of Chapters 19 and 20, this derandomization (as well as all the other results of this chapter) is *unconditional* and uses no unproven assumptions.

Both the constructions and analysis of this chapter are somewhat involved. You might wonder why should coming up with explicit construction be so difficult. After all, a proof of existence via the probabilistic method shows not only that an object with the desired property exists but in fact the vast majority of objects have the property. As Karloff said (see quote above), how difficult can it be to find a single one? But perhaps it's not so surprising this task is so difficult: after all, we know that almost all Boolean functions have exponential circuit complexity, but finding even a single one in **NP** with this property will show that $\mathbf{P} \neq \mathbf{NP}$!

21.1 Random walks and eigenvalues

In this section we study random walks on graphs. Using elementary linear algebra we relate eigenvalues of the graph's adjacency matrix to the behavior of the random walk on that graph. As a corollary we obtain the proof of correctness for the random-walk space-efficient algorithm for undirected connectivity described in Section 7.7. We restrict ourselves here to *regular* graphs, in which every vertex has the same degree. However, we do allow our graphs to have self-loops and parallel edges. Most of the definitions and results can be suitably generalized to undirected graphs that are not regular.

Some linear algebra. We will use some basic properties of the linear space $\mathbb{R}^n$. These are covered in Section A.5 of the appendix, but here is a quick review. If $\mathbf{u}, \mathbf{v} \in R^n$ are two vectors, then their *inner product* is defined as $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^{n} \mathbf{u}_i \mathbf{v}_i$. We say that $\mathbf{u}$ and $\mathbf{v}$ are *orthogonal*, denoted by $\mathbf{u} \perp \mathbf{v}$, if $\langle \mathbf{u}, \mathbf{v} \rangle = 0$. The *$L_2$-norm* of a vector $\mathbf{v} \in \mathbb{R}^n$, denoted by $\|\mathbf{v}\|_2$ is defined as $\sqrt{\langle \mathbf{v}, \mathbf{v} \rangle} = \sqrt{\sum_{i=1}^{n} \mathbf{v}_i^2}$. A vector whose L_2-norm equals 1 is called a *unit vector*. A simple but useful fact is the *Pythagorean Theorem*, that says that if $\mathbf{u}$ and $\mathbf{v}$ are orthogonal then $\|\mathbf{u} + \mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2$. The *$L_1$-norm* of $\mathbf{v}$, denoted by $|\mathbf{v}|_1$ is defined as $\sum_{i=1}^{n} |\mathbf{v}_i|$. Both these norms satisfy the basic properties **(1)** $\|\mathbf{v}\| > 0$ with $\|\mathbf{v}\| = 0$ iff $\mathbf{v}$ is the all zero vector, **(2)** $\|\alpha \mathbf{v}\| = |\alpha| \|\mathbf{v}\|$ for every $\alpha \in \mathbb{R}$, and **(3)** $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$. The relation between these norms is captured in the following claim, whose proof is left as Exercise 21.1:

Claim 21.1 *For every vector $\mathbf{v} \in \mathbb{R}^n$,*

$$|\mathbf{v}|_1 / \sqrt{n} \leq \|\mathbf{v}\|_2 \leq |\mathbf{v}|_1 . \qquad \diamondsuit$$

21.1.1 Distributions as vectors and the parameter $\lambda(G)$.

Let G be a d-regular n-vertex graph and let $\mathbf{p}$ be some probability distribution over the vertices of G. We can think of $\mathbf{p}$ as a (column) vector in $\mathbb{R}^n$ where $\mathbf{p}_i$ is the probability that vertex i is obtained by the distribution. Note that the L_1-norm of $\mathbf{p}$ is equal to 1. Now let $\mathbf{q}$ represent the distribution of the following random variable: choose a vertex i in G according to $\mathbf{p}$, then take a random neighbor of i in G. We can easily compute $\mathbf{q}$, since the probability $\mathbf{q}_j$ that j is chosen is equal to the sum over all of j's neighbors i of the probability $\mathbf{p}_i$ that i is chosen times $1/d$ (since vertex i touches d edges, for each edge $\overline{i\,j}$ the probability that conditioned on i being chosen then the next move will take this edge is $1/d$). Thus $\mathbf{q} = A\mathbf{p}$,

where $A = A(G)$ is the matrix such that for every two vertices i, j of G, $A_{i,j}$ is equal to the number of edges between i and j divided by d. (In other words, A is equal to the adjacency matrix of G multiplied by $1/d$.) We call A the *random-walk matrix* of G. Note that A is a symmetric matrix[1] with all its entries between 0 and 1, and the sum of entries in each row and column is exactly one. Such a matrix is called a symmetric *stochastic* matrix.

The relation between the matrix A and random walks on the graph G is straightforward—for every $\ell \in \mathbb{N}$ and $i \in [n]$, the vector $A^\ell \mathbf{e}^i$ (where $\mathbf{e}^i$ is the vector that has 1 in the i^{th} coordinate and zero everywhere else) represents the distribution X_ℓ of the last step in an ℓ-step random walk starting from the i^{th} vertex.

Definition 21.2 *(The parameter $\lambda(G)$.)*
Denote by $\mathbf{1}$ the vector $(1/n, 1/n, \ldots, 1/n)$ corresponding to the uniform distribution. Denote by $\mathbf{1}^\perp$ the set of vectors perpendicular to $\mathbf{1}$ (i.e., $\mathbf{v} \in \mathbf{1}^\perp$ if $\langle \mathbf{v}, \mathbf{1} \rangle = (1/n)\sum_i \mathbf{v}_i = 0$). The parameter $\lambda(A)$, denoted also as $\lambda(G)$, is the maximum value of $\|A\mathbf{v}\|_2$ over all vectors $\mathbf{v} \in \mathbf{1}^\perp$ with $\|\mathbf{v}\|_2 = 1$.

Relation to eigenvalues. The value $\lambda(G)$ is called the *second largest eigenvalue* of G. The reason is that since A is a symmetric matrix, we can find an orthogonal basis of eigenvectors $\mathbf{v}^1, \ldots, \mathbf{v}^n$ with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$ (see Section A.5.3) which we can sort to ensure $|\lambda_1| \geq |\lambda_2| \ldots \geq |\lambda_n|$. Note that $A\mathbf{1} = \mathbf{1}$. Indeed, for every i, $(A\mathbf{1})_i$ is equal to the inner product of the i^{th} row of A and the vector $\mathbf{1}$ which (since the sum of entries in the row is one) is equal to $1/n$. Thus, $\mathbf{1}$ is an *eigenvector* of A with the corresponding eigenvalue equal to 1. One can show that a symmetric stochastic matrix has all eigenvalues with absolute value at most 1 (see Exercise 21.5) and hence we can assume $\lambda_1 = 1$ and $\mathbf{v}^1 = \mathbf{1}$. Also, because $\mathbf{1}^\perp = \mathrm{Span}\{\mathbf{v}^2, \ldots, \mathbf{v}^n\}$, the value λ above will be maximized by (the normalized version of) $\mathbf{v}^2$, and hence $\lambda(G) = |\lambda_2|$. The quantity $1 - \lambda(G)$ is called the *spectral gap* of the graph. We note that some texts define the parameter $\lambda(G)$ using the standard (un-normalized) adjacency matrix (rather than the random-walk matrix), in which case $\lambda(G)$ is a number between 0 and d and the spectral gap is defined to be $d - \lambda(G)$. Knowledge of basic facts on eigenvalues and eigenvectors (all covered in the appendix) can serve as useful background for this chapter, but is not strictly necessary to follow the results and proofs.

One reason that $\lambda(G)$ is an important parameter is the following lemma:

Lemma 21.3 *Let G ba an n-vertex regular graph and $\mathbf{p}$ a probability distribution over G's vertices, then*

$$\|A^\ell \mathbf{p} - \mathbf{1}\|_2 \leq \lambda^\ell \qquad \diamondsuit$$

PROOF: By the definition of $\lambda(G)$, $\|A\mathbf{v}\|_2 \leq \lambda \|\mathbf{v}\|_2$ for every $\mathbf{v} \perp \mathbf{1}$. Note that if $\mathbf{v} \perp \mathbf{1}$ then $A\mathbf{v} \perp \mathbf{1}$ since $\langle \mathbf{1}, A\mathbf{v} \rangle = \langle A^\dagger \mathbf{1}, \mathbf{v} \rangle = \langle \mathbf{1}, \mathbf{v} \rangle = 0$ (as $A = A^\dagger$ and $A\mathbf{1} = \mathbf{1}$). Thus A maps the subspace $\mathbf{1}^\perp$ to itself. Note that the eigenvectors that are different from $\mathbf{1}$ span this subspace, and A shrinks each of these eigenvectors by at least λ factor in ℓ_2 norm. Hence A must shrink every vector in $\mathbf{1}^\perp$ by at least λ. Thus A^ℓ shrinks every vector in $\mathbf{1}^\perp$ by a factor at least λ^ℓ and we conclude $\lambda(A^\ell) \leq \lambda(A)^\ell$. (In fact, using the eigenvalue definition of λ, it can be shown that $\lambda(A^\ell) = \lambda(A)^\ell$.)

Let $\mathbf{p}$ be some vector. We can break $\mathbf{p}$ into its components in the spaces parallel and orthogonal to $\mathbf{1}$ and express it as $\mathbf{p} = \alpha\mathbf{1} + \mathbf{p}'$ where $\mathbf{p}' \perp \mathbf{1}$ and α is some number. If $\mathbf{p}$ is a probability distribution then $\alpha = 1$ since the sum of coordinates in $\mathbf{p}'$ is zero. Therefore,

$$A^\ell \mathbf{p} = A^\ell(\mathbf{1} + \mathbf{p}') = \mathbf{1} + A^\ell \mathbf{p}'$$

[1] A matrix A is *symmetric* if $A_{i,j} = A_{j,i}$ for every i, j. That is, $A = A^\dagger$ where $A^\dagger$ denotes the *transpose* of A (see Section A.5).

Since $\mathbf{1}$ and $\mathbf{p}'$ are orthogonal, $\|\mathbf{p}\|_2^2 = \|\mathbf{1}\|_2^2 + \|\mathbf{p}'\|_2^2$ and in particular $\|\mathbf{p}'\|_2 \leq \|\mathbf{p}\|_2$. Since $\mathbf{p}$ is a probability vector, $\|\mathbf{p}\|_2 \leq |\mathbf{p}|_1 = 1$ (see Claim 21.1). Hence $\|\mathbf{p}'\|_2 \leq 1$ and

$$\|A^\ell \mathbf{p} - \mathbf{1}\|_2 = \|A^\ell \mathbf{p}'\|_2 \leq \lambda^\ell \tag{1}$$

■

It turns out that every connected graph has a noticeable spectral gap:

Lemma 21.4 *If G is a regular connected graph with self-loops at each vertex, then $\lambda(G) \leq 1 - \frac{1}{12n^2}$.* ♢

PROOF: Let $\epsilon = \frac{1}{6n^2}$, let $\mathbf{u} \perp \mathbf{1}$ be a unit vector and let $\mathbf{v} = A\mathbf{u}$. We need to prove that $\|\mathbf{v}\|_2 \leq 1 - \epsilon/2$ and for this it suffices to prove that $1 - \|\mathbf{v}\|_2^2 \geq \epsilon$. (Indeed, if $\|\mathbf{v}\|_2 > 1 - \epsilon/2$ then $\|\mathbf{v}\|_2^2 > 1 - \epsilon$ and hence $1 - \|\mathbf{v}\|_2^2 < \epsilon$.) Since $\mathbf{u}$ is a unit vector, $1 - \|\mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 - \|\mathbf{v}\|_2^2$. We claim that this is equal to $\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2$ where i, j range from 1 to n. Indeed,

$$\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 = \sum_{i,j} A_{i,j}\mathbf{u}_i^2 - 2\sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j + \sum_{i,j} A_{i,j}\mathbf{v}_j^2 =$$
$$\|\mathbf{u}\|_2^2 - 2\langle A\mathbf{u}, \mathbf{v}\rangle + \|\mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 - 2\|\mathbf{v}\|_2^2 + \|\mathbf{v}\|_2^2 = \|\mathbf{u}\|_2^2 - \|\mathbf{v}\|_2^2 ,$$

where these equalities are due to the sum of each row and column in A equalling one, and to $\|\mathbf{v}\|_2^2 = \langle \mathbf{v}, \mathbf{v}\rangle = \langle A\mathbf{u}, \mathbf{v}\rangle = \sum_{i,j} A_{i,j}\mathbf{u}_i\mathbf{v}_j$.

Thus it suffices to show $\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \geq \epsilon$. Since $\mathbf{u}$ is a unit vector with coordinates summing to zero, there must exist vertices i, j such that $\mathbf{u}_i > 0, \mathbf{u}_j < 0$ and at least one of these coordinates has absolute value $\geq \frac{1}{\sqrt{n}}$, meaning that $\mathbf{u}_i - \mathbf{u}_j \geq \frac{1}{\sqrt{n}}$. Furthermore, because G is connected there is a path between i and j containing at most $D + 1$ vertices (where D is the *diameter* of the graph[2] G). By renaming vertices, let's assume that $i = 1$, $j = D + 1$ and the coordinates $2, 3, \ldots, D$ correspond to the vertices on this path in order. Now, we have

$$\tfrac{1}{\sqrt{n}} \leq \mathbf{u}_1 - \mathbf{u}_{D+1} = (\mathbf{u}_1 - \mathbf{v}_1) + (\mathbf{v}_1 - \mathbf{u}_2) + \ldots + (\mathbf{v}_D - \mathbf{u}_{D+1}) \leq$$
$$\mathbf{u}_1 - \mathbf{u}_{D+1} = |\mathbf{u}_1 - \mathbf{v}_1| + |\mathbf{v}_1 - \mathbf{u}_2| + \ldots + |\mathbf{v}_D - \mathbf{u}_{D+1}| \leq$$
$$\sqrt{(\mathbf{u}_1 - \mathbf{v}_1)^2 + (\mathbf{v}_1 - \mathbf{u}_2)^2 + \ldots + (\mathbf{v}_D - \mathbf{u}_{D+1})^2}\sqrt{2D+1}\,, \tag{2}$$

where the last inequality follows by relating the L_2 and L_1 norms of the vector $(\mathbf{u}_1 - \mathbf{v}_1, \mathbf{v}_1 - \mathbf{u}_2, \ldots, \mathbf{v}_D - \mathbf{u}_{D+1})$ using Claim 21.1. But this means that

$$\sum_{i,j} A_{i,j}(\mathbf{u}_i - \mathbf{v}_j)^2 \geq \tfrac{1}{dn(2D+1)}\,, \tag{3}$$

since the left hand side of (3) is a sum of non-negative terms and (2) implies that the terms of the form $A_{i,i}(\mathbf{u}_i - \mathbf{v}_i)^2$ and $A_{i,i+1}(\mathbf{v}_i - \mathbf{u}_{i+1})^2$ (for $i = 1, \ldots, D$) contribute at least $\frac{1}{dn(2D+1)}$ to this sum (both $A_{i,i}$ and $A_{i,i+1}$ are at least $1/d$ since they correspond to self-loops and edges of the graph).

Plugging in the trivial bound $D \leq n - 1$ this already shows that $\lambda(G) \leq 1 - \frac{1}{4dn^2}$. To prove the lemma as stated, we use the fact (left as Exercise 21.4) that for every d-regular connected graph, $D \leq 3n/(d+1)$. ■

The proof can be strengthened to show a similar result for every connected non-bipartite graph (not just those with self-loops at every vertex). Note that this condition is essential: if A is the random-walk matrix of a bipartite graph then one can find a vector $\mathbf{v}$ such that $A\mathbf{v} = -\mathbf{v}$ (Exercise 21.3).

[2]The *diameter* of a graph G is the maximum distance (i.e., length of shortest path) between any pair of vertices in G. Note that the diameter of a connected n-vertex graph is always at most $n - 1$.

21.1.2 Analysis of the randomized algorithm for undirected connectivity.

Together, lemmas 21.3 and 21.4 imply that, at least for regular graphs, if s is connected to t then a sufficiently long random walk from s will hit t in polynomial time with high probability:

Corollary 21.5 *Let G be a d-regular n-vertex graph with all vertices having a self-loop. Let s be a vertex in G. Let $\ell > 24n^2 \log n$ and let X_ℓ denote the distribution of the vertex of the ℓ^{th} step in a random walk from s. Then, for every t connected to s, $\Pr[X_\ell = t] > \frac{1}{2n}$.*$\diamondsuit$

PROOF: By Lemmas 21.3 and 21.4 , if we consider the restriction of an n-vertex graph G to the connected component of s, then for every probability vector $\mathbf{p}$ over this component and $\ell \geq 13n^2$, $\|A^\ell \mathbf{p} - \mathbf{1}\|_2 < (1 - \frac{1}{12n^2})^{24n^2 \log n} < \frac{1}{n^2}$, where $\mathbf{1}$ here is the uniform distribution over this component. But this means that in particular for every coordinate i, $|A^\ell \mathbf{p} - \mathbf{1}|_i < \frac{1}{n^2}$ and hence every element in the connected component appears in $A^\ell \mathbf{p}$ with probability at least $1/n - 1/n^2 \geq 1/(2n)$. ■

Note that Corollary 21.5 implies that if we repeat the $24n^2 \log n$ walk for $O(n \log n)$ times (or equivalently, if we take a walk of, say, length $100n^3 \log^2 n$) then we will hit *every* vertex t connected to s with high probability.

21.2 Expander graphs.

Expander graphs are extremely useful combinatorial objects, which we encounter several times in the book. They can be defined in two equivalent ways. At a high level, these two equivalent definitions can be described as follows:

- *Combinatorial definition:* A constant-degree regular graph G is an *expander* if for every subset S of less than half of G's vertices, a constant fraction of the edges touching S are from S to its complement in G; see Figure 21.1.
- *Algebraic expansion:* A constant-degree regular graph G is an *expander* if its parameter $\lambda(G)$ bounded away from 1 by some constant. That is, $\lambda(G) \leq 1 - \epsilon$ for some constant $\epsilon > 0$.

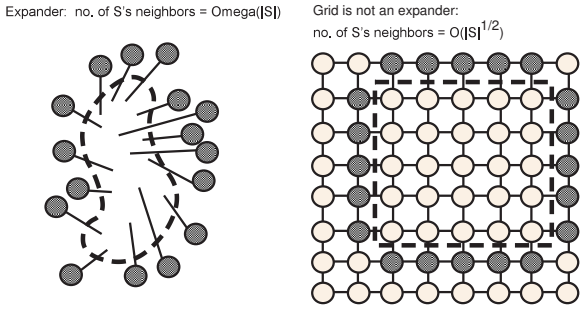


Figure 21.1 In an *edge expander*, every subset S of the vertices that is not too big has at least $\Omega(|S|)$ edges to neighbors outside the set. The grid (and every other planar graph) is not an edge expander as a $k \times k$ square in the grid has only $O(k)$ neighbors outside it.

What do we mean by a constant? By *constant* we refer to a number that is independent of the size of the graph. We will typically talk about graphs that are part of an infinite *family* of graphs, and so by constant we mean a value that is the same for all graphs in the family, regardless of their size. Below we make the definitions more precise, and show their equivalence.

21.2.1 The Algebraic Definition

The Algebraic definition of expanders is as follows:

Definition 21.6 *((n, d, λ)-expander graphs.)*
If G is an n-vertex d-regular G with $\lambda(G) \leq \lambda$ for some number $\lambda < 1$ then we say that G is an (n, d, λ)-graph.
A family of graphs $\{G_n\}_{n\in\mathbb{N}}$ is an *expander graph family* if there are some constants $d \in \mathbb{N}$ and $\lambda < 1$ such that for every n, G_n is an (n, d, λ)-graph.

Many texts use simply the name (n, d, λ)-graphs for such graphs. Also, as mentioned above, some texts use *un-normalized* adjacency matrices, and so have λ range between 0 and d. The smallest λ can be for a d-regular n-vertex graph is $(1-o(1))\frac{2\sqrt{d-1}}{d}$ where $o(1)$ denotes a function tending to 0 as the number of vertices grows. This is called the Alon-Boppana bound and graphs meeting this bound are called *Ramanujan graphs* (see also Exercises 21.9 and 21.10).

Explicit constructions. As we will see in Section 21.2.2, it is not hard to show that expander families exist using the probabilistic method. But this does not yield *explicit* constructions of such graphs which are often needed for applications. We say that an expander family $\{G_n\}_{n\in\mathbb{N}}$ is *explicit* if there is a polynomial-time algorithm that on input 1^n outputs the adjacency matrix of G_n (or, equivalently, the random-walk matrix). We say that the family is *strongly explicit* if there is a polynomial-time algorithm that on inputs $\langle n, v, i\rangle$ where $v \in [n], i \in [d]$ outputs the (index of the) i^{th} neighbor of v. Note that in the strongly explicit case, the lengths of the algorithm's inputs and outputs are $O(\log n)$ and so it runs in time polylog(n).

Fortunately, several explicit and strongly explicit constructions of expander graphs are known. Some of these constructions are very simple and efficient, but their analysis is highly non-trivial and uses relatively deep mathematics.[3] In Section 21.3 we will see a strongly explicit construction of expanders with elementary analysis. This construction also introduces a tool that we'll use to derandomize the random-walk algorithm for UPATH in Section 21.4.

21.2.2 Combinatorial expansion and existence of expanders.

We now describe a combinatorial criterion that is roughly equivalent to Definition 21.6. One advantage of this criterion is that it makes it easy to prove that a non-explicit expander family exists using the probabilistic method. It is also quite useful in several applications.

Definition 21.7 *(Combinatorial (edge) expansion)*
An n-vertex d-regular graph $G = (V, E)$ is called an (n, d, ρ)-*combinatorial edge expander* if for every subset S of vertices satisfying $|S| \leq n/2$,

$$|E(S, \overline{S})| \geq \rho d|S|\,,$$

where $\overline{S}$ denotes the complement of S and for subsets S, T of vertices, $E(S, T)$ denotes the set of edges $\overline{i\,j}$ with $i \in S$ and $j \in T$.

[3] An example is the following 3-regular expander graph: the vertices are the numbers 0 to $p-1$ for some prime p, and each number x is connected to $x+1$,$x-1$ and x^{-1} modulo p (letting $0^{-1} = 0$). The analysis uses some deep results in mathematics (i.e., Selberg's 3/16 Theorem), see Section 11.1.2 in [HLW06].

Note that in this case the bigger ρ is the better the expander. We will often use the shorthand "edge expander" (dropping the prefix "combinatorial"). Also we'll loosely use the term "expander" for any (n, d, ρ)-combinatorial edge expander with ρ a positive constant (independent of n). Using the probabilistic method, one can prove the following theorem: (Exercise 21.11 asks you to prove a slightly weaker version)

Theorem 21.8 *(Existence of expanders) Let $\epsilon > 0$ be any constant. Then there exists $d = d(\epsilon)$ and $N \in \mathbb{N}$ such that for every $n > N$ there exists an $(n, d, 1/2 - \epsilon)$ edge expander.* ♢

Theorem 21.8 is tight in the sense that there is no (n, d, ρ) edge expander for $\rho > 1/2$ (Exercise 21.13). The following theorem relates combinatorial expansion with our previous Definition 21.6

Theorem 21.9 *(Combinatorial vs. algebraic expansion)*

1. *If G is an (n, d, λ)-expander graph then it is an $(n, d, (1-\lambda)/2)$-edge expander.*
2. *If G is an (n, d, ρ) edge expander then its second largest eigenvalue (*without *taking absolute values) is at most $1 - \frac{\rho^2}{2}$. If furthermore G has all self loops then it is an $(n, d, 1-\epsilon)$-expander where $\epsilon = \min\left\{\frac{2}{d}, \frac{\rho^2}{2}\right\}$.*

The condition that G has all the self-loops of Theorem 21.9 is used again to rule out bipartite graphs, which can be very good edge expanders but have one eigenvalue equal to -1 and hence a spectral gap of zero.

21.2.3 Algebraic expansion implies combinatorial expansion.

The first part of Theorem 21.9 follows immediately from the following lemma:

Lemma 21.10 *Let G be an (n, d, λ) graph, S a subset of G's vertices and T its complement. Then*

$$|E(S,T)| \geq (1-\lambda)\frac{d|S||T|}{|S|+|T|}\,.$$ ♢

PROOF: Let $\mathbf{x} \in \mathbb{R}^n$ denote the following vector:

$$\mathbf{x}_i = \begin{cases} +|T| & i \in S \\ -|S| & i \in T \\ 0 & \text{otherwise} \end{cases}\,.$$

Note that $\|\mathbf{x}\|_2^2 = |S||T|^2 + |T||S|^2 = |S||T|(|S|+|T|)$ and $\mathbf{x} \perp \mathbf{1}$.

Let $Z = \sum_{i,j} A_{i,j}(x_i - x_j)^2$. On the one hand $Z = \frac{2}{d}|E(S,T)|(|S|+|T|)^2$, since every edge $\overline{i\,j}$ with $i \in S$ and $j \in T$ appears twice in this sum, each time contributing $\frac{1}{d}(|S|+|T|)^2$ to the total. On the other hand,

$$Z = \sum_{i,j} A_{i,j}x_i^2 - 2\sum A_{i,j}x_i x_j + \sum_{i,j} A_{i,j}x_j^2 = 2\|\mathbf{x}\|_2^2 - 2\langle \mathbf{x}, A\mathbf{x}\rangle$$

(using the fact that A's rows and columns sum up to one). Since $\mathbf{x} \perp \mathbf{1}$ and $\|A\mathbf{x}\|_2 \leq \lambda\|\mathbf{x}\|_2$, we get that

$$\tfrac{1}{d}|E(S,T)|(|S|+|T|)^2 \geq (1-\lambda)\|x\|_2^2\,.$$

Plugging in $\|x\|_2^2 = |S||T|(|S|+|T|)$ completes the proof. ■

Algebraic expansion also allows us to obtain an estimate on the number of edges between not-too-small subsets S and T, even if they are not disjoint:

Lemma 21.11 *(Expander Mixing Lemma) Let $G = (V, E)$ be an (n, d, λ)-expander graph. Let $S, T \subseteq V$, then*

$$\left| |E(S,T)| - \frac{d}{n}|S||T| \right| \leq \lambda d \sqrt{|S||T|}$$

♢

The Mixing Lemma gives a good idea of why expanders are "pseudorandom." In a random d-regular graph, we would expect $|E(S,T)|$ to be about $\frac{d}{n}|S||T|$. The Lemma says that in an expander, $|E(S,T)|$ is close to this expectation for *all* S, T that are sufficiently large. We leave the proof of Lemma 21.11 as Exercise 21.14.

21.2.4 Combinatorial Expansion Implies Algebraic Expansion

We now prove the second part of Theorem 21.9. Let $G = (V, E)$ be an n-vertex d-regular graph such that for every subset $S \subseteq V$ with $|S| \leq n/2$, there are $\rho|S|$ edges between S and $\overline{S} = V \setminus S$, and let A be G's random-walk matrix.

Let λ be the second largest eigenvalue of A (not taking absolute values). We need to prove that $\lambda \leq 1 - \rho^2/2$. By the definition of an eigenvalue there exists a vector $\mathbf{u} \perp \mathbf{1}$ such that $A\mathbf{u} = \lambda\mathbf{u}$. Write $\mathbf{u} = \mathbf{v} + \mathbf{w}$ where $\mathbf{v}$ is equal to $\mathbf{u}$ on the coordinates on which $\mathbf{u}$ is positive and equal to 0 otherwise, and $\mathbf{w}$ is equal to $\mathbf{u}$ on the coordinates on which $\mathbf{u}$ is negative, and equal to 0 otherwise. (Since $\mathbf{u} \perp \mathbf{1}$, both $\mathbf{v}$ and $\mathbf{w}$ are nonzero.) We can assume that $\mathbf{v}$ is nonzero on at most $n/2$ of its coordinates (otherwise take $-\mathbf{u}$ instead of $\mathbf{u}$). Let $Z = \sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2)$. Part 2 of the theorem (except for the "furthermore" clause) follows immediately from the following two claims:

Claim 1: $Z \geq 2\rho\|\mathbf{v}\|_2^2$.

Claim 2: $Z \leq \sqrt{8(1-\lambda)}\|\mathbf{v}\|_2^2$.

Proof of Claim 1: Sort the coordinates of $\mathbf{v}$ so that $\mathbf{v}_1 \geq \mathbf{v}_2 \geq \cdots \geq \mathbf{v}_n$ (with $\mathbf{v}_i = 0$ for $i > n/2$). Then, using $\mathbf{v}_i^2 - \mathbf{v}_j^2 = \sum_{k=i}^{j+1}(\mathbf{v}_k^2 - \mathbf{v}_{k+1}^2)$,

$$Z = \sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2) = 2\sum_{i<j} A_{i,j} \sum_{k=i}^{j-1}(\mathbf{v}_k^2 - \mathbf{v}_{k+1}^2)\,.$$

Note that every term $(\mathbf{v}_k^2 - \mathbf{v}_{k+1}^2)$ appears in this sum once (with a weight of $2/d$) per each edge $\overline{i\,j}$ such that $i \leq k < j$. Since $\mathbf{v}_k = 0$ for $k > n/2$, this means that

$$Z = \tfrac{2}{d}\sum_{k=1}^{n/2} |E(\{1..k\}, \{k+1..n\})|(\mathbf{v}_k^2 - \mathbf{v}_{k+1}^2) \geq \tfrac{2}{d}\sum_{k=1}^{n/2} \rho k(\mathbf{v}_k^2 - \mathbf{v}_{k+1}^2)\,,$$

by G's expansion. But, rearranging the terms (and using the fact that $\mathbf{v}_k = 0$ for $k > n/2$), the last sum is equal to

$$\tfrac{2}{d}d\rho\sum_{k=1}^{n/2} k\mathbf{v}_k^2 - (k-1)\mathbf{v}_k^2 = 2\sum_{k=1}^{n}\mathbf{v}_k^2 = 2\rho\|\mathbf{v}\|_2^2\,.$$

■

Proof of Claim 2: Since $A\mathbf{u} = \lambda\mathbf{u}$ and $\langle \mathbf{v}, \mathbf{w}\rangle = 0$,

$$\langle A\mathbf{v}, \mathbf{v}\rangle + \langle A\mathbf{w}, \mathbf{v}\rangle = \langle A(\mathbf{v}+\mathbf{w}), \mathbf{v}\rangle = \langle A\mathbf{u}, \mathbf{v}\rangle = \langle \lambda(\mathbf{v}+\mathbf{w}), \mathbf{v}\rangle = \lambda\|\mathbf{v}\|_2^2\,.$$

Since $\langle A\mathbf{w}, \mathbf{v}\rangle$ is not positive, $\langle A\mathbf{v}, \mathbf{v}\rangle / \|\mathbf{v}\|_2^2 \geq \lambda$, meaning that

$$1 - \lambda \geq 1 - \frac{\langle A\mathbf{v}, \mathbf{v}\rangle}{\|\mathbf{v}\|_2^2} = \frac{\|\mathbf{v}\|_2^2 - \langle A\mathbf{v}, \mathbf{v}\rangle}{\|\mathbf{v}\|_2^2} = \frac{\sum_{i,j} A_{i,j}(\mathbf{v}_i - \mathbf{v}_j)^2}{2\|\mathbf{v}\|_2^2}\,, \tag{4}$$

where the last equality is due to $\sum_{i,j} A_{i,j}(\mathbf{v}_i - \mathbf{v}_j)^2 = \sum_{i,j} A_{i,j}\mathbf{v}_i^2 - 2\sum_{i,j} A_{i,j}\mathbf{v}_i\mathbf{v}_j + \sum_{i,j} A_{i,j}\mathbf{v}_j^2 = 2\|\mathbf{v}\|_2^2 - 2\langle A\mathbf{v}, \mathbf{v}\rangle$. (We use here the fact that each row and column of A sums to one.)

Multiply both numerator and denominator of the last term in (4) by $\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 + \mathbf{v}_j^2)$. The new numerator is

$$\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i - \mathbf{v}_j)^2\right)\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i + \mathbf{v}_j)^2\right) \geq \left(\sum_{i,j} A_{i,j}(\mathbf{v}_i - \mathbf{v}_j)(\mathbf{v}_i + \mathbf{v}_j)\right)^2 .$$

using the Cauchy-Schwartz inequality.[4] Hence, using $(a-b)(a+b) = a^2 - b^2$,

$$1-\lambda \geq \frac{\left(\sum_{i,j} A_{i,j}(\mathbf{v}_i^2 - \mathbf{v}_j^2)\right)^2}{2\|\mathbf{v}\|_2^2 \sum_{i,j} A_{i,j}(\mathbf{v}_i + \mathbf{v}_j)^2} = \frac{Z^2}{2\|\mathbf{v}\|_2^2\left(\sum_{i,j} A_{i,j}\mathbf{v}_i^2 + 2\sum_{i,j} A_{i,j}\mathbf{v}_i\mathbf{v}_j + \sum_{i,j} A_{i,j}\mathbf{v}_j^2\right)} =$$
$$\frac{Z^2}{2\|\mathbf{v}\|_2^2\left(2\|\mathbf{v}\|_2^2 + 2\langle A\mathbf{v}, \mathbf{v}\rangle\right)} \geq \frac{Z^2}{8\|\mathbf{v}\|_2^4},$$

where the last inequality is due to the fact that A is a symmetric stochastic matrix, and hence $\|A\mathbf{v}\|_2 \leq \|\mathbf{v}\|_2$ for every $\mathbf{v}$, implying that $\langle A\mathbf{v}, \mathbf{v}\rangle \leq \|\mathbf{v}\|_2^2$.

The "furthermore" part is obtained by noticing that adding all the self-loops to a $d-1$-regular graph is equivalent to transforming its random-walk matrix A into the matrix $\frac{d-1}{d}A + \frac{1}{d}I$ where I is the identity matrix. Since A's smallest eigenvalue (not taking absolute values) is at least -1, the new smallest eigenvalue is at least $-\frac{d-1}{d} + \frac{1}{d} = -1 + \frac{2}{d}$.■

21.2.5 Error reduction using expanders.

Before constructing expanders, let us see one application for them in the area of probabilistic algorithms. Recall that in Section 7.4.1 we saw that we can reduce the error of a probabilistic algorithm from, say, 1/3 to $2^{-\Omega(k)}$ by executing it k times independently and taking the majority value. If the algorithm utilized m random coins, this procedure will use $m \cdot k$ random coins, and it seems hard to think of a way to save on randomness. Nonetheless, using expanders we can obtain such error reduction using only $m + O(k)$ random coins.

The idea is simple: take an expander graph G from a strongly explicit family that is an $(M = 2^m, d, 1/10)$-expander graph for some constant d. (Note that we can use graph powering to transform any explicit expander family into an expander family with parameter $\lambda < 1/10$; see also Section 21.3.) Choose a vertex v_1 at random, and take a length $k-1$ long random walk on G to obtain vertices $v_2, \ldots, v_k$ (note that choosing a random neighbor of a vertex requires $O(\log d) = O(1)$ random bits). Invoke the algorithm k times using $v_1, \ldots, v_k$ for the random coins (we identify the set $[M]$ of vertices with the set $\{0,1\}^m$ of possible random coins for the algorithm) and output the majority answer.

To keep things simple, we analyze here only the case of algorithms with one-sided error. For example, consider an **RP** algorithm that will never output "accept" if the input is not in the language, and for inputs in the language will output "accept" with probability 1/2 (the case of a **coRP** algorithm is analogous). For such an algorithm the procedure will output "accept" if the algorithm accepts even on a single set of coins v_i. If the input is not in the language, the procedure will never accept. If the input is in the language, then let $\mathcal{B} \subseteq [M]$ denote the "bad" set of coins on which the algorithms rejects. We know that $|\mathcal{B}| \leq \frac{M}{3}$. Plugging in $\beta = 1/3$ and $\lambda = 1/10$ in the following theorem immediately implies that the probability the above procedure will reject an input in the language is bounded by $2^{-\Omega(k)}$:

[4] The Cauchy-Schwartz inequality says that for every two vectors $\mathbf{x}, \mathbf{y}$, $\langle \mathbf{x}, \mathbf{y}\rangle \leq \|\mathbf{x}\|_2 \|\mathbf{y}\|_2$. Here we index over (i,j), and use $\mathbf{x}_{i,j} = \sqrt{A_{i,j}}(\mathbf{v}_i^2 - \mathbf{v}_j^2)$ and $\mathbf{y}_{i,j} = \sqrt{A_{i,j}}(\mathbf{v}_i^2 + \mathbf{v}_j^2)$.

Theorem 21.12 *(Expander walks)*
Let G be an (n, d, λ) graph, and let $\mathcal{B} \subseteq [n]$ satisfying $|\mathcal{B}| <= \beta n$ for some $\beta \in (0, 1)$. Let $X_1, \ldots, X_k$ be random variables denoting a $k-1$-step random walk in G from X_1, where X_1 is chosen uniformly in $[n]$. Then,

$$\Pr[\forall_{1 \leq i \leq k} X_i \in B] \leq ((1-\lambda)\sqrt{\beta} + \lambda)^{k-1} .$$

Note that if λ and β are both constants smaller than 1 then so is the expression $(1-\lambda)\sqrt{\beta}+\lambda$.

PROOF: For $1 \leq i \leq k$, let B_i be the event that $X_i \in \mathcal{B}$. Note that the probability we're trying to bound is

$$\Pr[\wedge_{i=1}^{k} B_i] = \Pr[B_1] \Pr[B_2|B_1] \cdots \Pr[B_k|B_1, \ldots, B_{k-1}] . \quad (5)$$

Denote by B the linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^n$ that "zeroes out" the coordinates that are not in $\mathcal{B}$. That is, for every $i \in [n]$, $(B\mathbf{u})_i = \mathbf{u}_i$ if $i \in \mathcal{B}$ and $(B\mathbf{u})_i = 0$ otherwise. It's not hard to verify that for every probability vector $\mathbf{p}$ over $[n]$, $B\mathbf{p}$ is a vector whose coordinates sum up to the probability that a vertex i is chosen according to $\mathbf{p}$ is in $\mathcal{B}$. Furthermore, if we normalize the vector $B\mathbf{p}$ to sum up to one, we get the probability vector corresponding to the conditional distribution of p conditioned on the event that the vertex chosen this way is in $\mathcal{B}$.

Thus, if we let $\mathbf{1} = (1/n, \ldots, 1/n)$ denote the uniform distribution over $[n]$ and $\mathbf{p}^i \in \mathbb{R}^N$ be the distribution of X_i conditioned on the events $B_1, \ldots, B_i$, then

$$\mathbf{p}^1 = \frac{1}{\Pr[B_1]} B\mathbf{1}$$
$$\mathbf{p}^2 = \frac{1}{\Pr[B_2|B_1]} \frac{1}{\Pr[B_1]} BAB\mathbf{1}$$

and more generally

$$\mathbf{p}^i = \frac{1}{\Pr[B_i|B_{i-1} \ldots B_1] \cdots \Pr[B_1]} (BA)^{i-1} B\mathbf{1} .$$

Since every probability vector $\mathbf{p}$ satisfies $|\mathbf{p}|_1 = 1$, it follows that the probability on the LHS of (5) is equal to

$$|(\hat{B}A)^{k-1} \hat{B}\mathbf{1}|_1 . \quad (6)$$

Using the relation between the L_1 and L_2 norms (Claim 21.1) we can bound (6) by showing

$$\|(\hat{B}A)^{k-1} B\mathbf{1}\|_2 \leq \frac{((1-\lambda)\sqrt{\beta}+\lambda)^{k-1}}{\sqrt{n}} . \quad (7)$$

To prove (7), we will use the following definition and Lemma:

Definition 21.13 *(Spectral norm)* For every matrix A, the *spectral norm* of A, denoted by $\|A\|$, is defined as the maximum of $\|A\mathbf{v}\|_2$ over all vectors $\mathbf{v}$ satisfying $\|\mathbf{v}\|_2 = 1$. ♢

Exercises 21.5 and 21.6 ask you to prove that the spectral norm of every random-walk matrix is 1, and that for every two n by n matrices A, B, $\|A + B\| \leq \|A\| + \|B\|$ and $\|AB\| \leq \|A\|\|B\|$.

Lemma 21.14 *Let A be a random-walk matrix of an (n, d, λ)-expander graph G. Let J be the random-walk matrix of the n-clique with self loops (i.e., $J_{i,j} = 1/n$ for every i, j). Then*

$$A = (1-\lambda)J + \lambda C \quad (8)$$

where $\|C\| \leq 1$. ♢

Note that for every probability vector $\mathbf{p}$, $J\mathbf{p}$ is the uniform distribution, and so this lemma tells us that in some sense, we can think of a step on a (n, d, λ)-expander graph as going to the uniform distribution with probability $1-\lambda$, and to a different distribution with probability λ. This is of course completely inaacurate, as a step on a d-regular graph will only go the one of the d neighbors of the current vertex, but we'll see that for the purposes of our analysis, the condition (8) will be just as good.[5]

PROOF OF LEMMA 21.14: Indeed, simply define $C = \frac{1}{\lambda}(A - (1-\lambda)J)$. We need to prove $\|C\mathbf{v}\|_2 \leq \|\mathbf{v}\|_2$ for very $\mathbf{v}$. Decompose $\mathbf{v}$ as $\mathbf{v} = \mathbf{u} + \mathbf{w}$ where $\mathbf{u} = \alpha\mathbf{1}$ for some $\alpha \in \mathbb{R}$ and $\mathbf{w} \perp \mathbf{1}$. Since $A\mathbf{1} = \mathbf{1}$ and $J\mathbf{1} = \mathbf{1}$ we get that $C\mathbf{u} = \frac{1}{\lambda}(\mathbf{u} - (1-\lambda)\mathbf{u}) = \mathbf{u}$. Now, let $\mathbf{w}' = A\mathbf{w}$. Then $\|\mathbf{w}'\|_2 \leq \lambda\|\mathbf{w}\|_2$ and, as we saw in the proof of Lemma 21.3, $\mathbf{w}' \perp \mathbf{1}$. In other words, the sum of the coordinates of $\mathbf{w}$ is zero, meaning that $J\mathbf{w} = \mathbf{0}$. We get that $C\mathbf{w} = \frac{1}{\lambda}\mathbf{w}'$. Since $\mathbf{w}' \perp \mathbf{u}$, $\|C\mathbf{v}\|_2^2 = \|\mathbf{u} + \frac{1}{\lambda}\mathbf{w}'\|_2^2 = \|\mathbf{u}\|_2^2 + \|\frac{1}{\lambda}\mathbf{w}'\|_2^2 \leq \|\mathbf{u}\|_2^2 + \|\mathbf{w}\|_2^2 = \|\mathbf{v}\|_2^2$, where we use twice the Pythagorean theorem that for $\mathbf{u} \perp \mathbf{w}$, $\|\mathbf{u}+\mathbf{w}\|_2^2 = \|\mathbf{u}\|_2^2 + \|\mathbf{w}\|_2^2$. ■

Returning to the proof of Theorem 21.12, we can write $BA = B\big((1-\lambda)J + \lambda C\big)$, and hence $\|BA\| \leq (1-\lambda)\|BJ\| + \lambda\|BC\|$. Since J's output is always a vector of the form $\alpha\mathbf{1}$, and it can be easily verified that $\|B\mathbf{1}\|_2 = \sqrt{\frac{\beta n}{n^2}} = \frac{\sqrt{\beta}}{\sqrt{n}} = \sqrt{\beta}\|\mathbf{1}\|_2$, $\|BJ\| = \sqrt{\beta}$. Also, because B is an operation that merely zeros out some parts of its input, $\|B\| \leq 1$ implying that $\|BC\| \leq 1$. Thus, $\|BA\| \leq (1-\lambda)\sqrt{\beta} + \lambda$. This means that $\|(BA)^{k-1}B\mathbf{1}\|_2 \leq ((1-\lambda)\sqrt{\beta} + \lambda)^{k-1}\frac{\sqrt{\beta}}{\sqrt{n}}$, establishing (7). ■

The success of the error reduction procedure for *two-sided error* algorithms is obtained by the following theorem, whose proof we omit (but see Exercise 21.12):

Theorem 21.15 *(Expander Chernoff Bound)*
Let G be an (n, d, λ)-expander graph and $B \subseteq [n]$ with $|B| = \beta N$. Let $X_1, \ldots, X_k$ be random variables denoting a $k-1$-step random walk in G (where X_1 is chosen uniformly). For every $i \in [k]$, define B_i to be 1 if $X_i \in B$ and 0 otherwise. Then, for every $\delta > 0$,

$$\Pr\left[\left|\frac{\sum_{i=1}^{k} B_i}{k} - \beta\right| > \delta\right] < 2e^{(1-\lambda)\delta^2 k/4}$$

21.3 Explicit construction of expander graphs

We now show a construction of a very explicit expander graph family. The main tools in our construction will be several types of graph products. A *graph product* is an operation that takes two graphs G, G' and outputs a graph H. Typically we're interested in the relation between properties of the graphs G, G' and the properties of the resulting graph H. In this section we will mainly be interested in three parameters: the number of vertices (denoted n), the degree (denoted d), and the 2^{nd} largest eigenvalue of the random-walk matrix (denoted λ), and study how different products affect these parameters. We then use these products to obtain a construction of a strongly explicit expander graph family. In the next section we will use the same products to show a *deterministic* logspace algorithm for undirected connectivity.

21.3.1 Rotation maps.

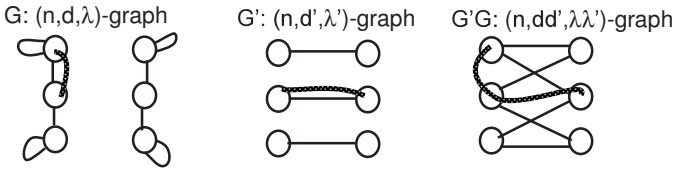
Thus far we usually represented a graph via its adjacency matrix, or as in this chapter, via its random-walk matrix. If the graph is d-regular we can also represent it via its *rotation*

[5] Algebraically, the reason (8) is not equivalent to going to the uniform distribution in each step with probability $1-\lambda$ is that C is not necessarily a stochastic matrix, and may have negative entries.

map. If G is an n-vertex degree-d graph this involves giving a number from 1 to d to each neighbor of each vertex, and then letting a rotation map $\hat{G}$ be a function from $[n] \times [d]$ to $[n] \times [d]$ that maps a pair $\langle v, i \rangle$ to $\langle u, j \rangle$ where u is the ith neighbor of v and v is the jth neighbor of u. Clearly, this map is a permutation (i.e., is one-to-one and onto) of $[n] \times [d]$. The reader may wonder why one should not renumber the neighbors at each node so that $\hat{G}(u, i) = (v, i)$ (i.e., v is the i^{th} neighbor of u iff u is the i^{th} neighbor of v). This is indeed possible but it requires some global computation that will turn out to be too complicated in the scenarios we will be interested in, where the graph is constructed by some space-bounded computation.

Below we will describe graph products, which is usually a way to map two graphs into one. We use whichever graph representation happens to be most natural, but it would be a good exercise for the reader to to work out the equivalent descriptions in the other representations (e.g., in terms of random-walk matrices and rotation maps).

21.3.2 The matrix/path product

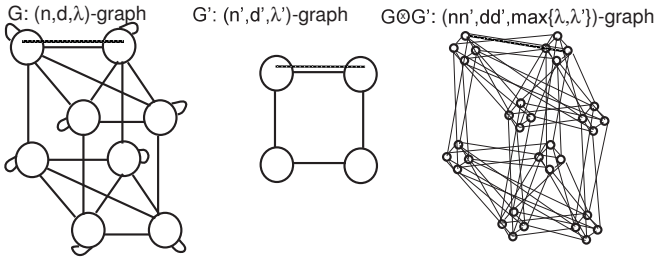


For every two n-vertex graphs G, G' with degrees d, d' and random-walk matrices A, A', the graph $G'G$ is the graph described by the random-walk matrix $A'A$. That is, $G'G$ has an edge (u, v) for every length 2-path from u to v where the first step in the path is taken on an edge of G and the second is on an edge of G'. Note that G has n vertices and degree dd'. Typically, we are interested in the case $G = G'$, where it is called *graph squaring*. More generally, we denote by G^k the graph $G \cdot G \cdots G$ (k times). We have already encountered this case before in Lemma 21.3, and similar analysis yields the following lemma (whose proof we leave as Exercise 21.8):

Lemma 21.16 *(Matrix product improves expansion)* $\lambda(G'G) \leq \lambda(G')\lambda(G')$ ♢

Note that one can easily compute the rotation map of $G'G$ using the rotation maps of G and G'.

21.3.3 The tensor product



Let G and G' be two graphs with n (resp n') vertices and d (resp. d') degree, and let $\hat{G} : [n] \times [d] \to [n] \times [d]$ and $\hat{G'} : [n'] \times [d'] \to [n'] \times [d']$ denote their respective *rotation maps*. The *tensor product* of G and G', denoted $G \otimes G'$, is the graph over nn' vertices and degree dd' whose rotation map $\widehat{G \otimes G'}$ is the permutation over $([n] \times [n']) \times ([d] \times [d'])$ defined as

follows

$$\widehat{G \otimes G'}(\langle u, v\rangle, \langle i, j\rangle) = \langle u', v'\rangle, \langle i', j'\rangle\,,$$

where $(u', i') = \hat{G}(u, i)$ and $(v', j') = \hat{G}'(v, j)$. That is, the vertex set of $G \otimes G'$ consists of pairs of vertices, one from G and the other from G', and taking a the step $\langle i, j\rangle$ on $G \otimes G'$ from the vertex $\langle u, v\rangle$ is akin to taking two independent steps: move to the pair $\langle u', v'\rangle$ where u' is the i^{th} neighbor of u in G and v' is the i^{th} neighbor of v in G'.

In terms of random-walk matrices, the tensor product is also quite easy to describe. If $A = (a_{i,j})$ is the $n \times n$ random-walk matrix of G and $A' = (a'_{i',j'})$ is the $n' \times n'$ random-walk matrix of G', then the random-walk matrix of $G \otimes G'$, denoted as $A \otimes A'$, will be an $nn' \times nn'$ matrix that in the $\langle i, i'\rangle^{th}$ row and the $\langle j, j'\rangle$ column has the value $a_{i,j} \cdot a'_{i',j'}$. That is, $A \otimes A'$ consists of n^2 copies of A', with the $(i, j)^{th}$ copy scaled by $a_{i,j}$:

$$A \otimes A' = \begin{pmatrix} a_{1,1}A' & a_{1,2}A' & \dots & a_{1,n}A' \\ a_{2,1}A' & a_{2,2}A' & \dots & a_{2,n}A' \\ \vdots & & & \vdots \\ a_{n,1}A' & a_{n,2}A' & \dots & a_{n,n}A' \end{pmatrix}$$

The tensor product can also be described in the language of graphs as having a cluster of n' vertices in $G \otimes G'$ for every vertex of G. Now if, u and v are two neighboring vertices in G, we will put a bipartite version of G' between the cluster corresponding to u and the cluster corresponding to v in G. That is, if (i, j) is an edge in G' then there is an edge between the i^{th} vertex in the cluster corresponding to u and the j^{th} vertex in the cluster corresponding to v.

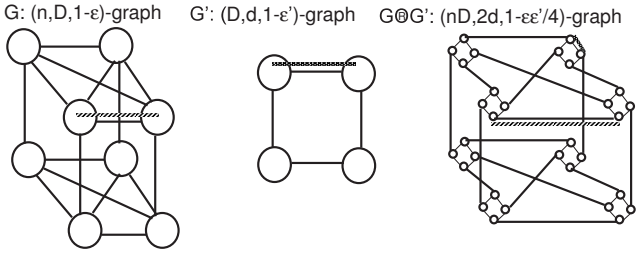
Lemma 21.17 *(Tensor product preserves expansion) Let $\lambda = \lambda(G)$ and $\lambda' = \lambda(G')$ then $\lambda(G \otimes G') \leq \max\{\lambda, \lambda'\}$.* ♢

One intuition for this bound is the following: taking a T step random walk on the graph $G \otimes G'$ is akin to taking two independent random walks on the graphs G and G'. Hence, if both walks converge to the uniform distribution within T steps, then so will the walk on $G \otimes G'$.

Proof of Lemma 21.17: This is immediate from some basic facts about tensor products and eigenvalues (see Exercise 21.22). If $\lambda_1, \dots, \lambda_n$ are the eigenvalues of A (where A is the random-walk matrix of G) and $\lambda'_1, \dots, \lambda'_{n'}$ are the eigenvalues of A' (where A' is the random-walk matrix of G'), then the eigenvalues of $A \otimes A'$ are all numbers of the form $\lambda_i \cdot \lambda'_j$, and hence the largest ones apart from 1 are of the form $1 \cdot \lambda(G')$ or $\lambda(G) \cdot 1$ ■

We note that one can show that $\lambda(G \otimes G') \leq \lambda(G) + \lambda(G')$ without relying on any knowledge of eigenvalues (see Exercise 21.23). Even this weaker bound suffices for our applications.

21.3.4 The replacement product



In both the matrix and tensor products, the degree of the resulting graph is larger than the degree of the input graphs. The following product will enable us to reduce the degree

of one of the graphs. Let G, G' be two graphs such that G has n vertices and degree D, and G' has D vertices and degree d. The *balanced replacement product* (below we use simply *replacement product* for short) of G and G' is denoted by $G \textcircled{r} G'$ is the nn'-vertex $2d$-degree graph obtained as follows:

1. For every vertex u of G, the graph $G \textcircled{r} G'$ has a copy of G' (including both edges and vertices).
2. If u, v are two neighboring vertices in G then we place d parallel edges between the i^{th} vertex in the copy of G' corresponding to u and the j^{th} vertex in the copy of G' corresponding to v, where i is the index of v as a neighbor of u and j is the index of u as a neighbor of v in G. (That is, taking the i^{th} edge out of u leads to v and taking the j^{th} edge out of v leads to u.)

Some texts use the term "replacement product" for the variant of this product that uses only a single edge (as opposed to d parallel edges) in Item 2 above. The addition of parallel edges ensures that a random step from a vertex v in $G \textcircled{r} G'$ will move with probability $1/2$ to a neighbor within the same cluster and with probability $1/2$ to a neighbor outside the cluster.

The replacement product also has a simple description in terms of rotation maps: since $G \textcircled{r} G'$ has nD vertices and $2d$ degree, its rotation map $G \hat{\textcircled{r}} G'$ can be thought of as a permutation over $([n] \times [D]) \times ([d] \times \{0,1\})$ that takes four inputs u, v, i, b where $u \in [n]$, $v \in [D]$, $i \in [d]$ and $b \in \{0,1\}$. If $b = 0$ then it outputs $u, \hat{G'}(v,i), b$ and if $b = 1$ then it outputs $\hat{G}(u,v), i, b$. That is, depending on whether b is equal to 0 or 1, the rotation map either treats v as a vertex of G' or as an edge label of G.

In the language of random-walk matrices the replacement product is described as follows:

$$A \textcircled{r} A' = 1/2\hat{A} + 1/2(I_n \otimes A'), \tag{9}$$

where A, A' denote the random-walk matrices of G and G' respectively, and $\hat{A}$ denotes the permutation matrix corresponding to the rotation map of G. That is, $\hat{A}$ is an $(nD) \times (nD)$ matrix whose $(i,j)^{th}$ column is all zeroes except a single 1 in the $(i',j')^{th}$ place where $(i',j') = \hat{G}(i,j)$.

If $D \gg d$ then the replacement product's degree will be significantly smaller than G's degree. The following Lemma shows that this dramatic degree reduction does not cause too much of a deterioration in the graph's expansion:

Lemma 21.18 *(Expansion of replacement product) If $\lambda(G) \leq 1 - \epsilon$ and $\lambda(H) \leq 1 - \delta$ then $\lambda(G \textcircled{r} H) \leq 1 - \frac{\epsilon\delta^2}{24}$.* ♢

The intuition behind Lemma 21.18 is the following: think of the input graph G as a good expander whose only drawback is that it has a too high degree D. This means that a k step random walk on G' requires $O(k \log D)$ random bits. However, as we saw in Section 21.2.5, sometimes we can use fewer random bits if we use an expander. So a natural idea is to generate the edge labels for the walk by taking a walk using a smaller expander G' that has D vertices and degree $d \ll D$. The definition of $G \textcircled{r} G'$ is motivated by this intuition: a random walk on $G \textcircled{r} G'$ is roughly equivalent to using an expander walk on G' to generate labels for a walk on G. In particular, each step a walk over $G \textcircled{r} G'$ can be thought of as tossing a coin and then, based on its outcome, either taking a a random step on G', or using the current vertex of G' as an edge label to take a step on G. Another way to gain intuition on the replacement product is to solve Exercise 21.24, that analyzes the *combinatorial* (edge) expansion of the resulting graph as a function of the edge expansion of the input graphs.

Proof of Lemma 21.18: It suffices to show that $\lambda(G \textcircled{r} H)^3 \leq 1 - \frac{\epsilon\delta^2}{8}$. Since for every graph F, $\lambda(F^k) = \lambda(F)^k$, we will do so by showing $\lambda((G \textcircled{r} H)^3) \leq 1 - \frac{\epsilon\delta^2}{8}$. Let A be the $n \times n$ random-walk matrix of G (with $\hat{A}$ the $(nD) \times (nD)$ permutation matrix corresponding to the rotation map $\hat{G}$), let B be the $D \times D$ random-walk matrix of H, and let C be the random-walk matrix of $(G \textcircled{r} H)^3$. Then, (9) implies that

$$C = (1/2\hat{A} + 1/2(I_n \otimes B))^3, \tag{10}$$

Now Lemma 21.14 implies that $B = (1-\delta)B' + \delta J_D$ for some matrix B' with norm at most 1 (where J_D is the $D \times D$ all $1/D$ matrix). We plug this into (10), expand all terms and then collect together all the terms except for the one corresponding to $1/2\delta(I_n \otimes J)1/2\hat{A}1/2\delta(I_n \otimes J)$. The reader can verify that all terms correspond to matrices of norm at most 1 and hence (10) becomes

$$C = (1 - \tfrac{\delta^2}{8})C' + \tfrac{\delta^2}{8}(I_n \otimes J_D)\hat{A}(I_n \otimes J_D)\,, \tag{11}$$

where C' is some $(nD) \times (nD)$ matrix of norm at most 1. The lemma will follow from the following claim:

CLAIM: $(I_n \otimes J_D)\hat{A}(I_n \otimes J_D) = A \otimes J_D$

PROOF: Indeed, the left-hand side is the random-walk matrix of the graph on nD vertices on which a step from a vertex (i, j) corresponds to: 1) choosing a random $k \in [D]$ 2) letting i' be the k^{th} neighbor of i in G 3) choosing j' at random in $[D]$ moving to the vertex (i, k). We can equivalently describe this as going to a random neighbor i' of i in G and choosing j' at random in $[D]$, which is the graph corresponding to the matrix $A \otimes J_D$. ■

The claim concludes the proof since $\lambda(A \otimes J_D) \leq \max\{\lambda(A), \lambda(J_D)\} = \max\{\lambda(A), 0\}$. The lemma follows by plugging this into (11) and using the fact that $\lambda(C') \leq 1$ for every matrix of norm at most 1. ■

21.3.5 The actual construction.

We now use the three graph products of described above to show a strongly explicit construction of an expander graph family. That is, we prove the following theorem:

Theorem 21.19 *(Explicit construction of expanders)*
There exists a strongly explicit (λ, d)-expander family for some constants $d \in \mathbb{N}$ and $\lambda < 1$.

Note that using the matrix/graph product, Theorem 21.19 can be improved to yield a strongly explicit (λ, d)-expander family for *every* $\lambda > 0$ (albeit at the expense of allowing d to be an arbitrarily large constant depending on λ.

PROOF: We will start by showing something slightly weaker: a very explicit family of graphs $\{G_k\}$ where G_k is not a graph on k vertices but on roughly c^k vertices for some constant c. That is, rather than showing a family of graphs for *every* size n, we will only show a family of graphs that contains a graph of size n for every n that is a power of c. We will then sketch how the construction can be improved to yield a graph family containing a graph of every size n.

The construction is recursive: we start by a finite size graph G_1 (which we can find using brute force search), and construct the graph G_k from the graph G_{k-1}. On a high level the construction is as follows: each of the three products will serve a different purpose in the construction. The *Tensor product* allows us to take G_{k-1} and increase its number of vertices, at the expense of increasing the degree and possibly some deterioration in the expansion. The *replacement product* allows us to dramatically reduce the degree at the expense of additional deterioration in the expansion. Finally, we use the *Matrix/Path product* to regain the loss in the expansion at the expense of a mild increase in the degree. The actual definition is as follows:

- Let H be a $(D = (2d)^{100}, d, 0.01)$-expander graph, which we can find using brute force search. (We choose d to be a large enough constant that such a graph exists) We let G_1 be a $((2d)^{100}, 2d, 1/2)$-expander graph and G_2 be a $((2d)^{200}, 2d, 1/2)$-expander graphs (again, such graphs can be easily found via brute force).

- For $k > 2$ define

$$G_k = \left(G_{\left\lfloor \frac{k-1}{2} \right\rfloor} \otimes G_{\left\lfloor \frac{k-1}{2} \right\rfloor} \right).$$

We prove the following claim:

CLAIM: For every k, G_k is a $((2d)^{100k}, 2d, 1 - 1/50)$-expander graph. Furthermore, there is a poly(k)-time algorithm that given a label of a vertex i in G_k and an index j in $[2d]$ finds the j^{th} neighbor of i in G_k.

PROOF: We prove the first part by induction. Verify directly that it holds for $k = 1, 2$. For $k > 2$, if we let n_k be the number of vertices of G_k then $n_k = n^2_{\lfloor (k-1)/2 \rfloor}(2d)^{100}$. By induction we assume $n_{\lfloor (k-1)/2 \rfloor} = (2d)^{100\lfloor (k-1)/2 \rfloor}$ which implies that $n_k = (2d)^{100k}$ (using the fact that $2\lfloor (k-1)/2 \rfloor + 1 = k$). It's also easy to verify that G_k has degree $2d$ for every j: if G has degree $2d$ then $G \otimes G$ has degree $(2d)^2$, $(G \otimes G)^{50})$ has degree $(2d)^{100}$ and $(G \otimes G)^{50}) \circledR H$ has degree $(2d)$. The eigenvalue analysis also follows by induction: if $\lambda(G) \leq 1 - 1/50$ then $\lambda(G \otimes G)^{50} \leq 1/e < 1/2$. Hence, by Lemma 21.18, $\lambda((G \otimes G)^{50} \circledR H) \leq 1 - 1/2(0.99)^2/24 \leq 1 - 1/50$.

For the furthermore part, note that there is a natural algorithm to compute the neighborhood function of G_k that makes 100 recursive calls to the neighborhood function of $G_{\lfloor (k-1)/2 \rfloor}$, thus running in time roughly $n^{\log 100}$. ■

The above construction and analysis yields an expander graph family containing an n vertex graph for every n of the form c^k for some constant c. The proof of Theorem 21.19 is completed by observing that one can transform an (n, d, λ)-expander graph to an (n', cd, λ')-expander graph (where $\lambda' < 1$ is a constant depending on λ, d) for any $n/c \leq n' \leq n$ by joining together into a "mega-vertex" sets of at most c vertices (Exercise 21.16). ■

Remark 21.20

The quantitative bounds obtained from the proof of Theorem 21.19 are pretty bad, both in terms of the relation between degree and expansion and the running time (in particular the initial brute force search alone will take more than 2^{100} steps). This is partly because for pedagogical reasons we chose to present this construction in its simplest form, without covering various known optimizations. However, even with these optimizations this construction is not the most efficient known.

There are different known constructions of expanders that are highly practical and efficient (e.g., [LPS86, Mar88]). However, their analysis typically uses deep facts in number theory. Also, the replacement product (and its close cousin, the zig-zag product) have found applications beyond the proof of Theorem 21.15. One such application is the deterministic logspace algorithm for undirected connectivity described in the next section. Another application is a construction of combinatorial vertex expanders with a greater expansion of small sets that what is implied by the parameter λ ([CRVW02], see also Exercise 21.15).

21.4 Deterministic logspace algorithm for undirected connectivity.

This section describes a recent result of Reingold, showing that at least the most famous randomized logspace algorithm, the random walk algorithm for the problem UPATH of s-t-connectivity in undirected graphs (see Chapter 7) can be completely "derandomized."

Theorem 21.21 *(Reingold's Theorem)*
UPATH $\in \mathbf{L}$.

Reingold describes a set of poly(n) walks starting from s such that if s is connected to t then one of the walks is guaranteed to hit t. The *existence* of such a small set of walks

can be shown using the probabilistic method and Corollary 21.5. The point here is that Reingold's enumeration of walks can be carried out deterministically in logspace.

Proof outline. As before we are interested in undirected graphs that may have parallel edges. We restrict our attention to checking connectivity for d-regular graphs for say $d = 4$. This is without loss of generality: if a vertex has degree $d'' < 3$ we add a self-loop of multiplicity to bring the degree up to d, and if the vertex has degree $d' \geq 3$ we can replace it by a cycle of d' vertices, and each of the d' edges that were incident to the old vertex then attach to one of the cycle nodes. Of course, a logspace machine does not have space to store the modified graph, but it can pretend that these modifications have taken place, since it can perform them on the fly whenever it accesses the graph. (To put this more formally, the transformation is implicitly computable in logspace as per Definition 4.16.) In fact, the proof below will perform a series of other local modifications on the graph, each with the property that the logspace algorithm can perform them on the fly.

We start by observing that checking connectivity in *expander* graphs is easy. Specifically, if every connected component in G is an expander, then there is a number $\ell = O(\log n)$ such that if s and t are connected then they are connected with a path of length at most ℓ. Indeed, Lemma 21.3 implies that in every n-vertex regular graph G, the distribution of the ℓ^{th} vertex in a random walk is within $\sqrt{n}\lambda^\ell$ statistical (or L_1) distance from the uniform distribution. In particular this means that if each connected component H of G is an *expander* graph, having $\lambda(H)$ bounded away from 1, then a random walk of length $\ell = O(\log n)$ from a vertex u in H will reach every vertex of H with positive probability.

The idea behind Reingold's algorithm is to transform the graph G (in an implicitly computable in logspace way) to a graph G' such that every connected component in G becomes an expander in G', but two vertices that were not connected will stay unconnected.

21.4.1 The logspace algorithm for connectivity (proof of Theorem 21.21)

By adding more self-loops we may assume that the input graph G is of degree d^{50} for some constant d that is sufficiently large to ensure the existence of a $(d^{50}, d/2, 0.01)$-expander graph H. Since the size of H is constant, we can store all of it in memory using $O(1)$ bits.[6] Let $G_0 = G$ and for $k \geq 1$, define $G_k = (G_{k-1} \textcircled{r} H)^{50}$, where $\textcircled{r}$ denotes the balanced replacement product defined in Section 21.3.4.

If G_{k-1} is an N-vertex graph with degree d^{50}, then $G_{k-1} \textcircled{r} H$ is a $d^{50}N$-vertex graph with degree d and thus $G_k = (G_{k-1} \textcircled{r} H)^{50}$ is a $d^{50}N$-vertex graph with degree d. Note also that if two vertices were connected (resp., disconnected) in G_{k-1}, then they are still connected (resp., disconnected) in G_k. The key observation that the graph $G_{10 \log n}$ is an expander, and therefore an easy instance of UPATH. Specifically, we have:

Claim: For every k, G_k is an $(d^{50k}n, d^{20}, \max\{1-1/20, 2^k/(12n^2)\})$-graph, where n denotes the number of vertices in $G = G_0$.

Proof: Indeed, by Lemmas 21.16 and 21.18, for every $\epsilon < 1/20$ and D-degree graph F, if $\lambda(F) \leq 1-\epsilon$ then $\lambda(F \textcircled{r} H) \leq 1 - \epsilon/25$ and hence $\lambda\left((F \textcircled{r} H)^{50}\right) \leq 1 - 2\epsilon$. By Lemma 21.4, every connected component of G has expansion parameter at most $1 - \frac{1}{12n^2}$ (note that n is at least as large as the number of vertices in the connect component). It follows that for $k = 10 \log n$, in the graph G_k every connected component has expansion parameter at most $\max\{1 - 1/20, 2^k/(12n^2)\} = 1 - 1/20$. ■

Since $G_{10 \log n}$ is an expander, to find whether a pair of vertices s, t are connected in $G_{10 \log n}$ we simply need to enumerate over all paths in $G_{10 \log n}$ that start at s and have length $\ell = O(\log n)$, and see whether any one of these hits t. The catch is of course that the graph provided to our algorithm is G, not $G_{10 \log n}$. A simpler question is whether, given G, our algorithm can perform even a *single* step of a random walk on G_k for $k = 10 \log n$.

[6] We can either use an explicit construction of such a graph or simply find it using an exhaustive search among all graphs of this size.

Specifically, given a description of a vertex s in G_k and an index $i \in [d^{20}]$, it has to compute the i^{th} neighbor of s in G_k using only logarithmic space. It is easy to see that if we can perform this single step in logarithmic space, then we can just as easily perform ℓ steps as well by repeating the single step again and again while keeping a counter, and reusing the same space to compute each step.

The graph G_k is equal to $(G_{k-1} \textcircled{R} H)^{50}$ and thus it suffices to show that we can take a single step in the graph $G_{k-1} \textcircled{R} H$ in logspace (we can then repeat the same process for 50 times). Now by the definition of the replacement product, a vertex in $G_{k-1} \textcircled{R} H$ is represented by a pair $\langle u, v \rangle$ where u is a vertex of G_{k-1} and v is a vertex of H. The index of a neighbor of $\langle u, v \rangle$ is represented by a pair $\langle b, i \rangle$ where $b \in \{0,1\}$ and $i \in [d/2]$. If $b = 0$ then the $\langle b, i \rangle^{th}$ neighbor of $\langle u, v \rangle$ is $\langle u, v' \rangle$ where v' is the i^{th} neighbor of v' in H. If $b = 1$ then the $\langle b, i \rangle^{th}$ neighbor of $\langle u, v \rangle$ is the pair $\langle u', v' \rangle$ denoting the result of applying G_{k-1}'s rotation map to $\langle u, v \rangle$. (That is, u' is the v^{th} neighbor of u in G_{k-1}, and v' is the index of u as a neighbor of u' in G_{k-1}.) This description already implies an obvious recursive algorithm to compute the rotation map of G_k. Letting s_k denotes the space needed to compute a rotation map of G_k by this algorithm, we see that s_k satisfies the equation $s_k = s_{k-1} + O(1)$, implying that $s_{10 \log n} = O(\log n)$.[7] ■

21.5 Weak Random Sources and Extractors

Suppose, that despite any philosophical difficulties, we are happy with probabilistic algorithms, and see no need to "derandomize" them, especially at the expense of some unproven assumptions. We still need to tackle the fact that real world sources of randomness and unpredictability rarely, if ever, behave as a sequence of perfectly uncorrelated and unbiased coin tosses. Can we still execute probabilistic algorithms using real-world "weakly random" sources?

21.5.1 Min Entropy

For starters, we try to define what we could mean by a weakly random source. Historically speaking, several definitions were proposed, which are recalled in Example 21.23. The following definition (due to D. Zuckerman) encompasses all previous definitions.

Definition 21.22 Let X be a random variable. The *min entropy* of X, denoted by $H_\infty(X)$, is the largest real number k such that $\Pr[X = x] \leq 2^{-k}$ for every x in the range of X.

If X is a distribution over $\{0,1\}^n$ with $H_\infty(X) \geq k$ then it is called an (n,k)-*source*. ♢

It is not hard to see that if X is a random variable over $\{0,1\}^n$ then $H_\infty(X) \leq n$ with $H_\infty(X) = n$ if and only if X is distributed according to the uniform distribution U_n. Our goal in this section is to be able to execute probabilistic algorithms given access to a distribution X with $H_\infty(X)$ as small as possible. It can be shown that min entropy is a *minimal requirement* in the sense that a general simulation of a probabilistic algorithm that uses k random bits requires access to a distribution X that is (close to) having min entropy at least k (see Exercise 21.18).

Example 21.23

We will now see that min entropy is a pretty general notion, and can allow us to model many other models of "imperfectly random" sources. Here are some examples for distributions X over $\{0,1\}^n$.

[7] When implementing the algorithm one needs to take care *not* to make a copy of the input when invoking the recursive procedure, but rather have all procedure operate on a globally accessible memory that contains the index k and the vertex and edge labels; otherwise we'd get an $O(\log n \log\log n)$-space algorithm. For more details see the original paper [Rei05] or [Gol08, Section 5.2.4].

- (von Neumann's model: biased coins) X is composed of n independent coin tosses, each outputting 1 with probability $\delta < 1/2$ and 0 with probability $1-\delta$. It is easily checked that[8] $H_\infty(X) = \log(1/(1-\delta))n$.
- (Santha-Vazirani sources) X has the property that for every $i \in [n]$, and every string $x \in \{0,1\}^{i-1}$, conditioned on $X_1 = x_1, \ldots, X_{i-1} = x_{i-1}$ it holds that both $\Pr[X_i = 0]$ and $\Pr[X_i = 1]$ are between δ and $1-\delta$. This generalizes von Neumann's model and can model sources such as stock market fluctuations, where current measurements do have some limited dependence on the previous history. It is easily checked that $H_\infty(X) \geq \log(1/(1-\delta))n$.
- (Bit fixing and generalized bit fixing sources) In a *bit-fixing* source, there is a subset $S \subseteq [n]$ with $|S| = k$ such that X's bits in the coordinates given by S are uniformly distributed over $\{0,1\}^k$, and X's bits in the coordinates given by $[n] \setminus S$ is a fixed string (say the all-zeros string). Then $H_\infty(X) = k$. The same holds if X's projection to $[n] \setminus S$ is a fixed deterministic function of its projection to S, in which case we say that X is a *generalized bit-fixing source*. For example, if the bits in the odd positions of X are independent and uniform and for every even position $2i$, $X_{2i} = X_{2i-1}$ then $H_\infty(X) = \lceil \frac{n}{2} \rceil$. This may model a scenario where we measure some real world data at too high a rate (think of measuring every second a physical event that changes only every minute).
- (Linear subspaces) If X is the uniform distribution over a linear subspace of $\mathrm{GF}(2)^n$ of dimension k, then $H_\infty(X) = k$. (In this case X is actually a generalized bit-fixing source — can you see why?)
- (Uniform over subset) If X is the uniform distribution over a set $S \subseteq \{0,1\}^n$ with $|S| = 2^k$ then $H_\infty(X) = k$. As we will see, this is a very general case that "essentially captures" all distributions X with $H_\infty(X) = k$.

21.5.2 Statistical distance

Next we formalize what it means to *extract* random —more precisely, almost random— bits from an (n,k) source. We will use the notion of *statistical distance* (see Section A.2.6 in the appendix) to qualify when two distributions are close to one another. Recall that if X and Y are two distributions over some domain Ω then the *statistical distance* between X and Y, denoted by $\Delta(X,Y)$ is equal to

$$\max_{f:\Omega\to\{0,1\}} |\mathsf{E}[f(X)] - \mathsf{E}[f(Y)]| \ . \tag{12}$$

It is also known that $\Delta(X,Y) = 1/2|\mathbf{x} - \mathbf{y}|_1$, where $\mathbf{x}$ and $\mathbf{y}$ are the vectors in $\mathbb{R}^\Omega$ that represent the distributions X and Y respectively. For any $\epsilon > 0$, we say that two distribution X and Y are ϵ-*close*denoted $X \approx_\epsilon Y$, if $\Delta(X,Y) \leq \epsilon$.

21.5.3 Definition of randomness extractors

We can now define randomness extractors - these are functions that transform an (n,k) source into an almost uniform distribution. The extractor uses a small number of additional truly random bits, called a *seed* and denoted by d in the definition below.

[8] In fact, as n grows X is close to a distribution with min-entropy $H(\delta)n$ where H is the Shannon entropy function defined as $H(\delta) = \delta \log \frac{1}{\delta} + (1-\delta)\log\frac{1}{1-\delta}$. The same holds for Santha-Vazirani sources defined below. See [DFR+07] for this and more general results of this form.

Definition 21.24 *(Randomness extractors)*
A function $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a (k,ϵ) extractor if for any (n,k)-source X, the distribution $\mathsf{Ext}(X, U_d)$ is ϵ-close to U_m. (For every ℓ, U_ℓ denotes the uniform distribution over $\{0,1\}^\ell$.)

Why an additional input? Our stated motivation for extractors is to execute probabilistic algorithms without access to perfect unbiased coins. Yet, it seems that an extractor is not sufficient for this task, as we only guarantee that its output is close to uniform if it is given an an additional *seed* that is uniformly distributed. We have two answers to this objection. First, note that the requirement of an additional input is necessary: for every function $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ and every $k \leq n-1$ there exists an (n,k)-source X such that the first bit of $\mathsf{Ext}(X)$ is constant (i.e, is equal to some value $b \in \{0,1\}$ with probability 1), and so is at least of statistical distance $1/2$ from the uniform distribution (Exercise 21.17). Second, if the length t of the second input is sufficiently short (e.g., $t = O(\log n)$) then, for the purposes of simulating probabilistic algorithms, we can do without any access to true random coins, by enumerating over all the 2^t possible inputs. Clearly, d has to be somewhat short for the extractor to be non-trivial. The completely trivial case is when $d \geq m$, in which case the extractor can simply ignore its first input and output the seed!

21.5.4 Existence proof for extractors.

It turns out that at least if we ignore issues of computational efficiency, very good extractors exist:

Theorem 21.25 *For every $k, n \in \mathbb{N}$ and $\epsilon > 0$, there exists a (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^k$ with $d = \log n + 2\log(1/\epsilon) + O(1)$* ♢

PROOF: Call an (n,k) source X *flat* if X is the uniform distribution over a 2^k-sized subset of $\{0,1\}^n$. In Exercise 19.7 it is shown that every (n,k) source can be expressed as a convex combination of flat (n,k)-sources. Because the statistical distance of a convex combination of distributions $Y_1, \ldots, Y_N$ from a distribution U is at most the maximum of $\Delta(Y_i, X)$ (Exercise 21.19), it suffices to show a function Ext such that $\mathsf{Ext}(X, U_d)$ is close to the uniform distribution when X is an (n,k)-flat source.

We will prove the existence of such an extractor by the probabilistic method, choosing Ext as a random function from $\{0,1\}^n \times \{0,1\}^d \to \{0,1\}^k$. Let X be an (n,k) flat source and let f be a function from $\{0,1\}^k \to \{0,1\}$. If we choose Ext at random then the expectation $\mathsf{E}[f(\mathsf{Ext}(X, U_d))]$ is obtained by evaluating f on $2^k \times 2^d$ random points, and hence by the Chernoff bound the probability that this expectation deviates from $\mathsf{E}[f(U_k)]$ by more than ϵ is bounded by $2^{-2^{k+d}/4\epsilon^2}$. This means that if $d > \log n + 2\log(1/e) + 3$ then this probability is bounded by $2^{-2n(2^k)}$. But the number of flat distributions is at most $(2^n)^{2^k}$ and the number of functions from $\{0,1\}^k \to \{0,1\}$ is 2^{2^k} and hence the union bound implies that there is a choice of Ext guaranteeing

$$|\mathsf{E}[f(\mathsf{Ext}(X, U_d))] - \mathsf{E}[f(U_k)]| < \epsilon$$

for *every* (n,k)-flat source and function $f : \{0,1\}^k \to \{0,1\}$. In other words, $\mathsf{Ext}(X, U_d)$ is ϵ-close to U_k for every (n,k)-flat source and hence for every (n,k)-source. ■

This extractor is optimal in the sense that there is an absolute constant c such that every (k,ϵ) extractor that is non-trivial (has output longer than seed length and $\epsilon < 1/2$) must satisfy $d \geq \log(n-k) + 2\log(1/\epsilon) - c$ [NZ93, RTS97].

21.5.5 Extractors based on hash functions

The non-explicit extractor of Theorem 21.25 is not very useful: for most applications we need *explicit* extractors— namely extractors computable in polynomial time. One such explicit extractor (though with a long seed length) can be obtained using pairwise independent hash functions.

Recall (Section 8.2.2) that a collection $\mathcal{H}$ of functions from $\{0,1\}^n$ to $\{0,1\}^m$ is *pairwise independent* if for every $x \neq x'$ in $\{0,1\}^n$ and $y, y' \in \{0,1\}^m$, the probability that $h(x) = y$ and $h(x') = y'$ for a random $h \in_R \mathcal{H}$ is 2^{-2m}. There are known construction of such collections where each function h can be described by a string of length $n+m$ (we abuse notation and call this string also h). Choosing a random function from the collection is done by choosing a random string in $\{0,1\}^{n+m}$. The next famous lemma shows that with an appropriate setting of parameters, the map $x, h \mapsto h(x) \circ h$ (where $\circ$ denotes concatenation) is an extractor. This is not a superb extractor in terms of parameter values but it is useful in many settings.

Lemma 21.26 *(Leftover hash lemma* [BBR88, ILL89]*) Let $m = k - 2\log(1/\epsilon)$, then for every (n,k) source X,*

$$\Delta(H(X) \circ H, U_n \circ H) < \epsilon\,,$$

where H denotes a randomly chosen (description of) function in a pairwise independent hash function collection from $\{0,1\}^n$ to $\{0,1\}^m$. ♢

PROOF: We study the *collision probability* of $H(X) \circ H$, where we identify H with U_ℓ where $\ell = n + m$ is the length of description of the hash function. That is, the probability that $h(x) \circ h = h'(x') \circ h'$ for random $h, h' \in_R \mathcal{H}$ and $x, x' \in_R X$. This is bounded by the probability that $h = h'$ (which is equal to $2^{-\ell}$) times the probability that $h(x) = h(x')$. The latter is bounded by 2^{-k} (a bound on the probability that $x = x'$ implies by the fact that X is an (n,k)-source) plus 2^{-m} (the probability that $h(x) = h(x')$ for a random $h \in_R \mathcal{H}$ and $x \neq x'$). Thus the collision probability of $(H(X), H)$ is at most $2^{-\ell}(2^{-k} + 2^{-m}) = 2^{-(\ell+m)} + 2^{-\ell-k}$.

Now, treat this distribution as a probability vector $\mathbf{p} \in \mathbb{R}^{2^{\ell+m}}$. Then the collision probability is precisely the L_2-norm of $\mathbf{p}$ squared. We can write $\mathbf{p} = \mathbf{1} + \mathbf{w}$ where $\mathbf{1}$ is the probability vector corresponding to the distribution $U_n \circ H = U_{n+\ell}$ and $\mathbf{w}$ is orthogonal to $\mathbf{1}$. (For a general vector $\mathbf{p}$ we'd only be able to write $\mathbf{p} = \alpha\mathbf{1} + \mathbf{w}$ for some $\alpha \in \mathbb{R}$, but since $\mathbf{p}$ is a probability vector it must hold that $\alpha = 1$, as otherwise the entries of the righthand side will not sum up to one.) Thus by the Pythagorean Theorem $\|\mathbf{p}\|_2^2 = \|\mathbf{u}\|_2^2 + \|\mathbf{w}\|_2^2$, and since $\|\mathbf{u}\|_2^2 = 2^{-\ell-m}$ we get that

$$\|\mathbf{w}\|_2^2 = \|\mathbf{p} - \mathbf{1}\|_2^2 \leq 2^{-\ell-m}\,.$$

Using the relation between the L_1 and L_2 norms (Claim 21.1), we see that

$$\Delta(H(X) \circ H, U_{\ell+m}) = 1/2|\mathbf{p} - \mathbf{1}|_1 \leq 1/2 2^{(m+\ell)/2}\|\mathbf{v} - \mathbf{1}\|_2 \leq 2^{k/2+\ell/2-\log(1/\epsilon)} 2^{-k/2-\ell/2} < \epsilon\,.$$

■

21.5.6 Extractors based on random walks on expanders

We can also construct explicit extractors using expander graphs:

Lemma 21.27 *Let $\epsilon > 0$. For every n and $k \leq n$ there exists an explicit (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^n$, where $t = O(n - k + \log 1/\epsilon)$.* ♢

PROOF: Suppose X is an (n,k)-source and we are given a sample a from it. Let G be a $(2^n, d, 1/2)$-expander graph for some constant d (see Definition 21.6 and Theorem 21.19).

Let z be a truly random seed of length $t = \log d(n/2 - k/2 + \log 1/\epsilon + 1) = O(n - k + \log 1/\epsilon)$. We interpret z as a random walk in G of length $\ell = n/2 - k/2 + \log 1/\epsilon + 1$ starting from the node whose label is a. (That is, we think of z as ℓ labels in $[d]$ specifying the steps taken in the walk.) The output $\mathsf{Ext}(a, z)$ of the extractor is the label of the final node on the walk.

Following the proof of Lemma 21.3 (see Equation (1)) we see that, letting $\mathbf{p}$ denote the probability vector corresponding to X and A the random-walk matrix of G,

$$\|A^\ell \mathbf{p} - \mathbf{1}\|_2 \leq 2^{-\ell} \|\mathbf{p} - \mathbf{1}\|_2 .$$

But since X is an (n, k) source, $\|\mathbf{p}\|_2^2$ (which is equal to the collision probability of X) is at most 2^{-k}, and hence in particular $\|\mathbf{p} - \mathbf{1}\|_2 \leq \|\mathbf{p}\|_2 + \|\mathbf{1}\|_2 \leq 2^{-k/2} + 2^{-n/2} \leq 2^{-k/2+1}$. Thus for our choice of ℓ,

$$\|A^\ell \mathbf{p} - \mathbf{1}\|_2 \leq 2^{-n/2+k/2-\log(1/\epsilon)+1} 2^{-k/2+1} \leq \epsilon 2^{-n/2} ,$$

which completes the proof using the relation between the L_1 and L_2 norms. ■

21.5.7 Extractors from pseudorandom generators

For many years explicit constructions of randomness extractors fell quite a bit behind the parameters achieved by the optimal non-explicit construction of Theorem 21.25. For example, we did not have explicit extractors that allowed us to run any randomized polynomial time algorithm using $\sim k$ bits using an (n, k) source where $k = n^\epsilon$ for arbitrarily small constant $\epsilon > 0$. (Generally, the smaller k is as a function of n, the harder the problem of constructing extractors; intuitively if $n \gg k$ then it's harder to "distill" the k bits of randomness that are hidden in the n-bit input.) To realize this goal, one should try to design an extractor that uses a seed of $O(\log n)$ bits to extract from an (n, n^ϵ)-source at least a polynomial number of bits (i.e., at least n^δ bits for some $\delta > 0$).[9] In 1999 Trevisan showed how to do this using an improved extractor construction. But more interesting than the result itself was Trevisan's idea: he showed that *pseudorandom generators* such as the ones we've seen in Chapters 20 and 19, when viewed in the right way, are in fact also randomness extractors. This was very surprising, since these pseudorandom generators rely on *hardness assumptions* (such as the existence of a function in **E** with high circuit complexity). Thus it would seem that these generators will not be useful in the context of randomness extractors, where we are looking for constructions with *unconditional* analysis and are not willing to make any unproven assumptions.

But thinking further, we realize that the above-mentioned difference between the two notions arises due to the type of "adversary" or "distinguisher" they have to work against. For a generator, the set of possible adversaries is the class of computationally limited algorithms (i.e., those that can be computed by circuits of some prescribed size). For an extractor, on the other hand, the set of adversaries is the set of all Boolean functions. The reason is that an extractor needs to produce a distribution $\mathcal{D}$ on $\{0,1\}^m$ whose statistical difference from U_m is at most ϵ, meaning that $|\Pr_{x \in \mathcal{D}}[D(x) = 1] - \Pr_{x \in U_m}[D(x) = 1]| \leq \epsilon$ for every function $D : \{0,1\}^m \to \{0,1\}$.

Trevisan noticed further that while we normally think of a pseudorandom generator G as having only one input, we can think of it as a function that takes two inputs: a short seed and the truth table of a candidate hard function f. While our theorems state that the pseudorandom generator works if f is a hard function, the proofs of these theorems are actually *constructive*: they transform a distinguisher D that distinguishes between the generator's output and a random string into a small circuit A that computes the function f. This circuit A uses the distinguisher D as a *black-box*. Therefore we can apply this transformation even when the distinguisher D is an arbitrary function that is not necessarily computable by a small circuit. This is the heart of Trevisan's argument.

[9] The work of Ta-Shma [TS96] did come close to this goal, achieving such an extractor with slightly super-logarithmic seed length.

Concretely, to make this all work we will need the stronger constructions of pseudorandom generators (e.g. of Theorem 20.7) that start with functions with high *worst-case* complexity. If there is a distinguisher D that distinguishes the output of such a generator from the uniform distribution, then the proof of correctness of the generator gives a way to compute the candidate hard function f on *every* input. Formally, we have the following theorem: (Below G^f refers to the algorithm G using f as a black box.)

Theorem 21.28 *(Constructive version of Theorem 20.7)*
For every time-constructible function $S : \mathbb{N} \to \mathbb{N}$ (the "security parameter"), there is a constant c and algorithms G and R satisfying the following:

- *On input a function $f : \{0,1\}^\ell \to \{0,1\}$ and a string $z \in \{0,1\}^{c\ell}$, algorithm G runs in $2^{O(\ell)}$ time and outputs a string $G^f(z)$ of length $m = S(\ell)^{1/c}$.*
- *If $D : \{0,1\}^m \to \{0,1\}$ is a function such that $\left|\mathsf{E}[D(G^f(U_{c\ell}))] - \mathsf{E}[D(U_m)]\right| > 1/10$ then there is an advice string a of length at most $S(\ell)^{1/4}$ such that on every input x, $R^D(a,x) = f(x)$ and furthermore R runs in time at most $S(\ell)^{1/4}$.*

The algorithm R mentioned in the theorem is just the reduction that is implicit in the proof of correctness of the pseudorandom generator in Chapter 20.

The following is Trevisan's extractor construction. Let G be as in Theorem 21.28. Let X be an (n,k)-source. Assume without loss of generality that n is a power of 2, and $n = 2^\ell$. Let $S(\ell)$, the "security parameter", stand for k. Given any string f from the source and the seed $z \in \{0,1\}^{c \log n}$, the extractor interprets f as a function from $\{0,1\}^\ell$ to $\{0,1\}$ and outputs

$$\mathsf{Ext}(f,z) = G^f(z)\,. \tag{13}$$

Thus given a string of length n and a seed of size $c \log n$, Ext produces $S(\ell)^{1/c} = k^{1/c}$ bits. Let us show that Ext is an extractor.

Claim 21.29 *For every k, n, the function Ext defined in (13) is a $(k, 1/5)$-extractor.* ♢

Proof: Suppose otherwise, that there is a (k,n)-source X and a Boolean function D that distinguishes between $\mathsf{Ext}(X, U_{c\ell})$ and U_m with bias at least $1/5$, where $m = S(\ell)^{1/c}$. Then, with probability at least $1/10$ over $f \in_R X$, function D distinguishes between $G^f(U_{c\ell})$ and U_m with bias at least $1/10$. Let's call an f for which this happens "bad". Note that for every bad f there exists an advice string $a \in \{0,1\}^{k^{1/4}}$ such that f is computed by the algorithm $x \mapsto R^D(a,x)$. Since R^D is a deterministic algorithm, this means that the number of bad f's is at most the number of choices for a, which is $2^{k^{1/4}}$. But since X is a k-source, it assigns probability not more than 2^{-k} to any particular string. Hence the probability of a random f being bad is at most $2^{k^{1/4}} 2^{-k} \ll 1/10$, and we've arrived at a contradiction to the assumption that D is a good distinguisher. ■

Remark 21.30
Readers mystified by this construction should try to look inside the generator G to get a better understanding. The extractor Ext turns out to do be very simple. Given a string $f \in \{0,1\}^n$ from the weak random source, the extractor first applies an error-correcting code (specifically, one that is *list decodable*) to f to get a string $\hat{f} \in \{0,1\}^{\mathrm{poly}(n)}$. Intuitively speaking, this has the effect of "smearing out"the randomness over the entire string. The extractor then selects a subset of the coordinates of $\hat{f}$ using the construction of the Nisan-Wigderson generator (see Section 20.2.2). That is, treating $\hat{f}$ as a Boolean function on $s = O(\log n)$ bits, we use a seed z of size $t = O(s)$ and output $\hat{f}(z_{I_1}) \circ \cdots \circ \hat{f}(z_{I_m})$, where $I_1, \ldots, I_m$ are s-sized subsets of $[t]$ that form a combinatorial design (see Definition 20.13).

21.6 Pseudorandom generators for space bounded computation

We now show how extractors can be used to obtain a pseudorandom generator for space-bounded randomized computation, which allows randomized logspace computations to be run with $O(\log^2 n)$ random bits. We stress that this generator does not use any unproven assumptions.

The goal here will be to derandomize randomized logspace computations, in other words, classes such as **BPL** and **RL**. Recall from Chapter 4 the notion of a *configuration graph* for a space-bounded TM. If we fix an input of size n for a logspace machine, then the configuration graph has size $\text{poly}(n)$. If the logspace machine is randomized, then it uses random coin tosses to make transitions within the configuration graph (i.e., each configuration has two outgoing edges, and each is taken with probability 1/2). To derandomize this computation we will replace the random string used by the logspace machine with the output of a "pseudorandom generator" (albeit one tailormade for fooling logspace computations) and show that the logspace machine cannot "tell the difference" (i.e., the probability it ends up in an accepting state at the end is not very different).

Theorem 21.31 *(Nisan's pseudorandom generator* [Nis90]*)*
For every d there is a $c > 0$ and a poly(n)-time computable function $g : \{0,1\}^{c \log^2 n} \to \{0,1\}^{n^d}$ (the "pseudorandom generator") such that for every space-bounded machine M that has a configuration graph of size $\leq n^d$ on inputs of size n:

$$\left| \Pr_{r \in \{0,1\}^{n^d}} [M(x,r) = 1] - \Pr_{z \in \{0,1\}^{c \log^2 n}} [M(x, g(z)) = 1] \right| < \frac{1}{10}. \quad (14)$$

By trying all possible choices for the $O(\log^2 n)$-bit input for the generator g in Nisan's theorem, we can simulate every algorithm in **BPL** in $O(\log^2 n)$ space. Note that Savitch's theorem (Theorem 4.14) also implies that **BPL** $\subseteq$ **SPACE**$(\log^2 n)$ but it doesn't yield such a pseudorandom generator. In fact Theorem 21.31 can be strengthened to show that **BPL** can be decided using simultaneously polynomial time and space $O(\log^2 n)$, though we will not prove it here. Saks and Zhou [SZ95] improved Nisan's ideas to show that **BPL** $\subseteq$ **SPACE**$(\log^{1.5} n)$, which leads many experts to conjecture that **BPL** $=$ **L** (i.e., randomness does not help logspace computations at all). Indeed, we've seen in Section 21.4 that the famous random-walk algorithm for undirected connectivity can be derandomized in logspace.

The main intuition behind Nisan's construction —and also the conjecture **BPL** $=$ **L**— is that the logspace machine has one-way access to the random string and only $O(\log n)$ bits of memory. So it can only "remember" $O(\log n)$ of the random bits it has seen. To exploit this we will use the following simple lemma, which shows how to recycle a random string about which only a little information is known.

Lemma 21.32 *(Recycling lemma) Let $f : \{0,1\}^n \to \{0,1\}^s$ be any function and* Ext $: \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ *be a $(k, \epsilon/2)$-extractor, where $k = n - (s+1) - \log \frac{1}{\epsilon}$. Then,*

$$\Delta\left(f(X) \circ U_m \,,\; f(X) \circ \mathsf{Ext}(X, U_t)\right) < \epsilon\,,$$

where X is a random variable distributed uniformly in $\{0,1\}^n$. ♢

To understand why we call it the Recycling Lemma, focus on the case $s \ll n$ and $n = m$. Suppose we use a random string X of length n to produce $f(X)$. Since $f(X)$ has length $s \ll n$, typically each string in $\{0,1\}^s$ will have many preimages under f. Thus anybody looking at $f(X)$ has only very little information about X. More formally, for every fixed choice of $f(X)$, the set of X that map to this value can be viewed as a weak random source. The Lemma says that applying an appropriate extractor (whose random seed z can have length as small as $t = O(s + \log(1/\epsilon))$ if we use Lemma 21.27) on X we can get a new m-bit string $\mathsf{Ext}(X, z)$ that looks essentially random, even to somebody who knows $f(X)$.

PROOF: For $v \in \{0,1\}^s$ we denote by X_v the random variable that is uniformly distributed over the set $f^{-1}(v)$. Then we can express $\Delta(f(X) \circ W, f(X) \circ \mathsf{Ext}(X,z))$ as

$$= \frac{1}{2} \sum_{v,w} \left| \Pr[f(X) = v \wedge W = w] - \Pr_z[f(X) = v \wedge \mathsf{Ext}(X,z) = w] \right|$$

$$= \sum_v \Pr[f(X) = v] \cdot \Delta(W, \mathsf{Ext}(X_v, z)) \tag{15}$$

Let $V = \left\{ v : \Pr[f(X) = v] \geq \epsilon/2^{s+1} \right\}$. If $v \in V$, then we can view X_v as a (n,k)-source, where $k \geq n - (s+1) - \log \frac{1}{\epsilon}$. Thus by definition of an extractor, $\mathsf{Ext}(X_v, r) \approx_{\epsilon/2} W$ and hence the contributions from $v \in V$ sum to at most $\epsilon/2$. The contributions from $v \notin V$ are upperbounded by $\sum_{v \notin V} \Pr[f(X) = v] \leq 2^s \times \frac{\epsilon}{2^{s+1}} = \epsilon/2$. The lemma follows. ■

Now we describe how the Recycling Lemma is useful in Nisan's construction. Let M be a logspace machine. Fix an input of size n. Then for some $d \geq 1$ the graph of all configurations of M on this input has $\leq n^d$ configurations and runs in time $L \leq n^d$. Assume without loss of generality —since unneeded random bits can always be ignored— that M uses 1 random bit at each step. Assume also (by giving M a separate worktape that maintains a time counter), that the configuration graph is leveled: it has L levels, with level i containing configurations obtainable at time i. The first level contains only the start node and the last level contains two nodes, "accept" and "reject;" every other level has $W = n^d$ nodes. Each level i node has two outgoing edges to level $i+1$ nodes and the machine's computation at this node involves using the next bit in the random string to pick one of these two outgoing edges. We sometimes call L the *length* of the configuration graph and W the *width*.

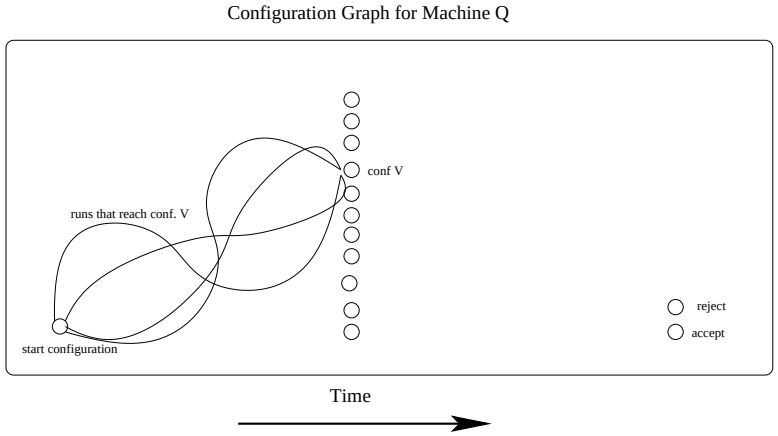


Figure 21.2 Configuration graph for machine M

For simplicity we first describe how to reduce the number of random bits by a factor 2. Think of the L steps of the computation as divided in two halves, each consuming $L/2$ random bits. Suppose we use some random string X of length $L/2$ to run the first half, and the machine is now at node v in the middle level. The only information known about X at this point is the index of v, which is a string of length $d \log n$. We may thus view the first half of the branching program as a (deterministic) function that maps $\{0,1\}^{L/2}$ bits to $\{0,1\}^{d \log n}$ bits. The Recycling Lemma allows us to use a random seed of length $O(\log n)$ to recycle X to get an almost-random string $\mathsf{Ext}(X,z)$ of length $L/2$, which can be used in the second half of the computation. Thus we can run L steps of computation using $L/2 + O(\log n)$ bits, a saving of almost a factor 2. Using a similar idea recursively, Nisan's generator runs L steps using $O(\log n \log L)$ random bits.

Now we formally define Nisan's generator.

Definition 21.33 *(Nisan's generator)* For some $r > 0$ let $\mathsf{Ext}_k : \{0,1\}^{kr} \times \{0,1\}^r \to \{0,1\}^{kr}$ be an extractor function for each $k \geq 0$. For every integer $k \geq 0$ the associated Nisan generator $G_k : \{0,1\}^{kr} \to \{0,1\}^{2^k}$ is defined recursively as (where $|a| = (k-1)r, |z| = r$)

$$G_k(a \circ z) = \begin{cases} z_1 \quad \text{(i.e., first bit of } z\text{)} & k = 1 \\ G_{k-1}(a) \circ G_{k-1}(\mathsf{Ext}_{k-1}(a,z)) & k > 1 \end{cases} \qquad \diamondsuit$$

Now we use this generator to prove Theorem 21.31. We only need to show that the probability that the machine goes from the start node to the "accept" node is similar for truly random strings and pseudorandom strings. However, we will prove a stronger statement involving intermediate steps as well.

If nodes u is a node in the configuration graph, and s is a string of length 2^k, then we denote by $f_{u,2^k}(s)$ the node that the machine reaches when started in u and its random string is s. Thus if s comes from some distribution $\mathcal{D}$, we can define a distribution $f_{u,2^k}(\mathcal{D})$ on nodes that are 2^k levels further from u.

Lemma 21.34 *Let $r = O(\log n)$ be such that for each $k \leq d \log n$, $\mathsf{Ext}_k : \{0,1\}^{kr} \times \{0,1\}^r \to \{0,1\}^{kr}$ is a $(kr - 2d\log n, \epsilon)$-extractor. For every machine of the type described in the previous paragraphs, and every node u in its configuration graph:*

$$\Delta(f_{u,2^k}(U_{2^k}), f_{u,2^k}(G_k(U_{kr}))) \leq 3^k \epsilon, \tag{16}$$

where U_l denotes the uniform distribution on $\{0,1\}^l$. $\diamondsuit$

To prove Theorem 21.31 from Lemma 21.34 let $u = u_0$, the start configuration, and $2^k = L$, the length of the entire computation. Choose $3^k\epsilon < 1/10$ (say), which means $\log 1/\epsilon = O(\log L) = O(\log n)$. Using the extractor of Section 21.5.6 as Ext_k, we can let $r = O(\log n)$ and so the seed length $kr = O(r \log L) = O(\log^2 n)$.

Proof of Lemma 21.34: Let ϵ_k denote the maximum value of the left hand side of (16) over all machines. The lemma is proved if we can show inductively that $\epsilon_k \leq 2\epsilon_{k-1} + 2\epsilon$. The case $k = 1$ is trivial. At the inductive step, we need to upper bound the distance between two distributions $f_{u,2^k}(\mathcal{D}_1)$, $f_{u,2^k}(\mathcal{D}_4)$, for which we introduce two distributions $\mathcal{D}_2, \mathcal{D}_3$ and use triangle inequality (which holds since $\Delta(\cdot,\cdot)$ is a distance function on distributions):

$$\Delta(f_{u,2^k}(\mathcal{D}_1), f_{u,2^k}(\mathcal{D}_4)) \leq \sum_{i=1}^{3} \Delta(f_{u,2^k}(\mathcal{D}_i), f_{u,2^k}(\mathcal{D}_{i+1}))\,. \tag{17}$$

The distributions will be:

$$\begin{aligned} \mathcal{D}_1 &= U_{2^k} \\ \mathcal{D}_4 &= G_k(U_{kr}) \\ \mathcal{D}_2 &= U_{2^{k-1}} \circ G_{k-1}(U_{(k-1)r}) \\ \mathcal{D}_3 &= G_{k-1}(U_{(k-1)r}) \circ G_{k-1}(U'_{(k-1)r}) \qquad (U, U' \text{ are identical but independent}). \end{aligned}$$

We bound the summands in (17) one by one.

Claim 1: $\Delta(f_{u,2^k}(\mathcal{D}_1) - f_{u,2^k}(\mathcal{D}_2)) \leq \epsilon_{k-1}$.
Denote $\Pr[f_{u,2^{k-1}}(U_{2^{k-1}}) = w]$ by $p_{u,w}$ and $\Pr[f_{u,2^{k-1}}(G_{k-1}(U_{(k-1)r})) = w]$ by $q_{u,w}$. According to the inductive assumption,

$$\frac{1}{2}\sum_w |p_{u,w} - q_{u,w}| = \Delta(f_{u,2^{k-1}}(U_{2^{k-1}}), f_{u,2^{k-1}}(G_{k-1}(U_{(k-1)r}))) \leq \epsilon_{k-1}.$$

Since $\mathcal{D}_1 = U_{2^k}$ may be viewed as two independent copies of $U_{2^{k-1}}$ we have

$$\Delta(f_{u,2^k}(\mathcal{D}_1), f_{u,2^k}(\mathcal{D}_2)) = \sum_v \frac{1}{2}\left|\sum_w p_{uw}p_{wv} - \sum_w p_{uw}q_{wv}\right|$$

where w, v denote nodes 2^{k-1} and 2^k levels respectively from u

$$= \sum_w p_{uw} \frac{1}{2} \sum_v |p_{wv} - q_{wv}|$$
$$\leq \epsilon_{k-1} \quad \text{(using inductive hypothesis and } \sum_w p_{uw} = 1)$$

Claim 2: $\Delta(f_{u,2^k}(\mathcal{D}_2), f_{u,2^k}(\mathcal{D}_3)) \leq \epsilon_{k-1}$.

The proof is similar to the previous case and is omitted.

Claim 3: $\Delta(f_{u,2^k}(\mathcal{D}_3), f_{u,2^k}(\mathcal{D}_4)) \leq 2\epsilon$.

We use the Recycling Lemma. Let $g_u : \{0,1\}^{(k-1)r} \to [1, W]$ be defined as $g_u(a) = f_{u,2^{k-1}}(G_{k-1}(a))$. (To put it in words, apply the Nisan generator to the seed a and use the result as a random string for the machine, using u as the start node. Output the node you reach after 2^{k-1} steps.) Let $X, Y \in U_{(k-1)r}$ and $z \in U_r$. According to the Recycling Lemma,

$$g_u(X) \circ Y \approx_\epsilon g_u(X) \circ \mathsf{Ext}_{k-1}(X, z),$$

and then Part 5 of Lemma A.21 implies that the equivalence continues to hold if we apply a (deterministic) function to the second string on both sides. Thus

$$g_u(X) \circ g_w(Y) \approx_\epsilon g_u(X) \circ g_w(\mathsf{Ext}_{k-1}(X, z))$$

for all nodes w that are 2^{k-1} levels after u. The left distribution corresponds to $f_{u,2^k}(\mathcal{D}_3)$ (by which we mean that $\Pr[f_{u,2^k}(\mathcal{D}_3) = v] = \sum_w \Pr[g_u(X) = w \wedge g_w(Y) = v]$) and the right one to $f_{u,2^k}(\mathcal{D}_4)$ and the proof is completed. ■

What have we learned?

- Often we can easily show that a random object has certain attractive properties, but it's non-trivial to come up with an *explicit* construction of an object with these properties. Yet, once found, such explicit constructions are often extremely useful.
- The behavior of random walks on a graph is tightly related to the eigenvalues of its adjacency matrix (or, equivalently, its normalized version— the random-walk matrix).
- An *expander* graph family is a collection of constant-degree graphs whose second largest eigenvalue is bounded away from 1. Such families can be shown to exist using the probabilistic method, but we also know of *explicit* constructions.
- An ℓ-step random walk on an expander graph is to a certain extend "pseudorandom" and behaves similarly to ℓ randomly chosen vertices under certain measures. This fact has been found useful in a variety of setting, from the randomness efficient error reduction procedure for **BPP** to the logspace algorithm for undirected connectivity.
- Extractors are functions that transform a distribution with a large min-entropy into (close to) the uniform distribution.
- Pseudorandom generators with a "black-box" analysis of their correctness can be used to construct randomness extractors, even though the latter are based on no unproven assumptions or lower bounds.

Chapter notes and history

Expanders were first defined by Bassalygo and Pinsker [BP73] and Pinsker [Pin73] proved their existence using the probabilistic method. They were motivated by the question of finding explicit

graphs to replace the random graphs in an error-correcting code construction by Gallager [Gal63]. Margulis [Mar73] gave the first explicit construction of an expander family although he did not give any bound on the parameter $\lambda(G)$ of graphs G in the family except to prove it is bounded away from 1. Gabber and Galil [GG79] improved Margulis's analysis and gave an explicit bound on $\lambda(G)$, a bound that was later improved by Jimbo and Marouka [JM85]. Lubotzky, Phillips and Sarnak [LPS86] and Margulis [Mar88] constructed *Ramanujan* graphs, that are expander with an optimal dependence between the parameter λ and their degree. The Alon-Boppanna lower bound on the second eigenvalue of a d-regular graph was first stated in [Alo86]; a tight bound on the $o(1)$ error term was given in [Nil04].

The relation between the algebraic (eigenvalue-based) and combinatorial definitions of expanders was developed by Dodziuk, Alon and Milman, and Alon in the papers [Dod84, AM84, AM85, Alo86]. Sinclair and Jerrum [SJ88] generalized this relation to the case of general reversible Markov chains. All of these results can be viewed as a discrete version of a result by Cheeger [Che70] on compact Riemannian manifolds.

Lemma 21.4 (every connected graph has some spectral gap) is from Alon and Sudakov [AS00a] and is an improved version of a result appearing as Problem 11.29 in Lovász's book [Lov07]. Lemma 21.11 (Expander Mixing Lemma) is from Alon and Chung [AC86] (though there it's stated with $T = V \setminus S$).

Karp, Pippenger and Sipser [KPS85] were the first to use expanders for derandomization, specifically showing how to use them to reduce the error of an **RP**-algorithm from 1/3 to $1/\sqrt{k}$ using only $O(k)$ additional random bits. Ajtai, Komlos, and Szemeredi [AKS87] were the first to use random walks on expander graphs for derandomization in their result that every **RL** algorithm using less $\log^2 n/\log\log n$ random bits can be simulated in deterministic log space. Cohen and Wigderson [CW89] and Impagliazzo-Zuckerman [IZ89] independently showed how to use the [AKS87] analysis to reduce the error of both **RP** and **BPP** algorithms as described in Section 21.2.5 (error reduction from 1/3 to 2^{-k} using $O(k)$ additional bits). An improved analysis of such walks was given by Gillman [Gil93] who proved the Expander Chernoff Bound (Theorem 21.15). Some additional improvements were given in [Kah97, WX05, Hea06].

The explicit construction of expanders presented in Section 21.3 is due to Reingold, Vadhan and Wigderson [RVW00], although our presentation follows [RV05, RTV06]. The expansion properties of the replacement product were also analyzed in a particular case of products of two cubes by Gromov [Gro83] and for general graphs (in a somewhat different context) by Martin and Randall [MR00].

Hoory, Linial and Wigderson [HLW06] give an excellent introduction to expander graphs and their computer science applications. The Alon-Spencer book [AS00b] also contains several results on expanders.

The problem of randomness extraction was first considered in the 1950s by von Neumann [vN51] who wanted to extract randomness from biased (but independent) random coins. This was generlized to Markov chains by Blum [Blu84]. Santha and Vazirani [SV84] studied extraction for the much more general class now known as "Santha Vazirani sources" (see Exercise 21.23), that necessitates adding a seed and allowing the output to have some small statistical distance from the uniform. Vazirani and Vazirani [VV85] showed how to simulate **RP** using a Santha-Vazirani source. Chor and Goldreich [CG85] improved the analysis of [SV84, VV85] and generalized further the class of sources. In particular they introduced the notion of *min-entropy*, and studied *block* sources, where each block has significant min-entropy even conditioned on the previous block. They also studied extraction from several (two or more) independent sources of high min-entropy (i.e., (k, n) sources for $k > n/2$). Zuckerman [Zuc90] put forward the goal of simulating probabilistic algorithms using a single source of high min-entropy and observed this generalizes all models that had been studied to date. (See [SZ94] for an account of various models considered by previous researchers.) Zuckerman also gave the first simulation of probabilistic algorithms from (k, n) sources assuming $k = \Omega(n)$. We note that extractors were also used implicitly in an early work of Sipser [Sip86] who showed certain conditional derandomization results under the assumption that certain (variants of) extractors exist (though he described them in a different way).

Extractors (albeit with long seed length) were also implicitly constructed and used in cryptography, using pairwise independent hash functions and the *leftover hash lemma* (Lemma 21.26) of Impagliazzo, Levin, and Luby [ILL89] and a related precursor by Bennett, Brassard and Robert [BBR88]. Nisan [Nis90] then showed that hashing (in particular the [VV85] generator) could be used to obtain provably good pseudorandom generators for logspace. Nisan and Zuckerman [NZ93] first defined extractors. They also gave a new extractor construction and used it to achieve their result that in general the amount of randomness used by a probabilistic algorithm can be reduced from polynomial to linear in the algorithm's space complexity. Since then a long sequence of beautiful works was dedicated to improving the parameters of extractors, on the way discovering many impor-

tant tools that were used in other areas of theoretical computer science. In particular, Guruswami et al [GUV07] (slightly improving over Lu et al [LRVW03]) constructed an extractor that has both seed length and output length within a constant factor of the optimal non-explicit extractor of Theorem 21.25. See [Sha02] for a good (though slightly outdated) survey on extractor constructions and their applications.

Trevisan's [Tre99] insight about using pseudorandom generators to construct extractors (see Section 21.5.7) has now been greatly extended. It is now understood that three combinatorial objects studied in three different fields are very similar: pseudorandom generators (cryptography and derandomization), extractors (weak random sources) and list-decodable error-correcting codes (coding theory and information theory). Constructions of any one of these objects often gives constructions of the other two. See the survey by Vadhan [Vad07].

Theorem 21.31 is by Nisan [Nis90], who also showed that all of **BPL** can be simulated using polynomial-time and $O(\log^2 n)$ space. The proof we presented is by Impagliazzo, Nisan, and Wigderson [INW94], with the extractor-based viewpoint due to Raz and Reingold [RR99]. Saks and Zhou [SZ95] extended Nisan's techniques to show an $O(\log^{1.5} n)$-space algorithm for every problem in **BPL**.

As perhaps the most important example of an **RL** problem, undirected connectivity has received special attention in the literature. Nisan, Szemeredi and Wigderson [NSW92] gave the first deterministic algorithm for undirected connectivity using $o(log^2 n)$ space, specifically $O(\log^{1.5} n)$; as mentioned above this result was later generalized to all of **RL** by [SZ95]. Armoni et al [ATSWZ97] improved the bound for undirected connectivity to $O(\log^{4/3} n)$ space. The deterministic space complexity of undirected connectivity was finally resolved by Reingold [Rei05] who showed that it lies in **L** (Theorem 21.21). Trifonov [Tri05] proved concurrently and independently the slightly weaker result of an $O(\log n \log\log n)$-space algorithm for this problem.

Exercises

21.1 Prove Claim 21.1 using the Cauchy-Schwartz Inquality— $|\langle \mathbf{u}, \mathbf{v}\rangle| \leq \|\mathbf{u}\|_2 \|\mathbf{v}\|_2$ for every two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$.

21.2 **(a)** Prove Hölder's Inequality (see Section A.5.4): For every p, q with $\frac{1}{p} + \frac{1}{q} = 1$, $\|\mathbf{u}\|_p \|\mathbf{v}\|_q \geq \sum_{i=1}^n |\mathbf{u}_i \mathbf{v}_i|$. Note that the Cauchy-Schwartz Inequality is the special case of Hölder's Inequality with $p = q = 2$. **H464**

(b) For a vector $\mathbf{v} \in \mathbb{R}^n$, define $\|\mathbf{v}\|_\infty = \max_{i\in[n]} |\mathbf{v}_i|$. Show that this is a norm and that for every $\mathbf{v}$,

$$\|\mathbf{v}\|_\infty = \lim_{p\to\infty} \left(\sum_{i=1}^n |\mathbf{v}_i|^p \right)^{1/p} .$$

(c) Prove that $\|\mathbf{v}\|_2 \leq \sqrt{|\mathbf{v}|_1 \|\mathbf{v}\|_\infty}$ for every vector $\mathbf{v} \in \mathbb{R}^n$. **H464**

21.3 Prove that if G is an n-vertex bipartite graph then there exists a vector $\mathbf{v} \in \mathbb{R}^n$ such that $A\mathbf{v} = -\mathbf{v}$ where A is the random-walk matrix of G.

21.4 Prove that for every n-vertex d-regular graph G, the diameter of G (maximum over all pairs of distinct vertices i, j in G of the length of the shortest path in G between i and j) is at most $3n/(d+1)$. **H465**

21.5 Recall that the *spectral norm* of a matrix A, denoted $\|A\|$, is defined as the maximum of $\|A\mathbf{v}\|_2$ for every unit vector $\mathbf{v}$. Let A be a symmetric stochastic matrix: i.e., $A = A^\dagger$ and every row and column of A has non-negative entries summing up to one. Prove that $\|A\| \leq 1$. **H465**

21.6 Let A, B be two $n \times n$ matrices.

(a) Prove that $\|A + B\| \leq \|A\| + \|B\|$.

(b) Prove that $\|AB\| \leq \|A\|\|B\|$.

21.7 Let A, B be two symmetric stochastic matrices. Prove that $\lambda(A + B) \leq \lambda(A) + \lambda(B)$.

21.8 Prove Lemma 21.16. **H465**

21.9 **(a)** Prove that if a probability distribution X has support of size at most d, its collision probability is at least $1/d$.

(b) Prove that if G is an (n, d, λ)-graph and X is the distribution over a random neighbor of the first vertex, then the collision probability of X is at most $\lambda^2 + 1/n$.

(c) Prove that $\lambda \geq \sqrt{\frac{1}{d} - \frac{1}{n}} = \frac{1}{\sqrt{d}} + o(1)$ (where $o(1)$ is a term that tends to 0 with n).

21.10 Recall that the *trace* of a Matrix A, denoted $\mathrm{tr}(A)$, is the sum of the entries along its diagonal.

(a) Prove that if an $n \times n$ matrix A has eigenvalues $\lambda_1, \ldots, \lambda_n$, then $\mathrm{tr}(A) = \sum_{i=1}^{n} \lambda_i$.

(b) Prove that if A is a random-walk matrix of an n-vertex graph G, and $k \geq 1$, then $\mathrm{tr}(A^k)$ is equal to n times the probability that a if we select a vertex i uniformly at random and take a k step random walk from i, then we end up back in i.

(c) Prove that for every d-regular graph G, $k \in \mathbb{N}$ and vertex i of G, the probability that a path of length k from i ends up back in i is at least as large as the corresponding probability in T_d, where T_d is the complete $(d-1)$-ary tree of depth k rooted at i. (That is, every internal vertex has degree d— one parent and $d-1$ children.)

(d) Prove that for even k the probability that a path of length k from the root of T_d ends up back at v is at least $2^{k - k \log d/2 - o(k)}$. H465

(e) Prove that for every n-vertex d-degree graph G, $\lambda(G) \geq \frac{2}{\sqrt{d}}(1 + o(1))$, where $o(1)$ denotes a term, depending on n and d that tends to 0 as n grows. H465

21.11 Let an n, d random graph be an n-vertex graph chosen as follows: choose d random permutations $\pi_1, \ldots, \pi_d$ from $[n]$ to $[n]$. Let the the graph G contains an edge (u, v) for every pair u, v such that $v = \pi_i(u)$ for some $1 \leq i \leq d$. Prove that a random n, d graph is an $(n, 2d, \frac{1}{10})$ edge expander with probability $1 - o(1)$ (i.e., tending to one with n). H465

21.12 In this exercise we show how to extend the error reduction procedure of Section 21.2.5 to two-sided (**BPP**) algorithms.

(a) Prove that under the conditions of Theorem 21.12, for every subset $I \subseteq [k]$,

$$\Pr[\forall_{1 \leq i \leq I} X_i \in B] \leq ((1-\lambda)\sqrt{\beta} + \lambda)^{|I|-1} .$$

(b) Conclude that if $|B| < n/10$ and $\lambda < 1/100$ then the probability that there exists a subset $I \subseteq [k]$ such that $|I| > k/10$ and $\forall_{1 \leq i \leq I} X_i \in B$ is at most $2^{-k/100}$.

(c) Use this to show a procedure that transforms every **BPP** algorithm A that uses m coins and decides a language L with probability 0.9 into an algorithm B that uses $m + O(k)$ coins and decides the language L with probability $1 - 2^{-k}$.

21.13 Prove that for every n-vertex d-regular graph, there exists a subset S of $n/2$ vertices, such that $E(S, \bar{S}) \leq dn/4$. Conclude that there does not exist an (n, d, ρ) edge expander for $\rho > 1/2$. H465

21.14 Prove the Expander Mixing Lemma (Lemma 21.11). H465

21.15 [Tan84] A graph where $|\Gamma(S)| \leq c|S|$ for every not-too-big set S (say, $|S| \leq n/(10d)$) is said to have *vertex expansion* c. This exercise shows that graphs with the minimum possible second eigenvalue $\frac{2}{\sqrt{d}}(1 + o(1))$ have vertex expansion roughly $d/4$. It is known that such graphs have in fact vertex expansion roughly $d/2$ [Kah92], and there are counterexamples showing this is tight. In contrast, random d-regular graphs have vertex expansion $(1 - o(1))d$.

(a) Prove that if $\mathbf{p}$ is a probability vector then $\|\mathbf{p}\|_2^2$ is equal to the probability that if i and j are chosen from $\mathbf{p}$, then $i = j$.

(b) Prove that if $\mathbf{s}$ is the probability vector denoting the uniform distribution over some subset S of vertices of a graph G with random-walk matrix A, then $\|A\mathbf{p}\|_2^2 \geq 1/|\Gamma(S)|$, where $\Gamma(S)$ denotes the set of S's neighbors.

(c) Prove that if G is an (n, d, λ)-expander graph, and S is a subset of ϵn vertices, then

$$|\Gamma(S)| \geq \frac{|S|}{2\lambda^2\left((1-\epsilon)^2 - 2\epsilon/\lambda^2\right)} .$$

H465

21.16 If G is a graph and S is a subset of G's vertices then by *contracting* S we mean transforming G into a graph H where all of S's members are replaced by a single vertex s with an edge $\overline{s\,v}$ in H for every edge $\overline{u\,v}$ in G where $u \in S$. Let G be an (n, d, ρ) edge expander, and let H be the $n' = n - (c-1)k$ vertex cd degree graph obtained by taking k disjoint c-sized subsets $S_1, \ldots, S_k$ of G's vertices and contracting them, and then adding self loops to the other vertices to ensure that the graph is regular. Prove that H is an $(n', cd, \rho/(2c))$ edge expander. Use this to complete the proof of Theorem 21.19. H465

21.17 Prove that for every function $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ and there exists an $(n, n-1)$-source X and a bit $b \in \{0,1\}$ such that $\Pr[\mathsf{Ext}(X)_1 = b] = 1$ (where $\mathsf{Ext}(X)_1$ denotes the first bit of $\mathsf{Ext}(X)$). Prove that this implies that $\Delta(\mathsf{Ext}(X), U_m) \geq 1/2$.

21.18 **(a)** Show that there is a deterministic poly(n)-time algorithm A that given an input distributed according to the distribution X with $H_\infty(X) \geq n^{100}$ and black box access to any function $f : \{0,1\}^n \to \{0,1\}$ outputs 1 with probability at least 0.99 if $\mathsf{E}[f(U_n)] \geq 2/3$ and outputs 0 with probability at least 0.99 if $\mathsf{E}[f(U_n)] \leq 1/3$. We call such an algorithm a *function approximator*.

(b) Show that there is no deterministic polynomial-time function approximator A without getting an additional randomized input (i.e., there is no deterministic function approximator). H465

(c) Show that for every probability distribution X, if $\Delta(X,Y) > 1/10$ for every Y with $H_\infty(Y) \geq n/2$, then there is no polynomial-time function approximator that gets X as an input. Conclude that access to a high min entropy distribution is necessary for black-box simulation of **BPP** algorithms. H465

21.19 . Say that a distribution Y is a *convex combination* of distributions $Y_1, \ldots, Y_N$ if there exist some non-negative numbers $\alpha_1, \ldots, \alpha_N$ summing up to 1 such that Y is the distribution obtained by picking i with probability α_i and sampling an element from Y_i. Prove that if this is the case then for every distribution X,

$$\Delta(X,Y) \leq \sum_i \alpha_i \Delta(X,Y_i) \leq \max_i \Delta(X,Y_i)\,.$$

H465

21.20 Suppose Boolean function f is (S,ϵ)-hard and let D be the distribution on m-bit strings defined by picking inputs $x_1, x_2, \ldots, x_m$ uniformly at random and outputting $f(x_1)f(x_2)\cdots f(x_m)$. Show that the statistical distance between D and the uniform distribution is at most ϵm.

21.21 Prove Lemma 21.26.

21.22 Let A be an $n \times n$ matrix with eigenvectors $\mathbf{u}^1, \ldots, \mathbf{u}^n$ and corresponding values $\lambda_1, \ldots, \lambda_n$. Let B be an $m \times m$ matrix with eigenvectors $\mathbf{v}^1, \ldots, \mathbf{v}^m$ and corresponding values $\alpha_1, \ldots, \alpha_m$. Prove that the matrix $A \otimes B$ has eigenvectors $\mathbf{u}^i \otimes \mathbf{v}^j$ and corresponding values $\lambda_i \cdot \alpha_j$.

21.23 Prove that for every two graphs G, G', $\lambda(G \otimes G') \leq \lambda(G) + \lambda(G')$ without using the fact that every symmetric matrix is diagonalizable. H465

21.24 Let G be an n-vertex D-degree graph with ρ edge expansion for some $\rho > 0$. (That is, for every a subset S of G's vertices of size at most $n/2$, the number of edges between S and its complement is at least $\rho d|S|$.) Let G' be a D-vertex d-degree graph with ρ' edge expansion for some $\rho' > 0$. Prove that $G \circledR G'$ has at least $\rho^2\rho'/1000$ edge expansion. H465

Chapter 22

Proofs of PCP Theorems and the Fourier Transform Technique

We saw in Chapter 11 that the **PCP** Theorem implies that computing approximate solutions to many optimization problems is **NP**-hard. This chapter gives a complete proof of the **PCP** Theorem. In Chapter 11 we also mentioned that the **PCP** Theorem does not suffice for proving several other similar results, for which we need stronger (or simply different) "**PCP** Theorems". In this chapter we survey some such results and their proofs. The two main results are Raz's *parallel repetition theorem* (see Section 22.3) and Håstad's 3-bit **PCP** theorem (Theorem 22.16). Raz's theorem leads to strong hardness results for the 2CSP problem over large alphabets. Håstad's theorem shows that certificates for **NP** languages can be probabilistically checked by examining only 3 bits in them. One of the consequences of Håstad's result is that computing a $(7/8+\epsilon)$-approximation for the MAX-3SAT problem is **NP**-hard for every $\epsilon > 0$. Since we know that 7/8-approximation is in fact possible in polynomial time (see Example 11.2 and Exercise 11.3), this shows (assuming $\mathbf{P} \neq \mathbf{NP}$) that the approximability of MAX-3SAT has an abrupt transition from easy to hard at 7/8. Such a result is called a *threshold* result, and threshold results are now known for a few other problems.

Håstad's result builds on the other results we have studied, including the (standard) **PCP** Theorem, and Raz's theorem. It also uses Håstad's method of analysing the verifier's acceptance probability using *Fourier transforms*. Such Fourier analysis has also proved useful in other areas in theoretical computer science. We introduce this technique in Section 22.5 by using it to show the correctness of the linearity testing algorithm of Section 11.5, which completes the proof of the result $\mathbf{NP} \subseteq \mathbf{PCP}(\mathrm{poly}(n), 1)$ in Section 11.5. We then use Fourier analysis to prove Håstad's 3-bit **PCP** Theorem.

In Section 22.8 we prove the hardness of approximating the SET-COVER problem. In Section 22.2.3 we prove that computing $n^{-\epsilon}$-approximation to MAX-INDSET in **NP**-hard. In Section 22.9 we briefly survey other **PCP** Theorems that have been proved, including those that assume the so-called *unique games conjecture*.

22.1 Constraint satisfaction problems with non-binary alphabet

In this chapter we will often use the problem $q\mathsf{CSP}_W$, which is defined by extending the definition of qCSP in Definition 11.11 from binary alphabet to an alphabet of size W.

Definition 22.1 *($q\mathsf{CSP}_W$)* For integers $q, W \geq 1$ the $q\mathsf{CSP}_W$ problem is defined analogously to the qCSP problem of Definition 11.11, except the underlying alphabet is $[W] = \{1, 2, \ldots, W\}$ instead of $\{0, 1\}$. Thus constraints are functions mapping $[W]^q$ to $\{0, 1\}$.

For $\rho < 1$ we define $\_\mathsf{GAP}q\mathsf{CSP}W\rho$ analogously to the definition of ρ-GAPqCSP for binary alphabet (see Definition 11.13). ♢

Example 22.2
3SAT is the subcase of $q\mathsf{CSP}_W$ where $q = 3$, $W = 2$, and the constraints are OR's of the involved literals.
Similarly, the **NP**-complete problem 3COL can be viewed as a subcase of $2\mathsf{CSP}_3$ instances where for each edge (i, j), there is a constraint on the variables u_i, u_j that is satisfied iff $u_i \neq u_j$. The graph is 3-colorable iff there is a way to assign a number in $\{0, 1, 2\}$ to each variable such that all constraints are satisfied.

22.2 Proof of the PCP Theorem

This section proves the PCP Theorem. We present Dinur's proof [Din06], which simplifies half of the original proof of [AS92, ALM+92]. Section 22.2.1 gives an outline of the main steps. Section 22.2.2 describes one key step, Dinur's gap amplification technique. Section 22.2.5 describes the other key step, which is from the original proof of the **PCP** Theorem [ALM+92] and its key ideas were presented in the proof of $\mathbf{NP} \subseteq \mathbf{PCP}(\mathrm{poly}(n), 1)$ in Section 11.5.

22.2.1 Proof outline for the PCP Theorem.

As we have seen, the **PCP** Theorem is equivalent to Theorem 11.14, stating that ρ-GAPqCSP is **NP**-hard for some constants q and $\rho < 1$. Consider the case that $\rho = 1 - \epsilon$ where ϵ is not necessarily a constant but can be a function of m (the number of constraints). Since the number of satisfied constraints is always a whole number, if φ is unsatisfiable then $\mathsf{val}(\varphi) \leq 1 - 1/m$. Hence, the gap problem $(1–1/m)$-GAP3CSP is a generalization of 3SAT and is **NP** hard. The idea behind the proof is to start with this observation, and iteratively show that $(1–\epsilon)$-GAPqCSP is **NP**-hard for larger and larger values of ϵ, until ϵ is as large as some absolute constant independent of m. This is formalized in the following definition and lemma.

Definition 22.3 Let f be a function mapping CSP instances to CSP instances. We say that f is a *CL-reduction* (short for *complete linear-blowup reduction*) if it is polynomial-time computable and for every CSP instance φ, satisfies:

Completeness: If φ is satisfiable then so is $f(\varphi)$.

Linear blowup: If m is the number of constraints in φ then the new qCSP instance $f(\varphi)$ has at most Cm constraints and alphabet W, where C and W can depend on the arity and the alphabet size of φ (but not on the number of constraints or variables). ♢

Lemma 22.4 *(**PCP** Main Lemma)*
There exist constants $q_0 \geq 3$, $\epsilon_0 > 0$, and a CL-reduction f such that for every q_0CSP-instance φ with binary alphabet, and every $\epsilon < \epsilon_0$, the instance $\psi = f(\varphi)$ is a q_0CSP (over binary alphabet) satisfying

$$\mathsf{val}(\varphi) \leq 1 - \epsilon \Rightarrow \mathsf{val}(\psi) \leq 1 - 2\epsilon$$

Lemma 22.4 can be succinctly described as follows:

	Arity	Alphabet	Constraints	Value
Original	q_0	binary	m	$1 - \epsilon$
	⇓	⇓	⇓	⇓
Lemma 22.4	q_0	binary	Cm	$1 - 2\epsilon$

This lemma allows us to easily prove the **PCP** Theorem.

Proving Theorem 11.5 from Lemma 22.4. Let $q_0 \geq 3$ be as stated in Lemma 22.4. As already observed, the decision problem q_0CSP is **NP**-hard. To prove the **PCP** Theorem we give a reduction from this problem to GAP q_0CSP. Let φ be a q_0CSP instance. Let m be the number of constraints in φ. If φ is satisfiable then $\mathsf{val}(\varphi) = 1$ and otherwise $\mathsf{val}(\varphi) \leq 1 - 1/m$. We use Lemma 22.4 to amplify this gap. Specifically, apply the function f obtained by Lemma 22.4 to φ a total of $\log m$ times. We get an instance ψ such that if φ is satisfiable then so is ψ, but if φ is not satisfiable (and so $\mathsf{val}(\varphi) \leq 1 - 1/m$) then $\mathsf{val}(\psi) \leq 1 - \min\{2\epsilon_0, 1 - 2^{\log m}/m\} = 1 - 2\epsilon_0$. Note that the size of ψ is at most $C^{\log m} m$, which is polynomial in m. Thus we have obtained a gap-preserving reduction from L to the $(1–2\epsilon_0)$-GAP q_0CSP problem, and the **PCP** theorem is proved. ■

The rest of this section proves Lemma 22.4 by combining two transformations: the first transformation amplifies the gap (i.e., fraction of violated constraints) of a given CSP instance, at the expense of increasing the alphabet size. The second transformation reduces back the alphabet to binary, at the expense of a modest reduction in the gap. The transformations are described in the next two lemmas.

Lemma 22.5 *(Gap Amplification* [Din06]*)*
For every $\ell, n \in \mathbb{N}$, there exist numbers $W \in \mathbb{N}, \epsilon_0 > 0$ and a CL-reduction $g_{\ell,q}$ such that for every qCSP instance φ with binary alphabet, the instance $\psi = g_{\ell,q}(\varphi)$ has arity only 2, uses alphabet of size at most W and satisfies:

$$\mathsf{val}(\varphi) \leq 1 - \epsilon \Rightarrow \mathsf{val}(\psi) \leq 1 - \ell\epsilon$$

for every $\epsilon < \epsilon_0$.

Lemma 22.6 *(Alphabet Reduction)*
There exists a constant q_0 and a CL- reduction h such that for every CSP instance φ, if φ had arity two over a (possibly non-binary) alphabet $\{0..W–1\}$ then $\psi = h(\varphi)$ has arity q_0 over a binary *alphabet and satisfies:*

$$\mathsf{val}(\varphi) \leq 1 - \epsilon \Rightarrow \mathsf{val}(h(\varphi)) \leq 1 - \epsilon/3$$

Lemmas 22.5 and 22.6 together imply Lemma 22.4 by setting $f(\varphi) = h(g_{6,q_0}(\varphi))$. Indeed, if φ was satisfiable then so will $f(\varphi)$. If $\mathsf{val}(\varphi) \leq 1-\epsilon$, for $\epsilon < \epsilon_0$ (where ϵ_0 the value obtained in Lemma 22.5 for $\ell = 6$, $q = q_0$) then $\mathsf{val}(g_{6,q_0}(\varphi)) \leq 1 - 6\epsilon$ and hence $\mathsf{val}(h(g_{6,q_0}(\varphi))) \leq 1 - 2\epsilon$. This composition is described in the following table:

	Arity	Alphabet	Constraints	Value
Original	q_0	binary	m	$1-\epsilon$
	⇓	⇓	⇓	⇓
Lemma 22.5 ($\ell = 6$, $q = q_0$)	2	W	Cm	$1-6\epsilon$
	⇓	⇓	⇓	⇓
Lemma 22.6	q_0	binary	$C'Cm$	$1-2\epsilon$

22.2.2 Dinur's Gap Amplification: Proof of Lemma 22.5

To prove Lemma 22.5, we need to exhibit a function g that maps a qCSP instance to a $2\mathsf{CSP}_W$ instance over a larger alphabet $\{0..W–1\}$ in a way that increases the fraction of violated constraints. In the proof verification viewpoint (Section 11.3), the fraction of

violated constraints is merely the soundness parameter. So at first sight, our task merely seems to be reducing the "soundness" parameter of a **PCP** verifier, which as already noted (in the Remarks following Theorem 11.5) can be easily done by repeating the verifier's operation 2 (or more generally, k) times. The problem with this trivial idea is that the CSP instance corresponding to k repeated runs of the verifier is not another 2CSP instance, but an instance of arity $2k$ since the verifier's decision depends upon $2k$ different entries in the proof. In the next chapter, we will see another way of "repeating" the verifier's operation using *parallel repetition*, which does result in 2CSP instances, but greatly increases the size of the CSP instance. By contrast, here we desire a CL-reduction, which means the size must only increase by a constant factor. The key to designing such a CL-reduction is walks in expander graphs, which we describe separately first in Section 22.2.3 since it is of independent interest.

22.2.3 Expanders, walks, and hardness of approximating INDSET

Dinur's proof uses expander graphs, which are described in Chapter 21. Here we recap the facts about expanders used in this chapter, and as illustration we use them to prove a hardness result for MAX-INDSET.

In Chapter 21 we define a parameter $\lambda(G) \in [0,1]$, for every regular graph G (see Definition 21.2). For every $c \in (0,1)$, we call a regular graph G satisfying $\lambda(G) \leq c$ a *c-expander graph*. If $c < 0.9$, we drop the prefix c and simply call G an *expander graph*. (The choice of the constant 0.9 is arbitrary.) As shown in Chapter 21, for every constant $c \in (0,1)$ there is a constant d and an algorithm that given input $n \in N$, runs in $\mathrm{poly}(n)$ time and outputs the adjacency matrix of an n-vertex d-regular c-expander (see Theorem 21.19).

The main property we need in this chapter is that for every regular graph $G = (V,E)$ and every $S \subseteq V$ with $|S| \leq |V|/2$,

$$\Pr_{(u,v)\in E}[u \in S, v \in S] \leq \frac{|S|}{|V|}\left(\frac{1}{2} + \frac{\lambda(G)}{2}\right) \quad (1)$$

(Exercise 22.1) Another property we use is that $\lambda(G^\ell) = \lambda(G)^\ell$ for every $\ell \in \mathbb{N}$, where G^ℓ is obtained by taking the adjacency matrix of G to the ℓ^{th} power (i.e., an edge in G^ℓ corresponds to an (ℓ–1)-step path in G). Thus (1) also implies that

$$\Pr_{(u,v)\in E(G^\ell)}[u \in S, v \in S] \leq \frac{|S|}{|V|}\left(\frac{1}{2} + \frac{\lambda(G)^\ell}{2}\right) \quad (2)$$

Example 22.7

As an application of random walks in expanders, we describe how to prove a stronger version of the hardness of approximation result for INDSET in Theorem 11.15. This is done using the next Lemma, which immediately implies (since $m = \mathrm{poly}(n)$) that there is some $\epsilon > 0$ such that computing $n^{-\epsilon}$-approximation to MAX-INDSET is **NP**-hard in graphs of size n. (See Section 22.9.2 for a survey of stronger hardness results for MAX-INDSET.) Below, $\tilde{\alpha}(F)$ denotes the fractional size of the largest independent set in F. It is interesting to note how this Lemma gives a more efficient version of the "self-improvement" idea of Theorem 11.15.

Lemma 22.8 *For every $\lambda > 0$ there is a polynomial-time computable reduction f that maps every n-vertex graph F into an m-vertex graph H such that*

$$(\tilde{\alpha}(F) - 2\lambda)^{\log n} \leq \tilde{\alpha}(H) \leq (\tilde{\alpha}(F) + 2\lambda)^{\log n}$$

PROOF: We use random walks to define a more efficient version of the "graph product" used in the proof of Corollary 11.17. Let G be an expander on n nodes that is d-regular (where d is some constant independent of n) and let $\lambda = \lambda(G)$. For notational ease we assume G, F have the same set of vertices. We will map F into a graph H of $nd^{\log n - 1}$ vertices in the following way:

- The vertices of H correspond to all the $(\log n - 1)$-step paths in the λ-expander G.
- We put an edge between two vertices u, v of H corresponding to the paths $\langle u_1, \ldots, u_{\log n} \rangle$ and $\langle v_1, \ldots, v_{\log n} \rangle$ if there exists an edge in G between two vertices in the set $\{u_1, \ldots, u_{\log n}, v_1, \ldots, v_{\log n}\}$.

It is easily checked that for any independent set in H if we take all vertices of F appearing in the corresponding walks, then that constitutes an independent set in F. From this observation the proof is concluded using Exercise 22.2. ■ ♢

22.2.4 Dinur's Gap-amplification

We say that a $q\mathsf{CSP}_W$ instance φ is "nice" if it satisfies the following properties:

Property 1: The arity q of φ is 2 (though the alphabet may be non binary).

Property 2: Let the *constraint graph* of φ be the graph G with vertex set $[n]$ where for every constraint of φ depending on the variables u_i, u_j, the graph G has the edge (i, j). We allow G to have parallel edges and self-loops. Then G is d-regular for some constant d (independent of the alphabet size) and at every node, half the edges incident to it are self-loops.

Property 3: The constraint graph is an expander. That is, $\lambda(G) \leq 0.9$.

It turns out that when proving Lemma 22.5 we may assume without loss of generality that the CSP instance φ is nice, since there is a relatively simple CL reduction mapping arbitrary $q\mathsf{CSP}$ instances to "nice" instances. (See Section 22.A; we note that these CL reductions will actually *lose* a factor depending on q in the soundness gap, but we can regain this factor by choosing a large enough value for t in Lemma 22.9 below.) The rest of the proof consists of a "powering" operation for nice 2CSP instances. This is described in the following lemma:

Lemma 22.9 *(Powering) There is an algorithm that given any $2\mathsf{CSP}_W$ instance ψ satisfying Properties 1 through 3 and an integer $t \geq 1$ produces an instance ψ^t of 2CSP such that:*

1. *ψ^t is a $2\mathsf{CSP}_{W'}$-instance with alphabet size $W' < W^{d^{5t}}$, where d denote the degree of ψ's constraint graph. The instance ψ^t has $d^{t+\sqrt{t}+1}n$ constraints, where n is the number of variables in ψ.*
2. *If ψ is satisfiable then so is ψ^t.*
3. *For every $\epsilon < \frac{1}{d\sqrt{t}}$, if $\mathsf{val}(\psi) \leq 1 - \epsilon$ then $\mathsf{val}(\psi^t) \leq 1 - \epsilon'$ for $\epsilon' = \frac{\sqrt{t}}{10^5 dW^4}\epsilon$.*
4. *The formula ψ^t is produced from ψ in time polynomial in m and W^{d^t}.* ♢

PROOF: Let ψ be a $2\mathsf{CSP}_W$-instance with n variables and $m = nd$ constraints, and as before let G denote the *constraint graph* of ψ. To prove Lemma 22.9, we first show how we construct the formula ψ^t from ψ. The main idea is a certain "powering" operation on constraint graphs using random walks of a certain length.

Construction of ψ^t. The formula ψ^t will have the same number of variables as ψ. The new variables $\mathbf{y} = y_1, \ldots, y_n$ take values over an alphabet of size $W' = W^{d^{5t}}$, and thus a value of a new variable y_i is a d^{5t}-tuple of values in $\{0..W\text{–}1\}$. To avoid confusion in the rest of the proof, we reserve the term "variable" for these new variables, and say "old variables" if we mean the variables of ψ.

We will think of a value of variable y_i as giving a value in $\{0..W\text{–}1\}$ to every old variable u_j where j can be reached from i using a path of at most $t+\sqrt{t}$ steps in G (see Figure 22.1). In other words it gives an assignment for every u_j such that j is in the ball of radius $t+\sqrt{t}$ and center i in G. Since graph G is d-regular, the number of such nodes is at most $d^{t+\sqrt{t}+1}$, which is less than d^{5t}, so this information can indeed be encoded using an alphabet of size W'.

Below, we will often say that an assignment to y_i "claims" a certain value for the old variable u_j. Of course, the assignment to a different variable $y_{i'}$ could claim a different value for u_j; these potential inconsistences make the rest of the proof nontrivial. In fact, the constraints in the $2\mathsf{CSP}_{W'}$ instance ψ^t are designed to reveal such consistencies.

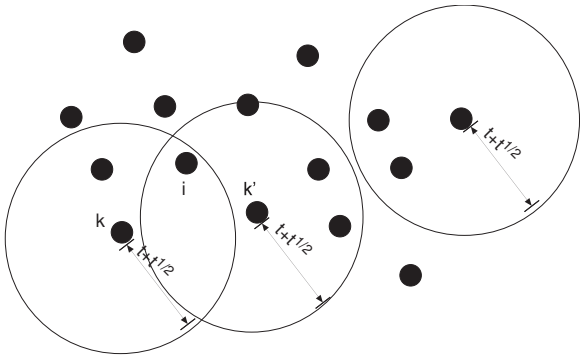


Figure 22.1 The CSP ψ^t consists of n variables taking values in an alphabet of size $W^{d^{5t}}$. An assignment to a new variable y_i encodes an assignment to all old variables of ψ corresponding to nodes that are in a ball of radius $t+\sqrt{t}$ around i in ψ's constraint graph. An assignment $y_1,\ldots,y_n$ to ψ^t may be *inconsistent* in the sense that if j falls in the intersection of two such balls centered at i and i', then y_i and $y_{i'}$ may claim different values for u_i.

For every $(2t+1)$-step path $p=\langle i_1,\ldots,i_{2t+2}\rangle$ in G, we have one corresponding constraint C_p in ψ^t (see Figure 22.2). The constraint C_p depends on the variables y_{i_1} and $y_{i_{2t+2}}$ (so we do indeed produce an instance of $2\mathsf{CSP}_{W'}$) and outputs FALSE if (and only if) there is some $j\in[2t+1]$ such that:

1. i_j is in the $t+\sqrt{t}$-radius ball around i_1.
2. i_{j+1} is in the $t+\sqrt{t}$-radius ball around i_{2t+2}
3. If w denotes the value y_{i_1} claims for u_{i_j} and w' denotes the value $y_{i_{2t+2}}$ claims for $u_{i_{j+1}}$, then the pair (w,w') violates the constraint in ψ that depends on u_{i_j} and $u_{i_{j+1}}$.

A few observations are in order. First, the time to produce this $2\mathsf{CSP}_{W'}$ instance is polynomial in m and W^{d^t}, so part 4 of Lemma 22.5 is trivial. Second, for every assignment to the old variables $u_1,u_2,\ldots,u_n$ we can "lift" it to a *canonical* assignment to $y_1,\ldots,y_n$ by simply assigning to each y_i the vector of values assumed by u_j's that lie in a ball of radius $t+\sqrt{t}$ and center i in G. If the assignment to the u_j's was a satisfying assignment for ψ, then this canonical assignment satisfies ψ^t, since it will satisfy all constraints encountered in walks of length $2t+1$ in G. Thus part 2 of Lemma 22.5 is also trivial. This leaves part 3 of the Lemma, the most difficult part. We have to show that if $\mathsf{val}(\psi)\leq 1-\epsilon$ then every assignment to the y_i's satisfies at most $1-\epsilon'$ fraction of constraints in ψ^t, where $\epsilon<\frac{1}{d\sqrt{t}}$ and $\epsilon'=\frac{\sqrt{t}}{10^5 dW^4}\epsilon$.

The plurality assignment. To prove part 3 of the lemma, we show how to transform every assignment $\mathbf{y}$ for ψ^t into an assignment $\mathbf{u}$ for ψ and then argue that if $\mathbf{u}$ violates a "few" (i.e., ϵ fraction) of ψ's constraints then $\mathbf{y}$ violates "many" (i.e., $\epsilon'=\Omega(\sqrt{t}\epsilon)$ fraction) of constraints of ψ^t.

From now on, let us fix some arbitrary assignment $\mathbf{y}=y_1,\ldots,y_n$ to ψ^t's variables. As already noted, the values y_i's may be mutually *inconsistent* and not correspond to any

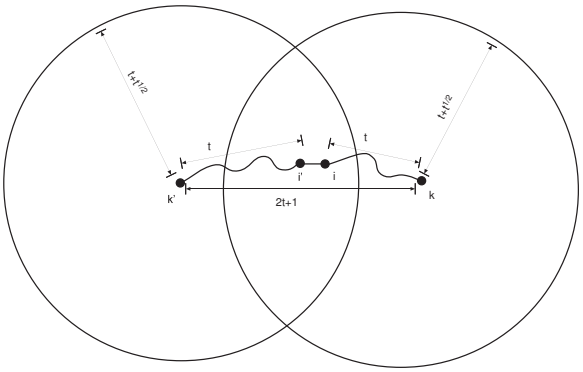


Figure 22.2 ψ^t has one constraint for every path of length $2t+1$ in ψ's constraint graph, checking that the views of the balls centered on the path's two endpoints are consistent with one another and the constraints of ψ.

obvious assignment for the old variable u_j's. The following notion is key because it tries to extract a single assignment for the old variables.

For every variable u_i of ψ, we define the random variable Z_i over $\{0,\ldots,W-1\}$ to be the result of the following process: starting from the vertex i, take a t step random walk in G to reach a vertex k, and output the value that y_k claims for u_i. We let z_i denote the most likely value of Z_i. If more than one value is most likely, we break ties arbitrarily. We call the assignment $z_1,\ldots,z_n$ the *plurality assignment* (see Figure 22.3). Note that $Z_i = z_i$ with probability at least $1/W$.

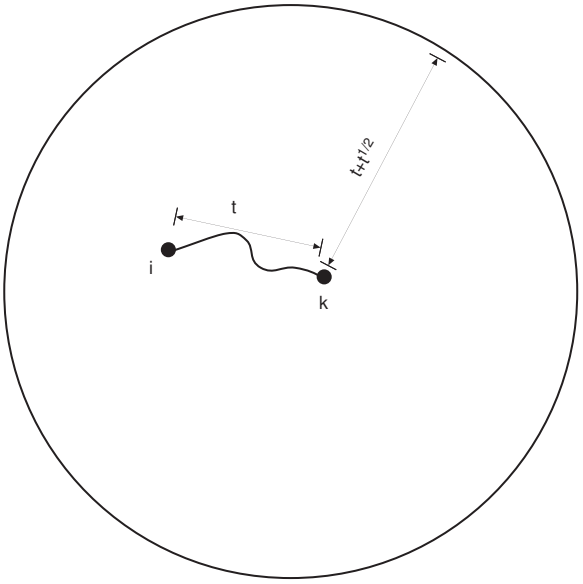


Figure 22.3 An assignment y for ψ^t induces a plurality assignment u for ψ in the following way: u_i gets the most likely value that is claimed for it by y_k, where k is obtained by taking a t-step random walk from i in the constraint graph of ψ.

Since $\mathsf{val}(\psi) \leq 1-\epsilon$, *every* assignment for ψ fails to satisfy ϵ fraction of the constraints, and this is therefore also true for the plurality assignment. Hence there exists a set F of $\epsilon m = \epsilon nd/2$ constraints in ψ that are violated by the assignment $\mathbf{z} = z_1,\ldots,z_n$. We will use this set F to show that at least an ϵ' fraction of ψ^t's constraints are violated by the assignment $\mathbf{y}$.

Analysis. The rest of the proof defines events in the following probability space: we pick a $(2t+1)$-step path, denoted $\langle i_1, \ldots, i_{2t+2} \rangle$, in G from among all such paths (in other words, pick a random constraint of ψ^t). For $j \in \{1, 2, \ldots, 2t+1\}$, say that the jth edge in the path, namely (i_j, i_{j+1}), is *truthful* if y_{i_1} claims the plurality value for i_j and $y_{i_{2t+2}}$ claims the plurality value for i_{j+1}. Observe that if the path has an edge that is truthful and also lies in F, then by definition of F the constraint corresponding to that path is unsatisfied. Our goal is to show that at least ϵ' fraction of the paths have such edges.

The proof will follow the following sequence of claims:

Claim 22.10 *For each edge e of G and each $j \in \{1, 2, \ldots, 2t+1\}$,*

$$\Pr[e \text{ is the } j\text{'th edge of the path}] = \frac{1}{|E|} = \frac{2}{nd}.$$

PROOF: It is easily checked that in a d-regular graph if we take a random starting point i_1 and pick a random path of length $2t+1$ going of it, then the j'th edge on a random path is also a random edge of G. ■

The next claim shows that edges that are roughly in the middle of the path (specifically, in the interval of size $\delta\sqrt{t}$ in the middle) are quite likely to be truthful.

Claim 22.11 *Let $\delta < \frac{1}{100W}$. For each edge e of G and each $j \in \{t, t, \ldots, t+\delta\sqrt{t}\}$,*

$$\Pr[j\text{th edge of path is truthful} \mid e \text{ is the } j\text{th edge}] \geq \frac{1}{2W^2}.$$

PROOF: The main intuition is that since half the edges of G are self-loops, a random walk of length in $[t-\delta\sqrt{t}, t+\delta\sqrt{t}]$ is statistically very similar to a random walk of length t.

Formally, the lemma is proved by slightly inverting the viewpoint of how the path is picked. By the previous claim the set of walks of length $2t+1$ that contain $e = (i_j, i_{j+1})$ at the jth step can be generated by concatenating a random walk of length j out of i_j and a random walk of length $2t-j$ out of i_{j+1} (where the two walks are chosen independently). Let i_1 and i_{2t+2} denote the endpoints of these two walks. Then we need to show that

$$\Pr[y_{i_1} \text{ claims plurality value for } i_j \bigwedge y_{i_{2t+2}} \text{ claims plurality value for } i_{j+1}] \geq \frac{1}{2W^2}. \quad (3)$$

Since the plurality assignment was defined using walks of length exactly t, it follows that if j is precisely t, then the expression on the left hand side in (3) is at least $1/W \times 1/W = 1/W^2$. (This crucially uses that the walks to y_{i_1} and $y_{i_{2t+2}}$ are independently chosen.)

However, here j varies in $\{t, t+1, \ldots, t+\delta\sqrt{t}\}$, so these random walks have lengths between $t-\delta\sqrt{t}$ and $t+\delta\sqrt{t}$. We nevertheless show that the expression cannot be too different from $1/W^2$ for each j.

Since half the edges incident to each vertex are self-loops, we can think of an ℓ-step random walk from a vertex i as follows: **(1)** throw ℓ coins and let S_ℓ denote the number of the coins that came up "heads" **(2)** take S_ℓ "real" (non self-loop) steps on the graph. Recall that S_ℓ, the number of heads in ℓ tosses, is distributed according to the familiar *binomial distribution*.

It can be shown that the distributions S_t and $S_{t+\delta\sqrt{t}}$ are within statistical distance at most 10δ for every δ, t (see Exercise 22.3). In other words,

$$\frac{1}{2}\sum_m \left|\Pr[S_t = m] - \Pr[S_{t+\delta\sqrt{t}} = m]\right| \leq 10\delta.$$

It follows that the distribution of the endpoint of a t-step random walk out of e will be *statistically close* to the endpoint of a $(t+\delta\sqrt{t})$-step random walk, and the same is true for the $(t-\delta\sqrt{t})$-step random walk. Thus the expression on the left hand side of (3) is at least

$$(\frac{1}{W} - 10\delta)(\frac{1}{W} - 10\delta) \geq \frac{1}{2W^2},$$

which completes the proof. ■

Now let V be the random variable denoting the number of edges among the middle $\delta\sqrt{t}$ edges that are truthful and in F. Since it is enough for a path to contain one such edge for the corresponding constraint to be violated, our goal is to to show that $\Pr[V > 0] \geq \epsilon'$.

The previous two claims imply that the chance that any particular one of the edges in the interval of size $\delta\sqrt{t}$ is truthful and in F is $\frac{|F|}{|E|} \times \frac{1}{2W^2}$. Hence linearity of expectations implies:

$$\mathsf{E}[V] \geq \delta\sqrt{t} \times \frac{|F|}{|E|} \times \frac{1}{2W^2} = \frac{\delta\sqrt{t}\epsilon}{2W^2}\,.$$

This shows that $\mathsf{E}[V]$ is high, but we are not done since the expectation could be high and yet V could still be 0 for most of the walks. To rule this out, we consider the second moment. This calculation is the only place we use the fact that the contraint graph is an expander.

Claim 22.12 $\mathsf{E}[V^2] \leq 30\epsilon\delta\sqrt{t}d.$ ♢

PROOF: Let random variable V' denote the number of edges in the middle interval that are in F. Since V counts the number of edges that are in F *and* are truthful, $V \leq V'$. It suffices to show $E[V'^2] \leq 30\epsilon\delta\sqrt{t}d$. To prove this we use the mixing property of expanders and the fact that F contains ϵ fraction of the edges.

Specifically, for $j \in \left\{t, t, \ldots, t+\delta\sqrt{t}\right\}$ let I_j be an indicator random variable that is 1 if the jth edge is in F and 0 otherwise. Then $V' = \sum_{j\in\left\{t,t,\ldots,t+\delta\sqrt{t}\right\}} I_j$. Let S be the set of vertices that have at least one end point in F, implying $|S|/n \leq d\epsilon$.

$$\begin{aligned}
\mathsf{E}[V'^2] &= \mathsf{E}[\sum_{j,j'} I_j I_{j'}] \\
&= \mathsf{E}[\sum_{j} I_j^2] + \mathsf{E}[\sum_{j\neq j'} I_j I_{j'}] \\
&= \epsilon\delta\sqrt{t} + \mathsf{E}[\sum_{j\neq j'} I_j I_{j'}] \quad \text{(linearity of expectation and Claim 22.10)} \\
&= \epsilon\delta\sqrt{t} + 2\sum_{j<j'} \Pr[(j\text{th edge is in } F) \wedge (j'\text{th edge is in } F)] \\
&\leq \epsilon\delta\sqrt{t} + 2\sum_{j<j'} \Pr[(j\text{th vertex of walk lies in } S) \wedge (j'\text{th vertex of walk lies in } S)] \\
&\leq \epsilon\delta\sqrt{t} + 2\sum_{j<j'} \epsilon d(\epsilon d + (\lambda(G))^{j'-j}) \quad \text{(using (2))} \\
&\leq \epsilon\delta\sqrt{t} + 2\epsilon^2\delta\sqrt{t}d^2 + 2\epsilon\delta\sqrt{t}d\sum_{k\geq 1}(\lambda(G))^k \\
&\leq \epsilon\delta\sqrt{t} + 2\epsilon^2\delta\sqrt{t}d^2 + 20\epsilon\delta\sqrt{t}d \quad \text{(using } \lambda(G) \leq 0.9) \\
&\leq 30\epsilon\delta\sqrt{t}d \quad \text{(using } \epsilon < \tfrac{1}{d\sqrt{t}}\text{, an assumption of Lemma 22.9)}\,.
\end{aligned}$$

■

Finally, since $\Pr[V > 0] \geq \mathsf{E}[V]^2/\mathsf{E}[V^2]$ for any nonnegative random variable (see Exercise 22.4), we conclude that $\Pr[V > 0] \geq \frac{\sqrt{t}}{10^5 dW^4}\epsilon = \epsilon'$, and Lemma 22.9 is proved. ■

22.2.5 Alphabet Reduction: Proof of Lemma 22.6

Interestingly, the main component in the proof of Lemma 22.6 is the exponential-sized **PCP** of Section 11.5 (An alternative proof is explored in Exercise 22.5.)

Let φ be a 2CSP instance as in the statement of Lemma 22.6, with n variables $u_1, u_2, \ldots, u_n$, alphabet $\{0..W\text{–}1\}$ and m constraints $C_1, C_2, \ldots, C_m$. Think of each variable as taking values

that are bit strings in $\{0,1\}^{\log W}$. Then if constraint C_s involves variables say u_i, u_j we may think of it as a circuit applied to the bit strings representing u_i, u_j where the constraint is said to be satisfied iff this circuit outputs 1. Say ℓ is an upper bound on the size of this circuit over all constraints. Note that ℓ is at most $2^{2\log W} < W^4$. We will assume without loss of generality that all circuits have the same size.

The idea in alphabet reduction will be to write a small CSP instance for each of these circuits, and replace each old variable by a set of new variables. This technique from [AS92] was called *verifier composition*, and more recently, a variant was called *PCP's of proximity,* and both names stem from the "proof verification" view of **PCP**'s (see Section 11.2). We state the result (a simple corollary of Theorem 11.19) first in the verification viewpoint and then translate into the CSP viewpoint.

Corollary 22.13 *(**PCP** of proximity) There exists a verifier V that given any circuit C with $2k$ input wires and size ℓ has the following property:*

1. *If $\mathbf{u_1}, \mathbf{u_2}$ are strings of k bits each such that $\mathbf{u_1} \circ \mathbf{u_2}$ is a satisfying assignment for circuit C, then there is a string π_3 of size $2^{\mathrm{poly}(\ell)}$ such that V accepts $\mathsf{WH}(\mathbf{u_1}) \circ \mathsf{WH}(\mathbf{u_2}) \circ \pi_3$ with probability* 1.

2. *For every three bit strings π_1, π_2, π_3, where π_1 and π_2 have size 2^k, if V accepts $\pi_1 \circ \pi_2 \circ \pi_3$ with probability at least $1/2$, then π_1, π_2 are 0.99-close to $\mathsf{WH}(\mathbf{u_1})$, $\mathsf{WH}(\mathbf{u_2})$ respectively for some k-bit strings $\mathbf{u_1}, \mathbf{u_2}$ where $\mathbf{u_1} \circ \mathbf{u_2}$ is a satisfying assignment for circuit C.*

3. *V runs in $\mathrm{poly}(\ell)$ time, uses $\mathrm{poly}(\ell)$ random bits and examines only $O(1)$ bits in the provided strings.* ♢

Before giving the proof, we describe how it allows us to do alphabet reduction, as promised. First we note that in the CSP viewpoint of Corollary 22.13,(see Table 11.1) the variables are the bits of π_1, π_2, π_3, and V can be represented as a CSP instance of size $2^{\mathrm{poly}(\ell)}$ in these new variables. The arity of the constraints is the number of bits that the verifier reads in the proof, which is *some fixed constant independent of W and ϵ*. The fraction of satisfied constraints is the acceptance probability of the verifier.

Returning to the instance whose alphabet size we want to reduce, we replace each original variable u_i from the alphabet $\{0, \ldots, W-1\}$ by a sequence $U_i = (U_{i,1}, \ldots, U_{i,2^W})$ of 2^W binary-valued variables, which in a valid assignment would be an encoding of u_i using the Walsh-Hadamard code. For each old constraint $C_s(u_i, u_j)$ we apply the constraint satisfaction view of Corollary 22.13, using C_s as the circuit whose assignment is being verified. Thus for each original constraint C_s we have a vector of $2^{\mathrm{poly}(\ell)}$ new binary-valued variables Π_s, which plays the role of π_3 in Corollary 22.13, whereas U_i, U_j play the role of π_1, π_2 respectively. The set of new constraints corresponding to C_s is denoted $\mathcal{C}_s$. As already noted the arity of the new constraints is some fixed constant independent of W, ϵ.

The overall CSP instance is the union of these constraints $\cup_{s=1}^{m} \mathcal{C}_s$; see Figure 22.4. Clearly, if the old instance was satisfiable then so is this union. Now we show that if some assignment satisfies more than $1 - \epsilon/3$ fraction of the new constraints, then we can construct an assignment for the original instance that satisfies more than $1-\epsilon$ fraction of its constraints. This is done by "decoding" the assignment for each each set of new variables U_i by the following rule: if U_i is 0.99-close to some linear function $\mathsf{WH}(a_i)$ then use a_i as the assignment for the old variable u_i, and otherwise use an arbitrary string. Now consider how well we did on any old constraint $C_s(u_i, u_j)$. If the decodings a_i, a_j of U_i, U_j do not satisfy C_s then Corollary 22.13 implies that at least $1/2$ the constraints of $\mathcal{C}_s$ were not satisfied anyway. Thus if δ is the fraction of old constraints that are not satisfied, then $\delta/2 \leq \epsilon/3$, implying $\delta < 2\epsilon/3$, and the Lemma is proved.

To finish, we prove Corollary 22.13.

PROOF: (of Corollary 22.13) The proof uses the reduction from CKT-SAT to QUADEQ (see Section 11.5.2 and Exercise 11.15). This reduction transforms a circuit C with ℓ wires (where "inputs" are considered as wires in the circuit) to an instance of QUADEQ of with ℓ variables and $O(\ell)$ equations where the variables in the QUADEQ instance correspond to

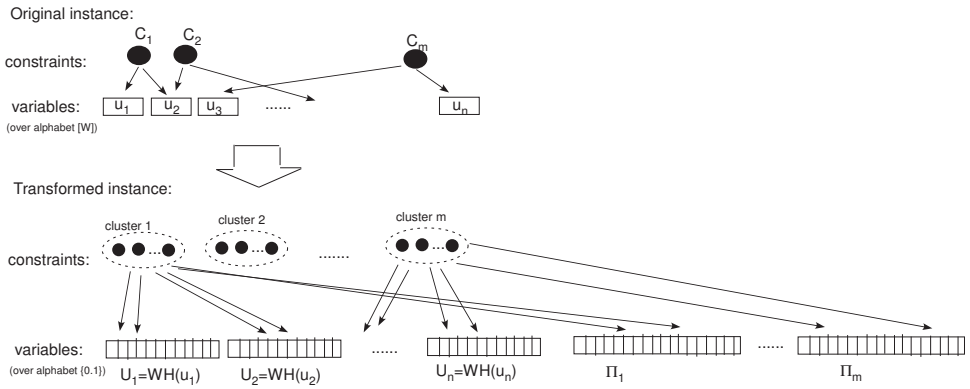


Figure 22.4 The alphabet reduction transformation maps a 2CSP instance φ over alphabet $\{0..W\text{–}1\}$ into a qCSP instance ψ over the binary alphabet. Each variable of φ is mapped to a block of binary variables that in the correct assignment will contain the Walsh-Hadamard encoding of this variable. Each constraint C_ℓ of φ depending on variables u_i, u_j is mapped to a cluster of constraints corresponding to all the **PCP** of proximity constraints for C_ℓ. These constraint depend on the encoding of u_i and u_j, and on additional auxiliary variables that in the correct assignment contain the **PCP** of proximity proof that these are indeed encoding of values that make the constraint C_ℓ true.

values of wires in the circuit. Thus every solution to the QUADEQ instance has ℓ bits, of which the first k bits give a satisfying assignment to the circuit.

The verifier expects π_3 to contain whatever our verifier of Theorem 11.19 expects in the proof for this instance of QUADEQ, namely, a linear function f that is $\mathsf{WH}(w)$, and another linear function g that is $\mathsf{WH}(w \otimes w)$ where w satisfies the QUADEQ instance. The verifier checks these functions as described in the proof of Theorem 11.19.

However, in the current setting our verifer is also given strings π_1, π_2, which we think of as functions $\pi_1, \pi_2 : \mathrm{GF}(2)^k \to \mathrm{GF}(2)$. The verifier checks that both are 0.99-close to linear functions, say $\tilde{\pi_1}, \tilde{\pi_2}$. Then to check that $\tilde{f}$ encodes a string whose first $2k$ bits are the same as the string encoded by $\tilde{\pi_1}, \tilde{\pi_2}$, the verifier does the following *concatenation test*, which uses the properties of the Walsh-Hadamard code.

Concatenation test. We are given three linear functions π_1, π_2, f that encode strings of lengths k, k, and ℓ respectively. Denoting by $\mathbf{u}$ and $\mathbf{v}$ the strings encoded by π_1, π_2 respectively (that is, $\pi_1 = \mathsf{WH}(\mathbf{u})$ and $\pi_2 = \mathsf{WH}(\mathbf{v})$), and by w the string encoded by f, we have to check by examining only $O(1)$ bits in these functions that $\mathbf{u} \circ \mathbf{v}$ is the same as the first $2k$ bits of w. By the random subsum principle, the following simple test rejects with probability $1/2$ if this is not the case. Pick random $\mathbf{x}, \mathbf{y} \in \{0,1\}^k$, and denote by $\mathbf{XY} \in \mathrm{GF}(2)^\ell$ the string whose first k bits are $\mathbf{x}$, the next k bits are $\mathbf{y}$ and the remaining bits are all 0. Accept if $f(\mathbf{XY} = \pi_1(\mathbf{x}) + \pi_2(\mathbf{y})$ and else reject. ■

22.3 Hardness of 2CSP$_W$: Tradeoff between gap and alphabet size

The problem 2CSP$_W$ often plays a role in proofs of advanced **PCP** theorems. The (standard) **PCP** theorem implies that there is some constant W and some $\nu < 1$ such that computing ν-approximation to 2CSP$_W$ is **NP**-hard (see Definition 22.1).

Corollary 22.14 *(of* **PCP** *Theorem) There is some $\nu < 1$ and some W such that GAP* 2CSP$_W(\nu)$ *is* **NP***-hard.* ♢

For advanced **PCP** theorems we would like to prove the same result for smaller ν, without making W too large. (Note: if W is allowed to be $\exp(n)$ then the problem is **NP**-hard

even for $\nu = 0$!) At first glance the "gap amplification" of Lemma 22.5 seems relevant, but that doesn't suffice because first, it cannot lower ν below some fixed constant, and second, because it greatly increases the alphabet size. The next theorem gives the best tradeoff possible (up to the value of c) between these two parameters. For further constructions of **PCP**'s, it is useful to restrict attention to a special subclass of 2CSP instances, which have the so-called *projection* property. This means that for each constraint $\varphi_r(y_1, y_2)$ and each value of y_1, there is a *unique* value of y_2 such that $\varphi_r(y_1, y_2) = 1$. Another way to state this is that for each constraint φ_r there is a function $h : [W] \to [W]$ such that the constraint is satisfied by (u, v) iff $h(u) = v$.

A 2CSP instance is said to be *regular* if every variable appears in the same number of constraints.

Theorem 22.15 *(Raz* [Raz95b]*) There is a $c > 1$ such that for every $t > 1$, GAP $2\mathsf{CSP}_W(\epsilon)$ is* **NP***-hard for $\epsilon = 2^{-t}, W = 2^{ct}$, and this is true also for* 2CSP *instances that are regular and have the projection property.* ♢

A weaker version of this theorem, with a somewhat simpler proof, was obtained by Feige and Kilian [FK93]. This weaker version is sufficient for many applications, including for Håstad's 3-bit **PCP** theorem (see Section 22.4 below).

22.3.1 Idea of Raz's proof: Parallel Repetition

Let φ be the $2\mathsf{CSP}_W$ instances produced by the reduction of Corollary 22.14. For some $\nu < 1$ it has the property that either $\mathsf{val}(\varphi) = 1$ or $\mathsf{val}(\varphi) = \nu < 1$ but deciding which case holds is hard. There is an obvious "powering" idea for trying to lower the gap while maintaining the arity at 2. Let φ^{*t} denote the following instance. Its variables are t-tuples of variables of φ. Its constraints correspond to t-tuples of constraints, in the sense that for every t-tuple of constraints $\varphi_1(y_1, z_1), \varphi_2(y_2, 2), \ldots, \varphi_t(y_t, z_t)$ the new instance has a constraint of arity 2 involving the new variables $(y_1, y_2, \ldots, y_t)$ and $(z_1, z_2, \ldots, z_t)$ and the Boolean function describing this constraint is simply

$$\bigwedge_{i=1}^{t} \varphi_i(y_i, z_i).$$

(To put it in words, the new constraint is satisfied iff all the t constituent constraints are.)

In the verification viewpoint, this new 2CSP instance corresponds to running the verifier in parallel t times, hence Raz's theorem is also called the *parallel repetition theorem*.

It is easy to convert a satisfying assignment for φ into one for φ^{*t} by taking t-tuples of the values. Furthermore, given an assignment for φ that satisfies ν fraction of the constraints, it is easy to see that the assignment that forms t-tuples of these values satisfies at least ν^t fraction of the constraints of φ^{*t}. It seemed "obvious" to researchers that no assignment can do better. Then a simple counterexample was found, whereby more than ν^t fraction of constraints in φ^{*t} could be satisfied (see Exercise 22.6). Raz shows, however, that no assignment can satisfy more than ν^{ct} fraction of the constraints of φ^{*t}, where c depends upon the alphabet size W. The proof is quite difficult, though there have been some simplifications (see the chapter notes and the book's web site).

22.4 Håstad's 3-bit PCP Theorem and hardness of MAX-3SAT

In Chapter 11 we showed $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$, in other words certificates for membership in **NP** languages can be checked by examining $O(1)$ bits in them. Now we are interested in keeping the number of query bits as low as possible, *while keeping the soundness around* 1/2. The next Theorem shows that the number of query bits can be reduced to 3, and

furthermore the verifier's decision process consists of simply looking at the parity of these three bits.

Theorem 22.16 *(Håstad's 3-bit* **PCP** [Hås97]*)*
For every $\delta > 0$ and every language $L \in \mathbf{NP}$ there is a **PCP***-verifier V for L making three (binary) queries having completeness parameter $1-\delta$ and soundness parameter at most $1/2 + \delta$.*
Moreover, the tests used by V are linear. *That is, given a proof $\pi \in \{0,1\}^m$, V chooses a triple $(i_1, i_2, i_3) \in [m]^3$ and $b \in \{0,1\}$ according to some distribution and accepts iff $\pi_{i_1} + \pi_{i_2} + \pi_{i_3} = b \pmod 2$.*

22.4.1 Hardness of approximating MAX-3SAT

We first note that Theorem 22.16 is intimately connected to the hardness of approximating a problem called MAX-E3LIN, which is a subcase of $3\mathsf{CSP}_2$ in which the constraints specify the *parity* of triples of variables. Another way to think of such an instance is that it gives a set of linear equations mod 2 where each equation has at most 3 variables. We are interested in determining the largest subset of equations that are simultaneously satisfiable. We claim that Theorem 22.16 implies that $(1/2+\nu)$-approximation to this problem is **NP**-hard for every $\nu > 0$. This is a *threshold result* since the problem has a simple 1/2-approximation algorithm. (It uses observations similar to those we made in context of MAX-3SAT in Chapter 11; a random assignment satisfies, in the expectation, half of the constraints, and this observation can be turned into a deterministic algorithm that satisfies at least 1/2 of the equations.)

To prove our claim about the hardness of MAX-E3LIN, we convert the verifier of Theorem 22.16 into an equivalent CSP by the recipe of Section 11.3. Since the verifier imposes parity constraints on triples of bits in the proof, the equivalent CSP instance is an instance of MAX-E3LIN where either $1-\delta$ fraction of the constraints are satisfiable, or at most $1/2+\delta$ are. Since distinguishing between the two cases is **NP**-hard, we conclude that it is **NP**-hard to compute a ρ-approximation to MAX-E3LIN where $\rho = \frac{1/2+\delta}{1-\delta}$. Since $\delta > 0$ is allowed to be arbitrarily small, ρ can be arbitrarily close to 1/2 and we conclude that $(1/2+\nu)$-approximation is **NP**-hard for every $\nu > 0$.

Also note that the fact that completeness is strictly less than 1 in Theorem 22.16 is inherent if $\mathbf{P} \neq \mathbf{NP}$, since determining if there is a solution satisfying *all* of the equations (in other words, the satisfiability problem for MAX-E3LIN) is possible in polynomial time using Gaussian elimination

Now we prove a hardness result for MAX-3SAT, which as mentioned earlier, is also a threshold result.

Corollary 22.17 *For every $\epsilon > 0$, computing $(7/8+\epsilon)$-approximation to* MAX-3SAT *is* **NP***-hard.* ♢

Proof: We reduce MAX-E3LIN to MAX-3SAT. Take the instance of MAX-E3LIN produced by the above reduction, where we are interested in determining whether $(1-\nu)$ fraction of the equations can be satisfied or at most $1/2+\nu$ are. Represent each linear constraint by four $3CNF$ clauses in the obvious way. For example, the linear constraint $a+b+c=0 \pmod 2$ is equivalent to the clauses $(\overline{a} \vee b \vee c), (a \vee \overline{b} \vee c), (a \vee b \vee \overline{c}), (\overline{a} \vee \overline{b} \vee \overline{c})$. If a, b, c satisfy the linear constraint, they satisfy all 4 clauses and otherwise they satisfy at most 3 clauses. We conclude that in one case at least $(1-\epsilon)$ fraction of clauses are simultaneously satisfiable, and in the other case at most $1 - (\frac{1}{2} - \nu) \times \frac{1}{4} = \frac{7}{8} + \frac{\nu}{4}$ fraction are. The ratio between the two cases tends to 7/8 as ν decreases. Since Theorem 22.16 implies that distinguishing between the two cases is **NP**-hard for every constant ν, the result follows. ■

22.5 Tool: the Fourier transform technique

Theorem 22.16 is proved using Fourier analysis. The continuous Fourier transform is extremely useful in mathematics and engineering. Likewise, the discrete Fourier transform has found many uses in algorithms and complexity, in particular for constructing and analyzing **PCP**'s. The Fourier transform technique for **PCP**'s involves calculating the maximum acceptance probability of the verifier using Fourier analysis of the functions presented in the proof string. (See Note 22.21 for a broader perspective of uses of discrete Fourier transforms in combinatorial and probabilistic arguments.) It is delicate enough to give "tight" inapproximability results for MAX-INDSET, MAX-3SAT, and many other problems.

To introduce the technique we start with a simple example: analysis of the linearity test over GF(2) (i.e., proof of Theorem 11.21). We then introduce the *Long Code* and show how to test for membership in it. These ideas are then used to prove Håstad's 3-bit PCP Theorem.

22.5.1 Fourier transform over $\mathrm{GF}(2)^n$

The Fourier transform over $\mathrm{GF}(2)^n$ is a tool to study functions on the Boolean hypercube. In this chapter, it will be useful to use the set $\{+1,-1\} = \{\pm 1\}$ instead of $\{0,1\}$. To transform $\{0,1\}$ to $\{\pm 1\}$, we use the mapping $b \mapsto (-1)^b$ (i.e., $0 \mapsto +1$, $1 \mapsto -1$). Thus we write the hypercube as $\{\pm 1\}^n$ instead of the more usual $\{0,1\}^n$. Note this maps the XOR operation (i.e., addition in GF(2)) into the multiplication operation over $\mathbb{R}$.

The set of functions from $\{\pm 1\}^n$ to $\mathbb{R}$ defines a 2^n-*dimensional Hilbert space* (i.e., a vector space with an associated inner product) as follows. Addition and multiplication by a scalar are defined in the natural way: $(f+g)(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x})$ and $(\alpha f)(\mathbf{x}) = \alpha f(\mathbf{x})$ for every $f, g : \{\pm 1\}^n \to \mathbb{R}$, $\alpha \in \mathbb{R}$. We define the inner product of two functions f, g, denoted $\langle f, g \rangle$, to be $\mathsf{E}_{\mathbf{x} \in \{\pm 1\}^n}[f(\mathbf{x})g(\mathbf{x})]$. (This is the *expectation inner product*.)

The *standard basis* for this space is the set $\{\mathbf{e_x}\}_{\mathbf{x} \in \{\pm 1\}^n}$, where $\mathbf{e_x}(\mathbf{y})$ is equal to 1 if $\mathbf{y} = \mathbf{x}$, and equal to 0 otherwise. This is an orthogonal basis, and every function $f : \{\pm 1\}^n \to \mathbb{R}$ can be represented in this basis as $f = \sum_{\mathbf{x}} a_{\mathbf{x}} \mathbf{e_x}$. For every $\mathbf{x} \in \{\pm 1\}^n$, the coefficient $a_{\mathbf{x}}$ is equal to $f(\mathbf{x})$.

The *Fourier basis* is an alternative orthonormal basis that contains, for every subset $\alpha \subseteq [n]$, a function χ_α where $\chi_\alpha(\mathbf{x}) = \prod_{i \in \alpha} x_i$. (We define $\chi_\emptyset$ to be the function that is 1 everywhere). This basis is actually the Walsh-Hadamard code (see Section 11.5.1) in disguise: the basis vectors correspond to the *linear* functions over GF(2). To see this, note that every linear function of the form $\mathbf{b} \mapsto \mathbf{a} \odot \mathbf{b}$ (with $\mathbf{a}, \mathbf{b} \in \{0,1\}^n$) is mapped by our transformation to the function taking $\mathbf{x} \in \{\pm 1\}^n$ to $\prod_{i \text{ s.t. } a_i = 1} x_i$. To check that the Fourier basis is indeed an orthonormal basis for $\mathbb{R}^{2^n}$, note that the random subsum principle implies that for every $\alpha, \beta \subseteq [n]$, $\langle \chi_\alpha, \chi_\beta \rangle = \delta_{\alpha,\beta}$ where $\delta_{\alpha,\beta}$ is equal to 1 iff $\alpha = \beta$ and equal to 0 otherwise.

Remark 22.18
Note that in the $\{-1,1\}$ view, the basis functions can be viewed as *multilinear polynomials* (i.e., multivariate polynomials whose degree in each variable is 1). Thus the fact that every real-valued function $f : \{-1,1\}^n$ has a Fourier expansion can also be phrased as "Every such function can be represented by a multilinear polynomial." This is very much in the same spirit as the polynomial representations used in Chapters 8 and 11.

Since the Fourier basis is an orthonormal basis, every function $f : \{\pm 1\}^n \to \mathbb{R}$ can be represented as $f = \sum_{\alpha \subseteq [n]} \hat{f}_\alpha \chi_\alpha$. We call $\hat{f}_\alpha$ the α^{th} *Fourier coefficient* of f. We will often use the following simple lemma:

Lemma 22.19 *Every two functions $f, g : \{\pm 1\}^n \to \mathbb{R}$ satisfy*

1. $\langle f, g \rangle = \sum_\alpha \hat{f}_\alpha \hat{g}_\alpha$.
2. (Parseval's Identity) $\langle f, f \rangle = \sum_\alpha \hat{f}_\alpha^2$.

PROOF: The second property follows from the first. To prove the first we expand

$$\langle f,g\rangle \;=\; \langle \sum_{\alpha}\hat{f}_\alpha\chi_\alpha, \sum_{\beta}\hat{g}_\beta\chi_\beta\rangle \;=\; \sum_{\alpha,\beta}\hat{f}_\alpha\hat{g}_\beta\langle\chi_\alpha,\chi_\beta\rangle \;=\; \sum_{\alpha,\beta}\hat{f}_\alpha\hat{g}_\beta\delta_{\alpha,\beta} \;=\; \sum_{\alpha}\hat{f}_\alpha\hat{g}_\alpha$$

■

Example 22.20

Some examples for the Fourier transform of particular functions:

1. The majority function on 3 variables (i.e., the function $MAJ(u_1,u_2,u_3)$ that outputs $+1$ if and only if at least two of its inputs are $+1$, and -1 otherwise) can be expressed as $1/2u_1 + 1/2u_2 + 1/2u_3 - 1/2u_1u_2u_3$. Thus, it has four Fourier coefficients equal to $1/2$ and the rest are equal to zero.
2. If $f(u_1,u_2,\ldots,u_n) = u_i$ (i.e., f is a *coordinate function*, a concept we will see again in context of long codes) then $f = \chi_{\{i\}}$ and so $\hat{f}_{\{i\}} = 1$ and $\hat{f}_\alpha = 0$ for $\alpha \neq \{i\}$.
3. If f is a random Boolean function on n bits, then each $\hat{f}_\alpha$ is a random variable that is a sum of 2^n binomial variables (equally likely to be $1,-1$) and hence looks like a normally distributed variable with standard deviation $2^{n/2}$ and mean 0. Thus with high probability, all 2^n Fourier coefficients have values in $[-\frac{\mathrm{poly}(n)}{2^{n/2}}, \frac{\mathrm{poly}(n)}{2^{n/2}}]$.

22.5.2 The connection to PCPs: High level view

In the PCP context we are interested in *Boolean-valued* functions, i.e., those from $GF(2)^n$ to $GF(2)$. Under our transformation they turn into functions from $\{\pm1\}^n$ to $\{\pm1\}$. Thus, we say that $f:\{\pm1\}^n \to \mathbb{R}$ is *Boolean* if $f(\mathbf{x}) \in \{\pm1\}$ for every $\mathbf{x} \in \{\pm1\}^n$. Note that if f is Boolean then $\langle f,f\rangle = \mathsf{E}_{\mathbf{x}}[f(\mathbf{x})^2] = 1$.

On a high level, we use the Fourier transform in the soundness proofs for PCP's to show that if the verifier accepts a proof π with high probability then π is "close to" being "well-formed" (where the precise meaning of "close-to" and "well-formed" is context dependent). Usually we relate the acceptance probability of the verifier to an expectation of the form $\langle f,g\rangle = \mathsf{E}_{\mathbf{x}}[f(\mathbf{x})g(\mathbf{x})]$, where f and g are Boolean functions arising from the proof. We then use techniques similar to those used to prove Lemma 22.19 to relate this acceptance probability to the Fourier coefficients of f,g, allowing us to argue that if the verifier's test accepts with high probability, then f and g have few relatively large Fourier coefficients. This will provide us with some nontrivial useful information about f and g, since in a "generic" or random function, all the Fourier coefficient are small and roughly equal.

22.5.3 Analysis of the linearity test over $GF(2)$

We will now prove Theorem 11.21, thus completing the proof of the **PCP** Theorem. Recall that the linearity test is provided a function $f:\mathrm{GF}(2)^n \to \mathrm{GF}(2)$ and has to determine whether f has significant agreement with a linear function. To do this it picks $\mathbf{x},\mathbf{y} \in \mathrm{GF}(2)^n$ randomly and accepts iff $f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$.

Now we rephrase this test using $\{\pm1\}$ instead of GF(2), so linear functions turn into Fourier basis functions. For every two vectors $\mathbf{x},\mathbf{y} \in \{\pm1\}^n$, we denote by $\mathbf{xy}$ their componentwise multiplication. That is, $\mathbf{xy} = (x_1y_1,\ldots,x_ny_n)$. Note that for every basis function $\chi_\alpha(\mathbf{xy}) = \chi_\alpha(\mathbf{x})\chi_\alpha(\mathbf{y})$.

For two Boolean functions f,g, their inner product $\langle f,g\rangle$ is equal to the fraction of inputs on which they *agree* minus the fraction of inputs on which they *disagree*. It follows

that for every $\epsilon \in [0,1]$ and functions $f, g : \{\pm 1\}^n \to \{\pm 1\}$, f has agreement $\frac{1}{2} + \frac{\epsilon}{2}$ with g iff $\langle f, g \rangle = \epsilon$. Thus, if f has a large Fourier coefficient then it has significant agreement with some Fourier basis function, or in the GF(2) worldview, f is close to some linear function. This means that Theorem 11.21 concerning the correctness of the linearity test can be rephrased as follows:

Theorem 22.22 *Suppose that* $f : \{\pm 1\}^n \to \{\pm 1\}$ *satisfies* $\Pr_{\mathbf{x},\mathbf{y}}[f(\mathbf{xy}) = f(\mathbf{x})f(\mathbf{y})] \geq \frac{1}{2} + \epsilon$. *Then, there is some* $\alpha \subseteq [n]$ *such* $\hat{f}_\alpha \geq 2\epsilon$. ♢

PROOF: We can rephrase the hypothesis as $E_{\mathbf{x},\mathbf{y}}[f(\mathbf{xy})f(\mathbf{x})f(\mathbf{y})] \geq (\frac{1}{2} + \epsilon) - (\frac{1}{2} - \epsilon) = 2\epsilon$. We note that from now on we do not need f to be Boolean, but merely to satisfy $\langle f, f \rangle = 1$.

Expressing f by its Fourier expansion,

$$2\epsilon \leq \mathop{\mathsf{E}}_{\mathbf{x},\mathbf{y}}[f(\mathbf{xy})f(\mathbf{x})f(\mathbf{y})] = \mathop{\mathsf{E}}_{\mathbf{x},\mathbf{y}}[(\sum_\alpha \hat{f}_\alpha \chi_\alpha(\mathbf{xy}))(\sum_\beta \hat{f}_\beta \chi_\beta(\mathbf{x}))(\sum_\gamma \hat{f}_\gamma \chi_\gamma(\mathbf{y}))].$$

Since $\chi_\alpha(\mathbf{x}y) = \chi_\alpha(\mathbf{x})\chi_\alpha(\mathbf{y})$ this becomes

$$= \mathop{\mathsf{E}}_{\mathbf{x},\mathbf{y}}[\sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \chi_\alpha(\mathbf{x})\chi_\alpha(\mathbf{y})\chi_\beta(\mathbf{x})\chi_\gamma(\mathbf{y})].$$

Using linearity of expectation:

$$\begin{aligned} &= \sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \mathop{\mathsf{E}}_{\mathbf{x},\mathbf{y}}[\chi_\alpha(\mathbf{x})\chi_\alpha(\mathbf{y})\chi_\beta(\mathbf{x})\chi_\gamma(\mathbf{y})] \\ &= \sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \mathop{\mathsf{E}}_{\mathbf{x}}[\chi_\alpha(\mathbf{x})\chi_\beta(\mathbf{x})] \mathop{\mathsf{E}}_{\mathbf{y}}[\chi_\alpha(\mathbf{y})\chi_\gamma(\mathbf{y})] \\ &\qquad \text{(because } \mathbf{x}, \mathbf{y} \text{ are independent).} \end{aligned}$$

By orthonormality $\mathsf{E}_{\mathbf{x}}[\chi_\alpha(\mathbf{x})\chi_\beta(\mathbf{x})] = \delta_{\alpha,\beta}$, so we simplify to

$$\begin{aligned} &= \sum_\alpha \hat{f}_\alpha^3 \\ &\leq (\max_\alpha \hat{f}_\alpha) \times (\sum_\alpha \hat{f}_\alpha^2) = \max_\alpha \hat{f}_\alpha \,, \end{aligned}$$

since $\sum_\alpha \hat{f}_\alpha^2 = \langle f, f \rangle = 1$. Hence $\max_\alpha \hat{f}_\alpha \geq 2\epsilon$ and the theorem is proved. ■

22.6 Coordinate functions, Long code and its testing

Håstad's 3-bit PCP Theorem uses a coding method called the *long code*. Let $W \in \mathbb{N}$. We say that $f : \{\pm 1\}^W \to \{\pm 1\}$ is a *coordinate function* if there is some $w \in [W]$, such that $f(x_1, x_2, \ldots, x_W) = x_w$; in other words, $f = \chi_{\{w\}}$.[1] (Aside: Unlike the previous section, here we use W instead of n for the number of variables; the reason is to be consistent with our use of W for the alphabet size in $2\mathsf{CSP}_W$ in Section 22.7.)

Definition 22.23 *(Long Code)* The *long code* for $[W]$ encodes each $w \in [W]$ by the table of all values of the function $\chi_{\{w\}} : \{\pm 1\}^{[W]} \to \{\pm 1\}$. ♢

[1] Some texts call such a function a *dictatorship* function, since one variable ("the dictator") completely determines the outcome. The name comes from social choice theory, which studies different election setups. That field has also been usefully approached using Fourier analysis ideas described in Note 22.21.

Note 22.21 *(Self-correlated functions, isoperimetry, phase transitions)*

Although it is surprising to see Fourier transforms used in proofs of PCP Theorems, in retrospect this is quite natural. We try to put this in perspective, and refer the reader to the survey of Kalai and Safra [KS06] and the web-based lecture notes of O'Donnell, Mossell and others for further background on this topic.

Classically, Fourier tranforms are very useful in proving results of the following form: "If a function is correlated with itself in some structured way, then it belongs to some small family of functions." In the PCP setting, the "self-correlation" of a function $f:\{0,1\}^n \to \{0,1\}$ means that if we run some designated verifier on f that examines only a few bits in it, then this verifier accepts with reasonable probability. For example, in the linearity test over GF(2), the acceptance probability of the test is $E_{x,y}[I_{x,y}]$ where $I_{x,y}$ is an indicator random variable for the event $f(x)+f(y)=f(x+y)$.

Another classical use of Fourier transforms is study of *Isoperimetry*, which is the study of subsets of "minimum surface area." A simple example is the fact that of all connected regions in $\mathbb{R}^2$ with a specified area, the circle has the minimum perimeter. Again, isoperimetry can be viewed as a study of "self-correlation", by thinking of the characteristic function of the set in question, and realizing that the "perimeter" of "surface" of the set consists of points in space where taking a small step in some direction causes the value of this function to switch from 1 to 0.

Håstad's "noise" operator of Section 22.7 appears in works of mathematicians Nelson, Bonamie, Beckner, and others on *hypercontractive estimates*, and the general theme is again one of identifying properties of functions based upon their "self-correlation" behavior. One considers the correlation of the function f with the function $T_\rho(f)$ obtained by (roughly speaking) computing at each point the average value of f in a small ball around that point. One can show that the norms of f and $T_\rho(f)$ are related — not used in Håstad's proof but very useful in the **PCP** Theorems surveyed in Section 22.9; see also Exercise 22.10 for a small taste.

Fourier transforms and especially hypercontractivity estimates have also proved useful in study of *phase transitions* in random graphs (e.g., see Friedgut [Fri99]). The simplest case is the graph model $G(n,p)$ whereby each possible edge is included in the graph independently with probability p. A *phase transition* is a value of p at which the graph goes from almost never having a certain property to almost always having that property. For example, it is known that there is some constant c such that around $p = c\log n/n$ the probability of the graph being connected suddenly jumps from close to 0 to close to 1. Fourier transforms are useful to study phase transition because a phase transition is as an isoperimetry problem on a "Graph" (with a capital G) where each "Vertex" is an n-vertex graph, and an "Edge" between two "Vertices" means that one of the graphs is obtained by adding a few edges to the graph. Note that adding a few edges corresponds to raising the value of p by a little.

Finally, we mention some interesting uses of Fourier transforms in the results mentioned in Sections 22.9.4 and 22.9.5. These involve isoperimetry on the hypercube $\{0,1\}^n$. One can study isoperimetry in a graph setting by defining "surface area" of a subset of vertices as the "number of edges leaving the set," or some other notion, and then try to study isoperimetry in such settings. The Fourier transform can be used to prove isoperimetry theorems about hypercube and hypercube-like graphs. The reason is that a subset $S \subseteq \{0,1\}^n$ is nothing but a Boolean function that is 1 on S and -1 elsewhere. Assuming the graph is D-regular, and $|S| = 2^{n-1}$

$$E_{(x,y):\ \mathrm{edge}}[f(x)f(y)] = \frac{1}{2^n D}\left(|E(S,S)| + \left|\overline{S},\overline{S}\right| - 2\left|E(S,\overline{S})\right|\right),$$

which implies that the fraction of edges of S that leave the set is $1/2 - E[f(x)f(y)]/2$. This kind of expression can be analysed using the Fourier transform; see Exercise 22.11.b.

Note that w, normally written using $\log W$ bits, is being represented using a table of 2^W bits, a doubly exponential blowup! This inefficiency is the reason for calling the code "long."

The problem of testing for membership in the Long Code is defined by analogy to the earlier test for the Walsh-Hadamard code. We are given a function $f:\{\pm 1\}^W \to \{\pm 1\}$, and wish to determine if f has good agreement with $\chi_{\{w\}}$ for some w, namely, whether $\hat{f}_{\{w\}}$ is significant. Such a test is described in Exercise 22.5, but it is not sufficient for the proof of Håstad's Theorem, which requires a test using only *three* queries. Below we show such a three query test albeit at the expense of achieving the following weaker guarantee: if the test passes with high probability then f has a good agreement with a function χ_α where $|\alpha|$ is small (but not necessarily equal to 1). This weaker conclusion will be sufficient in the proof of Theorem 22.16.

Let $\rho > 0$ be some arbitrarily small constant. The test picks two uniformly random vectors $\mathbf{x}, \mathbf{y} \in_R \{\pm 1\}^W$ and then a vector $\mathbf{z} \in \{\pm 1\}^W$ according to the following distribution: for every coordinate $i \in [W]$, with probability $1-\rho$ we choose $z_i = +1$ and with probability ρ we choose $z_i = -1$. Thus with high probability, about ρ fraction of coordinates in $\mathbf{z}$ are -1 and the other $1-\rho$ fraction are $+1$. We think of $\mathbf{z}$ as a "noise" vector. The test accepts iff $f(\mathbf{x})f(\mathbf{y}) = f(\mathbf{xyz})$. Note that the test is similar to the linearity test except for the use of the noise vector $\mathbf{z}$.

Suppose $f = \chi_{\{w\}}$. Then since $b \cdot b = 1$ for $b \in \{\pm 1\}$,

$$f(\mathbf{x})f(\mathbf{y})f(\mathbf{xyz}) = x_w y_w (x_w y_w z_w) = 1 \cdot z_w.$$

Hence the test accepts iff $z_w = 1$ which happens with probability $1-\rho$. We now prove a certain converse:

Lemma 22.24 *If the test accepts with probability $1/2 + \delta$ then $\sum_\alpha \hat{f}_\alpha^3 (1-2\rho)^{|\alpha|} \geq 2\delta$.* ♢

PROOF: If the test accepts with probability $1/2+\delta$ then $\mathsf{E}[f(\mathbf{x})f(\mathbf{y})f(\mathbf{xyz})] = 2\delta$. Replacing f by its Fourier expansion, we have

$$\begin{aligned} 2\delta &\leq \mathop{\mathsf{E}}_{\mathbf{x},\mathbf{y},\mathbf{z}} \left[(\sum_\alpha \hat{f}_\alpha \chi_\alpha(\mathbf{x})) \cdot (\sum_\beta \hat{f}_\beta \chi_\beta(\mathbf{y})) \cdot (\sum_\gamma \hat{f}_\gamma \chi_\gamma(\mathbf{xyz})) \right] \\ &= \mathop{\mathsf{E}}_{\mathbf{x},\mathbf{y},\mathbf{z}} \left[\sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \chi_\alpha(\mathbf{x}) \chi_\beta(\mathbf{y}) \chi_\gamma(\mathbf{x}) \chi_\gamma(\mathbf{y}) \chi_\gamma(\mathbf{z}) \right] \\ &= \sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \mathop{\mathsf{E}}_{\mathbf{x},\mathbf{y},\mathbf{z}} [\chi_\alpha(\mathbf{x}) \chi_\beta(\mathbf{y}) \chi_\gamma(\mathbf{x}) \chi_\gamma(\mathbf{y}) \chi_\gamma(\mathbf{z})] . \end{aligned}$$

Orthonormality implies the expectation is 0 unless $\alpha = \beta = \gamma$, so this is

$$= \sum_\alpha \hat{f}_\alpha^3 \mathop{\mathsf{E}}_{\mathbf{z}} [\chi_\alpha(z)]$$

Now $\mathsf{E}_{\mathbf{z}}[\chi_\alpha(\mathbf{z})] = \mathsf{E}_{\mathbf{z}}\left[\prod_{w\in\alpha} z_w\right]$ which is equal to $\prod_{w\in\alpha} \mathsf{E}[z_w] = (1-2\rho)^{|\alpha|}$ because each coordinate of $\mathbf{z}$ is chosen independently. Hence we get that

$$2\delta \leq \sum_\alpha \hat{f}_\alpha^3 (1-2\rho)^{|\alpha|} . \ \blacksquare$$

The conclusion of Lemma 22.24 is reminiscent of the calculation in the proof of Theorem 22.22, except for the extra factor $(1-2\rho)^{|\alpha|}$. This factor depresses the contribution of $\hat{f}_\alpha$ for large α, allowing us to conclude that the small α's must contribute a lot. This is formalized in the following corollary (which is a simple calculation and left as Exercise 22.8).

Corollary 22.25 *If f passes the long code test with probability $1/2+\delta$, then for $k = \frac{1}{2\rho}\log\frac{1}{\epsilon}$, there exists α with $|\alpha| \leq k$ such that $\hat{f}_\alpha \geq 2\delta - \epsilon$.* ♢

22.7 Proof of Theorem 22.16

We now prove Håstad's' Theorem. The starting point is the $2\mathsf{CSP}_W$ instance φ given by Theorem 22.15, so we know that φ is either satisfiable, or we can satisfy at most ϵ fraction of the constraints, where ϵ is arbitrarily small. Let W be the alphabet size, n be the number of variables and m the number of constraints. We think of an assignment as a function π from $[n]$ to $[W]$. Since the 2CSP instance has the *projection* property, we can think of each constraint φ_r as being equivalent to some function $h : [W] \to [W]$, where the constraint is satisfied by assignment π iff $\pi(j) = h(\pi(i))$.

Håstad's verifier uses the long code, but expects these encodings to be *bifolded*, a technical property we now define and is motivated by the observation that coordinate functions satisfy $\chi_{\{w\}}(-\mathbf{v}) = -\chi_{\{w\}}(\mathbf{v})$ for every vector $\mathbf{v}$.

Definition 22.26 A function $f : \{\pm 1\}^W \to \{\pm 1\}$ is *bifolded* if for all $\mathbf{v} \in \{\pm 1\}^W$, $f(-\mathbf{v}) = -f(\mathbf{v})$. ♢

(Aside: In mathematics we would call such a function *odd* but the term "folding" is more standard in the **PCP** literature where it has a more general meaning.)

Whenever the **PCP** proof is supposed to contain a codeword of the long code, we may assume without loss of generality that the function is bifolded. The reason is that the verifier can identify, for each pair of inputs $\mathbf{v}, -\mathbf{v}$, one designated representative —say the one whose first coordinate is $+1$— and just define $f(-\mathbf{v})$ to be $-f(\mathbf{v})$. One benefit —though of no consequence in the proof— of this convention is that bifolded functions require only half as many bits to represent. We will use the following fact:

Lemma 22.27 *If $f : \{\pm 1\}^W \to \{\pm 1\}$ is bifolded and $\hat{f}_\alpha \neq 0$ then $|\alpha|$ must be an odd number (and in particular, nonzero).* ♢

Proof: By definition,

$$\hat{f}_\alpha = \langle f, \chi_\alpha \rangle = \mathop{\mathsf{E}}_{\mathbf{v}}[f(\mathbf{v}) \prod_{i \in \alpha} \mathbf{v}_i] \,.$$

If $|\alpha|$ is even then $\prod_{i\in\alpha} \mathbf{v}_i = \prod_{i\in\alpha}(-\mathbf{v}_i)$. So if f is bifolded, the contributions corresponding to $\mathbf{v}$ and $-\mathbf{v}$ cancel each other and the entire expectation is 0. ■

Håstad's verifier. Now we describe Håstad's verifier V_H. V_H expects the proof $\tilde{\pi}$ to consist of a satisfying assignment to φ where the value of each of the n variables is encoded using the (bifolded) long code. Thus the proof consists $n2^W$ bits (rather, $n2^{W-1}$ if we take the bifolding into account), which V_H treats as n functions $f_1, f_2, \ldots, f_n$ each mapping $\{\pm 1\}^W$ to $\{\pm 1\}$. The verifier V_H randomly picks a constraint, say $\varphi_r(i, j)$, in the $2\mathsf{CSP}_W$ instance. Then V_H tries to check (while reading only three bits!) that functions f_i, f_j encode two values in $[W]$ that would satisfy φ_r, in other words, they encode two values w, u satisfying $h(w) = u$ where $h : [W] \to [W]$ is the function describing constraint φ_r. Now we describe this test, which is reminiscent of the long code test we saw earlier.

The Basic Håstad Test.
Given: Two functions $f, g : \{\pm 1\}^W \to \{\pm 1\}$. A function $h : [W] \to [W]$.
Goal: Check if f, g are long codes of two values w, u such that $h(w) = u$.
Test: For $u \in [W]$ let $h^{-1}(u)$ denote the set $\{w : h(w) = u\}$. Note that the sets $\{h^{-1}(u) : u \in [W]\}$ form a partition of $[W]$. For a string $\mathbf{y} \in \{\pm 1\}^W$ we define $\mathcal{H}^{-1}(\mathbf{y})$ as the string in $\{\pm 1\}^W$ such that for every $w \in [W]$, the wth bit of $\mathcal{H}^{-1}(\mathbf{y})$ is $y_{h(w)}$. In other words, for each $u \in [W]$, the bit y_u appears in $\mathcal{H}^{-1}(\mathbf{y})$ in all coordinates corresponding to $h^{-1}(u)$. V_H chooses uniformly at random $\mathbf{v}, \mathbf{y} \in \{\pm 1\}^W$ and chooses $\mathbf{z} \in \{\pm 1\}^W$ by letting $z_i = +1$ with probability $1 - \rho$ and $z_i = -1$ with probability ρ. It then accepts if

$$f(\mathbf{v})g(\mathbf{y}) = f(\mathcal{H}^{-1}(\mathbf{y})\mathbf{v}\mathbf{z}) \tag{4}$$

and rejects otherwise.

Translating back from $\{\pm 1\}$ to $\{0,1\}$, note that V_H's test is indeed linear, as it accepts iff $\tilde{\pi}[i_1] + \tilde{\pi}[i_2] + \tilde{\pi}[i_3] = b$ for some $i_1, i_2, i_3 \in [n2^W]$ and $b \in \{0,1\}$. (The bit b can indeed equal 1 because of the way V_H ensures the bifolding property.)

Now since ρ, ϵ can be arbitrarily small the next claim suffices to prove the Theorem. (Specifically, making $\rho = \epsilon^{1/3}$ makes the completeness parameter at least $1 - \epsilon^{1/3}$ and the soundness at most $1/2 + \epsilon^{1/3}$.)

Claim 22.28 *(Main) If φ is satisfiable, then there is a proof which V_H accepts with probability $1 - \rho$. If $\mathsf{val}(\varphi) \leq \epsilon$ then V_H accepts no proof with probability more than $1/2 + \delta$ where $\delta = \sqrt{\epsilon/\rho}$.* ♢

The rest of the section is devoted to proving Claim 22.28.

Completeness part; easy. If φ is satisfiable, then take any satisfying assignment $\pi : [n] \to [W]$ and form a proof for V_H containing the bifolded long code encodings of the n values. (As already noted, coordinate functions are bifolded.) To show that V_H accepts this proof with probability $1-\rho$, it suffices to show that the Basic Håstad Test accepts with probability $1 - \rho$ for every constraint.

Suppose f, g are long codes of two integers w, u satisfying $h(w) = u$. Then, using the fact that for $x \in \{\pm 1\}$, $x^2 = 1$,

$$\begin{aligned} f(\mathbf{v})g(\mathbf{y})f(\mathcal{H}^{-1}(\mathbf{y})\mathbf{v}\mathbf{z}) &= \mathbf{v}_w \mathbf{y}_u (\mathcal{H}^{-1}(\mathbf{y})_w \mathbf{v}_w \mathbf{z}_w) \\ &= \mathbf{v}_w \mathbf{y}_u (\mathbf{y}_{h(w)} \mathbf{v}_w \mathbf{z}_w) \qquad = \mathbf{z}_w . \end{aligned}$$

Hence V_H accepts iff $\mathbf{z}_w = 1$, which happens with probability $1 - \rho$.

Soundness of V_H; more difficult. We first show that if the Basic Håstad Test accepts two functions f, g with probability significantly more than $1/2$, then the Fourier transforms of f, g must be correlated. To formalize this we define for $\alpha \subseteq [W]$,

$$h_2(\alpha) = \left\{ u \in [W] : \ \left| h^{-1}(u) \cap \alpha \right| \text{ is odd} \right\} \tag{5}$$

Notice in particular that for *every* $t \in h_2(\alpha)$ there is at least one $w \in \alpha$ such that $h(w) = t$.

In the next Lemma δ is allowed to be negative. It is the only place where we use the bifolding property.

Lemma 22.29 *Let $f, g : \{\pm 1\}^W \to \{\pm 1\}$, be bifolded functions and $h : [W] \to [W]$ be such that they pass the Basic Håstad Test (4) with probability at least $1/2 + \delta$. Then*

$$\sum_{\alpha \subseteq [W], \alpha \neq \emptyset} \hat{f}_\alpha^2 \hat{g}_{h_2(\alpha)} (1 - 2\rho)^{|\alpha|} \geq 2\delta \tag{6}$$

♢

Proof: By hypothesis, f, g are such that $E[f(\mathbf{v})f(\mathbf{v}\mathcal{H}^{-1}(\mathbf{y})\mathbf{z})g(\mathbf{y})] \geq 2\delta$. Replacing f, g by their Fourier expansions we get:

$$\begin{aligned} 2\delta \leq &= \mathop{\mathsf{E}}_{\mathbf{v},\mathbf{y},\mathbf{z}} \left[\left(\sum_\alpha \hat{f}_\alpha \chi_\alpha(\mathbf{v})\right)\left(\sum_\beta \hat{g}_\beta \chi_\beta(\mathbf{y})\right)\left(\sum_\gamma \hat{f}_\gamma \chi_\gamma(\mathbf{v}\mathcal{H}^{-1}(\mathbf{y})\mathbf{z})\right) \right] \\ &= \sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{g}_\beta \hat{f}_\gamma \mathop{\mathsf{E}}_{\mathbf{v},\mathbf{y},\mathbf{z}} \left[\chi_\alpha(\mathbf{v}) \chi_\beta(\mathbf{y}) \chi_\gamma(\mathbf{v}) \chi_\gamma(\mathcal{H}^{-1}(\mathbf{y})) \chi_\gamma(\mathbf{z}) \right] . \end{aligned}$$

By orthonormality this simplifies to

$$\begin{aligned} &= \sum_{\alpha,\beta} \hat{f}_\alpha^2 \hat{g}_\beta \mathop{\mathsf{E}}_{\mathbf{y},\mathbf{z}} \left[\chi_\beta(\mathbf{y}) \chi_\alpha(\mathcal{H}^{-1}(\mathbf{y})) \chi_\alpha(\mathbf{z}) \right] \\ &= \sum_{\alpha,\beta} \hat{f}_\alpha^2 \hat{g}_\beta (1 - 2\rho)^{|\alpha|} \mathop{\mathsf{E}}_{\mathbf{y}} \left[\chi_\alpha(\mathcal{H}^{-1}(\mathbf{y}) \chi_\beta(\mathbf{y}) \right] \end{aligned} \tag{7}$$

since $\chi_\alpha(\mathbf{z}) = (1-2\rho)^{|\alpha|}$, as noted in our analysis of the long code test. Now we have

$$\begin{aligned}\mathop{\mathsf{E}}_{\mathbf{y}}[\chi_\alpha(\mathcal{H}^{-1}(\mathbf{y}))\chi_\beta(\mathbf{y})] &= \mathop{\mathsf{E}}_{\mathbf{y}}[\prod_{w\in\alpha}\mathcal{H}^{-1}(\mathbf{y})_w \prod_{u\in\beta}\mathbf{y}_u] \\ &= \mathop{\mathsf{E}}_{\mathbf{y}}[\prod_{w\in\alpha}\mathbf{y}_{h(w)} \prod_{u\in\beta}\mathbf{y}_u],\end{aligned}$$

which is 1 if $h_2(\alpha) = \beta$ and 0 otherwise. Hence (7) simplifies to

$$\sum_\alpha \hat{f}_\alpha^2 \hat{g}_{h_2(\alpha)}(1-2\rho)^{|\alpha|}.$$

Finally we note that since the functions are assumed to be bifolded, the Fourier coefficients $\hat{f}_\emptyset$ and $\hat{g}_\emptyset$ are zero. Thus those terms can be dropped from the summation and the Lemma is proved. ■

The following Lemma completes the proof of the Claim 22.28 and hence of Håstad's 3-bit PCP Theorem.

Lemma 22.30 *Suppose φ is an instance of $2\mathsf{CSP}_W$ such that $\mathsf{val}(\varphi) < \epsilon$. If ρ, δ satisfy $\rho\delta^2 > \epsilon$ then verifier V_H accepts any proof with probability at most $1/2 + \delta$.* ♢

Proof: Suppose V_H accepts a proof $\tilde{\pi}$ of length $n2^W$ with probability at least $1/2 + \delta$. We give a probabilistic construction of an assignment π to the variables of φ such that the expected fraction of satisfied constraints is at least $\rho\delta^2$, whence it follows by the probabilistic method that a specific assignment π exists that lives up to this expectation. This contradicts the hypothesis if $\rho\delta^2 > \epsilon$.

The distribution from which π is chosen. We can think of $\tilde{\pi}$ as providing, for every $i \in [n]$, a function $f_i : \{\pm 1\}^W \to \{\pm 1\}$. The probabilistic construction of assignment π comes in two steps: we first use f_i to define a distribution $\mathcal{D}_i$ over $[W]$ as follows: select $\alpha \subseteq [W]$ with probability $\hat{f}_\alpha^2$ where $f = f_i$ and then select w at random from α. This is well-defined because $\sum_\alpha \hat{f}_\alpha^2 = 1$ and (due to bifolding) the fourier coefficient $f_\emptyset$ corresponding to the empty set is 0. We then pick $\pi[i]$ by drawing a random sample from distribution $\mathcal{D}_i$. Thus the assignment π is a random element of the product distribution $\prod_{i=1}^m \mathcal{D}_i$. We wish to show

$$\mathop{\mathsf{E}}_{\pi}[\mathop{\mathsf{E}}_{r\in[m]}[\pi \text{ satisfies } r\text{th constraint}]] \geq \rho\delta^2. \tag{8}$$

The analysis. For every constraint φ_r where $r \in [m]$ denote by $1/2 + \delta_r$ the conditional probability that the Basic Håstad Test accepts $\tilde{\pi}$, conditioned on V_H having picked φ_r. (Note: δ_r could be negative.) Then the acceptance probability of V_H is $\mathsf{E}_r[\frac{1}{2} + \delta_r]$ and hence $\mathsf{E}_r[\delta_r] = \delta$. We show that

$$\Pr_{\pi}[\pi \text{ satisfies } \varphi_r] \geq \rho\delta_r^2, \tag{9}$$

whence it follows that the left hand side of (8) is (by linearity of expectation) at least $\rho E_{r\in[m]}[\delta_r^2]$. Since $E[X^2] \geq E[X]^2$ for any random variable, this in turn is at least $\rho(E_r[\delta_r])^2 \geq \rho\delta^2$. Thus to finish the proof it only remains to prove (9).

Let $\varphi_r(i,j)$ be the rth constraint and let h be the function describing this constraint, so that

$$\pi \text{ satisfies } \varphi_r \qquad \text{iff} \qquad h(\pi[i]) = \pi[j].$$

Let I_r be the indicator random variable for the event $h(\pi[i] = \pi[j])$. From now on we use the shorthand $f = f_i$ and $g = f_j$. What is the chance that a pair of assignments $\pi[i] \in_R D_i$ and $\pi[j] \in_R D_j$ will satisfy $\pi[j] = h(\pi[i])$? Recall that we pick these values by choosing α with probability $\hat{f}_\alpha^2$, β with probability $\hat{g}_\beta^2$ and choosing $\pi[i] \in_R \alpha, \pi[j] \in_R \beta$. Assume that α is picked first. The conditional probability that $\beta = h_2(\alpha)$ is $\hat{g}^2_{h_2(\alpha)}$. If $\beta = h_2(\alpha)$, we claim

that the conditional probability of satisfying the constraint is at least $1/|\alpha|$. The reason is that by definition, $h_2(\alpha)$ consists of u such that $\left|h^{-1}(u) \cap \alpha\right|$ is odd, and an odd number cannot be 0! Thus regardless of which value $\pi[j] \in h_2(\alpha)$ we pick, there *exists* $w \in \alpha$ with $h(w) = \pi[j]$, and the conditional probability of picking such a w as $\pi[i]$ is at least $1/|\alpha|$. Thus, we have that

$$\sum_{\alpha} \frac{1}{|\alpha|} \hat{f}_{\alpha}^2 \hat{g}_{h_2(\alpha)}^2 \leq \mathop{\mathsf{E}}_{D_i, D_j}[I_r] \tag{10}$$

This is similar to (but not quite the same as) the expression in Lemma 22.29, according to which

$$2\delta_r \leq \sum_{\alpha} \hat{f}_{\alpha}^2 \hat{g}_{h_2(\alpha)} (1-2\rho)^{|\alpha|}.$$

However, since one can easily see that $(1-2\rho)^{|\alpha|} \leq \frac{2}{\sqrt{\rho\,|\alpha|}}$ we have

$$2\delta_r \leq \sum_{\alpha} \hat{f}_{\alpha}^2 \left|\hat{g}_{h_2(\alpha)}\right| \frac{2}{\sqrt{\rho\,|\alpha|}}.$$

Rearranging,

$$\delta_r \sqrt{\rho} \leq \sum_{\alpha} \hat{f}_{\alpha}^2 \left|\hat{g}_{h_2(\alpha)}\right| \frac{1}{\sqrt{|\alpha|}}.$$

Applying the Cauchy-Schwartz inequality, $\sum_i a_i b_i \leq (\sum_i a_i^2)^{1/2} (\sum_i b_i^2)^{1/2}$, with $\hat{f}_{\alpha} \left|\hat{g}_{\pi_2(\alpha)}\right| \frac{1}{\sqrt{|\alpha|}}$ playing the role of the a_i's and $\hat{f}_{\alpha}$ playing that of the b_i's, we obtain

$$\delta_r \sqrt{\rho} \leq \sum_{\alpha} \hat{f}_{\alpha}^2 \left|\hat{g}_{h_2(\alpha)}\right| \frac{1}{\sqrt{|\alpha|}} \leq \left(\sum_{\alpha} \hat{f}_{\alpha}^2\right)^{1/2} \left(\sum_{\alpha} \hat{f}_{\alpha}{}^2 \hat{g}_{h_2(\alpha)}^2 \frac{1}{|\alpha|}\right)^{1/2} \tag{11}$$

Since $\sum_{\alpha} \hat{f}_{\alpha}^2 = 1$, by squaring (11) and combining it with (10) we get that for every r,

$$\delta_r^2 \rho \leq \mathop{\mathsf{E}}_{\mathcal{D}_i, \mathcal{D}_j}[I_r],$$

which proves (9) and finishes the proof. ■

22.8 Hardness of approximating SET-COVER

In the SET-COVER problem we are given a ground set $\mathcal{U}$ and a collection of its subsets $S_1, S_2, \ldots, S_n$ whose union is $\mathcal{U}$, and we desire the smallest subcollection I such that $\cup_{i \in I} S_i = \mathcal{U}$. Such a subcollection is called a *set cover* and its *size* is $|I|$. An algorithm is said to ρ-approximate this problem, where $\rho < 1$ if for every instance it finds a set cover of size at most OPT/ρ, where OPT is the size of the smallest set cover.

Theorem 22.31 *([LY94]) If for any constant $\rho > 0$ there is an algorithm that ρ-approximates* SET-COVER *then* $\mathbf{P} = \mathbf{NP}$. *Specifically for every $\epsilon, W > 0$ there is a polynomial-time transformation f from* $2\mathsf{CSP}_W$ *instances to instances of* SET-COVER *such that if the* $2\mathsf{CSP}_W$ *instance is regular and satisfies the projection property then*

$$\mathsf{val}(\varphi) = 1 \Rightarrow f(\varphi) \text{ has a set cover of size } N$$
$$\mathsf{val}(\varphi) < \epsilon \Rightarrow f(\varphi) \text{ has no set cover of size} \frac{N}{4\sqrt{\epsilon}},$$

where N depends upon φ. ♢

Actually one can prove a somewhat stronger result; see the note at the end of the proof.

The proof needs the following gadget.

Definition 22.32 *((k,ℓ)-set gadget)* A (k,ℓ)-*set gadget* consists of a ground set $\mathcal{B}$ and some of its subsets $C_1, C_2, \ldots, C_\ell$ with the following property: every collection of at most k sets out of $C_1, \overline{C_1}, C_2, \overline{C_2}, \ldots, C_\ell, \overline{C_\ell}$ that is a set cover for $\mathcal{B}$ must include both C_i and $\overline{C_i}$ for some i. ♢

The following Lemma is left as Exercise 22.13.

Lemma 22.33 *There is an algorithm that given any k, ℓ, runs in time* $\mathrm{poly}(m, 2^\ell)$ *and outputs a (k,ℓ)-set gadget.* ♢

We can now prove Theorem 22.31. We give a reduction from $2\mathsf{CSP}_W$, specifically, the instances obtained from Theorem 22.15.

Let φ be an instance of $2\mathsf{CSP}_W$ such that either $\mathsf{val}(\varphi) = 1$ or $\mathsf{val}(\varphi) < \epsilon$ where ϵ is some arbitrarily small constant. Suppose it has n variables and m constraints. Let Γ_i denote the set of constraints in which the ith variable is the first variable, and Δ_i the set of constraints in which it is the second variable.

The construction. Construct a (k, W)-set gadget $(\mathcal{B}; C_1, C_2, \ldots, C_W)$ where $k > 2/\sqrt{\epsilon}$. Since variables take values in $[W]$, we can associate a set C_u with each variable value u.

The instance of SET-COVER is as follows. The ground set is $[m] \times \mathcal{B}$, which we will think of as m copies of $\mathcal{B}$, one for each constraint of φ. The number of subsets is nW; for each variable $i \in [n]$ and value $u \in [W]$ there is a subset $S_{i,u}$ which is the union of the following sets: $\{r\} \times C_u$ for each $r \in \Delta_i$ and $\{r\} \times \mathcal{B} \setminus C_{h(u)}$ for each $r \in \Gamma_i$. The use of complementary sets like this is at the root of how the gadget allows 2CSP (with projection property) to be encoded as SET-COVER.

The analysis. If the $2\mathsf{CSP}_W$ instance is satisfiable then we exhibit a set cover of size n. Let $\pi : [n] \to W$ be any satisfying assignment where $\pi(i)$ is the value of the ith variable. We claim that the collection of n subsets given by $\{S_{i,\pi[i]} : i \in [n]\}$ is a set cover. It suffices to show that their union contains $\{r\} \times \mathcal{B}$ for each constraint r. But this is trivial since if i is the first variable of constraint r and j the second variable, then by definition $S_{j,\pi[j]}$ contains $\{r\} \times C_{\pi[j]}$ and $S_{i,\pi[i]}$ contains $\{r\} \times \mathcal{B} \setminus C_{\pi[j]}$, and thus $S_{i,\pi[i]} \cup S_{j,\pi[j]}$ contains $\{r\} \times \mathcal{B}$.

Conversely, suppose less than ϵ fraction of the constraints in the $2\mathsf{CSP}_W$ instance are simultaneously satisfiable. We claim that every set cover must have at least nT sets, for $T = \frac{1}{4\sqrt{\epsilon}}$. For contradiction's sake suppose a set cover of size less than nT exists. Let us probabilistically construct an assignment for the $2\mathsf{CSP}_W$ instance as follows. For each variable i, say that a value u is *associated* with it if $S_{i,u}$ is in the set cover. We pick a value for i by randomly picking one of the values associated with it. It suffices to prove the following claim since our choice of k ensures that $8T < k$.

CLAIM: *If $8T < k$ then the expected number of constraints satisfied by this assignment is more than $\frac{m}{16T^2}$.*

PROOF: Call a variable *good* if it has less than $4T$ values associated with it. The average number of values associated per variable is less than T, so less than $1/4$ of the variables have more than $4T$ values associated with them. Thus less than $1/4$ of the variables are not good.

Since the 2CSP instance is regular, each variable occurs in the same number of clauses. Thus the fraction of constraints containing a variable that is not good is less than $2 \times 1/4 = 1/2$. Thus for more than $1/2$ of the constraints both variables in them are good. Let r be such a constraint and i, j be the variables in it. Then $\{r\} \times \mathcal{B}$ is covered by the union of $\cup_u S_{i,u}$ and $\cup_v S_{j,v}$ where the unions are over values associated with the variables i, j respectively. Since $8T < k$, the definition of a (k, W)-set gadget implies that any cover of $\mathcal{B}$ by less than $8T$ sets must contain two sets that are complements of one another. We conclude that there are values u, v associated with i, j respectively such that $h(u) = v$. Thus when we randomly construct an assignment by picking for each variable one of the values

associated with it, these values are picked with probability at least $1/4T \times 1/4T = 1/16T^2$, and then the rth constraint gets satisfied. The claim (and hence Theorem 22.31) now follows by linearity of expectation. ■

Remark 22.34
The same proof actually can be used to prove a stronger result: there is a constant $c > 0$ such that if there is an algorithm that α-approximates SET-COVER for $\alpha = c/\log n$ then $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log n)})$. The idea is to use Raz's parallel repetition theorem where the number of repetitions t is superconstant so that the soundness is $1/\log n$. However, the running time of the reduction is $n^{O(t)}$, which is slightly superpolynomial.

22.9 Other PCP Theorems: A Survey

As mentioned in the introduction, proving inapproximability results for various problems often requires proving new **PCP** Theorems. We already saw one example, namely, Håstad's 3-bit **PCP** Theorem. Now we survey some other variants of the **PCP** Theorem that have been proved.

22.9.1 PCP's with sub-constant soundness parameter

The way we phrased Theorem 22.15, the soundness is an arbitrary small constant 2^{-t}. From the proof of the theorem it was clear that the reduction used to prove this **NP**-hardness runs in time n^t (since it forms all t-tuples of constraints). Thus if t is larger than a constant, the running time of the reduction is superpolynomial. Nevertheless, several hardness results use superconstant values of t. They end up not showing **NP**-hardness, but instead showing the nonexistence of a good approximation algorithms assuming **NP** does not have say $n^{\log n}$ time deterministic algorithms (this is still a very believable assumption). We mentioned this already in Remark 22.34 at the end of Section 22.8.

It is still an open problem to prove the **NP**-hardness of 2CSP for a factor ρ that is smaller than any constant. If instead of 2CSP one looks at 3CSP or 4CSP then one can achieve low soundness using larger alphabet size, while keeping the running time polynomial [RS97]. Often these suffice in applications.

22.9.2 Amortized query complexity

Some applications require binary-alphabet **PCP** systems enjoying a tight relation between the number of queries (which can be an arbitrarily large constant) and the soundness parameter. The relevant parameter here turns out to be the *free bit complexity* [FK93, BS94]. This parameter is defined as follows. Suppose the number of queries is q. After the verifier has picked its random string, and picked a sequence of q addresses, there are 2^q possible sequences of bits that could be contained in those addresses. If the verifier accepts for only t of those sequences, then we say that the free bit parameter is $\log t$ (note that this number need not be an integer). In fact, for proving hardness result for MAX-INDSET and MAX-CLIQUE, it suffices to consider the *amortized free bit complexity* [BGS95]. This parameter is defined as $\lim_{s\to 0} f_s/\log(1/s)$, where f_s is the number of free bits needed by the verifier to ensure the soundness parameter is at most s (with completeness at least say $1/2$). Håstad constructed systems with amortized free bit complexity tending to zero [Hås96]. That is, for every $\epsilon > 0$, he gave a **PCP**-verifier for **NP** that uses $O(\log n)$ random bits and ϵ amortized free bits. The completeness is 1. He then used this **PCP** system to show (borrowing the basic framework from [FGL+91, FK93, BS94, BGS95]) that MAX-INDSET (and so, equivalently, MAX-CLIQUE) is **NP**-hard to $n^{-1+\epsilon}$-approximate for arbitrarily small $\epsilon > 0$.

22.9.3 2-bit tests and powerful fourier analysis

Recent advances on Håstad's line of work consist of using more powerful ideas from Fourier analysis. The Fourier analysis in Håstad's proof hardly uses the fact that the functions being tested are Boolean. However, papers of Kahn, Kalai, Linial [KKL88], Friedgut [Fri99], and Bourgain [Bou02] have led to important new insights into the Fourier coefficients of Boolean functions, which in turn have proved useful in analysing **PCP** verifiers. (See also Note 22.21.) The main advantage is for designing verifiers that read only 2 bits in the proof, which arise while proving hardness results for a variety of graph problems such as VERTEX-COVER, MAX-CUT and SPARSEST-CUT.

These new results follow Håstad's overall idea, namely, to show that if the verifier accepts some provided functions with good probability, then the function has a few large fourier coefficients (see Corollary 22.25 for example). However, Håstad's analysis (even for the long code test in Section 22.6) inherently requires the verifier to query 3 bits in the proof, and we briefly try to explain why. For simplicity we focus on the long code test. We did a simple analysis of the long code test to arrive at the conclusion of Lemma 22.24:

$$\sum_{\alpha} \hat{f}_{\alpha}^{3}(1-2\rho)^{|\alpha|} \geq 2\delta,$$

where $1/2+\delta$ is the probability that the verifier accepts the function. From this fact, Corollary 22.25 concludes that at least one fourier coefficient has value at least $c = c(\delta,\rho) > 0$. This is a crucial step because it lets us conclude that f has some (admittedly very weak) connection with some small number of codewords in the long code.

One could design an analogous 2-bit test. The first part of the above analysis still goes through but in the conclusion the cubes get replaced by squares:

$$\sum_{\alpha} \hat{f}_{\alpha}^{2}(1-2\rho)^{|\alpha|} \geq 2\delta. \tag{12}$$

For a non-Boolean function this condition is not enough to let us conclude that some fourier coefficient of f has value at least $c = c(\delta,\rho) > 0$. However, the following lemma of Bourgain allows such a conclusion if f is Boolean. We say that a function $f : \{0,1\}^n \to \{0,1\}$ is a *k-junta* if it depends only on k of the n variables. Note that Parseval's identity implies that at least one fourier coefficient of a k-junta is $1/2^{k/2}$. The next Lemma implies that if a boolean function f is such that the LHS of (12) is at least $1-\rho^t$ where $t > 1/2$, then f is close to a k-junta for a small k.

Lemma 22.35 *([Bou02]) For every $\epsilon,\delta > 0$ and integer r there is a constant $k = k(r,\epsilon,\delta)$ such that if*

$$\sum_{\alpha:|\alpha|>r} \hat{f}_{\alpha}^{2} < \frac{1}{r^{1/2+\epsilon}},$$

then f has agreement $1-\delta$ with a k-junta. ♢

We suspect that there will be many other uses of fourier analysis in **PCP** constructions.

22.9.4 Unique games and threshold results

Håstad's ideas led to determination of the approximation threshold for several problems. But the status of other problems such as VERTEX-COVER and MAX-CUT remained open. In 2002 Khot [Kho02] proposed a new complexity theoretic conjecture called the *unique games conjecture* (UGC) that is stronger than $\mathbf{P} \neq \mathbf{NP}$ but still consistent with current knowledge. This conjecture concerns a special case of $2\mathsf{CSP}_W$ in which the constraint function is a *permutation* on $[W]$. In other words, if the constraint φ_r involves variables i,j, the constraint function h is a bijective mapping from $[W]$ to $[W]$. Then assignment $u_1,u_2,\ldots,u_n$ to the variables satisfies this constraint iff $u_j = h(u_i)$. According to UGC, for every constants $\epsilon,\delta > 0$ there is a domain size $W = W(\epsilon,\delta)$ such that there is no polynomial-time algorithm

that given such an instance of $2\mathsf{CSP}_W$ with $\mathsf{val}(\cdot) \geq 1-\epsilon$ produces an assignment that satisfies δ fraction of constraints.[2]

Khot suggested that current algorithmic techniques seem unable to design such an algorithm (an insight that seems to have been confirmed by lack of progress in the last few years, despite much effort). He also showed that this conjecture implies several strong results about hardness of approximation. The reason in a nutshell is that the Fourier analysis technique of Håstad (fortified with the above-mentioned discoveries regarding Fourier analysis of Boolean functions) can be sharpened if one starts with the instances of $2\mathsf{CSP}_W$ with the uniqueness constraint.

Subsequently, a slew of results have shown optimal or threshold results about hardness of approximation (often using some of the advanced fourier analysis mentioned above) assuming UGC is true.f For instance UGC implies that there is no polynomial-time algorithm that computes a $1/2+\delta$-approximation to VERTEX-COVER for any $\delta > 0$ [KR08], and similarly no algorithm that computes a 0.878-approximation to MAX-CUT [KKMO05, MOO05]. Both of these lead to threshold results since these ratios are also the ones achieved by the current approximation algorithms.

Thus it is of great interest to prove or disprove the unique games conjecture. Algorithms designers have tried to disprove the conjecture using clever tools from semidefinite programming, and currently the conjecture seems truly on the fine line between being true and being false. It is known that it will suffice to restrict attention to the further subcase where the constraint function h is linear —i.e., the constraints are linear equations mod W in two variables.

22.9.5 Connection to Isoperimetry and Metric Space Embeddings

A *metric space* (X,d) consists of set of points X and a function d mapping pairs of points to nonnegative real numbers satisfying (a) $d(i,j)=0$ iff $i=j$. (b) $d(i,j)+d(j,k) \geq d(i,k)$ (*triangle inequality*). An *embedding* of space (X,d) into space (Y,d') is a function $f: X \to Y$. Its *distortion* is the maximum over all point pairs $\{i,j\}$ of the quantities $\frac{d'(f(i),f(j))}{d(i,j)}, \frac{d(i,j)}{d'(f(i),f(j))}$. It is of great interest in algorithm design (and in mathematics) to understand the minimum *distortion* required to embed one family of metric spaces into another. One interesting subcase is where the host space (Y,d') is a subset of the ℓ_1 metric space on $\Re^n$ for some n (in other words, distance d' is defined using the ℓ_1 norm). Bourgain showed that every n-point metric space embeds in ℓ_1 with distortion $O(\log n)$. This fact is important in design of algorithms for graph partitioning problems such as SPARSEST-CUT. In that context, a metric called ℓ_2^2 had been identified. Goemans and Linial conjectured that this metric would be embeddable in ℓ_1 with distortion $O(1)$. If the conjecture were true we would have an $O(1)$-approximation algorithm for SPARSEST-CUT. Khot and Vishnoi [KV05] disproved the conjecture, using a construction of an interesting ℓ_2^2 metric that is inspired by the advanced **PCP** Theorems discussed in this chapter. The main idea is that since there is an intimate relationship between ℓ_1 metrics and cuts, one has to construct a graph whose cut structure and isoperimetry properties are tightly controlled. So Khot and Vishnoi use a hypercube-like graph, and use Fourier analysis to show its isoperimetry properties. (See Note 22.21.)

Their work has inspired other results about lower bounds on the distortions of metric embeddings.

Chapter notes and history

As mentioned in the notes to Chapter 11, the **PCP** Theorem was proved in 1992 in the early versions of the papers [AS92, ALM+92]. The AS-ALMSS proof of the **PCP** Theorem resisted simplification

[2] In fact, Khot phrased the UGC as the even stronger conjecture that solving this problem is **NP**-hard.

for over a decade. The overall idea of that proof (as indeed in **MIP** = **NEXP**) is similar to the proof of Theorem 11.19. (Indeed, Theorem 11.19 is the only part of the original proof that still survives in our writeup.) However, in addition to using encodings based upon the Walsh-Hadamard code the proof also used encodings based upon low degree multivariate polynomials. These have associated procedures analogous to the linearity test and local decoding, though the proofs of correctness are a fair bit harder. The proof also drew intuition from the topic of self-testing and self-correcting programs [BLR90, RS92], which was surveyed in Section 8.6. The alphabet reduction in the AS-ALMSS proof was also somewhat more complicated. A draft writeup of the original proof is available on this book's website. (We dropped it from the published version in favor of Dinur's proof but feel it is interesting in its own right and may be useful in future research.)

Dinur's main contribution in simplifying the proof is the gap amplification lemma (Lemma 22.5), which allows one to iteratively improve the soundness parameter of the **PCP** from very close to 1 to being bounded away from 1 by some positive constant. This allows her to use a simpler alphabet reduction. In fact, the alphabet reduction is the only part of the proof that now uses the "proof verification" viewpoint, and one imagines that in a few years this too will be replaced by a purely combinatorial construction. A related open problem is to find a Dinur-style proof of **MIP** = **NEXP**.

We also note that Dinur's general strategy is somewhat reminiscent of the zig-zag construction of expander graphs and Reingold's deterministic logspace algorithm for undirected connectivity described in Chapter 20, which suggests that more connections are waiting to be made between these different areas of research.

As mentioned in the notes at the end of Chapter 11, Papadimitriou and Yannakakis [PY88] had shown around 1988 that if it is **NP**-hard to ρ-approximate MAX-3SAT for some $\rho < 1$ then it is also **NP**-hard to ρ'-approximate a host of other problems where ρ' depends upon the problem. Thus after the discovery of the **PCP** Theorem, attention turned towards determining the exact approximation threshold for problems; see for example [BS94, BGS95]. Håstad's threshold results for MAX-CLIQUE [Hås96] and MAX-3SAT [Hås97] came a few years later and represented a quantum jump in our understanding.

The issue of parallel repetition comes from the paper of Fortnow, Rompel, and Sipser [FRS88] that erroneously claimed that $\mathsf{val}(\varphi^{*t}) = \mathsf{val}(\varphi)^t$ for every 2CSP φ and $t \in \mathbb{N}$. However, Fortnow [For89] soon found a counter example (see Exercise 22.6). Papers Lapidot and Shamir [LS91], Feige-Lovasz [FL92], which predate Raz's paper, imply hardness results for 2CSP but the running time of the reduction is superpolynomial. Verbitsky [Ver94] and Feige and Kilian [FK93] proved weaker versions of Raz's Theorem (Theorem 22.15). Raz's proof of the parallel repetition is based on an extension of techniques developed by Razborov [Raz90] in the context of communication complexity. The proof is beautiful but quite complex, though recently Holenstein [Hol07] gave some simplifications for Raz's proof; a writeup of this simplified proof is available from this book's website.

The hardness result for INDSET in Lemma 22.8 can be improved so that for all $\epsilon > 0$, $n^{-1+\epsilon}$-approximation in **NP**-hard in graphs with n vertices. This result is due to [Hås96], which caps a long line of other work [FGL+91, AS92, ALM+92, BS94, BGS95]. The use of expanders in the reduction of Lemma 22.8 is from [AFWZ95]. Note that a $1/n$-approximation is trivial: just output a single vertex, which is always an independent set. Thus this result can be viewed as a *threshold* result.

The hardness of SET-COVER is due to Lund and Yannakakis [LY94], which was also the first paper to implicitly use $\mathsf{2CSP}_W$ with projection property; the importance of this problem was identified in [Aro94, ABSS93], where it was called *label cover* used to prove other results. This problem is now ubiquitous in **PCP** literature.

A threshold result for SET-COVER was shown by Feige [Fei96]: computing $(1+\delta)/\ln n$ approximation is hard for every $\delta > 0$, whereas we know a simple $1/\ln n$-approximation algorithm.

See Arora and Lund [AL95] for a survey circa 1995 of how to prove the basic results about hardness of approximation. See Khot [Kho05] for a more recent survey about the results that use fourier analysis.

Exercises

22.1 Prove Equation (1). H465

22.2 Let $G = (V, E)$ be a λ-expander graph for some $\lambda \in (0, 1)$. Let S be a subset of V with $|S| = \beta|V|$ for some $\beta \in (0, 1)$. Let $(X_1, \ldots, X_\ell)$ be a tuple of random variables denoting the vertices of a

uniformly chosen (ℓ–1)-step path in G. Then, prove that

$$(\beta - 2\lambda)^k \leq \Pr[\forall_{i\in[\ell]} X_i \in S] \leq (\beta + 2\lambda)^k$$

H465

22.3 Let S_t be the binomial distribution over t balanced coins. That is, $\Pr[S_t = k] = \binom{t}{k}2^{-t}$. Prove that for every $\delta < 1$, the statistical distance (see Section A.2.6) of S_t and $S_{t+\delta\sqrt{t}}$ is at most 10δ. H465

22.4 Prove that for every non-negative random variable V, $\Pr[V > 0] \geq \mathsf{E}[V]^2/\mathsf{E}[V^2]$.

H465

22.5 In this problem we explore an alternative approach to the Alphabet Reduction Lemma (Lemma 22.6) using Long Codes instead of Welsh-Hadamard codes. We already saw that the *long-code* for a set $\{0,\ldots,W-1\}$ is the function $\mathsf{LC} : \{0,\ldots,W-1\} \to \{0,1\}^{2^W}$ such that for every $i \in \{0..W\text{–}1\}$ and a function $f : \{0..W\text{–}1\} \to \{0,1\}$, (where we identify f with an index in $[2^w]$) the f^{th} position of $\mathsf{LC}(i)$, denoted by $\mathsf{LC}(i)_f$, is $f(i)$. We say that a function $L : \{0,1\}^{2^W} \to \{0,1\}$ is a *long-code codeword* if $L = \mathsf{LC}(i)$ for some $i \in \{0..W\text{–}1\}$.

- **(a)** Prove that LC is an error-correcting code with distance half. That is, for every $i \neq j \in \{0..W\text{–}1\}$, the fractional Hamming distance of $\mathsf{LC}(i)$ and $\mathsf{LC}(j)$ is half.
- **(b)** Prove that LC is *locally-decodable*. That is, show an algorithm that given random access to a function $L : 2^{\{0,1\}^W} \to \{0,1\}$ that is (1–ϵ)-close to $\mathsf{LC}(i)$ and $f : \{0..W\text{–}1\} \to \{0,1\}$ outputs $\mathsf{LC}(i)_f$ with probability at least 0.9 while making at most 2 queries to L.
- **(c)** Let $L = \mathsf{LC}(i)$ for some $i \in \{0..W\text{–}1\}$. Prove the for every $f : \{0..W\text{–}1\} \to \{0,1\}$, $L(f) = 1 - L(\overline{f})$, where $\overline{f}$ is the negation of f (i.e. , $\overline{f}(i) = 1 - f(i)$ for every $i \in \{0..W\text{–}1\}$).
- **(d)** Let T be an algorithm that given random access to a function $L : 2^{\{0,1\}^W} \to \{0,1\}$, does the following:
  - **(a)** Choose f to be a random function from $\{0..W\text{–}1\} \to \{0,1\}$.
  - **(b)** If $L(f) = 1$ then output TRUE.
  - **(c)** Otherwise, choose $g : \{0..W\text{–}1\} \to \{0,1\}$ as follows: for every $i \in \{0..W\text{–}1\}$, if $f(i) = 0$ then set $g(i) = 0$ and otherwise set $g(i)$ to be a random value in $\{0,1\}$.
  - **(d)** If $L(g) = 0$ then output TRUE; otherwise output FALSE.

  Prove that if L is a long-code codeword (i.e., $L = \mathsf{LC}(i)$ for some i) then T outputs TRUE with probability one.

  Prove that if L is a *linear function* that is non-zero and not a long code codeword then T outputs TRUE with probability at most 0.9.
- **(e)** Prove that LC is *locally testable*. That is, show an algorithm that given random access to a function $L : \{0,1\}^W \to \{0,1\}$ outputs TRUE with probability one if L is a long-code codeword and outputs FALSE with probability at least $1/2$ if L is not 0.9-close to a long-code codeword, while making at most a constant number of queries to L. H466
- **(f)** Using the test above, give an alternative proof for the Alphabet Reduction Lemma (Lemma 22.6). H466

22.6 ([For89, Fei91]) Consider the following 2CSP instance φ on an alphabet of size 4 (which we identify with $\{0,1\}^2$). The instance φ has 4 variables $x_{0,0}, x_{0,1}, x_{1,0}, x_{1,1}$ and four constraints $C_{0,0}, C_{0,1}, C_{1,0}, C_{1,1}$. The constraint $C_{a,b}$ looks at the variables $x_{0,a}$ and $x_{1,b}$ and outputs TRUE if and only if $x_{0,a} = x_{1,b}$ and $x_{0,a} \in \{0a, 1b\}$.

- (a) Prove that $\mathsf{val}(\varphi^{*2}) = \mathsf{val}(\varphi)$, where φ^{*t} denotes the 2CSP over alphabet W^t that is the t-times parallel repeated version of φ as in Section 22.3.1. H466
- (b) Prove that for every t, $\mathsf{val}(\varphi^{*t}) \geq \mathsf{val}(\varphi)^{t/2}$.

22.7 (Solvability of Unique Games) We encountered unique games in Section 22.9.4; it is a special case of $2\mathsf{CSP}_W$ in which the constraint function h is a *permutation* on $[W]$. In other words, if constraint φ_r involves variables i, j, then assignment $u_1, u_2, \ldots, u_n$ to the variables satisfies this constraint iff $u_j = h(u_i)$. Prove that there is a polynomial-time algorithm that given such an instance, finds a satisfying assignment if one exists.

22.8 Prove Corollary 22.25.

22.9 Prove that the **PCP** system resulting from the proof of Claim 22.36 (Chapter 11) satisfies the projection property.

22.10 This question explores the notion of noise-senstivity of Boolean functions, which ties in to the discussion in Note 22.21. Let $f : \{\pm 1\}^n \to \{\pm 1\}$ and let $I \subseteq [n]$. Let M_I be the following distribution: we choose $z \in_R M_I$ by for $i \in I$, choose z_i to be $+1$ with probability $1/2$ and -1 with probability $1/2$ (independently of other choices), for $i \notin I$ choose $z_i = +1$. We define the *variation of f on I* to be $\Pr_{\mathbf{x}\in_R\{\pm1\}^n,\mathbf{z}\in_R M_I}[f(\mathbf{x}) \neq f(\mathbf{xz})]$.

Suppose that the variation of f on I is less than ϵ. Prove that there exists a function $g : \{\pm 1\}^n \to \mathbb{R}$ such that **(1)** g does not depend on the coordinates in I and **(2)** g is 10ϵ-close to f (i.e., $\Pr_{\mathbf{x}\in_R\{\pm1\}^n}[f(\mathbf{x}) \neq g(\mathbf{x})] < 10\epsilon$). Can you come up with such a g that outputs values in $\{\pm 1\}$ only?

22.11 For $f : \{\pm 1\}^n \to \{\pm 1\}$ and $\mathbf{x} \in \{\pm 1\}^n$ we define $N_f(\mathbf{x})$ to be the number of coordinates i such that if we let y to be $\mathbf{x}$ flipped at the i^{th} coordinate (i.e., $y = x\mathbf{e}^i$ where $\mathbf{e}^i$ has -1 in the i^{th} coordinate and $+1$ everywhere else) then $f(\mathbf{x}) \neq f(\mathbf{y})$. We define the *average sensitivity* of f, denoted by $as(f)$ to be the expectation of $N_f(\mathbf{x})$ for $\mathbf{x} \in_R \{\pm 1\}^n$.

(a) Prove that for every balanced function $f : \{\pm 1\}^n \to \{\pm 1\}$ (i.e., $\Pr[f(\mathbf{x}) = +1] = 1/2$), $as(f) \geq 1$.

(b) Let f be balanced function from $\{\pm 1\}^n$ to $\{\pm 1\}$ with $as(f) = 1$. Prove that f is a coordinate function or its negation (i.e., $f(\mathbf{x}) = x_i$ or $f(\mathbf{x}) = -x_i$ for some $i \in [n]$ and for every $\mathbf{x} \in \{\pm 1\}^n$). (*Restatement using the language of isoperimetry as in Note 22.21*: If a subset of half the vertices of the hypercube $\{0,1\}^n$ has exactly 2^{n-1} edges leaving it, then there is some i such that this subset is simply the set of vertices where $x_i = 0$ (or $x_i = 1$).)

22.12 ([KM91]) This exercise asks you to give an alternative proof of the Goldreich-Levin Theorem 9.12 using Fourier analysis.

(a) For every function $f : \{\pm 1\}^n \to \mathbb{R}$, denote $\tilde{f}_{\alpha\star} = \sum_{\beta\in\{0,1\}^{n-k}} \hat{f}^2_{\alpha\circ\beta}$, where $\circ$ denotes concatenation and we identify strings in $\{0,1\}^n$ and subsets of $[n]$ in the obvious way. Prove that

$$\tilde{f}_{0^k\star} = \mathop{\mathsf{E}}_{\substack{\mathbf{x},\mathbf{x}'\in_R\{0,1\}^k \\ \mathbf{y}\in_R\{0,1\}^{n-k}}} [f(\mathbf{x}\circ\mathbf{y})f(\mathbf{x}'\circ\mathbf{y})]$$

H466

(b) Prove that for every $\alpha \in \{0,1\}^k$,

$$\tilde{f}_{\alpha\star} = \mathop{\mathsf{E}}_{\substack{\mathbf{x},\mathbf{x}'\in_R\{0,1\}^k \\ \mathbf{y}\in_R\{0,1\}^{n-k}}} [f(\mathbf{x}\circ\mathbf{y})f(\mathbf{x}'\circ\mathbf{y})\chi_\alpha(\mathbf{x})\chi_\alpha(\mathbf{x}')] \tag{13}$$

H466

(c) Show an algorithm `Estimate` that given $\alpha \in \{0,1\}^k$, $\epsilon > 0$ and oracle access to $f : \{\pm 1\}^n \to \{\pm 1\}$, runs in time $\mathrm{poly}(n, 1/\epsilon)$ and outputs an estimate of f_α within ϵ accuracy with probability $1 - \epsilon$. H466

(d) Show an algorithm `LearnFourier` that given $\epsilon > 0$ and oracle access to $f : \{\pm 1\}^n \to \{\pm 1\}$, runs in $\mathrm{poly}(n, 1/\epsilon)$ time and outputs a set L of $\mathrm{poly}(1/\epsilon)$ strings such that with probability at least 0.9, for every $\alpha \in \{0,1\}^n$, if $|\hat{f}_\alpha| > \epsilon$ then $\alpha \in L$. H466

(e) Show that the above algorithm implies Theorem 9.12.

22.13 Prove Lemma 22.33, albeit using a randomized algorithm. H466

22.14 Derandomize the algorithm of the previous exercise. H466

22.15 ([ABSS93]) In Problem 11.16 we explored the approximability of the problem of finding the largest feasible subsystem in a system of linear equations over the rationals. Show that there is an $\epsilon > 0$ such that computing an $n^{-\epsilon}$-approximation to this problem is **NP**-hard. H466

22.16 ([PY88]) Suppose we restrict attention to MAX-3SAT in which each variable appears in at most 5 clauses. Show that there is still a constant $\rho < 1$ such that computing a ρ-aproximation to this problem is **NP**-hard. H466

22.17 ([PY88]) In the MAX-CUT problem we are given a graph $G = (V, E)$ and seek to partition the vertices into two sets $S, \overline{S}$ such that we maximize the number of edges $\left|E(S, \overline{S})\right|$ between them. Show that there is still a constant $\rho < 1$ such that computing a ρ-aproximation to this problem is **NP**-hard.

22.A Transforming qCSP instances into "nice" instances.

We can transform a qCSP-instance φ into a "nice" 2CSP-instance ψ through the following three claims:

Claim 22.36 *There is a CL- reduction mapping any qCSP instance φ into a $2\mathsf{CSP}_{2^q}$ instance ψ such that*

$$\mathsf{val}(\varphi) \leq 1-\epsilon \Rightarrow \mathsf{val}(\psi) \leq 1-\epsilon/q \qquad \diamondsuit$$

PROOF: Given a qCSP-instance φ over n variables $u_1,\ldots,u_n$ with m constraints, we construct the following $2\mathsf{CSP}_{2^q}$ formula ψ over the variables $u_1,\ldots,u_n,y_1,\ldots,y_m$. Intuitively, the y_i variables will hold the restriction of the assignment to the q variables used by the i^{th} constraint, and we will add constraints to check consistency: that is to make sure that if the i^{th} constraint depends on the variable u_j then u_j is indeed given a value consistent with y_i. Specifically, for every φ_i of φ that depends on the variables $u_1,\ldots,u_q$, we add q constraints $\{\psi_{i,j}\}_{j\in[q]}$ where $\psi_{i,j}(y_i,u_j)$ is true iff y_i encodes an assignment to $u_1,\ldots,u_q$ satisfying φ_i and u_j is in $\{0,1\}$ and agrees with the assignment y_i. Note that the number of constraints in ψ is qm.

Clearly, if φ is satisfiable then so is ψ. Suppose that $\mathsf{val}(\varphi)\leq 1-\epsilon$ and let $u_1,\ldots,u_k,y_1,\ldots,y_m$ be any assignment to the variables of ψ. There exists a set $S\subseteq[m]$ of size at least ϵm such that the constraint φ_i is violated by the assignment $u_1,\ldots,u_k$. For any $i\in S$ there must be at least one $j\in[q]$ such that the constraint $\psi_{i,j}$ is violated. ■

Claim 22.37 *There is an absolute constant d and a CL- reduction mapping any $2\mathsf{CSP}_W$ instance φ into a $2\mathsf{CSP}_W$ instance ψ such that*

$$\mathsf{val}(\varphi) \leq 1-\epsilon \Rightarrow \mathsf{val}(\psi) \leq 1-\epsilon/(100Wd).$$

and the constraint graph of ψ is d-regular. That is, every variable in ψ appears in exactly d constraints. $\diamondsuit$

PROOF: Let φ be a $2\mathsf{CSP}_W$ instance, and let $\{G_n\}_{n\in\mathbb{N}}$ be an explicit family of d-regular expanders. Our goal is to ensure that each variable appears in φ at most $d+1$ times (if a variable appears less than that, we can always add artificial constraints that touch only this variable). Suppose that u_i is a variable that appears in k constraints for some $n>1$. We will change u_i into k variables $y_i^1,\ldots,y_i^k$, and use a different variable of the form y_i^j in the place of u_i in each constraint u_i originally appeared in. We will also add a constraint requiring that y_i^j is equal to $y_i^{j'}$ for every edge (j,j') in the graph G_k. We do this process for every variable in the original instance, until each variable appears in at most d equality constraints and one original constraint. We call the resulting 2CSP-instance ψ. Note that if φ has m constraints then ψ will have at most $m+dm$ constraints.

Clearly, if φ is satisfiable then so is ψ. Suppose that $\mathsf{val}(\varphi)\leq 1-\epsilon$ and let $\mathbf{y}$ be any assignment to the variables of ψ. We need to show that $\mathbf{y}$ violates at least $\frac{\epsilon m}{100W}$ of the constraints of ψ. Recall that for each variable u_i that appears k times in φ, the assignment $\mathbf{y}$ has k variables $y_i^1,\ldots,y_i^k$. We compute an assignment $\mathbf{u}$ to φ's variables as follows: u_i is assigned the plurality value of $y_i^1,\ldots,y_i^k$. We define t_i to be the number of y_i^j's that *disagree* with this plurality value. Note that $0\leq t_i\leq k(1-1/W)$ (where W is the alphabet size). If $\sum_{i=1}^n t_i\geq\frac{\epsilon}{4}m$ then we are done. Indeed, by (1) (see Section 22.2.3), in this case we will have at least $\sum_{i=1}^n\frac{t_i}{10W}\geq\frac{\epsilon}{40W}m$ equality constraints that are violated.

Suppose now that $\sum_{i=1}^n t_i<\frac{\epsilon}{4}m$. Since $\mathsf{val}(\varphi)\leq 1-\epsilon$, there is a set S of at least ϵm constraints violated in φ by the plurality assignment $\mathbf{u}$. All of these constraints are also present in ψ and since we assume $\sum_{i=1}^n t_i<\frac{\epsilon}{4}m$, at most half of them are given a different value by the assignment $\mathbf{y}$ than the value given by $\mathbf{u}$. Thus the assignment $\mathbf{y}$ violates at least $\frac{\epsilon}{2}m$ constraints in ψ. ■

Claim 22.38 *There is an absolute constant d and a CL-reduction mapping any $2\mathsf{CSP}_W$ instance φ with d'-regular constraint graph for $d\geq d'$ into a $2\mathsf{CSP}_W$ instance ψ such that*

$$\mathsf{val}(\varphi) \leq 1-\epsilon \Rightarrow \mathsf{val}(\psi) \leq 1-\epsilon/(10d)$$

and the constraint graph of ψ is a $4d$-regular expander, with half the edges coming out of each vertex being self-loops. ♢

PROOF: There is a constant d and an explicit family $\{G_n\}_{n\in\mathbb{N}}$ of graphs such that for every n, G_n is a d-regular n-vertex 0.1-expander graph (See Section 22.2.3).

Let φ be a 2CSP-instance as in the claim's statement. By adding self loops, we can assume that the constraint graph has degree d (this can at worst decrease the gap by factor of d). We now add "null" constraints (constraints that always accept) for every edge in the graph G_n. In addition, we add $2d$ null constraints forming self-loops for each vertex. We denote by ψ the resulting instance. Adding these null constraints reduces the fraction of violated constraints by a factor at most four. Moreover, because any regular graph H satisfies $\lambda(H) \leq 1$ and because of λ's subadditivity (see Exercise 21.7, Chapter 21), $\lambda(\psi) \leq \frac{3}{4} + \frac{1}{4}\lambda(G_n) \leq 0.9$ where by $\lambda(\psi)$ we denote the parameter λ of ψ's constraint graph. ■

Chapter 23

Why are circuit lower bounds so difficult?

Why have we not been able to prove strong lower bounds for general circuits? Despite the dramatic success in proving lower bounds on restricted circuit classes as described in Chapter 14, we seem utterly at a loss comes to showing limitations of general Boolean circuits.

In 1994 Razborov and Rudich [RR94] described what they view as the main technical limitation of current approaches for proving circuit lower bounds. They defined a notion of "natural mathematical proofs" for a circuit lower bound. They pointed out that current lower bound arguments involve such mathematical proofs, and showed that obtaining strong lower bound with such proof techniques would violate a stronger form of the $\mathbf{P} \neq \mathbf{NP}$ conjecture —specifically, the conjecture that strong one-way functions exist which cannot be inverted by algorithms running in subexponential time. Since current evidence suggests that such strong one-way functions do exist (factoring integers, discrete log, etc., as described in Chapter 9), we conclude that current techniques are inherently incapable of proving strong lower bounds on general circuits.

The Razborov-Rudich result may be viewed as a modern analogue of the 1970's results on the limits of diagonalization (see Chapter 3). What is particularly striking is that computational complexity (namely, the existence of strong one-way functions) is used here to shed light on a metamathematical question about computational complexity: "Why have we been unable to prove $\mathbf{P} \neq \mathbf{NP}$?" This is a good example of our claim at the start of the book that computational tractability has an intimate connection with issues of mathematical tractability amd proveability.

This chapter is organized as follows. We define natural proofs in Section 23.1, and then in Section 23.2 we discuss why they are indeed "natural" . We then prove in Section 23.3 that under widely-believed assumptions, such techniques will not be able to prove $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$.

Can we design lower bound techniques that circumvent the "natural proof barrier?" We describe an interesting such example in Section 23.4. We end in Section 23.5 with some philosophical musings on the meaning of the natural proof barrier, and our personal viewpoint.

23.1 Definition of natural proofs

Let $f : \{0,1\}^n \to \{0,1\}$ be some Boolean function and $c \geq 1$ be some number. Any proof that f does not have n^c-sized circuits can be viewed as exhibiting some property that f has, and that every function with an n^c-sized circuit does not possess. That is, such a proof can be viewed as providing a predicate $\mathcal{P}$ on Boolean functions such that $\mathcal{P}(f) = 1$ but

$$\mathcal{P}(g) = 0 \text{ for every } g \in \mathbf{SIZE}(n^c) \ . \tag{1}$$

We call the condition (1) n^c-*usefulness*. We say such a predicate $\mathcal{P}$ is *natural* if it satisfies in addition the following two conditions:

Constructiveness: There is an $2^{O(n)}$ time algorithm that on input (the truth table of) a function $g : \{0,1\}^n \to \{0,1\}$ outputs $\mathcal{P}(g)$. Note that the truth table has size 2^n, so this algorithm runs in time that is polynomial in its input size.

Largeness: The probability that a random function $g : \{0,1\}^n \to \{0,1\}$ satisfies $\mathcal{P}(g) = 1$ is at least $1/n$.

We will discuss in Section 23.2 the motivation behind these conditions, but for now note that the largeness condition does not contradict the n^c-usefulness condition since only a very small fraction of functions have polynomial-sized circuits (see the proof of Theorem 6.21). The following theorem says that, under reasonable assumptions, natural proofs cannot be used to prove that a function is not in $\mathbf{P}_{/\mathbf{poly}}$:

Theorem 23.1 *(Natural proofs* [RR94]*)*
Suppose that sub-exponentially strong one-way functions exist. Then there exists a constant $c \in \mathbb{N}$ such that there is no n^c-useful natural predicate $\mathcal{P}$.

One-way functions were defined in Chapter 9 (Section 9.2) and by a sub-exponentially strong one-way function we mean one that resists inverting even by a 2^{n^ϵ}-time adversary for some fixed $\epsilon > 0$. It is widely believed that such one-way functions exist. We defer the proof of Theorem 23.1 to Section 23.3, but first discuss why such predicates do deserve the name "natural".

Example 23.2
To develop some understanding of the definition of natural proofs, let us consider two predicates.
The first is $\mathcal{P}(g) = 1$ iff g is a Boolean function on n bits that has circuit complexity more than $n^{\log n}$. This predicate is n^c-useful for every constant c since $n^c = o(n^{\log n})$. The predicate also satisfies largeness since a random Boolean function satisfies it with probability almost 1 (see the proof of Theorem 6.21). However, we do not know if this predicate is *constructive*, since the trival algorithm for computing it would involve enumerating all circuits of size $n^{\log n}$, which requires $2^{n^{\log n}}$ time.
The second example is $\mathcal{P}'(g) = 1$ iff g correctly solves the decision problem 3SAT on inputs of size n. This function is *constructive*: to compute it, simply enumerate all inputs of size n, and verify using a trivial 2^n-time algorithm for 3SAT that g gives the correct answer on all inputs. If $\mathsf{3SAT} \notin \mathbf{SIZE}(n^c)$ (an open problem of course) then $\mathcal{P}'$ satisfies n^c-usefulness since it is 0 on all functions in $\mathbf{SIZE}(n^c)$. However, $\mathcal{P}'$ does not satisfy largeness since it is 1 for only one function.

23.2 What's so natural about natural proofs?

Now we recall some of the circuit lower bounds we proved earlier and check that they implicitly involve natural proofs. (This of course is the justification for the name "natural.")

Example 23.3 *($\mathbf{AC^0}$)*
The result that the parity function is not computable in $\mathbf{AC}^0$ (Section 14.1) involved the following steps: **(a)** Showing that every $\mathbf{AC}^0$ circuit can be simplified by restricting at most $n - n^\epsilon$ input bits so that it then becomes a constant function and **(b)** Showing that the parity function cannot be made constant by restricting at most $n - n^\epsilon$ of its input bits.
Clearly, we can verify whether the property defined in **(a)** holds for a function $f : \{0,1\}^n \to \{0,1\}$ in $2^{O(n)}$ time — just enumerate all possible choices for the subsets of variables and all ways of setting them to 0/1. Thus, this proof satisfies the "constructiveness" condition. Moreover, it's not hard to show that a random function also cannot be made constant by fixing at most $n - n^\epsilon$ of its input bits (see Exercise 23.2), and so this proof satisfies the "largeness' condition as well.

Example 23.4 *(Two-party communication complexity)*
To prove that f has high 2-party communication complexity, it suffices to prove that the $n \times n$ matrix $M(f)$ introduced in Chapter 13 (namely, one whose (x, y) entry is $f(x, y)$) has no large subrectangle that is monochromatic. Now imagine the algorithmic complexity of checking this condition, where the input to the algorithm is $M(f)$ (i.e., a string of length 2^{2n}). The statement "$M(f)$ has no $k \times l$ monochromatic rectangle" is a **coNP** statement, and in fact is **coNP**-complete for general f (it's equivalent to the bipartite clique problem). However, the lower bound methods considered in Chapter 13 such as computing the rank or eigenvalues involve polynomial-time computation, which mean that they satisfy the "constructiveness" condition. The lower bound method using discrepancy is actually not a polynomial-time computation, but discrepancy can also be approximated within a factor $O(1)$ in polynomial time (see the notes of Chapter 13) and hence this proof satisfies the "constructiveness" condition as well.
Moreover, all of the conditions used in these lower bounds, namely having small second-largest eigenvalue, high rank, or low discrepancy, are satisfied by a random matrix with high probability, and hence all these proofs satisfy the "largeness" condition as well.

We see that many lower bounds do use natural proofs, and in fact it turns out that all the known "combinatorial" circuit lower bounds are natural (e.g., lower bounds such as the ones in chapters 12–16 that argue directly about the structure of circuits or protocols). But is there a more general principle at work here? Is there some inherent reason why lower bounds should satisfy the constructiveness and largeness conditions?

23.2.1 Why constructiveness?

Note that there is an old controversy within mathematics about "nonconstructive" proofs, whereby the existence of an object is established (usually by some argument about infinite sets) without giving an explicit algorithm for constructing the object. Most mathematicians today are completely comfortable with nonconstructive proofs.

In the context of natural proofs, we are insisting upon a much stronger form of "constructiveness"— the proof must not only yield a finite algorithm, but in fact a polynomial-time algorithm. Many proofs that would be "constructive" for a mathematician would be nonconstructive under our definition. Surprisingly, even with this stricter definition, proofs in combinatorial mathematics are usually constructive, and the same is true of current circuit lower bounds as well.

In fact, circuit lower bounds usually rely upon techniques from combinatorics and in general, combinatorics techniques tend to be constructive in our sense of the word. In

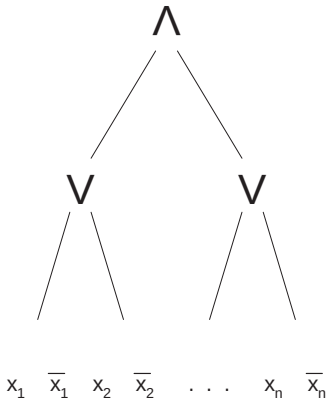


Figure 23.1 A Boolean formula.

a few cases, combinatorial results initially proved "nonconstructively" later turned out to have constructive proofs: a famous example is the Lovàsz Local Lemma (discovered in 1975 [EL75]; algorithmic version discovered in 1991 [Bec91]). The same is true for several circuit lower bounds— Razborov and Rudich found a "naturalized" version of the lower bound for $\mathbf{ACC}^0[q]$ of Section 14.2, and Raz [Raz00] gave a natural proof (presented in Section 13.3) of the lower bound for multiparty communication complexity originally proved non-constructively by Babai et al in 1992 [BNS89].

Though non-constructive techniques do exist is combinatorics— probabilistic method, nullstellensatz, topological arguments, etc.— we have not been able to use them to find better lower bounds for explicit functions. For speculative musings on these topics, please see Section 23.5.

23.2.2 Why largeness?

Why should a lower bound for a specific function, whether it's parity or 3SAT, use a property that holds with good probability for a random function as well? Below, we try to formalize this. The intuition in a nutshell is that every proof that a specific function $f_0 : \{0,1\}^n \to \{0,1\}$ does not have a size S circuit, actually implies that at least half of the functions from $\{0,1\}^n$ to $\{0,1\}$ do not have a circuit of size $S/2 - 10$. The reason is that if we choose a random $g : \{0,1\}^n \to \{0,1\}$, and write $f_0 = (f_0 \oplus g) \oplus g$ (where $g \oplus h$ denotes the function that maps every input x to $g(x) \oplus h(x)$), then we see that if both $(f_0 \oplus g)$ and g have circuits of size $< S/2 - 10$ then f_0 has a circuit of size $< S$. Since both g and $(f_0 \oplus g)$ are uniformly distributed, it follows that a lower bound on the circuit complexity of f_0 implies a lower bound on the complexity of half the functions.

23.2.3 Natural proofs from complexity measures

More generally, a large class of lower bound techniques turns out to yield properties that simultaneously satisfy the constructiveness and largeness properties (i.e., are natural). For concreteness, let us focus on Boolean formulae (see Figure 23.1), that are Boolean circuits where gates have indegree 2 and outdegree 1. It is tempting to prove a lower bound using some kind of induction. Suppose we have a function that we believe to be "complicated," in the sense that it requires a large Boolean formula to compute. Since the function computed at the output is "complicated", at least one of the functions on the incoming edges to the output gate should also be "pretty complicated" (after all those two functions can be combined with a single gate to produce a "complicated" function). Now we try to

formalize this intuition, and point out why one ends up proving a lower bound on the formula complexity of random functions.

The most obvious way to formalize a "complicatedness" is as a function μ that maps every Boolean function on $\{0,1\}^n$ to a nonnegative integer. We say that μ is a *formal complexity measure* if it satisfies the following properties: First, the measure is low for trivial functions: $\mu(x_i) \leq 1$ and $\mu(\bar{x}_i) \leq 1$ for all i. Second, we require that

- $\mu(f \wedge g) \leq \mu(f) + \mu(g)$ for all f, g; and
- $\mu(f \vee g) \leq \mu(f) + \mu(g)$ for all f, g.

For instance, the following function ρ is trivially a formal complexity measure

$$\rho(f) = 1 + \text{the smallest formula size for } f. \tag{2}$$

In fact, it is easy to prove the following by induction.

Theorem 23.5 *If μ is any formal complexity measure, then $\mu(f)$ is a lower bound on the formula complexity of f.* ♢

Thus to formalize the inductive approach outlined earlier, it suffices to define a measure μ such that, say, $\mu(\mathsf{3SAT})$ is super-polynomial. For example, one could try "fraction of inputs for which the function agrees with the 3SAT function" or some suitably modified version of this. In general, one imagines that defining a measure that lets us prove a good lower bound for 3SAT would involve some deep observation about the 3SAT function. The next lemma seems to show, however, that even though all we care about is the 3SAT function, our lower bound necessarily must reason about random functions.

Lemma 23.6 *Suppose μ is a formal complexity measure and there exists a function $f : \{0,1\}^n \to \{0,1\}$ such that $\mu(f) \geq S$ for some large number S. Then for at least $1/4$ of all functions $g : \{0,1\}^n \to \{0,1\}$ we must have $\mu(g) \geq S/4$.* ♢

Proof: The proof follows by the same observation as above. For a random function $g : \{0,1\}^n \to \{0,1\}$, we can write $f = h \oplus g$ where $h = f \oplus g$. So $f = (\bar{h} \wedge g) \vee (h \wedge \bar{g})$ and $\mu(f) \leq \mu(g) + \mu(\overline{g}) + \mu(h) + \mu(\overline{h})$. But if more than $3/4$ of the functions have measure less than $S/4$, then by the union bound with positive probability all four functions $g, h, \overline{g}, \overline{h}$ will have measure less than $S/4$, implying that $\mu(f) < S$ and contradicting our assumption. ■

In fact, the following stronger theorem holds:

Theorem 23.7 *If $\mu(f) > S$ then for all $\epsilon > 0$ and for at least $1 - \epsilon$ of all functions g we have that,*

$$\mu(g) \geq \Omega\left(\frac{S}{(n + \log(1/\epsilon))^2}\right). \tag{♢}$$

The idea behind the proof of the theorem is to write f as the Boolean combination of a small number of functions and then proceed similarly as in the proof of the lemma. These results mean that every lower bound that is obtained through a $2^{O(n)}$-time computable formal complexity measure μ will be natural.

23.3 Proof of Theorem 23.1

Now we prove Theorem 23.1. We will use the key fact from Section 9.5.1 that we can build from every pseudorandom generator a *pseudorandom function family*. Recall that this is a family of functions $\{f_s\}_{s \in \{0,1\}^*}$, where for $s \in \{0,1\}^m$, f_s is a function from $\{0,1\}^m$ to $\{0,1\}$. This family has the following two properties: **(a)** there is a polynomial-time algorithm that given s, x outputs $f_s(x)$ and **(b)** no polynomial-time algorithm can distinguish with non-negligible probability between oracle access to the function $f_s(\cdot)$ for a randomly chosen $s \in_R \{0,1\}^m$ and oracle access to a random function from $\{0,1\}^m$ to $\{0,1\}$.

Since pseudorandom generators can be based on any one-way function [HILL99], we can obtain such a family from this assumption as well. In fact, one can verify by going over these reductions that if we start from a one-way function that cannot be inverted by 2^{n^ϵ}-time algorithms for some constant $\epsilon > 0$ then we obtain a pseudorandom function family $\{f_s\}_{s\in\{0,1\}^*}$ such that $f_s(\cdot)$ for $s \in_R \{0,1\}^m$ cannot be distinguished from a random function by $2^{m^{\epsilon'}}$-time algorithms for some constant ϵ'.

What does all this have to do with natural proofs? Suppose $\mathcal{P}$ is a natural property on n-bit functions that is n^c-useful. It is an algorithm (albeit one running in $2^{O(n)}$ time) that (a) outputs 0 on functions with circuit complexity lower than n^c. (b) outputs 1 on a nonnegligible fraction of functions. Thus one can hope that such an algorithm allows us some nonnegligible chance of distinguishing a pseudorandom function from a truly random function, and this is what we show now.

Let $\{f_s\}$ be a 2^{m^ϵ}-secure pseudorandom function collection as described earlier. We use the natural property $\mathcal{P}$ to design an algorithm that distinguishes between a random function from $\{0,1\}^m$ to $\{0,1\}$ and f_s (to both of which it has oracle access) with nonnegligible probability.

Given oracle access to an unknown function h (which could either be f_s for some s or a random function), the distinguisher lets n be $m^{\epsilon/2}$ and constructs the truth table of the function g from $\{0,1\}^n$ to $\{0,1\}$ defined as: $g(x) = h(x0^{m-n})$. Constructing this truth table only requires $2^{O(n)}$ time. Then the distinguisher runs algorithm $\mathcal{P}$ on this function, and outputs whatever $\mathcal{P}$ does. Now consider the two cases under consideration. In one case, the provided function h was a random function, and so this new function g is also a random function on $\{0,1\}^n$. Hence the probability that $\mathcal{P}$ outputs 1 is at least $1/n$. In the other case, the provided function h was f_s for some s. Then function g has circuit complexity at most n^c since the map $s, x \mapsto f_s(x)$ can be computed in $\text{poly}(m)$ time, and hence the map $x \mapsto g(x)$ is computable by circuit of size $\text{poly}(m) = n^c$ that has s "hardwired" into it. (To be sure, the distinguisher does not know s or this circuit; we are only asserting that the circuit *exists*.) Hence $\mathcal{P}$ given the truth table for g must output 0.

Thus the distinguisher distinguishes between f_s and a random function with probability at least $1/n$ and furthermore does so in $2^{O(n)}$ time, which is less than 2^{m^ϵ}. Viewed contrapositively, this implies that if the pseudorandom function was subexponentially strong, then the natural property cannot exist. ■

23.4 An "unnatural" lower bound

Can we prove circuit lower bounds using proofs that are not natural? Here we show an interesting example that uses (among other ideas) simple old diagonalization! After presenting the result we comment more on why it is not natural.

To present this result we'll need the notion of a *promise problem*, which is a partially defined Boolean function from $\{0,1\}^*$ to $\{0,1\}$. That is, we can think of such a problem as a function $f : \{0,1\}^* \to \{0,1,\perp\}$ where $\perp$ represents "undefined". We say that an algorithm A solves a promise problem f, if whenever $f(x) \in \{0,1\}$ then $A(x) = f(x)$, but we make no requirement on A's output when $f(x) = \perp$. We can generalize the definition of every complexity class to promise problems, and in particular denote by **PromiseMA** the corresponding generalization of the class **MA** defined in Section 8.2. That is, a promise problem f is in **PromiseMA** if there is a probabilistic polynomial-time algorithm A and a polynomial $p(\cdot)$ such that for every $x \in \{0,1\}^*$, **(a)** if $f(x) = 1$ then there exists $y \in \{0,1\}^{p(|x|)}$ such that $\Pr[A(x,y) = 1] \geq 2/3$ and **(b)** if $f(x) = 0$ then for every $y \in \{0,1\}^{p(|x|)}$, $\Pr[A(x,y) = 1] \leq 1/3$. We have the following lower bound:

Theorem 23.8 *([San07])*
For every $c \in \mathbb{N}$,

$$\mathbf{PromiseMA} \not\subseteq \mathbf{SIZE}(n^c) ,$$

where $\mathbf{SIZE}(n^c)$ denotes the set of promise problems with n^c-sized circuits.

PROOF: Recall that in the interactive proof for **PSPACE** shown in Section 8.3, the prover algorithm can be implemented itself in polynomial space. This means that if L is a **PSPACE**-complete problem, then there is an interactive proof for L where the prover can prove that a string x is in L using polynomial time and oracle access to the language L itself. In fact, it turns out that there is a such a language L_0 where on inputs of length n the prover needs only to make queries of length at most n [TV02]. This means that if this language L_0 can be decided by a circuit of size $S(n)$, then the prover can simply send this circuit, which a probabilistic verifier can then use to run the interactive protocol on its own. Therefore, we see that if $L_0 \in \mathbf{SIZE}(S(n))$ then there's a poly$(S(n))$-time **MA** protocol for L_0. (We saw similar reasoning in Theorem 8.22 and Lemma 20.18.)

Define $S(n)$ to be one plus the size of the smallest circuit that solves L_0 on length-n inputs. Now if $S(n) \leq \text{poly}(n)$, this means that $\mathbf{PSPACE} \subseteq \mathbf{MA}$ but in this case **MA** clearly has a language outside of $\mathbf{SIZE}(n^c)$ for every c (see Exercise 6.5 of Chapter 6). In fact, the same reasoning holds even if there's a constant c such that $S(n) \leq n^c$ infinitely often, and so we may assume that $S(n) = n^{\omega(1)}$. Note that L_0 has a poly$(S(n))$-time **MA** protocol but has no $S(n)$-sized circuit. If only $S(n)$ was time constructible, we could "scale down" this separation by defining the language L_1 to be $\{x01^{S(|x|)^{1/c}-|x|-1} : x \in L_0\}$, implying that L_1 is in **MA** but not in $\mathbf{SIZE}(n^c)$. Unfortunately, we cannot assume that $S(n)$ is time constructible and hence cannot ensure that L_1 is in **MA**. Nonetheless, we can define the following *promise problem* f_1: it is defined only on inputs of the form $y = x01^{S(|x|)^{1/c}-|x|-1}$ and on such inputs $f_1(y) = L_0(x)$. It's not hard to see that $f_1 \in \mathbf{PromiseMA} \setminus \mathbf{SIZE}(n^c)$. ■

This proof is unnatural because underlying it is the proof that $\mathbf{PSPACE} \not\subseteq \mathbf{SIZE}(n^c)$ which uses diagonalization - an inherently unnatural technique that focuses on one very specific function and hence violates the largeness condition. Alternatively, one can also view a diagonalization proof as showing that a function has the property that it disagrees with every small circuit on some input— a property that satisfies largeness but not constructiveness. In fact, Theorem 23.1 shows that there are no natural proofs for Theorem 23.8, unless subexponentially strong one-way functions do not exist. It is also known that this lower bound does not relativize [Aar06]. Unfortunately, "pushing down" these diagonalization/arithmetization based techniques to obtain a lower bound on a function in **NP** seems very hard.

23.5 A philosophical view

We think that the natural proof idea and other negative results of this nature are very valuable. When one is stuck on a difficult question, it is useful to try to prove that it can't be solved, or can't be solved with particular methods. This can give additional insight on the question that might otherwise be very hard to obtain. By understanding the obstacles, we know what we'll have to tackle or bypass to solve certain problems, and this has proven to be extremely useful countless times in complexity theory and theoretical computer science at large. In this case, the natural proofs paradigm shows that any complexity class that has a plausible pseudorandom function generator is going to pose a problem to known lower bound techniques. Since even low classes like $\mathbf{NC}^1$ and TC_0 contain plausible pseudorandom functions, one gets a fairly good understanding of why the project of proving lower bounds ground to a halt at the class $\mathbf{ACC}^0$.

However, perhaps natural proofs have been so successful at encompassing known lower bound techniques, that this discouraged researchers from thinking too hard about circuit lower bounds. This need not be the case. There are techniques in combinatorics that do not satisfy either the constructiveness or the largeness properties. Personally, we feel that the constructiveness property may be easier to get around, and one sees this already in the nonnatural proof of the previous section. Looking more broadly at combinatorics, a relevant example is Lovàsz's lower bound of the chromatic number of the Kneser graph [Lov78]. Lower bounding the chromatic number is **coNP**-complete in general. Lovàsz gives a topological proof (using the famous Borsuk-Ulam fixed point theorem) that determines the chromatic number of the Kneser graph exactly. From his proof one can indeed obtain an algorithm for solving chromatic number on all graphs([MZ04]) —but it runs in **PSPACE** for general graphs! So if this were a circuit lower bound we could call it "nonconstructive." Nevertheless, Lovàsz's reasoning for the particular case of the Kneser graph is not overly complicated because the graph is highly symmetrical. This suggests we should not blindly trust the intuition that "nonconstructive $\equiv$ difficult." We should also remember the lesson learned from the results on limitations of relativizing techniques (Section 3.4). We've seen that one new non-relativizing technique— arithemtization— allowed us to prove a host of results in Chapters 8, 11, etc. that cannot be proven using relativizing techniques. It may very well be that a single new "unnatural" technique will open the floodgates for a great many lower bounds.

Chapter notes and history

The observation that circuit lower bounds may unwittingly end up reasoning about random functions first appears in Razborov [Raz89]'s result about the limitations of the method of approximation. We did not cover the full spectrum of ideas in the Razborov-Rudich paper [RR94], where it is observed that candidate pseudorandom function generators exist even in the class TC^0, which lies between $\mathbf{ACC}^0$ and $\mathbf{NC}^1$. Thus natural proofs will probably not allow us to separate even TC^0 from **P**. Razborov's observation about submodular measures in Exercise 23.4 below is important because many existing approaches for formula complexity use submodular measures; thus they will fail to even prove superlinear lower bounds. The lower bound of Section 23.4 is due to Santhanam [San07]; similar techniques were first used to show hierarchy theorems for probabilistic algorithms with small advice [Bar02, FS04, GST04].

In contrast with our limited optimism, Razborov himself expresses (in the introduction to [Raz03b]) a view that the obstacle posed by the natural proofs observation is very serious. He observes that existing lower bound approaches use weak theories of arithmetic such as Bounded Arithmetic. He conjectures that any circuit lower bound attempt in such a logical system must be natural (and hence unlikely to work). But there are several theorems even in discrete mathematics use reasoning (e.g., fixed point theorems like Borsuk-Ulam) that does not seem to be formalizable in Bounded Arithmetic, which is our reason for optimism. Some researchers are far more pessimistic: they fear that **P** versus **NP** may be independent of mathematics (say, of Zermelo-Fraenkel set theory). See Aaronson's survey [Aar03] for more on this issue.

Very recently, Aaronson and Wigderson [AW08] showed a new obstacles for complexity results called algebraization. A complexity class separation $\mathcal{C} \not\subseteq \mathcal{D}$ cannot be solved using "algebrizing techniques" if there is there is an oracle O such that $\mathcal{C}^{\tilde{O}} \subseteq \mathcal{D}^O$, where $\tilde{O}$ denotes the *low degree extension* of the Boolean function O to a larger field or ring such as the integers. Roughly speaking, algebrizing techniques capture all results such as $\mathbf{IP} = \mathbf{PSPACE}$ and the **PCP** theorems that are proven by arithmetization. In particular, the lower bound of Section 23.4 uses algebrizing techniques, but [AW08] show that one cannot prove even a superlinear lower bound on **NP** using such techniques.

Exercises

23.1 Prove Theorem 23.7.

23.2 Prove that a random function $g : \{0,1\}^n \to \{0,1\}$ satisfies $\mathcal{P}(g) = 1$ with high probability, where $\mathcal{P}$ is the property, defined in Example 23.3, that for no fixing of $n - n^\epsilon$ of g's turns g into a constant function.

23.3 Prove Wigderson's observation: There is no natural proof that the DISCRETE LOG problem (i.e., given a prime p, and $g, y \in \mathbb{Z}_p^*$, with $g \neq 1$, find $x \in \mathbb{Z}_p^*$ such that $y = g^p \pmod p$) requires circuits of 2^{n^ϵ} size for some constant $\epsilon > 0$. H466

23.4 (Razborov [Raz92]) A *submodular* complexity measure is a complexity measure that satisfies $\mu(f \vee g) + \mu(f \wedge g) \leq \mu(f) + \mu(g)$ for all functions f, g. Show that for every n-bit function f_n, such a measure satisfies $\mu(f_n) = O(n)$. H466

23.5 Let L be the language containing all triples $\langle \varphi, P, i \rangle$ such that the i^{th} bit of $\varphi \pmod P$ is equal to 1, where P is a number and φ is an expression involving constants, the arithmetic operations $+, -, \cdot$ and sum and product quantifiers of the form $\sum_{x_i \in \{0,1\}}$ or $\prod_{x_i \in \{0,1\}}$, satisfying the following property: if we sort $x_1, \ldots, x_n$ according to their order of appearance in φ, then for every variable x_i there is at most a single $\prod$ quantifier involving x_j (for $j > i$) appearing before the last occurrence of x_i in φ. Show that L is **PSPACE** complete and furthermore, there is an interactive proof for L where the prover algorithm runs in polynomial time using an oracle for L, and when proving that some $x \in \{0,1\}^n$ is in L it uses queries of length at most n to its oracle. H466

Appendix A

Mathematical Background.

This appendix reviews the mathematical notions used in this book. However, most of these are only used in few places, and so the reader might want to only quickly review Sections A.1 and A.2, and come back to the other sections as needed. In particular, apart from probability, the first part of the book essentially requires only comfort with mathematical proofs and some very basic notions of discrete math.

The topics described in this appendix are covered in greater depth in many texts and online sources. Almost all of the mathematical background needed is covered in a good undergraduate "discrete math for computer science" course as currently taught at many computer science departments. Some good sources for this material are the lecture notes by Papadimitriou and Vazirani [PV06], and the book of Rosen [Ros06].

The mathematical tool we use most often is discrete probability. Alon and Spencer [AS00b] is a great resource in this area. Also, the books of Mitzenmacher and Upfal [MU05] and Motwani and Raghavan [MR95] cover probability from a more algorithmic perspective.

Although knowledge of algorithms is not strictly necessary for this book, it would be quite useful. It would be helpful to review either one of the two recent books by Dasgupta et al [DPV06] and Kleinberg and Tardos [KT06] or the earlier text by Cormen et al [CLRS01]. This book does not require prior knowledge of computability and automata theory, but some basic familiarity with that theory could be useful: see Sipser's book [Sip96] for an excellent introduction. See Shoup's book [Sho05] for a computer-science introduction to algebra and number theory.

Perhaps *the* mathematical prerequisite needed for this book is a certain level of comfort with *mathematical proofs*. The fact that a mathematical proof has to be absolutely convincing does not mean that it has to be overly formal and tedious. It just has to be clearly written, and contain no logical gaps. When you write proofs try to be clear and concise, rather than using too much formal notation. Of course, to be absolutely convinced that some statement is true, we need to be certain of what that statement means. This why there is a special emphasis in mathematics (and this book) on very precise *definitions*. Whenever you read a definition, try to make sure you completely understand it, perhaps by working through some simple examples. Oftentimes, understanding the meaning of a mathematical statement is more than half the work to prove that it is true.

A.1 Sets, Functions, Pairs, Strings, Graphs, Logic.

Sets. A *set* contains a finite or infinite number of elements, without repetition or respect to order, for example $\{2, 17, 5\}$, $\mathbb{N} = \{1, 2, 3, \ldots\}$ (the set of natural numbers), $[n] = \{1, 2, \ldots, n\}$ (the set of natural numbers from 1 ro n), $\mathbb{R}$ (the set of real numbers). For a finite set A, we denote by $|A|$ the number of elements in A. Some operations on sets are: **(1)** *union*: $A \cup B = \{x : x \in A \text{ or } x \in B\}$, **(2)** *intersection* : $A \cap B = \{x : x \in A \text{ and } x \in B\}$, and **(3)** *set difference*: $A \setminus B = \{x : x \in A \text{ and } x \notin B\}$.

Functions. We say that f is a *function* from a set A to B, denoted by $f : A \to B$, if it maps any element of A into an element of B. If B and A are finite, then the number of possible functions from A to B is $|B|^{|A|}$. We say that f is *one to one* if for every $x, w \in A$ with $x \neq w$, $f(x) \neq f(w)$. If A,B are finite, the existence of such a function implies that $|A| \leq |B|$. We say that f is *onto* if for every $y \in B$ there exists $x \in A$ such that $f(x) = y$. If A, B are finite, the existence of such a function implies that $|A| \geq |B|$. We say that f is a *permutation* if it is both one-to-one and onto. For finite A, B, the existence of a permutation from A to B implies that $|A| = |B|$.

Pairs and tuples. If A,B are sets, then the $A \times B$ denotes the set of all ordered pairs $\langle a, b \rangle$ with $a \in A, b \in B$. Note that if A, B are finite then $|A \times B| = |A| \cdot |B|$. We can define similarly $A \times B \times C$ to be the set of ordered triples $\langle a, b, c \rangle$ with $a \in A, b \in B, c \in C$. For $n \in \mathbb{N}$, we denote by A^n the set $A \times A \times \cdots \times A$ (n times). We will often use the set $\{0,1\}^n$, consisting of all length-n sequences of bits (i.e., length n strings), and the set $\{0,1\}^* = \cup_{n \geq 0} \{0,1\}^n$ ($\{0,1\}^0$ has a single element: a binary string of length zero, which we call the empty word and denote by ε). As mentioned in Section 0.1 we can represent various objects (numbers, graphs, matrices, etc...) as binary strings, and use $\llcorner x \lrcorner$ (not to be confused with the floor operator $\lfloor x \rfloor$) to denote the representation of x. Moreover, we often drop the $\llcorner \lrcorner$ symbols and use x to denote both the object and its representation.

Graphs. A *graph* G consists of a set V of *vertices* (which we often assume is equal to the set $[n] = \{1, \ldots, n\}$ for some $n \in N$) and a set E of *edges*, which consists of unordered pairs (i.e., size two subsets) of elements in V. We denote the edge $\{u, v\}$ of the graph by $\overline{u\,v}$. For $v \in V$, the *neighbors* of v are all the vertices $u \in V$ such that $\overline{u\,v} \in E$. In a *directed graph*, the edges consist of *ordered pairs* of vertices, and to stress this we sometimes denote the edge $\langle u, v \rangle$ in a directed graph by $\overrightarrow{u\,v}$. One can represent an n-vertex graph G by its *adjacency matrix* which is an $n \times n$ matrix A such that $A_{i,j}$ is equal to 1 if the edge $\overrightarrow{i\,j}$ is present in G i^{th} and is equal to 0 otherwise. One can think of an undirected graph as a directed graph G that satisfies that for every u, v, G contains the edge $\overrightarrow{u\,v}$ if and only if it contains the edge $\overrightarrow{v\,u}$. Hence, one can represent an undirected graph by an adjecancy matrix that is *symmetric* ($A_{i,j} = A_{j,i}$ for every $i, j \in [n]$).

Boolean operators. A *Boolean variable* is a variable that can be either TRUE or FALSE (we sometimes identify TRUE with 1 and FALSE with 0). We can combine variables via the logical operations AND ($\wedge$), OR ($\vee$) and NOT ($\neg$, sometimes also denoted by an overline), to obtain *Boolean formulae*. For example, the following is a Boolean formulae on the variables u_1, u_2, u_3: $(u_1 \wedge \overline{u}_2) \vee \neg(u_3 \wedge \overline{u}_1)$. The definitions of the operations are the usual: $a \wedge b =$ TRUE if $a =$ TRUE and $b =$ TRUE and is equal to FALSE otherwise; $\overline{a} = \neg a =$ TRUE if $a =$ FALSE and is equal to FALSE otherwise; $a \vee b = \neg(\overline{a} \vee \overline{b})$. We sometimes use other Boolean operators such as the XOR ($\oplus$) operator, but they can be always replaced with the equivalent expression using $\wedge, \vee, \neg$ (e.g., $a \oplus b = (a \wedge \overline{b}) \vee (\overline{a} \wedge b)$). If φ is a formulae in n variables $u_1, \ldots, u_n$, then for any *assignment* of values $u \in \{\text{FALSE}, \text{TRUE}\}^n$ (or equivalently, $\{0,1\}^n$), we denote by $\varphi(u)$ the value of φ when its variables are assigned the values in u. We say that φ is *satisfiable* if there exists a u such that $\varphi(u) =$ TRUE.

Quantifiers. We will often use the *quantifiers* $\forall$ (for all) and $\exists$ (exists). That is, if φ is a condition that can be TRUE or FALSE depending on the value of a variable x, then we write $\forall_x \varphi(x)$ to denote the statement that φ is TRUE for *every* possible value that can be assigned to x. If A is a set then we write $\forall_{x \in A} \varphi(x)$ to denote the statement that φ is TRUE for every assignment for x from the set A. The quantifier $\exists$ is defined similarly. Formally, we say that $\exists_x \varphi(x)$ holds if and only if $\neg(\forall_x \neg\varphi(x))$ holds.

Big-Oh Notation. We will often use the big-Oh notation (i.e., $O, \Omega, \Theta, o, \omega$) as defined in Section 0.3.

A.2 Probability theory

A *finite probability space* is a finite set $\Omega = \{\omega_1, \ldots, \omega_N\}$ along with a set of numbers $p_1, \ldots, p_N \in [0,1]$ such that $\sum_{i=1}^{N} p_i = 1$. A random element is selected from this space by choosing ω_i with probability p_i. If x is chosen from the sample space Ω then we denote this by $x \in_R \Omega$. If no distribution is specified then we use the uniform distribution over the elements of Ω (i.e., $p_i = \frac{1}{N}$ for every i).

An *event* over the space Ω is a subset $A \subseteq \Omega$ and the *probability* that A occurs, denoted by $\Pr[A]$, is equal to $\sum_{i:\omega_i \in A} p_i$. To give an example, the probability space could be that of all 2^n possible outcomes of n tosses of a fair coin (i.e., $\Omega = \{0,1\}^n$ and $p_i = 2^{-n}$ for every $i \in [2^n]$) and the event A can be that the number of coins that come up "heads" (or, equivalently, 1) is even. In this case, $\Pr[A] = 1/2$ (exercise). The following simple bound —called the *union bound*—is often used in the book. For every set of events $A_1, A_2, \ldots, A_n$,

$$\Pr[\cup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} \Pr[A_i]. \tag{1}$$

Inclusion exclusion principle. The union bound is a special case of a more general principle. Indeed, note that if the sets $A_1, \ldots, A_n$ are not *disjoint* then the probability of $\cup_i A_i$ could be smaller than $\sum_i \Pr[A_i]$ since we are overcounting elements that appear in more than one set. We can correct this by substracting $\sum_{i<j} \Pr[A_i \cap A_j]$ but then we might be undercounting, since we subtracted elements that appear in at least 3 sets too many times. Continuing this process we get

Claim A.1 *(Inclusion-Exclusion principle) For every $A_1, \ldots, A_n$,*

$$\Pr[\cup_{i=1}^{n} A_i] = \sum_{i=1}^{n} \Pr[A_i] - \sum_{1 \leq i < j \leq n} \Pr[A_i \cap A_j] + \cdots + (-1)^{n-1} \Pr[A_1 \cap \cdots \cap A_n] .$$

Moreover, this is an alternating sum *which means that if we take only the first k summands of the right hand side, then this upper bounds the left-hand side if k is odd, and lower bounds it if k is even.* ♢

We sometimes use the following corollary of this claim, known as the Bonefforni Inequality:

Corollary A.2 *For every events $A_1, \ldots, A_n$,*

$$\Pr[\cup_{i=1}^{n} A_i] \geq \sum_{i=1}^{n} \Pr[A_i] - \sum_{1 \leq i < j \leq n} \Pr[A_i \cap A_j]$$

♢

A.2.1 Random variables and expectations.

A *random variable* is a mapping from a probability space to $\mathbb{R}$. For example, if Ω is as above (i.e., the set of all possible outcomes of n tosses of a fair coin), then we can denote by X the number of coins that came up heads.

The *expectation* of a random variable X, denoted by $\mathsf{E}[X]$, is its weighted average. That is, $\mathsf{E}[X] = \sum_{i=1}^{N} p_i X(\omega_i)$. The following simple claim follows from the definition:

Claim A.3 *(Linearity of expectation) For X, Y random variables over a space Ω, denote by $X + Y$ the random variable that maps ω to $X(\omega) + Y(\omega)$. Then,*

$$\mathsf{E}[X + Y] = \mathsf{E}[X] + \mathsf{E}[Y]$$

♢

This claims implies that the random variable X from the example above has expectation $n/2$. Indeed $X = \sum_{i=1}^{n} X_i$ where X_i is equal to 1 if the i^{th} coins came up heads and is equal to 0 otherwise. But clearly, $\mathsf{E}[X_i] = 1/2$ for every i.

For a real number α and a random variable X, we define αX to be the random variable mapping ω to $\alpha \cdot X(\omega)$. Note that $\mathsf{E}[\alpha X] = \alpha \mathsf{E}[X]$.

Example A.4

Suppose that we choose k random numbers $x_1, \ldots, x_k$ independently in $[n]$. What is the expected number of *collisions*: unordered pairs $\{i, j\}$ such that $x_i = x_j$? For every $i \neq j$, define the random variable $Y_{i,j}$ to equal 1 if $x_i = x_j$ and 0 otherwise. Since for every choice of x_i, the probability that $x_j = x_i$ is $1/n$, we have that $\mathsf{E}[Y_{i,j}] = 1/n$. The number of collisions is the sum of $Y_{i,j}$ over all $i \neq j$ in $[k]$. Thus, by linearity of expectation the expected number of collisions is

$$\sum_{1 \leq i < j \leq n} \mathsf{E}[Y_{i,j}] = \binom{k}{2} \frac{1}{n} .$$

This means that we expect at least one collision once $\binom{k}{2} \geq n$, which happens once k is larger than roughly $\sqrt{2n}$. This fact is often known as the *birthday paradox* because it explains the seemingly strange phenomenon that a class of more than 27 or so students is quite likely to have a pair of students sharing the same birthday, even though there are 365 days in the year.

Note that in contrast, if $k \ll \sqrt{n}$ then by the union bound, the probability there will be even one collision is at most $\binom{k}{2}/n \ll 1$.

Notes: (1) We sometimes also consider random variables whose range is not $\mathbb{R}$, but other sets such as $\mathbb{C}$ or $\{0,1\}^n$. (2) Also, we often identify a random variable X over the sample space Ω with the distribution $X(\omega)$ for $\omega \in_R \Omega$. For example, we may use both $\Pr_{x \in_R X}[x^2 = 1]$ and $\Pr[X^2 = 1]$ to denote the probability that for $\omega \in_R \Omega$, $X(\omega)^2 = 1$.

A.2.2 The averaging argument

The following simple fact can be surprisingly useful:

> *The Averaging Argument*: If $a_1, a_2, \ldots, a_n$ are some numbers whose average is c then some $a_i \geq c$.

Equivalently, we can state this in probabilistic terms as follows:

Lemma A.5 *("The Probabilistic Method") If X is a random variable which takes values from a finite set and $\mathsf{E}[X] = \mu$ then the event "$X \geq \mu$" has nonzero probability.* ♢

The following two facts are also easy to verify

Lemma A.6 *If $a_1, a_2, \ldots, a_n \geq 0$ are numbers whose average is c then the fraction of a_i's that are at least kc is at most $1/k$.* ♢

Lemma A.7 *("Markov's inequality") Any* non-negative *random variable X satisfies*

$$\Pr\left(X \geq k\,\mathsf{E}[X]\right) \leq \frac{1}{k}.$$ ♢

Can we give any meaningful upper bound on the probability that X is much *smaller* than its expectation? Yes, if X is bounded.

Lemma A.8 *If $a_1, a_2, \ldots, a_n$ are numbers in the interval $[0,1]$ whose average is ρ then at least $\rho/2$ of the a_i's are at least as large as $\rho/2$.* ♢

PROOF: Let γ be the fraction of i's such that $a_i \geq \rho/2$. Then the average of the a_i's is bounded by $\gamma \cdot 1 + (1-\gamma)\rho/2$. Hence, $\rho \leq \gamma + \rho/2$, implying $\gamma \geq \rho/2$. ■

More generally, we have

Lemma A.9 *If $X \in [0,1]$ and $\mathsf{E}[X] = \mu$ then for any $c < 1$ we have*

$$\Pr[X \leq c\mu] \leq \frac{1-\mu}{1-c\mu}.$$

Example A.10
Suppose you took a lot of exams, each scored from 1 to 100. If your average score was 90 then in at least half the exams you scored at least 80.

A.2.3 Conditional probability and independence

If we already know that an event B happened, this reduces the space from Ω to $\Omega \cap B$, where we need to scale the probabilities by $1/\Pr[B]$ so they will sum up to one. Thus, the probability of an event A *conditioned* on an event B, denoted $\Pr[A|B]$, is equal to $\Pr[A \cap B]/\Pr[B]$ (where we always assume that B has positive probability).

We say that two events A, B are *independent* if $\Pr[A \cap B] = \Pr[A]\Pr[B]$. Note that this implies that $\Pr[A|B] = \Pr[A]$ and $\Pr[B|A] = \Pr[B]$. We say that a set of events $A_1, \ldots, A_n$ are *mutually independent* if for every subset $S \subseteq [n]$,

$$\Pr[\cap_{i \in S} A_i] = \prod_{i \in S} \Pr[A_i]\,. \tag{2}$$

We say that $A_1, \ldots, A_n$ are *k-wise independent* if (2) holds for every $S \subseteq [n]$ with $|S| \leq k$.

We say that two random variables X, Y are *independent* if for every $x, y \in \mathbb{R}$, the events $\{X = x\}$ and $\{Y = y\}$ are independent. We generalize similarly the definition of mutual independence and k-wise independence to sets of random variables $X_1, \ldots, X_n$. We have the following claim:

Claim A.11 *If $X_1, \ldots, X_n$ are mutually independent then*

$$\mathsf{E}[X_1 \cdots X_n] = \prod_{i=1}^{n} \mathsf{E}[X_i]$$

♢

PROOF:

$$\mathsf{E}[X_1 \cdots X_n] = \sum_x x \Pr[X_1 \cdots X_n = x] =$$

$$\sum_{x_1,\ldots,x_n} x_1 \cdots x_n \Pr[X_1 = x_1 \text{ and } X_2 = x_2 \cdots \text{ and } X_n = x_n] = \text{(by independence)}$$

$$\sum_{x_1,\ldots,x_n} x_1 \cdots x_n \Pr[X_1 = x_1] \cdots \Pr[X_n = x_n] =$$

$$(\sum_{x_1} x_1 \Pr[X_1 = x_1])(\sum_{x_2} x_2 \Pr[X_2 = x_2]) \cdots (\sum_{x_n} x_n \Pr[X_n = x_n]) = \prod_{i=1}^{n} \mathsf{E}[X_i]$$

where the sums above are over all the possible real numbers that can be obtained by applying the random variables or their products to the finite set Ω. ■

A.2.4 Deviation upper bounds

Under various conditions, one can give better upper bounds on the probability of a random variable "straying too far" from its expectation. These upper bounds are usually derived by clever use of Markov's inequality.

The *variance* of a random variable X is defined to be $\mathsf{Var}[X] = \mathsf{E}[(X - \mathsf{E}(X))^2]$. Note that since it is the expectation of a non-negative random variable, $\mathsf{Var}[X]$ is always non-negative. Also, using linearity of expectation, we can derive that $\mathsf{Var}[X] = \mathsf{E}[X^2] - (\mathsf{E}[X])^2$. The *standard deviation* of a variable X is defined to be $\sqrt{\mathsf{Var}[X]}$.

The first bound is Chebyshev's inequality, useful when only the variance is known.

Lemma A.12 *(Chebyshev inequality) If X is a random variable with standard deviation σ, then for every $k > 0$,*

$$\Pr[|X - \mathsf{E}[X]| > k\sigma] \leq 1/k^2 \qquad \diamondsuit$$

PROOF: Apply Markov's inequality to the random variable $(X - \mathsf{E}[X])^2$, noting that by definition of variance, $E[(X - \mathsf{E}[X])^2] = \sigma^2$. ■

Chebyshev's inequality is often useful in the case that X is equal to $\sum_{i=1}^{n} X_i$ for pairwise independent random variables $X_1, \ldots, X_n$. This is because of the following claim, that is left as an exercise:

Claim A.13 *If $X_1, \ldots, X_n$ are pairwise independent then*

$$\mathsf{Var}(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} \mathsf{Var}(X_i) \qquad \diamondsuit$$

The next inequality has many names, and is widely known in theoretical computer science as the *Chernoff bound* (see also Note 7.11. It considers scenarios of the following type. Suppose we toss a fair coin n times. The expected number of heads is $n/2$. How tightly is this number concentrated? Should we be very surprised if after 1000 tosses we have 625 heads? The bound we present is slightly more general, since it concerns n different coin tosses of possibly different expectations (the expectation of a coin is the probability of obtaining "heads"; for a fair coin this is 1/2). These are sometimes known as Poisson trials.

Theorem A.14 *("Chernoff" bounds) Let $X_1, X_2, \ldots, X_n$ be mutually independent random variables over $\{0, 1\}$ (i.e., X_i can be either 0 or 1) and let $\mu = \sum_{i=1}^{n} \mathsf{E}[X_i]$. Then for every $\delta > 0$,*

$$\Pr[\sum_{i=1}^{n} X_i \geq (1+\delta)\mu] \leq \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}. \tag{3}$$

$$\Pr[\sum_{i=1}^{n} \leq (1-\delta)\mu] \leq \left[\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right]^{\mu}. \tag{4}$$

$\diamondsuit$

Often, we will only use the following corollary:

Corollary A.15 *Under the above conditions, for every $c > 0$*

$$\Pr\left[\left|\sum_{i=1}^{n} X_i - \mu\right| \geq c\mu\right] \leq 2 \cdot e^{-\min\{c^2/4, c/2\}\mu}.$$

In particular this probability is bounded by $2^{-\Omega(\mu)}$ (where the constant in the Ω notation depends on c). $\diamondsuit$

PROOF: Surprisingly, the Chernoff bound is also proved using the Markov inequality. We only prove the first inequality; the second inequality can be proved similarly. We introduce a positive dummy variable t, and observe that

$$\mathsf{E}[\exp(tX)] = \mathsf{E}[\exp(t\sum_i X_i)] = \mathsf{E}[\prod_i \exp(tX_i)] = \prod_i \mathsf{E}[\exp(tX_i)], \tag{5}$$

where $\exp(z)$ denotes e^z and the last equality holds because the X_i r.v.s are independent. Now,

$$\mathsf{E}[\exp(tX_i)] = (1-p_i) + p_i e^t,$$

therefore,

$$\begin{aligned}\prod_i \mathsf{E}[\exp(tX_i)] = \prod_i [1+p_i(e^t-1)] &\leq \prod_i \exp(p_i(e^t-1)) \\ &= \exp(\sum_i p_i(e^t-1)) = \exp(\mu(e^t-1)),\end{aligned} \tag{6}$$

as $1+x \leq e^x$. Finally, apply Markov's inequality to the random variable $\exp(tX)$, viz.

$$\Pr[X \geq (1+\delta)\mu] = \Pr[\exp(tX) \geq \exp(t(1+\delta)\mu)] \leq \frac{\mathsf{E}[\exp(tX)]}{\exp(t(1+\delta)\mu)} = \frac{\exp((e^t-1)\mu)}{\exp(t(1+\delta)\mu)},$$

using (5), (6) and the fact that t is positive. Since t is a dummy variable, we can choose any positive value we like for it. Simple calculus shows that the right hand side is minimized for $t = \ln(1+\delta)$ and this leads to the theorem statement. ■

So, if all n coin tosses are fair (Heads has probability 1/2) then the the probability of seeing N heads where $|N - n/2| > a\sqrt{n}$ is at most $2e^{-a^2/4}$. In particular, the chance of seeing at least 625 heads in 1000 tosses of an unbiased coin is less than 5.3×10^{-7}.

A.2.5 Some other inequalities.

Jensen's inequality.

The following inequality, generalizing the inequality $\mathsf{E}[X^2] \geq \mathsf{E}[X]^2$, is also often useful:

Lemma A.16 *(Jensen's Inequality) A function $f : \mathbb{R} \to \mathbb{R}$ is* convex *if for every $p \in [0,1]$ and $x, y \in \mathbb{R}$, $f(px + (1-p)y) \leq p \cdot f(x) + (1-p) \cdot f(y)$. For every random variable X and convex function f, $f(\mathsf{E}[X]) \leq \mathsf{E}[f(X)]$.* ♢

Approximating the binomial coefficient

Of special interest is the *Binomial* random variable B_n denoting the number of coins that come up "heads" when tossing n fair coins. For every k, $\Pr[B_n = k] = 2^{-n}\binom{n}{k}$ where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ denotes the number of size-k subsets of $[n]$. Clearly, $\binom{n}{k} \leq n^k$, but sometimes we will need a better estimate for $\binom{n}{k}$ and use the following approximation:

Claim A.17 *For every n, $k < n$, $\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$* ♢

The best approximation can be obtained via Stirling's formula:

Lemma A.18 *(Stirling's formula) For every n,*

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n+1}} < n! < \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$$ ♢

It can be proven by taking natural logarithms and approximating $\ln n! = \ln(1 \cdot 2 \cdots n) = \sum_{i=1}^n \ln i$ by the integral $\int_1^n \ln x \, dx = n \ln n - n + 1$. It implies the following corollary:

Corollary A.19 *For every $n \in \mathbb{N}$ and $\alpha \in [0,1]$,*

$$\binom{n}{\alpha n} = (1 \pm O(n^{-1}))\frac{1}{\sqrt{2\pi n\alpha(1-\alpha)}} 2^{H(\alpha)n}$$

where $H(\alpha) = \alpha \log(1/\alpha) + (1-\alpha)\log(1/(1-\alpha))$ and the constants hidden in the O notation are independent of both n and α. ♢

More useful estimates.

The following inequalities can be obtained via elementary calculus:

- For every $x \geq 1$, $\left(1-\frac{1}{x}\right)^x \leq \frac{1}{e} \leq \left(1-\frac{1}{x+1}\right)^x$
- For every k, $\sum_{i=1}^{n} i^k = \Theta\left(\frac{n^{k+1}}{k+1}\right)$
- For every $k > 1$, $\sum_{i=1}^{\infty} n^{-k} < O(1)$.
- For every $c, \epsilon > 0$, $\sum_{i=1}^{\infty} \frac{n^c}{(1+\epsilon)^n} < O(1)$.
- For every n, $\sum_{i=1}^{n} \frac{1}{n} = \ln n \pm O(1)$

A.2.6 Statistical distance

The following notion of when two distributions are close to one another is often very useful.

Definition A.20 *(Statistical Distance)* Let Ω be some finite set. For two random variables X and Y with range Ω, their *statistical distance* (also known as *variation distance*) is defined as $\Delta(X,Y) = \max_{S \subseteq \Omega}\{|\Pr[X \in S] - \Pr[Y \in S]|\}$. $\diamondsuit$

Some texts use the name *total variation distance* for the statistical distance. The next lemma gives some useful properties of this distance:

Lemma A.21 *Let X, Y, Z be any three distributions taking values in the finite set Ω. Then,*

1. *$\Delta(X,Y) \in [0,1]$ where $\Delta(X) = \Delta(Y)$ iff X is identical to Y.*
2. *(Triangle inequality) $\Delta(X,Z) \leq \Delta(X,Y) + \Delta(Y,Z)$.*
3. *$\Delta(X,Y) = \frac{1}{2}\sum_{x \in \Omega} |\Pr[X=x] - \Pr[Y=x]|$.*
4. *$\Delta(X,Y) \geq \epsilon$ iff there is a Boolean function $f : \Omega \to \{0,1\}$ such that $|\mathsf{E}[f(X)] - E[f(Y)]| \geq \epsilon$.*
5. *For every finite set Ω' and function $f : \Omega \to \Omega'$, $\Delta(f(X), f(Y)) \leq \Delta(X,Y)$. (Here $f(X)$ is a distribution on Ω' obtained by taking a sample of X and applying f.)*

Note that Item 3 means that $\Delta(X,Y)$ is equal to the L_1-distance of X and Y divided by 2 (see Section A.5.4 below). That is, if we think of X as a vector in $\mathbb{R}^{\Omega}$ where $X_\omega = \Pr[X=\omega]$, and define for every vector $\mathbf{v} \in \mathbb{R}^{\Omega}$, $|\mathbf{v}|_1 = \sum_{\omega \in \Omega} |\mathbf{v}_\omega|$, then $\Delta(X,Y) = 1/2|X-Y|_1$.

Proof of Lemma A.21: We start with Item 3. For every pairs of distributions X, Y over $\{0,1\}^n$ let S be the set of strings x such that $\Pr[X=x] > \Pr[X=y]$. Then it is easy to see that this choice of S maximizes the quantity $b(S) = \Pr[X \in S] - \Pr[Y \in S]$ and in fact $b(S) = \Delta(X,Y)$ since if we had a set T with $b(T) < -b(S)$ then the complement $\overline{T}$ of T would satisfy $b(T) > b(S)$. But,

$$\sum_{x \in \{0,1\}^n} |\Pr[X=x] - \Pr[Y=x]| =$$

$$\sum_{x \in S} \Pr[X=x] - \Pr[Y=x] + \sum_{x \notin S} \Pr[Y=x] - \Pr[X=x] =$$

$$\Pr[X \in S] - \Pr[Y \in S] + (1 - \Pr[Y \in S]) - (1 - \Pr[X \in S]) =$$

$$2\Pr[X \in S] - 2\Pr[Y \in S],$$

establishing Item 3.

The triangle inequality (Item 2) follows immediately from Item 3 since $\Delta(X,Y) = 1/2|X-Y|_1$ and the L_1 norm satisfies the triangle inequality. Item 3 also implies Item 1 since $|X-Y|_1 = 0$ iff $X = Y$ and $|X-Y|_1 \leq \|X\| + |Y|_1 = 1+1$.

Item 4 is just a rephrasing of the definition of statistical distance, identifying a set $S \subseteq \{0,1\}^n$ with the function $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = 1$ iff $x \in S$. Item 5 follows from Item 4 noting that if $\Delta(X,Y) \leq \epsilon$ then $|\mathsf{E}[g(f(X))] - \mathsf{E}[g(f(Y))]| \leq \epsilon$ for every function g. ■

A.3 Number theory and groups

The *integers* are the set $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$ while the natural numbers are the subset $\mathbb{N} = \{0, 1, 2, \ldots\}$.[1] A basic fact is that we can *divide* any integer n by an nonzero integer k to obtain ℓ, r such that $n = k\ell + r$ and $r \in \{0, \ldots, n-1\}$. If $r = 0$ then we say that k *divides* n and denote this by $k|n$. The *factors* of n are the set of positive integers that divide n.

The *greatest common divisor* of two integers n, m, denoted by $gcd(n,m)$ is the largest integers d such that $d|n$ and $d|m$. We say that n and m are *co-prime* if their greatest common divisor is equal to 1. The following basic facts are not hard to verify:

- If a nonzero integer c divides both n and m then $c|d$.
- The greatest common divisor of n and m is the smallest positive integer d such that there exist integers x, y satisfying $nx + my = d$.
- There is a polynomial-time (i.e., polylog(n,m)-time) algorithm that on input n, m outputs the greatest common divisor d of n, m and the integers x, y satisfying $nx + my = d$. (This algorithm is known as *Euclid's Algorithm*.)

A number $p > 1$ is *prime* if its only factors are 1 and p. The following basic facts are known about prime numbers:

- Every positive integer n can be written uniquely (up to ordering) as a product of prime numbers. This is called the *prime factorization* of n.
- If $gcd(p,a) = 1$ and $p|ab$ then $p|b$. In particular, if a prime p divides $a \cdot b$ then either $p|a$ or $p|b$.

A fundamental question in number theory is how many primes exist. A celebrated result is:

Theorem A.22 *(The Prime Number Theorem (Hadamard, de la Vallée Poussin 1896)) For $n > 1$, let $\pi(n)$ denote the number of primes between 1 and n then*

$$\pi(n) = \frac{n}{\ln n}(1 \pm o(1)) \qquad \diamondsuit$$

The original proofs of the prime number theorem used rather deep mathematical tools, and in fact people have conjectured that this is *inherently* the case. But in 1949 both Erdös and Selberg (independently) found elementary proofs for this theorem. For most computer science applications, the following weaker statement proven by Chebychev suffices:

Theorem A.23 $\pi(n) = \Theta(\frac{n}{\log n})$ $\diamondsuit$

PROOF: Consider the number $\binom{2n}{n} = \frac{2n!}{n!n!}$. By Stirling's formula we know that $\log \binom{2n}{n} = (1 - o(1))2n$ and in particular $n \leq \log \binom{2n}{n} \leq 2n$. Also, all the prime factors of $\binom{2n}{n}$ are

[1] Some texts exclude 0 in $\mathbb{N}$; in most cases this does not any difference.

between 0 and $2n$, and each factor p cannot appear more than $k = \left\lfloor \frac{\log 2n}{\log p} \right\rfloor$ times. Indeed, for every n, the number of times p appears in the factorization of $n!$ is $\sum_i \left\lfloor \frac{n}{p^i} \right\rfloor$, since we get $\left\lfloor \frac{n}{p} \right\rfloor$ times a factor p in the factorizations of $\{1, \ldots, n\}$, $\left\lfloor \frac{n}{p^2} \right\rfloor$ times a factor of the form p^2, etc... Thus the number of times p appears in the factorization of $\binom{2n}{n} = \frac{(2n)!}{n!n!}$ is equal to $\sum_i \left\lfloor \frac{2n}{p^i} \right\rfloor - 2 \left\lfloor \frac{n}{p^i} \right\rfloor$: a sum of at most k elements (since $p^{k+1} > 2n$) each of which is either 0 or 1.

Thus, $\binom{2n}{n} \leq \prod_{\substack{1 \leq p \leq 2n \\ p \text{ prime}}} p^{\left\lfloor \frac{\log 2n}{\log p} \right\rfloor}$. Taking logs we get that

$$n \leq \log \binom{2n}{n} \leq \sum_{\substack{1 \leq p \leq 2n \\ p \text{ prime}}} \left\lfloor \frac{\log 2n}{\log p} \right\rfloor \log p \leq \sum_{\substack{1 \leq p \leq 2n \\ p \text{ prime}}} \log 2n = \pi(2n) \log 2n \,,$$

establishing $\pi(n) = \Omega(\frac{n}{\log n})$.

To prove that $\pi(n) = O(\frac{n}{\log n})$, we define the function $\vartheta(n) = \sum_{\substack{1 \leq p \leq n \\ p \text{ prime}}} \log p$. It suffices to prove that $\vartheta(n) = O(n)$ (exercise!). But since all the primes between $n+1$ and $2n$ divide $\binom{2n}{n}$ at least once, $\binom{2n}{n} \geq \prod_{\substack{n+1 \leq p \leq 2n \\ p \text{ prime}}} p$. Taking logs we get

$$2n \geq \log \binom{2n}{n} \geq \sum_{\substack{n+1 \leq p \leq 2n \\ p \text{ prime}}} \log p = \vartheta(2n) - \vartheta(n) \,,$$

thus getting a recursive equation $\vartheta(2n) \leq \vartheta(n) + 2n$ which solves to $\vartheta(n) = O(n)$. ■

A.3.1 Groups.

A *group* is an abstraction that captures some properties of mathematical objects such as the integers, matrices, functions and more. Formally, a group is a set that has a binary operation, say $\star$, defined on it that is associative and has an inverse. That is, $(G, \star)$ is a group if

1. For every $a, b, c \in G$, $(a \star b) \star c = a \star (b \star c)$
2. There exists a special element $id \in G$ such that $a \star id = a$ for every $a \in G$, and for every $a \in G$ there exists $b \in G$ such that $a \star b = b \star a = id$. (This element b is called the *inverse* of a, and is often denote as a^{-1} or $-a$.)

Examples for groups are the integers, with addition being the group operation (and zero the identity element), the non-zero real numbers with multiplication being the group operation(and one the identity element), and the set of functions from a domain A to itself, with function composition being the group operation.

Often, it is natural to use additive ($+$) or multiplicative ($\cdot$) notation to denote the group operation rather than $\star$. In these cases we will use ℓa (or respectively a^ℓ) to denote the result of applying the operation to a ℓ times.

A.3.2 Finite groups

A group is *finite* if it has a finite number of elements. We denote by $|G|$ the number of elements of G. Examples for finite groups are the following:

- The group $\mathbb{Z}_n$ of the integers from 0 to $n-1$ with the operation being addition modulo n. In particular $\mathbb{Z}_2$ is the set $\{0, 1\}$ with the XOR operation.

- The group S_n of the permutations on $[n]$, with the operation being function composition.
- The group $(\mathbb{Z}_2)^n$ of n-bit strings with the operation being bitwise XOR. More generally for every two groups G and H, we can define the group $G \times H$ to be a group whose elements are pairs $\langle g, h \rangle$ with $g \in G$ and $h \in H$ and with the group operation corresponding to applying the group operations of G and H componentwise. Similarly, we define G^n to be the group $G \times G \times \cdots \times G$ (n times).
- For every n, the group $\mathbb{Z}_n^*$ consists of the set $\{k : 1 \leq k \leq n-1 \ , \ gcd(k,n) = 1\}$ and the operation of multiplication modulo n. Note that if $gcd(k,n) = 1$ then there exist x, y such that $kx + ny = 1$ or in other words $kx = 1 \pmod{n}$, meaning that x is the inverse of k modulo n. This also means that we can find this inverse in polynomial time using Euclid's algorithm. The size of $\mathbb{Z}_n^*$ is denoted by $\varphi(n)$ and the function φ is known as *Euler's Quotient function*. Note that if n is prime then $\varphi(n) = n - 1$. It is known that for every $n > 6$, $\varphi(n) \geq \sqrt{n}$.

A *subgroup* of G is a subset of G that is itself a group (i.e., closed under the group operation and taking inverses). The following result is often quite useful

Theorem A.24 *If G is a finite group and H is a subgroup of G then $|H|$ divides $|G|$.* ♢

PROOF: Consider the family of sets of the form $aH = \{ah : h \in H\}$ for all $a \in G$ (we're using here multiplicative notation for the group). It is easy to see that the map $x \mapsto ax$ is one-to-one and hence $|aH| = |H|$ for every a. Hence it will suffice to show that we can partition G into disjoint sets from this family. Yet this family clearly covers G (as $a \in aH$ for every $a \in G$) and hence it suffices to show that for every a, b either $aH = bH$ or aH and bH are disjoint. Indeed, suppose that there exist $x, y \in H$ such that $ax = by$ then for every element $az \in aH$, we have that $az = (byx^{-1})z$ and since $yx^{-1}z \in H$ we get that $az \in bH$. ■

Corollary A.25 *(Fermat's Little Theorem) For every n and $x \in \{1, \ldots, n-1\}$, $x^{\varphi(n)} = 1 \pmod{n}$. In particular, if n is prime then $x^{n-1} = 1 \pmod{n}$.* ♢

PROOF: Consider the set $H = \left\{x^\ell : \ell \in \mathbb{Z}\right\}$. This is clearly a subgroup of $\mathbb{Z}_n^*$ and hence $|H|$ divides $\varphi(n)$. But the size of H is simply the smallest number k such that $x^k = 1 \pmod{n}$. Indeed, there must be such a number since, because $\mathbb{Z}_n^*$, if we consider the sequence of numbers $1, x, x^2, x^3, \ldots$ then eventually we get i, j such that $x^i = x^j$ for $i < j$, meaning that $x^{i-j} = 1 \pmod{n}$. Thus, the above sequence looks like $1, x, x^2, \ldots, x^{k-1}, 1, x, x^2, \ldots$, meaning that $|H| = k$.

Since $x^{|H|} = 1 \pmod{n}$, obviously taking x to the power $\varphi(n)$ (which is a multiple of $|H|$) yields also 1 modulo n. ■

The *order* of an element x of a group G is the smallest integer k such that x^k is equal to the identity element. The proof above shows that in a finite group G, every element has a finite order and furthermore this order divides the size of G. An element x of G with order $|G|$ is called a *generator* of G, since in this case the subgroup $\left\{x, x^1, x^2, \ldots\right\}$ is all of G.[2] If a group G has a generator then we say that G is *cyclic*. An example for a simple cyclic group is the group $\mathbb{Z}_n$ of the numbers $\{0, \ldots, n-1\}$ with addition modulo n, that is generated by the element 1 (and also by any other element that is co-prime to n— exercise).

A.3.3 The Chinese Remainder Theorem

Let $n = pq$ where p, q are co-prime. The Chinese Remainder Theorem (CRT) says that the group $\mathbb{Z}_n^*$ (multiplicative group modulo n) is isomorphic to the group $Z_p^* \times \mathbb{Z}_q^*$ (pairs of numbers with multiplication done componentwise modulo p and q respectively).

[2] A more general definition (that works also for infinite groups) is that x is a generator of G if the subgroup $\left\{x^\ell : \ell \in \mathbb{Z}\right\}$ is equal to G.

Theorem A.26 *If $n = pq$ where p, q coprime then function f that maps x to $\langle x \pmod p, x \pmod q\rangle$ is one-to-one on $\mathbb{Z}_n^*$. Furthermore f is an isomorphism in the sense that $f(xy) = f(x)f(y)$ (where multiplication on the left hand side is modulo n and on the right hand side is componentwise modulo p and q respectively).* ♢

PROOF: The furthermore part can be easily verified and so we focus on showing that f is one-to-one. We need to show that if $f(x) = f(x')$ then $x = x'$. Since $f(x-x') = f(x)-f(x')$, it suffices to show that if $x = 0 \pmod p$ (i.e., $p|x$) and $x = 0 \pmod q$ (i.e., $q|x$) then $x = 0 \pmod n$ (i.e., $pq|x$). Yet, assume that $p|x$ and write $x = pk$. Then since $gcd(p, q) = 1$ and $q|x$ we know that $q|k$, meaning that $pq|x$. ■

The Chinese Remainder Theorem can be easily generalized to show that for every $n = p_1 p_2 \ldots p_k$, where all the p_i's are co-prime, there is an isomorphism between Z_n^* to $\mathbb{Z}_{p_1}^* \times \cdots \times \mathbb{Z}_{p_k}^*$, meaning that for every n, the group $\mathbb{Z}_n^*$ is isomorphic to a product of groups of the form $\mathbb{Z}_q^*$ for q a prime power (i.e., number of the form p^ℓ for prime p). In fact, it can be generalized even further to show that every Abelian group G is isomorphic to a product $G_1 \times G_2 \times \cdots \times G_k$ where all the G_i's are cyclic. (This can be viewed as a generalization of the CRT because all the groups of the form $\mathbb{Z}_q^*$ for q a power of an odd prime are cyclic, and all groups of the form $\mathbb{Z}_{2^k}^*$ are either cyclic or products of two cyclic groups.)

A.4 Finite fields

A *field* is a set $\mathbb{F}$ that has an addition ($+$) and multiplication ($\cdot$) operations that behave in the expected way: satisfy associative, commutative and distributive laws, have both additive and multiplicative inverses, and neutral elements 0 and 1 for addition and multiplication respectively. In other words, $\mathbb{F}$ is a field if it is an Abelian group with the operation $+$ and an identity element 0, and has an additional operation $\cdot$ such that $\mathbb{F}\setminus\{0\}$ and $\cdot$ forms an Abelian group, and furthermore the two operation satisfy the distributive rule $a(b + c) = ab + ac$.

Familiar fields are the real numbers ($\mathbb{R}$), the rational numbers ($\mathbb{Q}$) and the complex numbers ($\mathbb{C}$), but there are also *finite* fields. Recall that for a prime p, the set $\{0, \ldots, p-1\}$ is an Abelian group with the addition modulo p operation and the set $\{1, \ldots, p-1\}$ is an Abelian group with the multiplication modulo p operation. Hence $\{0, \ldots, p-1\}$ form a field with these two operations, which we denote by $\mathrm{GF}(p)$. The simplest example for such a field is the field $\mathrm{GF}(2)$ consisting of $\{0, 1\}$ where multiplication is the AND ($\wedge$) operation and addition is the XOR operation.

Every finite field $\mathbb{F}$ has a number ℓ such that for every $x \in F$, $x + x + \cdots + x$ (ℓ times) is equal to the zero element of $\mathbb{F}$ (exercise). This number ℓ is called the *characteristic* of $\mathbb{F}$. For every prime q, the characteristic of $\mathrm{GF}(q)$ is equal to q.

A.4.1 Non-prime fields.

One can see that if n is not prime, then the set $\{0, \ldots, n-1\}$ with addition and multiplication modulo n is not a field, as there exist two non-zero elements x, y in this set such that $x \cdot y = n = 0 \pmod n$. Nevertheless, there are finite fields of size n for non-prime n. Specifically, for every prime q, and $k \geq 1$, there exists a field of q^k elements, which we denote by $\mathrm{GF}(q^k)$. We will very rarely need to use such fields in this book, but still provide an outline of their construction below.

For every prime q and k there exists an *irreducible* degree k polynomial P over the field $\mathrm{GF}(q)$ (P is irreducible if it cannot be expressed as the product of two polynomials P', P'' of lower degree). We then let $\mathrm{GF}(q^k)$ be the set of all $k-1$-degree polynomials over $\mathrm{GF}(q)$. Each such polynomial can be represented as a vector of its k coefficients. We perform both addition and multiplication modulo the polynomial P. Note that addition corresponds to standard vector addition of k-dimensional vectors over $\mathrm{GF}(q)$, and both addition and multiplication can be easily done in $\mathrm{poly}(n, \log q)$ time (we can reduce a polynomial S

modulo a polynomial P using a similar algorithm to long division of numbers). It turns out that no matter how we choose the irreducible polynomial P, we will get the same field, up to renaming of the elements. There is a deterministic $\text{poly}(q,k)$-time algorithm to obtain an irreducible polynomial of degree k over $\text{GF}(q)$. There are also probabilistic algorithms (and deterministic algorithms whose analysis relies on unproven assumptions) that obtain such a polynomial in $\text{poly}(\log q, k)$ time (see the book [Sho05]).

For us, the most important example of a finite field is $\text{GF}(2^k)$, which consists of the set $\{0,1\}^k$, with addition being component-wise XOR, and multiplication being polynomial multiplication via some irreducible polynomial which we can find in $\text{poly}(k)$ time. In fact, we will mostly not even be interested in the multiplicative structure of $\text{GF}(2^k)$ and only use the addition operation (i.e., use it as the vector space $\text{GF}(2)^k$, see below).

A.5 Basic facts from linear algebra

For $\mathbb{F}$ a field and $n \in \mathbb{N}$, we denote by $\mathbb{F}^n$ the set of n-length tuples (or *vectors*) of elements of $\mathbb{F}$. If $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$ and $x \in \mathbb{F}$ then we denote by $\mathbf{u}+\mathbf{v}$ the vector obtained by componentwise addition of $\mathbf{u}$ and $\mathbf{v}$ and by $x\mathbf{u}$ the vector obtained by multiplying each entry of $\mathbf{u}$ by x.

A set of vectors $\mathbf{u}^1, \ldots, \mathbf{u}^k$ in $\mathbb{F}^n$ is *linearly independent* if the only solution to the equation $x_1\mathbf{u}^1 + \cdots + x_k\mathbf{u}^k = \mathbf{0}$ (where $\mathbf{0}$ denotes the all-zero vector) is $x_1 = x_2 = \cdots = x_k = 0$. It can be shown that if $\mathbf{u}^1, \ldots, \mathbf{u}^k$ are linearly independent then $k \leq n$ (exercise). A set of n linearly independent vectors in $\mathbb{F}^n$ is called a *basis* of $\mathbb{F}^n$. It is not hard to see that if $\mathbf{u}^1, \ldots, \mathbf{u}^n$ is a basis of $\mathbb{F}^n$ then every vector $\mathbf{v} \in \mathbb{F}^n$ can be expressed as a linear combination $\mathbf{v} = \sum_i x_i \mathbf{u}^i$ of the vectors $\mathbf{u}^1, \ldots, \mathbf{u}^n$ and furthermore this expression is unique. The *standard basis* of $\mathbb{F}^n$ is the set $\mathbf{e}^1, \ldots, \mathbf{e}^n$, where $\mathbf{e}^i_j$ is equal to 1 if $j = i$ and to 0 otherwise.

A subset $S \subseteq \mathbb{F}^n$ is called a *subspace* if it is closed under addition and scalar multiplication (i.e., $\mathbf{u}, \mathbf{v} \in S$ and $x, y \in \mathbb{F}$ implies that $x\mathbf{u} + y\mathbf{v} \in S$). The *dimension* of S, denoted by $\dim(S)$ is defined to be the maximum number k such that there are k linearly independent vectors in S. Such a set of $\dim(S)$ linearly independent vectors in S is called a *basis* and one can see that every vector in S can be expressed as a linear combination of the vectors in the basis.

A function $f : \mathbb{F}^n \to \mathbb{F}^m$ is *linear* if $f(\mathbf{u}+\mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$. It's not hard to verify that the following hold for every linear function f:

- If $\mathbf{u}^1, \ldots, \mathbf{u}^n$ is a basis for $\mathbb{F}^n$ then for every $\mathbf{v} \in \mathbb{F}^n$, $f(\mathbf{v}) = \sum_i x_i f(\mathbf{u}_i)$ where $x_1, \ldots, x_n$ are the elements such that $\mathbf{v} = \sum x_i \mathbf{u}^i$. Thus, to know f's value at every point it suffices to know its value on the basis elements.
- The set $Im(f) = \{f(\mathbf{v}) : \mathbf{v} \in \mathbb{F}^n\}$ is a subspace of $\mathbb{F}^m$.
- The set $Ker(f) = \{\mathbf{v} : f(\mathbf{v}) = 0\}$ is a subspace of $\mathbb{F}^n$.
- $\dim(Im(f)) + \dim(Ker(f)) = n$

A linear function $f : \mathbb{F}^n \to \mathbb{F}^m$ is often described by an $m \times n$ matrix A whose i^{th} column is $f(\mathbf{e}^i)$. The *multiplication* of an $m \times n$ matrix A and an $n \times k$ matrix B is the $n \times k$ matrix $C = AB$ where $C_{i,j} = \sum_{\ell \in [n]} A_{i,\ell} B_{\ell,j}$. One can verify that if A describes a function $f : \mathbb{F}^n \to \mathbb{F}^m$ and B describes a function $g : \mathbb{F}^k \to \mathbb{F}^n$ then C describes the function $h : \mathbb{F}^k \to \mathbb{F}^m$ mapping $\mathbf{v}$ to $f(g(\mathbf{v}))$. It can also be verified that if we identify members of $\mathbb{F}^n$ with $n \times 1$ matrices (i.e., column vectors) then $f(\mathbf{v}) = A\mathbf{v}$.

The *determinant* of an $n \times n$ matrix A, denoted by $\det(A)$ is equal to $\sum_{\sigma \in S_n} (-1)^{sgn(\sigma)} \prod_{i=1}^n A_{i,\sigma(i)}$ where S_n denotes the group of permutations over $[n]$ and $sgn(\sigma)$ is equal to 1 if the number of pairs $\langle i, j \rangle$ such that $i < j$ but $\sigma(i) > \sigma(j)$ is odd, and is equal to 0 otherwise. We have the following two facts:

- $\det(AB) = \det(A)\det(B)$. This can be verified by direct computation.

- If A is an upper triangular matrix (i.e., $A_{i,j} = 0$ whenever $i > j$) then $\det(A) = A_{1,1}A_{2,2}\cdots A_{n,n}$. Indeed, for a permutation σ to give a non-zero contribution to the determinant in this case it must satisfy $\sigma(i) \geq i$ for every i, which means that it is the identity permutation.

Together these two rules give a polynomial-time algorithm to compute the determinant of a matrix A by following the well known Gaussian elimination algorithm to express A as $E_1E_2\cdots E_mD$ where the E_i's are elementary matrices (multiplication by which corresponds to switching two columns, multiplying a column by a field element, or adding one column to another) and the D is upper diagonal. Since the determinant is easy to compute for all these matrices, we can compute the determinant of A as well.

The following lemma relates the determinant of a matrix to the function it represents:

Lemma A.27 *For a function $f : \mathbb{F}^n \to \mathbb{F}^n$ represented by an $n \times n$ matrix A, the following conditions are equivalent:*

- *The columns of A are a basis for $\mathbb{F}^n$.*
- *f is one-to-one.*
- $\dim(Im(f)) = n$.
- $\dim(Ker(f)) = 0$.
- $\det(A) \neq 0$.
- *There exists $\mathbf{v} \in \mathbb{F}^n$ such that the equation $A\mathbf{x} = \mathbf{v}$ has exactly one solution.*
- *For every $\mathbf{v} \in \mathbb{F}^n$, the equation $A\mathbf{x} = \mathbf{v}$ has exactly one solution.*

Furthermore, if f is one-to-one then the mapping f^{-1} is linear and is represented by an $n\times n$ matrix A^{-1} whose $(i,j)^{th}$ entry is $\frac{\det(A_{-(i,j)})}{\det(A)}$, where $A_{-(i,j)}$ denotes the $(n-1)\times(n-1)$ matrix obtained by removing the i^{th} row and j^{th} column from A. ♢

A.5.1 Inner product

The vector spaces $\mathbb{R}^n$ and $\mathbb{C}^n$ have an additional structure that is often quite useful.[3] An *inner product* over $\mathbb{C}^n$ to be a function mapping two vectors $\mathbf{u}, \mathbf{v}$ to a complex number $\langle \mathbf{u}, \mathbf{v} \rangle$ satisfying the following conditions:

- $\langle x\mathbf{u} + y\mathbf{w}, \mathbf{v} \rangle = x\langle \mathbf{u}, \mathbf{v} \rangle + y\langle \mathbf{w}, \mathbf{v} \rangle$
- $\langle \mathbf{v}, \mathbf{u} \rangle = \overline{\langle \mathbf{u}, \mathbf{v} \rangle}$ where $\overline{z}$ denotes complex conjugation (i.e., if $z = a+ib$ then $\overline{z} = a-ib$).
- For every $\mathbf{u}$, $\langle \mathbf{u}, \mathbf{u} \rangle$ is a non-negative real number with $\langle \mathbf{u}, \mathbf{u} \rangle = 0$ iff $\mathbf{u} = \mathbf{0}$.

The two examples for inner products we will use are the standard inner product mapping $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$ to $\sum_{i=1}^n \mathbf{x}_i\overline{\mathbf{y}}_i$ and the expectation or normalized inner product mapping $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$ to $\frac{1}{n}\sum_{i=1}^n \mathbf{x}_i\overline{\mathbf{y}}_i$. We can also define inner products over the space $\mathbb{R}^n$, in which case we drop the conjugation.

If $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ we say that $\mathbf{u}$ and $\mathbf{v}$ are *orthogonal* and denote this by $\mathbf{u} \perp \mathbf{v}$. We have the following result:

Lemma A.28 *If non-zero vectors $\mathbf{u}^1, \ldots, \mathbf{u}^k$ satisfy $\mathbf{u}^i \perp \mathbf{u}^j$ for all $i \neq j$ then they are linearly independent.* ♢

[3] The reason we restrict ourselves to these fields is that they have characteristic zero which means that there does not exist a number $k \in \mathbb{N}$ and nonzero $a \in \mathbb{F}$ such that $ka = 0$ (where ka is the result of adding a to itself k times). You can check that if there is such a number for a field $\mathbb{F}$ then there will not be an inner product over $\mathbb{F}^n$.

PROOF: Suppose that $\sum_i x_i \mathbf{u}^i = \mathbf{0}$ and consider take an inner product of this vector with itself. We get that

$$0 = \langle \sum_i x_i \mathbf{u}^i, \sum_j x_j \mathbf{u}^j \rangle = \sum_{i,j} x_i \overline{x}_j \langle \mathbf{u}^i, \mathbf{u}^j \rangle = \sum_i |x_i|^2 \langle \mathbf{u}^i, \mathbf{u}^j \rangle \,, \tag{7}$$

where the last equality follows from the fact that $\langle \mathbf{u}^i, \mathbf{u}^j \rangle = 0$ for $i \neq j$. But unless all the x_i's are zero, the righthand side of (7) is strictly positive. (Recall that for a complex number $x = a + ib$, $|x| = \sqrt{a^2 + b^2}$ and $|x|^2 = x\overline{x}$.) ■

A set $\mathbf{u}^1, \ldots, \mathbf{u}^n$ of nonzero vectors in $\mathbb{C}^n$ satisfying $\langle \mathbf{u}^i, \mathbf{u}^j \rangle = 0$ for $i \neq j$ is called an *orthogonal basis* of $\mathbb{C}^n$. If in addition $\langle \mathbf{u}^i, \mathbf{u}^i \rangle = 1$ for all i then we say this is an *orthonormal basis*. An orthonormal basis consists of n linearly independent vectors and hence as its name implies is a basis of $\mathbb{C}^n$, meaning that every vector $\mathbf{v}$ can be expressed as $\mathbf{v} = \sum_i x_i \mathbf{u}^i$. By taking an inner product of this equality with $\mathbf{u}^i$, one can see that $x_i = \langle \mathbf{v}, \mathbf{u}^i \rangle$

The following identity (that can be viewed as a generalization of the Pythagorean theorem) is often useful:

Lemma A.29 *(Parseval's identity) If $\mathbf{u}^1, \ldots, \mathbf{u}^n$ is an orthonormal basis for $\mathbb{C}^n$, then for every $\mathbf{v}$,*

$$\langle \mathbf{v}, \mathbf{v} \rangle = \sum_{i=1}^{n} |x_i|^2 \,,$$

where $x_1, \ldots, x_n$ are the numbers such that $\mathbf{v} = \sum_i x_i \mathbf{u}^i$. ♢

PROOF: As in the proof of Lemma A.28,

$$\langle \mathbf{v}, \mathbf{v} \rangle = \langle \sum_i x_i \mathbf{u}^i, \sum_j x_j \mathbf{u}^j \rangle = \sum_i |x_i|^2 \langle \mathbf{u}^i, \mathbf{u}^i \rangle \,. \; ■$$

Vector spaces with an inner product are known as *Hilbert spaces*.

A.5.2 Dot product

Even in a field $\mathbb{F}$ that doesn't have an inner product, we can define the *dot product* of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$, denoted by $\mathbf{u} \odot \mathbf{v}$, as $\sum_{i=1}^n \mathbf{u}_i \mathbf{v}_i$. For every subspace $S \subseteq \mathbb{F}^n$, we define $S^\perp = \{\mathbf{u} : \mathbf{u} \odot \mathbf{v} = 0 \forall \mathbf{v} \in S\}$. We leave the following simple claim as an exercise:

Claim A.30 $\dim(S) + \dim(S^\perp) = n$ ♢

In particular for every nonzero vector $\mathbf{u} \in \mathbb{F}^n$, the subspace $\mathbf{u}^\perp$ of vectors $\mathbf{v}$ satisfying $\mathbf{u} \odot \mathbf{v} = 0$ has dimension $n - 1$ and hence cardinality $|\mathbb{F}|^{n-1}$. As a corollary we get the following very useful fact:

Claim A.31 *(The random subsum principle) For every nonzero $\mathbf{u} \in \mathrm{GF}(2)$ (the field $\{0, 1\}$ with addition and multiplication modulo 2):*

$$\Pr_{\mathbf{v} \in_R \mathrm{GF}(2)^n} [\mathbf{u} \odot \mathbf{v} = 0] = 1/2$$ ♢

A.5.3 Eigenvectors and eigenvalues

If A is an $n \times n$ complex matrix and $\mathbf{v} \in \mathbb{C}^n$ is a nonzero vector, we say that $\mathbf{v}$ is an *eigenvector* of A if there exists $\lambda \in \mathbb{C}$ such that $A\mathbf{v} = \lambda \mathbf{v}$. We say that A is *diagonalizable* if there is a basis $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of eigenvectors for A. In other words, there is an invertible matrix P such that PAP^{-1} is a diagonal matrix.

Note that A has an eigenvector with eigenvalue λ if and only if the matrix $A - \lambda I$ is non-invertible, where I is the identity matrix. Thus in particular λ is a root of the polynomial $p(x) = \det(A - xI)$. Thus the fundamental Theorem of Algebra (that every complex polynomial has as many roots as the degree) that every square matrix has at least one eigenvector. (A non-invertible matrix has an eigenvector zero.)

For a matrix A, the *conjugate transpose* of A, denoted A^*, is the matrix such that for every i, j, $A^*_{i,j} = \overline{A}_{j,i}$ where $\bar{}$ denotes the complex conjugate operation. We say that an $n \times n$ matrix A is *Hermitian* if $A = A^*$. An Hermitian matrix with only real entries is called *symmetric*. That is, a real matrix is symmetric if $A = A^\dagger$ where $\dagger$ is the transpose operation (i.e., $A^\dagger_{i,j} = A_{j,i}$). An equivalent condition (exercise) is that A is Hermitian if and only if

$$\langle A\mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{u}, A\mathbf{v} \rangle \,. \tag{8}$$

An important useful fact about Hermitian matrices is the following theorem:

Theorem A.32 *If A is an $n \times n$ Hermitian matrix then there exists an orthogonal basis of eigenvectors for A.* ♢

Proof: We prove this by induction on n. We know that A has one eigenvector $\mathbf{v}$ with eigenvalue λ. Now let $S = \mathbf{v}^\perp$ be the $n-1$ dimensional space of all vectors orthogonal to $\mathbf{v}$. We claim that for every $\mathbf{u} \in S$, $A\mathbf{u} \in S$. Indeed, if $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ then

$$\langle A\mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{u}, A\mathbf{v} \rangle = \overline{\lambda} \langle \mathbf{u}, \mathbf{v} \rangle = 0 \,.$$

Thus the restriction of A to S is an $n-1$ dimensional linear operator satisfying (8) and hence by induction this restriction has an orthogonal basis of eigenvectors $\mathbf{v}^2, \ldots, \mathbf{v}^n$. Adding $\mathbf{v}$ to this set we get an n-dimensional orthogonal basis of eigenvectors for A. ■

Note that if A is real and symmetric then all its eigenvalues must be real also (with no imaginary components). Indeed, if $A\mathbf{v} = \lambda \mathbf{v}$ then

$$\lambda \langle \mathbf{v}, \mathbf{v} \rangle = \langle A\mathbf{v}, \mathbf{v} \rangle = \langle \mathbf{v}, A\mathbf{v} \rangle = \overline{\lambda} \langle \mathbf{v}, \mathbf{v} \rangle \,,$$

meaning that for a nonzero $\mathbf{v}$, $\lambda = \overline{\lambda}$. This implies that the eigenvectors, that are obtained by solving a linear equation with real coefficients, are also real.

A.5.4 Norms

A *norm* of a vector in $\mathbb{C}^n$ is a function mapping a vector $\mathbf{v}$ to a real number $\|\mathbf{v}\|$ satisfying:

- For every $\mathbf{v}$, $\|\mathbf{v}\| \geq 0$ with $\|\mathbf{v}\| = 0$ iff $\mathbf{v} = \mathbf{0}$.
- If $x \in \mathbb{C}$ then $\|x\mathbf{v}\| = |x| \|\mathbf{v}\|$.
- (Triangle inequality) For every $\mathbf{u}, \mathbf{v}$, $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$.

For every $\mathbf{v} \in \mathbb{C}^n$ and number $p \geq 1$, the L_p *norm* of $\mathbf{v}$, denoted $\|\mathbf{v}\|_p$, is equal to $(\sum_{i=1}^n |\mathbf{v}_i|^p)^{1/p}$. One particularly interesting case is $p = 2$, the so-called *Euclidean norm*, in which $\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^n |\mathbf{v}_i|^2} = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$. Another interesting case is $p = 1$, where we use the single bar notation and denote $|\mathbf{v}|_1 = \sum_{i=1}^n |\mathbf{v}_i|$. Another case is $p = \infty$, where we denote $\|\mathbf{v}\|_\infty = \lim_{p \to \infty} \|\mathbf{v}\|_p = \max_{i \in [n]} |\mathbf{v}_i|$.

Some relations between the different norms can be derived from the *Hölder inequality*, stating that for every p, q with $\frac{1}{p} + \frac{1}{q} = 1$, $\|\mathbf{u}\|_p \|\mathbf{v}\|_q \geq \sum_{i=1}^n |\mathbf{u}_i \mathbf{v}_i|$. To prove it, note that by simple scaling, it suffices to consider norm one vectors, and so it enough to show that if $\|\mathbf{u}\|_p = \|\mathbf{v}\|_q = 1$ then $\sum_{i=1}^n |\mathbf{u}_i||\mathbf{v}_i| \leq 1$. But $\sum_{i=1}^n |\mathbf{u}_i||\mathbf{v}_i| = \sum_{i=1}^n |\mathbf{u}_i|^{p(1/p)} |\mathbf{v}_i|^{q(1/q)} \leq \sum_{i=1}^n \frac{1}{p}|\mathbf{u}_i|^p + \frac{1}{q}|\mathbf{v}_i|^q = \frac{1}{p} + \frac{1}{q} = 1$, where the last inequality uses the fact that for every $a, b > 0$ and $\alpha \in [0, 1]$, $a^\alpha b^{1-\alpha} \leq \alpha a + (1 - \alpha) b$.

The Hölder inequality implies the following relations between the L_2, L_1 and L_∞ norms of every vector (see Exercise 21.2):

$$|\mathbf{v}|_1 / \sqrt{n} \leq \|\mathbf{v}\|_2 \leq \sqrt{|\mathbf{v}|_1 \|\mathbf{v}\|_\infty} \tag{9}$$

Vector spaces with a norm are sometimes known as *Banach spaces*.

A.5.5 Metric spaces

For any set Ω and $d : \Omega^2 \to \mathbb{R}$, we say that d is a *metric* on Ω if it satisfies the following conditions:

1. $d(x, y) \geq 0$ for every $x, y \in \Omega$ where $d(x, y) = 0$ if and only if $x = y$.
2. $d(x, y) = d(y, x)$ for every $x, y \in \Omega$.
3. (Triangle Inequality) For every $x, y, z \in \Omega$, $d(x, z) \leq d(x, y) + d(y, z)$.

That is, $d(x, y)$ denotes the *distance* between x and y according to some measure. If Ω is a vector space with a norm then the function $d(x, y) = \|x - y\|$ is a metric over Ω, but there are other examples for metrics that do not come from any norm. For example, for every graph G we can define a metric over the vertex set of G by letting the distance of x and y be the length of the shortest path between them. Various metric spaces and the relations between them have found recently many applications in theoretical computer science, see Chapter 15 of [Mat02] for a good survey.

A.6 Polynomials

We list some basic facts about univariate polynomials.

Theorem A.33 *A nonzero polynomial of degree d has at most d distinct roots.* ♢

PROOF: Suppose $p(x) = \sum_{i=0}^{d} c_i x^i$ has $d + 1$ distinct roots $\alpha_1, \ldots, \alpha_{d+1}$ in some field $\mathbb{F}$. Then

$$\sum_{i=0}^{d} \alpha_j^i \cdot c_i = p(\alpha_j) = 0,$$

for $j = 1, \ldots, d + 1$. This means that the system $\mathbf{A}\mathbf{y} = \mathbf{0}$ with

$$\mathbf{A} = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^d \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^d \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & \alpha_{d+1} & \alpha_{d+1}^2 & \ldots & \alpha_{d+1}^d \end{pmatrix}$$

has a solution $\mathbf{y} = \mathbf{c}$. The matrix $\mathbf{A}$ is a *Vandermonde* matrix, and it can be shown that

$$\det \mathbf{A} = \prod_{i>j} (\alpha_i - \alpha_j),$$

which is nonzero for distinct α_i. Hence rank$\mathbf{A} = d + 1$. The system $\mathbf{A}\mathbf{y} = \mathbf{0}$ has therefore only a trivial solution — a contradiction to $\mathbf{c} \neq \mathbf{0}$. ■

This theorem has an interesting corollary:

Corollary A.34 *For every finite field $\mathbb{F}$, the multiplicative group $\mathbb{F}^*$ is cyclic.* ♢

PROOF: The fact that the polynomial $x^k - 1$ has at most k roots implies that the group $\mathbb{F}^*$ has the property **(*)** that for every k the number of elements x satisfying $x^k = 1$ is always at most k. We will prove by induction that every group G satisfying **(*)** is cyclic.

Let $n = |G|$. We consider three cases:

- n is prime. In this case every element of G has either order 1 or order n. Since the only element with order 1 is the identity element, we see that G has an element of order n— G is cyclic.

- $n = p^c$ for some prime p and $c > 1$. In this case if there is no element of order n, then all the orders must divide p^{c-1}. We get $n = p^c$ elements x such that $x^{p^{c-1}} = 1$, violating (*).
- $n = pq$ for co-prime p and q. In this case let H and F be two subgroups of G defined as follows: $H = \{a : a^p = 1\}$ and $F = \{b : b^q = 1\}$. Then $|H| \leq p < n$ and $|F| \leq q < n$ and also as subgroups of G both H and F satisfy (*). Thus by the induction hypothesis both H and F are cyclic and have generators a and b respectively. We claim that ab generates the entire group G. Indeed, let c be any element in G. Since p, q are coprime, there are x, y such that $xq + yp = 1$ and hence $c = c^{xq+yp}$. But $(c^{xq})^p = 1$ and $(c^{yp})^q = 1$ and hence c is a product of an element of H and an element of F, and hence $c = a^i b^j$ for some $i \in \{0, \ldots, p-1\}$ and $j \in \{0, \ldots, q-1\}$. Thus, to show that $c = (ab)^z$ for some z all we need to do is to find z such that $z = i \pmod p$ and $z = j \pmod q$, but this can be done using the Chinese Remainder Theorem.

■

Theorem A.35 *For any set of pairs $(a_1, b_1), \ldots, (a_{d+1}, b_{d+1})$ there exists a unique polynomial $g(x)$ of degree at most d such that $g(a_i) = b_i$ for all $i = 1, 2, \ldots, d+1$.* ♢

PROOF: The requirements are satisfied by *Lagrange Interpolating Polynomial*:

$$\sum_{i=1}^{d+1} b_i \cdot \frac{\prod_{j \neq i}(x - a_j)}{\prod_{j \neq i}(a_i - a_j)}.$$

If two polynomials $g_1(x), g_2(x)$ satisfy the requirements then their difference $p(x) = g_1(x) - g_2(x)$ is of degree at most d, and is zero for $x = a_1, \ldots, a_{d+1}$. Thus, from the previous theorem, polynomial $p(x)$ must be zero and polynomials $g_1(x), g_2(x)$ identical. ■

The following elementary result is usually attributed to Schwartz and Zippel in the computer science community, though it was certainly known earlier (see e.g. DeMillo and Lipton [DLne]).

Lemma A.36 *If a polynomial $p(x_1, x_2, \ldots, x_m)$ over $F = GF(q)$ is nonzero and has total degree at most d, then*

$$\Pr[p(a_1..a_m) \neq 0] \geq 1 - \frac{d}{q},$$

where the probability is over all choices of $a_1..a_m \in F$. ♢

PROOF: We use induction on m. If $m = 1$ the statement follows from Theorem A.33. Suppose the statement is true when the number of variables is at most $m - 1$. Then p can be written as

$$p(x_1, x_2, \ldots, x_m) = \sum_{i=0}^{d} x_1^i p_i(x_2, \ldots, x_m),$$

where p_i has total degree at most $d - i$. Since p is nonzero, at least one of p_i is nonzero. Let k be the largest i such that p_i is nonzero. Then by the inductive hypothesis,

$$\Pr_{a_2, a_3, \ldots, a_m}[p_i(a_2, a_3, \ldots, a_m) \neq 0] \geq 1 - \frac{d-k}{q}.$$

Whenever $p_i(a_2, a_3, \ldots, a_m) \neq 0$, $p(x_1, a_2, a_3, \ldots, a_m)$ is a nonzero univariate polynomial of degree k, and hence becomes 0 only for at most k values of x_1. Hence

$$\Pr[p(a_1..a_m) \neq 0] \geq (1 - \frac{k}{q})(1 - \frac{d-k}{q}) \geq 1 - \frac{d}{q},$$

and the induction is completed. ■

Hints for selected exercises

Chapter 0

0.2 Answers are: (a) n (b) n^2 (c) 2^n (d) $\log n$ (e) n (f) $n \log n$ (g) $n^{\log 3}$ (h) n^2.

Chapter 1

1.1 Follow the gradeschool algorithms.

1.5 Use the proof of Claim 1.6.

1.6 show that the universal TM $\mathcal{U}$ obtained by the proof of Theorem 1.9 can be tweaked to be oblivious.

1.12.b By possibly changing from $\mathcal{S}$ to its complement, we may assume that the empty function $\emptyset$ (that is not defined on any input) is in $\mathcal{S}$ there is some function f that is defined on some input x that is not in $\mathcal{S}$. Use this to show that an algorithm to compute $f_{\mathcal{S}}$ can compute the function HALT_x which outputs 1 on input α iff M_α halts on input x. Then reduce computing HALT to computing HALT_x thereby deriving Rice's Theorem from Theorem 1.11.

Chapter 2

2.2 CONNECTED and 2COL are shown to be in **P** in Exercise 1.14 (though 2COL is called BIPARTITE there). 3COL is shown to be **NP**-complete in Exercise 2.21, and hence it is unlikely that it is in **P**.

2.3 First show that for every rational matrix A, the determinant of A can always be represented using a number of bits that is polynomial in the representation of A. Then use Cramer's rule for expressing the solution of linear equations in terms of determinants.

2.4 Use the previous question.

2.5 The certificate that n is prime is the list of prime factors $q_1, \ldots, q_\ell$ of $n-1$ along with the corresponding numbers $a_1, \ldots, a_\ell$ and (recursive) primality certificates for $q_1, \ldots, q_\ell$.

2.6 A simulation in $O(|\alpha| t \log t)$ time can be obtained by a straightforward adaptation of the proof of Theorem 1.9. To do a more efficient simulation, the main idea is to first run a simulation of M without actually reading the contents of the work tapes, but rather simply non-deterministically guessing these contents, and writing those guesses down. Then, go over tape by tape and verify that all guesses were consistent.

2.11 Why is this language in **NP**? Is Boolean satisfiability a mathematical statement?

2.13.a Modify the machine M so that it clears up its work tape before outputting a 1 and moves both heads to one end of the tape. Then the final snapshot and head locations are unique.

2.15 Reduce from INDSET.

2.17 For Exactly One 3SAT replace each occurrence of a literal v_i in a clause C by a new variable $z_{i,C}$ and clauses and auxiliary variables ensuring that if v_i is TRUE then $z_{i,C}$ is allowed to be either TRUE or FALSE but if v_i is false then $z_{i,C}$ must be FALSE. The approach for the reduction of Exactly One 3SAT to SUBSET SUM is that given a formula φ, we map it to a SUBSET SUM instance by mapping each possible literal u_i to the number $\sum_{j \in S_i} (2n)^j$ where S_i is the set of clauses that the literal u_i satisfies, and setting the target T to be $\sum_{j=1}^{m} (2n)^j$. An additional trick is required to ensure that the solution to the subset sum instance will not include two literals that correspond to a variable and its negation.

2.19 Reduce from SAT

2.20 You can express the constraint $x \in \{0,1\}$ using the equation $x^2 = x$.

2.21 Reduce from 3SAT.

2.22 Reduce from SAT.

2.30 If there is a n^c time reduction from 3SAT to a unary language L, then this reduction can only map size n instances of 3SAT to some string of the form 1^i where $i \leq n^c$. Use this observation to obtain a polynomial-time algorithm for SAT using the downward self reducibility argument of Theorem 2.18.

2.31 Start with an exponential-time recursive algorithm for SUBSET SUM, and show that in this case you can make it into a polynomial-time algorithm by storing previously computed values in a table.

Chapter 3

3.6.a To compute $H(n)$ we need to **(1)** compute $H(i)$ on every $i \leq \log n$, **(2)** simulate at most $\log\log n$ machines on inputs of lengths at most $\log n$ for less than $\log\log n(\log n)^{\log\log n} = o(n)$ steps, and **(3)** compute SAT on inputs of size at most $\log n$. Thus, if $T(n)$ denotes the time to compute $H(n)$, then $T(n) \leq \log nT(\log n) + O(n^2)$, and hence $T(n) = O(n^2)$.

3.6.b If f is the reduction from SAT to SAT_H that runs in time $O(n^i)$, let N be the number such that $H(n) > i$ for $n > N$. The following recursive algorithm A solves SAT in polynomial time: on input a formula φ, if $|\varphi| \leq N$ then compute the output using brute force; otherwise compute $x = f(\varphi)$. If x is not of the form $\psi 01^{n^{H(|\psi|)}}$ then output FALSE . Otherwise, output $A(\psi)$.

Chapter 4

4.6 The proof of the Cook-Levin Theorem in Chapter 2 used oblivious TMs. You need to verify that the construction of oblivious TMs implied in Remark 1.7 and Exercise 1.5 is such that the position of the head at any step can be computed in logarithmic space.

4.7 Use the previous exercise

Chapter 5

5.1 Use the **NP**-completeness of SAT.

5.7 The nontrivial direction $\mathbf{EXP} \subseteq \mathbf{APSPACE}$ uses ideas similar to those in the proof of Theorem 5.11.

5.13.b Reduce from $\mathbf{\Sigma}_3$-3SAT. Also, the collection $\mathcal{S}$ produced by your reduction can use the same set multiple times.

Chapter 6

6.1.a Use the equation $f(x_1,\ldots,x_n) = x_n \wedge f(x_1,\ldots,x_{n-1},1) \vee \overline{x}_n \wedge f(x_1,\ldots,x_{n-1},0)$ to build recursively a $O(2^n)$ circuit for f.

6.1.b There only 2^{2^k} functions on k bits, which means that we can trivially use $2^{2^k} \cdot (k2^{2^k})$ gates to compute every possible such function on $x_1,\ldots,x_k$. But after we have done this, we can use the recursive circuit of the previous item only for $n-k$ levels of the recursion, using up $O(2^{n-k})$ gates. Setting k to equal, say, $\log n - 2$ gives the result.

6.5 Keep in mind the proof of the *existence* of functions with high circuit complexity, and try to show that you can compute, say, the lexicographically smallest such function using a constant number of quantifier alternations.

6.7 Keep the previous problem in mind.

6.9 Show a recursive exponential-time algorithm S that on input a n-variable formula φ and a string $v \in \{0,1\}^n$ outputs 1 iff φ has a satisfying assignment v such that $v > u$ when both are interpreted as the binary representation of a number in $[2^n]$. Use the reduction from SAT to L to prune possibilities in the recursion tree of S.

6.12.a You can use a different processor to compute each entry of AB.

6.12.b Use repeated squaring: $A^{2^k} = (A^{2^{k-1}})^2$.

6.12.c Let A be the adjacency matrix of a graph. What is the meaning of the (i,j)th entry of A^n?

6.13 A formula may be viewed —once we exclude the input nodes—as a directed binary tree, and in a binary tree of size m there is always a node whose removal leaves subtrees of size at most $2m/3$ each.

6.16 First design **NC** circuits for matrix multiplication and then, using fast exponentiation, for computing A^r in $\text{poly}(\log n + \log r)$ depth. Then use the fact that the determinant is the product of the eigenvalues, and that $\text{trace}(A^r)$ is the sum of the rth power of the eigenvalues. Then use manipulations of the symmetric functions of eigenvalues.

6.19 In your reduction, express the CIRCUIT-EVAL problem as a linear program and use the fact that $x \vee y = 1$ iff $x + y \geq 1$. Be careful; the variables in a linear program are real-valued and not Boolean!

Chapter 7

7.3 Use the binary representation of n and repeated squaring.

7.4 Use the fact that if $B_1, \ldots, B_k$ are k independent events each occurring with probability at most p, then the probability that $\wedge_{i \in [n]} B_i$ occurs is at most p^n.

7.5 Think of the real number ρ as an advice string. How can its bits be recovered?

7.8 Follow the ideas of the proof of the Karp-Lipton Theorem (Theorem 6.19).

7.9 Try to compute the probability that the machine ends up in the accept configuration using either dynamic programming or matrix multiplication.

7.11.c Consider the infinite random walk starting from u. If $E_u > K$ then by standard bounds (e.g., Chernoff), u appears in less than a $2/K$ fraction of the places in this walk.

7.11.d Start with the case $k = 1$ (i.e., u and v are connected by an edge), the case of $k > 1$ can be reduced to this using linearity of expectation. Note that the expectation of a random variable X over $\mathbb{N}$ is equal to $\sum_{m \in \mathbb{N}} \Pr[X \geq m]$ and so it suffices to show that the probability that an ℓn^2-step random walk from u does not hit v decays exponentially with ℓ.

Chapter 8

8.1.c Use **IP** = **PSPACE**.

8.5 First note that in the current set lower bound protocol we can have the prover choose the hash function. Consider the easier case of constructing a protocol to distinguish between the case $|S| \geq K$ and $|S| \leq \frac{1}{c}K$ where $c > 2$ can be even a function of K. If c is large enough the we can allow the prover to use *several* hash functions $h_1, \ldots, h_i$, and it can be proven that if i is large enough we'll have $\cup_i h_i(S) = \{0,1\}^k$. The gap can be increased by considering instead of S the set S^ℓ, that is the ℓ times Cartesian product of S.

8.7 Start by showing that **MAM** $\subseteq$ **AM**, where **MAM** denotes the class of languages that can be proven by a 3-message protocol in which the prover sends one message, the verifier sends random coins, and then the prover sends another message (see Footnote 2). We can change an **MAM** protocol to an **AM** protocol by having the verifier send its random coins as the first message. This will not harm completeness. Show that if we first use parallel repetition to reduce the soundness error to a low enough value (as a function of the length of the prover's messages) then the new protocol will still be sound.

8.8.a Show that in this case there is at most a blowup of 2 in the degree due to a product operation

8.8.b If ψ is not already of this form and has a fragment of the form $\forall x_j ....\forall x_{j'} p(x_i, ...)$ where $j' > j > i$ and p is some formula involving x_i and possibly other variables, then we can introduce a new variable y_i and change the formula to the equivalent form$\forall x_j \exists y_i s.t.(y_i = x_i) AND ....\forall x_{j'} p(y_i, ...)$. Apply this procedure iteratively from right to left.

8.13 Show how to simulate poly(n) provers using two. In this simulation, one of the provers plays the role of all $m(n)$ provers, and the other prover is asked to simulate one of the provers, chosen randomly from among the $m(n)$ provers. Then repeat this a few times.

Chapter 9

9.2 Can all the distributions of the form $\mathsf{E}_{U_n}(x)$ have the same support?

9.4 Define $\mathcal{D}$ to be the following distribution over $\{0,1\}^{n+10}$: choose y at random from $\mathsf{E}_{U_n}(0^{n+5})$, choose k at random in $\{0,1\}^n$, and let $x = \mathsf{D}_k(y)$. Show a function A such that if we set $x_0 = 0^{n+10}$ and (11) fails for every x_1, then for every $x \in \{0,1\}^{n+10}$, $\Pr[\mathcal{D} = x] > 2^{-n}$. Derive from this a contradiction.

9.6.a Use padding.

9.7 Show that if $X^2 = Y^2 \pmod M$ and $X \neq \pm Y \pmod M$ then one can find a factor of M by computing the greatest common denominator (gcd) of M and $X - Y$. Then show that you can find such a pair X, Y using an invertion algorithm.

9.8 For every prime p, generator g of $\mathbb{Z}_p^*$, and $x \in \{0, .., p-1\}$, if we choose $y \in_{\mathrm{R}} \{0, .., p-1\}$ then $g^x g^y \pmod p$ is uniformly distributed in $\mathbb{Z}_p^*$.

9.9.b For the algorithm B use $A(\mathsf{E}_{U_n}(0^m))$.

9.9.c Use the same algorithm B as above.

9.10 Use the ideas of the proof of Theorem 9.13.

9.13 You need to show that a certain determinant is nonzero.

9.16 Prove this first for the case where the language 3COL is replaced by $L = \{(y, r, b) : \exists x \text{ s.t. } y = f(x), b = r \odot x\}$, where f is a one-way permutation.

Chapter 10

10.2 First prove that Condition 3 holds iff Condition 1 holds iff Condition 4 holds. This follows almost directly from the definition of the inner product and the fact that for every matrices A, B it holds that $(AB)^* = B^* A^*$ and $(A^*)^* = A$. Then prove that Condition 3 implies Condition 2, which follows from the fact that the norm is invariant under a change of basis. Finally, prove that Condition 2 implies Condition 3 by showing that if two orthogonal unit vectors $\mathbf{v}, \mathbf{u}$ are mapped to non-orthogonal unit vectors $\mathbf{v}', \mathbf{u}'$, then the norm of the vector $\mathbf{u} + \mathbf{v}$ is not preserved.

10.5 Add another qubit to the register with the semantic that when this qubit is zero, all amplitudes correspond to the real part of the amplitudes in the original algorithm and when it is one, the amplitudes correspond to the imaginary part of the amplitudes of the original algorithm.

10.10 Start by solving the case that $x = 2^k$ for some k. Then, show an algorithm for general x by using x's binary expansion.

10.12 Use the fact that if N and A are co-prime then there are whole numbers α, β such that $\alpha N + \beta A = 1$ and multiply this equation by B.

10.15 let $d = gcd(r, M)$, $r' = r/d$ and $M' = M/d$. Now use the same argument as in the case that M and r are coprime to argue that there exist $\Omega(\frac{r}{d \log r})$ values $x \in \mathbb{Z}_{M'}$ satisfying this condition, and that if x satisfies it then so does $x + cM$ for every c.

Chapter 11

11.3 Show that a random assignment is expected to satisfy at least a $7/8$ fraction of the clauses, and then use Markov's inequality to show that the probability of satisfying at least a $7/8 - 1/(2m)$ fraction (where m is the number of clauses) is at least $1/\operatorname{poly}(m)$.

11.4 Use the method of *conditional expectation*. Given any partial assignment to the variables $u_1, \ldots, u_i$, one can compute in polynomial time the expectation of the fraction of clauses satisfied if the variables $u_{i+1}, \ldots, u_n$ are chosen at random. There is a way to assign values to the variables $u_1, u_2, \ldots$ in order so that the invariant that this expectation is at least $7/8$ is always maintained. (Another approach for obtaining a deterministic algorithm is to select the assignment using a 3-wise independent sample space; see hint to Exercise 11.14.))

11.8 Use the hypothesis to infer a downward-self-reducibility property for SAT.

11.9 Design a verifier for 3SAT. The trivial idea would be that the proof contains a satisfying assignment and the verifier randomly picks a clause and reads the corresponding three bits in the proof to check if the clause is satisfied. This doesn't work. Why? The better idea is to require the "proof" to contain many copies of the satisfying assignment. The verifiers uses pairwise independence to run the previous test on these copies —which may or may not be the same string.

11.11 The Cook-Levin reduction actually transforms *every* $x \in \{0,1\}^*$ into a formula almost all of whose clauses are satisfiable since almost all of the clauses are various consistency checks that are satisfied by the transcript of the execution of the corresponding TM M on x and *every* string u, even if $M(x,u) = 0$.

11.12 First show that the problem can be solved exactly using dynamic programming in time $\operatorname{poly}(n,m)$ if all the numbers involved are in the set $[m]$. Then, show one can obtain an approximation algorithm by keeping only the $O(\log(1/\epsilon) + \log n)$ most significant bits of every number.

11.14 As in Exercise 11.4, the randomized algorithm can be derandomized using either the method of conditional expectation or using q-wise independent functions. These can be obtained by generalizing the construction of pairwise independent hash functions from Section 8.2.2 to use a polynomials of degree $q-1$ over $\mathrm{GF}(2^n)$ instead of linear functions.

11.15 Show you can express satisfiability for SAT formulas using quadratic equations.

11.16 Reduce from MAX-3SAT.

Chapter 12

12.1 Let $x_1, \ldots, x_n$ be such that $f(x_i) \neq f(x_i^i)$, prove that for every k, there is a set X of at least $n/2^k$ of the x_i's such that the decision tree sees the same answers for its first k queries on every member of X.

12.2 Use induction.

Chapter 13

13.3 Show that there is no one-tape TM M solving PAL such that for every input of the form $x_{n/2} \cdots x_1 0^n x_1 \cdots x_{n/2}$, and every index $i \in [n/2+1, .., 3n/2-1]$, M travels less than $o(n)$ times between the i^{th} and $i+1^{th}$ cells of its tape. Otherwise by letting Alice simulate M's execution when its head is in the first i cells, and Bob simulate M when the head is in the rest of the tape, we can design a communication complexity protocol for equality that uses $o(n)$ communication for more than $2^{n/2}/n$ inputs.

13.4 As in the previous question, make this into a communication complexity protocol, where Alice and Bob transmit to one another the contents of the working tape. (This time the input tape is read only.) Create a "buffer zone" of zeroes, forcing the machine to take n steps just to transmit every message between Alice and Bob.

13.5 Arbitrarily number the rectangles in the monochromatic tiling and let $N = \chi(f)$. Define graphs G_R, G_C on $\{1, \ldots, N\}$ where $\{i, j\}$ is an edge in G_R (resp., G_C) iff rectangles i, j share a row (resp., column). Let $deg_R(\cdot)$ and $deg_C(\cdot)$ denote degrees in these graphs. At each step, the row player tries to look for a rectangle i containing his input with $deg_L(i) \leq 3|G_R|/4$ and sends such an index i if it exists. Both players then remove from G_L, G_C all vertices that are not neighbors of i. Similarly, the column player tries to find a column j containing his input such that $deg_C(j) \leq 3|G_C|/4$. We claim if either such an i, j can be found, it represents progress— can you see why? Furthermore, can you show they will always find such i, j? It may be helpful to note that in a N-vertex graph with minimum degree at least $N/2+1$, each two vertices have a common neighbor.

13.6 First, show that for every two matrices A, B, $\text{rank}(A + B) \leq \text{rank}(A) + \text{rank}(B)$, implying that if $A = \sum_{i=1}^{\ell} \alpha_i B_i$ for rank-1 matrices $B_1, \ldots, B_\ell$ then $\text{rank}(A) \leq \ell$. Then, use the fact that if A has rank at most ℓ then it has ℓ rows such that all other rows are linear combination of these rows to express A as a sum of ℓ rank-1 matrices $B_1, \ldots, B_\ell$ (the rows of the matrix B_i will be scalar multiples of some row of A).

13.9 Use the fact that $M' = J - 2M$ where J is the all 1's matrix.

13.10 Transform the problem to ± 1 first and compute rank over the reals. Could you prove this by taking rank in GF(2)?

13.11 Lower bound the rank.

13.12 Use the fact that $-1^{a \odot b} - 1^{a' \odot b} - 1^{a \odot b'} - 1^{a' \odot b'} = -1^{(a+a') \odot (b+b')}$.

13.15 Use the *fingerprinting* technique encountered in Section 7.2.3.

13.16 To turn the circuit into a communication protocol, imagine two players, OR and AND. The OR player gets an input such that $f(x) = 0$ and the AND player gets one where $f(y) = 1$. They know that their inputs differ on at least one bit, and use the circuit to figure out which bit this is. They both evaluate the circuit on their inputs. If the top gate is an OR then the OR player sees both incoming wires as 0, whereas the AND player sees at least one incoming wire with a 1 on it. So the AND player communicates a bit about which wire this was. They continue this way down the tree.

To turn a communication protocol into a circuit you have to do something similar and use induction.

13.19 Reduce the task to a communication complexity protocol for disjointness, where Alice sees the first, say, $n/4$ inputs and Bob sees the rest.

Chapter 14

14.1 Each gate in the old circuit gets a twin that computes its negation.

14.3 Start with the trivial representation of f as a CNF that has a clause per each assignment x such that $f(x) = 0$. Then show that each clause C can be replaced with a clause D which is contains at most s of C's literals while still ensuring that if $f(x) = 0$ then $D(x) = 0$ for one of the reduced clauses D.

14.4 Use the equality $\binom{n}{t+k} = \binom{n}{t}\binom{n-t}{k}/\binom{t+k}{t}$ and the estimate $\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k$.

14.10 Show that if $I \subseteq [\ell]$ and $x_1 < x_2 < \ldots < x_m$ is an increasing sequence of numbers in $[2^\ell - 1]$ such that for every i, the most significant bit in which x_i and x_{i-1} differ is *not* in I, then the sequence $x'_1, \ldots, x'_m$ is still increasing, where x'_i is obtained from x_i by "zeroing out" all the bits in I. Conclude that $m \leq 2^{\ell - |I|}$.

Chapter 15

15.1 Try to mimic the obvious exponential-time algorithm for finding a satisfying assignment for φ.

15.2.a For every j, let $d_j(\mathbf{c})$ be zero if the j^{th} clause $\tilde{C}_j$ in the "stripped" refutation can be derived using the $\mathbf{y}$ variables only. Show **(1)** that every $d_j(\mathbf{c})$ can be computed by an $O(S^2)$-sized monotone circuit in $\mathbf{c}$ and **(2)** that we can set $I(\mathbf{c}) = d_S(\mathbf{c})$ in the proof of Theorem 15.4.

15.2.b Use the assignment $z'_i = \neg z_i$ and the function $I' = \neg I$ to reduce to the previous case.

15.4 The difficult part is completeness. A simpler subcase is when the set of axioms include $0 \leq X_i \leq 1$. In this case, try to prove that the derivation rule with D restricted to the value 2 suffices. As warmup in this case, first try to prove that all resolution proofs can be recast as cutting planes proofs of essentially the same size that only involve $D = 2$.

15.6 For $i \leq n+1, j \leq n$, have a variable x_{ij} that is 1 iff i maps to j.

Chapter 16

16.4.a Start by proving this for n's that are powers of k. If $n = k^\ell$ then you can decompose a $k^\ell \times k^\ell$ matrix into k^2 blocks of size $k^{\ell-1}$— use recursion to multiply blocks and the program Π_k to combine the results of the recursion.

16.4.b We don't have a good intuition how to find this program, but since these are just 2×2 matrices, one can do so by trial and error.

16.6.c First use the fact that the determinant of A can be expressed in terms of the determinant of its minors to show that $p(x) = (A_{1,1} - x)\det(M - xI) + \mathbf{r}ADJ(M - xI)\mathbf{c}$, where for every matrix B, $ADJ(B)$ is the matrix whose i, j^{th} entry is equal to $(-1)^{i+j}$ times the determinant of the minor of B with the i^{th} row and j^{th} column removed (i.e., for nonsingular B, $ADJ(B) = \det(B)B^{-1}$). Then use the Caley-Hamilton Theorem (that says that $q_M(M)$ is equal to the zero matrix) to express the coefficients of the matrix-valued polynomial $ADJ(M - xI)$ using the coefficients of q_A and powers of M.

16.8 See Example 16.9.

16.10 First show that it suffices to compute $k!$ where k is the smallest nontrivial factor of n, and in fact it suffices to compute $s!$ where s is a power of 2 larger than k.

Then, noting that $\binom{2r}{r} = \frac{(2r)!}{(r!)^2}$, it suffices to compute $\binom{2r}{r}$ for arbitrary r. But this is just one of the terms of $(t^2+1)^{2r}$. How large does t need to be before $\binom{2r}{r}$ can be "read out" of $(t^2+1)^{2r}$ using an appropriate mod operation?

Chapter 17

17.5 Use hashing and ideas similar to those in the proof of Toda's theorem, where we also needed to estimate the size of a set of strings. If you find this question difficult you might want to come back to it after seeing the Goldwasser-Sipser set lower bound protocol of Chapter 8. To make the algorithm deterministic use the ideas of the proof that $\mathbf{BPP} \subseteq \mathbf{PH}$ (Theorem 7.15).

17.6 Use the proof of Lemma 17.17.

17.7 Real numbers can be approximated by rationals, so it suffices to prove this in the case where the matrices representing the quantum operations only involve rational numbers.

Chapter 18

18.1 A 3-colorable graph better not contain a complete graph on 4 vertices.

18.2 The probability that a random graph has a independent set of size at least k is at most $\binom{n}{k}2^{-\binom{k}{2}}$.

18.5 Construct a sampleable distribution $\mathcal{D}$ on CNF formulae such that it's possible to compute the number of satisfying assignments of a formula φ from the probability of φ in $\mathcal{D}$.

18.6 Use the fact that for every non-negative random variable X and $d \geq 1$, $\mathsf{E}[X^d] \geq \mathsf{E}[X]^d$.

Chapter 19

19.1 Define $Y_i = (-1)^{X_i}$ and $Y = \prod_{i=1}^{k} Y_i$. Then, use the fact that the expectation of a product of independent random variables is the product of their expectations.

19.2 Choose $x \in \{0,1\}^n$ to be in I with probability $\delta 2^n \Pr[H = x]$. Prove that **(1)** $\Pr[|I| \geq \frac{\delta}{2}2^n] > 1/2$ and **(2)** for every circuit C, if we define $SUCESS_C(I)$ to be the probability that $C(x) = f(x)$ for a random $x \in I$ then the probability (over the choice of I) that $SUCCESS_C(I) \geq 1/2 + 2\epsilon$ is smaller than $1/2 2^{-S}$.

19.3 It might help to look at G, H, U as 2^n-dimensional vectors of probabilities.

19.5 Take $\mathbf{z}$ to be the shortest vector of the form $\mathbf{x} - \mathbf{y}$ for $\mathbf{x} \in C$ and $\mathbf{y} \in D$ ($\mathbf{z}$ can be shown to exist and be non-zero using the fact that C, D are closed and D is compact, which means that we can restrict attention to the intersection of C with a sufficiently large ball).

19.6 Note that $\max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{q}A\mathbf{p} > c$ if and only if the convex set $D = \{A\mathbf{p} : \mathbf{p} \in [0,1]^n \ \sum_i p_i = 1\}$ does not intersect with the convex set $C = \{\mathbf{x} \in \mathbb{R}^m : \forall_{i \in [m]} x_i \leq c\}$. Use the Hyperplane Separating Theorem to show that this implies the existence of a probability vector $\mathbf{q}$ such that $\langle \mathbf{q}, \mathbf{y} \rangle \geq c$ for every $\mathbf{y} \in D$.

19.7 Assume that there is a 2^{-k}-density distribution that is outside of this convex set and use the separating hyperplane theorem to derive a contradiction, by rearranging the terms of the distribution according to their inner product with the normal of the hyperplane, and shifting weight until we get a flat distribution.

19.9 Use a greedy strategy, to select the codewords of E one by one, never adding a codeword that is within distance δ to previous ones. When will you get stuck?

19.10 Follow the proof of the Johnson bound and present the problem as asking how many unit vectors in $\mathbb{R}^m$ you can have such that every pair of vectors is pretty far apart.

19.14 See the discussion before the theorem's statement and the proof of Theorem 19.21.

19.15 The first polynomial describes f in an ϵ fraction of points say S_1, the second polynomial describes f in $\epsilon - d/|\mathbb{F}|$ fraction of points S_2 where $S_1 \cap S_2 = \emptyset$, etc.

19.16 Think of $Q(x,y)$ as a univariate polynomial in y with its coefficients being polynomials in x (i.e., elements in the ring $\mathbb{F}[x]$). Then, divide $Q(x,y)$ by $y - P(x)$ to obtain $Q(x,y) = (y - P(x))A(x,y) + R(x,y)$ where the remainder $R(x)$ has $y-$degree smaller than $(y - P(x))$.

19.17.b Use the probabilistic method - show this holds for a random matrix.

19.17.c Use the concatenation of Reed-Solomon over GF(2^k) with the Walsh-Hadamard code.

19.18.c Use concatenation of Reed-Solomon code with the binary code obtained in the previous item. Note that we only apply the binary code on inputs of length $O(\log n)$ and hence can allow exponential-time encoding and decoding algorithms.

Chapter 20

20.2 show that if for every n, a random function mapping n bits to $2^{n/10}$ bits will have desired properties with high probabilities.

20.4 Let G be a pseudorandom generator and consider the following function f: on input $x \in \{0,1\}^{\ell+1}$, $f(x) = 1$ iff there exists $z \in \{0,1\}^\ell$ such that $G(z) = x$.

20.6 Use Theorem 20.6.

20.8 Show that the proofs Theorems 20.6 and 19.27 imply that given a function $f \in \mathbf{EXP}$ with $\mathsf{H}_{\mathsf{avg}}(f)$ that is not bounded from above by any polynomial, one can obtain an $S(\ell)$-pseudorandom generator for a function S that is also not bounded from above by any polynomial (and hence for every polynomial p, $S(\ell) > p(\ell)$ for infinitely many ℓ's).

20.9 Use the fact that the algorithm D can with high probability compute a circuit that decides the language L.

Chapter 21

21.2.a Use the fact that the log function is concave (has negative second derivative) implying that for $a, b > 0$, $\alpha \log a + (1-\alpha)\log b \leq \log(\alpha a + (1-\alpha)b)$.

21.2.c The expression $|\mathbf{v}|_1^2 = \sum_{i,j} |\mathbf{v}_i||\mathbf{v}_j|$ includes all terms occurring in $\|\mathbf{v}\|_2^2$ plus additional non-negative terms.

21.4 Show that for every shortest path between two vertices, if we pick any third vertex in the path then the $d+1$-sized neighborhoods of all the picked vertices are disjoint.

21.5 first show that $\|A\|$ is at most say n^2. Then, prove that for every $k \geq 1$, A^k is also stochastic and $\|A^{2k}\mathbf{v}\|_2 \geq \|A^k\mathbf{v}\|_2^2$ using the equality $\langle \mathbf{w}, B\mathbf{z}\rangle = \langle B^\dagger \mathbf{w}, \mathbf{z}\rangle$ and the inequality $\langle \mathbf{w}, \mathbf{z}\rangle \leq \|\mathbf{w}\|_2 \|\mathbf{z}\|_2$.

21.8 Use the fact that if A is a random-walk matrix of a graph and $\mathbf{v} \perp \mathbf{1}$ then $A\mathbf{v} \perp \mathbf{1}$.

21.10.d Such a path is obtained by taking $k/2$ moves away from the root and $k/2$ moves back. We have $d-1$ choices for every move away from the root, and so this gives us a factor $2^{k \log d/2}$. The choices of when to make the "back moves" give us an additional factor of roughly $\binom{k}{k/2} = 2^{k-o(k)}$. In fact we have to be more careful since we can't make a "back move" when we're already in the root and so have to ensure that we place the moves in a way so that at any point in time we never made more "back moves" than "away moves". This can be ensured by fixing the first t moves to be "away" moves and the last t moves to be "back moves"— for $t = 100 \log k\sqrt{k}$ (which is $o(k)$) this ensures the vast majority of the $\binom{k-2t}{k/2-2t} = 2^{k-o(k)}$ choices for placing the remaining $k/2-2t$ "back moves" will not result in an invalid path. Alternatively we can observe that the number of valid paths is exactly the number of length k valid expressions involving only opening and closing parenthesis. If can be shown that this number is equal to $\frac{1}{k/2+1}\binom{k}{k/2}$ (this is known as the $k/2^{th}$ *Catalan number*).

21.10.e Use the previous items to show that, $1+(n-1)\lambda^k \geq n2^{k-k\log d/2-o(k)}$. The bound follows by taking logs of both sides.

21.11 For every set $S \subseteq n$ with $|S| \leq n/2$, try to bound probability that the number of edges between S and $\bar{S}$ deviates strongly from its expectation.

21.13 Use the probabilistic method - choose S to be a random $n/2$-sized subset of the vertices. For every pair of distinct vertices u, v, the probability that $u \in S$ and $v \in \bar{S}$ or vice versa is at most $1/2$ (it would be exactly half if we chose S with replacements). Hence, since there are $dn/2$ edges in the graph, the expected value of in $E(S, \bar{S})$ is at most $dn/4$.

21.14 You can use Lemma 21.14.

21.15.c Show that if $\mathbf{s}$ is the uniform distribution over S then $\|A\mathbf{s}\|_2^2 \leq 1/n + \lambda^2(\epsilon n + 1/n)$.

21.16 A subset S of at most $n'/2$ vertices in H corresponds to a subset S' of size at most $(1-1/(2c))n$ vertices in G. Use G's expansion to argue about the number of edges between the complement of S' and S.

21.18.b Show that any deterministic function must query the function an exponential number of times.

21.18.c Show that under this condition there is a set S of size at most $2^{n/2}$ such that $\Pr[X \in S] \geq 1/20$.

21.19 Represent distributions over an M-element domain as vectors in $\mathbb{R}^m$, and use the triangle inequality for the L_1 norm.

21.23 Use Lemma 21.14.

21.24 Every subset of the replacement product of G and G' can be thought of as n subsets of the individual clusters. Treat differently the subsets that take up more than $1-\rho/10$ portion of their clusters and those that take up less than that. For the former use the expansion of G, while for the latter use the expansion of G'.

Chapter 22

22.1 Use Lemma 21.10 with $T = V \setminus S$.

22.2 The upper bound is implied by Theorem 21.12. The lower bound can be proven using similar techniques.

22.3 approximate the binomial coefficient using Stirling's formula for approximating factorials.

22.4 Consider the random variable V' defined as V conditioned on $V > 0$, and use the inequality $\mathsf{E}[V'^2] \geq \mathsf{E}[V']^2$.

22.5.e use the test T above combined with linearity testing, self correction, and a simple test to rule out the constant zero function.

22.5.f To transform a $2\mathsf{CSP}_W$ formula φ over n variables into a $q\mathsf{CSP}$ ψ over binary alphabet, use 2^W variables $u_j^1, \ldots, u_j^{2^W}$ for each variable u_j of φ. In the correct proof these variables will contain the long code encoding of u_j. Then, add a set of 2^{W^2} variables $y_i^1, \ldots, y_i^{2^{W^2}}$ for each constraint φ_i of φ. In the correct proof these variables will contain the long code encoding of the assignment for the constraint φ_i. For every constraint of φ, ψ will contain constraints for testing the long code of both the x and y variables involved in the constraint, testing consistency between the x variables and the y variables, and testing that the y variables actually encode a satisfying assignment.

1 $\mathsf{val}(\varphi) = \mathsf{val}(\varphi^{*2}) = 1/2$.

22.12.a Express the function f in the Fourier basis, and use the basic properties of the characters and the fact that $\mathbf{x}, \mathbf{x}'$ and $\mathbf{y}$ are independent.

22.12.b Reduce to the previous case by considering the function $g(\mathbf{x} \circ \mathbf{y}) = f(\mathbf{x} \circ \mathbf{y})\chi_\alpha(\mathbf{x})$.

22.12.c You can estimate the expectation in (13) by evaluating the corresponding functions on randomly chosen inputs.

22.12.d Think of the full depth-n binary labeled by binary strings of length $\leq n$ (with the root being the empty word and the two children of α are $\alpha 0$ and $\alpha 1$), then by Parseval you can show that at any level of this tree there can be at most $1/\epsilon^2$ strings α such that $\tilde{f}_{\alpha\star} \geq \epsilon$. Use the procedure `Estimate` to prune this tree from the root to the leaves, throwing away all branches α for which $\tilde{f}_{\alpha\star} < 10\epsilon$. At the end output the remaining leaves.

22.13 Show that a randomly chosen family of subsets suffices.

22.14 Requires constructions of ϵ-biased random variables, which have not been covered in this book, though cab be obtained from linear error correcting codes.

22.15 Think of ways to "amplify" the gap of a constant factor in Problem 11.16. You need to combine equations to get new equations.

22.16 Introduce a bunch of new variables for each variable that occurs in more than 5 clauses. Design a "gadget" consisting of new clauses that force this bunch of new variables to have the same value in the optimum assignment. You might need to use an expander. This is essentially the same problem as Claim 22.37.

Chapter 23

23.3 If DISCRETE LOG is hard on worst-case inputs with respect to a particular prime p then it is hard on most inputs with respect to this prime p, and then it can be used to construct pseudorandom functions (assuming p is used as non-uniform advice).

23.4 It suffices to prove this when f_n is a random function. Use induction on the number of variables, and the fact that both f_n and $\overline{f_n}$ are random functions.

23.5 See Exercise 8.8.

Main Theorems and Definitions

Bibliography

[Aar03] S. Aaronson. Is P versus NP formally independent? *Bulletin of the EATCS*, 81:109–136, 2003.

[Aar05] S. Aaronson. NP-complete problems and physical reality. *SIGACT News*, 36, 2005.

[Aar06] S. Aaronson. Oracles Are Subtle But Not Malicious. *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, pages 340–354, 2006.

[Aar08] S. Aaronson. The limits of quantum computers. *Scientific American*, pages 62–69, Mar. 2008.

[AB87] N. Alon and R. B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[ABSS93] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *J. Comput. Syst. Sci.*, 54(2):317–331, 1997.

[AC86] N. Alon and F. R. K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 72:15–19, 1988. Prelim version Japan Conf on Graph Theory and Applications '86.

[ACR96] A. E. Andreev, A. E. F. Clementi, and J. D. P. Rolim. A new general derandomization method. *J. ACM*, 45(1):179–213, 1998. Prelim version ICALP' 96.

[ACR+07] A. Ambainis, A. M. Childs, B. Reichardt, R. Spalek, and S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^1/2 + o(1)$ on a quantum computer. In *FOCS*, pages 363–372. IEEE, 2007.

[AD97] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC*, pages 284–293. ACM, 1997.

[adH88] F. M. auf der Heide. Fast algorithms for N-dimensional restrictions of hard problems. *J. ACM*, 35(3):740–747, 1988.

[ADH97] L. M. Adleman, J. Demarrais, and M.-D. A. Huang. Quantum computability. *SIAM J. Comput*, 26(5):1524–1540, Oct. 1997.

[Adl78] L. Adleman. Two theorems on random polynomial time. In *FOCS*, pages 75–83. IEEE, 1978.

[AFWZ95] N. Alon, U. Feige, A. Wigderson, and D. Zuckerman. Derandomized graph products. *Computational Complexity*, 5(1):60–75, 1995.

[AG94] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM J. Comput*, 23(5):1026–1049, Oct. 1994.

[AGIK07] D. Aharonov, D. Gottesman, S. Irani, and J. Kempe. The power of quantum systems on a line. In *FOCS*, pages 373–383. IEEE, 2007.

[AIK04] B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. *SIAM J. Comput*, 36(4):845–888, 2006. Prelim version FOCS' 04.

[AIV93] S. Arora, R. Impagliazzo, and U. Vazirani. Relativizing versus nonrelativizing techniques: the role of local checkability. Unpublished manuscript, available from the authors' web pages., 1993.

[Ajt83] M. Ajtai. Σ^1_1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[Ajt88] M. Ajtai. The complexity of the pigeonhole principle. In *FOCS*, pages 346–355. IEEE, 1988.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108. ACM, 1996.

[AKL+79] R. Aleliunas, R. M. Karp, L. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *FOCS*, pages 218–223. IEEE, 29–31 Oct. 1979.

STOC— ACM Symposium on Theory of computing; FOCS— IEEE Annual Symposium on Foundations of Computer Science.

[AKS87] M. Ajtai, J. Komlos, and E. Szemeredi. Deterministic simulation in LOGSPACE. In *STOC*, pages 132–140. ACM, 1987.

[AKS98] N. Alon, M. Krivelevich, and B. Sudakov. Finding a large hidden clique in a random graph. *Random Struct. Algorithms*, 13(3-4):457–466, 1998. Prelim version SODA' 98.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Ann. of Math. (2)*, 160(2):781–793, 2004.

[AL95] S. Arora and C. Lund. Hardness of approximations. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 10. PWS, 1995.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998. Prelim version FOCS '92.

[Alo86] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[AM84] N. Alon and V. D. Milman. Eigenvalues, expanders and superconcentrators (extended abstract). In *FOCS*, pages 320–322. IEEE, 24–26 Oct. 1984.

[AM85] N. Alon and V. D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *J. Comb. Theory Series B*, 38:73–88, 1985.

[Amb04] A. Ambainis. Quantum search algorithms. *SIGACT News*, 35(2):22–35, 2004.

[AMS96] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. Prelim version STOC '96.

[AN04] N. Alon and A. Naor. Approximating the cut-norm via grothendieck's inequality. *SIAM J. Comput*, 35(4):787–803, 2006. Prelim versoin in STOC' 04.

[And85] A. E. Andreev. On a method for obtaining lower bounds for the complexity of individual monotone functions. *Soviet Math. Dokl*, 31(3):530–534, 1985.

[AR04] D. Aharonov and O. Regev. Lattice problems in NP intersect coNP. *J. ACM*, 52:749–765, 2005. Prelim version FOCS '04.

[Aro94] S. Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems*. PhD thesis, UC Berkeley, 1994.

[Aro96] Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45, 1998. Prelim version FOCS '96.

[AS92] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, Jan. 1998. Prelim version FOCS '92.

[AS00a] N. Alon and B. Sudakov. Bipartite subgraphs and the smallest eigenvalue. *Combinatorics, Probability & Computing*, 9(1), 2000.

[AS00b] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley, 2000.

[ATSWZ97] R. Armoni, A. Ta-Shma, A. Widgerson, and S. Zhou. An $O(\log(n)^{4/3})$ space algorithm for (s,t) connectivity in undirected graphs. *J. ACM*, 47(2):294–311, Mar. 2000. Prelim version STOC '97.

[AUY83] A. V. Aho, J. D. Ullman, and M. Yannakakis. On notions of information transfer in VLSI circuits. In *STOC*, pages 133–139. ACM, 1983.

[AvDK+04] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM J. Comput*, 37(1):166–194, 2007. Prelim version FOCS '04.

[AW08] S. Aaronson and A. Wigderson. Algebrization: a new barrier in complexity theory. In *STOC*, pages 731–740. ACM, 2008.

[Bab85] L. Babai. Trading group theory for randomness. In *STOC*, pages 421–429. ACM, 1985.

[Bab90] L. Babai. E-mail and the unexpected power of interaction. In *Proceedings, Fifth Annual Structure in Complexity Theory Conference*, pages 30–44. IEEE, 8–11 July 1990.

[Bab94] L. Babai. Transparent proofs and limits to approximations. In *First European Congress of Mathematicians*, 1994.

[Bar86] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, Feb. 1989. Prelim version STOC' 86.

[Bar02] B. Barak. A probabilistic-time hierarchy theorem for "slightly non-uniform" algorithms. In *RANDOM*, volume 2483 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2002.

[BB84] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, 175, 1984.

[BBBV97] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput*, 26(5):1510–1523, Oct. 1997.

[BBC+98] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001. Prelim version FOCS' 98.

[BBR88] C. H. Bennett, G. Brassard, and J. Robert. Privacy amplification by public discussion. *SIAM J. Comput*, 17(2):210–229, Apr. 1988.

[BBR92] D. A. M. Barrington, R. Beigel, and R. Rudich. Representing Boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4(4):367–382, 1994. Prelim version STOC '92.

[BC06] M. Braverman and S. Cook. Computing over the reals: Foundations for scientific computing. *Notices of the AMS*, 2006.

[BCC86] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, Oct. 1988. Prelim versions by Brassard and Crépeau (CRYPTO '86, FOCS '86) and Chaum (CRYPTO' 86).

[BCE+95] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. In *STOC*, pages 303–314. ACM, 1995.

[BCS97] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Springer Verlag, 1997.

[BCSS97] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer Verlag, 1997.

[BDCGL89] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, Apr. 1992. Prelim version Structures in Complexity '89.

[BdW02] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288:21–43, 2002.

[BE76] B. Bollobás and P. Erdös. Cliques in random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 80(41):419–427, 1976.

[Bec91] J. Beck. An algorithmic approach to the lovàsz local lemma. *Random Structures and Algorithms*, 2(4):367–378, 1991.

[Bel64] J. S. Bell. On the Einstein-Podolsky-Rosen paradox. *Physics*, 1(3):195–290, 1964.

[Ben87] C. H. Bennett. Demons, engines and the second law. *Scientific American*, 257(5):88–96, 1987.

[Berch] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.*, 18(3):147–150, 1984, March.

[BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *lncs*, pages 37–48. Springer, 22–24 Feb. 1990.

[BFL90] L. Babai, L. Fortnow, and L. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991. Prelim version FOCS '90.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–32. ACM, 1991.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BFT98] H. Buhrman, L. Fortnow, and T. Thierauf. Nonrelativizing separations. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CCC-98)*, pages 8–12. IEEE, June 15–18 1998.

[BG94] M. Bellare and S. Goldwasser. The complexity of decision versus search. *SIAM J. Comput*, 23(1):97–119, 1994.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM J. Comput*, 4(4):431–442, Dec. 1975.

[BGS95] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and nonapproximability-towards tight results. *SIAM J. Comput*, 27(3):804–915, 1998.

[BHZ87] R. B. Boppana, J. Hastad, and S. Zachos. Does co-NP have short interactive proofs? *Inf. Process. Lett.*, 25(2):127–132, 1987.

[BK95] M. Blum and S. Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, Jan. 1995.

[BLR90] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[Blu67] M. Blum. A machine-independent theory of the complexity of recursive functions. *J. ACM*, 14, 1967.

[Blu84] M. Blum. Independent unbiased coin flips from a correlated biased source: a finite state Markov chain. In *FOCS*, pages 425–433. IEEE, 24–26 Oct. 1984.

[Blu87] M. Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987.

[BLY92] A. Björner, L. Lovász, and A. C. C. Yao. Linear decision trees: volume estimates and topological bounds. In *STOC*, pages 170–177. ACM, 1992.

[BM82] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput*, 13(4):850–864, Nov. 1984. Prelim version FOCS' 82.

[BM88] L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, Apr. 1988.

[BMMS00] J. C. Birget, S. Margolis, J. Meakin, and M. Sapir. *Algorithmic Problems in Groups and Semigroups*. Birkhauser, 2000.

[BNS89] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *J. Comput. Syst. Sci.*, 45(2):204–232, 1992. Prelim version STOC '89.

[BO83] M. Ben-Or. Lower bounds for algebraic computation trees. In *STOC*, pages 80–86. ACM, 1983.

[BOGKW88] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *STOC*, pages 113–131. ACM, 2–4 May 1988.

[Bol01] B. Bollobás. *Random Graphs*. Cambridge University Press, 2001.

[Bou02] J. Bourgain. On the distribution of the fourier spectrum of boolean functions. *Israel J. Math.*, 131(1):269–276, 2002.

[BP73] L. A. Bassalygo and M. S. Pinsker. The complexity of an optimal non-blocking commutation scheme without rezzzorganization. *Problemy Peredaci Informacii*, 9(1):84–87, 1973. Translated into english in Problems of Information Transmission, 9(1974) 64-66.

[BPR97] M. Bonet, T. Pitassi, and R. Raz. Lower bounds for cutting planes proofs with small coefficients. *J. Symbolic Logic*, 62(3):708–728, 1997.

[Bra04] G. Brassard. Quantum communication complexity: A survey. In *ISMVL*, page 56. IEEE, 2004.

[Bru04] C. Bruce. *Schrodinger's Rabbits: Entering The Many Worlds Of Quantum*. Joseph Henry Press, 2004.

[BS90] R. B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1. Elsevier and MIT Press, 1990.

[BS94] M. Bellare and M. Sudan. Improved non-approximability results. In *STOC*, pages 184–193. ACM, 1994.

[BS96] E. Bach and J. Shallit. *Algorithmic Number Theory — Efficient Algorithms*, volume I. MIT Press, 1996.

[BS08] D. Boneh and V. Shoup. *A graduate course in applied cryptography*. 2008. To appear. Prelim drafts available on `http://crypto.stanford.edu/~dabo/cryptobook/`.

[BSS89] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *American Mathematical Society*, 21(1), 1989.

[BT91] R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4(4):350–366, 1994. Prelim version FOCS '91.

[Bus90] S. R. Buss. Axiomatizations and conservations results for fragments of Bounded Arithmetic. In *Logic and Computation, Contemporary Mathematics* 106, pages 57–84. American Math. Society, 1990.

[BV93] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comput*, 26(5):1411–1473, Oct. 1997. Prelim version STOC 1993.

[BW86] E. R. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent Number 4,633,470, 1986.

[BYJKS02] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004. Prelim version FOCS '02.

[CA08] A. Chattopadhyay and A. Ada. Multiparty communication complexity of disjointness. *ECCC archive*, 2008. Report TR08-002.

[Can96] R. Canetti. More on BPP and the polynomial-time hierarchy. *Inf. Process. Lett.*, 57(5):237–241, 1996.

[CDR86] S. Cook, C. Dwork, and R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput*, 15(1):87–97, Feb. 1986.

[CFL83] A. K. Chandra, M. L. Furst, and R. J. Lipton. Multi-party protocols. In *STOC*, pages 94–99. ACM, 25–27 Apr. 1983.

[CG85] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput*, 17(2):230–261, Apr. 1988. Prelim version FOCS' 85.

[Cha94] B. Chazelle. A spectral approach to lower bounds with applications to geometric searching. *SIAM J. Comput*, 27(2):545–556, 1998. Prelim version FOCS' 94.

[Che70] J. Cheeger. A lower bound for the smallest eigenvalue of the Laplacian. In *Problems in analysis (Papers dedicated to Salomon Bochner, 1969)*, pages 195–199. Princeton Univ. Press, 1970.

[CHSH69] J. F. Clauser, M. A. Horne, A. Shimony, and R. A. Holt. Proposed experiment to test local hidden-variable theories. *Phys. Rev. Lett.*, 23(15):880–884, Oct. 1969.

[Chu36] A. Church. An Unsolvable Problem of Elementary Number Theory. *Amer. J. Math.*, 58(2):345–363, 1936.

[Chu90] F. R. K. Chung. Quasi-random classes of hypergraphs. *Random Struct. Algorithms*, 1(4):363–382, 1990.

[Chv73] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[CK00] P. Crescenzi and V. Kann. A compendium of NP optimization problems. `http://www.nada.kth.se/~viggo/problemlist/`, 2000. Website tracking the tractability of many NP problems.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[Cob64] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30. Elsevier/North-Holland, 1964.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd Ann. ACM Symp. Theory of Computing*, pages 151–158. ACM, 1971.

[Coo72] S. A. Cook. A hierarchy for nondeterministic time complexity. *J. Comput. Syst. Sci.*, 7(4):343–353, Aug. 1973. Prelim version STOC '72.

[Coo75] S. A. Cook. Feasibly constructive proofs and the propositional calculus. In *STOC*, pages 83–97. ACM, 1975.

[CRVW02] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *STOC*, pages 659–668. ACM, May 19–21 2002.

[CS88] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *J. ACM*, 35(4):759–768, 1988.

[Csa76] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM J. Comput*, 1976.

[CSWY01] A. Chakrabarti, Y. Shi, A. Wirth, and A. Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *FOCS*, pages 270–278. IEEE, Oct. 14–17 2001.

[CT65] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Computing*, 19:297–301, 1965.

[CW77] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979. Prelim version STOC '77.

[CW89] A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *FOCS*, pages 14–19. IEEE, 30 Oct.–1 Nov. 1989.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, Mar. 1990.

[Dav65] M. Davis. *The Undecidable*. Dover Publications, 1965.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM J. Comput*, 30(2):391–437, 2000. Prelim version STOC '91.

[Deu85] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc Roy Soc Lond A*, A400:97–117, 1985.

[Deu89] D. Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London Ser. A*, A425:73–90, 1989.

[DFR+07] I. Damgård, S. Fehr, R. Renner, L. Salvail, and C. Schaffner. A tight high-order entropic quantum uncertainty relation with applications. In *Proceedings of 27th CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 360–378. Springer, 2007.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(5):644–654, Nov. 1976.

[DHS94] M. Dietzfelbinger, J. Hromkovic, and G. Schnitger. A comparison of two lower-bound methods for communication complexity. *Theor. Comput. Sci*, 168(1):39–51, 1996. Prelim version MFCS' 94.

[Din06] I. Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3), 2007.

[DJ92] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proc Roy Soc Lond A*, 439:553–558, Oct. 1992.

[DK00] D. Z. Du and K. I. Ko. *Theory of Computational Complexity*. Wiley, 2000.

[DL93] P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is NP-hard. *Artificial Intelligence*, 1993.

[DLne] R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978, June.

[dLMSS56] K. de Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro. Computability by probabilistic machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 183–212. Princeton University Press, 1956. Annals of Mathematics Studies #34.

[Dod84] J. Dodziuk. Difference equations, isoperimetric inequality and transience of certain random walks. *Trans. Amer. Math. Soc.*, 284(2):787–794, 1984.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 7(3):201–215, 1960.

[DPV06] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. *Algorithms*. McGraw Hill, 2006. Draft available from the authors' webpage.

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[DvM05] S. Diehl and D. van Melkebeek. Time-space lower bounds for the polynomial hierarchy on randomized machines. *SIAM J. Comput*, 56:563–594, 2006. Prelim version ICALP' 05.

[Edm65] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math*, 17:449–467, 1965.

[EL75] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. II*, pages 609–627. Colloq. Math. Soc. János Bolyai, Vol. 10. North-Holland, 1975.

[Ell99] J. H. Ellis. The history of non-secret encryption. *Cryptologia*, 23(3):267–273, Apr. 1999.

[EPR35] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47(10):777–780, May 1935.

[ER59] P. Erdos and A. Renyi. On random graphs. *Publ. Math. Debrecen*, 6(290), 1959.

[ER60] P. Erdős and R. Rado. Intersection theorems for systems of sets. *J. London Math. Soc.*, 35:85–90, 1960.

[Fei91] U. Feige. On the success probability of the two provers in one-round proof systems. In *Proceedings of the 6th Annual Conference on Structure in Complexity Theory, CSCT'91 (Chicago, Illinois, June 30-July 3, 1991)*, pages 116–123. IEEE, 1991.

[Fei95] U. Feige. A tight upper bound on the cover time for random walks on graphs. *Random Struct. Algorithms*, 6(1):51–54, 1995.

[Fei96] U. Feige. A threshold of ln for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[Fey82] R. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6&7):467–488, 1982.

[FG06] J. Flum and M. Grohe. *Parametrized Complexity*. Springer, 2006.

[FGG07] E. Farhi, J. Goldstone, and S. Gutmann. A quantum algorithm for the Hamiltonian NAND tree. *Arxiv preprint quant-ph/0702144*, 2007.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996. Prelim version FOCS '91.

[Fis04] E. Fischer. The art of uninformed decisions: A primer to property testing. *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, 2004.

[FK93] U. Feige and J. Kilian. Two-prover protocols - low error at affordable rates. *SIAM J. Comput*, 30(1):324–34, 2000.

[FKO06] U. Feige, J. H. Kim, and E. Ofek. Witnesses for non-satisfiability of dense random 3CNF formulas. In *FOCS*, pages 497–508. IEEE, 2006.

[FL92] U. Feige and L. Lovász. Two-prover one-round proof systems: Their power and their problems (extended abstract). In *STOC*, pages 733–744. ACM, 1992.

[FLvMV00] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-space lower bounds for satisfiability. *J. ACM*, 52:835–865, 2005. Prelim version CCC' 2000.

[FM05] G. S. Frandsen and P. B. Miltersen. Reviewing bounds on the circuit size of the hardest functions. *Information Processing Letters*, 95(2):354–357, 2005.

[FO04] U. Feige and E. Ofek. Easily refutable subformulas of large random 3CNF formulas. In *Proceedings of 31st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3142 of *Lecture Notes in Computer Science*, pages 519–530. Springer, 2004.

[For89] L. J. Fortnow. *Complexity-Theoretic Aspects of Interactive Proof Systems*. PhD thesis, Massachussets Institute of Technology, 1989.

[For97a] L. Fortnow. Time-space tradeoffs for satisfiability. *J. Comput. Syst. Sci.*, 60(2):337–353, 2000. Prelim version CCC' 1997.

[For97b] L. Fortnow. Counting complexity. In L. Hemaspaandra and A. Selman, editors, *A Complexity Theory Retrospective II*. Springer Verlag, 1997.

[FR98] A. M. Frieze and B. Reed. Probabilistic analysis of algorithms. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, volume 16 of *Algorithms and Combinatorics*, pages 36–92. Springer-Verlag, 1998.

[Fri99] E. Friedgut. Sharp thresholds of graph properties, and the k-sat problem. *J. Amer. Math. Soc.*, 12(4):1017–1054, 1999. With an appendix by Jean Bourgain.

[FRS88] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.

[FS88] L. Fortnow and M. Sipser. Are there interactive protocols for coNP-languages? *Inf. Process. Lett.*, 28(5):249–251, 1988.

[FS04] L. Fortnow and R. Santhanam. Hierarchy theorems for probabilistic polynomial time. In *FOCS*, pages 316–324. IEEE, 2004.

[FSS81] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984. Prelim version FOCS '81.

[Gal63] R. G. Gallager. *Low-Density Parity-Check Codes*. The M.I.T. Press, 1963.

[Gar72] M. R. Garey. Optimal binary identification procedures. *SIAM Journal on Applied Mathematics*, 23(2):173–186, Sept. 1972.

[GG79] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. Syst. Sci.*, 22(3):407–420, June 1981. Prelim version FOCS' 79.

[GG00] O. Goldreich and S. Goldwasser. On the limits of nonapproximability of lattice problems. *J. Comput. Syst. Sci.*, 60, 2000.

[GGH+07] S. Goldwasser, D. Gutfreund, A. Healy, T. Kaufman, and G. N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449. ACM, 2007.

[GGM84] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, Oct. 1986. Prelim version FOCS '84.

[Gil74] J. T. Gill. Computational complexity of probabilistic Turing machines. In *STOC*, pages 91–95. ACM, 1974.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM J. Comput*, 6(4):675–695, Dec. 1977.

[Gil93] D. Gillman. A chernoff bound for random walks on expander graphs. *SIAM J. Comput*, 27(4):1203–1220, 1998. Prelim version FOCS' 93.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GK01] A. Goerdt and M. Krivelevich. Efficient recognition of random unsatisfiable k-SAT instances by spectral methods. In *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001, Proceedings*, volume 2010 of *Lecture Notes in Computer Science*, pages 294–304. Springer, 2001.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32. ACM, 15–17 May 1989.

[GLR+91] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *STOC*, pages 32–42. ACM, 1991.

[GM82] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, Apr. 1984. Prelim version STOC' 82.

[GMR84] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput*, 17(2):281–308, Apr. 1988. Prelim version CRYPTO' 84.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput*, 18(1):186–208, Feb. 1989. Prelim version STOC '85.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, July 1991. Prelim version FOCS '86.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[GNW95] O. Goldreich, N. Nisan, and A. Wigderson. On yao's XOR-lemma. *Electronic Colloquium on Computational Complexity (ECCC)*, 2(50), 1995.

[Gol97] O. Goldreich. Notes on levin's theory of average-case complexity. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(58), 1997.

[Gol04] O. Goldreich. *Foundations of Cryptography, Volumes 1 and 2*. Cambridge University Press, 2001,2004.

[Gol08] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge Univerity Press, 2008. Online drafts available at `http://www.wisdom.weizmann.ac.il/~oded/cc-drafts.html`.

[Gom63] R. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent advances in mathematical programming*, pages 269–302. McGraw-Hill, 1963.

[Gow01] W. Gowers. A new proof of Szemerédi's theorem. *Geometric And Functional Analysis*, 11(3):465–588, 2001.

[Gow07] W. T. Gowers. Hypergraph regularity and the multidimensional Szemerédi theorem. *Ann. of Math. (2)*, 166(3):897–946, 2007.

[GR06] V. Guruswami and A. Rudra. Explicit capacity-achieving list-decodable codes. In *STOC*, pages 1–10. ACM, 2006.

[Gra66] R. Graham. Bounds for certain multiprocessor anomalies. *Bell Sys Tech J*, 45:1563–1581, 1966.

[Gro83] M. Gromov. Filling Riemannian manifolds. *J. Differential Geom.*, 18(1):1–147, 1983.

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 22–24 May 1996.

[GS87] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali, editor, *Randomness and Computation*. JAI Press, 1987. Extended Abstract in *Proc. 18th ACM Symp. on Theory of Computing*, 1986.

[GS92] P. Gemmell and M. Sudan. Highly resilient correctors for polynomials. *Information Processing Letters*, 43(4):169–174, 28 Sept. 1992.

[GS98] V. Guruswami and M. Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999. Prelim version FOCS' 98.

[GST04] O. Goldreich, M. Sudan, and L. Trevisan. From logarithmic advice to single-bit advice. *Electronic Colloquium on Computational Complexity (ECCC)*, 2004. Report TR04-093.

[GUV07] V. Guruswami, C. Umans, and S. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. In *Proc. of CCC*, pages 96–108. IEEE, 2007.

[GVW00] O. Goldreich, S. Vadhan, and A. Wigderson. Simplified derandomization of BPP using a hitting set generator. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(4), 2000.

[Haj90] P. Hajnal. On the power of randomness in the decision tree model. In *Proceedings, Fifth Annual Structure in Complexity Theory Conference*, pages 66–77. IEEE, 8–11 July 1990.

[Hak85] A. Haken. The intractability or resolution. *Theoretical Comput. Sci.*, 39:297–308, 1985.

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, Apr. 1950. Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[Hås86] J. Håstad. Almost optimal lower bounds for small depth circuits. In *STOC*, pages 6–20. ACM, 28–30 May 1986.

[Hås96] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.*, 182(1):105–142, 1999.

[Hås97] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. Prelim version STOC '97.

[Hea06] A. Healy. Randomness-efficient sampling within NC ^{1}. In *APPROX-RANDOM*, volume 4110 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2006.

[HILL99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput*, 28(4):1364–1396, Aug. 1999.

[HLW06] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *BAMS: Bulletin of the American Mathematical Society*, 43, 2006.

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *"Introduction to Automata Theory, Language, and Computation"*. Addison–Wesley, 2nd edition edition, 2001.

[HO02] L. A. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*. Springer-Verlag, 2002.

[Hoc97] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, 1997.

[Hod83] A. Hodges. *Alan Turing: the enigma*. Burnett Books, 1983.

[Hof82] C. M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism*, volume 136 of *Lecture Notes in Computer Science*. Springer, 1982.

[Hoh30] G. Hoheisel. Primzahlprobleme in der analysis. *Sitzunsberichte der Kniglich Preussischen Akademie der Wissenschaften zu Berlin*, 33:3–11, 1930.

[Hol07] T. Holenstein. Parallel repetition: simplifications and the no-signaling case. In *STOC*, pages 411–419. ACM, 2007.

[HPV75] J. Hopcroft, W. Paul, and L. Valiant. On Time Versus Space. *J. ACM*, 24(2):332–337, 1977. Prelim version FOCS '75.

[HR76] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Inf. Process. Lett*, 5(1):15–17, 1976.

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[HS66] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape Turing machines. *J. ACM*, 13(4):533–546, Oct. 1966.

[HVV04] A. Healy, S. Vadhan, and E. Viola. Using nondeterminism to amplify hardness. In *STOC*, pages 192–201. ACM, June 13–15 2004.

[IKW01] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002. Prelim version CCC' 01.

[IL90] R. Impagliazzo and L. A. Levin. No better ways to generate hard NP instances than picking uniformly at random. In *FOCS*, pages 812–823. IEEE, Oct. 1990.

[ILL89] R. Impagliazzo, L. A. Levin, and L. Luby. Pseudo-random generation from one-way functions. In *STOC*, pages 12–24. ACM, May 1989.

[ILMR05] K. Iwama, O. Lachish, H. Morizumi, and R. Raz. An explicit lower bound of $5n - o(n)$ for boolean circuits. manuscript, 2006. Prelim versions by Raz and Lachish (STOC '01), and Iwama and Morizumi (manuscript, 2005).

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput*, 17(5):935–938, 1988.

[Imm99] N. Immerman. *Descriptive Complexity*. Springer, 1999.

[Imp95a] R. Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory Conference*, pages 134–147, 1995.

[Imp95b] R. Impagliazzo. Hard-core distributions for somewhat hard problems. In *FOCS*, pages 538–544. IEEE, 1995.

[Ing37] A. E. Ingham. On the difference between consecutive primes. *Quarterly Journal of Mathematics (Oxford Series)*, 8:255–266, 1937.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *STOC*, pages 356–364. ACM, 23–25 May 1994.

[IPZ98] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. Prelim version FOCS '98.

[ISW99] Impagliazzo, Shaltiel, and Wigderson. Reducing the seed length in the nisan-wigderson generator. *Combinatorica*, 26, 2006. Prelim versions in FOCS' 99 and STOC '00.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC*, pages 220–229, New York, 1997. ACM.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs time: Derandomization under a uniform assumption. *J. Comput. Syst. Sci.*, 63, 2001. Prelim version FOCS '98.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *FOCS*, pages 248–253. IEEE, 30 Oct.–1 Nov. 1989.

[JM85] S. Jimbo and A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7(4):343–355, 1987. Prelim version STOC' 85.

[Joh62] S. Johnson. A new upper bound for error correcting codes. *IRE Transaction on Information Theory*, pages 203–207, 1962.

[Joh74] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[Joh84] D. S. Johnson. Solving **NP**-hard problems quickly (on average). *J. Algorithms*, 5(2):284–299, 1984.

[JS93] M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM J. Computing*, 1993.

[JSV01] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *J. ACM*, 51(4), 2004. Prelim version STOC '01.

[JVV86] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comp. Sc.*, 43(2–3):169–188, 1986.

[Kab00] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *J. Comput. Syst. Sci.*, 63(2):236–252, 2001. Prelim version CCC' 00.

[Kah92] N. Kahale. Eigenvalues and Expansion of Regular Graphs. *J. ACM*, 42:1091–1106, 1995. Prelim version FOCS' 92.

[Kah96] D. Kahn. *The codebreakers: the story of secret writing*. Scribner, revised edition, 1996.

[Kah97] N. Kahale. Large deviation bounds for markov chains. *Combinatorics, Probability and Computing*, 6(04):465–474, 1997.

[Kan81] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1–3):40–56, 1982. Prelim version FOCS '81.

[Kan83] R. Kannan. Alternation and the power of nondeterminism. In *STOC*, pages 344–346. ACM, 1983.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, 1972.

[Kha79] L. C. Khačiyan. Polynomial algorithm for linear programming. *Soviet Doklady*, 244:1093–1096, 1979. Typed translation.

[Kho02] S. Khot. On the power of unique 2-prover 1-round games. In *STOC*, pages 767–775. ACM, 2002.

[Kho05] S. Khot. Guest column: inapproximability results via long code based PCPs. *SIGACT News*, 36(2):25–42, 2005.

[KI03] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *STOC*, pages 355–364. ACM, 2003.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31. ACM, 2–4 May 1988.

[Kin88] V. King. An omega($n^5/4$) lower bound on the randomized complexity of graph properties. *Combinatorica*, 11(1):23–32, 1991. Prelim version STOC' 88.

[Kit97] A. Y. Kitaev. Quantum computations: Algorithms and error correction. *RMS: Russian Mathematical Surveys*, 52(6):1191–1249, 1997.

[KKL88] J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions (extended abstract). In *FOCS*, pages 68–80. IEEE, 1988.

[KKMO05] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM J. Comput*, 37(1):319–357, 2007.

[KL80] R. Karp and R. Lipton. Turing machines that take advice. *L' Ensignement Mathématique*, 28:191–210, 1982.

[KL07] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[KM91] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. *SIAM J. Comput*, 22(6):1331–1348, 1993. Prelim version STOC '91.

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[Knu69] D. E. Knuth. *Art of Computer Programming, Volume II*. Addison Wesley, 1969. Current edition: 1997.

[Knu73] D. E. Knuth. *The Art of computer programming, Vol. 3 : Sorting and Searching*. Series in Computer Science and Information Processing. Addison-Wesley, 1973. Current edition: 1997.

[Koi97] P. Koiran. Randomized and deterministic algorithms for the dimension of algebraic varieties. In *FOCS*, pages 36–45. IEEE, 1997.

[Koz97] D. C. Kozen. *Automata and Computability*. Springer-Verlag, 1997.

[Koz06] D. C. Kozen. *Theory of Computation*. Springer, 2006.

[KPS85] R. M. Karp, N. Pippenger, and M. Sipser. A time randomness tradeoff. In *AMS Conf. on Probabilistic Computational Complexity*, 1985. This paper gives the first example of deterministic amplification using expander graphs.

[KR81] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, Mar. 1987. Prelim version report TR-31-81, Harvard University, 1981.

[KR08] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[Kra94] J. Krajíček. Lower bounds to the size of constant-depth propositional proofs. *J. Symbolic Logic*, 59(1):73–86, 1994.

[Kra95] J. Krajíček. *Bounded arithmetic, propositional logic and complexity theory*. Cambridge University Press, 1995.

[Kra97] J. Krajíček. Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *J. Symbolic Logic*, 62(2):457–486, 1997.

[KRW95] M. Karchmer, R. Raz, and A. Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational Complexity*, 5(3/4):191–204, 1995.

[KS99] A. R. Klivans and R. A. Servedio. Boosting and hard-core set construction. *Machine Learning*, 51(3):217–238, 2003. Prelim version FOCS' 99.

[KS06] G. Kalai and S. Safra. Threshold phenomena and influence. In C. M. A.G. Percus, G. Istrate, editor, *Computational Complexity and Statistical Physics*. Oxford University Press, 2006.

[KSS83] J. Kahn, M. E. Saks, and D. Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984. Prelim version FOCS' 83.

[KT06] J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson Studium, 2006.

[Kuc95] L. Kucera. Expected complexity of graph partitioning problems. *Discrete Applied Mathematics*, 57(2-3):193–212, 1995.

[KUW85] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6:35–48, 1986. Prelim version STOC '85.

[KV89] M. Kearns and L. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994. Prelim version STOC '89.

[KV94] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[KV05] S. Khot and N. K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1. In *FOCS*, pages 53–62. IEEE, 2005.

[KVS02] A. Y. Kitaev, M. Vyalyi, and A. Shen. *Classical and Quantum computation*. AMS Press, 2002.

[KW88] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM Journal on Discrete Mathematics*, 3(2):255–265, May 1990. Prelim version STOC '88.

[Lad75] R. E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, Jan. 1975.

[Lauer] C. Lautemann. BPP and the polynomial hierarchy. *Inf. Process. Lett.*, 17(4):215–217, 1983, November.

[Lea05] D. Leavitt. *The man who knew too much: Alan Turing and the invention of the computer*. Great discoveries. W. W. Norton & Co., 2005.

[Lei91] F. T. Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. Morgan Kaufmann, 1991.

[Lev73] L. A. Levin. Universal sequential search problems. *PINFTRANS: Problems of Information Transmission (translated from Problemy Peredachi Informatsii (Russian))*, 9, 1973.

[Lev86] L. A. Levin. Average case complete problems. *SIAM J. Comput*, 15(1):285–286, Feb. 1986.

[Lev87] L. A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992, October. Prelim version FOCS '90.

[Lip90] R. J. Lipton. Efficient checking of computations. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *lncs*, pages 207–215. Springer, 22–24 Feb. 1990.

[Lip91] R. J. Lipton. New directions in testing. In *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. American Mathematics Society, 1991.

[Liv06] N. Livne. All natural NPC problems have average-case complete versions. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 2006.

[LLKS85] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley, 1985.

[LLMP90] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. In *STOC*, pages 564–572. ACM, 14–16 May 1990.

[Llo06] S. Lloyd. *Programming the universe: A quantum computer scientist takes on the cosmos*. Knopf, 2006.

[Lov78] L. Lovàsz. Kneser's conjecture, chromatic number, and homotopy. *J. Combin. Theory Ser. A*, 1978.

[Lov79] L. Lovàsz. On determinants, matchings, and random algorithms. In L. Budach, editor, *Fundamentals of Computation Theory FCT '79*, pages 565–574. Akademie-Verlag, 1979.

[Lov07] L. Lovász. *Combinatorial problems and exercises*. AMS Chelsea Publishing, Providence, RI, second edition, 2007. Corrected reprint of 1993 second edition.

[LPS86] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988. Prelim version STOC' 86.

[LRVW03] C.-J. Lu, O. Reingold, S. Vadhan, and W. Wigderson. Extractors: optimal up to constant factors. In *STOC*, pages 602–611. ACM, 2003.

[LS88] L. Lovász and M. E. Saks. Communication complexity and combinatorial lattice theory. *J. Comput. Syst. Sci.*, 47(2):322–349, 1993. Prelim version FOCS' 88.

[LS91] D. Lapidot and A. Shamir. Fully parallelized multi prover protocols for NEXP-time (extended abstract). In *FOCS*, pages 13–18. IEEE, 1991.

[LS07] T. Lee and A. Shraibman. Disjointness is hard in the multi-party number on the forehead model, Dec. 27 2007. Comment: 21 pages.

[Lup58] O. Lupanov. The synthesis of contact circuits. *Dokl. Akad. Nauk SSSR*, 119:23–26, 1958.

[LW06] M. Luby and A. Wigderson. Pairwise independence and derandomization. *Found. Trends Theor. Comput. Sci.*, 1(4):237–301, 2006.

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[LY02] L. Lovász and N. E. Young. Lecture notes on evasiveness of graph properties. *CoRR*, cs.CC/0205031, 2002. informal publication.

[Maa84] W. Maass. Quadratic lower bounds for deterministic and nondeterministic one-tape turing machines. In *STOC*, pages 401–408. ACM, 1984.

[Mah80] S. R. Mahaney. Sparse complete sets of NP: solution of a conjecture of berman and hartmanis. *J. Comput. Syst. Sci.*, 25(2):130–143, 1982, October. Prelim version FOCS '80.

[Mar73] G. A. Margulis. Explicit constructions of concentrators. *Probl. Peredachi Inf.*, 9(4):71–80, 1973. Translation in: Problems of Information Transmission 10, Plenum, N.Y., 1975.

[Mar88] G. A. Margulis. Explicit group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24(1):39–46, 1988.

[Mat76] D. W. Matula. The largest clique size in a random graph. Technical report, Dept. of Computer Science, Southern Methodist University, 1976.

[Mat02] J. Matoušek. *Lectures on Discrete Geometry*. Springer, 2002. Online updated chapters available on `http://kam.mff.cuni.cz/~matousek/dg.html`.

[MG02] D. Micciancio and S. S. Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*. Kluwer Academic Publishers, 2002.

[MOO05] E. Mossel, R. O'Donnell, and K. Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. In *FOCS*, pages 21–30. IEEE, 2005.

[Mor73] J. Morgenstern. Note on a lower bound on the linear complexity of the fast fourier transform. *J. ACM*, 20(2):305–306, 1973.

[MR95] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[MR00] R. Martin and D. Randall. Sampling adsorbing staircase walks using a new markov chain decomposition method. In *FOCS*, pages 492–502. IEEE, Nov. 12–14 2000.

[MS72] A. R. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *FOCS*, pages 125–129. IEEE, 1972.

[MS82] K. Mehlhorn and E. M. Schmidt. Las Vegas is better than determinism in VLSI and distributed computing (extended abstract). In *STOC*, pages 330–337. ACM, 5–7 May 1982.

[MU05] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[Mul54] D. Muller. Application of boolean algebra to switching circuit design and to error detection. *IRE Transactions on Electronic Computation*, EC-3:6–12, 1954.

[MVV87] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[MZ04] J. Matoušek and G. M. Ziegler. Topological lower bounds for the chromatic number: a hierarchy. *Jahresber. Deutsch. Math.-Verein.*, 106(2):71–90, 2004.

[NC00] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[Nil04] A. Nilli. Tight estimates for eigenvalues of regular graphs. *Electr. J. Comb*, 11(1), 2004.

[Nis89] N. Nisan. CREW PRAMs and decision trees. *SIAM J. Comput*, 20(6):999–1007, Dec. 1991. Prelim version STOC' 89.

[Nis90] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. Prelim version STOC '90.

[NS92] N. Nisan and M. Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, 1994. Prelim version STOC' 92.

[NSW92] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(log^{1.5}n)$ space. In *FOCS*, pages 24–29. IEEE, Oct. 1992.

[NW88] N. Nisan and A. Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, Oct. 1994. Prelim version FOCS' 88.

[NW94] N. Nisan and A. Wigderson. On rank vs. communication complexity. *Combinatorica*, 15(4):557–565, 1995. Prelim version FOCS' 94.

[NZ93] N. Nisan and D. Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996. Prelim version STOC '93.

[O'D04] O'Donnell. Hardness amplification within NP. *J. Comput. Syst. Sci.*, 69, 2004.

[Ost54] A. M. Ostrowski. *Studies in Mathematics and Mechanics*, chapter On two problems in abstract algebra connected with Horner's rule. Academic, 1954.

[Pap85] C. H. Papadimitriou. Games against nature. *J. Comput. System Sci.*, 31(2):288–301, 1985.

[Pap90] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994. Prelim version FOCS '90.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Mass., 1994.

[Pen89] R. Penrose. *The Emperor's New Mind*. Oxford University Press, 1989.

[Pen90] R. Penrose. The Emperors New Mind. *Bull. Amer. Math. Soc. 23 (1990), 606-616.*, 1990.

[Pet60] W. W. Peterson. Encoding and error-correction procedures for bose-chaudhuri codes. *IEEE Transaction on Information Theory*, 6:459–470, 1960.

[Pin73] M. S. Pinsker. On the complexity of a concentrator. *Proc. 7th int. teletraffic cong.*, 1973.

[Pol65] S. L. Pollack. Conversion of limited-entry decision tables to computer programs. *Commun. ACM*, 8(11):677–682, 1965.

[Pra75] V. R. Pratt. Every prime has a succinct certificate. *SIAM J. Comput*, 4:214–220, 1975.

[Pre97] J. Preskill. Fault tolerant quantum computation. *Arxive e-print*, 1997. `arXiv:quant-ph/9712048v1`.

[Pre98] J. Preskill. Reliable quantum computers. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 454(1969):385–410, 1998.

[PS82] C. H. Papadimitriou and M. Sipser. Communication complexity. *J. Comput. Syst. Sci.*, 28(2):260–269, Apr. 1984. Prelim version STOC '82.

[Pud97] P. Pudlák. Lower bounds for resolution and cutting planes proofs and monotone computations. *J. Symbolic Logic*, 62(3):981–998, 1997.

[PV05] F. Parvaresh and A. Vardy. Correcting errors beyond the guruswami-sudan radius in polynomial time. In *FOCS*, pages 285–294. IEEE, 2005.

[PV06] C. Papadimitriou and U. Vazirani. Lecture notes for CS70: Discrete mathematics for computer science, 2006. Available from the authors' home pages.

[PY82] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *J. Comput. Syst. Sci.*, 28(2):244–259, 1982.

[PY88] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, Dec. 1991. Prelim version STOC 1988.

[Rab72] M. O. Rabin. Proving simultaneous positivity of linear forms. *J. Comput. Syst. Sci.*, 1972.

[Rab79] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology, Jan. 1979.

[Rab80] M. O. Rabin. A probabilistic algorithm for testing primality. *J. Number Theory*, 12, 1980.

[Raz85a] A. A. Razborov. Lower bounds on the monotone complexity of some boolean functions. *Doklady Akademii Nauk. SSSR*, 281(4):798–801, 1985. Translation in Soviet Math. Doklady 31, 354–357.

[Raz85b] A. A. Razborov. A lower bound on the monotone network complexity of the logical permanent. *Matematicheskie Zametki*, 37(6):887–900, 1985. In Russian, English translation in Mathematical Notes of the Academy of Sciences of the USSR 37:6 485-493.

[Raz87] A. A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *MATHNASUSSR: Mathematical Notes of the Academy of Sciences of the USSR*, 41, 1987.

[Raz89] A. A. Razborov. On the method of approximations. In *STOC*, pages 167–176. ACM, 15–17 1989.

[Raz90] A. A. Razborov. On the distributed complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 14 Dec. 1992. Prelim version ICALP '90.

[Raz92] A. A. Razborov. On submodular complexity measures. In *Boolean Function Complexity, (M. Paterson, Ed.)*, pages 76–83. London Mathematical Society Lecture Note Series 169, Cambridge University Press, 1992.

[Raz95a] A. A. Razborov. Unprovability of lower bounds on the circuit size in certain fragments of bounded arithmetic. *Izvestiya of the RAN*, 59(1):201–224, 1995.

[Raz95b] R. Raz. A parallel repetition theorem. *SIAM J. Comput*, 27(3):763–803, June 1998. Prelim version STOC '95.

[Raz98] A. A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7:291–324, 1998.

[Raz00] R. Raz. The BNS-chung criterion for multi-party communication complexity. *Computational Complexity*, 9(2):113–122, 2000.

[Raz01] A. A. Razborov. Improved resolution lower bounds for the weak pigeonhole principle. Technical Report TR01-055, Electronic Colloquium on Computational Complexity, 2001.

[Raz02] R. Raz. On the complexity of matrix product. *SIAM J. Comput*, 32(5):1356–1369, 2003. Prelim version STOC' 02.

[Raz03a] A. A. Razborov. Resolution lower bounds for the weak functional pigeonhole principle. *Theoretical Computer Science*, 303(1):233–243, 2003.

[Raz03b] A. A. Razborov. Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution, 2003.

[Raz04a] R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. In *STOC*. ACM, 2004.

[Raz04b] A. A. Razborov. Resolution lower bounds for perfect matching principles. *J. Comput. Syst. Sci.*, 69(1):3–27, 2004.

[Raz04c] A. A. Razborov. Feasible proofs and computations: Partnership and fusion. In *Proceedings of the* 31*st International Colloquium, Lecture Notes in Computer Science,* 3142, pages 8–14. Springer-Verlag, 2004. Also appeared in Proceedings of the 19th LICS conference.

[Ree54] I. S. Reed. A class of multiple error-correcting codes and the decoding scheme. *IRE Transaction on Information Theory*, PGIT-4:38–49, 1954.

[Reg06] O. Regev. Lattice-based cryptography. In *Proceedings of 26th CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 131–141. Springer, 2006.

[Rei05] O. Reingold. Undirected ST-connectivity in log-space. In *STOC*, pages 376–385. ACM, 2005.

[Rog87] H. Rogers Jr. *Theory of recursive functions and effective computability.* MIT Press Cambridge, MA, USA, 1987.

[Ros06] K. H. Rosen. *Discrete Mathematics and Its Applications.* McGraw-Hill, 2006.

[Rot93] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2):273–302, 1996. Prelim version IJCAI '93.

[RR94] A. A. Razborov and S. Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, Aug. 1997. Prelim version STOC '94.

[RR99] R. Raz and O. Reingold. On recycling the randomness of states in space bounded computation. In *STOC*, pages 159–168. ACM, 1999.

[RS60] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. Soc. Industrial Applied Math.*, 8(2):300–304, June 1960.

[RS91] C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology—CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer-Verlag, 1992, 11–15 Aug. 1991.

[RS92] R. Rubinfeld and M. Sudan. Self-testing polynomial functions efficiently and over rational domains. In *SODA*, pages 23–32, 1992.

[RS93] R. Raz and B. Spieker. On the "log rank"-conjecture in communication complexity. *Combinatorica*, 15, 1995. Prelim version FOCS '93.

[RS95] A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998. Prelim version MIT Tech report, 1995.

[RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test. In *STOC*, pages 475–484. ACM, 1997.

[RSA78] R. L. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RTS97] J. Radhakrishnan and A. Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, 2000. Prelim version FOCS '97.

[RTV06] O. Reingold, L. Trevisan, and S. Vadhan. Pseudorandom walks on regular digraphs and the RL vs. L problem. In *STOC*, pages 457–466. ACM, 2006.

[Rub90] R. Rubinfeld. *A Mathematical Theory of Self-Checking, Self-Testing and Self-Correcting Programs*. PhD thesis, U.C. Berkeley, 1990.

[RV76] R. L. Rivest and J. Vuillemin. On recognizing graph properties from adjacency matrices. *Theor. Comput. Sci*, 3(3):371–384, 1976.

[RV05] E. Rozenman and S. Vadhan. Derandomized squaring of graphs. In *RANDOM: International Workshop on Randomization and Approximation Techniques in Computer Science*. LNCS, 2005.

[RVW00] O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *FOCS*, pages 3–13. IEEE, 2000.

[RW93] A. A. Razborov and A. Wigderson. $n^{\Omega(\log n)}$ Lower bounds on the size of depth-3 threshold circuits with AND gates at the bottom. *Information Processing Letters*, 45(6):303–307, 16 Apr. 1993.

[San07] R. Santhanam. Circuit lower bounds for Merlin-Arthur classes. *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 275–283, 2007.

[Sav70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4, 1970.

[Sav72] J. E. Savage. Computational work and time on finite machines. *J. ACM*, 19(4):660–674, Oct. 1972.

[SBR02] M. V. Sapir, J.-C. Birget, and E. Rips. Isoperimetric and isodiametric functions of groups. *Ann. of Math. (2)*, 156(2):345–466, 2002.

[Sch37] A. Scholz. Aufgabe 253. *Jahresber. DMV*, 1937.

[Sch44] E. Schroedinger. *What is Life?* Cambridge University Press, 1944.

[Sch96] M. Schaefer. Deciding the Vapnik-Cervonenkis dimension is Σ_3^p-complete. *J. Comput. Syst. Sci.*, 58(2):177–182, 1999. Prelim version CCC '96.

[SG76] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976.

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[Sha49a] C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949.

[Sha49b] C. E. Shannon. Communication theory of secrecy systems. *Bell Systen Technicl Journal*, 28:656–715, Oct. 1949.

[Sha79] A. Shamir. Factoring numbers in O(log n) arithmetic steps. *Inf. Process. Lett.*, 8(1):28–31, 1979.

[Sha81] A. Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. on Computer Sys.*, 1(1):38–44, 1983. Prelim version presented at Crypto'81 and ICALP'81.

[Sha90] A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, Oct. 1992. Prelim version FOCS' 90.

[Sha02] R. Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.

[She92] A. Shen. IP = PSPACE: Simplified proof. *JACM*, 1992.

[She07] A. A. Sherstov. The pattern matrix method for lower bounds on quantum communication. Technical Report CS-TR-07-46, The University of Texas at Austin, Department of Computer Sciences, Sept. 6 2007. Thu, 3 Jan 108 21:31:32 GMT.

[Shi03] Y. Shi. Both toffoli and controlled-NOT need little help to universal quantum computing. In *Quantum Information and Computation*, volume 3. Rinton Press, 2003.

[SHL65] R. E. Stearns, J. Hartmanis, and P. M. Lewis. Hierarchies of memory limited computations. In *FOCS*, pages 179–190. IEEE, 1965.

[Sho95] P. W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):2493–2496, 1995.

[Sho97] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput*, 26(5):1484–1509, Oct. 1997.

[Sho05] V. Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2005.

[Sim94] D. R. Simon. On the power of quantum computation. *SIAM J. Comput*, 26(5):1474–1483, Oct. 1997. Prelim version FOCS' 94.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *STOC*, pages 330–335. ACM, 25–27 Apr. 1983.

[Sip86] M. Sipser. Expanders, randomness, or time versus space. *J. Comput. Syst. Sci.*, 36:379–383, 1988. Prelim version CSCT '86.

[Sip92] M. Sipser. The history and status of the P versus NP question. In *STOC*, pages 603–618. ACM, 1992.

[Sip96] M. Sipser. *Introduction to the Theory of Computation*. PWS, 1996.

[SJ88] A. Sinclair and M. Jerrum. Approximative counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989, July. Prelim version International Workshop on Graph-Theoretic Concepts in Computer Science '88.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *STOC*, pages 1–9. ACM, 1973.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *STOC*, pages 77–82. ACM, 1987.

[Sni81] M. Snir. Lower bounds on probabilistic linear decision trees. *Theor. Comput. Sci.*, 38(1):69–82, 1985. Prelim version ICALP '81.

[SS71] A. Schoenhage and V. Strassen. Multiplikation groer zahlen. *Computing*, 1971.

[SS77] R. Solovay and V. Strassen. A fast Monte Carlo test for primality. *SIAM J. Comput*, 6(1):84–85, 1977.

[ST01] D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004. Prelim version STOC' 01.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[Str72] V. Strassen. Berechnung und programm. I. *Act. Inf.*, 1:320–335, 1972. In German.

[Str73] V. Strassen. Vermeidung von divisionen. *J. reine angew. Math.*, 264:184–202, 1973. In German.

[STV99] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001. Prelim version STOC '99.

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005. Prelim version FOCS' 01.

[SU02a] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: Part I. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 33, Sept. 2002.

[SU02b] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: Part II. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 33, Dec. 2002.

[Sud96] M. Sudan. Decoding of reed solomon codes beyond the error-correction bound. *J. Complexity*, 13(1):180–193, 1997. Prelim version FOCS '96.

[Sud01] M. Sudan. Coding theory: tutorial & survey. In *FOCS*, pages 36–53. IEEE, 2001.

[SV84] M. Santha and U. V. Vazirani. Generating quasi-random sequences from semi-random sources. *J. Comput. Syst. Sci.*, 33(1):75–87, 1986, August. Prelim version FOCS '84.

[SV85] S. Skyum and L. G. Valiant. A complexity theory based on boolean algebra. *J. ACM*, 32(2):484–502, 1985.

[SW86] M. Saks and A. Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *FOCS*, pages 29–38. IEEE, Oct. 1986.

[SZ94] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM J. Comput*, 28(4):1433–1459, Aug. 1999. Prelim version FOCS 1994.

[SZ95] M. Saks and S. Zhou. $BP_HSPACE(S) \subseteq DSPACE(S^{3/2})$. *J. Comput. Syst. Sci.*, 58(2):376–403, 1999. Prelim version FOCS' 95.

[Sze76] E. Szemerédi. Regular partitions of graphs. *Problèmes combinatoires et théorie des graphes, Orsay*, 1976.

[Sze87] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Bulletin of the European Association for Theoretical Computer Science*, 33:96–100, Oct. 1987. Technical Contributions.

[Tan84] R. M. Tanner. Explicit construction of concentrators from generalized n-gons. *SIAM J. Algebraic Discrete Methods*, 5:287–293, 1984.

[Tar88] É. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput*, 20(5):865–877, 1991.

[Tra84] B. A. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force search) algorithms. *Annals of the History of Computing*, 6(4):384–400, Oct./Dec. 1984. Also contains a good translation of [Lev73].

[Tre99] L. Trevisan. Extractors and pseudorandom generators. *J. ACM*, 48(4):860–879, July 2001. Prelim version STOC '99.

[Tre05] L. Trevisan. Inapproximability of combinatorial optimization problems. In V. Paschos, editor, *Optimisation Combinatiore*, volume 2. Hermes, 2005. English version available from author's web page.

[Tri05] V. Trifonov. An O(log n log log n) space algorithm for undirected st-connectivity. In *STOC*, pages 626–633. ACM, 2005.

[TS96] A. Ta-Shma. On extracting randomness from weak random sources. In *STOC*, pages 276–285. ACM, May 1996.

[Tur36] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings, London Mathematical Society,*, pages 230–265, 1936. Published as Proceedings, London Mathematical Society,, volume 2, number 42.

[TV02] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity, 2002. Proceedings. 17th IEEE Annual Conference on*, pages 103–112, 2002.

[TV06] T. Tao and V. H. Vu. *Additive Combinatorics*. Cambridge University Press, 2006.

[Uma01] C. Umans. The minimum equivalent DNF problem and shortest implicants. *J. Comput. Syst. Sci.*, 63, 2001.

[Uma03] C. Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.

[Urq87] A. Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.

[Vad99] S. Vadhan. *A Study of Statistical Zero-Knowledge Proofs*. PhD thesis, Massachusetts Institute of Technology, Aug. 1999. Updated version to be published by Springer-Verlag (est. 2008).

[Vad00] S. Vadhan. On transformation of interactive proofs that preserve the prover's complexity. In *STOC*, pages 200–207. ACM, 2000.

[Vad07] S. Vadhan. The unified theory of pseudorandomness. *SIGACT News*, 38(2), 2007.

[Val75a] L. G. Valiant. Graph-theoretic properties in computational complexity. *J. Comput. Syst. Sci.*, 13(3):278–285, Dec. 1976. Prelim version STOC' 75.

[Val75b] L. G. Valiant. On non-linear lower bounds in computational complexity. In *STOC*, pages 45–53. ACM, 1975.

[Val79a] L. G. Valiant. Completeness classes in algebra. In *STOC*, pages 249–261. ACM, 1979.

[Val79b] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[Val79c] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput*, 8(3):410–421, 1979.

[Val84] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[Vaz01] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[vDMV01] W. van Dam, M. Mosca, and U. Vazirani. How powerful is adiabatic quantum computation? In *FOCS*, pages 279–287. IEEE, Oct. 14–17 2001.

[Ver94] O. Verbitsky. Towards the parallel repetition conjecture. In *Structure in Complexity Theory Conference*, pages 304–307, 1994.

[VG99] J. Von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[vN45] J. von Neumann. First draft of a report on the EDVAC. Report for the u.s. army ordinance department, University of Pensylvania, 1945. Reprinted in part in Brian Randell, ed. The Origins of Digital Computers: Selected Papers, Springer Verlag, 1982.

[vN51] J. von Neumann. Various techniques used in connection with random digits. *Applied Math Series*, 12:36–38, 1951.

[VSBR81] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput*, 12(4):641–644, Nov. 1983. Prelim version Mathematical Foundations of CS '81.

[VV85] U. V. Vazirani and V. V. Vazirani. Random polynomial time is equal to slightly-random polynomial time. In *FOCS*, pages 417–428. IEEE, 1985.

[VV86] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(1):85–93, 1986.

[vzG88] J. von zur Gathen. Algebraic complexity theory. *Annual Reviews Computer Science*, 1988.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[Wat03] J. Watrous. PSPACE has constant-round quantum interactive proof systems. *Theor. Comput. Sci*, 292(3):575–588, 2003.

[Weg87] I. Wegener. *The complexity of Boolean functions*. Wiley-Teubner Series in Computer Science, 1987. Online version available from `http://eccc.hpi-web.de/eccc-local/ECCC-Books/wegener_book_readme.html`.

[Wel93] D. J. A. Welsh. *Complexity: knots, colourings and counting*. Cambridge University Press, 1993.

[Wig06] A. Wigderson. P, NP and mathematics - a computational complexity perspectivy. In *Proceedings of ICM' 06*, 2006.

[Wil05] R. Williams. Better time-space lower bounds for SAT and related problems. In *IEEE Conference on Computational Complexity*, pages 40–49. IEEE, 2005.

[WX05] A. Wigderson and D. Xiao. A randomness-efficient sampler for matrix-valued functions and applications. In *FOCS*, pages 397–406. IEEE, 2005. See also correction in ECCC TR05-107 Revision 01.

[WZ82] W. K. Wooters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299:802f, Oct. 1982.

[Yam97] P. Yam. Bringing schroedinger's cat back to life. *Scientific American*, pages 124–129, June 1997.

[Yan88] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991. Prelim version STOC '88.

[Yao79] A. C. C. Yao. Some complexity questions related to distributive computing(prelim report). In *STOC*, pages 209–213. ACM, 1979.

[Yao82a] A. C. C. Yao. Theory and applications of trapdoor functions. In *FOCS*, pages 80–91. IEEE, 3–5 Nov. 1982.

[Yao82b] A. C. C. Yao. Protocols for secure computations. In *FOCS*, pages 160–164. IEEE, 3–5 Nov. 1982.

[Yao85] A. C. C. Yao. Separating the polynomial-time hierarchy by oracles. In *FOCS*, pages 1–10. IEEE, 1985.

[Yao87] A. C. C. Yao. Lower bounds to randomized algorithms for graph properties. *J. Comput. Syst. Sci.*, 42(3):267–287, 1991. Prelim version FOCS' 87.

[Yao90] A. C. C. Yao. On ACC and threshold circuits. In *FOCS*, volume II, pages 619–627. IEEE, 22–24 Oct. 1990.

[Yao93] A. C. C. Yao. Quantum circuit complexity. In *FOCS*, pages 352–361. IEEE, 1993.

[Yao94] A. C. C. Yao. Decision tree complexity and betti numbers. In *STOC*, pages 615–624. ACM, 1994.

[Zak83] S. Zak. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, Oct. 1983.

[Zuc90] D. Zuckerman. General weak random sources. In *FOCS*, volume II, pages 534–543. IEEE, 22–24 Oct. 1990.