<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Background: Before we start</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Decidable and Undecidable Languages:</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Background: Basic Analysis</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Some basic inequalities</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Measure spaces</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Probability Spaces</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Normed spaces</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Hilbert Spaces</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>The $L_p$ spaces</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Time complexity</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Our computational model</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Time complexity</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Time Hierarchy theorems</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>Polynomial Hierarchy</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>Space complexity</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Time and Space</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Logarithmic Space L and NL</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>Savitch’s Theorem</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Savitch’s theorem</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>NL versus coNL: Immerman-Szelepcsényi’s Theorem</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>NL versus coNL: Immerman-Szelepcsényi’s Theorem</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>Ladner’s Theorem: NP-completeness</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Warm-up: Proof assuming the Exponential Time Hypothesis</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>The full proof of Ladner’s Theorem</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>37</td>
</tr>
<tr>
<td>9</td>
<td>Oracles and relativization</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>Relativization</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Relativization as a proof-barrier</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>42</td>
</tr>
<tr>
<td>10</td>
<td>Randomized Complexity Classes</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Probabilistic Turing machines</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Polynomial time randomized complexity classes</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>The complexity class RP</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>The complexity class ZPP</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>The complexity class BPP</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>The complexity class PP</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>53</td>
</tr>
<tr>
<td>11</td>
<td>Non-uniform complexity: Circuits</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>What is a Boolean circuit?</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>The class P/Poly</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Karp-Lipton Theorem</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>Shannon-Muller Lower bound</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>Adleman’s theorem and advice: $BPP \subseteq P/Poly$</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>59</td>
</tr>
<tr>
<td>12</td>
<td>$AC^0$: Bounded Depth Alternating Circuits</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>Lower-bounds for AC circuits</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>PARITY and MAJ are not in $AC^0$</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>69</td>
</tr>
<tr>
<td>13</td>
<td>$AC^0$ with parity gates: Razborov-Smolensky</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>The class $AC^0[\oplus]$: Algebraic techniques</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>Razborov-Smolensky over $F_2$</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>Concluding Remarks</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>76</td>
</tr>
<tr>
<td>14</td>
<td>Razborov’s monotone circuit lower-bound</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>Approximating the monotone circuit</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>The preliminary lemmas</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>Approximation by a small number of cliques</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Concluding remarks</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>83</td>
</tr>
<tr>
<td>15</td>
<td>Razborov-Rudich: Natural Proof Barrier</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>What are natural proofs?</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>The Razborov-Rudich Theorem</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>Preparing for the proof: Pseudo-random Generators</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>The proof of Razborov-Rudich Theorem</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>Concluding Remarks</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>Exercises</td>
<td>89</td>
</tr>
<tr>
<td>16</td>
<td>Fourier analysis and Polynomial Representations</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>Polynomial representations of $f : {-1,1}^n \rightarrow \mathbb{R}$</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>Fourier analysis of Finite Abelian Groups</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>Basic Fourier Theory</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>Infinite Ableian groups and beyond</td>
<td>100</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Concluding remarks:</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>Exercises</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td><strong>17 Learning and $\mathit{AC^0}$ circuits</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAC learning from uniform samples</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>Decision Trees</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>Linial-Mansour-Nisan</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>Mansour-Nisan conjecture</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>Exercises</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td><strong>18 Communication complexity</strong></td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>Monochromatic rectangles, and Fooling sets</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>Rectangle Size Bounds</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>Rank Lower-Bound</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>Non-deterministic Communication complexity</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>Exercises</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td><strong>19 Randomized Communication complexity</strong></td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>Yao’s Min-Max Theorem</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>Exercises</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td><strong>20 Discrepancy and Sign-rank</strong></td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>Discrepancy</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>Other formulations of discrepancy</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>Unbounded-Error model</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>Sign-rank</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>Exercises</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td><strong>21 Machine Learning: Dimension and Margin</strong></td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>Sign-rank in Machine Learning: Dimension</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>Margin and Discrepancy</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>The Matrix Theory Background</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>Forster’s Sign-rank Lower-bound</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>Exercises</td>
<td>127</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Background: Before we start

These lectures are designed for an advanced course on complexity theory. We will assume that the reader is familiar with the basics of the theory of computation: Turing Machines, Non-determinism, Decidability, Undecidability, Turing reductions. We will also assume that the reader is familiar with the basic complexity theory: the complexity classes \( P \), \( NP \), \( coNP \), the definition of \( NP \) through efficient verifiers, the notion of \( NP \)-completeness, and the Cook-Levin theorem. Similar to any theoretical computer science course, we shall use the asymptotic notations of \( O(\cdot) \), \( \Theta(\cdot) \), \( \Omega(\cdot) \), \( o(\cdot) \), \( \omega(\cdot) \).

In this short lecture, we quickly remind the reader of some of these concepts.

Decidable and Undecidable Languages:

An alphabet \( \Sigma \) is just a finite set, where the elements are interpreted as “symbols”. In this course, almost always \( \Sigma = \{0, 1\} \). We denote by \( \Sigma^* \) the set of all finite strings constructed from the symbols in \( \Sigma \). For example, if \( \Sigma = \{0, 1\} \), then \( \Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\} \). Here \( \varepsilon \) denotes the empty string of length zero. We emphasis that all the elements in \( \Sigma^* \) are of finite length.

Every subset \( L \subseteq \Sigma^* \) is called a language. Languages correspond to decision problems, i.e., YES/NO problems in a natural way: Given an input \( x \), we would like to know whether \( x \in L \) or \( x \notin L \). A Turing Machine decides a language \( L \) if it terminates on all inputs and accepts \( x \) if and only if \( x \in L \). A language is \emph{decidable} if a Turning Machine decides it. The Turing Machines that always halt are called \emph{algorithms}. Since the set of all languages is uncountable, and the set of all Turing Machines over any fixed alphabet is countable, there exist languages that are \emph{undecidable}.

Recall that \( P \) is the class of the languages that are decidable by polynomial-time algorithms, and \( NP \) is the class of the problems that are decidable by non-deterministic polynomial-time algorithms. It is also useful to define \( NP \) through the notions of the verifiers. A \emph{verifier} for a language \( L \) is an algorithm \( V \) that takes a pair \((x, y)\) as input and has the property that

\[
 x \in L \iff \exists y, V(x, y) = \text{Accept}.
\]

Here we can think of \( y \) as a certificate/proof for \( x \in L \): If \( x \in L \), then there is a certificate for it, and our algorithm can verify it. However if \( x \notin L \), then such a certificate should not exist, and \( V \) should reject \((x, y)\) for every \( y \). A verifier is \emph{efficient} if its running time is bounded by a polynomial \( p \) in the length of \( x \). Note that this is stronger than saying that \( V \) runs in polynomial time because the input of \( V \) is \((x, y)\), but we bound its running time by \( p(|x|) \) rather than \( p(|(x, y)|) \).

It is easy to see that \( NP \) is the set of languages with efficient verifiers. A canonical problem in \( NP \) is the \emph{SAT} problem, which, in a certain sense, is as hard as any other problem in this class. First, we define the notion of a polynomial-time reduction.

\begin{definition}
A language \( A \) is \emph{polynomial time mapping reducible}, or \emph{simply polynomial time reducible} to a language \( B \), written as \( A \leq_P B \), if a polynomial time computable function \( f : \Sigma^* \to \Sigma^* \) exists, where for every \( w \),

\[
 w \in A \iff f(w) \in B.
\]

A \emph{Boolean formula} over variables \( x_1, \ldots, x_n \) is a formula consisting of variables and logical operators \( \land, \lor, \neg \) corresponding to AND, OR, and negation, respectively. For example

\[
 \phi(x_1, x_2, x_3) := (x_1 \land x_2) \lor (x_1 \land x_3) \lor (x_2 \land x_3)
\]

is a Boolean formula over the variables \( x_1, x_2, x_3 \), and one can easily check that \( \phi \) evaluates to \text{TRUE} if and only if majority of the variables \( x_1, x_2, x_3 \) are \text{TRUE}. We will identify \( 1 \) with \text{TRUE} and \( 0 \) with \text{FALSE}; for example \( \phi(1, 1, 0) = 1 \).
Definition 1.2 (Conjunctive Normal Form (CNF)). A Boolean formula over variables $x_1, \ldots, x_n$ is in **CNF form** if it is an AND of ORs of variables or their negations. We call formulas that are in CNF form, simply CNF formulas. The terms $x_i$ or $\neg x_i$ appearing in a formula are called **terms** or **litersals**. For a CNF formula

$$\phi = \bigwedge_{i=1}^{m} \left( \bigvee_{j=1}^{k_i} t_{ij} \right),$$

where $t_{ij}$ are terms, we call each $(\bigvee_{j=1}^{k_i} t_{ij})$ a **clause** of the formula.

For example

$$(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor x_3)$$

is a CNF formula with two clauses $(x_1 \lor \neg x_2 \lor x_3)$ and $(x_2 \lor x_3)$.

**Theorem 1.3** (Cook-Levin Theorem, Cook [Coo71], Levin [Lev73]). The language of all satisfiable CNF formulas

$$\text{SAT} = \{ \phi | \phi \text{ is a CNF, and there exists } y \text{ such that } \phi(y) = \text{TRUE} \},$$

is NP-complete. That is it belongs to NP and furthermore $X \leq_p \text{SAT}$ for every $X \in \text{NP}$.

It is easy to see that $\text{SAT}$ is in NP. To see this, note that a $y$ that satisfies $\phi(y) = \text{TRUE}$ can be used as a certificate to verify that $\phi$ is satisfiable. In other words, an efficient certifier $V$ takes the pair $(\phi, y)$, plugs $y$ into $\phi$, and checks to see if $\phi$ is satisfied. If it is, then it accepts the pair; otherwise, it rejects. Note that

$$\phi \in \text{SAT} \iff \exists y, V(\phi, y) = \text{Accept}.$$

The interesting and challenging part of the Cook-Levin theorem is the proof that every language in NP reduces to SAT. The proof is by encoding the computation of a Turing machine as a CNF.

**Big O notation**: Finally let us finish this lecture by recalling the asymptotic notations. Let $f, g : \mathbb{N} \to \mathbb{N}$. We say

- $f(n) = O(g(n))$ if there exists $n_0, c > 0$ such that $f(n) < cg(n)$ for all $n > n_0$.
- $f(n) = \Omega(g(n))$ if there exists $n_0, c > 0$ such that $f(n) > cg(n)$ for all $n > n_0$.
- $f(n) = \Theta(g(n))$ if there exists $n_0, c_1, c_2 > 0$ such that $c_1 g(n) < f(n) < c_2 g(n)$ for all $n > n_0$.
- $f(n) = o(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.
- $f(n) = \omega(g(n))$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$.

**Exercises**

**Exercise 1.1.** Where is the error in the following false proof that $n = O(1)$. The proof is by induction. The base case is obvious: $1 = O(1)$. Let us assume the induction hypothesis that $n = O(1)$. Then we have $n + 1 = O(1) + 1 = O(1)$, and hence we have established the induction hypothesis for $n + 1$.

**Exercise 1.2.** Prove that NP is the set of languages $L$ such that there exists an efficient (i.e. polytime) algorithm $M$, and a polynomial $p(n)$ such that $x \in L$ if and only if $M$ accepts $(x, y)$ for some $y$ with $|y| < p(n)$.

**Exercise 1.3.** Prove that $\text{2SAT} \in \text{P}$ where $\text{2SAT}$ is SAT restricted to CNF’s such that every clause is of size exactly 2.

**Exercise 1.4.** Prove that $\text{3SAT}$ is NP-complete by showing that $\text{SAT} \leq_p \text{3SAT}$.
Chapter 2

Background: Basic Analysis

The aim of this chapter is to introduce the necessary definitions, notations, and basic results from measure theory, probability theory, and functional analysis. We will not immediately need all these definitions and results, and the reader can skip this chapter, or some parts of it in the first reading.

We will only need these definitions in the finitary setting, but to provide a reference for the interested reader and put them in a broader context, we state them in the more general form.

Some basic inequalities

One of the most basic inequalities in analysis concerns the arithmetic mean and the geometric mean. It is sometimes called the AM-GM inequality.

**Theorem 2.1.** The geometric mean of $n$ non-negative reals is less than or equal to their arithmetic mean: If $a_1, \ldots, a_n$ are non-negative reals, then

$$\left(\prod_{i=1}^{n} a_i\right)^{1/n} \leq \frac{a_1 + \ldots + a_n}{n}.$$

In 1906 Jensen founded the theory of convex functions. This enabled him to prove a considerable extension of the AM-GM inequality. Recall that a subset $D$ of a real vector space is called convex if every convex linear combination of a pair of points of $D$ is in $D$. Equivalently, if $x, y \in D$, then $tx + (1-t)y \in D$ for every $t \in [0,1]$. Given a convex set $D$, a function $f : D \to \mathbb{R}$ is called convex if for every $t \in [0,1]$,

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

If the inequality is strict for every $t \in (0,1)$, then the function is called strictly convex.

Trivially $f$ is a convex function if and only if $\{ (x, y) \in D \times \mathbb{R} : y \geq f(x) \}$ is a convex set. Also note that $f : D \to \mathbb{R}$ is convex if and only if $f_{xy} : [x, y] \to \mathbb{R}$ defined as $f_{xy} : tx + (1-t)y \mapsto tf(x) + (1-t)f(y)$ is always convex. By Rolle’s theorem if $f_{xy}$ is twice differentiable, then this is equivalent to $f_{xy}'' \geq 0$.

A function $f : D \to \mathbb{R}$ is concave if $-f$ is convex. The following important inequality is often called Jensen’s inequality.

**Theorem 2.2.** If $f : D \to \mathbb{R}$ is a concave function, then for every $x_1, \ldots, x_n \in D$ and $t_1, \ldots, t_n \geq 0$ with $\sum_{i=1}^{n} t_i = 1$ we have

$$t_1f(x_1) + \ldots + t_nf(x_n) \leq f(t_1x_1 + \ldots + t_nx_n).$$

Furthermore if $f$ is strictly concave, then the equality holds if and only if all $x_i$ are equal.

The most frequently used inequalities in functional analysis are the Cauchy-Schwarz inequality, Hölder’s inequality, and Minkowski’s inequality.

**Theorem 2.3 (Cauchy-Schwarz).** If $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are complex numbers, then

$$\left| \sum_{i=1}^{n} x_iy_i \right| \leq \left( \sum_{i=1}^{n} |x_i|^2 \right)^{1/2} \left( \sum_{i=1}^{n} |y_i|^2 \right)^{1/2}.$$

Hölder’s inequality is an important generalization of the Cauchy-Schwarz inequality.
Theorem 2.4 (Hölder’s inequality). Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be complex numbers, and $p, q > 1$ be such that $\frac{1}{p} + \frac{1}{q} = 1$. Then

$$\left| \sum_{i=1}^{n} x_i y_i \right| \leq \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p} \left( \sum_{i=1}^{n} |y_i|^q \right)^{1/q}.$$ 

The numbers $p$ and $q$ appearing in Theorem 2.4 are called conjugate exponents. In fact 1 and $\infty$ are also called conjugate exponents, and Hölder’s inequality in this case becomes:

$$\left| \sum_{i=1}^{n} x_i y_i \right| \leq \left( \sum_{i=1}^{n} |x_i| \right) \left( \max_{i=1}^{n} |y_i| \right).$$

The next theorem is called Minkowski’s inequality.

Theorem 2.5 (Minkowski’s inequality). If $p > 1$ is a real number, and $x_1, \ldots, x_n$ are complex numbers, then

$$\left( \sum_{i=1}^{n} |x_i + y_i|^p \right)^{1/p} \leq \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p} + \left( \sum_{i=1}^{n} |y_i|^p \right)^{1/p}.$$ 

The case of $p = \infty$ of Minkowski’s inequality is the following:

$$\max_{i=1}^{n} |x_i + y_i| \leq \left( \max_{i=1}^{n} |x_i| \right) + \left( \max_{i=1}^{n} |y_i| \right).$$
Measure spaces

A $\sigma$-algebra (sometimes sigma-algebra) over a set $\Omega$ is a collection $\mathcal{F}$ of subsets of $\Omega$ that satisfies the following three properties:

- It includes $\emptyset$. That is $\emptyset \in \mathcal{F}$.
- It is closed under complementation. That is, if $A \in \mathcal{F}$, then the complement of $A$ also belongs to $\mathcal{F}$.
- It is closed under countable unions of its members. That is, if $A_1, A_2, \ldots$ belong to $\mathcal{F}$, then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$.

**Example 2.6.** Let $\Omega$ be an arbitrary set. Then the family consisting only of the empty set and the set $\Omega$ is called the minimal or trivial $\sigma$-algebra over $\Omega$. The power set of $\Omega$, denoted by $\mathcal{P}(\Omega)$, is the maximal $\sigma$-algebra over $\Omega$.

There is a natural partial order between $\sigma$-algebras over $\Omega$. For two $\sigma$-algebras $\mathcal{F}_1$ and $\mathcal{F}_2$ over $\Omega$, if $\mathcal{F}_1 \subseteq \mathcal{F}_2$ then we say that $\mathcal{F}_2$ is finer than $\mathcal{F}_1$, or that $\mathcal{F}_1$ is coarser than $\mathcal{F}_2$. Note that the trivial $\sigma$-algebra is the coarsest $\sigma$-algebra over $\Omega$, while the maximal $\sigma$-algebra is the finest $\sigma$-algebra over $\Omega$.

**Definition 2.7.** A measure space is a triple $(\Omega, \mathcal{F}, \mu)$ where $\mathcal{F}$ is a $\sigma$-algebra over $\Omega$ and the measure $\mu : \mathcal{F} \to [0, \infty) \cup \{\infty\}$ satisfies the following axioms:

- Null empty set: $\mu(\emptyset) = 0$.
- Countable additivity: if $\{E_i\}_{i \in I}$ is a countable set of pairwise disjoint sets in $\mathcal{F}$, then
  \[ \mu(\bigcup_{i \in I} E_i) = \sum_{i \in I} \mu(E_i). \]

The function $\mu$ is called a measure, and the elements of $\mathcal{F}$ are called measurable sets. If furthermore $\mu : \mathcal{F} \to [0, 1]$ and $\mu(\Omega) = 1$, then $(\Omega, \mathcal{F}, \mu)$ is called a probability measure.

**Example 2.8.** The counting measure on $\Omega$ is defined in the following way. The measure of a subset is taken to be the number of elements in the subset, if the subset is finite, and $\infty$ if the subset is infinite.

A measure space $M = (\Omega, \mathcal{F}, \mu)$ is called $\sigma$-finite, if $\Omega$ is a countable union of measurable sets of finite measure: That is $\Omega = \bigcup_{i=1}^{\infty} S_i$ with $S_i \in \mathcal{F}$, and $\mu(S_i) < \infty$.

Every measure space in this this course is assumed to be $\sigma$-finite.

For many natural measure spaces $M = (\Omega, \mathcal{F}, \mu)$, it is difficult to specify the elements of the $\sigma$-algebra $\mathcal{F}$. Instead one specifies an “algebra” of elements of $\Omega$ which generates $\mathcal{F}.$

**Definition 2.9.** For a set $\Omega$, a collection $\mathcal{A}$ of subsets of $\Omega$ is called an algebra if

- $\emptyset \in \mathcal{A}$.
- $A, B \in \mathcal{A}$, then $A \cup B \in \mathcal{A}$.
- $A, B \in \mathcal{A}$, then $A \setminus B \in \mathcal{A}$.

The minimal $\sigma$-algebra containing $\mathcal{A}$ is called the $\sigma$-algebra generated by $\mathcal{A}$.

**Example 2.10.** Let $\mathcal{A}$ be the set of all finite unions of (open, closed, or half-open) intervals in $\mathbb{R}$. Then $\mathcal{A}$ is an algebra over $\mathbb{R}$, but it is not a $\sigma$-algebra as it is not closed under countable union of its members.

Before proceeding, let us mention that $\mu : \mathcal{A} \to [0, \infty) \cup \{\infty\}$ is called a measure over an algebra $\mathcal{A}$ if for every finite set of $E_1, \ldots, E_m \in \mathcal{A}$, we have

$$\mu(\bigcup_{i=1}^{m} E_i) = \sum_{i=1}^{m} \mu(E_i).$$

The following theorem, due to Carathéodory, is one of the basic theorems in measure theory. It says that if the measure $\mu$ is defined on the algebra, then we can automatically and uniquely extend it to the $\sigma$-algebra generated by $\mathcal{A}$.

**Theorem 2.11** (Carathéodory’s extension theorem). Let $\mathcal{A}$ be an algebra of subsets of a given set $\Omega$. One can always extend every $\sigma$-finite measure defined on $\mathcal{A}$ to the $\sigma$-algebra generated by $\mathcal{A}$; moreover, the extension is unique.
Example 2.12. Let $\mathcal{A}$ be the algebra on $\mathbb{R}$, defined in Example 2.10. Let $\mu$ be the measure on $\mathcal{A}$, defined by setting the measure of an interval to its length. By Carathéodory’s extension theorem, $\mu$ extends uniquely to the $\sigma$-algebra generated by $\mathcal{A}$. The resulting measure is called the Borel measure on $\mathbb{R}$.

Consider two measure spaces $\mathcal{M} := (\Omega, \mathcal{F}, \mu)$ and $\mathcal{N} := (\Sigma, \mathcal{G}, \nu)$. The product measure $\mu \times \nu$ on $\Omega \times \Sigma$ is defined in the following way: For $F \in \mathcal{F}$ and $G \in \mathcal{G}$, define $\mu \times \nu (F \times G) = \mu(F) \times \nu(G)$. So far we defined the measure $\mu \times \nu$ on $A := \{F \times G : F \in \mathcal{F}, G \in \mathcal{G}\}$. Note that $A$ is an algebra in that $\emptyset \in A$, and $A$ is closed under complementation and finite unions of its members. However, $A$ is not necessarily a $\sigma$-algebra, as it is possible that $A$ is not closed under countable unions of its members. Let $F \times G$ be the $\sigma$-algebra generated by $A$, i.e. it is obtained by closing $A$ under complementation and countable unions. It should be noted that $F \times G$ is not the cartesian product of the two sets $\mathcal{F}$ and $\mathcal{G}$, and instead it is the $\sigma$-algebra generated by the cartesian product of $\mathcal{F}$ and $\mathcal{G}$. Theorem 2.11 shows that $\mu \times \nu$ extends uniquely from $A$ to a measure over all of $F \times G$. We denote the corresponding measure space by $\mathcal{M} \times \mathcal{N}$ which is called the product measure of $\mathcal{M}$ and $\mathcal{N}$.

Probability Spaces

A measure space $(\Omega, \mathcal{F}, \mu)$ is called a probability space if $\mu : \mathcal{F} \to [0, 1]$, and $\mu(\emptyset) = 0$, and $\mu(\Omega) = 1$.

Consider two measure spaces $\mathcal{M} = (\Omega, \mathcal{F}, \mu)$ and $\mathcal{N} = (\Sigma, \mathcal{G}, \nu)$. A function $X : \Omega \to \Sigma$ is called measurable if the preimage of every set in $\mathcal{G}$ belongs to $\mathcal{F}$. If $\mathcal{M}$ is a probability space, then $X$ is called a random variable. In this case, for every $S \in \mathcal{G}$, we have

$$\Pr[X \in S] := \mu(\{a \in \Omega : X(a) \in S\}).$$

Example 2.13. Let $\Omega = \{00, 01, 10, 11\}$, $\mathcal{F} = \mathcal{P}(\Omega)$, and $\mu(A) = |A|/4$ for all $A \in \mathcal{F}$. Here $\mu$ is the uniform probability measure on $\Omega$. Let $X : \Omega \to \mathbb{N}$ be defined as $X(00) = 0$, $X(10) = X(01) = 1$, and $X(11) = 2$. Here $\mathbb{N}$ is considered with the discrete counting measure. Now $X$ is a random variable, and for example

$$\Pr[X \in \{1, 2\}] = \mu(\{10, 01, 11\}) = \frac{3}{4}.$$

We finish this section by stating the Borel-Cantelli theorem.

Theorem 2.14 (Borel-Cantelli). Let $(E_n)$ be a sequence of events in some probability space. If the sum of the probabilities of the $E_n$ is finite, then the probability that infinitely many of them occur is 0, that is,

$$\sum_{n=1}^{\infty} \Pr[E_n] < \infty \Rightarrow \Pr[\limsup_{n\to\infty} E_n] = 0,$$

where

$$\limsup_{n\to\infty} E_n := \bigcap_{n=1}^{\infty} \bigcup_{k=1}^{n} E_k.$$
Normed spaces

A **metric space** is an ordered pair \((M, d)\) where \(M\) is a set and \(d\) is a **metric** on \(M\), that is, a function \(d : M \times M \to [0, \infty)\) such that

- Non-degeneracy: \(d(x, y) = 0\) if and only if \(x = y\).
- Symmetry: \(d(x, y) = d(y, x)\), for every \(x, y \in M\).
- Triangle inequality: \(d(x, z) \leq d(x, y) + d(y, z)\), for every \(x, y, z \in M\).

A sequence \(\{x_n\}_{n=1}^\infty\) of elements of a metric space \((M, d)\) is called a **Cauchy sequence** if for every \(\varepsilon > 0\), there exist an integer \(N_\varepsilon\), such that for every \(m, n \geq N_\varepsilon\), we have \(d(x_m, x_n) \leq \varepsilon\). A metric space \((M, d)\) is called **complete** if every Cauchy sequence has a limit in \(M\). A metric space is **compact** if and only if every sequence in the space has a convergent subsequence.

Now that we have defined the measure spaces in Section 2, let us state the Hölder’s and Minkowski’s inequalities in a more general form.

**Theorem 2.15** (Hölder’s inequality). Consider a measure space \(M = (\Omega, \mathcal{F}, \mu)\), and two reals \(1 < p, q < \infty\) with \(\frac{1}{p} + \frac{1}{q} = 1\). If the two measurable functions \(f, g : \Omega \to \mathbb{C}\) are such that both \(|f|^p\) and \(|g|^q\) are integrable, then

\[
\left(\int |f(x)g(x)|d\mu(x)\right)^{1/p} \leq \left(\int |f(x)|^pd\mu(x)\right)^{1/p} \left(\int |g(x)|^qd\mu(x)\right)^{1/q}.
\]

**Theorem 2.16** (Minkowski’s inequality). Consider a measure space \(M = (\Omega, \mathcal{F}, \mu)\), a real \(p \geq 1\), and two measurable functions \(f, g : \Omega \to \mathbb{C}\) such that \(|f|^p\) and \(|g|^p\) are both integrable. Then

\[
\left(\int |f(x) + g(x)|^pd\mu(x)\right)^{1/p} \leq \left(\int |f(x)|^pd\mu(x)\right)^{1/p} + \left(\int |g(x)|^pd\mu(x)\right)^{1/p}.
\]

Next we define concept of a normed space which is central to function analysis.

**Definition 2.17.** A **normed space** is a pair \((V, \| \cdot \|)\), where \(V\) is a vector space over \(\mathbb{R}\) or \(\mathbb{C}\), and \(\| \cdot \|\) is a function from \(V\) to nonnegative reals satisfying

- (non-degeneracy): \(\|x\| = 0\) if and only if \(x = 0\).
- (homogeneity): For every scalar \(\lambda\), and every \(x \in V\), \(\|\lambda x\| = |\lambda|\|x\|\).
- (triangle inequality): For \(x, y \in V\), \(\|x + y\| \leq \|x\| + \|y\|\).

We call \(\|x\|\), the **norm** of \(x\). A **semi-norm** is a function similar to a norm except that it might not satisfy the non-degeneracy condition.

The spaces \((\mathbb{C}, | \cdot |)\) and \((\mathbb{R}, | \cdot |)\) are respectively examples of 1-dimensional complex and real normed spaces.

Every normed space \((V, \| \cdot \|)\) has a metric space structure where the distance of two vectors \(x\) and \(y\) is \(\|x - y\|\).

Consider two normed spaces \(X\) and \(Y\). A **bounded operator** from \(X\) to \(Y\), is a linear function \(T : X \to Y\), such that

\[
\|T\| := \sup_{x \neq 0} \frac{\|Tx\|_Y}{\|x\|_X} < \infty. \tag{2.1}
\]

The set of all bounded operators from \(X\) to \(Y\) is denoted by \(B(X, Y)\). Note that the **operator norm** defined in (2.1) makes \(B(X, Y)\) a normed space.

A **functional** on a normed space \(X\) over \(\mathbb{C}\) (or \(\mathbb{R}\)) is a bounded linear map \(f\) from \(X\) to \(\mathbb{C}\) (respectively \(\mathbb{R}\)), where bounded means that

\[
\|f\| := \sup_{x \neq 0} \frac{|f(x)|}{\|x\|} < \infty.
\]

The set of all bounded functionals on \(X\) endowed with the operator norm, is called the **dual** of \(X\) and is denoted by \(X^*\). So for a normed space \(X\) over complex numbers, \(X^* = B(X, \mathbb{C})\), and similarly for a normed space \(X\) over real numbers, \(X^* = B(X, \mathbb{R})\).

For a normed space \(X\), the set \(B_X := \{x : \|x\| \leq 1\}\) is called the **unit ball** of \(X\). Note that by the triangle inequality, \(B_X\) is a convex set, and also by homogeneity it is symmetric around the origin, in the sense that \(\|\lambda x\| = \|x\|\) for every scalar \(\lambda\) with \(|\lambda| = 1\). The non-degeneracy condition implies that \(B_X\) has non-empty interior.
Every compact symmetric convex subset of \( \mathbb{R}^n \) with non-empty interior is called a **convex body**. Convex bodies are in one-to-one correspondence with norms on \( \mathbb{R}^n \). A convex body \( K \) corresponds to the norm \( \| \cdot \|_K \) on \( \mathbb{R}^n \), where

\[
\|x\|_K := \sup\{\lambda \in [0, \infty) : \lambda x \in K\}.
\]

Note that \( K \) is the unit ball of \( \| \cdot \|_K \). For a set \( K \subseteq \mathbb{R}^n \), define its **polar conjugate** as

\[
K^\circ = \{ x \in \mathbb{R}^n : \sum x_i y_i \leq 1, \forall y \in K \}.
\]  
(2.2)

The polar conjugate of a convex body \( K \) is a convex body, and furthermore \( (K^\circ)^\circ = K \).

Consider a normed space \( X \) on \( \mathbb{R}^n \). For \( x \in \mathbb{R}^n \) define \( T_x : \mathbb{R}^n \to \mathbb{R} \) as \( T_x(y) := \sum_{i=1}^n x_i y_i \). It is easy to see that \( T_x \) is a functional on \( X \), and furthermore every functional on \( X \) is of the form \( T_x \) for some \( x \in \mathbb{R}^n \). For \( x \in \mathbb{R}^n \) define \( \|x\|^* := \|T_x\| \). This shows that we can identify \( X^* \) with \((\mathbb{R}^n, \| \cdot \|)^*\). Let \( K \) be the unit ball of \( \| \cdot \| \). It is easy to see that \( K^\circ \), the polar conjugate of \( K \), is the unit ball of \( \| \cdot \|^* \).

### Hilbert Spaces

Consider a vector space \( V \) over \( \mathbb{K} \), where \( \mathbb{K} = \mathbb{R} \) or \( \mathbb{K} = \mathbb{C} \). Recall that an **inner product** \( \langle \cdot, \cdot \rangle \) on \( V \), is a function from \( V \times V \) to \( \mathbb{K} \) that satisfies the following axioms.

- **Conjugate symmetry**: \( \langle x, y \rangle = \overline{\langle y, x \rangle} \).
- **Linearity in the first argument**: \( \langle ax + z, y \rangle = a \langle x, y \rangle + \langle z, y \rangle \) for \( a \in \mathbb{K} \) and \( x, y \in V \).
- **Positive-definiteness**: \( \langle x, x \rangle > 0 \) if and only if \( x \neq 0 \), and \( \langle 0, 0 \rangle = 0 \).

A vector space together with an inner product is called an **inner product space**.

**Example 2.18.** Consider a measure space \( \mathcal{M} = (\Omega, \mathcal{F}, \mu) \), and let \( \mathcal{H} \) be the space of measurable functions \( f : \Omega \to \mathbb{C} \) such that \( \int |f(x)|^2 d\mu(x) < \infty \). For two functions \( f, g \in \mathcal{H} \) define

\[
\langle f, g \rangle := \int f(x) \overline{g(x)} d\mu(x).
\]

It is not difficult to verify that the above mentioned function is indeed an inner product.

An inner product can be used to define a norm on \( V \). For a vector \( x \in V \), define \( \|x\| = \sqrt{\langle x, x \rangle} \).

**Lemma 2.19.** *For an inner product space \( V \), the function \( \| \cdot \| : x \mapsto \sqrt{\langle x, x \rangle} \) is a norm.*

**Proof.** The non-degeneracy and homogeneity conditions are trivially satisfied. It remains to verify the triangle inequality. Consider two vectors \( x, y \in V \) and note that by the axioms of an inner product:

\[
0 \leq \langle x + \lambda y, x + \lambda y \rangle = \langle x, x \rangle + |\lambda|^2 \langle y, y \rangle + \lambda \overline{\langle x, y \rangle} + \overline{\lambda} \langle x, y \rangle.
\]

Now taking \( \lambda := \sqrt{\frac{\langle x, x \rangle \cdot \langle y, y \rangle}{|\langle x, y \rangle|}} \) will show that

\[
0 \leq 2 \langle x, x \rangle \langle y, y \rangle - 2 \sqrt{\langle x, x \rangle \langle y, y \rangle} \langle x, y \rangle,
\]

which leads to the triangle inequality. \( \square \)

A complete inner-product space is called a **Hilbert space**.

**Example 2.20.** Consider the vector space \( V \) of all functions \( f : \mathbb{N} \to \mathbb{R} \) that have finite supports, meaning that \( \{ x : f(x) \neq 0 \} \) is finite. This is clearly a vector space over \( \mathbb{R} \), and can be turned into an inner product space with the inner product

\[
\langle u, v \rangle = \sum_{i \in \mathbb{N}} u_i v_i.
\]

However this is not a Hilbert space as it is not complete. For example consider the sequence of vectors

\[
u^{(k)} = (1, 2^{-k}, 3^{-k}, \ldots, 2^{-k}, 0, 0, \ldots).
\]

It is easy to see that \( \nu^{(1)}, \nu^{(2)}, \ldots \) is a Cauchy sequence, but it does not have a limit in \( V \), and hence \( V \) is not a Hilbert space. However we can complete \( V \) to a Hilbert space by extending it to include all functions \( f : \mathbb{N} \to \mathbb{R} \) with

\[
\sum_{i \in \mathbb{N}} |f(i)|^2 < \infty.
\]
The $L_p$ spaces

Consider a measure space $\mathcal{M} = (\Omega, \mathcal{F}, \mu)$. For $1 \leq p < \infty$, the space $L_p(\mathcal{M})$ is the space of all functions $f : \Omega \to \mathbb{C}$ such that

$$
\|f\|_p := \left( \int |f(x)|^p d\mu(x) \right)^{1/p} < \infty.
$$

Strictly speaking the elements of $L_p(\mathcal{M})$ are equivalent classes. Two functions $f_1$ and $f_2$ are equivalent and are considered identical, if they agree almost everywhere or equivalently $\|f_1 - f_2\|_p = 0$.

**Proposition 2.21.** For every measure space $\mathcal{M} = (\Omega, \mathcal{F}, \mu)$, $L_p(\mathcal{M})$ is a normed space.

**Proof.** Non-degeneracy and homogeneity are trivial. It remains to verify the triangle inequality (or equivalently prove Minkowski’s inequality). By applying Hölder’s inequality:

$$
\|f + g\|_p^p = \int |f(x) + g(x)|^p d\mu(x) = \int |f(x) + g(x)|^{p-1}|f(x) + g(x)| d\mu(x)
\leq \int |f(x) + g(x)|^{p-1}|f(x)| d\mu(x) + \int |f(x) + g(x)|^{p-1}|g(x)| d\mu(x)
\leq \left( \int |f(x) + g(x)|^p d\mu(x) \right)^{q-1/p} \|f\|_p + \left( \int |f(x) + g(x)|^p d\mu(x) \right)^{q-1/p} \|g\|_p
= \|f + g\|^{p-1}(\|f\|_p + \|g\|_p),
$$

which simplifies to the triangle inequality.

Another useful fact about the $L_p$ norms is that when they are defined on a probability space, they are increasing.

**Theorem 2.22.** Let $\mathcal{M} = (\Omega, \mathcal{F}, \mu)$ be a probability space, $1 \leq p \leq q \leq \infty$ be real numbers, and $f \in L_q(\mathcal{M})$. Then

$$
\|f\|_p \leq \|f\|_q.
$$

**Proof.** The case $q = \infty$ is trivial. For the case $q < \infty$, by Hölder’s inequality (applied with conjugate exponents $\frac{q}{p}$ and $\frac{q}{q-p}$), we have

$$
\|f\|_p^p = \int |f(x)|^p \times 1 d\mu(x) \leq \left( \int |f(x)|^q d\mu(x) \right)^{p/q} \left( \int 1^{q-p} d\mu(x) \right)^{q-p/q} = \|f\|_q^p.
$$

Note that Theorem 2.22 does not hold when $\mathcal{M}$ is not a probability space. For example consider the set of natural numbers $\mathbb{N}$ with the counting measure. We shall use the notation $\ell_p := L_p(\mathbb{N})$. In this case the $\ell_p$ norms are actually decreasing.

**Exercises**

**Exercise 2.1.** Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ be complex vectors. By studying the derivative of $\langle x + ty, x + ty \rangle$ with respect to $t$, prove Theorem 2.3.

**Exercise 2.2.** Deduce Theorem 2.5 from Hölder’s inequality.

**Exercise 2.3.** Let $1 \leq p \leq q \leq \infty$. Show that for every $f \in \ell_p$, we have $\|f\|_q \leq \|f\|_p$.

**Exercise 2.4.** Recall that by Hölder’s inequality, if $p, q \geq 1$ are conjugate exponents and $a_1, \ldots, a_n, b_1, \ldots, b_n$ are complex numbers, then

$$
\left| \sum_{i=1}^n a_i b_i \right| \leq \left( \sum_{i=1}^n |a_i|^p \right)^{1/p} \left( \sum_{i=1}^n |b_i|^q \right)^{1/q}.
$$

Deduce from this, that if $p_1, \ldots, p_n$ and $a_1, \ldots, a_n, b_1, \ldots, b_n$ are non-negative numbers with $\sum_{i=1}^n p_i = 1$, then

$$
\left| \sum_{i=1}^n a_i b_i p_i \right| \leq \left( \sum_{i=1}^n |a_i|^{p_i} \right)^{1/p} \left( \sum_{i=1}^n |b_i|^{q_i} \right)^{1/q}.
$$
**Exercise 2.5.** Let $X$ be a probability space, and $p, q \geq 1$ be conjugate exponents. Show that for every $f \in L_p(X)$, we have

$$\|f\|_p = \sup_{g : \|g\|_q = 1} |\langle f, g \rangle|.$$ 

**Exercise 2.6.** Suppose that $(X, \mu)$ is a measure space and $\frac{1}{p} + \frac{1}{q} + \frac{1}{r} = 1$, for $p, q, r \geq 1$. Show that if $f \in L_p(X)$, $g \in L_q(X)$, and $h \in L_r(X)$, then

$$\left| \int f(x)g(x)h(x)d\mu(x) \right| \leq \|f\|_p \|g\|_q \|h\|_r.$$ 

**Exercise 2.7.** Suppose that $X$ is a measure space and $\frac{1}{p} + \frac{1}{q} = \frac{1}{r}$, for $p, q, r \geq 1$. Show that if $f \in L_p(X)$ and $g \in L_q(X)$, then

$$\|fg\|_r \leq \|f\|_p \|g\|_q.$$ 

**Exercise 2.8.** Let $X$ be a probability space. Let $\|T\|_{p \to q}$ denote the operator norm of $T : L_p(X) \to L_q(X)$. In other words

$$\|T\|_{p \to q} := \sup_{f : \|f\|_p = 1} \|Tf\|_q.$$ 

Recall that the adjoint of $T$ is an operator $T^*$ such that

$$\langle Tf, g \rangle = \langle f, T^*g \rangle,$$

for all $f, g \in L_2(X)$. Prove that for conjugate exponents $p, q \geq 1$, and every linear operator $T : L_2(X) \to L_2(X)$, we have

$$\|T\|_{p \to 2} = \|T^*\|_{2 \to q}.$$
Chapter 3

Time complexity

In the first half of the twentieth century, theoretical computer scientists focused on understanding which problems are solvable by algorithms. By the 1950s, thanks to the great works of Turing, Church, Gödel, and others, we acquired a deep understanding of this division of problems into decidable and undecidable ones. However, it quickly became evident that this dividing line is too coarse as many decidable problems remain intractable since the best-known algorithms to solve them will terminate long after anyone cared or lived to see the answer. These observations created a need for a much more refined theory that will account for the performance of different algorithms. Thus was born computational complexity theory in the 1960s, with the initial charge of understanding efficient computation in the most general sense: determining the minimal amounts of natural resources such as time, memory, and communication needed to solve natural computational tasks by natural computational models [Wig19].

Our computational model

All the strongest known natural models of computation are essentially equivalent in the sense that the complexity of a problem can only change polynomially if we change the model. While this polynomial change is not an issue for defining classes such as \( P \) and \( \text{NP} \), it can become an issue for other notations that we will define soon. For example, suppose we define \( \text{DTIME}(n) \) as the class of problems that can be decided deterministically in linear time \( O(n) \). In that case, this definition depends on the model of a Turing Machine: a problem that might require quadratic time using a single-tape Turing Machine could potentially become solvable in linear time with a double-tape Turing Machine. Hence, to discuss the complexity of a problem rigorously, we need to be more specific about which model we use. When defining this model, we should also pay attention to another important detail regarding space complexity. Consider for example the language \( L \subseteq \{0,1\}^* \) consisting of all strings of even length. Note that deciding whether a string is of even or odd length requires very little “working” memory: As we read the input, we only need to keep one bit that we flip each time we read a new input symbol. However, if we define the space complexity as the total number of the used memory bits, even this simple language would have a linear space complexity because we start with the input written in the memory.

Since when discussing the space complexity, one is somewhat concerned with the working memory, our model will be a multi-tape Turing machine with a separate dedicated tape for the working space. The input alphabet in our model is \( \Sigma = \{0,1\} \) unless specified otherwise. We will not be as specific about the tape-alphabet, as our definitions use the \( O(\cdot) \) notation, and moving from one tape-alphabet to different tape-alphabet changes our complexity measures only by a constant factor. Finally, for this course, rather than an accept/reject Turing Machine, it is more convenient to work with Turing Machines that compute functions. Without further ado, let us formally describe the Three-Tape Turing Machine model that we shall use:

- The Turing Machine has a finite set of states, including a halt state, and a start state.
- There are three tapes:
  - A read-only input tape.
  - A read/write work tape.
  - A write-only output tape. The output-tape works slightly differently. The options for the transitions on this tape are either to print a character, in which case the tape-head moves to the next cell after printing the character, or not to print anything, in which case the tape-head remains at its current position.
Each tape is equipped with a tape-head pointing to a particular position on the tape. Initially, the input followed by blanks is put on the “input tape”, and the rest of the tapes are filled with only blanks. Initially, the tape-heads point to the left-most entry of the tapes.

The function computed by such a Turing Machine $M$ is defined as

$$M(x) = \begin{cases} \text{Output-tape content} & \text{if } M \text{ halts on the input } x \\ \bot & \text{otherwise.} \end{cases}$$

We say that $M$ decides a language $L \subseteq \{0,1\}^*$ if for every input $x$, we have that

$$x \in L \Rightarrow M(x) = 1;$$

$$x \notin L \Rightarrow M(x) = 0.$$ 

That is we interpret the output 1 as Accept and the output 0 as Reject, and thus we can use these Turing Machines to Accept or Reject inputs.

The running time of $M$ on an input $x$ is the number of steps that $M$ takes on $x$ until it halts.

Next we need to decide how to define the space complexity of $M$ on an input $x$. The main question is whether to include the output-tape as part of the used space, or to define the space solely based on the work-tape. Consider the function $f$ defined as $f(x) = x$. That is $f$ simply outputs its input without processing. As we shall see in the discussion of reductions, it is desirable to define the space complexity of this function to be $O(1)$. This suggests that we should not include the output tape to contribute to the space complexity.

The space complexity of $M$ on the input $x$ is the furthest location that the work-tape pointer-head reaches during the computation of $M$ on $x$. That is the number of cells in the work-tape that are “touched” by the tape-head.

### Time complexity

Let us start by discussing the time complexity, arguably the most important notion of complexity. Let $T : \mathbb{N} \to \mathbb{N}$ be a function. Recall that we say that a Turing Machine runs in time $T(n)$ if it halts after at most $T(n)$ steps on every input of length $n$.

**Definition 3.1.** Let $T : \mathbb{N} \to \mathbb{N}$ be a function.

- $\text{DTIME}(T(n))$ is the class of languages $L$ that be decided in deterministic time $O(T(n))$.
- $\text{NTIME}(T(n))$ is the class of languages $L$ that be decided in nondeterministic time $O(T(n))$.
- $\mathcal{P} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k)$ is the class of the languages that can be decided in polynomial deterministic time.
- $\mathcal{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$ is the class of the languages that can be decided in polynomial nondeterministic time.
- $\text{EXP} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{n^k})$ is the class of the languages that can be decided in exponential deterministic time.
- $\text{NEXP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(2^{n^k})$ is the class of the languages that can be decided in exponential nondeterministic time.

The most famous question of complexity theory is whether $\mathcal{P} = \mathcal{NP}$. This question has several interpretations, some almost philosophical. We will not discuss those in this course and assume that the students are already familiar with this question and related topics such as NP-completeness. See the introduction of Aaronson’s survey [Aar16] for a discussion on the importance of this question. It is conjectured that $\mathcal{P} \neq \mathcal{NP}$, but nobody knows how to prove this. In fact the question seems to be well beyond the reach of the current known techniques in complexity theory and mathematics. Even though we seem to be very far from solving this question, it plays a central and fruitful role in complexity theory as it provides a direction for the area. In fact many of the results and theories covered in this course were initially developed as attempts to get us closer to answering this question. We proceed by some remarks on the current state of the complexity theory.
• **Reductions:** This is something we are generally good at. We can often show that if one can solves a problem $X$ using certain amount of computational resources, then we can also solve a problem $Y$ using a similar amount of computational resources. Perhaps the greatest example of this is the extensive list of $\text{NP}$-complete problems that has been compiled over the few decades since the proof of the Cook-Levin theorem, and Karp’s subsequent seminal paper [Kar72].

• **Counting arguments:** Sometimes we are able to show that there are problems that are in some class $Y$ but not in $X$ by counting arguments. That is by showing that $Y$ is a much larger set compared to $X$. This is more or less a finitary version of the countability argument that shows that there are languages that are not decidable: the set of decidable languages is countable (i.e. small) while the set of all languages is uncountable (i.e. large).

• **Diagonalization:** This is another important technique in complexity theory that allows us to sometimes separate complexity classes.

• We are extremely bad at proving statements which say that a specific given problem do not belong to a complexity class. For example $\text{P} \neq \text{NP}$ is equivalent to the statement that $\text{SAT} \notin \text{P}$. However as we shall see, we cannot even show that $\text{SAT}$ does not belong to much smaller classes than $\text{P}$.

• Generally we are quite clueless at separating complexity classes. We can often show containment results that a class is a subset of the other, but there are very few known results in complexity theory that separate complexity classes.

**Time Hierarchy theorems**

The first theorem that we prove in this course uses the diagonalization technique to separate $\text{P}$ from $\text{EXP}$. That is we want to design a language $L$ that can be decided in exponential time, but cannot be decided in polynomial time. We will need the following technical fact.

**Remark 3.2.** Let $M$ be a Turing Machine with running time $T(n)$ and space $S(n)$. One can use the universal Turing Machine $U$ to simulate $M$ on an input $w$ with $|w| = n$ in time $O(T(n) \log T(n))$ and space $O(S(n) + \log(n))$. In other words simulating $M$ on $w$ requires a multiplicative time overlay of $\log(T(n))$ and an additive overlay of $\log(n)$ in space. The reason for these overlays are that simple operations such as updating the location of the tape-head of $M$ on its input requires time and space $\Omega(\log(n))$.

Let us start with an easier task.

**Remark 3.3.** We are given the description of a Turing Machine $M$, and we are asked to design a Turing Machine $N$ such that $N$ runs in time $O(2^n)$, and $L(N) \neq L(M)$ if $M$ is a polynomial time algorithm.

Note that in order to satisfy $L(N) \neq L(M)$, we just need to have one element $w$ such that only one of $M$ or $N$ accepts $w$. However the difficulty of the question is in that we cannot determine the running time of $M$. For example, as a naive attempt, we might pick a fixed $w$, say $w = 0$, simulate $M$ on it, if $M$ accepts it, then reject $w$, and otherwise accept it. The issue is that $M$ might go to an infinite loop on $w$. We have no way of knowing when to stop: Even though we only care about $M$’s that have polynomial running time, this running time could be $n$, $10n$, $100n^{100}$, etc. Here is however a trick that solves this problem. Construct $N$ as in the following.

**Algorithm 1:** Turing Machine $N$

<table>
<thead>
<tr>
<th>Data:</th>
<th>On input $w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set $n =</td>
<td>w</td>
</tr>
<tr>
<td>Simulate $M$ on $w$ for at most $t$ steps;</td>
<td></td>
</tr>
<tr>
<td>if $M$ halts and accepts $w$ in these $t$ steps then</td>
<td></td>
</tr>
<tr>
<td>REJECT;</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>ACCEPT;</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

By remark 3.2 simulating a Turing Machine $M$ on an input $w$ for $t$ steps can be done in time $O(t \log(t))$. Hence the running time of $N$ is $O(2^{n/2}(n/2)) = O(2^n)$. Furthermore if the running time of $M$ is bounded by some polynomial $p(n)$, there exists a $k$ such that $p(k) < 2^{k/2}$. Now take any input $w$ of length $k$, and note that $N$ accepts $w$ if and only if $M$ rejects $w$, and thus in this case $L(M) \neq L(N)$.

In Remark 3.3, we managed to construct a Turing Machine $N$ that runs in exponential time and its language is different from the language of a specific $M$ if $M$ runs in polynomial time. However, in order to prove $\text{P} \neq \text{EXP}$, we
Definition 3.5. We have \( P \neq \text{EXP} \).

Proof. Consider the following Turing Machine:

\[
\text{Algorithm 2: Turing Machine } N
\]

<table>
<thead>
<tr>
<th>Data: On input ( w = \langle M, x \rangle ), where ( M ) is a description of a TM.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let ( n =</td>
</tr>
<tr>
<td>Simulate ( M ) on ( w = \langle M, x \rangle ) for at most ( t ) steps.</td>
</tr>
<tr>
<td>\text{if } M \text{ halts and accepts } w \text{ in these } t \text{ steps then}</td>
</tr>
<tr>
<td>\hspace{1cm} REJECT;</td>
</tr>
<tr>
<td>\text{else}</td>
</tr>
<tr>
<td>\hspace{1cm} ACCEPT;</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Note that we are simulating \( M \) on \( w = \langle M, x \rangle \) rather than on \( x \). The running time of \( N \) is \( O(2^n) \). On the other hand, consider any Turing Machine \( M \) with polynomial running time \( p(n) \). Consider a sufficiently long string \( x \) such that for \( n = |\langle M, x \rangle| \), we have \( p(n) < 2^{n/2} \). Such a string exists because \( 2^{n/2} \) grows faster than any polynomial \( p \). Let \( w = \langle M, x \rangle \). Note that \( M \) will halt on \( w \) in the first \( t \) steps, and thus \( N \) accepts \( w \) if and only if \( M \) rejects \( w \).

This shows \( L(N) \neq L(M) \) for any Turing Machine \( M \) that has a polynomial running time. Thus \( L(N) \in \text{EXP} \) but \( L(N) \notin \text{P} \). \( \square \)

We can easily generalize Theorem 3.4 to other running times. However one must be careful. In the description of the Turing Machine \( N \) in the proof of Theorem 3.4, we compute \( t = 2^{n/2} \). This does not change the running time of \( N \) by much because computing \( 2^{n/2} \) can be done very efficiently. However if \( t \) was a function that was difficult to compute, then just computing \( t \) could have changed the total running time drastically. Hence in order to state the so-called time-hierarchy theorem, we need to restrict to running time functions that are computable efficiently.

Definition 3.5 (Time-constructible function). A function \( T : \mathbb{N} \to \mathbb{N} \) is called time-constructible if it is non-decreasing, \( T(n) \geq n \log(n) \) for every \( n \), and there is a Turing Machine that computes \( x \to 1^{T(|x|)} \) in time \( O(T(|x|)) \).

Theorem 3.6 (Time Hierarchy Theorem). Let \( T_1, T_2 : \mathbb{N} \to \mathbb{N} \) be such that \( T_2 \) is time-constructible, and \( T_1(n) \log T_1(n) = o(T_2(n)) \). Then \( \text{DTIME}(T_1(n)) \neq \text{DTIME}(T_2(n)) \).

Proof. The log factor in the statement is an artifact of Remark 3.2. We leave the details to the reader. \( \square \)

Exercises

Exercise 3.1. Where is the error in the following false proof of the false claim that every decidable language is in \( P \)? Consider an algorithm \( M \) that decides a language \( L \). For every input \( x \), there exists a polynomial \( p \) such that \( p(|x|) \) is larger than the running time of \( M \) on \( x \). Hence for every \( x \), we can simulate \( M \) on \( x \), and in polynomial time decide whether \( x \in L \).

Exercise 3.2. Prove Theorem 3.6.

Exercise 3.3. Prove a Non-deterministic Time Hierarchy theorem: If \( T_2 \) is time-constructible and \( T_1(n + 1) \log T_1(n + 1) = o(T_2(n)) \), then \( \text{NTIME}(T_1(n)) \neq \text{NTIME}(T_2(n)) \).

- First explain why the same diagonalization used for the deterministic time complexity does not work here without any changes.
- Show how to fix the argument so that it works for the non-deterministic time complexity. (\textbf{Hint:} Use a \textit{lazy diagonalization} argument: Only try to disagree at least once in an exponentially large interval. Consider a large intervals \( [\ell_k, u_k] \), and for an input \( 1^n \in (\ell_k, u_k) \) if \( n = u_k \), then deterministically run \( M_k \) on \( 1^{\ell_k+1} \) for an appropriate number of steps, and negate its output. If \( n < u_k \), then use a non-deterministic simulation.)
Chapter 4

Polynomial Hierarchy

In order to motivate the definition of the polynomial hierarchy, it is useful to rewrite the definitions of P, NP, coNP in the following manner:

- P is the set of all languages $L$ such that there exists an algorithm $M$ that runs in polynomial-time in $|x|$ and $x \in L \iff M(x) = \text{Accept}$.

- NP is the set of all languages $L$ such that there exists an algorithm $M$ that runs in polynomial-time in $|x|$ and $x \in L \iff \exists y, M(x, y) = \text{Accept}$.

- coNP is the set of all languages $L$ such that there exists an algorithm $M$ that runs in polynomial-time in $|x|$ and $x \in L \iff \forall y, M(x, y) = \text{Accept}$.

These definitions suggest certain generalizations of these classes:

**Definition 4.1.** For $k \in \mathbb{N}$, the complexity classes $\Sigma_k$ and $\Pi_k$ are defined as in the following by alternating the quantifiers.

- The class $\Sigma_k$ is the set of all languages $L$ such that there exists an algorithm $M$ that runs in polynomial-time in $|x|$ and $x \in L \iff \exists y_1 \forall y_2 \exists y_3 \ldots y_k, M(x, y_1, \ldots, y_k) = \text{Accept}$, (4.1)

where the quantifiers are alternating.

- The class $\Pi_k$ is the set of all languages $L$ such that there exists an algorithm $M$ that runs in polynomial-time in $|x|$ and $x \in L \iff \forall y_1 \exists y_2 \forall y_3 \ldots y_k, M(x, y_1, \ldots, y_k) = \text{Accept}$, (4.2)

where the quantifiers are alternating.

Note that $P = \Sigma_0 = \Pi_0$, $NP = \Sigma_1$ and $coNP = \Pi_1$, and we have the inclusions $\Sigma_k \cup \Pi_k \subseteq \Sigma_{k+1} \cap \Pi_{k+1}$. Note further that $\Pi_k = co\Sigma_k$ in the sense that $L \in \Sigma_k$ if and only if $L^c \in \Pi_k$. Indeed if $L \in \Sigma_k$, then there exists an algorithm $M$ such that (4.1) holds. Then consider the algorithm $M'$ that runs $M$ and flips its output. That is $M'(x, y_1, \ldots, y_k) = \text{Accept} \iff M(x, y_1, \ldots, y_k) = \text{Reject}$. Note that by negating (4.1) we have

$x \not\in L \iff \forall y_1 \exists y_2 \forall y_3 \ldots y_k, M'(x, y_1, \ldots, y_k) = \text{Accept}$,

which shows that $L^c \in \Pi_k$. Similarly we can start from $L^c \in \Pi_k$, and obtain $L \in \Sigma_k$. 

19
Example 4.2. Define the language Unique SAT (denoted by $\neg SAT$) as

$$\neg SAT = \{ \phi : \phi \text{ is a CNF that has a unique satisfying assignment} \}.$$  

Note that $\neg SAT \in \Sigma_2$. Indeed let $M(\phi, y_1, y_2)$ be an efficient algorithm that takes a CNF formula $\phi$, and two truth assignments $y_1$ and $y_2$, and accepts them if and only if $y_1$ satisfies $\phi$ and either $y_2 = y_1$ or $y_2$ does not satisfy $\phi$. Now we have

$$\phi \in \neg SAT \iff \exists y_1 \forall y_2, \ M(\phi, y_1, y_2) = \text{Accept},$$

which shows that $\neg SAT \in \Sigma_2$.

Finally, the polynomial hierarchy is defined as

$$\text{PH} = \bigcup_{k=1}^{\infty} \Sigma_k = \bigcup_{k=1}^{\infty} \Pi_k.$$  

In other words $L \in \text{PH}$ if $L \in \Sigma_k$ for some $k$.

Theorem 4.3. We have $\text{PH} \subseteq \text{EXP}$.

Proof. Since $\text{PH} = \bigcup_{k=1}^{\infty} \Sigma_k$, it suffices to show that for every $k \in \mathbb{N}$, we have $\Sigma_k \subseteq \text{EXP}$. Consider $L \in \Sigma_k$, and let $M$ be a fast algorithm such that (4.1) holds. We know that there exists a polynomial $p(n)$ such that $M(x, y_1, \ldots, y_k)$ runs in time $t = p(|x|)$. In particular $M$ can never read beyond the first $t$ letters of any of the $y_i$'s, and thus it suffices to consider $y_i$'s that are of length at most $t$.

We use $M$ to design an exponential algorithm $N$ that decides $L$. Indeed we can have $k$ nested loops, where the $j$'th loop goes over all $y_j$'s of length at most $t$. Inside the innermost loop we simulate $M$ on $(x, y_1, \ldots, y_n)$. We use these loops to verify whether the right hand side of (4.2) holds. If it holds, we accept $x$, otherwise we reject.

Note that $k = O(1)$, and it does not depend on the size of the input. Thus the running time of this algorithm is $2^{O(k)} t \log(t) = 2^{O(t)} = 2^p(|x|)$, where the $t \log(t)$ accounts for simulating $M$ on $(x, y_1, \ldots, y_n)$.

Finally we want to show that if $P = \text{NP}$, then the polynomial hierarchy collapses to $\text{PH} = P$. To simplify the presentation, we present the proof that if $P = \text{NP}$, then $P = \Sigma_2$.

Proposition 4.4. If $P = \text{NP}$, then $P = \Sigma_2$.

Proof. Since $P = \text{NP}$, and $P$ is closed under taking complements, we conclude that $P = \text{NP} = \text{coNP}$.

Consider $L \in \Sigma_2$. Let $M$ be such that

$$x \in L \iff \exists y_1 \forall y_2, \ M(x, y_1, y_2) = \text{Accept},$$

and $M$ runs in polynomial time in $|x|$. Define

$$\tilde{L} = \{(x, y_1) : \forall y_2 M(x, y_1, y_2) = \text{Accept} \}.$$  

Note that $\tilde{L} \in \Pi_1 = \text{coNP}$, because the condition $M(x, y_1, y_2) = \text{Accept}$ can be verified in time $O(p(|x|))$ which is obviously polynomial in $|x, y_1|$. Hence since we assumed $P = \text{NP} = \text{coNP}$, we conclude that $\tilde{L} \in P$, and thus there is a Turing Machine $N$ that decides whether $(x, y_1) \in \tilde{L}$ in time that is polynomial in $|x, y_1|$.  

Now note that

$$x \in L \iff \exists y_1, |y_1| \leq p(|x|) \text{ and } (x, y_1) \in \tilde{L},$$

which together with the above discussion shows that $L \in \text{NP}$ as checking $|y_1| \leq p(|x|)$ and $(x, y_1) \in \tilde{L}$ can be done in a time that is polynomial in $|x|$.

One can use the above quantifier elimination argument to prove that if $\Sigma_k = \Sigma_{k+1}$, then $\Sigma_k = \Sigma_{k+2}$. This implies the following theorem.

Theorem 4.5. If $P = \text{NP}$, then $P = \text{PH}$. More generally if $\Sigma_k = \Sigma_{k+1}$, then $\Sigma_k = \text{PH}$.
Exercises

Exercise 4.1. Establish the inclusion $\Sigma_k \cup \Pi_k \subseteq \Sigma_{k+1} \cap \Pi_{k+1}$ for all $k$.

Exercise 4.2. Complete the proof of Theorem 4.5.

Exercise 4.3. Prove that under polynomial-time (mapping) reductions, $\text{QSAT}_k$ is a complete problem for $\Sigma_k$. Here $\text{QSAT}_k$ is the set of true statements of the form

$$\exists(x_1, \ldots, x_{i_2}) \forall(x_{i_2+1}, \ldots, x_{i_3}) \exists(x_{i_3+1}, \ldots, x_{i_4}) \cdots (x_{i_k+1}, \ldots, x_n) \phi(x_1, \ldots, x_n),$$

where $\phi$ is a CNF with $n$ variables $x_1, \ldots, x_n$ and $1 = i_1 < i_2 < i_3 < \ldots < i_k$. Here the number of quantifiers is $k$, it starts with $\exists$ and they alternate. Note that $\text{QSAT}_1 = \text{SAT}$.

You may use the fact that $\text{SAT}$ is NP-complete.

Exercise 4.4. Give the pseudo-code of the algorithm described in the proof of Theorem 4.3. What is the space complexity of this algorithm?

Exercise 4.5. Let $\text{QSAT}$ be the set of the true statements of the form

$$*x_1 * x_2 * x_3 \ldots * x_n \phi(x_1, \ldots, x_n),$$

where $\phi$ is a CNF and each $*$ is either a $\exists$ or a $\forall$ quantifier. Recall that $\text{PH} = \cup_{k=1}^{\infty} \Sigma_k$. Can we conclude that $\text{QSAT} \in \text{PH}$? Why?

Exercise 4.6. Consider the following problem. We are given an arrangement of an $n \times n$ chess board with chess pieces. We are asked if white starts and plays cleverly, can she always win after making at most 10 moves, no matter how black plays? Which complexity class in the polynomial hierarchy captures this problem?

Exercise 4.7. Show that if $\text{PH}$ does not collapse, that is $\text{PH} \neq \Sigma_k$ for any $k$, then it does not have a complete problem under the polynomial-time reductions.
Chapter 5

Space complexity

In the previous lectures we discussed several time complexity classes, deterministic and non-deterministic. Next we turn our attention to the space complexity. Recall that in our computational model, we used a separate tape for the work-space so that the sizes of the input and the output do not contribute to the space complexity. More precisely the space complexity of $M$ on an input $x$ is defined as the number of cells in the work-tape that are visited during the computation. Similar to the time complexity, the space complexity of $M$ is a function $S : \mathbb{N} \rightarrow \mathbb{N}$, where $S(n)$ is the maximum space complexity of $M$ over inputs of size $n$.

**Definition 5.1.** Let $S : \mathbb{N} \rightarrow \mathbb{N}$ be a function.

- $\text{SPACE}(S(n))$ is the class of languages $L$ that be decided in deterministic space $O(S(n))$.
- $\text{NSPACE}(S(n))$ is the class of languages $L$ that be decided in nondeterministic space $O(S(n))$.
- $L = \text{SPACE} \left( \log(n) \right)$ is referred to as log-space.
- $\text{NL} = \text{NSPACE} \left( \log(n) \right)$ is referred to as non-deterministic log-space.
- $\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k)$ is the class of the languages that can be decided in polynomial deterministic space.

**Theorem 5.3 (Space Hierarchy Theorem).** Let $S_1, S_2 : \mathbb{N} \rightarrow \mathbb{N}$ be such that $S_2$ is space-constructible, and $S_1(n) = o(S_2(n))$. Then $\text{SPACE}(S_1(n)) \neq \text{SPACE}(S_2(n))$.

Note that unlike Theorem 3.6, we do not need to multiply $S_1$ by $\log(S_1)$. Indeed as it is mentioned in Remark 3.2, simulating $M$ on $x$ for $S_1$ steps does not require the multiplicative overlay of $\log(S_1)$. Similar to the time complexity, by using the proof of the hierarchy theorem, we obtain

$$L \neq \text{PSPACE}.$$ 

**Time and Space**

Next we will investigate the relations between the time and space complexity classes. Recall the inclusions

$$P \subseteq \text{NP} \subseteq \text{PH} \subseteq \text{EXP}.$$ 

How large is the class $\text{PSPACE}$? Where does $\text{PSPACE}$ fit in the above picture? Does it contain $P$? This latter question is easy. Indeed the answer is affirmative because if a Turing Machine runs in polynomial time, then it can never use more than a polynomial amount of memory. This shows that more generally

$$\text{DTIME}(T(n)) \subseteq \text{SPACE}(T(n)).$$
How about NP, or PH, or even EXP? We will show that

$$\text{PH} \subseteq \text{PSPACE} \subseteq \text{EXP}.$$  

It is believed that these inclusions are proper, but yet again, we do not know how to separate these classes.  

First let us establish PSPACE \(\subseteq\) EXP. This is in fact an instance of a larger phenomenon. If the space complexity of an algorithm \(M\) is bounded by \(S(n)\), then the time complexity of \(M\) is bounded by \(2^{O(S(n) + \log(n))}\). This is quite easy to see. If the space is bounded by \(S(n)\), there are only at most \(2^{S(n)}\) different possibilities for the content of the work-tape, \(S(n)\) different possibilities for the location of the work-tape-head, \(n\) possibilities for the location of the input-tape-head, and \(O(1)\) possibilities for the state of the Turing Machine. These parameters together form a \emph{configuration}, which is essentially a snapshot of the Turing Machine at a particular time during its computation. Since the number of possible configurations is at most \(T(n) = O(2^{S(n)}S(n)n) = 2^{O(S(n) + \log(n))}\), if the running time on input \(M\) is larger that \(T(n)\) on any input of size \(n\), then there will be configuration that will be repeated during the computation. This however would imply that \(M\) will come back to the same configuration again and again, and thus will never halt. This is a contradiction as \(M\) is an algorithm.

**Theorem 5.4.** Let \(S : \mathbb{N} \to \mathbb{N}\) be a space-constructible function. Then \(\text{SPACE}(S) \subseteq \bigcup_{k=1}^{\infty} \text{DTIME}(2^{kS}).\)

Note that another consequence of Theorem 5.4 is that \(L \subseteq P\). We conclude

**Theorem 5.5.** We have

\[
L \subseteq P \subseteq \text{PH} \subseteq \text{PSPACE} \subseteq \text{EXP}.
\]

**Proof.** The inclusions \(L \subseteq P\), and \(\text{PSPACE} \subseteq \text{EXP}\) are immediate from Theorem 5.4. The inclusion \(\text{PH} \subseteq \text{PSPACE}\) follows from the proof of Theorem 4.3. It is not difficult to see that the algorithm presented in the proof of Theorem 4.3 uses a polynomial amount of space (See the Exercise 4.4 in the previous lecture). \(\square\)

**Remark 5.6.** The only known separations in Theorem 5.5 are \(L \neq \text{PSPACE}\) and \(P \neq \text{EXP}\).

Perhaps the most famous example of a language in PSPACE is QSAT. This language is defined similar to SAT, but now every variable can be quantified with either a \(\exists\) or \(\forall\) quantifier. If all the quantifiers are \(\exists\), then we recover the original SAT problem which asks whether there exists a truth assignment that satisfies the CNF formula. However QSAT allows using \(\forall\) quantifiers for some variables and this makes the problem more general, and probably more difficult.

**Definition 5.7 (QSAT).** Let QSAT be the set of the true statements of the form

\[
* x_1 * x_2 * x_3 \ldots * x_n \phi(x_1, \ldots, x_n),
\]

where \(\phi\) is a CNF and each \(*\) is either a \(\exists\) or a \(\forall\) quantifier.

For example

\[
\exists x_1 \forall x_2 \forall x_3 (x_1 \lor x_2) \land (x_1 \lor \neg x_2 \lor \neg x_3) \in \text{QSAT},
\]

while

\[
\forall x_1 \exists x_2 \forall x_3 (x_1 \lor x_2) \land (x_1 \lor \neg x_2 \lor \neg x_3) \not\in \text{QSAT}.
\]

It is easy to see that QSAT is in PSPACE. See Exercise 5.5, where the reader is also asked to prove that QSAT is PSPACE-complete under polynomial-time reductions.

**Logarithmic Space L and NL**

In the previous section, we saw that PSPACE is a powerful complexity class that includes the polynomial hierarchy. Indeed this class might be too powerful to capture space-efficiency. After all, every language in P already belongs to PSPACE, and we deem the languages that are not in P as intractable for time-related reasons. Hence when it comes to discussing the space-efficient algorithms, the classes L is arguably a better choice. It capture the set of problems that can be solved very efficiently in terms of space. It is easy to see that the two basic arithmetic operations \(+, \times\) fall into this class. It is also true, but much harder to prove, that the division \(\div\) also belongs to L. This has been established in 2001 by Chiu, Davida, and Litow [CDL01]. The class L is also important for practical reasons as sometimes the real-world data are so massive that we cannot store them, and we have to process the data as we read it. This is captured in the field of streaming algorithms where the input is presented as a sequence of items and can be examined in only a few passes, typically just one. In most models, these algorithms have access to limited memory,
generally logarithmic in the size of the stream. As we shall see later in the course, the class \( L \) is also related to parallel computing.

When discussing the logarithmic space classes, polynomial-time reductions are no longer an appropriate way of comparing the difficulty of problems. Indeed it is no longer true that if \( X \preceq_p Y \) and \( Y \in L \), then \( X \in L \). Instead we need the refiner notion of Logspace mapping reductions:

**Definition 5.8.** We say that \( X \) is Logspace mapping reducible to \( Y \), and write \( X \preceq_L Y \) if there exists a map \( f : \Sigma^* \to \Sigma^* \) that can be computed in logarithmic space, and it satisfies

\[
x \in X \iff f(x) \in Y.
\]

Note that even though \( f : \Sigma^* \to \Sigma^* \) is computed in logarithmic space, it could be the case that \( f(w) \) is not of logarithmic size. For example in the trivial reduction \( X \preceq_L X \) given by \( f : w \to w \), we can clearly compute \( f \) in logarithmic space, but \( f(|w|) = |w| = \omega(\log(|w|)) \). This might seem an obstacle for proving that if \( X \preceq_L Y \) and \( Y \in L \), then \( X \in L \), as given \( w \in X \), since we cannot simply afford to generate \( f(w) \) on the work-tape. Nevertheless as the following proof shows that there is a way around this.

**Theorem 5.9.** If \( X \preceq_L Y \) and \( Y \in L \), then \( X \in L \).

**Proof.** Let \( M_Y \) be the Turing Machine that decides \( Y \) in logarithmic space, and let \( f : \Sigma^* \to \Sigma^* \) be the logspace reduction from \( X \) to \( Y \) computable by a logspace algorithm \( M_f \). Given an input \( w \) for problem \( X \), we would like to run \( M_Y \) on \( f(w) \), and accept \( w \) if \( M_Y \) accept \( f(w) \), and reject otherwise. Note that we cannot simply generate \( f(w) \) on the work-tape. Instead to simulate \( M_Y \) on \( f(w) \), on the work tape we keep a number \( c \) that refers to the location of the tape-head for the input-tape of \( M_Y \) with input \( f(w) \). It follows from Theorem 5.4 that \( |f(w)| = 2^{O(\log |w|)} \) and thus \( c \) can be stored using a logarithmic space. Since our goal is to simulate \( M_Y \) on \( f(w) \), we need to find out the \( c \)-th bit of \( f(w) \). We can simply simulate \( M_f \) on \( w \), ignoring all its outputs and just keeping a count of the number of output bits, until it prints the \( c \)-th bit of \( f(w) \). Every time the value of \( c \) changes, we repeat this step and simulate \( M_f \) on \( w \) from scratch until we reach the new value of \( c \). Note that with this approach we always know the bit under the input-tape-head of \( M_Y \) on \( f(w) \), and thus can proceed with the simulation. \( \square \)

The logspace mapping reduction is the strongest form of a reduction that we have seen so far. Since \( L \subseteq P \), it implies in particular the existence of a poly-time reduction.

Figure 5.1 summarizes the inclusion relation between the complexity classes we have seen so far.

**Exercises**

**Exercise 5.1.** Prove that the following function \( f : \{0,1\}^* \to \{0,1\}^* \) defined as \( f(x) = 0^{2|x|} \) can be computed in polynomial space.

**Exercise 5.2.** Prove that the following language is in \( L \).

\[
\{0^n1^n : n \in \mathbb{N}\}.
\]

**Exercise 5.3.** Prove that the following language is in \( L \).

\[
\{ww^R : w \in \{0,1\}^*\},
\]

where \( w^R \) is the reverse of \( w \).

**Exercise 5.4.** Prove that the addition function is in \( L \). That is \( f(x,y) = x + y \), where \( x, y \) and \( x + y \) are in binary basis.

**Exercise 5.5.** First write a pseudo-code which shows \( QSAT \in PSPACE \). Next prove that \( QSAT \) is \( PSPACE \)-complete under polytime reductions.
Figure 5.1: A summary of the inclusion relation between some complexity classes
Chapter 6

Savitch’s Theorem

In the previous lecture, we introduced two important complexity classes $L$ and $NL$. Furthermore, we discussed log space mapping reductions, and showed that if $X \leq^L Y$ and $Y \in L$, then $X \in L$. Now let us discuss an important language, 

$PATH = \{ \langle G,s,t \rangle : G \text{ is a directed graph with an } s \rightarrow t \text{ path} \}$.

Here we can assume that $G$ is given to us as an adjacency matrix $A_G$, where $A_{uv} = 1$ if and only if there is an edge from $u$ to $v$, and $A_{uv} = 0$ otherwise. The next theorem shows that $PATH \in NL$.

**Theorem 6.1.** The language $PATH$ is in $NL$.

**Proof.** To show that $PATH \in NL$, on an input $\langle G,s,t \rangle$, we start from the initial vertex $s$, and at each step, non-deterministically guess one of the neighbours of the current vertex as the next vertex. If we reach $t$ in at most $|V(G)|$ steps, then we accept the input, otherwise we reject. Obviously if there is an $s \rightarrow t$ path, then there will be a sequence of non-deterministic choices that lead to $Accept$, and on the other hand if there are no $s \rightarrow t$ paths, then no matter how we make the non-deterministic choices, the input is going to be rejected.

Finally note that since we only need to keep track of the current vertex and just the number of the visited vertices, this algorithm requires only logarithmic space. □

$PATH$ is an important language in $NL$. The following theorem shows that in fact every language in $NL$ can be log-space reduced to $PATH$.

**Theorem 6.2.** The language $PATH$ is $NL$-complete under the log-space reductions.

**Proof.** We have already shown that $PATH \in NL$. It remains to prove the completeness. Consider an arbitrary $X \in NL$, and let $M$ be a non-deterministic Turing Machine that decides $X$ in logarithmic space $S(n)$. Given an input $w$, construct an instance of $\langle G,s,t \rangle$ of $PATH$ as in the following. The nodes of $G$ are all the possible configurations of $M$ with $w$ on the input-tape, and the work-tape of size at most $S(|w|)$. Note that each such configuration can be represented by at most $c \log(n)$ bits for some constant $c$, encoding the location of the heads, and the content of the work-tape. For two configurations $c_1$ and $c_2$, we include the edge $(c_1,c_2)$ in the graph if $c_2$ is one of the possible next configurations of $M$ if we start from $c_1$ and make a non-deterministic choice. We assume that $M$ is modified so that it has a unique accepting configuration. We call this configuration $t$, and let $s$ be the starting configuration.

Obviously this is a mapping reduction from $X$ to $PATH$ as $M$ accepts $w$ if and only if there is an $s \rightarrow t$ path in $G$. Furthermore the reduction can be implemented using logarithmic space. We will show how the nodes and the edges of $G$ can be listed by a log space algorithm. We go through all the strings of size $c \log(n)$ one by one, and for each one we test to see if it is a valid potential configuration. To list the edges, we generate pairs of vertices $(c,c')$ one by one, and for each pair we check to see whether there is a transition in $M$ that converts $c$ to $c'$. If there is, then we output $(c,c')$ as an edge, and otherwise output it as a non-edge.

□

It is not known whether $PATH$ is in $L$ or not. By Theorem 6.2 if $PATH \in L$, then $NL = L$. This is a very interesting question, as unlike with the $P$ versus $NP$ question, there are fewer reasons to believe that $L$ and $NL$ are necessarily different. Towards answering these questions, in 2005, Riengold in a breakthrough paper [Rei08] showed that the undirected version of $PATH$ belongs to $L$. That is $UPATH \in L$, where

$UPATH = \{ \langle G,s,t \rangle : G \text{ is an undirected graph with an } s \rightarrow t \text{ path} \}$. 

27
Savitch’s theorem

In this section, we discuss the relation between the deterministic and non-deterministic space complexity. In particular, we prove Savitch’s theorem which shows that the gap between the deterministic and non-deterministic space complexity is not very large. For time complexity we know that a non-deterministic Turing Machine with running time $T(n)$ can be simulated by a deterministic Turing Machine with an exponential blow-up in the running time $2^{O(T(n))}$. It is widely believed that this gap is necessary, and that NP-complete problems such as SAT that can be solved in polynomial non-deterministic time require exponential deterministic time. However for the space complexity, Savitch [Sav70] showed that the simulation can be done in a more efficient manner.

Theorem 6.3 (Savitch’s Theorem). We have

$$NL \subseteq SPACE(\log^2(n)).$$

Proof. By Theorem 6.2 the statement is equivalent to showing that $\text{PATH} \in SPACE(\log^2(n))$. Consider an instance $\langle G, s, t \rangle$, where $G$ is a graph with $m$ vertices. If we try to imitate the algorithm of Theorem 6.1, in order to be able to know that all the paths have been checked, we need to remember the list of the vertices that have been visited at any point during the execution of the algorithm. However this requires an $\Omega(|V|)$ amount of space, which is much larger than the desired $O(\log^2(n))$. Instead we will take a different approach.

The idea is to devise a recursive algorithm whose maximum recursion depth is logarithmic: Obviously $\langle G, s, t \rangle \in \text{PATH}$ if and only if there is a path of length at most $m$ from $s$ to $t$. We will do this into two subtasks For every vertex $w$, we see if there is an $s \rightarrow w$ path of length at most $m/2$, and a $w \rightarrow t$ path of length at most $m/2$. If they both exist, then we know there is an $s \rightarrow t$-path. Now to solve those two subproblems, we can repeat the same idea, and look for paths of length $m/4$, and then $m/8$, etc. Note that the maximum depth of this recursion is $O(\log(m))$, and furthermore, for each level, we need to remember one vertex. So in total at any point, the number of stored data is $O(\log(m)^2)$ as desired.

Algorithm 3: The subroutine that searches for the existence of a $u \rightarrow v$ path of length at most $k$.

**Data:** On input $w = \langle G, u, v, k \rangle$.

if $k \leq 1$ then
    if $uv \in E$ or $u = v$ then
        ACCEPT;
    else
        REJECT;
    end
else
    for $w \in V$ do
        if $\text{Path}(G, u, w, \lceil k/2 \rceil) = \text{ACCEPT}$ and $\text{Path}(G, w, v, \lceil k/2 \rceil) = \text{ACCEPT}$ then
            ACCEPT;
        end
    end
end
REJECT;

Corollary 6.4. We have $\text{PSPACE} = \text{NPSPACE}$.

Proof. The same proof as in Savitch’s theorem can be used to show that

$$\text{NSPACE}(n^k) \subseteq \text{SPACE}(n^{2k}),$$

which in particular implies $\text{PSPACE} = \text{NPSPACE}$.

What is the running time of the algorithm of Theorem 6.3? Note that the running time corresponds to the recursion $T(m) \leq 2m T(m/2)$ which solves to $m^{O(\log(m))} = 2^{O(\log(n)^2)}$. So on the one hand, we have an algorithm that runs in the small space of $O(\log^2(n))$ and rather slow time of $2^{O(\log(n)^2)}$. On the other hand, we can simply run a depth first search that has an efficient time of $O(n \log(n))$ but requires a large space of $O(n)$. Can one achieve the best of both worlds?

Problem 6.5 (Open Problem). Is there an algorithm for $\text{PATH}$ that runs in time $n^{O(1)}$ and uses space $\log^{O(1)}(n)$?
Exercises

Exercise 6.1. Prove that UPATH $\leq_t$ PATH.

Exercise 6.2. A randomized Turing Machine is a Turing Machine that has an additional infinite randomness tape. Initially the tape is filled with an infinite sequence of independent unbiased random bits. I other words, for every cell, a fair coin is flipped independently to decide the value of the cell.

Let $C$ be the class of problems that can be solved with a deterministic \textit{log-space one-sided randomized} algorithm: That is there is a randomized Turing Machine that it never accepts incorrectly, but is allowed to reject incorrectly with probability at most $1/3$. Prove that $C = \text{NL}$.

Let $RL$ be the class of problems that can be solved with a deterministic \textit{log-space one-sided randomized} algorithm in \textit{polynomial time}. Conclude that $L \subseteq RL \subseteq \text{NL}$. It is not known whether $L \neq RL$, but many believe that they might be equal.

Exercise 6.3. Recall that a Boolean formula is an expression involving Boolean variables, and Boolean operations $\land, \lor, \neg$ (but no quantifiers $\exists, \forall$). Two Boolean formulas are called equivalent if they are on the same set of variables, and are true on the same set of truth assignments. Consider the following language

$$\text{MinFormula} = \{ \phi : \phi \text{ has no shorter equivalent formula} \}.$$

- Prove that $\text{MinFormula} \in \text{PSPACE}$.
- Prove that if $P = \text{NP}$, then $\text{MinFormula} \in P$.

Exercise 6.4. Prove that the following language is in \text{PSPACE}.

$$\{ \langle r, t \rangle : r \text{ and } t \text{ are equivalent regular expressions} \}.$$

Exercise 6.5. Prove that the set of properly nested parenthesis is in $L$. For example $((()))$ is properly nested, while $())($ is not.

Exercise 6.6. Prove that the following is in $L$.

$$\{ G : G \text{ is an undirected graph with at least a cycle} \}.$$

Exercise 6.7. An $\text{NL}$-certifier is an $\text{ACCEPT/REJECT}$ Turing Machine with three tapes:

- Input Tape: (Read-Only).
- Working Memory: (Read-Write) It is small $O(\log(n))$, where $n$ is the input size.
- Certificate/Proof Tape: (Read-Once) The tape-head cannot move to the left (every memory cell gets to be read at most once).

Prove that $L \in \text{NL}$ if and only if there is an $\text{NL}$-certifier $M$ such that

$$x \in L \iff \exists y, \ M(x, y) = \text{ACCEPT}.$$

Exercise 6.8. Prove that if in the definition of the $\text{NL}$-certifier if relax the read-once condition on the certificate-tape to an ordinary read-only tape, then we obtain the class $\text{NP}$.
Chapter 7

NL versus coNL: Immerman-Szelepcsényi Theorem

In the previous lecture, we showed that $L \subseteq NL \subseteq L^2$, where $L^2 = SPACE(\log^2(n))$. In this lecture, we consider the relationship between $NL$ and $coNL$. It is not known whether $L = NL$, but if true, it would imply $NL = coNL$. The Immerman-Szelepcsényi Theorem shows that the latter is true regardless. Recall that the directed PATH problem is $NL$-complete under the log-space reductions, and consequently its complement $PATH^c$ is $coNL$-complete. Hence, in order to show that $NL = coNL$, it suffices to prove that $PATH^c \in NL$. This is rather nontrivial. The power of a non-deterministic Turing Machine is its ability to "guess" a proper certificate. For example to show $PATH \in NL$, simply for $|V(G)|$ steps, we guessed the next vertex until we reach $t$.

To see the difficulty of this problem, imagine someone with a very poor memory. To convince this person that there is a path from $s$ to $t$ in a graph $G$, it suffices to show the edges of an $st$-path one by one, and the person can verify those without needing to memorize anything else. But how can we convince someone with poor memory that there are no $st$-paths?

**NL versus coNL: Immerman-Szelepcsényi Theorem**

To present the proof of the Immerman-Szelepcsényi Theorem, it is useful to think of the $NL$ algorithms in the language of verifiers. We refer the reader to Exercise 6.7, where the reader is asked to prove that $L \in NL$ if and only if there is an $NL$-certifier $M$ such that $x \in L \iff \exists y, M(x, y) = \text{Accept}$, where an $NL$-certifier is an $\text{Accept/Reject}$ Turing Machine with three tapes:

- **Input Tape:** (Read-Only).
- **Working Memory:** (Read-Write) It is small $O(\log(n))$, where $n$ is the input size.
- **Certificate/Proof Tape:** (Read-Once) The tape-head cannot move to the left (every memory cell gets to be read at most once).

Note that here the certificate tape corresponds to the non-deterministic decisions made by the non-deterministic Turing Machine, and as a consequence it is natural to assume that it is read-once.

Immerman [Imm88] and Szelepcsényi [Sze88] independently proved that $NL = coNL$.

**Theorem 7.1.** We have $NL = coNL$.

**Proof.** By $NL$-completeness of $PATH$, it suffices to prove that $\text{NonPATH} = \{(G, s, t) : G$ is a directed graph with no $s \rightarrow t$ paths$\} \in NL$.

Let $m = |V(G)|$, and let us label the vertices as $u_1, \ldots, u_m$. The input size is roughly $n = \Theta(m^2)$ if we present the graph with its adjacency matrix, and thus $\log(n) = \Theta(\log(m))$. Note that each vertex can be represented with $\log(m)$ bits. For $k = 0, \ldots, m$, define $A_k = \{u : \exists u\text{-path of length } \leq k\}$.  

31
Now let us try to build a “certificate” for not containing any st-paths. For each $k = 0, \ldots, n$, define
\[ P_k = [B^k_1, \ldots, B^k_m], \]
where the blocks are defined as
\[ B^k_i = \begin{cases} \{ \text{an } su_i\text{-path of length } \leq k \} & u_i \in A_k \\ \{ \} & u_i \notin A_k \end{cases} \]
That is, when $u_i \in A_k$, then the evidence is provided to us, whereas if $u_i \notin A_k$, then $B^k_i$ includes no evidence for this claim. We make two key observations

1. If we know the size $c_k = |A_k|$, then we can verify the correctness of $P_k$ in log-space. Indeed we can verify the memberships $u_i \in A_k$ by looking at their provided $su_i$-paths, and as we do so we keep a count of them. If the total count matches $c_k$, then we will know that the non-memberships $u_i \notin A_k$ are also given to us correctly. The issue remains that we cannot simply rely on the certificate to provide the number $c_k$ to us, as we need to be able to verify its correctness.

2. If we have a fixed vertex $v$ in mind, then as we are verifying $P_k$ (assuming we know $c_k$), we can find out whether $v \in A_{k+1}$ or $v \notin A_{k+1}$. Each time we see a vertex $u_i \in A_k$, we check to see if it has an edge to $v$. If it has, then $v \in A_{k+1}$. If we do not find such a $u_i v$ with $u_i \in A_k$, and also see that $v \notin A_k$, then we shall know that $v \notin A_{k+1}$.

**Algorithm 4:** The subroutine that given $c_k$, verifies the correctness of $P_k$ and $v \in A_{k+1}$ in log-space.

Data: On input $c_k$, $k$, $P_k$, $v$, .

Initialize $c := 0$, and $B := \langle v \notin A_{k+1} \rangle$

for $i = 1, \ldots, n$ do

if $\langle u_i \notin A_k \rangle$ according to $P_k$ then

Verify the claim by following the provided path;

Increase $c$;

if $u_i v \in E$ or $v = u_i$ then

$B := \langle v \in A_{k+1} \rangle$

end

end

Verify that $c = c_k$ and return $B$.

It remains to compute $c_k = |A_k|$ from the certificate. We have shown that given $c_k$ and a correct $P_k$, we can find out whether any given $v$ belongs to $A_{k+1}$ or not. This should allow us to compute $c_{k+1}$ from $c_k$ and $P_k$ except for one issue. We are only allowed to read $P_k$ once as the certificate-tape is read-once. With a meagre memory of $O(\log(n))$ we cannot detect all the vertices that belong to $A_{k+1}$ in one reading of $P_k$. There is a simple trick to overcome this: repeat each $P_k$ several times. The full certificate that is provided to us is going to be the following:

\[ P = [P_0 \ldots P_0, P_1 \ldots P_1, \ldots, P_m \ldots P_m]. \]

Now computing $c_{k+1}$ from $c_k$ is straightforward. Indeed as we read the $m$ copies of $P_k$, we can use the $j$-th copy to find out whether $u_j \in A_{k+1}$. Note that $c_0 = 1$ and $A_0 = \{s\}$. We start with $c_0 = 1$, and for each $k = 0, \ldots, m-1$, we compute $c_{k+1}$ from $c_k$.

Let us remark that we cannot require all the $m$ repetitions of $P_k$ to be identical, as a log-space verifier does not have sufficient memory to verify it. All that is required is that each individual $P_k$ correctly provides an $su$-paths of length at most $k$ for each $v \in A_k$.

Finally note that there are no st-paths if and only if $t \notin A_m$. So given a valid $P$ we can verify its correctness and then see that $t \notin A_m$. On the other hand, if $P$ is not valid, our algorithm will detect it, and reject it.

**Exercises**

**Exercise 7.1.** Prove that the following is in NL.

\[ \{\langle G \rangle : G \text{ is a bipartite graph} \}. \]
Exercise 7.2. Where is the error in the following incorrect proof that NonPATH ∈ NL?

Given an input \( \langle G, s, t \rangle \), it is easy to see that there are no \( st \)-paths in \( G \) if and only if there exists a partition of the vertices into two parts \( A \) and \( B \) such that \( s \in A \), \( t \in B \), and there are no edges going from \( A \) to \( B \). Let the vertices of \( G \) be \( u_1, \ldots, u_m \), and let \( x = (x_1, \ldots, x_n) \in \{0, 1\}^n \) be such that \( x_i = 0 \) if \( u_i \in A \) and \( x_i = 1 \) if \( u_i \in B \). To verify that \( \langle G, s, t \rangle \in \text{NonPATH} \), the verifier is going to be provided with a certificate \( C \) consisting of \( k \) copies of \( x \), where \( k \) is the number of edges in \( G \):

\[
C = [x, x, \ldots, x].
\]

Now as the verifier reads the \( i \)-th \( x \), it verifies that the edge \( e_i \) does not go from \( A \) to \( B \). It will also use \( x \) to verify that \( s \in A \) and \( t \in B \). Once it finishes reading all of \( C \) it will have verified that none of the edges \( e_i \) goes from \( A \) to \( B \). Thus it will successfully verify that \( \langle G, s, t \rangle \in \text{NonPATH} \).
Chapter 8

Ladner’s Theorem: NP-completeness

Shortly after Cook and Levin introduced the notion of NP-completeness, Karp [Kar72] proved that several fundamental combinatorial optimization problems are NP-complete, and subsequently many other NP-complete problems were discovered by other mathematicians. By 1979 the list of the known NP-complete problems was so extensive that Gary and Johnson compiled a compendium of known NP-complete problems in a 330 page book [GJ79]. It is curious that most natural problems in the class NP seem to be either NP-complete or fall into P. A partial explanation for this phenomena is given by dichotomy theorems. Such theorems state that for certain classes of problems, each problem is either in P or NP-complete. Consider the following combinatorial problem as an example: for a fixed undirected graph $H$, the problem of deciding whether a given graph $G$ has a homomorphism to $H$ is called the $H$-colouring problem. Hell and Nesetril [HN90] proved a dichotomy theorem for the $H$-coloring problem, stating that if $H$ is bipartite, then the $H$-coloring problem is in P, and otherwise it is NP-complete. The $H$-coloring problem and many other natural combinatorial problems can be expressed as part of a broad class of problems referred to as constraint satisfaction problems. This class of problems is known to be NP-complete in general, but certain restrictions on the form of the constraints can ensure polynomial-time tractability. A dichotomy theorem for constraint satisfaction problems was conjectured by Feder and Vardi in 1998 [FV98], and proved after about twenty years [Zhu20]. A natural question arises:

Is it the case that NP consists only of NP-complete problems and the problems in P?

Ladner’s theorem [Lad75] refutes this by showing that if $P \neq NP$, then there are problems in NP that are neither NP-complete nor belong to P.

**Theorem 8.1** (Ladner’s theorem [Lad75]). *If $P \neq NP$, then there is a problem $L \in NP \setminus P$ that is not NP-complete.*

The proof presented below is due to Russell Impagliazzo. We will unfortunately not be able to find a natural language that is NP-intermediate assuming $P \neq NP$. In fact, to this day no such decision problem is known, and there are very few candidates for such problems such as Factorization and Graph Isomorphism. Instead, the idea of the proof is to take an NP-complete problem such as SAT, and use padding to make it strictly easier, but not easy enough to fall in P. In particular, we will consider a padded version of SAT, $\{(\phi,1^{f(|\phi|)}) \mid \phi \in SAT\}$ for some carefully chosen relatively large $f(|\phi|)$.

**Warm-up: Proof assuming the Exponential Time Hypothesis**

We first give a proof of Ladner’s Theorem under the stronger assumption that the exponential time hypothesis is true. This hypothesis states that 3SAT cannot be solved in sub-exponential time. Note that the exponential time hypothesis, if true, would imply that $P \neq NP$, but it is potentially a stronger statement as it assumes a concrete and very strong lower-bound on the running time of any algorithm that solves 3SAT.

Now consider the language

$$L = \{(\phi,1^{h(|\phi|)}) \mid \phi \text{ is a satisfiable 3CNF}\},$$

where $h(n) = n^{\log(n)} = 2^{\log^2 n}$, so that for a 3CNF $\phi$ of size $n$, the size of the new input $\langle \phi,1^{h(|\phi|)} \rangle$ is $\Theta(2^{\log^2 n})$, which is super-polynomial and sub-exponential in $n$.

- $L \in NP$: This obviously is true as a truth assignment that satisfies $\phi$ can be used to verify in polynomial time that $\langle \phi,1^{h(|\phi|)} \rangle \in L$. 

• $L \not\in \text{P}$: Note that if $L \in \text{P}$, then given a 3CNF $\phi$, we can decide in time $2^{O(\log^2 n)}$ whether $\phi \in 3\text{SAT}$ by running the polynomial time algorithm of $L$ on $\langle \phi, 1^{h(|\phi|)} \rangle$. This contradicts the exponential time hypothesis as $2^{O(\log^2 n)}$ is sub-exponential.

• $L$ is not NP-complete: If it is, then there is a polynomial time reduction from 3SAT to $L$. In particular there exists a polynomial time computable $g : \Sigma^* \rightarrow \Sigma^*$ that given a 3CNF $\phi$ of size $n$ produces $\langle \psi, 1^{h(|\psi|)} \rangle = g(\phi)$ such that

$$\phi \in 3\text{SAT} \iff \langle \psi, 1^{h(|\psi|)} \rangle \in L \iff \psi \in 3\text{SAT}.$$ 

Since this is a polynomial-time reduction, we have $|g(\phi)| = n^{O(1)}$, which shows that $\psi$ must be small enough to satisfy $h(|\psi|) \leq |g(\phi)| = n^{O(1)}$. Recalling $h(k) = 2^{\log^2 k}$, we conclude $|\psi| \leq 2^{\log^2 n} = o(n)$, which is in fact much smaller than $n$. Note that now in order to see whether $\phi \in 3\text{SAT}$ we can try all the truth assignments to the much smaller 3CNF $\psi$. This will require time $O(2^{\sqrt{n^2}}) = 2^{o(n)}$, which is sub-exponential in $n$.

**The full proof of Ladner’s Theorem**

The difficulty in proving Ladner’s theorem without the exponential time hypothesis is that we cannot assume any concrete super-polynomial lower-bound on the running time of the algorithms that solve 3SAT. The assumption $\text{P} \neq \text{NP}$ only tells us that every such algorithm must run in super-polynomial without proving us with any concrete functions.

**Proof of Theorem 8.1.** Note that for every polynomial $p(n)$, there exists a constant $k \in \mathbb{N}$ such that $p(n) \leq n^k + k$ for all $n \geq 0$. Consequently we can construct an enumeration\(^1\) of Turing Machines $M_1, M_2, \ldots$ that are clocked to guarantee that $M_k$ always halts before time $n^k + k$, and furthermore for every language $L \in \text{P}$, there is some $k$ such that $M_k$ decides $L$.

For any function $H : \mathbb{N} \rightarrow \mathbb{N}$, define the language

$$\text{SAT}_H = \{ \langle \phi, 1^{h(|\phi|)} \rangle : \langle \phi \rangle \in \text{SAT} \},$$

so that a CNF $\phi$ is padded to a length $|\phi| + H(|\phi|)$ input for $\text{SAT}_H$. Provided that $H$ can be computed efficiently, this blow-up in the input size is going to make the problem easier than $\text{SAT}$ because the running time is always measured with respect to the input length. On the other hand, we have to choose $H$ carefully so that $\text{SAT}_H \not\in \text{P}$ if $\text{SAT} \not\in \text{P}$.

We will define the function $H : \mathbb{N} \rightarrow \mathbb{N}$ as in the following. Define $H(n) = 1$ for all $n \leq 100$. For $n > 100$, we define $H(n)$ to be the smallest $k < \log \log n$ such that $M_k$ decides $\text{SAT}_H$ correctly on all inputs of length at most $\log n$, and we set $H(n) = \log \log n$ if such $k$ does not exist. Since, we do care about the complexity of $H$ itself, we will also present an algorithm that recursively computes $H$.

$H(n)$:

1. Recursively compute $H(1), \ldots, H(\log(n))$, and store them in $h(1), \ldots, h(\log(n))$.

2. For each $k = 1$ to $\log \log n$ do
   
   (a) $\text{flag} = \text{True}$.
   
   (b) For every $x$ of length $|x| \leq \log(n)$:

   i. Decide whether $x \in \text{SAT}$ by checking that $x = \langle \phi, 1^{h(|\phi|)} \rangle$ for some $\phi \in \text{SAT}$ with $|\phi| \leq \log(n)$, and iterating over all assignments to the inputs of $\phi$ to determine its satisfiability (Note that we know what $h(|\phi|) = H(|\phi|)$ by the first step.).
   
   ii. Run $M_k$ on input $x$ and if $M_k(x)$ decides incorrectly whether $x \in \text{SAT}_H$ then set $\text{flag} = \text{False}$.

   (c) If $\text{flag} = \text{False}$, then return $\log \log n$.

3. return $\log \log n$.

An important property of $H$ that is immediate from its definition, is that $H$ is a nondecreasing function. We proceed with some easy claims.

---

\(^1\)Here and throughout the notes, when we mention such an enumeration, we assume that given $i$, $M_i$ can be constructed in log-space. This for example can be achieved by looking at the binary representation of $i$, and interpreting it as a potential Turing Machine description.
Claim 8.2. $H(n)$ can be computed in time $n^{O(1)}$.

Note that we are not claiming $H$ is polynomial-time computable, since that would require a running time that is polynomial in $\log n$.

Proof. Let $T(n)$ denote the running time of computing $H(n)$ using the above algorithm. Within each iteration for a fixed $x$, Step (i) takes $n^{O(1)}$ time as there are only $O(n)$ possible assignments to the variables of $\phi$ (as $|\phi| \leq \log n$), and for each assignment, $\phi$ can be evaluated in time $O(|\phi|) = O(\log n)$. Similarly, the contribution from Step (ii) within each iteration is bounded by the running time of $M_k$ on an input of size at most $\log(n)$, which is bounded by $O((\log(n))^k) \leq O((\log(n)\log\log(n))) = o(n)$. Finally, observing that there are $O(n)$ choices of $x$, with $|x| \leq \log n$, we arrive at the recursion $T(n) \leq T(1) + \ldots + T(\log(n)) + n^{O(1)}$ which solves to $T(n) = n^{O(1)}$. \hfill \Box

Claim 8.3. We have $SAT_H \in NP$.

Proof. Since $H(n)$ can be computed in $n^{O(1)}$, given an input $w$, we can check (even deterministicaly) in polynomial time whether it is of the right format $w = \langle \phi, 1^{\lceil H(|\phi|) \rceil - |\phi|} \rangle$, and then use the fact that $SAT \in NP$. \hfill \Box

Claim 8.4. If $P \neq NP$ then $SAT_H \notin P$, and $\lim_{n \rightarrow \infty} H(n) = \infty$.

Proof. Suppose for contradiction that $SAT_H \in P$. Then there exists a $k < \infty$ such that $M_k$ decides $SAT_H$ correctly on all inputs. This means that for every $n$ with $\log \log(n) \geq k$, we have $H(n) \leq k = O(1)$. Then the mapping

$$f : \langle \phi \rangle \mapsto \langle \phi, 1^{\lceil H(|\phi|) \rceil - |\phi|} \rangle,$$

is a polynomial time mapping reduction from $SAT$ to $SAT_H$. This together with $SAT_H \in P$ implies $SAT \in P$, which is in contradiction with $NP \neq P$.

Now suppose for contradiction that $\lim_{n \rightarrow \infty} H(n) < \infty$. Then since $H$ is a nondecreasing function, there exists $k, n_0$, such that $H(n) = k$ for all $n > n_0$. This implies that $M_k$ decides $SAT_H$ for all inputs. Since $M_k$ is a polynomial-time algorithm, this contradicts the previous assertion that $SAT_H \notin P$. \hfill \Box

Claim 8.5. If $P \neq NP$ then $SAT_H$ is not NP-complete.

Proof. Suppose that $SAT_H$ is NP-complete. Then $SAT \leq_p SAT_H$, and hence there is a polynomial-time computable map $f$ such that

$$\phi \in SAT \iff f(\phi) \in SAT_H.$$

Since $f$ can be computed in polynomial time, there exists a constant $c$ such that $|f(\phi)| \leq |\phi|^c$, for every $\phi$ of size at least 2. For a CNF $\phi$, define $\pi(\phi) := \psi$ if

$$f(\phi) = \langle \psi, 1^{\lceil H(|\psi|) \rceil - |\psi|} \rangle,$$

and otherwise $\psi_\phi = \perp$ (\(\perp\) representing the constant contradiction CNF). Note that

$$\phi \in SAT \iff \pi(\phi) \in SAT.$$

Since $\lim_{n \rightarrow \infty} H(n) = \infty$, there exists $n_0$ such that $H(n) > c$ for all $n > n_0$. Hence if $|\phi| > n_0$, then since

$$|\pi(\phi)|^c < |\pi(\phi)|^{H(|\pi(\phi)|)} = |f(\phi)| < |\phi|^c,$$

and thus $|\psi_\phi| < |\phi|$.

Since $\pi$ is polynomial-time computable, we have spent a polynomial amount of time, and have reduced the satisfiability of $\phi$ to the satisfiability of a shorter CNF $\pi(\phi)$. We can repeat this process by at most $|\phi|$ many times to eventually obtain a formula $\eta = \pi(\pi(\cdots(\phi)))$ that is of size at most $n_0 = O(1)$. We can then solve the satisfiability of $\eta$ in $O(1)$. This would lead to a polynomial-time algorithm for $SAT$ which contradicts our assumption $NP \neq P$. \hfill \Box

The above claims imply Theorem 8.1.

Exercises

Exercise 8.1. Prove that if $P \neq NP$, then there is an infinite sequence of languages $L_1, L_2, \ldots$ in $NP$ such that neither of them is in $P$, and moreover $L_{i+1} \leq_p L_i$ but $L_i \nleq_p L_{i+1}$, for all $i$. 

37
Chapter 9

Oracles and relativization

In general terms, an algorithm is a systematic procedure that produces in a finite number of steps, the answer to a question or the solution of a problem. What is allowed to be a step could be left for interpretation. In the first half of the 20th century, the works of Church and Turing among others lead to the understanding (known as the Church-Turing thesis) that all the natural and practical computational models can be simulated by Turing Machines. In other words, a handful of simple basic steps suffice to implement even the most sophisticated algorithms. However, we know that Turing Machines have their limitations, in that there are languages such as

\[
\text{HALT}_{TM} = \{ \langle M, w \rangle : M \text{ halts on } w \},
\]

that cannot be decided by any Turing Machine.

While this is a limitation in practice, theoretically one can define a computational model that is capable of solving this problem by simply allowing queries of the form “\( \langle M, w \rangle \in \text{HALT}_{TM}? \)” as part of the legal basic steps of an algorithm. We can formalize such models through the notion of an oracle. Informally an oracle for a language \( B \) is a hypothetical black-box that is able to produce a solution for any instance of the problem in a single computation step. An oracle Turing Machine \( M^B \) is a Turing Machine that has access to such an oracle. Intuitively \( M^B \) is a Turing Machine that is further allowed to make queries whether \( y \in B \) for different strings \( y \). Each such query takes only one computation step, and will always be answered correctly. Note that in this definition \( B \) can be any language of any complexity, and in fact even undecidable problems, such as the halting problem, can be used to define oracle Turing Machines.

**Definition 9.1.** An oracle for a language \( B \) is a device that is capable of reporting whether any string \( w \) is a member of \( B \). An oracle Turing machine \( M^B \) is a modified Turing Machine that has the additional capability of querying an oracle for \( B \). This is implemented by an additional write-only query tape, and a special state that is called the query state. Whenever \( M^B \) enters the query state, it receives an answer to whether the content of the query tape is in \( B \) or not. The query tape resets immediately. These happen instantaneously so that each query to the oracle counts as a single computation step.

**Remark 9.2.** We have assumed that the query tape is a write-only tape, and thus the oracle Turing Machine cannot use it as a work space. This allows us to define the space complexity of \( M^B \) as the amount of the used work space, which could be much smaller than the sizes of the queries. This is important as it allows us to decide \( B \) with an oracle Turing Machine \( M^B \) in time and space \( O(1) \).

**Remark 9.3.** Note that the description of an oracle Turing Machine is independent of the oracle that it uses. It is only during the computation that the oracle plays a role. That is, we can consider a description of an oracle Turing Machine \( M \) without knowing the oracle. Using two different oracles \( B_1 \) and \( B_2 \) with this will result in two different oracle Turing Machines \( M^{B_1} \) and \( M^{B_2} \).

The concept of oracle Turing Machines arises naturally in the study of reducibility of languages: Can we solve \( A \) if we know how to solve \( B \)? Indeed oracle Turing Machines can be used to define some of the most liberal notions of reducibility.

- We say \( A \) is Turing reducible to \( B \) (denoted \( A \leq_T B \)) if there is an oracle Turing Machine \( M^B \) that decides \( A \).
- We say \( A \) is polynomial time Turing reducible (a.k.a. Cook reducible) to \( B \) (denoted \( A \leq_P^T B \)) if there is a polynomial time oracle Turing Machine \( M^B \) that decides \( A \).
Note that if $A \leq_T B$ and $B$ is decidable, then $A$ must also be decidable because we can replace the oracle part of $M^B$ with an ordinary Turing Machine that decides $B$. Namely, every time a query of the form “$y \in B$?” is made, we run the algorithm that decides $B$ to find the answer to the query. Similarly if $A \leq^p_B B$, and $B \in P$, then $A \in P$, since in this case the oracle queries can be computed by an ordinary Turing Machine in polynomial time.

Oracle-based notions of reducibility are more liberal than mapping reducibility notions: To answer questions of the form “$x \in A$?”, not only we can query the oracle $B$ more than once, but the queries can also be adaptive, i.e. depend on the answer to previous queries. See Exercises 9.1, 9.2, 9.3 for some rigorous comparison between these reductions.

**Relativization**

We can relativize most of the standard complexity classes to a language $B$ simply by adding $B$ as an oracle. For example, one might wonder, which problems are decidable relative to $B$? Which problems can be solved in polynomial time relative to $B$?

**Definition 9.4 (Semi-formal).** Let $C$ be the class of languages that can be decided by Turing Machines with certain resources. Then $C^B$ is the class of languages that can be solved by $B$-oracle Turing Machines with the same resources. More generally for a set $B$ of languages, we define $C^B = \bigcup_{B \in B} C^B$.

Note that $P^B$ is exactly the set of languages $A$ that are Cook reducible to $B$.

$$P^B := \{A : A \leq^C_B B\}.$$

**Remark 9.5 (Warning).** Note that Definition 9.4 is not very formal. It provides a template to define analogous complexity classes with respect to oracles by taking the definition of the complexity class and replacing algorithms with oracle algorithms in them. As a result, if a complexity class has two different definitions (say $NL = coNP$), then depending on what definition one chooses, one might end up in two different complexity classes. In other words it is possible that $C = D$ while $C^B \neq D^B$ for some oracle $B$. See Exercise 9.8.

Next, let us consider some easy examples. We have $P^P = P$ and $NP^P = NP$ as we can replace the oracle for any language in $P$ with a polynomial time algorithm that decides it.

We have $P^{SAT} = P^{NP}$, and in general, the following preposition is easy to prove.

**Proposition 9.6.** Suppose that a language $B$ is complete for some class $D$ under some notion of reduction. Then $C^B = C^D$ provided that the machines in the definition of $C$ can execute that notion of reduction.

Note that $NP$ and $coNP$ are both subsets of $P^{NP}$, so it is very likely that $P^{NP} \neq NP$.

We can also consider the clearly stronger class $NP^{NP}$, which turns out to belong to the second level of the polynomial hierarchy.

**Proposition 9.7.** We have $NP^{NP} = \Sigma_2$.

**Proof.** First note that if $L \in \Sigma_2$, then

$$x \in L \iff \exists y_1 \forall y_2 M(x, y_1, y_2) = \text{Accept},$$

where $M$ is a deterministic Turing Machine that runs in polynomial time in $|x|$. Let

$$B = \{(x, y_1) : \exists y_2 M(x, y_1, y_2) = \text{Reject}\},$$

and note that $B \in NP$ and furthermore

$$x \in L \iff \exists y_1 (x, y_1) \notin B.$$

This shows that $L \in NP^B$, and consequently $\Sigma_2 \subseteq NP^{NP}$. We leave the proof of the opposite direction as an exercise. See Exercise 9.5.

**Definition 9.8 (Oracle Definition of the Polynomial Hierarchy).** Define $\Sigma_0 = \Pi_0 = \Delta_0 = P$, and for every $i \geq 0$, let

- $\Delta_{i+1} = P^{\Sigma_i}$;
- $\Sigma_{i+1} = NP^{\Sigma_i}$;
- $\Pi_{i+1} = coNP^{\Sigma_i}$.

In Exercise 9.6 you are asked to prove that this definition of $\Sigma^p_i$ and $\Pi^p_i$ is consistent with the earlier definition of these classes involving quantifiers and polynomial-time verifiers. Note also that here we defined a new set of complexity classes $\Delta_i$. It is straightforward to see that $\Sigma_i \cup \Pi_i \subseteq \Delta_{i+1} \subseteq \Sigma_{i+1} \cap \Pi_{i+1}$.
Relativization as a proof-barrier

Many of the known lower-bound techniques in the theory of computation and complexity theory can be relativized to any oracle model: they hold even if we allow access to an oracle B. Take for example the following argument that shows there are languages which are not recognizable by any Turing Machine: The set of Turing Machines is countable while the set of all languages is uncountable, and thus there must be a language that is not recognizable by any Turing Machine. This argument relativizes to any oracle B. Indeed for any language B, the set of oracle Turing Machines MB is countable, and thus for every B, there are languages that are not recognizable by any oracle Turing Machine MB.

The diagonalization argument that we used to prove the time and space hierarchy theorems merely uses the fact that our computational model allows us to construct an efficient universal machine capable of simulating the computation of any machine M on an input w without much overlay. This obviously is true for oracle Turing Machines MB as well. Hence the same diagonalization argument implies similar hierarchy theorems: e.g. P̸=EXP and PSPACE̸=EXPSPACE for every language B.

To celebrate or to grieve? The proofs that relativize imply impressively broad statements. For example, as we discussed above, for every language B, we have P̸=EXP. While we can celebrate the broadness of such results, we should suspect that unfortunately this robustness may be a sign of limitations of these proof techniques. Such robust techniques cannot be employed to prove the statements that do not relativize, as any statement established by them must suspect that unfortunately this robustness may be a sign of limitations of these proof techniques. Such robust techniques cannot be employed to prove the statements that do not relativize, as any statement established by them must be true in any oracle model. With this realization, one immediately wonders about the techniques cannot be employed to prove the statements that do not relativize, as any statement established by them must be true in any oracle model. With this realization, one immediately wonders about the

Theorem 9.9 (Baker, Gill, and Solovay [BGS75]). There exist languages A and B such that PA = NP̸A, and P̸=NPB.

Proof. Note that for every L, we have PL ⊆ NP L, thus the statement is equivalent to ensuring that NP A ⊆ PA and NP B ⊆ PB.

The first part is easy. We will show that for any PSPACE-complete language A (such as TQBF), we have PA = NP A ⊆ PSPACE. It suffices to show NP A ⊆ PSPACE, as the inclusions PSPACE ⊆ PA is obvious from the completeness of A. Let L ∈ NP A. Then there exists a polynomial time non-deterministic oracle Turing Machine MA that decides L. Note that with an additional polynomial amount of space, we can go over all the computational paths of MA, and for each occurring query, run the PSPACE algorithm to decide its result. The total space is still going to be polynomially bounded.

Next, we turn to the less obvious part of the theorem. Our goal is to construct a language B such that NP B ⊆ P B. In order to achieve this, we need to guarantee the existence of a language LB ∈ NP B such that LB /∈ PB. A good candidate is

LB = {1n | ∃x ∈ B, |x| = n},

which is the set of the lengths of the words in B, represented in unary. Note that independent of the choice of B, the language LB is always in NP B, as to accept 1n, we simply need to non-deterministically guess an x of length n, and use the oracle to verify that x ∈ B.

The set LB is a good candidate for a problem in NP B \ PB because the membership of 1n depends on the membership situation of 2n elements in B, and it seems unlikely that for a generic B we can deduce the required information from only polynomially many queries to the B-oracle. In particular, we will choose B to be very sparse. For every value of n, B has at most one element in the set {0,1}n, and thus any polynomial time deterministic oracle Turing Machine will be incapable of finding whether such an element exists.

We will prove a very strong separation, showing that not only no polynomial time MB can decide LB, but any oracle Turing machine deciding LB must essentially query all the 2n strings of size n.

Claim 9.10. There exists a language B such that every deterministic oracle Turing Machine MB that decides LB must have running time at least 2n on infinitely many inputs w = 1n.

Proof. We will construct B using an enumeration M1, M2, . . . of the descriptions of all oracle Turing Machines (recall Remark 9.3). We will assume that every description is repeated infinitely many times in this enumeration. We will construct B and a sequence of integers 0 < n1 < n2 < . . . such that MB cannot correctly decide 1n ∈ LB in time less that 2nk. Since every Turing Machine appears infinitely many times in the sequence, we conclude that any MB that does not have running time ≥ 2n for infinitely many n, will eventually fail on some nk.

1Using say 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, . . .
The set \( B \) is going to be constructed in such a way that \( M_k \) will fail to correctly decide whether \( 1^{n_k} \in L_B \) unless it queries all the \( 2^{n_k} \) strings \( w \in \{0,1\}^{n_k} \).

In the process we will keep track of a set \( S \) of all the strings \( w \) whose memberships/non-memberships to \( B \) are already decided.

Initialize \( S = \emptyset \). For \( k = 1, 2, \ldots \), let \( n_k = 1 + \max_{w \in S} |w| \), so that no element in \( \{0,1\}^{n_k} \) belongs to \( S \). Run \( M_k \) on \( 1^{n_k} \) for at most \( 2^{n_k} - 1 \) steps, and each time it makes a query “\( w \in B ? \)”, if \( w \in S \), then reply according to what that has been decided about \( w \), and otherwise reply “\( w \notin B \)”, set \( w \notin B \), and add \( w \) to \( S \). When our simulation terminates, \( M_k \) has made at most \( 2^{n_k} - 1 \) queries to the oracle, and thus there is at least one \( y_0 \in \{0,1\}^{n_k} \) that has not been queried by \( M_k \). We will use this \( y_0 \) to make the outcome of \( M_k \) incorrect, but first for every \( y \notin S \cup \{y_0\} \) with \( |y| \leq \max_{w \in S} |w| \) we set \( y \notin B \), and update \( S \). It remains to decide the fate of \( y_0 \). We can have three different cases:

- \( M_k \) rejected \( 1^{n_k} \). In this case, set \( y_0 \in B \), and update \( S \). Since \( M_k \) has never queried \( y_0 \), this does not influence its outcome. Now \( M_k \) is rejecting \( 1^{n_k} \) but \( 1^{n_k} \in L_B \) because \( y_0 \in B \).
- \( M_k \) accepted \( 1^{n_k} \). In this case, set \( y_0 \notin B \), and update \( S \). Now \( M_k \) is accepting \( 1^{n_k} \) but \( 1^{n_k} \notin L_B \) because \( \{0,1\}^{n_k} \cap B = \emptyset \).
- \( M = M_k \) has not terminated yet. In this case, the running time of \( M_k \) is at least \( 2^{n_k} \) on \( 1^{n_k} \), so it is fine if it correctly decides \( 1^{n_k} \). We can do either \( y_0 \in B \) or \( y_0 \notin B \).

Note that with the construction of \( B \) we achieved our goal that \( M_k \) fails to correctly decide whether \( 1^{n_k} \in L_B \) unless it queries all the \( 2^{n_k} \) string \( w \in \{0,1\}^{n_k} \).

What does this the non-relativization of \( P \) vs \( NP \) mean? Theorem 9.9 means that any proof of \( NP \neq P \) must contain elements that are non-relativizing. For example, we cannot just rely on the fact that it is possible to use a universal Turing Machine to simulate a polynomial time \( M \) on an input \( w \) in polynomial time. Also note that the proof of Claim 9.10 provides some intuition why one would expect to have \( P \neq NP \). For problems such as SAT, a non-deterministic Turing Machine can simply guess a satisfying truth assignment, while if the set of satisfying truth assignments is very sparse, a deterministic Turing Machine will have to somehow find the needle in a haystack of unsatisfying truth assignments.

An interesting related story is the \( IP \) versus \( PSPACE \) problem, where \( IP \) stands for Interactive Polynomial time, a class that we will not discuss in this course. It was first shown by Fortnow and Sipser that there exists a language such that \( IP^B \neq PSPACE^B \). This to some extend was interpreted as evidence that \( IP \neq PSPACE \). However a few years later in a breakthrough paper Shamir [Sha92] proved that in fact \( IP = PSPACE \).

Below is a quotation from a paper of Lance Fortnow [For94] entitled “The Role of Relativization in Complexity Theory”.

The recent result \( IP = PSPACE \) surprised the theoretical computer science community in more ways than one. A few years earlier, Fortnow and Sipser created an oracle relative to which \( coNP \) did not have interactive proofs [Thus \( IP^B \neq PSPACE^B \) for that oracle \( B \)]. The \( IP = PSPACE \) result was honestly a non-relativizing theorem.

Several questions immediately popped up: Why didn’t this result relativize? What specific techniques were used that avoided relativization? How can we use these techniques to prove other non-relativizing facts?

Also much older questions resurfaced: What exactly do oracle results mean? What should we infer, if anything, from a relativization?

Exercises

Exercise 9.1. Prove that if \( A \leq_m B \), then \( A \leq_T B \), and if \( A \leq_p B \), then \( A \leq_T^p B \).

Exercise 9.2. Give an example of languages \( A \) and \( B \) such that \( A \leq_T B \) but \( A \nless_m B \). Similarly show that if \( NP \neq coNP \), then there are languages \( A \) and \( B \) such that \( A \leq_T^p B \) but \( A \nless_p B \).
Exercise 9.3. Prove that there are languages $A, B \in \text{EXP}$ such that $A \leq_T^P B$ but $A \not\leq_P B$. (Hint: Construct $A$ and $B$ such that $n \in A$ if and only if either $2n \in B$ or $2n + 1 \in B$.)

Exercise 9.4. Prove that $\text{NP} \cup \text{coNP} \subseteq \Delta_2$, and that under the assumption $\text{NP} \neq \text{coNP}$, this inclusion is proper.

Exercise 9.5. Prove that $\text{NP} \subseteq \Sigma_2$ by showing that if $L \in \text{NP}^B$ and $B \in \text{NP}$, then $L \in \Sigma_2$.

Exercise 9.6. Prove that Definition 9.8 is consistent with the usual definition of the polynomial hierarchy.

Exercise 9.7. Does the proof of $\text{NL} = \text{coNL}$ relativize? In other words, is it true that for every $B$ we have $\text{NL}^B = \text{coNL}^B$?

Exercise 9.8. Let $A$ be an oracle such that $P^A = \text{NP}^A$. Prove that there is an oracle such that $P^{A,B} \neq \text{NP}^{A,B}$. 
Chapter 10

Randomized Complexity Classes

A randomized algorithm is one that receives, in addition to its input data, a stream of random bits that it can use for the purpose of making random choices. As a result, even for a fixed input, different executions of the same randomized algorithm may end in different results; thus it is inevitable that a description of the properties of a randomized algorithm will involve probabilistic statements, and allow for some probability of error. For example, even when the input is fixed, the execution time and space complexity of a randomized algorithm are random variables.

It is important to understand that the area of randomized algorithms is not about analyzing the algorithms on randomly chosen inputs. The latter is called the average case analysis of algorithms, in which one analyzes the performance of an algorithm when the inputs are randomly drawn from a distribution. In contrast, when analyzing randomized algorithms, similar to the deterministic algorithms, we are interested in the worst-case analysis of these algorithms, meaning that we analyze the performance and accuracy of the algorithm on its worst possible input.

There are two principal reasons that randomized algorithms are often superior to the best known deterministic algorithms. First, to this day, for many problems the best known randomized algorithms are faster and more space-efficient than their deterministic counterparts for the same problem. Second, if we survey the various randomized algorithms that have been discovered, we notice that invariably they are extremely simpler to understand and to implement; often, the introduction of randomization suffices to convert a simple and naive deterministic algorithm with bad worst-case behaviour into a randomized algorithm that performs well with high probability on every possible input.

The success of randomized algorithms raises important and fundamental theoretical questions:

Are randomized algorithms inherently stronger than deterministic algorithms, or to the contrary, every problem that can be solved by a randomized algorithm can also be solved by a deterministic algorithm using a comparable amount of resources?

To formalize this question, we will introduce the randomized complexity classes, and we shall try to compare them to their deterministic complexity counterparts. Most of these classes were introduced by Gil [Gil77] in 1977.

Probabilistic Turing machines

There are two equivalent ways to define a probabilistic Turing machine. The first definition of a probabilistic Turing machine is a Turing machine with two transition functions, where in each nondeterministic step a fair “coin-flip” is used to decide which transition function to use. Note the similarity with the definition of nondeterministic Turing Machines. The difference is the way computation is carried over; a nondeterministic Turing Machine accepts an input if there is a sequence of choices that leads to halting at an accept state, however, a probabilistic Turing Machine’s output is a random variable, depending on the random choices it makes.
We prefer the following equivalent definition as it has the advantage that one can keep track of the number of used random bits, and interpret it as another complexity measure. That is time, space, and randomness are each considered a valuable resource.

**Definition 10.1.** A **probabilistic Turing machine** is a type of a Turing Machine that has an additional infinite read-once randomness tape. Initially this tape is filled with an infinite sequence of independent unbiased random bits. In other words, for every cell, a fair coin is flipped independently to decide the value of the cell.

Read-once access to a tape means that at each computation step, the tape-head can either stay at its current position, or move to the right. The equivalence of the two definitions is immediate from corresponding the random bits on the tape to the random choices made in the former definition.

Note that the content of the randomness tape is a random variable $R$ which is uniformly distributed over $\{0,1\}^N$, and any fixation of $R$ to a string $r \in \{0,1\}^N$ results in a deterministic Turing Machine $M_r$. It is convenient to denote such a probabilistic Turing Machine as $M_R$ to emphasize that the randomness is in the random variable $R$. We denote the outcome of $M_R$ on an input $x$ by $M_R(x)$, which is a random variable that could take three different possible values ACCEPT, REJECT, ∅, the latter corresponding to an infinite loop. Let us consider an example to demonstrate the power of randomized algorithms.

**Example 10.2** (Matrix Multiplication). Let $L$ be the set of all triples $(A, B, C)$, where $A$, $B$, $C$ are $n \times n$ matrices with entries in $\{0,1\}$ satisfying $AB = C$. To solve this problem deterministically, one can compute the matrix product $AB$ and then compare it to $C$ entry-wise. Computing this product in the most straightforward way results in a running time of $\Omega(n^3)$ addition and multiplication operations, but this can be improved by using Strassen’s matrix multiplication algorithm to $O(n^{2.81})$. Matrix multiplication algorithms have seen many improvements over the decades, and currently the fastest known [LG14] algorithm for computing the product of two $n \times n$ matrices involves $\Omega(n^{2.372863})$ operations. Now, let us consider the following very simple randomized algorithm.

**Algorithm 5:** A randomized matrix product verification $AB = C$

<table>
<thead>
<tr>
<th>Data:</th>
<th>Three $n \times n$ matrices $A, B, C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>$u \in {0,1}^n$ uniformly at random;</td>
</tr>
<tr>
<td>If</td>
<td>$ABu \neq Cu$ then</td>
</tr>
<tr>
<td></td>
<td>REJECT</td>
</tr>
<tr>
<td>else</td>
<td>ACCEPT</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

Note that the vector $v = Bu$ can be computed using $O(n^2)$ operations, and then $ABu = Av$ can be computed in an additional $O(n^2)$ steps. Hence verifying $ABu \neq Cu$ can be done in $O(n^2)$ operations. Hence the above algorithm takes only $O(n^2)$ addition and multiplication operations to perform. It remains to analyze the probability of correctness. It is not difficult to see that if $D$ is a nonzero $n \times n$ matrix, then for at least half of the vectors $u \in \{0,1\}^n$, we have $Du \neq 0$. It follows that if $AB \neq C$, then for at least half of the vectors $u \in \{0,1\}^n$, we have $ABu \neq Cu$. Hence

- If $AB = C$, the algorithm is correct with probability 1.
- If $AB \neq C$, the algorithm is correct with probability at least 1/2.

This is an algorithm with one-sided error—it does not err if $AB = C$. The bound 1/2 is not very satisfactory, but fortunately for algorithms with one-sided error we can easily reduce the error probability. For example, we can simply run the above algorithm 1000 times with fresh randomness, and if at any of these runs we conclude that $AB \neq C$, then we know confidently that $AB \neq C$. If all the runs return $AB = C$, then we accept the input. The probability that we are wrong now is bounded by $(1/2)^{1000} \leq 10^{-300}$ which is extremely small. The number of operations of the new algorithm is still $O(n^2 \log(n))$, which is faster than any known deterministic algorithm for this problem.

**Example 10.3** (Polynomial Identity Testing). The polynomial identity testing (PIT) problem involves deciding whether two given multivariate polynomials $f(x_1, ..., x_n)$ and $g(x_1, ..., x_n)$ are equal, that is, we want to efficiently decide whether $f = g$.

In order to make the PIT problem statement explicit, we need to specify the space of the coefficients of the polynomials, what we mean by $f = g$, and in what format are the polynomials $f$ and $g$ given to the algorithm as inputs.

---

1If $D_{ij} \neq 0$, then no matter what the other coordinates of $u$ are, always at least one of the two choices $u_j \in \{0,1\}$ leads to $Du \neq 0$
Coefficient space: Recall that $n$-variate polynomial $f(x_1, \ldots, x_n)$ over $F$, where $F$ is a field (such as $\mathbb{R}, \mathbb{Q}, \mathbb{C}, \mathbb{F}_p$) or an integral domain (such as $\mathbb{Z}$) is a linear combination of monomials of the form $x_1^{e_1}x_2^{e_2}\cdots x_n^{e_n}$, where the coefficients belong to $F$. The total degree of a monomial $x_1^{e_1}x_2^{e_2}\cdots x_n^{e_n}$ is $e_1 + \cdots + e_n$, and the total degree of a polynomial is equal to the largest total degree of a monomial with a nonzero coefficient. To specify the PIT problem, we need to fix $F$. In the discussion of algorithms, we will fix $F = \mathbb{Z}$, and give an efficient randomized algorithm for PIT over the integers.

Identity: There are two ways that polynomials can be considered equal:

1. $f = g$ syntactically, meaning that each monomial has the exact same coefficient in $f$ and $g$.

2. $f = g$ as functions. Meaning that for every $x \in F^n$, $f(x) = g(x)$.

The two above definitions of polynomial identity are not the same over all domains. For example over any finite field $\mathbb{F}_p$, the univariate polynomial $f(x) = \prod_{i=0}^{p-1}(x - i)$ is equal to $g(x) = 0$ as a function, but syntactically $f \neq 0$. When $F$ is either of $\mathbb{Z}, \mathbb{R}, \mathbb{Q}$, or $\mathbb{C}$, this is not a concern and both the definitions coincide. In the case of finite fields, the two definitions still coincide as long as the size of the field is larger than the degree of $f$ and $g$.

The inputs: In order to study efficient algorithms for PIT, we need to specify how the algorithm is given access to the input polynomials $f$ and $g$. The first choice is when the algorithm is given $f$ and $g$, via their list of coefficients. In this case, the problem becomes trivial, as the algorithm can in linear time (in input length) compare the two polynomials syntactically. There are two nontrivial settings that are both interesting:

1. We are given $f,g$ as Arithmetic circuits: An arithmetic circuit is a directed acyclic graph with a single sink node labeled as the output node, the nodes of indegree 0 are labeled by input variables $x_1,\ldots,x_n$ or domain elements, and all other nodes (called “gates”) are labelled by $+$ or $\times$ operations. An arithmetic circuit naturally corresponds to a polynomial over the input variables on its source nodes. The reason this form of input makes the problem more interesting is that there are polynomials with exponentially many monomials, but polynomial-sized circuits. For example, the polynomial $\prod_{i=1}^{n}(1 + x_i)$ has $2^n$ monomials, however, there is a simple linear-sized circuit computing it.

2. We are given black-box access to $f,g$: In this case, the algorithm is not given any representation of the polynomials $f$ and $g$; Instead, at any point of its computation, the algorithm can ask for the value of $f(x)$ or $g(x)$ for a particular input $x \in F^n$. In this case, we have to specify the parameters based on which we judge the efficiency of the algorithm. When $F$ is finite, it is reasonable to demand for polynomial-time algorithms in $n \log |F|$. Otherwise, we can ask for polynomial-time algorithm in $n \log d$ where $d$ is the maximum degree of $f$ and $g$.

Randomized PIT Algorithm. No efficient deterministic algorithm is known for the PIT problem in any of the nontrivial settings above. Here we present a simple randomized algorithm when the domain is $\mathbb{Z}$ (in both black-box and arithmetic circuit settings). The key tool is the Schwarz-Zippel Lemma, which gives useful information about the roots of a degree $d$ multivariate polynomial.

Lemma 10.4 (Schwarz-Zippel). Let $f(x_1,\ldots,x_n)$ be a nonzero polynomial of total degree at most $d$ over $\mathbb{Z}$. For every finite set $S \subseteq \mathbb{Z}$, choosing $(x_1,\ldots,x_n) \in S^n$ uniformly at random, we have

$$\Pr[f(x_1,\ldots,x_n) = 0] \leq \frac{d}{|S|}.$$

This lemma suggests a simple black-box algorithm for PIT.

**Algorithm 6:** Black-box randomized PIT algorithm:

**Data:** Query access to two degree $d$ polynomials $f$ and $g$

Pick $S \subseteq \mathbb{Z}$ of cardinality $|S| = 3d$

Pick $x_1,\ldots,x_n$ independently and uniformly from $S$

if $f(x_1,\ldots,x_n) \neq g(x_1,\ldots,x_n)$ then

Reject

else

Accept

end
When $f = g$, the algorithm outputs the correct answer with probability 1. However, when $f \neq g$, since $f - g \neq 0$, by Lemma 10.4, the above algorithm outputs the correct answer with probability at least $2/3$. Note that the running time of the algorithm is $O(n \log d)$, and moreover, the constant $2/3$ can be improved to any $k/(k + 1)$ by picking $|S| = (k + 1)d$.

The algorithm for the arithmetic setting is a slight modification of the above algorithm. In this case we are not given query access to $f$ and $g$, and are also not given a bound on their degree. The degree, however, is not a problem, since a size $m$ arithmetic circuit corresponds to a polynomial of degree at most $2^m$. Thus the above algorithm can be modified to pick $|S| = 3 \cdot 2^m$. The next issue, is that we are no longer given query access to $f$ and $g$, and thus instead we can try and verify whether $f(x_1, \ldots, x_n) \neq g(x_1, \ldots, x_n)$ by evaluating the input circuits. In the case when the field is small, this can obviously be done efficiently. However, over $\mathbb{Z}$, $f(x_1, \ldots, x_n)$ can be as large as $(3 \cdot 2^m)^2$, which takes exponential time and space to even write down. The way around this issue is to pick a random prime $p \leq 2^{2m}$, and do all the arithmetic module that prime. That is, we compute $f(x_1, \ldots, x_n) \mod p$ and compare it to $g(x_1, \ldots, x_n) \mod p$, and we reject if they are different. Note that this can be done in $\text{poly}(m)$ time, and the algorithm still makes no error when $f = g$. Thus all that is left to show is that when $f(x_1, \ldots, x_n) \neq g(x_1, \ldots, x_n)$, then with high probability over the randomness of $p$ we have $f(x_1, \ldots, x_n) \neq_p g(x_1, \ldots, x_n)$.

The key here is to use the Prime Number Theorem, which guarantees that there are at least $2^{2m}/4m$ prime numbers $p \leq 2^{2m}$. Moreover, since $\log|f(x_1, \ldots, x_n) - g(x_1, \ldots, x_n)| \leq \log(3 \cdot 2^m)^2 = O(m \cdot 2^m) = o(2^m/4m)$, then $f(x_1, \ldots, x_n) - g(x_1, \ldots, x_n)$ has at most $3 \cdot 2^{2m}$ prime factors, and thus with probability $1 - o(1)$, $f(x_1, \ldots, x_n) \neq_p g(x_1, \ldots, x_n)$.

### Polynomial time randomized complexity classes

We introduce four complexity classes $\mathsf{ZPP}$, $\mathsf{RP}$, $\mathsf{BPP}$, $\mathsf{PP}$, all concerning polynomial time randomized algorithms, but allowing different types and probabilities of error. In the following, error refers to accepting an $x \notin L$, or rejecting an $x \in L$.

#### The complexity class $\mathsf{RP}$

The complexity class $\mathsf{RP}$ (randomized polynomial time) consists of languages $L$ for which a probabilistic Turing machine exists with the following properties:

- It always runs in polynomial time in the input size.
- If $x \notin L$, then the probability of REJECT is 1.
- If $x \in L$, then the probability of ACCEPT is $\geq 1/2$.

In particular, the only type of errors allowed are false-negatives.

In other words, this is the class of languages $L$ for which $x \in L$ can be decided probabilistically with one-sided error. Due to asymmetry in the roles of accept and rejects, we also consider the class $\text{coRP}$, which only allows error when $x \notin L$. Note that the algorithm presented for PIT and Matrix Product Verification both place the corresponding languages in $\text{coRP}$ as both the algorithms make no error when $x \in L$, and make error with probability at most $1/2$ when $x \notin L$.

**Error-reduction in RP:** The constant $1/2$ in the definition of $\mathsf{RP}$ is quite arbitrary as one can repeat the same algorithm several times to decrease the constant. We can run the algorithm $k = n^{O(1)}$ times and if the algorithm outputs accept at any of these trials, we can accept $x$ confidently, as we know the original algorithm never outputs accept for $x \notin L$. We will reject the input if all the $k$ trials output REJECT. Note that if $x \in L$, then the probability of an erroneous reject is bounded by

$$\left(\frac{1}{2}\right)^k = 2^{-n^{O(1)}}.$$ 

By a similar argument, observing that $(1 - n^{-c})^{n^c} < 1/2$, we could have equivalently defined $\mathsf{RP}$ with a milder requirement of

- If $x \in L$, then the probability of accept is $\geq n^{-c}$, for some fixed $c > 0$.  

48
How large is \( \text{RP} \)? It is often helpful to interpret the randomness of a probabilistic algorithm as some sort of advice/certificate. In the context of non-deterministic classes such as \( \text{NP} \), we are interested in existence of an advice that leads to accept, whereas in the probabilistic classes such as \( \text{RP} \), we demand certain bounds on the probability that a random advice leads to a correct answer. In other words, \( \text{NP} \) can be thought of a probabilistic class: \( L \in \text{NP} \) if there exists a polynomial time probabilistic Turing Machine \( M_R \) such that

- If \( x \notin L \), then the probability of REJECT is 1.
- If \( x \in L \), then the probability of ACCEPT is \( > 0 \).

Using this view, it is easy to observe that \( \text{RP} \) is contained in \( \text{NP} \), that is nondeterminism is at least as powerful as one-sided error randomness.

**Theorem 10.5.** We have \( \text{RP} \subseteq \text{NP} \), and \( \text{coRP} \subseteq \text{coNP} \).

**Proof.** It suffices to prove \( \text{RP} \subseteq \text{NP} \). Consider \( L \in \text{RP} \). There exists a polynomial time probabilistic Turing Machine \( M_R \) such that

- If \( x \notin L \), then the probability of REJECT is 1.
- If \( x \in L \), then the probability of ACCEPT is \( \geq 1/2 \).

Then obviously

\[
x \in L \iff \exists r \ M_r(x) = \text{ACCEPT},
\]

which establishes \( L \in \text{NP} \). \( \square \)

**The complexity class ZPP**

The complexity class \( \text{ZPP} \) (Zero-error probabilistic polynomial time) consists of problems that can be solved by a probabilistic Turing machine \( M_R \) that returns ACCEPT, REJECT, or DO NOT KNOW such that

- \( M_R \) runs in polynomial time.
- For every \( x \),
  \[
  \Pr_R[M_R(x) = \text{Do Not Know}] \leq \frac{1}{2}.
  \]
- For every \( x \), the output of \( M_R(x) \) is always either DO NOT KNOW, or the correct answer.

This class is very restricted as despite the randomness, the algorithm is not allowed to make any errors.

**Amplifying correctness in ZPP:** One can easily decrease the probability of DO NOT KNOW by repeating the algorithm. We can simply run the algorithm \( k = n^{O(1)} \) times, and if any of these \( k \) trials lead to accept or reject, then we will know whether \( x \in L \) or not. The probability of DO NOT KNOW will be bounded by

\[
\Pr[R][\text{Do Not Know}] \leq \left( \frac{1}{2} \right)^k = 2^{-n^{O(1)}}.
\]

Note also that in the definition of the class \( \text{ZPP} \), we could have asked for a milder condition of

\[
\Pr[R][M_R(x) = \text{Do Not Know}] \leq 1 - n^{-c},
\]

for any fixed constant \( c > 0 \). Repeating this \( k = n^c \) would have resulted\(^2\) to

\[
\Pr[\text{Do Not Know}] \leq (1 - n^{-c})^n \leq e^{-1} \leq \frac{1}{2}.
\]

The next theorem shows that the class \( \text{ZPP} \) can alternatively be defined as the intersection of \( \text{RP} \) and \( \text{coRP} \).

**Theorem 10.6.** We have \( \text{ZPP} = \text{RP} \cap \text{coRP} \).

\(^2\)Using the inequality \( 1 + x \leq e^x \) that holds for all \( x \).
Proof. First consider \( L \in \text{ZPP} \). Then there is a polynomial time probabilistic Turing Machine \( M_R \) such that it always outputs \text{Accept}, \text{Reject}, or \text{Do Not Know}, such that for every input

\[
\Pr[R[M_R = \text{Do Not Know}] \leq 1/2.
\]

To convert this to an RP algorithm, we can simply output \text{Reject} if \( M_R(x) \in \{ \text{Reject}, \text{Do Not Know} \} \), and otherwise output \text{Accept}. Note that if \( x \notin L \), we will always reject, and if \( x \in L \), then we will accept with probability at least 1/2. Changing the roles of \text{accept} and \text{reject} gives similarly \( \text{ZPP} \subseteq \text{coRP} \).

Next suppose that \( L \in \text{RP} \cap \text{coRP} \), and thus there are polynomial time probabilistic algorithms \( M_R, N_R \) such that

- If \( x \notin L \), then \( \Pr[M_R(x) = \text{REJECT}] = 1 \), and \( \Pr[N_R(x) = \text{REJECT}] \geq 1/2 \).
- If \( x \in L \), then \( \Pr[M_R(x) = \text{ACCEPT}] \geq 1/2 \), and \( \Pr[N_R(x) = \text{ACCEPT}] = 1 \).

Now we can simply run \( M_R(x) \) and \( N_R(x) \), and

- Accept if \( M_R(x) = \text{ACCEPT} \);
- Reject if \( N_R(x) = \text{REJECT} \);
- Otherwise output \text{Do Not Know}.

It is straightforward to see that this is a \( \text{ZPP} \) algorithm. \( \square \)

The complexity class \text{BPP}

The complexity class \( \text{BPP} \) (bounded-error probabilistic polynomial time) consists of languages \( L \) that can be decided by a probabilistic Turing machine such that the following conditions hold.

- It always runs in polynomial time in the input size.
- For every \( x \), the probability of error is at most \( \leq 1/3 \).

In other words the (two-sided) error probability is bounded by 1/3 for all instances. Note that achieving an error probability of \( \leq 1/4 \) is always obvious as a randomized algorithm can simply output a random \text{ACCEPT} or \text{REJECT}, which will be correct with probability 1/2. The class \( \text{BPP} \) is the set of problems for which we can improve this obvious bound by a significant margin.

Example 10.7. Define \( L \) to be the language, consisting of triples of polynomials \((p_1,p_2,p_3)\) such that exactly two of the polynomials are the same. We leave it as an exercise to show that \( L \in \text{BPP} \). It is not known whether \( L \) is in \( \text{RP} \) or \( \text{coRP} \).

In the above definition, we used the constant 1/3, but in fact one could use any \( \frac{1}{2} - \varepsilon \) as long as \( \varepsilon \) is not too small. More precisely \( \varepsilon > n^{-c} \) for some constant \( c \). This follows from the following error-reduction argument.

Error-reduction in \text{BPP}: We wish to show that similar to RP algorithms, one can decrease the probability of error of a BPP algorithm by repeating it. Suppose that there is a probabilistic Turing Machine \( M_R \) that decides \( L \) in polynomial time such that for every \( x \), the probability that \( M_R(x) \) is incorrect is at most \( \frac{1}{2} - \varepsilon \), for some \( \varepsilon > 0 \). Consider the following randomized algorithm. Run \( k \) independent simulations of \( M_R(x) \) . Let \( X \) be the number of times that \( M_R(x) \) outputs accept. If \( X \geq k/2 \) then accept, and otherwise reject.

Every single run of the algorithm is biased towards the correct answer, and hence one expects (by law of large numbers) that with a large probability the majority of outputs to be correct. To be more rigorous, we will apply the Chernoff bound. Without loss of generality consider \( x \in L \). Let \( X_i \) be the random variable such that \( X_i = 1 \) if the outcome of the \( i \)-th trial is correct, and \( X_i = 0 \) otherwise. By our assumption \( \mathbb{E}[X_i] \geq \frac{1}{2} + \varepsilon \), and hence \( \mu := \mathbb{E} \sum X_i \geq \frac{k}{2} + \varepsilon k \). Note that our majority count algorithm makes an error if

\[
\sum X_i < \frac{k}{2} \leq \mu(1 - \varepsilon).
\]
Applying Chernoff bound\footnote{Chernoff bound says that if $X$ is the sum of $k$ i.i.d. Bernoulli random variables, then denoting $\mu = E[X]$, for every $\delta > 0$, we have  

$$\Pr[X < \mu(1 - \delta)] \leq e^{-\frac{\delta^2}{2}}$$

and

$$\Pr[X > \mu(1 + \delta)] \leq e^{-\frac{\delta^2}{2\mu}}.$$} we obtain

$$\Pr[\text{Error}] \leq \Pr \left[ \sum X_i \leq (1 - \varepsilon)\mu \right] \leq e^{-\frac{\varepsilon^2}{2}} \leq e^{-\frac{\delta^2}{2}}.$$ 

Note that this is a very strong bound, and is comparable to the error-reduction of RP. In particular by taking $k = n^{O(1)}$, we can guarantee that

$$\Pr[\text{Error}] \leq 2^{-n^c},$$

for any constant $c = O(1)$ that we wish, even if the starting success probability was guaranteed to be at least $\frac{1}{2} + n^{-O(1)}$.

**How large is BPP?** It is believed by many that $\mathsf{BPP} = \mathsf{P}$. It is easy to show that $\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{PSpace}$. The one-sided version of $\mathsf{BPP}$, the class $\mathsf{RP}$, is contained in $\mathsf{NP}$, hence naturally one wonders if $\mathsf{BPP}$ is contained in a smaller class than $\mathsf{PSpace}$. We will show that $\mathsf{BPP}$ is contained in the second level of the polynomial hierarchy.

**Theorem 10.8.** We have $\mathsf{BPP} \subseteq \Sigma_2 \cap \Pi_2$.

**Proof.** Since $\mathsf{BPP}$ is closed under taking complements, it suffices to show that $\mathsf{BPP} \subseteq \Sigma_2$. Let $L \in \mathsf{BPP}$. By definition there is a polynomial time probabilistic algorithm that correctly decides every membership query with probability at least $2/3$. This constant is not large enough for the argument to go through, so we start by applying a strong error reduction argument to obtain a randomized algorithm with polynomial running time $p(n)$ such that $M_R(x)$ is incorrect with probability at most $2^{-n}$. Fix the input length $n$, and let $\ell = p(n)$. For every $x$ of length $n$, let

$$A_x = \{ r \in \{0,1\}^\ell : M_r(x) = \text{Accept} \}.$$ 

From the correctness probabilities of $M_R$ we know that

- If $x \in L$, then $|A_x| \geq 2^\ell \left(1 - \frac{1}{2^\ell}\right)$;
- If $x \notin L$, then $|A_x| < 2^\ell \left(\frac{1}{2^\ell}\right)$;

In other words, if $x \in L$, then $A_x$ is almost all the points in $2^\ell$, and if $x \notin L$, then $A_x$ is very sparse. In fact we will only need such strong bounds for the case $x \notin L$. These bounds will allow us to extract some structure about $A_x$ so that we can describe $L$ as a $\Sigma_2$ language.

For any subset $A \subseteq \{0,1\}^\ell$, and any $t \in \{0,1\}^\ell$, define the corresponding shift of $A$ as

$$A \oplus t := \{ s \oplus t : s \in A \}.$$ 

How many shifts of a set $A$ does one need to cover all of $\{0,1\}^\ell$?

Since $|A \oplus t| = |A|$, if $|A| \leq 2^{-n}2^\ell$, then one needs at least $2^n$ shifts of $A$ to cover all of $\{0,1\}^\ell$. We will use this for the case $x \notin L$, where $|A_x|$ is very small. Now how well can we do in the other case where $A_x$ is almost everything?

**Claim 10.9.** If $A$ is a subset of $\{0,1\}^\ell$ with $|A| \geq \left(\frac{2}{3}\right)2^\ell$, then there are $t_1, \ldots, t_\ell \in \{0,1\}^\ell$ such that $\{0,1\}^\ell = \bigcup_{i=1}^\ell A \oplus t_i$.

**Proof.** Pick $t_1, \ldots, t_\ell$ independently and uniformly at random from $\{0,1\}^\ell$. Consider a fixed $y_0 \in \{0,1\}^\ell$. Note that $y_0 \in A \oplus t$ if and only if $y_0 \oplus t \in A$, and moreover for a uniformly random $t$, $y_0 \oplus t$ also has uniform distribution, and thus $\Pr[y_0 \in A \oplus t] = |A|/2^\ell$. Hence the probability that $y_0$ is not covered by any of our random shifts is bounded by

$$\Pr \left[ \forall_{i = 1}^\ell A \oplus t_i \right] = \left(1 - \frac{|A|}{2^\ell}\right)^\ell \leq (1/3)^\ell.$$ 

Applying the union bound over all $2^\ell$ choices of $y \in \{0,1\}^\ell$, we have

$$\Pr \left[ \exists_{i = 1}^\ell \bigcup_{i=1}^\ell A \oplus t_i \right] \leq 2^\ell (1/3)^\ell < 1.$$ 

Since this probability is less than 1, there must be at least one choice of shifts such that $y \notin \bigcup_{i=1}^\ell A \oplus t_i$ does not exist. \qed
Since \( \ell = p(n) \) is a polynomial, there exists a constant \( n_0 \) such that for every \( n \geq n_0, 2^n > \ell \). From the above discussion we conclude that for every \( x \) with \( |x| \geq n_0 \), the set \( \{0, 1\}^\ell \) can be covered with \( \ell \) shifts of \( A_x \) if and only if \( x \) is accepted by \( M \). Hence

\[
x \in L \iff \exists t_1, \ldots, t_\ell \in \{0, 1\}^\ell \forall r \in \{0, 1\}^\ell, \bigvee_{i=1}^\ell M_{r \oplus t_i}(x) = \text{Accept}.
\]

Note that this is a \( \Sigma_2 \) formula with polynomially bounded number of variables, each in polynomial size. More precisely, we have a deterministic Turing Machine that given

\[
(x, [t_1, \ldots, t_\ell], r),
\]

if \( |x| \geq n_0 \), runs \( M_{r \oplus t_i} \) on \( x \) for \( i = 1, \ldots, \ell \), and accepts \( x \) if any of these runs outputs accept (if \( |x| < n_0 \), then it accepts if \( x \in L \). There are only finitely many such \( x \)). The running time of \( N \) is obviously polynomial in \( |x| \), and

\[
x \in L \iff \exists [t_1, \ldots, t_\ell] \forall r \text{ } N(x, [t_1, \ldots, t_\ell], r) = \text{Accept}.
\]

Theorem 10.8 shows that \( \text{BPP} \) is contained in the second level of the polynomial hierarchy. It is not known whether \( \text{BPP} \subseteq \text{NP} \), but it is conjectured that \( \text{BPP} = \text{P} \). One reason to believe this conjecture was given by the “hardness vs randomness” line of work that in essence shows that existence of explicit difficult function implies \( \text{P} = \text{BPP} \). To be more precise, as an example, Impagliazzo and Wigderson [IW97] proved that \( \text{P} = \text{BPP} \) if \( \text{E} \) requires exponential size circuits (see Chapter 11 and Conjecture 11.6), where \( \text{E} \) is the class of languages that can be solved in deterministic time \( 2^{O(n)} \). This assumption is very natural, and it is something that is widely believed.

For a long time, one of the most famous problems that was known to be in \( \text{BPP} \) but was not known to be in \( \text{P} \) was the problem of determining whether a given number is prime. However, in a 2002 paper Agrawal, Kayal, and Saxena [AKS04, AKS19] discovered a deterministic polynomial-time algorithm for this problem, thus showing that it is in \( \text{P} \).

It is also known that the problem of \( \text{P} \) versus \( \text{BPP} \) does not relativize [BG81]. In other words there are oracles \( A, B \) such that \( \text{BPP}^A = \text{P}^A \) and \( \text{BPP}^B \neq \text{P}^B \).

### The complexity class \( \text{PP} \)

This is the class of decision problems solvable by a probabilistic Turing machine in polynomial time, with an error probability of strictly less than 1/2 for all instances. The abbreviation \( \text{PP} \) refers to probabilistic polynomial time. As discussed above, an error probability of 1/2 can be obviously achieved by outputting \( \text{Accept}/\text{Reject} \) by tossing a fair coin. The condition in the definition of \( \text{PP} \) asks for beating this obvious bound.

Since every \( \text{PP} \) algorithm must have a polynomial \( n^{O(1)} \) running time, it can never read more than that many random bits from its randomness tape. This shows that if the probability of correctness is at least 1/2, then it must be at least \( \frac{1}{2} + 2^{-n} \) for some fixed \( c > 0 \). In other words, even though our definition does not require any explicit advantage over the obvious probability of correctness 1/2, such an explicit lower-bound is implied by the restriction on the running time. Note that this is different from the definition of \( \text{BPP} \), where the error reduction argument allowed us to assume probability of correctness of \( \frac{1}{2} + n^{-O(1)} \).

The following theorem shows that we can relax one of the strict inequalities.

**Theorem 10.10.** The class \( \text{PP} \) can be equivalently defined as the class of languages that can be decided by a polynomial time probabilistic Turing Machine such that

- If \( x \in L \), then \( \Pr[\text{Accept}] > \frac{1}{2} \).
- If \( x \notin L \), then \( \Pr[\text{Reject}] \geq \frac{1}{2} \).

**Proof.** See Exercise 10.3.

**Example 10.11.** Define the language Majority SAT to be the set of CNF’s \( \phi \) such that more than half of truth assignments to \( \phi \)'s satisfy it.

\[ \text{MAJSAT} = \{\phi : \text{more than half of the truth assignments satisfy } \phi \}. \]

Note that \( \text{MAJSAT} \in \text{PP} \). We can pick a truth assignment uniformly at random, accept if it satisfies \( \phi \) and reject otherwise. Note that if \( \phi \in \text{MAJSAT} \), then it will be accepted with probability \( > \frac{1}{2} \), and if \( \phi \notin \text{MAJSAT} \), then it will be rejected with probability \( \leq \frac{1}{2} \). Hence by Theorem 10.10 MAJSAT is in \( \text{PP} \). In Exercise 10.4 you are asked to prove that MAJSAT is in fact \( \text{PP} \)-complete under polynomial-time many-one reductions.
How large is \( \text{PP} \)? With Theorem 10.6, and a quick glance at the definitions, we see

\[
P \subseteq \text{ZPP} = \text{RP} \cap \text{coRP} \subseteq \text{RP} \cup \text{coRP} \subseteq \text{BPP} \subseteq \text{PP}.
\]

How large is the largest of these classes \( \text{PP} \)?

**Theorem 10.12.** We have \( \text{NP} \subseteq \text{PP} \subseteq \text{PSPACE} \).

**Proof.** First we show \( \text{PP} \subseteq \text{PSPACE} \). Consider a \( \text{PP} \) algorithm \( M_R \) with polynomial running time \( p(n) \), for a language \( L \).

Since the running time is bounded by \( p(n) \), the algorithm can never read beyond the first \( p(n) \) bits of its randomness, and hence it suffices to consider random bit sequences of length \( p(n) \). In other words, we are assuming that \( R \) is randomly and uniformly chosen from \( \{0, 1\}^{p(n)} \). To convert this to a \( \text{PSPACE} \) algorithm, we can simply go over all \( r \in \{0, 1\}^{p(n)} \), and for each one, run the deterministic algorithm \( M_r \) on \( x \).

As we do so, we reuse the space, and only keep a count of the number of the runs that have resulted in \( \text{Accept} \). This will require a space of \( O(p(n)) \). By the definition of \( \text{PP} \), if \( x \in L \), then this count must be strictly larger than \( |\{0, 1\}^{p(n)}| / 2 = 2^{p(n)-1} \), and if \( x \notin L \), then it must be strictly less than \( 2^{p(n)-1} \). Hence we can deduce whether \( x \in L \) from the count.

Next we show that \( \text{NP} \subseteq \text{PP} \). Consider \( L \in \text{NP} \). Then there is a deterministic algorithm \( V \) with polynomial running time \( p(n) \) where \( n = |x| \) such that

\[
x \in L \iff \exists y V(x, y) = \text{Accept}.
\]

Without loss of generality we can assume \( |y| = p(n) \). We construct a \( \text{PP} \) algorithm as follows:

- Pick \( Y \) uniformly at random from \( \{0, 1\}^{p(n)} \).
- Run \( V(x, y) \) and if it accepts, then accept.
- If it rejects then
  - With probability \( \frac{1}{2} \) accept.
  - With probability \( \frac{1}{2} \) reject.

If \( x \in L \), then there is at least a probability of \( 2^{-p(n)} \) that we pick one of the existing \( y \)'s with \( V(x, y) = \text{Accept} \). Hence in this case

\[
\Pr[\text{Accept}] = 2^{-p(n)} + \frac{1}{2}(1 - 2^{-p(n)}) \geq \frac{1}{2} + 2^{-p(n)-1} > \frac{1}{2}.
\]

On the other hand if \( x \notin L \), then

\[
\Pr[\text{Reject}] \geq \frac{1}{2}.
\]

The proof now follows from Theorem 10.10. \( \square \)

In contrast to \( \text{BPP} \) which some believe to be equal to \( \text{P} \), it is widely believed that \( \text{PP} \) is a much larger class. The following theorem of Toda [Tod91] can be thought of as evidence for this belief.

**Theorem 10.13 (Toda’s theorem).** We have \( \text{PH} \subseteq \text{P}^{\text{PP}} \).

Since \( \text{MAJSAT} \) is \( \text{PP} \)-complete (see Exercise 10.4), this theorem is equivalent to the statement that given an oracle access to \( \text{MAJSAT} \), one can solve any problem in \( \text{PH} \) in polynomial time. In Exercise 10.6 shows that if \( \text{PP} \subseteq \text{PH} \), then the polynomial hierarchy collapses.

**Exercises**

**Exercise 10.1.** Prove that \( \text{ZPP} \) can be alternatively defined as the class of languages for which a probabilistic Turing machine exists that with probability 1 returns the correct \( \text{Accept/Reject} \) answer, and furthermore for every input \( x \), the running time is polynomial in expectation for every input.

- For every \( x \),
  \[
  \Pr_R[M_R(x) \text{ is correct}] = 1.
  \]
• There is a polynomial such that for every $x$,
\[ \mathbb{E}_R[\text{Running time of } M_R \text{ on } x] \leq p(|x|). \]

Exercise 10.2. Prove that the language in Example 10.7 is in BPP.

Exercise 10.3. Prove Theorem 10.10.

Exercise 10.4. Prove that MAJSAT is PP-complete under polynomial time mapping reductions.

Exercise 10.5. Prove that
\[ \text{NP} \subseteq \text{BPP} \iff \text{NP} = \text{RP}. \]

Exercise 10.6. Prove that if MAJSAT $\in$ PH then there is some $k$ such that PH $\subseteq \Sigma_k$. 
Chapter 11

Non-uniform complexity: Circuits

Computational models are divided into two classes: The *uniform models* where a single machine such as a Turing Machine, a context-free grammar, a push-down automaton, or a Finite automaton is supposed to handle the inputs of all sizes, and the *non-uniform* models such as decision trees, circuits, branching programs, communication protocols that can vary depending on the length of the input.

We will start our study of non-uniform complexity theory by discussing circuit complexity. In 1949 Shannon [Sha49] proposed the size of Boolean circuits as a measure of the computational difficulty of a function. Circuits are closely related in computational power to Turing machines, and thus they provide an excellent framework for understanding the time complexity. On the other hand, their especially simple definition makes them amenable to combinatorial, algebraic, probabilistic, and analytic methods. Such a variety of techniques makes circuit complexity one of the richest areas of complexity theory, with an abundance of beautiful results and ingenious proofs with deep connections to other areas of mathematics.

What is a Boolean circuit?

In the discussion of non-uniform models of computation, it is more convenient to work with Boolean functions, rather than sets. A Boolean function on $k$ variables is a map from $\{0, 1\}^k$ to $\{0, 1\}$. It assigns a 0 or 1 value to each vector from $\{0, 1\}^k$.

A *Boolean circuit* is a directed acyclic graph. The vertices of in-degree 0 are called *inputs*. Each input is labeled with a variable $x_i$ or a constant 0 or 1. The vertices of in-degree $k > 0$ are called *gates*, and each such gate is labeled with a $k$-ary Boolean function. In the context of circuits, the in-degrees and out-degrees of vertices are respectively referred to as their fan-ins and fan-outs. The most standard circuits are restricted to have gates $\neg$, $\land$, and $\lor$, where $\land$ and $\lor$ have fan-in 2. One of the nodes of the circuit is designated the *output* node, and with this, the circuit represents a Boolean function in a natural way. Sometimes we allow multiple output nodes to represent functions $f : \{0, 1\}^n \to \{0, 1\}^m$. The size of a circuit is the number of its gates\(^1\).

Example 11.1. Figure 11.1 illustrates a simple circuit with 3 inputs and six gates. It computes a function $f : \{0, 1\}^3 \to \{0, 1\}$. For example, as it is illustrated in the picture, $f(0, 1, 0) = 1$.

---

\(^1\)In some texts, the input nodes are also counted towards the size of the circuit.
Figure 11.1: A circuit with 3 inputs and six gates.

It is sometimes useful to study sets of gates other than \( \neg, \land, \) and \( \lor \). In the context of the circuits, the set of the allowed gates is called a basis. A basis \( B \) is called universal if all Boolean functions can be computed with gates from \( B \).

- \( \{\neg, \land, \lor\} \) is a universal basis, where \( \land \) and \( \lor \) are binary AND and OR functions. To see this recall that every function \( f : \{0,1\}^n \rightarrow \{0,1\} \) can be expressed as a DNF:

\[
 f(x) = \bigvee_{y : f(y) = 1} \left( \left( \bigwedge_{i : y_i = 1} x_i \right) \land \left( \bigwedge_{i : y_i = 0} \bar{x_i} \right) \right).
\]

The right-hand side can be broken further to use only binary \( \land \) and \( \lor \)'s.

- For every \( k \geq 2 \), the set \( B_k \) of all \( 2^k \) functions \( g : \{0,1\}^k \rightarrow \{0,1\} \) is a universal basis.

- \( \{\text{NAND}\} \) by itself is a universal basis, where \( x \text{ NAND } y = \neg(x \land y) \).

- \( \{\land, \lor\} \) is a not a universal basis, but every monotone function can be computed with gates from this set. Here monotone means that changing an input bit from 0 to 1 cannot change the output value from 1 to 0.

- \( \{\land, \oplus\} \) is a universal basis where \( \oplus \) is the binary XOR function.

**Definition 11.2.** Let \( B \) be a basis. The corresponding circuit complexity of a function \( f : \{0,1\}^n \rightarrow \{0,1\} \), denoted by \( \text{size}_B(f) \), is the size of the smallest circuit with gates from \( B \) that computes \( f \). If such a circuit does not exist, then we define \( \text{size}_B(f) \) to be infinity.

One important feature of Definition 11.2 is that as long as finite universal basis are considered, the circuit size is essentially independent of the particular choice of the universal basis, at least up to a constant factor. More precisely if \( B \) and \( B' \) are two finite universal basis, then since we can replace the gates in \( B' \) with \( B \)-circuits and vice versa, there are constants \( c_1, c_2 > 0 \) such that

\[
\text{size}_B(f) \leq c_1 \text{size}_{B'}(f) \leq c_2 \text{size}_B(f),
\]

for every \( f \).

In light of the above discussion, the circuit complexity of a function \( f \) is often defined as the size of the smallest circuit (of fan-in 2) over the basis \( B_2 \), that computes \( f \).

**The class P/Poly**

Consider a deterministic Turing Machine \( M \) with a polynomial running time \( p(n) \). We claim that if we fix the input length \( n \), the computation of \( M \) can be simulated by a polynomial-size fan-in 2 circuit. This can be easily seen by considering the configurations. Since the Turing Machine will never use beyond the first \( p(n) \) cells of the tape, for the sake of convenience, let every configuration contain the content of the first \( p(n) \) cells of the tape, the location of the
tape-head, and the state of the Turing Machine, so that all these configurations are of the same length \( \ell = O(p(n)) \). Note that the computation of \( M \) on an input \( x \) of length \( n \) can be represented as an \( \ell \times p(n) \) table where the \( i \)-th row contains the \( i \)-th configuration. We can convert this to a circuit. The first row forms the inputs of the circuit. It consists of the \( n \) input bits \( x_1, \ldots, x_n \), and an additional set of constant bits, corresponding to the initial state, the initial location of the tape-head, and the rest (the blanks) of the content of the tape. The Turing Machine steps are so elementary that one can easily construct a simple polynomial-size circuit that takes a configuration row as input and outputs the next configuration row. We can put a copy of this circuit between each row and its successor row. Finally, we can feed the last row to a simple circuit that outputs 1 if the row corresponds to an accept configuration and outputs 0 otherwise. We conclude:

**Proposition 11.3.** Let \( M \) be a deterministic Turing Machine with a polynomial running time that decides a language \( L \). There is a \( \text{poly}(n) \)-time algorithm that given \( n \), outputs a \( B_2 \)-circuit \( \phi_n \) such that for every \( x \) with \( |x| = n \),

\[
x \in L \iff \phi_n(x) = 1.
\]

Proposition 11.3 gives an alternative equivalent definition for the class \( P \).

**Definition 11.4** (Alternative Definition of \( P \)). A language \( L \) is in \( P \) if and only if there is a \( \text{poly}(n) \)-time algorithm \( M_L \) that given \( n \), outputs a \( B_2 \)-circuit \( \phi_n \) (of size \( \text{poly}(n) \)) such that for every \( x \) with \( |x| = n \),

\[
x \in L \iff \phi_n(x) = 1.
\]

The uniformity in this definition comes from the fact that a single Turing Machine \( M_L \) can generate these circuits for different values of \( n \). Removing this condition leads to the definition of the non-uniform analog of \( P \) called \( P/\text{Poly} \).

**Definition 11.5.** \( P/\text{Poly} \) is the class of languages \( L \) such that for every \( n \), there is a \( B_2 \)-circuit \( \phi_n \) of size \( \text{poly}(n) \) such that for every \( x \) with \( |x| = n \),

\[
x \in L \iff \phi_n(x) = 1.
\]

Of course, \( P \subseteq P/\text{Poly} \), and one immediately wonders whether the two classes are the same. The answer is a strong “No”, and in fact, \( P/\text{Poly} \) contains some languages that are not even Turing recognizable. For example, every language over the unary alphabet \( \Sigma = \{1\} \) belongs to \( P/\text{Poly} \), since one can take \( \phi_n \equiv 1 \) if \( 1^n \in L \), and \( \phi_n \equiv 0 \) otherwise. Note that some unary languages are not even Turing recognizable as there are uncountably many of them. Also note that every function \( f : \{0,1\}^n \to \{0,1\} \) can be represented as a DNF, which is a circuit of size \( O(n2^n) \). It follows that if we define the class \( \text{EXP}/\text{Exponential} \), analogous to \( P/\text{Poly} \), as the set of the languages with exponential size circuits, then all languages fall into this class.

The above discussion makes a further point. Circuit complexity, as well as other notions of non-uniform computation, are not about computability. They are purely about the number of basic operations needed to compute a function \( f : \{0,1\}^n \to \{0,1\} \). This, in fact, is very useful as many of the important problems in uniform complexity theory seem to be, in essence, about non-uniform lower bounds. Take for example the question of \( P \neq \text{NP} \), which is equivalent to the statement \( \text{SAT} \notin P \). The reason we believe that \( \text{SAT} \notin P \) is not because the problem varies so much from one value of \( n \) to another that makes it impossible for an efficient Turing Machine to solve the problem on all values of \( n \); We simply believe that for large \( n \) it is not possible to solve \( \text{SAT} \) with polynomially many elementary operations. In other words we believe that \( \text{SAT} \notin P/\text{Poly} \). Obviously if we succeed in proving \( \text{SAT} \notin P/\text{Poly} \), then we would establish \( P \neq \text{NP} \). This allows us to focus on the circuit complexity of \( \text{SAT} \), a simpler object to study than the time complexity.

**Conjecture 11.6.** \( \text{SAT} \) has circuit complexity \( 2^{\Omega(n)} \).

**Karp-Lipton Theorem**

In the previous section, we discussed that it is reasonable to believe that \( \text{SAT} \notin P/\text{Poly} \), which would imply \( P \neq \text{NP} \). However, it is not clear from that discussion whether the statement \( \text{SAT} \in P/\text{Poly} \) would have any unexpected implications. Is it possible that \( \text{SAT} \in P/\text{Poly} \), and yet \( P \neq \text{NP} \)? As far as it is known, this is a possibility. However, Karp and Lipton [KL80] showed that if \( \text{SAT} \in P/\text{Poly} \) then the polynomial hierarchy collapses. The original proof of Karp and Lipton collapses \( \text{PH} \) to \( \Sigma_3 \), but Sipser improved it to \( \Sigma_2 \) in the following theorem.

**Theorem 11.7** (Karp-Lipton-Sipser). If \( \text{SAT} \in P/\text{Poly} \), or equivalently \( \text{NP} \subseteq P/\text{Poly} \), then \( \text{PH} = \Sigma_2 \).
Proof. We start with a simple observation: Any algorithm $A$ that can decide whether a given CNF $\phi$ is satisfiable can be modified so that it also outputs a satisfying truth assignment in the case where $\phi$ is satisfiable. Indeed let $x_1, \ldots, x_n$ be the variables. We use $A$ to see if $\phi$ is satisfiable. If it is, then we use $A$ again to see if $\phi|_{x_1=0}$ is still satisfiable. If it is, then we know that we can safely fix $x_1 = 0$, and thus we replace $\phi$ with $\phi|_{x_1=0}$, and continue with the rest of the variables $x_2, x_3, \ldots, x_n$. On the other hand if $\phi|_{x_1=0}$ is not satisfiable, then $\phi|_{x_1=1}$ must be satisfiable, and thus we can fix $x_1 = 1$ and continue similarly.

This argument can also be applied to circuits. That is, assuming $\text{SAT} \in \text{P/Poly}$, there exists a sequence of circuits $C_m$ of polynomial-size $p(m)$ such that given any CNF $\phi$ of size $m$, the output of $C_m(\phi)$ is a truth assignment that satisfies $\phi$ if $\phi$ is satisfiable. We will show that this implies $\Pi_2 \subseteq \Sigma_2$ from which one can easily deduce $\text{PH} = \Sigma_2$ (why?).

Consider the $\Pi_2$-complete problem of $L = \forall \exists \text{SAT}$. Here, for CNF formulas $\phi$,

$$\phi \in L \iff \forall y \exists z \quad \phi(y, z) = \text{TRUE}. $$

Now, by the above discussion, denoting $m = |\phi|$, we have

$$\phi \in L \iff \exists C_m \forall y \quad |C_m| \leq p(m) \quad \operatorname{and} \quad \phi(y, C_m(\phi|_y)) = \text{TRUE}. $$

This shows $L \in \Sigma_2$. \qed

Shannon-Muller Lower bound

We saw that to prove $\text{P} \neq \text{NP}$ it suffices to prove a super-polynomial circuit lower bound for $\text{SAT}$. At first glance, this might not seem a difficult problem considering that a simple counting argument shows that most functions require exponential size circuits: Roughly speaking, there are too many Boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ compared to the number of small circuits.

**Theorem 11.8 (Shannon).** Almost every Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ requires $B_2$ circuits of size $2^n/20n$.

**Proof.** There are exactly $2^{2^n}$ Boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$. The number of circuits with $t$ gates can be upper-bounded as follows: Since $|B_2| = 16$, there are $16^t$ choices assigning gates to nodes. The are $(n+2+t)^2$ choices for the two incoming wires of a gate: The $n$ input variables, the two constant inputs 0 and 1, or the $t$ other nodes. Finally we need to designate one of the $t$ gates as the output gate. Hence the number of circuits of size $t$ over $B_2$ is at most

$$16^t(t + n + 2)^2t. $$

If $t = 2^n/20n$, then

$$\lim_{n \rightarrow \infty} \frac{16^t(t + n + 2)^2t}{2^{2n}} = 0. $$

Thus almost every function has a circuit complexity larger than $2^n/20n$. \qed

On the other hand, we know from the DNF representation that every function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ can be computed by a fan-in 2 circuit of size $O(n2^n)$. In fact, with some extra work (proved first by Shannon and Lupanov), this bound can be improved to $O(2^n/n)$ that matches the lower bound of Theorem 11.8.

Theorem 11.8 has a major shortcoming. It does not provide any explicit examples of functions that require large circuits. Also, unfortunately, it does not provide any example of a function in $\text{NP}$ that requires circuits of super-polynomial size. Note that any function on $n$ bits that depends on all its inputs requires fan-in circuits of size at least $n - 1$, just to read the inputs. Despite the incredible body of research on circuit complexity lower-bounds, the best explicit known construction due to Blum 1984 provides a function that requires fan-in 2 circuits of size $3n - o(n)$. The main open problem of circuit complexity is beating this linear lower bound for natural problems (say, in $\text{NP}$).

**Problem 11.9.** Find an explicit function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ with circuit complexity $\omega(n)$.

Adleman’s theorem and advice: $\text{BPP} \subseteq \text{P/Poly}$

We finish this lecture by proving $\text{BPP} \subseteq \text{P/Poly}$, suggesting that possibly $\text{BPP} = \text{P}$.

**Theorem 11.10 (Adleman [Adl78, BG81]).** $\text{BPP} \subseteq \text{P/Poly}$.
Proof. Suppose $L \in \text{BPP}$. Consider any $n \in \mathbb{N}$. By applying the error-reduction for $\text{BPP}$, we can obtain a polynomial-time randomized algorithm $M_R$ such that for every $x$,

$$\Pr_R[\text{$M_R(x)$ is incorrect}] < \frac{1}{2^n}.$$ 

Since there are at most $2^n$ different inputs $x$ of length $n$, by applying the union bound

$$\Pr_R[\exists x, |x| = n \text{ and } M_R(x) \text{ is incorrect}] < 1.$$ 

Hence there exists a value of $r$ such that fixing the randomness to $r$, the deterministic Turing Machine $M_r$ correctly decides whether $x \in L$ for all $x \in \{0,1\}^n$.

Obviously one can convert $M_r$ to a polynomial size circuit. Hence $L \in \text{P/Pol}$. \qed

The above proof also suggests a different definition of $\text{P/Pol}$ based on “advice” strings.

**Definition 11.11** (Alternative definition of $\text{P/Pol}$). A language $L$ is in $\text{P/Pol}$ if there exists a sequence of advice strings $\{r_1, r_2, \ldots\}$ and a polynomial time Turing Machine $M$ such that $r_n$ is polynomial in size in $n$, and for every $x$,

$$x \in L \iff M(r_{|x|}, x) = \text{ACCEPT}.$$ 

Note that $r_n$ only depends on $n$ and not on the particular choice of $x$. To see the equivalence of this definition and the circuit-based definition of $\text{P/Pol}$, note that we can take $r_n$ to be the polynomial-size circuit $\phi_n$ that computes $L$ on inputs of size $n$. The Turing Machine $M$ plugs $x$ into the circuit and follows the circuit to compute its output. For the other direction, note that given $r_n$, we can convert $M(r_n, x)$ to a polynomial-size circuit.

**Exercises**

**Exercise 11.1.** Prove that the 2-bit NAND is a universal basis.

**Exercise 11.2.** Prove that the following language is $\text{P}$-complete with respect to $\leq_{\ell}$ reductions:

$$\text{CircuitValue} = \{ (\phi, x) : \phi(x) = 1 \},$$

where $\phi$ is an $\{\neg, \vee, \land\}$ circuit. Recall that $\leq_{\ell}$ refers to log-space reduction, and note that using poly-time $\leq_p$ reductions, every nontrivial language in $\text{P}$ would be $\text{P}$-complete.

**Exercise 11.3.** Suppose there is a randomized Turing Machine $M_R$ that decides $L$ with two-sided error probability at most $\varepsilon$. Prove that there exists a set $S$ of size $2^{O(\log(n) + \log(1/\delta))}$ such that

$$\Pr_{R \sim S}[\text{$M_R(x)$ is incorrect}] \leq \varepsilon + \delta,$$

where $R \sim S$ means that $R$ is randomly and uniformly sampled from the rather small set $S$.

**Exercise 11.4.** Prove that for every constant $k$, we have $\Sigma_3 \not\subseteq \text{Size}(n^k)$, where $\text{Size}(n^k)$ is the set of languages with circuit complexity at most $O(n^k)$.

**Exercise 11.5.** Let $C \subseteq \{0,1\}^n$, and $f : C \to \{0,1\}$ be known to us. In the terminology of machine learning, $C$ is called a concept class, and $f(x)$ is called the label of the concept $x \in C$. The goal is to determine the label of an unknown $x$ by making the smallest possible number of queries about $x$. A query model specifies what queries are allowed. Some commonly used query models are coordinate queries $x_i$, parity queries $\oplus_{i \in S} x_i$, and threshold queries $\text{sign}(t - \sum_{i=1}^n w_i x_i)$. Consider a query model $Q$ (e.g., the threshold query model).

The deterministic query complexity of $C$, denoted by $d^Q(C)$, is the smallest $k$ such that the label of every point $x \in C$ can be determined by making at most $k$ queries about $x$. Note that this corresponds to a decision tree of depth at most $k$, where the internal nodes correspond to queries, and the leaves correspond to the predicted labels (see Definition 12.9).

The randomized query complexity of $C$, denote by $\text{rdt}^Q(C)$, is the smallest $k$ such that the label of every point $x \in C$ can be determined with error probability of at most $\frac{1}{3}$ by making at most $k$ random queries. Prove that

$$d^Q(C) = O(\text{rdt}^Q(C) \log |C|).$$
Chapter 12

AC<sup>0</sup>: Bounded Depth Alternating Circuits

Considering our inability to prove lower bounds on the circuit complexity of explicit Boolean functions, we need to impose substantial restrictions on the circuits in order to be able to prove meaningful lower bounds. We will start by restricting to bounded depth circuits. The depth of a circuit is the longest distance from the input nodes to the output node.

While the size of a circuit essentially measures the time required to compute a function using a single simple processor, the depth of a polynomial-size circuit corresponds to the amount of time it takes a parallel algorithm to compute it.

When defining bounded-depth circuits, we need to decide what each parallel processor can compute at a time step. The NC family is defined by such circuits, with ∧, ∨ gates of fan-in 2.

**Definition 12.1** (Nick’s class (NC)). For every \( i \geq 0 \), \( \text{NC}^i \) consists of languages \( L \subseteq \{0,1\}^* \) that can be decided by polynomial-sized circuit families of depth \( O(\log^i n) \) with gates of fan-in 2.

Define \( \text{NC} = \bigcup_{i=0}^\infty \text{NC}^i \).

Since the 2-bit gates ∧ and ∨ together with ¬ form a universal basis, we could have defined the classes \( \text{NC}^i \) equivalently by allowing only these gates.

Due to the bound on fan-in of the gates, \( \text{NC}^0 \) circuits compute functions that only depend on \( O(1) \) input variables. Allowing gates of arbitrary fan-in leads to a more interesting (and more powerful) class of constant depth circuits.

**Definition 12.2** (AC). For every \( i \geq 0 \), \( \text{AC}^i \) consists of languages \( L \subseteq \{0,1\}^* \) that can be decided by polynomial-sized circuit families of depth \( O(\log^i n) \) with ∧, ∨ gates of arbitrary fan-in and ¬ gates.

Define \( \text{AC} = \bigcup_{i=0}^\infty \text{AC}^i \).

The name AC was inspired by NC, with the ‘A’ standing for “alternating” as will be clear later. It is easy to see

\[
\text{NC}^0 \subseteq \text{AC}^0 \subseteq \text{NC}^1 \subseteq \text{AC}^1 \subseteq \text{NC}^2 \subseteq \cdots
\]

Moreover, since each ∧ and ∨ gate of polynomial fan-in can be simulated by a simple \( \text{NC}^1 \) circuit, for every \( i \) we have

\[
\text{NC}^i \subseteq \text{AC}^{i+1},
\]

and thus

\[
\text{AC} = \text{NC} \subseteq P/\text{Poly}.
\]

**Lower-bounds for AC circuits**

We are interested in proving lower bounds for bounded-depth circuits. Since \( \text{NC}^0 \) circuits only depend on a constant portion of their input variables, it is easy to prove lower-bounds for \( \text{NC}^0 \). In particular, any function that depends on all its input variables is not computable by \( \text{NC}^0 \) circuits. For example simple functions such as \( x_1 \wedge x_2 \wedge \cdots \wedge x_n \) do not have constant-depth \( \text{NC}^0 \) circuits. On the other hand, \( \text{AC}^0 \) is more interesting, and it is not obvious how to prove lower-bounds against it. Before we pursue this task, we will make some simplifying assumptions on the structure of the \( \text{AC}^0 \) circuits.

1. Note that by De Morgan’s laws

\[
\neg(p_1 \lor \cdots \lor p_k) = (\neg p_1) \land \cdots \land (\neg p_k),
\]
and
\[ -(p_1 \land \ldots \land p_k) = (\neg p_1) \lor \ldots \lor (\neg p_k). \]

This will allow us to push down the negation gates all the way down to the inputs without much increase in the size of the circuit. Hence we will assume that there are no \( \neg \) gates in the circuit, and instead the inputs are either of the form \( x_i \) or \( \neg x_i \) for variables \( x_i \), or constants 0 and 1.

2. We will assume that the circuits are of the particular form where all \( \land \) and \( \lor \) gates form alternating levels with edges only between each level the immediate level above it. To achieve the latter, we can add the dummy \( \land \) or \( \lor \) gates of fan-in 1 on the wires that skip levels at little cost in circuit size.

3. Finally, we may assume that each gate has a fan-out of exactly 1. To achieve this, we can repeat gates with multiple outputs several times. It is easy to see that this will not increase the depth, and it will at most increase the size by a power of the depth of the circuit (see Exercise 12.1).

In particular, every depth \( d \) circuit with \( S \) gates can be computed by a depth \( d \) circuit with \( O((dS)^d) \) gates which satisfies all the above properties.

**Remark 12.3.** Circuits that satisfy (1) and (2) are called alternating circuits, and it is easy to see that such assumptions can be made without loss of generality for \( \text{AC}^i \) for any \( i \geq 0 \).

On the other hand, circuits that satisfy (3) are called formulas, and only in the constant depth setting do we know of a procedure that turns a circuit into a formula with only a polynomial loss in size.

Note that the depth of an alternating circuit is precisely the number of levels in the circuit.

Alternating circuits of depth 2 are particularly important. Note that because of the “alternation” condition, there are two different types of depth 2 alternating circuits, which correspond to conjunctive normal form (CNF) and disjunctive normal form (DNF) formulas. We recall their definitions next.

**Definition 12.4 (Conjunctive Normal Form, \( \land \) of \( \lor \)).** A formula is in conjunctive normal form, abbreviated to CNF, if it is a conjunction (i.e. \( \lor \)) of clauses, where a clause is a disjunction (i.e. \( \lor \)) of literals (i.e. \( x_i \) or \( \neg x_i \)), where a literal and its negation cannot appear in the same clause. A \( t \)-CNF is a CNF, where each clause consists of at most \( t \) literals.

For example \((x_1 \lor x_2) \land (\neg x_1 \lor x_2 \lor x_3)\) is a formula in conjunctive normal form. It corresponds to an alternating circuit of depth 2 with 3 gates.

**Definition 12.5 (Disjunctive Normal Form, \( \lor \) of \( \land \)).** A formula is in disjunctive normal form, abbreviated to DNF, if it is a disjunction (i.e. \( \lor \)) of conjunctive clauses (i.e. \( \lor \) of literals). A \( t \)-DNF is a DNF, where each clause consists of at most \( t \) literals.

Consider a fixed point \( y = (y_1, \ldots, y_n) \in \{0, 1\}^n \), and \( T = \{i : y_i = 1\} \). Note that the only assignment that satisfies the clause
\[
\left( \bigwedge_{i \in T} x_i \right) \land \left( \bigwedge_{i \notin T} \neg x_i \right)
\]
is the assignment \( x := y \). Hence given a Boolean function \( f : \{0, 1\}^n \to \{0, 1\} \), for every point \( y \) with \( f(y) = 1 \) we can create a clause which is satisfied only if \( x = y \). By taking the \( \lor \) of these clauses we create a DNF formula that represents the function \( f \).
Example 12.6. Consider the function \( f : \{0,1\}^2 \to \{0,1\} \) such that \( f(0,0) = f(0,1) = f(1,1) = 1 \) and \( f(1,0) = 0 \). Then the DNF
\[
(\neg x_1 \land \neg x_2) \lor (\neg x_1 \land x_2) \lor (x_1 \land x_2)
\]
from Fig. 12.1 represents \( f \).

By changing the role of 0’s and 1’s and \( \land \) and \( \lor \), we can represent \( f \) in CNF. We conclude the following observation: the depth 2 alternating circuits are powerful enough to compute any Boolean function.

**Observation 12.7.** Every function \( f : \{0,1\}^n \to \{0,1\} \) can be represented in both DNF and CNF formulas using at most \( 2^n \) clauses.

The DNF/CNF representations show that even alternating circuits of depth 2 are powerful enough to compute every function \( f : \{0,1\}^n \to \{0,1\} \), albeit often requiring exponentially many gates. Thus it is a nontrivial question to establish lower-bounds even for alternating circuits of depth 2.

**Parity and Maj are not in \( AC^0 \)**

The class \( AC^0 \) is one of the few complexity classes that is relatively well understood. We will show that the parity function \( \text{Parity} : \{0,1\}^n \to \{0,1\} \) defined as
\[
\text{Parity}(x) = x_1 \oplus \ldots \oplus x_n,
\]
or equivalently
\[
\text{Parity}(x) = x_1 + \ldots + x_n \mod 2,
\]
does not belong \( AC^0 \). Note that in contrast, if we allow mod 2 gates of arbitrary fan-in, there is a depth 1, size 1 circuit that computes this function.

The fact that \( \text{Parity} \not\in AC^0 \) was first established by Ajtai [Ajt83] and Furst, Saxe, Sipser [FSS84]. They proved super-polynomial lower bounds for any constant depth alternating circuits that compute the parity function. Later Yao [Yao85] gave a sharper exponential lower bound. In 1986 Hästad [Has86] further strengthened and simplified this argument and obtained near-optimal exponential lower bounds.

**Theorem 12.8 ([Has86]).** Any depth \( d \) alternating circuit that computes \( \text{Parity} \) is of size \( 2^\Omega(n^{1/d}) \).

**Proof strategy:** Consider an \( AC^0 \) circuit of depth \( d = O(1) \), and let us assume that the first level (i.e., the gates directly connected to inputs) consists of \( \land \) gates, the second level consists of the \( \lor \) gates, etc. The basic idea of Ajtai [Ajt83] and Furst, Saxe, Sipser [FSS84] for proving lower-bounds on bounded depth \( AC \) circuits was to assign random values to a random subset of variables. Select a random subset of the variables and assign an independent random bit to each of these variables. Such restrictions simplify a small size \( AC^0 \) circuit greatly. Consider one of the \( \land \) gates at level 1. If this gate has a large fan-in, then there is a high chance that our random partial assignment determines its value. Indeed an \( \land \) gate only needs one 0 input to be set to 0. Note that the bottom two circuit levels are a collection of DNF’s (\( \lor \) of \( \land \)’s). After applying the random restriction, all the gates with large fan-in will disappear from the bottom level, and as a result, all these DNF’s will have small clauses. Now we apply a second random restriction, setting values to more randomly chosen variables. A vital element of the proof, known as switching lemma, shows that starting with a DNF with small clauses, an application of such a random restriction will simplify the DNF to a function that has a \emph{decision tree} of small depth \( h \). A decision tree with depth \( h \) can then be converted to an \( h \)-CNF. Hence we can take those DNF’s and replace them with \( h \)-CNFs. Since now we have CNF’s at the bottom, the 2nd and the 3rd level, both being \( \land \) gates, can be collapsed into a single layer. Hence we reduced the depth of the circuit to \( d - 1 \). Repeating this process \( d \) times, we arrive at a function that depends only on \( O(1) \) variables. This conclusion contradicts the assumption that the circuit computes \( \text{Parity} \), as no matter how many of the variables we set (and the values we set them to), the \( \text{Parity} \) function will still depend on all of the remaining variables.

**Decision tree complexity**

A key complexity measure used in the proof of \( \text{Parity} \not\in AC^0 \) is the decision tree complexity of a Boolean function.

**Definition 12.9.** A \emph{decision tree} over variables \( x_1, \ldots, x_n \) is a binary tree where each internal node has two children, left and right. Moreover each internal node is labeled with a variable, and each leaf is labeled with a value 0 or 1. To evaluate a decision tree at a point \( x = (x_1, \ldots, x_n) \), we start from the root, and at each internal node with label \( x_i \) we \emph{query} the value of \( x_i \), go left if \( x_i = 0 \) and right if \( x_i = 1 \) until we reach a leaf. The value of the leaf is the output of the decision tree for \( x \).

For a Boolean function \( f : \{0,1\}^n \to \{0,1\} \), let \( \text{dt}(f) \) be the \emph{depth of the smallest decision tree} computing \( f \).
Obviously, for every Boolean function $f$, we have $\text{dt}(f) \leq n$. And as a result, the parity function is as hard as it gets for decision trees.

**Observation 12.10.** $\text{dt}(\text{Parity}_n) = n$.

A decision tree can be converted to a DNF formula in the following manner: For every leaf $v$ with value 1, include a clause that is true if and only if we take the path from the root to $v$. That is, if $T$ is the set of variables on the path that returned the value 1, and $F$ is the set of the variables that returned the value 0, we include the clause

$$(\bigwedge_{i \in T} x_i) \land (\bigwedge_{i \in F} \neg x_i).$$

One can apply a similar argument to obtain a $\text{dt}(f)$-CNF.

**Proposition 12.11.** Let $f$ be a function with decision tree complexity $\text{dt}(f)$. Then $f$ is computable by a $\text{dt}(f)$-DNF and a $\text{dt}(f)$-CNF with the number of clauses bounded by the number of the leaves.

Figure 12.2: A depth 2 Decision Tree computing the function $f : \{0,1\}^3 \rightarrow \{0,1\}$ that can be represented by 2-DNF $(\neg x_1 \land x_2) \lor (x_1 \land x_3)$ and 2-CNF $(x_1 \lor x_2) \land (\neg x_1 \lor \neg x_3)$.

**Håstad’s switching lemma**

As we mentioned earlier, Håstad obtained near-optimal bounds for the size of a depth $d$ circuit that computes $\text{Parity}$. The core of his proof is an important lemma known as the switching lemma. It is a vital tool for proving lower bounds on the size of constant-depth Boolean circuits.

**Definition 12.12.** Let $X = \{x_1, \ldots, x_n\}$ be the input variables to a circuit $C$ computing a function $f$. A restriction $\rho$ is an element in $\{0,1,*\}^X$.

A restriction $\rho$ sets the values of the variables assigned 0 or 1 and leaves those assigned stars alive. Under $\rho$, we may simplify $C$ by eliminating gates whose values become determined. Call this the induced circuit $C_\rho$ computing the induced function $f_\rho$.

**Lemma 12.13** (Håstad’s switching lemma). Let $f$ be given by a $t$-CNF formula. Choose a random restriction $\rho$ by setting every variable independently to $*$ with probability $p$, and to 0 and 1 each with probability $1 - p$. Then for every $s \in \mathbb{N}$,

$$\Pr[\text{dt}(f_\rho) > s] \leq (5pt)^s.$$ 

In particular for $p = \frac{1}{10t}$,

$$\Pr[\text{dt}(f_\rho) > s] \leq 2^{-s}.$$ 

**Remark 12.14.** Note that the bound in the switching lemma does not depend on the number of clauses in the CNF. The only parameter about the CNF that appears in the assertion is its width $t$.

We will prove the switching lemma by induction on the number $m$ of clauses; however, since the bound does not depend on $m$, we cannot afford to lose anything in the induction step: Starting with the bound $(5pt)^s$ for $t$-CNF’s with $m - 1$ clauses, we must conclude the same bound for $t$-CNFs with $m$ clauses. The general proof strategy is simple. Consider the first clause, and without loss of generality, assume that this clause is $(x_1 \lor \ldots \lor x_t)$.

- (Case I) If the random restriction assigns any 1’s to this clause, then this clause evaluates to 1 and we can remove it and apply the induction hypothesis to the remaining $m - 1$ clauses.

\[1 \{0,1,*\}^X \text{ means the set of functions } \rho : X \rightarrow \{0,1,*\}\]
• (Case II) If the random restriction assigns 0’s to all of \(x_1, \ldots, x_t\), then the clause evaluates to 0, and as a result \(f_\rho \equiv 0\), which satisfies \(\Pr(\text{dt}(f_\rho) = 0)\).

• (Case III) The remaining case is when \(\rho\) assigns some *’s (and no 1’s) to \(x_1, \ldots, x_t\). Let \(T\) be the subset of the variables in this clause that receive *’s. In this case, it suffices to find a decision tree of depth \(s - |T|\) for the remaining \(m - 1\) clauses, as we can extend such a decision tree to a decision tree of depth \(s\) by always querying the values of the variables in \(T\). The induction hypothesis tells us that the probability that the remaining clauses do not have such a decision tree is at most \((5pt)^{-|T|}\). This bound is worse than our goal \((5pt)^s\), but fortunately, *’s are generally unlikely, and the probability that all the variables in \(T\) receive *’s is at most \(p^{|T|}\). Putting these together and taking a union bound over all possibilities of \(T\) gives us the desired bound \((5pt)^s\).

As we mentioned above, we are going to prove this lemma by induction. In the sketched proof, we assumed that what happens in the rest of the \(m - 1\) clauses is independent of the variables \(x_1, \ldots, x_t\). However, this is not the case, for example, in Case I. To deal with this technical issue, we need to strengthen the statement of the lemma.

**Lemma 12.15** (Håstad’s switching lemma, stronger version). Let \(f\) be given by a t-CNF formula. Choose a random restriction \(\rho\) by setting every variable independently to * with probability \(p\), and to 0 and 1 each with probability \(\frac{1-p}{2}\). For every \(s \in \mathbb{N}\), and every function \(F : \{0,1\}^n \rightarrow \{0,1\}\), we have

\[
\Pr[\text{dt}(f_\rho) > s | F_\rho \equiv 1] \leq (5pt)^s, \tag{12.1}
\]

where \(F_\rho \equiv 1\) is the event when \(F_\rho\) is the constant 1 function.

**Proof.** Set \(\alpha := 5pt\), and suppose that \(f = \land_{i=1}^m C_i\) where \(C_i\)’s are clauses of size at most \(t\). We prove this statement by induction on \(m\) the number of clauses in \(f\). If \(m = 0\), then \(f \equiv 1\) and the lemma is obvious. For the induction step let us study what happens to \(C_1\), the first clause in the circuit. First note that by possibly changing the role of 0’s and 1’s for some variables, we can assume without loss of generality that there are no negated literals in \(C_1\) and hence

\[
C_1 = \bigvee_{i \in T} x_i,
\]

for a subset \(T \subseteq \{1, \ldots, n\}, |T| \leq t\). First we split the LHS of Eq. (12.1) into two terms based on whether \(C_1\) receives a 1 from the restriction:

\[
\Pr[\text{dt}(f_\rho) > s | F_\rho \equiv 1] = \Pr[\text{dt}(f_\rho) > s, \rho_T \notin \{0,*\}^T | F_\rho \equiv 1] + \Pr[\text{dt}(f_\rho) > s, \rho_T \in \{0,*\}^T | F_\rho \equiv 1].
\]

Hence in order to prove (12.1) it suffices to prove both

\[
\Pr[\text{dt}(f_\rho) > s | F_\rho \equiv 1, \rho_T \notin \{0,*\}^T] \leq \alpha^s, \tag{12.2}
\]

and

\[
\Pr[\text{dt}(f_\rho) > s | F_\rho \equiv 1, \rho_T \in \{0,*\}^T] \leq \alpha^s, \tag{12.3}
\]

as then we would have

\[
\Pr[\text{dt}(f_\rho) > s | F_\rho \equiv 1] \leq \Pr[\rho_T \notin \{0,*\}^T | F_\rho \equiv 1] \alpha^s + \Pr[\rho_T \in \{0,*\}^T | F_\rho \equiv 1] \alpha^s = \alpha^s.
\]

To prove (12.2) note that for \(g = \land_{i=2}^m C_i\) (which has only \(m - 1\) clauses),

\[
\text{L.H.S of (12.2) } = \Pr[\text{dt}(g_\rho) > s | F_\rho \equiv 1, \rho_T \notin \{0,*\}^T] = \Pr[\text{dt}(g_\rho) > s | (F \land C_1)_\rho \equiv 1] \leq \alpha^s,
\]

where in the last inequality we used the induction hypothesis applied to \(g\) and \(F \land C_1\). It remains to prove (12.3). We break (12.3) into \(2^{|T|}\) terms based on which coordinates in \(T\) are *’s and which ones are 0’s:

\[
\text{L.H.S of (12.3) } = \sum_{Y \subseteq T} \Pr[\text{dt}(f_\rho) > s, \rho_Y = *, \rho_{T-Y} = 0 | F_\rho \equiv 1, \rho_T \in \{0,*\}^T] \\
\leq \sum_{Y \subseteq T} \Pr[\rho_Y = *, \rho_{T-Y} = 0 | F_\rho \equiv 1, \rho_T \in \{0,*\}^T] \times \Pr[\text{dt}(f_\rho) > s | F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = 0, \rho_T \in \{0,*\}^T] \\
\leq \sum_{Y \subseteq T} \Pr[\rho_Y = * | F_\rho \equiv 1, \rho_T \in \{0,*\}^T] \times \Pr[\text{dt}(f_\rho) > s | F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = 0].
\]

65
First note that if $Y = \emptyset$, then $\rho T = 0$, and thus $C_1$ is not satisfied and $f_\rho \equiv 0$, and consequently $dt(f_\rho) = 0$. Hence we can remove the corresponding term from the above calculation and obtain:

$$L.H.S$ of \eqref{eq:12.3} \leq \sum_{\substack{Y \subseteq T \\ Y \neq \emptyset}} \text{Pr} \left[ \rho Y = * | F_\rho \equiv 1, \rho_T \in \{0, * \}^T \right] \times \text{Pr} \left[ dt(f_\rho) > s | F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = 0 \right]. \quad \text{(12.4)}$$

We bound the two terms in the product separately.

**First observation (bounding $\text{Pr} \left[ \rho Y = * | F_\rho \equiv 1, \rho_T \in \{0, * \}^T \right]$):** Since setting variables in $Y$ to $*$ cannot increase the probability that $F_\rho \equiv 1$, we have

$$\text{Pr}[F_\rho \equiv 1 | \rho_Y = *, \rho_T \in \{0, * \}^T] \leq \text{Pr}[F_\rho \equiv 1 | \rho_T \in \{0, * \}^T],$$

Hence using $\text{Pr}[A|B] \text{Pr}[B] = \text{Pr}[A \land B]$ we have

$$\text{Pr}[\rho_Y = * | F_\rho \equiv 1, \rho_T \in \{0, * \}^T] = \frac{\text{Pr}[F_\rho \equiv 1 | \rho_Y = *, \rho_T \in \{0, * \}^T]}{\text{Pr}[F_\rho \equiv 1 | \rho_T \in \{0, * \}^T]} \text{Pr}[\rho_Y = * | \rho_T \in \{0, * \}^T] \leq \text{Pr}[\rho_Y = * | \rho_T \in \{0, * \}^T] = \left( \frac{2p}{1+p} \right)^{|Y|} \leq (2p)^{|Y|}$$

**Second observation: (bounding $\text{Pr} \left[ dt(f_\rho) > s | F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = 0 \right]$):** Note that the variables in $Y$ can contribute by at most $|Y|$ to the decision tree depth, or more precisely if for every $\sigma \in \{0, 1 \}^{|Y|}$, we have $dt(f_\sigma) \leq s - |Y|$, then $dt(f_\rho) \leq s$. Indeed to verify this, note that we can always build a decision tree of depth at most $|Y| + \max_{\sigma} dt(f_\sigma)$ as follows: In the first $|Y|$ levels, we query all the variables $x_i$ for $i \in Y$ to obtain a $\sigma \in \{0, 1 \}^Y$, and then we follow a decision tree of depth $dt(f_\sigma)$ afterwards. Hence for $Y \neq \emptyset$, recalling that $g = \land_{i=2}^m C_i$, we have

$$\text{Pr}[dt(f_\rho) > s | F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = 0] \leq \text{Pr} \left[ \exists \sigma \in \{0, 1 \}^{|Y|}, \ dt(f_\sigma) > s - |Y| | F_\rho \equiv 1, \rho_{T-Y} = 0 \right]$$

$$\leq \sum_{\sigma \in \{0, 1 \}^{|Y|}} \text{Pr}[dt(f_\sigma) > s - |Y| | F_\rho \equiv 1, \rho_{T-Y} = 0]$$

$$= \sum_{\sigma \in \{0, 1 \}^{|Y|}} \text{Pr}[dt(f_\sigma) > s - |Y| | (F \land \land_{i \in T \setminus Y} \mathbf{x}_i)p \equiv 1]$$

$$= \sum_{\sigma \in \{0, 1 \}^{|Y|}} \text{Pr}[dt(g_\sigma) > s - |Y| | (F \land \land_{i \in T \setminus Y} \mathbf{x}_i)p \equiv 1]$$

$$\leq \sum_{\sigma \in \{0, 1 \}^{|Y|}} \alpha^{|Y|} \alpha^{|Y|} \leq 2^{|Y|} \alpha^{|Y|} = \alpha^{|Y|} \leq 2^{|Y|} \alpha^{|Y|}.$$

where we applied the union bound, and then the induction hypothesis.

Combining the two observations with \eqref{eq:12.4}, we finish the proof:

$$L.H.S$ of \eqref{eq:12.3} \leq \sum_{\substack{Y \subseteq T \\ Y \neq \emptyset}} 2^{|Y|} \alpha^{|Y|} (2p)^{|Y|} = \alpha^{|Y|} \sum_{\substack{Y \subseteq T \\ Y \neq \emptyset}} \left( \frac{4p}{\alpha} \right)^{|Y|} = \alpha^{|Y|} \left( 1 + \frac{4p}{\alpha} \right)^{|Y|} - 1 \right)$$

$$= \alpha^{|Y|} \left( 1 + \frac{4}{\alpha} \right)^d - 1 \right) \leq \alpha^s (e^{\frac{4}{\alpha}} - 1) \leq \alpha^s.$$  

\[\square\]

**Remark 12.16.** Since the negation of a CNF is a DNF of similar size and vice versa, the switching lemma can be used to convert a $t$-DNF formula to an $s$-CNF in the same way as Lemma 12.15.

**Corollary 12.17.** Let $f$ be a Boolean function computed by an AC circuit of size $M$ and depth $d$ whose output gate is $\land$. Choose a random restriction $\rho$ by setting every variable independently to $*$ with probability $p = \frac{1}{10^s - 1}$, and to 0 and 1 each with probability $\frac{1-p}{2}$. Then

$$\text{Pr}[dt(f_\rho) > s] \leq M 2^{-s}.$$  

**Proof.** We view the restriction $\rho$ as obtained by first having a random restriction $\rho_0$ with $\text{Pr}[\rho] = 1/10$, and then $d - 1$ consecutive restrictions $\rho_1, \ldots, \rho_{d-1}$ each with $\text{Pr}[\rho] = \frac{1}{10^s}$. With high probability, after the restriction $\rho_0$, at the bottom level of the circuit all fan-ins are at most $s$. To see this, consider two cases for each gate at the bottom level of the original circuit:  

66
1. The original fan-in is at least 2s. In this case, the probability that the gate was not eliminated by \( \rho_0 \), that is, that no input to this gate got assigned a 1 (assuming without loss of generality that the bottom level is an \( \lor \) level) is at most \((0.55)^{2s} < 2^{-s}\).

2. The original fan-in is at most 2s. In this case, the probability that at least \( s \) inputs got assigned a * by \( \rho_0 \) is at most \((2^s)(1/10)^s \leq 2^{-s}\).

Thus, the probability of failure after the first restriction is at most \( m_1 2^{-s} \), where \( m_1 \) is the number of gates at the bottom level.

We now apply the next \( d-2 \) restrictions, each with \( \Pr[x] = \frac{1}{|M|} \). After each of these, we use Håstad’s switching lemma (see Remark 12.16) to convert the lower two levels from CNF to DNF (or vice versa), collapse the second and third levels (from the bottom) to one level, reducing the depth by one. For each gate of distance two from the inputs, the probability that it corresponds to a function \( g \) with \( dt(g) > s \), is hence bounded by \((5 \frac{1}{10s})^s \leq 2^{-s}\). The probability that a particular gate fails to satisfy the desired property is no more than \( 2^{-s} \). Since the top gate is \( \land \), after these \( d-2 \) stages, we are left with a CNF formula of bottom fan-in at most \( s \). We now apply the last restriction and by switching lemma we get a function \( f_{\rho} \) with \( dt(f_{\rho}) \leq s \). The probability of failure at this stage is at most \( 2^{-s} \). To compute the total probability of failure, we observe that each gate of the original circuit contributed \( 2^{-s} \) probability of failure exactly once, and hence applying the union bound yields the desired bound.

Using the above statement, if \( M2^{-s} \) is bounded away from 1, and \( s << pn \), then with positive probability over the restriction the circuit simplifies to a decision tree of depth at most \( s \) while the number of surviving variables is larger than \( pn \) while \( \text{PARITY}_\rho \) requires decision trees of full depth (Observation 12.7), which is \( > s \). Comparing the parameters proves Theorem 12.8.

Note that if in the above proof we stop before applying the last restriction \( \rho_{d-1} \), then we obtain the following corollary, which uses a larger value for \( p \).

**Corollary 12.18.** Let \( f \) be a Boolean function computed by an AC circuit of size \( M \) and depth \( d \geq 2 \) whose output gate is \( \land \). Choose a random restriction \( \rho \) by setting every variable independently to * with probability \( p = \frac{1}{10^{d-2} + s} \), and to 0 and 1 each with probability \( \frac{1}{2} - p \). Then

\[
\Pr[f_{\rho} \text{ does not have a CNF with fan-in } \leq s] \leq M2^{-s}.
\]

Similarly, if the output gate of the original circuit is \( \lor \), then the probability that \( f_{\rho} \) does not have a DNF with fan-in \( \leq s \) is bounded by \( M2^{-s} \).

In the next section, we will show that this improvement implies a better lower bound of \( 2^{\Omega(n^{1/d})} \) for \( \text{PARITY} \), as well as a lower-bound for the Majority function.

**Influences in bounded depth circuits**

Let us introduce an essential notion in the study of Boolean functions, namely the influence of a variable.

**Definition 12.19 (Influence).** Let \( f : \{0,1\}^n \to \{0,1\} \). The influence of the \( i \)-th variable on \( f \) is the probability that changing the \( i \)-th coordinate changes the the value of \( f \). That is,

\[
I_i(f) = \Pr[f(x) \neq f(x \oplus e_i)],
\]

where \( x \in \{0,1\}^n \) is sampled uniformly and \( e_i \) is the \( i \)-th standard basis vector, and \( \oplus \) refers to entry-wise XOR. The total influence of \( f \) is defined as

\[
I_f = \sum_{i=1}^n I_i(f).
\]

Note that always \( 0 \leq I_i(f) \leq 1 \) and \( 0 \leq I_f \leq n \). The parity function has total influence \( n \), and a constant function has total influence 0. The influence of the \( i \)-th variable on \( f \) captures the changes in \( f \) with respect to the changes in the \( i \)-th coordinate in the input. In that sense it is the discrete analogue of the average (magnitude) of the partial derivative of a continuous function \( h \) with respect to a particular variable \( \int \frac{\partial h}{\partial x_i} = \int \lim_{\delta \to 0} \frac{h(x_i + \delta e_i) - h(x)}{\delta} \).

The sensitivity of a point \( x \) with respect to \( f \), denoted by \( s_f(x) \), is the number of coordinates \( i \) for which \( f(x) \neq f(x \oplus e_i) \). This is the discrete analogue of the magnitude of the gradient in the setting of the differentiable functions. Writing

\[
I(f) = \sum_{i=1}^n \Pr[f(x) \neq f(x \oplus e_i)],
\]

67
linearity of expectation shows
\[ I_f = \mathbb{E}[s_f(x)], \] (12.5)
and thus sometimes \( I_f \) is called the \textit{average sensitivity} of \( f \).

Our next goal is to show that the total influence of small circuits of small depth cannot be large. First, we consider the CNF and the DNF circuits with small clauses.

\textbf{Lemma 12.20.} Let \( f \) be a CNF or a DNF formula where all the clauses are of size at most \( t \). Then \( I_f \leq 2t \).

\textit{Proof.} We prove the lemma for the DNF case, and the CNF case follows by replacing \( \mathbb{E} \) with \( \mathbb{E}^* \). Let \( f \) be a Boolean function computed by an AC circuit of depth \( d \) and size \( M \) (including the input gates), then
\[ I_f \leq (20 \log M)^d. \]

\textit{Proof.} Applying Corollary 12.17 with \( s = 2 \log M \) and \( p = \frac{1}{10^{20^n \log n}} \), and combining it with the easy fact that \( I_f \leq d I(f) \), we conclude that
\[ \mathbb{E}_\rho[I_{f_\rho}] \leq M2^{-s} \leq \frac{1}{M} \leq \frac{1}{n}. \]

Here we are using \( n \leq M \) by counting the input gates in the size of the circuit. Hence
\[ \mathbb{E}_\rho[I_{f_\rho}] \leq \mathbb{P}(I_{f_\rho} > s) n + s \leq \frac{1}{n} n + s \leq s + 1 \leq 2s. \]

On the other hand, note that for every \( i \),
\[ \mathbb{P}_x[f_\rho(x) \neq f_\rho(x \oplus e_i)] = p \mathbb{P}_x[f(x) \neq f(x \oplus e_i)], \]
where \( p \) is the probability that the \( i \)th variable is not fixed by \( \rho \). Hence
\[ \mathbb{E}_\rho[I_{f_\rho}] = p I_f. \]
We conclude
\[ I_f \leq \frac{2s}{p} \leq 2s \cdot (10s)^{d-1} = (20 \log M)^d. \]

\textbf{Theorem 12.22} (Boppana [Bop97]). Let \( f \) be a Boolean function computed by an AC circuit of depth \( d \) and size \( M \), then
\[ I_f \leq (20 \log M)^{d-1}. \]

The majority function \( \text{Maj} \) is defined as \( \text{Maj}(x) = 1 \) if and only if \( \sum x_i \geq n/2 \). As we saw above the total influence of \( \text{Parity} \) is \( n \). It is also straightforward to see (Exercise 12.2) that the total influence of \( \text{Maj} \) is \( \Theta(\sqrt{n}) \). We conclude

\textbf{Corollary 12.23.} The functions \( \text{Parity} \) and \( \text{Maj} \) are not in \( \text{AC}^0 \). In particular, \( \text{Parity} \) and \( \text{Maj} \) require AC circuits of depth \( d \) and size at least \( 2^{\Omega(n^{\frac{1}{d-1}})} \) and \( 2^{\Omega(n^{\frac{1}{\sqrt{d-1}}})} \) respectively.
Exercises

Exercise 12.1. Prove that every depth $d$ alternating circuit of size $S$ can be computed by a depth $d$ formula of size at most $S^d$. A circuit is called a formula if its gates have fan-out at most 1.

Exercise 12.2. Prove that the total influence of $\text{Maj} : \{0, 1\}^n \to \{0, 1\}$ is $\Theta(\sqrt{n})$.

Exercise 12.3. Consider the DNF $T_{r,t}(x) = \bigvee_{i=1}^r \bigwedge_{j=1}^t x_{ij}$ on $n = rt$ variables $x_{ij}$. For $r = n/t$ and $t = \log(n) - \log \log(n)$ prove that $I_i(T_{r,t}) = \Theta(\frac{\log(n)}{n})$.

Exercise 12.4. Prove that $\text{Var}[f] \leq I_f$.

Exercise 12.5. Prove that every function $f : \{0, 1\}^n \to \mathbb{R}$ has a unique polynomial representation $f(x) = \sum_{S \subseteq \{1, \ldots, n\}} \alpha_S x^S$, where $x^S$ denotes the monomial $\prod_{i \in S} x_i$. Prove that the degree of this polynomial $\text{deg}(f)$ is bounded by $\text{dt}(f)$. Is it true that $\text{deg}(f) \leq t$ if $f$ has a $t$-CNF representation?

Exercise 12.6. Let $\text{size}_{dt}(f)$ denote the smallest number of leaves in a decision tree computing $f$. Prove that for every $f : \{0, 1\}^n \to \{0, 1\}$ if $\rho$ is a random restriction with parameter $p$ (that is $\Pr[\ast] = p$), we have

$$\mathbb{E}_p[\text{size}_{dt}(f_{\rho})] \leq \text{size}_{dt}(f)^{\log(1+p)}.$$  

Prove that this bound cannot be improved by giving an example where the bound is sharp.
Chapter 13

\( \text{AC}^0 \) with parity gates: Razborov-Smolensky

In the last few sections, we saw that \( \text{AC} \) circuits of bounded depth cannot compute simple functions such as \( \text{PARITY} \) and \( \text{MAJ} \). A natural way to strengthen such circuits is to allow gates capable of computing such hard functions. In particular, two classes \( \text{TC}^0 \) and \( \text{ACC}^0 \) are defined as follows.

For \( i \geq 0 \), let \( \text{TC}^i \) be the class of decision problems decidable by Boolean threshold circuits with a polynomial number of gates and depth \( O(\log^i n) \). Threshold circuits are Boolean circuits with \( \land, \lor, \neg \), and \( \text{MAJ} \) gates. Since \( \text{MAJ} \in \text{NC}^i \subseteq \text{AC}^i \), we have

\[ \text{AC}^i \subseteq \text{TC}^i \subseteq \text{NC}^{i+1} \subseteq \text{AC}^{i+1}. \]

Our understanding of the class \( \text{TC}^0 \) is very limited. It is not known whether \( \text{P} \nsubseteq \text{TC}^0 \).

On the other hand, define \( \text{AC}^i[m] \) to be the class of decision problems decidable by Boolean circuits with ability to count modulo the integer \( m \), with a polynomial number of gates and depth \( O(\log^i n) \). More specifically, \( \text{AC}^i[m] \) circuits are allowed gates \( \land, \lor, \neg \) in addition to \( \text{mod}_m(x) \) gates which evaluate to 1 if and only if \( \sum x_i \equiv 0 \mod m \). Define \( \text{ACC}^i = \bigcup_m \text{AC}^i[m] \). It is easy to verify that

\[ \text{AC}^i \subseteq \text{ACC}^i \subseteq \text{TC}^i. \]

It is conjectured that \( \text{MAJ} \notin \text{AC}^0[6] \), however, it is not yet ruled out whether \( \text{P}/\text{poly} \nsubseteq \text{AC}^0[6] \). It is not even ruled out whether \( \text{EXP} \nsubseteq \text{AC}^0[6] \). The strongest lower-bounds known for \( \text{ACC}^0 \) are due to recent work of Murray and Williams [MW18] who built on a breakthrough of Williams [Wil14] to prove that \( \text{NQP} \nsubseteq \text{ACC}^0 \). Here \( \text{NQP} \) refers to nondeterministic quasipolynomial time.

The case of \( \text{AC}^0[2] \) (also denoted \( \text{AC}^0[\oplus] \)) turns out to be very different, and as we will see in this section, much stronger lower-bounds can be proved for very simple functions.

While of the two hard functions \( \text{PARITY} \) and \( \text{MAJ} \), the former belongs to \( \text{AC}^0[\oplus] \), Razborov [Raz87] and Smolensky [Smo87] proved that the latter, \( \text{MAJ} \), is hard even for \( \text{AC}^0[\oplus] \). Their proof has an important implication that every \( \text{AC}^0 \) function can be approximated point-wise by a low-degree polynomial.

The key to answering such questions is the polynomial representation of Boolean functions. Every function \( f : \{0, 1\}^n \rightarrow \mathbb{R} \) has a unique representation as a multivariate polynomial in \( \mathbb{R}(x_1, \ldots, x_n) \) with monomials \( \prod_{i \in S} x_i \). Since the variables take only 0 and 1 values, every variable appears with power 0 or 1 in each monomial.

**Example 13.1.** We have

\[ \text{MAJ}(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_1x_3 - 2x_1x_2x_3. \]

\[ \text{PARITY}(x_1, x_2, x_3) = x_1 + x_2 + x_3 - 2x_1x_2 - 2x_2x_3 - 2x_1x_3 + 4x_1x_2x_3. \]

\[ \land(x_1, \ldots, x_n) = x_1x_2 \cdots x_n, \]

and

\[ \lor(x_1, \ldots, x_n) = 1 - (1 - x_1)(1 - x_2) \cdots (1 - x_n) = \sum_{\emptyset \subseteq S \subseteq [n]} (-1)^{|S|+1} \prod_{i \in S} x_i. \]

Note that even the simplest \( \text{AC}^0 \) functions such as \( \land \) and \( \lor \) have a full degree as a polynomial. However, the following theorem shows that every \( \text{AC}^0 \) function can be approximated well with a low-degree polynomial.

71
Theorem 13.2 ([Raz87], [Smo87]). Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be computed by an AC circuit of depth $d$ and size $M$. For every $s$, there is a polynomial $g \in \mathbb{Z}[x_1, \ldots, x_n]$ with degree $\leq (s \log M)^d$ such that

$$\Pr_x[f(x) \neq g(x)] \leq \left(1 - \frac{1}{2e}\right)^s M,$$

where $x$ is chosen randomly and uniformly from $\{0, 1\}^n$. In particular, taking $s = 100 \log(M)$, there is a polynomial of degree $\leq (100 \log M)^d$, such that

$$\Pr_x[f(x) \neq g(x)] \leq 0.01.$$

Proof. The key is to approximate the $\land$ and $\lor$ gates with low-degree polynomials. The function $g$ is constructed inductively. We will show how to make a step with an $\land$ gate. Since the whole construction is symmetric with respect to 0 and 1, the step also holds for an $\lor$ gate. Let

$$f = \land_{i=1}^k f_i$$

where $k < M$. For convenience, let us assume that $k = 2^t$ is a power of 2. For every $j = 1, \ldots, t$, pick $r$ random subsets of $\{1, \ldots, k\}$ by including every element in the subset independently with probability $p = 2^{-j}$. We obtain a collection of sets $S_1, \ldots, S_t$ with $t := sl \leq s \log M$. Let $g_1, \ldots, g_k$ be the approximating functions for $f_1, \ldots, f_k$ provided by the previous inductive step. We set

$$g := \prod_{i=1}^t (1 - |S_i| + \sum_{j \in S_i} g_j).$$

By the induction assumption, the degrees of $g_j$ are $\leq (s \log M)^{d-1}$. Hence, the degree of $f$ is bounded by $t(s \log M)^{d-1} \leq (s \log M)^d$. Next, we bound the probability of $f(x) \neq g(x)$ conditioned on the event that all of the inputs $f_1, \ldots, f_k$ are computed correctly. We have

$$\Pr[f(x) \neq g(x) | g_j(x) = f_j(x) \text{ for all } j] \leq \Pr \left[ \prod_{i=1}^t \left(1 - |S_i| + \sum_{j \in S_i} f_j\right) \neq \prod_{j=1}^k f_j \right].$$

To bound this, we fix a vector of specific values $f_1(x), \ldots, f_k(x)$ and calculate the probability that an error occurs over the possible choices of the random sets $S_i$.

- Note that if all the $f_j(x)$'s are 1, then the value of $f(x) = 1$ is calculated correctly with probability 1.
- Suppose that $f(x) = 0$, and thus at least one of the $f_j$'s is 0. Note that in order for the product

$$\prod_{i=1}^t \left(1 - |S_i| + \sum_{j \in S_i} f_j\right)$$

to evaluate to 0, it suffices to have one of the terms $1 - |S_i| + \sum_{j \in S_i} f_j$ to be 0. Let $1 \leq z \leq k$ be the number of zeros among $f_1(x), \ldots, f_k(x)$, and $\alpha \in \mathbb{Z}$ be such that $2^\alpha \leq z < 2^{\alpha+1}$. Let $S$ be a random set with parameter $p = 2^{-\alpha-1}$. Our approximation will be correct if $S$ hits exactly one 0 among the $z$ zeros of $f_1(x), \ldots, f_k(x)$, as in this case, we would get $1 - |S| - \sum_{j \in S} f_j = 0$, making the whole product 0. The probability of this event is exactly

$$zp(1-p)^{z-1} > \frac{1}{2}(1-p)^{\frac{k}{2}-1} > \frac{1}{2e}.$$

Hence the probability that all the $s$ sets that are chosen with parameter $p = 2^{-\alpha-1}$ fail is bounded by $(1 - \frac{1}{2e})^s$ and

$$\Pr \left[ \prod_{i=1}^t (1 - |S_i| + \sum_{j \in S_i} f_j) \neq \prod_{j=1}^k f_j \right] < \left(1 - \frac{1}{2e}\right)^s.$$

By making the same probabilistic argument at every node, by the union bound over all the $\leq M$ gates in the circuit, we conclude that the probability that an error happens is at most $M \left(1 - \frac{1}{2e}\right)^s$. Hence the polynomial $g$ that we randomly constructed satisfies: For every $x \in \{0, 1\}^n$,

$$\Pr_g[f(x) \neq g(x)] \leq M \left(1 - \frac{1}{2e}\right)^s.$$
Since this holds for every $x$, we have
\[ \Pr_{g,x}[f(x) \neq g(x)] \leq M \left( 1 - \frac{1}{2e} \right)^s, \]
which shows that there is a fixed polynomial $g_0$ such that
\[ \Pr_x[f(x) \neq g_0(x)] \leq M \left( 1 - \frac{1}{2e} \right)^s, \]
as desired.

\[ \square \]

**The class $\text{AC}^0[\oplus]$: Algebraic techniques**

In this section, we study the class $\text{AC}^0[\oplus]$ and ultimately show that $\text{Maj} \not\in \text{AC}^0[\oplus]$. Since we are interested in “mod 2” additions, working in the two-element field $\mathbb{F}_2$ is more convenient. Here $\mathbb{F}_2 = \{0, 1\}$ with addition and multiplication defined modulus 2. In particular, addition corresponds to the $\oplus$ function.

**Boolean functions as $f : \mathbb{F}_2^n \to \mathbb{F}_2$:** Identifying $\{0, 1\}$ with $\mathbb{F}_2$, we can think of Boolean functions as $f : \mathbb{F}_2^n \to \mathbb{F}_2$. Note that the set of all functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is a $2^n$ dimensional vector space over the field $\mathbb{F}_2$. One obvious linear basis for this vector space is the set of Dirac functions $\delta_y$ defined as
\[ \delta_y : x \mapsto \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}. \]

Indeed there are $2^n$ Dirac functions, and every function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be expressed as a linear combination of $\delta_y$'s with coefficients from $\mathbb{F}_2$ as
\[ f = \sum_{y \in \mathbb{F}_2^n} f(y) \delta_y. \]

Monomials $\prod_{i \in S} x_i$ form a different set of linear basis for this vector space. This basis lead to the polynomial representation of $f$ in the ring of polynomials $\mathbb{F}_2[x_1, \ldots, x_n]$. To be more precise, for every $S \subseteq \{1, \ldots, n\}$, define the function $x^S = \prod_{i \in S} x_i$. Here we use the convention $x^\emptyset = \prod_{i \in \emptyset} x_i \equiv 1$. Now let us establish that these functions are linearly independent.

**Lemma 13.3.** The monomials $x^S$, where $S \subseteq \{1, \ldots, n\}$, are linearly independent over $\mathbb{F}_2$.

**Proof.** Consider any ordering $S_1, S_2, \ldots, S_{2^n}$ of subsets of $\{1, \ldots, n\}$ such that $i \leq j$ implies $|S_i| \geq |S_j|$. For every $S \subseteq \{1, \ldots, n\}$ denote $\phi_S(x) = x^S$. Moreover, identify $S$ with the vector $y \in \mathbb{F}_2^n$ defined as $y_i = 1 \iff i \in S$. With this notation, note that
\[ \phi_S(S) = 1, \]
while
\[ \phi_T(S) = 0, \]
for all $T \neq S$ with $|T| \geq |S|$. This shows that
\[ \phi_S \notin \text{span}_{\mathbb{F}_2}\{\phi_{S_j} : j < i\}. \]

Thus $\phi_{S_1}, \phi_{S_2}, \ldots, \phi_{S_{2^n}}$ are linearly independent. \[ \square \]

Since there are exactly $2^n$ monomials $x^S$, which matches the dimension of the vector space of all functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$, we conclude that they must form a linearly independent basis. Consequently, every such function has a unique polynomial representation
\[ f(x) = \sum_{S \subseteq \{1, \ldots, n\}} a_S x^S. \]

The proof of Lemma 13.3 has the following corollaries that we will use later in the lecture.

**Corollary 13.4.** For every $A \subseteq \mathbb{F}_2^n$, the set of monomials $x^S$ for $S \in A$ is a linear basis for the $\mathbb{F}_2$-vector space of all functions $f : A \to \mathbb{F}_2$. (Here, we are identifying $y \in A \subseteq \mathbb{F}_2^n$ with the set $S = \{i : y_i = 1\}$).

**Corollary 13.5.** For every $A \subseteq \mathbb{F}_2^n$, the set of monomials $(1-x)^S := \prod_{i \in [n] \setminus S} (1-x_i)$ for $S \in A$ is a linear basis for the $\mathbb{F}_2$-vector space of all functions $f : A \to \mathbb{F}_2$. 

73
Razborov-Smolensky over $\mathbb{F}_2$

In Theorem 13.2, we saw how to approximate every $\mathsf{AC}$ circuit of size $M$ and depth $d$ with a low degree polynomial in $\mathbb{Z}(x_1, \ldots, x_n)$. The result immediately translates over to $\mathbb{F}_2$. Indeed simplifying the coefficients modulus 2, we obtain a low-degree polynomial in $\mathbb{F}_2(x_1, \ldots, x_n)$ that approximates the circuit.

Now consider a circuit in $\mathsf{AC}^0[\oplus]$. It is no longer true that such a circuit can be approximated well by a low-degree polynomial in $\mathbb{Z}(x_1, \ldots, x_n)$, simply because even a single $\oplus$-gate with a large fan-in (i.e., the parity function) does not have such a low degree approximation. However, over the field $\mathbb{F}_2$ this changes, as the $\oplus$-gates are simply linear polynomials over $\mathbb{F}_2$:

$$\bigoplus_{i=1}^{k} f_i = \sum_{i=1}^{k} f_i,$$

where the addition is in $\mathbb{F}_2$. We conclude the following version of Theorem 13.2.

**Theorem 13.6** (Theorem 13.2 over $\mathbb{F}_2$). Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be computed by an $\mathsf{AC}[\oplus]$ circuit of depth $d$ and size $M$. For every $s$, there is a polynomial $g \in \mathbb{F}_2(x_1, \ldots, x_n)$ of degree $\leq (s \log M)^d$ such that

$$\Pr_x[f(x) \neq g(x)] \leq \left(1 - \frac{1}{2e}\right)^s M,$$

where $x$ is chosen randomly and uniformly from $\{0, 1\}^n$.

**Proof.** The $\oplus$-gates are linear functions:

$$\bigoplus_{i=1}^{k} f_i = \sum_{i=1}^{k} f_i.$$

For the $\land$ and $\lor$ gates we can use the same approximation as in Theorem 13.2 but simplify everything mod 2, or equivalently carry all the operations over $\mathbb{F}_2$. \hfill $\Box$

Finally, we will show that $\mathsf{Maj} : \mathbb{F}_2^n \to \mathbb{F}_2$ cannot be approximated by any polynomial of degree at most $\sqrt{n}$, and thus conclude that $\mathsf{Maj} \not\in \mathsf{AC}^0[\oplus]$.

**Theorem 13.7.** Let $p \in \mathbb{F}_2(x_1, \ldots, x_n)$ be a polynomial of degree $t$. Then

$$\Pr_x[\mathsf{Maj}(x) = p(x)] \leq \frac{1}{2} + O(t/\sqrt{n}).$$

**Proof.** The proof is a clever algebraic proof based on the notion of rank. We will start by proving the following claim.

**Claim 13.8.** For every function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ there are two $\mathbb{F}_2$-polynomials $g, h$ of degrees at most $\frac{n}{2}$ such that

$$f(x) = g(x) + \mathsf{Maj}(x) \cdot h(x).$$

**Proof.** Let $A_0, A_1 \subseteq \mathbb{F}_2^n$ be $A_0 = \mathsf{Maj}^{-1}(0)$ and $A_1 = \mathsf{Maj}^{-1}(1)$. Applying Corollary 13.4 for $A_0$, there is a linear combination $g(x)$ of $x^S$ for $S \subseteq A_0$ such that

$$f(x) = g(x) \quad \forall x \in A_0.$$

Since $A_0 = \mathsf{Maj}^{-1}(0)$, every $S \in A_0$ satisfies $|S| \leq n/2$ and thus $\deg(g) \leq n/2$.

To handle $A_1$, consider $\hat{f}(x) = f(x) - g(x)$. Applying Corollary 13.5 to $A_1$, we obtain a polynomial $h$ of degree at most $n/2$ such that

$$\hat{f}(x) = h(x) \quad \forall x \in A_1.$$

It is easy to check that, $f(x) = g(x) + \mathsf{Maj}(x)h(x)$: For $x \in A_0$,

$$g(x) + \mathsf{Maj}(x)h(x) = g(x) = f(x).$$

And for $x \in A_1$,

$$g(x) + \mathsf{Maj}(x)h(x) = g(x) + h(x) = g(x) + \hat{f}(x) = f(x).$$

\hfill $\Box$
By the above claim, for every $f : \mathbb{F}_2^n \to \mathbb{F}_2$, there are polynomials $g, h$ of degree at most $n/2$ such that

$$f(x) = g(x) + \text{Maj}(x) \cdot h(x) \quad \forall x \in \mathbb{F}_2^n.$$  

Let

$$T = \{ x : p(x) = \text{Maj}(x) \},$$

so that

$$f(x) = g(x) + p(x) \times h(x) \quad \forall x \in T.$$  

Note that $\deg(g + ph) \leq \frac{n}{2} + t$. We have shown that the $|T|$-dimensional space of all functions $f : T \to \mathbb{F}_2$ is in the span of all monomials of degree at most $\frac{n}{2} + t$. Consequently,

$$|T| \leq \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{\frac{n}{2} + t} \leq 2^n + \sum_{k=\frac{n}{2}}^{\frac{n}{2} + t} \binom{n}{k} \leq 2^n \left( \frac{1}{2} + O\left( \frac{t}{\sqrt{n}} \right) \right),$$

using the fact that $\binom{n}{k} \leq \binom{n}{n/2} \sim \frac{2^n}{\sqrt{\pi n}} = O\left( \frac{2^n}{\sqrt{n}} \right)$, for all $k$.  

\[ \square \]

**Theorem 13.9.** Let $C$ be an $\mathcal{AC}^0[\oplus]$ circuit of size $M$ and depth $d$ such that

$$\Pr_x[\text{Maj}(x) \neq C(x)] \leq \frac{1}{3}.$$  

Then $M \geq 2^\Omega(n^{\frac{1}{d+1}})$.

**Proof.** See Exercise 13.1.  

\[ \square \]

**Concluding Remarks**

By this point, we have proved that small depth circuits are limited in their power in that even uncomplicated functions such as MAJ and PARITY require exponentially many gates if we limit the depth to be a fixed constant. The core idea behind both proofs can be interpreted as the broad statement that every low-depth polynomial-size circuit $C(x)$ can be approximated by a low-degree polynomial $p(x)$. In the case of the Razborov-Smolensky, this approximation is formalized by providing a bound on the point-wise error

$$\Pr_x[C(x) \neq p(x)].$$

In the case of the proofs by random restrictions of Chapter 12, this is less apparent, but as we saw in Chapter 17, Linial, Mansour, and Nisan [LMN93] used Corollary 12.17 to show the existence of a low-degree polynomial $p(x)$ such that the so called $L_2$ error term

$$\mathbb{E}_x |C(x) - p(x)|^2$$

is small.

The proof of Razoborov-Smolensky allows one to prove correlation bounds as small as $\frac{1}{2} + \frac{d}{\sqrt{n}}$ with polynomials of degree $d$. This is a nontrivial correlation bound for degrees as high as $\sqrt{n}$. Achieving stronger than $\frac{1}{2} + \frac{1}{\sqrt{n}}$ correlation bounds even for degree $\log n$ remains a major open problem with applications that we will not discuss here.

**Problem 13.10** (Major open problem). Construct an explicit function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ such that for every polynomial of degree $d = \log(n)$, we have

$$\Pr_x[p(x) = f(x)] \leq \frac{1}{2} + \frac{1}{n}.$$  

For an in-depth look into the importance of such correlation bounds for polynomials, we refer the reader to [Vio09].
Exercises


Exercise 13.2. In this exercise we consider Boolean functions as $f : \{-1, 1\}^n \to \{-1, 1\}$. Note that in this setting $\text{Parity}(x) = \prod_{i \in \{1, \ldots, n\}} x_i$.

Consider the vector space of all functions $f : \{-1, 1\}^n \to \mathbb{R}$. Prove that every such function can be written as

$$f = g + \text{Parity} \times h,$$

where $g$ and $h$ are polynomials of degree at most $n/2$.

Exercise 13.3. Let $f : \{0, 1\}^n \to \{0, 1\}$, and let $g$ be a random polynomial of degree $\leq d$ such that for every $x$, we have

$$\Pr_g[f(x) = g(x)] \geq \frac{1}{2} + \varepsilon.$$

Prove that for every $k$, there is a random polynomial $h$ of degree $\leq kd$ such that for every $x$, we have

$$\Pr_h[f(x) \neq h(x)] \leq e^{-k\varepsilon^2/4}.$$

Exercise 13.4. A parity decision tree is generalization of a decision tree in which at every node $v$ we branch according to the value of $\bigoplus_{i \in S_v} x_i$ for a subset $S_v \subseteq \{1, \ldots, n\}$. Prove that there is a (one-sided error) randomized parity decision tree $T_R$ of constant depth that computes the OR function.

That is for all $x$, if $\forall_{i=1}^n x_i = 0$, then $\Pr[R[T_R(x) = 0]] = 1$, and if $\forall_{i=1}^n x_i = 1$, then

$$\Pr[R[T_R(x) = \bigvee_{i=1}^n x_i]] \geq \frac{1}{2}.$$

Exercise 13.5. Let $k > 0$ be a fixed constant, and let the threshold function $t_k : \{0, 1\}^n \to \{0, 1\}$ be defined as $t_k(x) = 1$ if and only if $\sum x_i \geq k$. Prove that there is a (one-sided error) randomized parity decision tree $T_R$ of constant depth that computes $t_k(x)$.

That is for all $x$, if $t_k(x) = 0$, then $\Pr[R[T_R(x) = 0]] = 1$, and if $t_k(x) = 1$, then

$$\Pr[R[T_R(x) = t_k(x)]] \geq \frac{1}{2}.$$

Note that $k = 1$ corresponds to the previous exercise.

Exercise 13.6. Deduce an analogue of Theorem 13.7 for the Parity function from the Switching lemma methods: There are constants $c_1, c_2 > 0$ such that for every $\text{AC}$ circuit $C$ of depth $d$ and size at most $2^{c_1 n^{1/d}}$,

$$\Pr_C[C(x) = \text{Parity}(x)] \leq \frac{1}{2} + 2^{-c_2 n^{1/d}}.$$

Exercise 13.7. In this exercise our goal is to prove that unlike majority, approximate majority can be solved by an $\text{AC}^0$ circuit. More precisely we want to construct a polynomial size circuit $C$ of depth 3 such that if $|x| := \sum x_i \geq \frac{3n}{4}$, then $C(x) = 1$, and if $|x| < \frac{n}{4}$, then $C(x) = 0$. For $\frac{n}{4} \leq |x| < \frac{3n}{4}$, we do not care about the output of $C$.

We construct a sequence of random circuit as follows:

- $C_0$ is a single variable randomly chosen from $x_1, \ldots, x_n$.
- $C_1 = \bigwedge_{i=1}^{10 \log n} D_i$, where $D_i$ are independent copies of the random variable $C_0$.
- $C_2 = \bigvee_{i=1}^{15} D_i$, where $D_i$ are independent copies of the random variable $C_1$.
- $C_3 = \bigwedge_{i=1}^{n^2} D_i$, where $D_i$ are independent copies of the random variable $C_2$.

For any fixed $x$, prove that if $|x| < \frac{n}{4}$, then

$$\Pr_{C_3}[C_3(x) = 1] < 2^{-n^2},$$

and if $|x| \geq \frac{3n}{4}$, then

$$\Pr_{C_3}[C_3(x) = 0] < 2^{-n^2}.$$

Conclude that there is a polynomial size depth 3 circuit $C$ such that for all $|x| < \frac{n}{4}$, $C(x) = 0$, and for all $|x| \geq \frac{3n}{4}$, $C(x) = 1$. 

76
Exercise 13.8 (Difficult). In this exercise, we see another proof where we need to pick random elements with different scales.

Let \((X, d)\) be a metric space with \(n\) points. That is \(|X| = n\), and the distance function \(d : X \times X \to \mathbb{R}^{\geq 0}\) satisfies the following properties:

- \(d(x, y) = 0\) implies \(x = y\).
- \(d(x, y) = d(y, x)\) for all \(x, y\).
- \(d(x, y) + d(y, z) \geq d(x, z)\) for all \(x, y, z\).

Prove that there is an embedding of \(X\) into the Euclidean space that has distortion \(O(\log(n))\). More precisely, there is a map \(\psi : X \to \mathbb{R}^d\) for some \(d\) that does not distort the distances by more than a factor of \(O(\log(n))\): For all \(x, y\):

\[
d(x, y) \leq \|\psi(x) - \psi(y)\| \leq O(\log(n)) \times d(x, y).
\]

Hint: Pick subsets \(S_1, \ldots, S_d \subseteq X\) randomly and according to carefully chosen parameters for an appropriate value of \(d\), and define \(\phi(x) = (d(x, S_1), \ldots, d(x, S_d))\), where \(d(x, S) := \min_y d(x, S)\). Obtain \(\psi\) by scaling \(\phi\) so that \(d(x, y) \leq \|\psi(x) - \psi(y)\|\).
Chapter 14

Razborov’s monotone circuit lower-bound

We continue our study of the computational power of restricted classes of circuits. In Chapters 12 and 13, we considered bounded depth circuits. In this chapter, we turn our attention to monotone circuits but with no restriction on depth. Since we allow arbitrary depth, we can assume that all gates have fan-in 2, as breaking the larger fan-in $\land$ and $\lor$ gates into binary gates does not increase the size of the circuit by much.

Many natural functions, such as $\text{Maj}$, are monotone in the sense that they are increasing in the input bits: Flipping input bits from 0 to 1 cannot decrease the value of the function from 1 to 0. Every such function can be computed by a DNF without any negated terms:

$$f(x) = \bigvee_{y: f(y) = 1} \left( \bigwedge_{j: y_j = 1} x_j \right).$$

This observation suggests removing the $\neg$ gate and considering the circuits with the gates $\land$ and $\lor$ as a natural model of computation for monotone functions. Such circuits are called monotone circuits. It is also easy to see that every monotone circuit computes a monotone function.

Example 14.1. Let $\text{CLIQUE}_k : \{0, 1\}^{\binom{n}{2}} \to \{0, 1\}$ be the function that, given a graph $G$ on $n$ vertices, outputs 1 iff $G$ contains a complete subgraph of size $k$. Here, the graph $G$ is represented by $\binom{n}{2}$ input variables $x_{ij}$, where $x_{ij} = 1$ if and only if there is an edge between the vertices $i$ and $j$.

Obviously $\text{CLIQUE}_k$ is a monotone function, and can be represented by a monotone circuit of size $O(n^k) = O(2^{\log(n) k})$ as

$$\bigvee_{S \subseteq \{1, \ldots, n\} | |S| = k} \bigwedge_{i, j \in S, i \neq j} x_{ij}. \quad (14.1)$$

A counting argument similar to the one we used for general circuits shows that most monotone functions require exponential-size monotone circuits. Nevertheless, proving a super-polynomial lower bound on an explicit monotone function was open for more than 40 years until the invention of the approximation method by Razborov.

Theorem 14.2. [Raz85a, AB87] For sufficiently large $n$, $\text{CLIQUE}_{n^{1/6}}$ requires monotone circuits of size at least $2^{n^{1/12}}$.

Note that Theorem 14.2 looks incredibly impressive, as unlike the $\text{AC}$ lower-bounds of Chapters 12 and 13, it does not restrict the depth of the circuit. Indeed at the time, many people believed that this was very close to proving a general circuit lower-bound and hence establishing $\text{P} \neq \text{NP}$. After all, at first glance, it appears that one only needs to squeeze in the negation gate somewhere in the proof, and one would have an exponential lower bound for $\text{CLIQUE}_k$. However, as later research showed, there are several barriers to this, and indeed we are still nowhere near proving $\text{P} \neq \text{NP}$.

Let $V = \{1, \ldots, n\}$ denote the vertex set. For every $S \subseteq V$, define $C_S(G)$ to be the indicator function that outputs 1 if and only if $S$ is a clique in the input graph $G$. With this notation, we can rewrite (14.1) as

$$\text{CLIQUE}_k(G) = \bigvee_{S \in \binom{V}{k}} C_S(G),$$

where $\binom{V}{k}$ denotes the collection of all subsets of size $k$ of $V$. 

79
Remark 14.4. Erdős and Rado [ER60] conjectured that their $(p-1)^\ell \ell!$ bound could be improved to $\Theta(p^\ell)$. Recently, Alweiss, Lovett, Wu, and Zhang, in a breakthrough paper [ALWZ20], used some ideas from Razborov’s proof [Raz95] of Hastad’s switching lemma to improve the bound in Theorem 14.3 to $\Theta(p \log(\ell))$. However, for our application, the bound in Theorem 14.3 suffices.

Approximating the monotone circuit

The notion of approximation appearing in Razborov’s proof is defined according to how well the circuit handles the “hardest” YES instances and the “hardest” NO instances of CLIQUE$_k$.

- Sparsest YES instances: $G$ consists of a single $k$-clique, and no other edges.
- Densest NO instances: $G$ is a complete $(k-1)$-partite graph, where partitions are chosen of nearly equal sizes.

Note that intuitively these instances are hard: for the case where $G$ is a single $k$-clique, even though most of the graph is empty, we need to detect the clique and accept it. In the case of the $(k-1)$-partite graph, there are many edges present, and yet the circuit should be able to detect that the graph is free of any cliques of size $k$. We define two distributions to capture these hard instances.

- $\mu_{\text{YES}}$: Here $G$ is generated by picking a subset $K \subseteq V$ of size $|K| = k$ uniformly at random and placing a $k$-clique on $K$.
- $\mu_{\text{NO}}$: Pick a colouring of the vertices $c : V \to \{1, \ldots, k-1\}$ uniformly at random, and for all $i, j \in V$, connect $i$ to $j$ if $c(i) \neq c(j)$.

Note that if a circuit $C$ correctly computes CLIQUE$_k$, then

$$\Pr_{G \sim \mu_{\text{YES}}} [C(G) = \text{YES}] = 1$$

and

$$\Pr_{G \sim \mu_{\text{NO}}} [C(G) = \text{NO}] = 1.$$  

Obviously solving CLIQUE$_k$ on either of $\mu_{\text{YES}}$ or $\mu_{\text{NO}}$ is very easy. For $\mu_{\text{YES}}$ it suffices to always output YES, and for $\mu_{\text{NO}}$ one has to always output NO. However Razborov’s proof shows that no sub-exponential size monotone circuit can be simultaneously successful on both distributions.

To this end he shows that if $C$ is a small monotone circuit that computes CLIQUE$_k$ correctly, then there exists a small collection $\mathcal{T}$ of subsets of $V$ such that $\overline{C}(G) = \bigvee_{S \in \mathcal{T}} C_S(G)$ satisfies

$$\Pr_{G \sim \mu_{\text{YES}}} [\overline{C}(G) = \text{YES}] \geq 0.9$$

and

$$\Pr_{G \sim \mu_{\text{NO}}} [\overline{C}(G) = \text{NO}] \geq 0.9.$$  

In other words, $\overline{C}$ is a good approximation of $C$ if the inputs are drawn from either of these two distributions.

We conclude the theorem by showing that one needs a large $\mathcal{T}$ in order for $\overline{C}$ to do well on the distributions $\mu_{\text{YES}}$ and $\mu_{\text{NO}}$. 

![Figure 14.1: A sunflower with 3 petals.](image)
The preliminary lemmas

Throughout the proof we set the following parameters:

\[ k = n^{1/6}, \quad \ell = \sqrt{k}, \quad p = 10\sqrt{k}\log k, \quad t = k^{2\ell} \geq (pt)^\ell \geq (p-1)^\ell t. \] (14.2)

We start by showing that a single \( C_S \) is a very poor approximation of \( \text{CLIQUE}_k \).

**Lemma 14.5.** Let \( \ell \) and \( t \) be as in (14.2), and consider a set \( S \subseteq V \).

(I) If \( |S| \leq \ell \), then

\[
\Pr_{G \sim \mu_{\text{NS}}} [C_S(G) = \text{YES}] \geq \frac{1}{2}.
\]

(II) If \( |S| \geq \ell \), then

\[
\Pr_{G \sim \mu_{\text{YES}}} [C_S(G) = \text{YES}] \leq t^{-2-2\ell}. \]

**Proof.** First consider \( |S| \leq \ell \). Let \( c : V \to \{1, \ldots, k-1\} \) be chosen uniformly at random. For distinct \( i, j \in S \), the probability that \( c(i) = c(j) \) is \( \frac{1}{k^2} \). Applying the union bound over all \( \binom{|S|}{2} \) pairs, the probability that there exist distinct \( i, j \in S \) with \( c(i) = c(j) \) is at most

\[
\binom{|S|}{2} \frac{1}{k-1} = \left( \frac{\ell}{2} \right) \frac{1}{k-1} \leq 0.5.
\]

Hence with probability at least \( \frac{1}{2} \), the set \( S \) is a clique in \( G \).

Next consider \( |S| \geq \ell \), and let \( G \) be drawn from \( \mu_{\text{YES}} \) by picking a random set \( K \subseteq V \) of size \( k \) and planting a clique on \( K \). In this case \( C_S(G) = 1 \) if and only if \( S \subseteq K \). Thus using \( n - \ell + 1 \geq n/2 \), and \( 4k^5 < n \),

\[
\Pr[C_S(G) = 1] = \binom{n-\ell}{k-\ell} \frac{k(k-1)\ldots(k-\ell+1)}{n(n-1)\ldots(n-\ell+1)} \leq \left( \frac{2k}{n} \right)^\ell \leq t^{-2-\ell}.
\]

\[ \square \]

The next lemma shows that replacing a sunflower with its core never damages the circuit on \( \text{YES} \) instances, and it is very unlikely to damage the output of the circuit on a \( G \) drawn from \( \mu_{\text{NS}} \).

**Remark 14.6.** One convenient point to keep in mind throughout the proof is that since the function, the initial circuit, and our approximating circuit are all monotone, for \( \mu_{\text{YES}} \) it is fine if our approximation turns the output of a gate from 0 to 1, and similarly for \( \mu_{\text{NS}} \), it is fine if our approximation turns the output of a gate from 1 to 0. These cannot change the output of the circuit in an erroneous way.

**Lemma 14.7.** Let \( Z_1, \ldots, Z_p \) be a sunflower with core \( R \) and sets of sizes at most \( \ell \). Then \( \bigvee_{i=1}^p C_{Z_i}(G) \leq C_R(G) \) for all \( G \), and furthermore for \( G \sim \mu_{\text{NS}} \),

\[
\Pr_{G \sim \mu_{\text{NS}}} \left[ \bigvee_{i=1}^p C_{Z_i}(G) \neq C_R(G) \right] \leq 2^{-p}.
\]

**Proof.** It is obvious that \( \bigvee_{i=1}^p C_{Z_i} \leq C_R \), and furthermore the inequality is strict only if \( C_R \) detects a clique on \( R \), while all \( Z_1, \ldots, Z_p \) have missing edges. We will show that this happens with small probability if \( G \sim \mu_{\text{NS}} \). We want to bound

\[
\Pr_{G \sim \mu_{\text{NS}}} [C_R(G) = 1 \text{ and } \bigvee_{i=1}^p C_{Z_i}(G) = 0] = \Pr_{\mu_{\text{NS}}} [C_R(G) = 1] \Pr_{\mu_{\text{NS}}} [\bigvee_{i=1}^p C_{Z_i}(G) = 0 | C_R(G) = 1].
\]

We have

\[
\Pr_{\mu_{\text{NS}}} [\bigvee_{i=1}^p C_{Z_i}(G) = 0 | C_R(G) = 1] = \prod_{i=1}^p \Pr_{\mu_{\text{NS}}} [C_{Z_i}(G) = 0 | C_R(G) = 1] \leq \prod_{i=1}^p \Pr_{\mu_{\text{NS}}} [C_{Z_i}(G) = 0] \leq \left( \frac{1}{2} \right)^p,
\]

where the first equality uses the sunflower structure: once we fix the core \( R \), the events \( C_{Z_i}(G) = 0 \) become independent as \( Z_i \setminus R \) are all disjoint and \( C_{Z_i}(G) = 0 \) only depends on the random colours assigned to the vertices in \( Z_i \setminus R \). The second inequality uses the fact that conditioning on \( C_R(G) = 1 \) can only decrease the probability that \( C_{Z_i}(G) = 0 \). Finally in the last inequality we used Lemma 14.5. \( \square \)
Approximation by a small number of cliques

Lemma 14.8. Consider the parameters in (14.2). Let \( C \) be a monotone circuit of size \( M \). There exists a collection \( \mathcal{T} \) of at most \( t \) subsets of \( V \), each of size at most \( \ell \), such that \( \tilde{C}(G) = \bigvee_{S \in \mathcal{T}} C_S(G) \) satisfies

\[
\Pr_{G \sim \mu_{\text{yes}}} [\tilde{C}(G) = \text{No}] \leq \frac{M^{2-\sqrt{\ell}}}{4} \quad \text{and} \quad \Pr_{G \sim \mu_{\text{no}}} [\tilde{C}(G) = \text{Yes}] \leq \frac{M^{2-\sqrt{\ell}}}{4}.
\]

We will prove the lemma by induction, traversing the circuit from the bottom to the top and replacing each gate by OR’s of \( C \). We start by replacing input wires \( x_{ij} \) by indicators for 2-cliques, and working our way up, finally replacing the output gate. Throughout this process we will always use sets \( S \) of size at most \( \ell \), and moreover we will ensure that the replaced gate approximates the original gate with an error probability of at most \( 2^{-\sqrt{\ell}} \) on both of the distributions \( \mu_{\text{yes}} \) and \( \mu_{\text{no}} \). This latter condition will allow us to apply the union bound over all the gates to conclude that the probability that our approximation fails on either of these distributions is bounded by \( M2^{-\sqrt{\ell}} \).

Handling the OR gates: Consider \( f \lor g \) where \( f = \bigvee_{S \in \mathcal{T}_1} C_S \) and \( g = \bigvee_{S \in \mathcal{T}_2} C_S \) with \( |\mathcal{T}_1|, |\mathcal{T}_2| \leq t \). Obviously \( f \lor g = \bigvee_{S \in \mathcal{T}} C_S \) where \( \mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \), which already gives us a set \( \mathcal{T} \). However, it might no longer be true that the size of \( \mathcal{T} \) is bounded by \( t \). Fortunately the sunflower lemma can save the day: If \( |\mathcal{T}| > t \), the collection \( \mathcal{T} \) contains a sunflower \( Z_1, \ldots, Z_p \) with some core \( R \). Using Lemma 14.7 we can replace the sunflower with its core creating an error of at most \( 2^{-p} \) with respect to \( \mu_{\text{no}} \), and no error with respect to \( \mu_{\text{yes}} \). Since \( |\mathcal{T}| \leq |\mathcal{T}_1| + |\mathcal{T}_2| \leq 2t \), repeating this process at most \( t \) times will reduce the size to the desired \( t \), and create an error probability of at most

\[
t2^{-p} \leq 2^{-2t} \leq 2^{-\sqrt{\ell}} - 2,
\]

for either of the distributions.

Handling the AND gates: Note that for every \( G \) we have

\[
C_{S_1}(G) \land C_{S_2}(G) \geq C_{S_1 \cup S_2}(G),
\]

and for \( G \sim \mu_{\text{yes}} \), since \( G \) is a clique, we have

\[
C_{S_1}(G) \land C_{S_2}(G) = C_{S_1 \cup S_2}(G).
\]

Consider \( f \land g \) where \( f = \bigvee_{S \in \mathcal{T}_1} C_S \) and \( g = \bigvee_{S \in \mathcal{T}_2} C_S \). In this case we initially replace \( f \land g \) with \( \bigvee_{S \in \mathcal{T}} C_S(G) \), where

\[
\mathcal{T} = \{S_1 \cup S_2 : S_1 \in \mathcal{T}_1, S_2 \in \mathcal{T}_2\}.
\]

By the above observations this cannot create any error in the case of \( G \sim \mu_{\text{yes}} \) nor in the case of \( G \sim \mu_{\text{no}} \). However this can cause two different issues:

- As in the case of OR gates, this can lead to \( |\mathcal{T}| > t \): We can handle this case using the sunflower lemma. Since \( |\mathcal{T}| \leq |\mathcal{T}_1||\mathcal{T}_2| \leq t^2 \), we need to apply Lemma 14.7 at most \( t^2 - t \) times, replacing the sunflowers with their cores. This will create an error probability of at most

\[
t2^{-p} \leq 2^{-2t}.
\]

- A new issue that is specific to the AND gates is that we could have \( |S_1 \cup S_2| > \ell \). In this case, we simply drop those sets \( S \) of size larger than \( \ell \) from \( \mathcal{T} \). This will not create any errors for the \( \mu_{\text{no}} \), and for the \( \mu_{\text{yes}} \), by Lemma 14.5 (II), this creates an error with probability of at most

\[
|\mathcal{T}|t2^{-2t} \leq 2^{-2t}.
\]

Adding these two cases, we see that the probability of error is at most

\[
2^{-2t} + 2^{-2t} = 2^{-2t+1} \leq 2^{-\sqrt{\ell}}.
\]

Proof Theorem 14.2: Note that \( n^{1/12} = \sqrt{k} \). If there is a monotone circuit of size \( M \leq 2^{n^{1/12}} \) that computes \( \text{CLIQUE}_k \), then by Lemma 14.8 there exists a collection \( \mathcal{T} \) of at most \( t \) subsets of \( V \), each of size at most \( \ell \), such that \( \tilde{C}(G) = \bigvee_{S \in \mathcal{T}} C_S(G) \) satisfies

\[
\Pr_{G \sim \mu_{\text{yes}}} [\tilde{C}(G) = \text{No}] \leq \frac{1}{4} \quad \text{and} \quad \Pr_{G \sim \mu_{\text{no}}} [\tilde{C}(G) = \text{Yes}] \leq \frac{1}{4}.
\]

By Lemma 14.5 (I), the set \( \mathcal{T} \) cannot contain any sets of size at most \( \ell \) and thus must be empty. This is a contradiction.
Concluding remarks

Are monotone circuits the most suited circuits for computing monotone functions? In particular, if a monotone function is in $P$, can it be computed by a polynomial-size monotone circuit? The answer turns out to be negative. Razborov [Raz85b] used the approximation method to prove an $n^{\Omega(\log n)}$ lower bound for the size of a monotone circuit computing the perfect matching problem—which is monotone and is in $P$. While this is a super-polynomial lower bound, interestingly, to this day, no exponential lower bound is known for monotone circuits for the perfect matching problem. However, Tardos [Tar88] looked at a different problem in $P$ and proved an exponential gap between monotone and non-monotone circuits for the problem of computing the (monotone, threshold version of the) Lovász’ Theta function. This problem is in $P$ via semidefinite programming.

Exercises

Exercise 14.1. Optimize the parameters in the proof of Theorem 14.2 to show that as long as $k \leq n^{\frac{4}{5} - \varepsilon}$ for some $\varepsilon > 0$, and $k$ is sufficiently large, then $\text{CLIQUE}_k$ requires monotone circuit size of at least $2^{\Omega(\sqrt{k})}$.

Exercise 14.2. Let $\varepsilon > 0$ be a parameter, and $\phi$ be a DNF with $s$ clauses. Prove that there exists a DNF $\psi$ with width at most $O(\log(s/\varepsilon))$ such that

$$Pr_x[\phi(x) \neq \psi(x)] \leq \varepsilon.$$  

Exercise 14.3. Use the sunflower theorem to prove the following statement: For every monotone DNF $\phi$ with width $w$ and size $m$, every $\varepsilon > 0$, there exists a DNF formula $\psi$ with width $w$ and size at most $(w \log(m/\varepsilon))^{O(w)}$ such that $\psi(x) \geq \phi(x)$ for all $x$, and

$$Pr_x[\phi(x) \neq \psi(x)] \leq \varepsilon.$$  

Exercise 14.4. Obviously $\text{MAJ} \in P$ and thus has a polynomial size circuit. In this exercise we will prove that $\text{MAJ}$ has a polynomial size monotone circuit.

1. First establish that the majority of three copies of a random bit amplifies its bias. More precisely, let $B_1, B_2, B_3$ be independent identically distributed Bernoulli random variables with $Pr[B_i = 1] = p = \frac{1}{2} + \delta$ for $\delta \in [-0.5, 0.5]$. Prove that

$$Pr[\text{MAJ}_3(B_1, B_2, B_3) = 1] = p + \frac{1}{2}(1 - 4\delta^2)\delta.$$  

Note $1 - 4\delta^2 \geq 0$, and thus the bias is amplified slightly.

2. Note that for $x \in \{0, 1\}^n$, if we set $B = x_i$ for a random $i \in \{1, \ldots, n\}$, then $Pr[B = 1] \geq \frac{1}{2}$ if $\text{MAJ}(x) = 1$, and otherwise $Pr[B = 1] \leq \frac{1}{2} - \frac{1}{n}$. Construct a random polynomial-size $O(\log(n))$-depth circuit $C$, using only $\text{MAJ}_3$ gates such that for all $x$,

$$Pr_C[\text{MAJ}(x) = C(x)] > 1 - \frac{1}{2^n}.$$  

3. Conclude that there is a polynomial size monotone circuit $C$ of depth $O(\log(n))$ such that for all $x$, we have $\text{MAJ}(x) = C(x)$.  

83
Chapter 15

Razborov-Rudich: Natural Proof Barrier

We have already seen that relativization is a barrier to answering the question of \( P \neq NP \). It overrules several potential proof approaches. The proofs that are not specific to the computation of Turing Machines relativize and thus cannot answer questions that do not relativize. In particular, they cannot separate complexity classes \( P \) and \( NP \), as there are oracles \( A \) and \( B \) such that \( P^A = NP^A \) and \( P^B \neq NP^B \).

The non-uniform approach of circuit complexity overcomes this barrier. Polynomial-time algorithms can be converted to polynomial size (\( \neg, \lor, \land \))-circuits, something that is not true for general polynomial-time oracle Turing Machines. Furthermore, the simple structure of circuits makes them amenable to the application of tools from other areas of mathematics. The 1980s saw a flurry of new techniques for proving circuit lower bounds on natural, restricted circuit classes. Clever proof techniques such as the random restriction and the switching lemma method of Furst, Saxe, Sipser [FSS84], Ajtai [Ajt83], and Hastad [Has86], the polynomial approximation of Razborov [Raz87] and Smolensky [Smo87], or Razborov’s monotone circuit lower-bound [Raz85a] were all discovered during this period. However, despite the initial excitement by this rapid progress, curiously, they, and all subsequent results, fell short of obtaining any nontrivial lower bounds for general circuits, and particularly proving that \( P \neq NP \).

Is there a fundamental reason for this failure? The same may be asked about any longstanding mathematical problem, for example, the Riemann Hypothesis. One explanation, albeit vague, could be that perhaps the current arsenal of tools and ideas does not suffice. Remarkably, in 1997 Razborov and Rudich [RR97] turned this vague statement into a formal theorem, thus giving us an excuse for our failure so far: they classified a general set of ideas and tools, which are responsible for virtually all restricted circuit lower bounds known, yet must necessarily fail for proving \( P \neq NP \). They dubbed such proofs as natural proofs. Very briefly, a lower bound proof is natural if it applies to a large, easily recognizable set of functions.

In this chapter, we will study their paradigm and prove that there are no natural proofs for \( P \neq NP \) under reasonable complexity-theoretic assumptions.

What are natural proofs?

Almost all known circuit lower bounds follow the following approach:

- One starts by showing that our function of choice, say \( SAT \), satisfies certain nice and simple property \( P \) (e.g., have large total influence, cannot be approximated by a low-degree polynomial, does not simplify under random restrictions, etc.)

- Then one proceeds by showing that functions computable by circuits in the class \( C \) do not satisfy \( P \), and thus cannot include our function.

Here a property \( P \) just means a subset of all Boolean functions \( f : \{0,1\}^n \to \{0,1\} \), and when we say \( f \) satisfies \( P \), we mean \( f \in P \). Obviously, for the above approach to be sensible, \( P \) must be simpler and easier to understand than the property of being computable by a circuit in \( C \). Razborov and Rudich require two characteristics from the property:

1. Constructivity: Given the truth table of any function \( f : \{0,1\}^n \to \{0,1\} \), whether \( f \) satisfies \( P \) or not can be decided in polynomial time. Note that since the input is a truth table, it is of size \( N = O(2^n) \), and thus polynomial-time means \( N^{O(1)} = 2^{O(n)} \).
2. **Largeness:** A non-negligible fraction of functions belong to $\mathcal{P}$. Formally, there exists a constant $c > 0$ such that for every sufficiently large $n$, a randomly selected function $f : \{0,1\}^n \rightarrow \{0,1\}$ satisfies

$$\Pr[f \in \mathcal{P}] \geq c.$$ 

Every property that satisfies the above conditions is called a *natural property*. A proof that shows $h \notin C$ by establishing that $h \in \mathcal{P}$ and that no function in $C$ satisfies $\mathcal{P}$ is called a *natural proof*. See Figure ??.

**Example 15.1.** If $\mathcal{P} := \{ f : I_f \geq n/4 \}$, then $\mathcal{P}$ is constructive because we can compute $I_f$ in $2^{O(n)}$, and $\mathcal{P}$ is large because $E_f[I_f] = n/2$, and thus $\Pr_f[I_f \geq n/4] \geq \frac{1}{2}$.

As we have seen earlier in Theorem 12.21, every function that can be computed with an $\text{AC}^0$ circuit of size $n^{O(1)}$ and depth $O(1)$ satisfies $I_f = \log^{O(1)}(n)$, and thus for sufficiently large $n$, no such $f$ belongs to $\mathcal{P}$. On the other hand, $\text{Parity}_n \in \mathcal{P}$, and thus does not belong to the set of functions that have $\text{AC}^0$ circuits of size $n^{O(1)}$ and depth $O(1)$.

The above proof is a *natural proof* for the fact that $\text{Parity} \notin \text{AC}^0$. □

### The Razborov-Rudich Theorem

Their article [RR97] states:

Consider a commonly envisioned proof strategy for proving $\mathsf{P} \neq \mathsf{NP}$:

- Formulate some mathematical notion of “discrepancy” or “scatter” or “variation” of the values of a Boolean function, or of an associated polytope or other structure.
- Show by an inductive argument that polynomial-sized circuits can only compute functions of “low” discrepancy. […]
- Then show that $\text{SAT}$, or some other function in $\mathsf{NP}$, has “high” discrepancy.

Such a proof is likely to be natural. *If cryptography is correct, such proofs cannot work for $\mathsf{P} \neq \mathsf{NP}$.*

Razborov and Rudich first show that their natural proof framework essentially encapsulates all known circuit lower bounds. Then they show that natural proofs of general circuit lower-bounds are unlikely, in the following sense. Any natural proof of a general circuit lower-bound implies, as a side-effect, a subexponential algorithm for inverting every candidate *one-way function*. Specifically, a natural-proof lower bound would imply subexponential algorithms for such functions as integer factoring and discrete logarithm, generally believed to be exponentially difficult—to the extent that the security of electronic commerce worldwide relies on such assumptions.

Before giving the formal statement and its proof, we need first to discuss the *one-way functions*, which originate from cryptography. Intuitively, a one-way function is a function that is easy to compute but computationally hard to invert, i.e., hard to find the preimage given the image. Moreover, it is even computationally hard to guess the inverse correctly with reasonable probability.

**Definition 15.2.** A function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ is a one-way function if

- Given $x$, $f(x)$ can be computed in $\text{poly}(|x|)$ time.
- For all probabilistic polynomial time algorithms $M_R$, we have

$$\Pr_{x \in \{0,1\}^n}[M_R(f(x)) \notin f^{-1}(f(x))] \leq \frac{1}{n^{\omega(1)}}.$$ 

Cryptography is based on the assumption that functions such as integer factorization are one-way functions. Recall that $\text{size}(n^k)$ denotes the set of functions that can be computed by a circuit of size $n^k$.

**Theorem 15.3** (Razborov-Rudich [RR97]). *Assuming that one-way functions exist, no natural proof can show* $\mathsf{NP} \not\subseteq \text{size}(n^k)$, for all $k \in \mathbb{N}$.  

86
Preparing for the proof: Pseudo-random Generators

An important element in the proof is the notion of a pseudorandom generator. Consider a probabilistic algorithm that uses a random string \( R \) of at most \( m \) bits as its source of randomness. We can derandomize the algorithm by trying all the \( 2^m \) possible strings \( R \in \{0,1\}^m \) instead of picking one at random, and see if the majority of them lead to accept or to reject. We are paying in running time as this multiplies the running time by \( 2^m \). This is not an issue if \( m = O(\log(n)) \) because, in that case, the increase in the running time is only a polynomial factor. However, when \( m \) is large, say linear in the input size \( n \), the increase in the running time can turn a polynomial-time algorithm into an exponential algorithm.

Now suppose that for some \( \ell \) that is much smaller than \( m \), we are given a function \( G : \{0,1\}^\ell \rightarrow \{0,1\}^m \) such that the algorithm \( M_R \) cannot distinguish a truly random string \( R \in \{0,1\}^m \) from a random string \( R' = G(y) \) where \( y \) is randomly selected from \( \{0,1\}^\ell \). Indistinguishability here means that

\[
\Pr_{R \in \{0,1\}^m} [M_R(x) = \text{Accept}] \approx \Pr_{y \in \{0,1\}^\ell} [M_{G(y)}(x) = \text{Accept}].
\]

On the left hand side we are using a truly random \( R \), while on the right hand side we are using an artificially constructed random string \( G(y) \) that is constructed from a smaller seed of random bits. Such a \( G \) essentially decreases the required number of random bits from \( m \) to \( \ell \). Hence, for example, we can proceed with the above derandomization process and turn our probabilistic algorithm into a deterministic algorithm, paying now only a factor of \( 2^\ell \) rather than \( 2^m \) in the running time.

To summarize, a pseudorandom generator \( G \) for a complexity class \( C \) takes a truly random seed \( y \) of length \( \ell \), and from it constructs a much longer string \( G(y) \) of length \( m \) that is indistinguishable from a truly random string of length \( m \) by functions in \( C \).

**Definition 15.4** (Pseudo-Random Generator). A function \( G : \{0,1\}^\ell \rightarrow \{0,1\}^m \) is an \( \varepsilon \)-PRG against a class \( C \) if for every \( T \in C \),

\[
\left| \Pr_{y \in \{0,1\}^\ell} [T(G(y)) = 1] - \Pr_{R \in \{0,1\}^m} [T(R) = 1] \right| \leq \varepsilon.
\]

We say \( G \) is efficient if \( G(y) \) runs in polynomial time in the *output length* \( m \).

Blum and Micali [BM82] gave an elegant construction that shows how to construct a pseudorandom generator based on the assumption that the discrete logarithm problem is a one-way function. Much later Hastad, Impagliazzo, Levin, and Luby [HsILL99] showed that the existence of any one-way function suffices to guarantee the existence of wonderfully strong pseudo-random generators that double the length of the seed and can fool polynomial size circuits.

**Theorem 15.5** ([HsILL99]). If one-way functions exist, there is a sequence of pseudo-random generators \( G_k : \{0,1\}^k \rightarrow \{0,1\}^{2k} \), computable in polynomial time in \( k \) such that for all \( c > 0 \), all circuits \( T \) of polynomial size \( S = k^c \) satisfy

\[
\left| \Pr_{y \in \{0,1\}^k} [T(G(y)) = 1] - \Pr_{R \in \{0,1\}^{2k}} [T(R) = 1] \right| \leq \frac{1}{S},
\]

provided that \( k \) is sufficiently large.

We will not directly apply this theorem, but rather use a consequence of this theorem due to Goldreich, Goldwasser, and Micali [GGM85]. They showed that one can use the above pseudo-random generators to construct functions that “look random” to sub-exponential algorithms.

Consider a probabilistic algorithm \( A \) with running time \( 2^{O(m^c)} \) that has oracle access to a function \( f \). That is, during its computation, it can query the values of \( f(x) \) for different points \( x \in \{0,1\}^m \). We want to construct a small set of functions \( f_s \) such that \( A \) cannot distinguish between a completely random function \( f \) and a function that is selected randomly from our much smaller set of functions \( f_s \). Goldreich, Goldwasser, and Micali [GGM85] show that this is possible.

**Theorem 15.6** (Goldreich, Goldwasser, Micali [GGM85]). If one-way functions exist, there exists a collection of functions \( f_s : \{0,1\}^m \rightarrow \{0,1\} \) for \( s \in \{0,1\}^m \) such that

- Given \( s, x \in \{0,1\}^m \), we can compute \( f_s(x) \) in \( \text{poly}(m) \) time.

- For any probabilistic \( 2^{O(m^c)} \) time algorithm \( A \), where \( \varepsilon > 0 \) is an absolute constant, we have

\[
\left| \Pr_{x \in \{0,1\}^m} [A^{f_s} = \text{Accept}] - \Pr_{f : \{0,1\}^m \rightarrow \{0,1\}} [A^f = \text{Accept}] \right| = o(1),
\]

where the probabilities are also over the randomness of \( A \).

Note that in Theorem 15.6, on the left hand side, we are randomly choosing a function from only \( 2^m \) functions \( f_s \), while on the right hand side we are choosing the function out of all the \( 2^{2m} \) functions.
The proof of Razborov-Rudich Theorem

Finally, we are ready to prove Theorem 15.3. Consider a natural property $\mathcal{P}$ that is useful for proving that some function, such as SAT, does not belong to the circuit classes $\text{size}(n^k)$ for all $k \in \mathbb{N}$. We have

- **Constructivity**: It can be decided in time $2^{O(n)}$ whether $f : \{0,1\}^n \rightarrow \{0,1\}$ has the property.
- **Largeness**: $\Pr[f \in \mathcal{P}] \geq \delta$.
- **Usefulness**: For every $k$, for sufficiently large $n$,  
  $\mathcal{P} \cap \text{size}(n^k) = \emptyset$.

Let $n = m^{\varepsilon/2} < m$, where $\varepsilon$ is as in Theorem 15.6, and consider the functions $f_s : \{0,1\}^m \rightarrow \{0,1\}$ provided by the theorem. For $s \in \{0,1\}^m$, define

$$g_s : \{0,1\}^n \rightarrow \{0,1\} \quad \text{as} \quad g_s(x) = f_s(x0^{m-n}).$$

Note that given $s \in \{0,1\}^m$ and $x \in \{0,1\}^n$, by the first part of Theorem 15.6 the value of $g_s(x) = f_s(x0^{m-n})$ can be computed in time $m^{O(1)} = n^{O(1)}$. Consequently, there is a $\text{size}(n^{O(1)})$ circuit that given $x, s$ computes $g_s(x)$. Hence by the usefulness property

$$g_s \notin \mathcal{P} \quad \text{for every } s \in \{0,1\}^m,$$

while by the largeness property

$$\Pr[g \in \mathcal{P}] \geq \delta \quad \text{where } g : \{0,1\}^n \rightarrow \{0,1\} \text{ is chosen randomly.} \quad (15.1)$$

This suggests a probabilistic algorithm for distinguishing a randomly selected $f_s : \{0,1\}^m \rightarrow \{0,1\}$ from a randomly selected function from all the functions $\{0,1\}^m \rightarrow \{0,1\}$, and thus contradicting the assertion of Theorem 15.3.

Namely, given a function $f : \{0,1\}^m \rightarrow \{0,1\}$ from an unknown distribution which is either uniform over all $f_s$ or uniform over all $n$-bit functions, we look at the function $g : \{0,1\}^n \rightarrow \{0,1\}$ defined as $g(x) = f(x0^{m-n})$. By the constructivity property of $\mathcal{P}$, we can decide in time $2^{O(n)} = 2^{O(m^{\varepsilon/2})}$ whether $g \in \mathcal{P}$. We accept if this is the case. Let us call this algorithm $A$. If $f$ was selected from the set $\{f_s : s \in \{0,1\}^m\}$, then by (15.1)

$$\Pr[A^f = \text{Accept}] = 0.$$ 

On the other hand if $f$ was selected from all of the functions $\{0,1\}^m \rightarrow \{0,1\}$, then note that $g : \{0,1\}^n \rightarrow \{0,1\}$ is also a completely random function, and thus by (15.2), we have

$$\Pr[A^f = \text{Accept}] \geq \delta.$$ 

In other words, we have algorithm $A$ that runs in time $2^{O(m^{\varepsilon/2})}$ such that

$$\left| \Pr_{s \in \{0,1\}^m} [A^f = \text{Accept}] - \Pr_{f : \{0,1\}^m \rightarrow \{0,1\}} [A^f = \text{Accept}] \right| \geq \delta,$$

contradicting Theorem 15.6.

**Concluding Remarks**

The natural proof frameworks was extended in several directions. See [Wig93] for a survey. Recently, [CIKK16] showed that a natural proof of a circuit lower bound against any sufficiently powerful circuit class yields a learning algorithm for the same circuit class.

Even today, decades later, after the discovery of the relativization and natural proof barriers, most complexity results either relativize or fall into the framework of natural proofs. One way to overcome these barriers follows from the so-called arithmetization technique on interactive proofs. This technique is used to prove some lower bounds, for example, Santhanam [San09] and Vindochandran [Vin04], on interactive proof classes. To curb the power of this technique, Aaronson and Wigderson [AW08] defined the algebraization barrier, a generalization of relativization that incorporates proofs that use arithmetization. They show that proofs that algebrize are still too weak to resolve the $P \neq NP$ question and other complexity challenges.

The only circuit lower bound technique that avoids all known barriers originates with Williams [Wil14]. It uses a brilliant combination of diagonalization and simulation on the one hand and circuit-complexity techniques. Unfortunately, this line of work has so far delivered relatively few lower bounds for rather weak circuit classes.
Exercises

Exercise 15.1. Prove that the Razborov-Smolensky proof for $\text{MAJ} \notin \text{AC}^0[\oplus]$ is a natural proof.
Chapter 16

Fourier analysis and Polynomial Representations

Consider a Boolean function
\[ f : \{\text{False, True}\}^n \to \{\text{False, True}\}. \]

Identifying \( \{\text{False, True}\} \) with each of \( \{0, 1\} \), the field \( \mathbb{F}_2 \), or \( \{-1, 1\} \) leads to a different representation of \( f \). Each of these representation places us in the domain of a different field of mathematics, and allows us to benefit from the rich collection of all the tools, theorems, methods, and the machinery that have been developed in that area.

(I) \( f : \mathbb{F}_2^n \to \mathbb{F}_2 \). Here we can consider \( f \) as an element of the vector space of functions \( \mathbb{F}_2^n \to \mathbb{F}_2 \) over the field \( \mathbb{F}_2 \).

This allows us to consider the polynomial representation of \( f \) as
\[ f(x) = \sum_{S \subseteq \{1, \ldots, n\}} a_S \prod_{i \in S} x_i \]
with coefficients \( a_S \in \mathbb{F}_2 \), and appeal to linear algebraic tools, and the finite field theory.

(II) \( f : \{0, 1\}^n \to \mathbb{R} \). This will allow us to consider the polynomial representation of \( f \) as
\[ f(x) = \sum_{S \subseteq \{1, \ldots, n\}} a_S \prod_{i \in S} x_i \]
with the coefficients \( a_S \in \mathbb{R} \).

(III) \( f : \{-1, 1\}^n \to \mathbb{R} \). This will allow us to consider the polynomial representation of \( f \) as
\[ f(x) = \sum_{S \subseteq \{1, \ldots, n\}} b_S \prod_{i \in S} x_i \]
with the coefficients \( b_S \in \mathbb{R} \). The coefficients \( b_S \) are different from the coefficients \( a_s \) in (II). Using the \( \{-1, 1\} \)-valued variables makes the monomials satisfy certain orthogonality properties, allowing us to use geometric tools.

(IV) \( f : \mathbb{Z}_2^n \to \mathbb{R} \), where \( \mathbb{Z}_2 \) is the Abelian group \( \{0, 1\} \) with addition defined modulus 2. This is closely related to (III) and allows us to consider the Fourier transform of \( f \) over the Abelian group \( \mathbb{Z}_2^n \). Indeed as we shall see the coefficient \( b_S \) are exactly the Fourier coefficients of \( f \) as a function on the Abelian group \( \mathbb{Z}_2^n \). This not only allows us to use the large repository of tools and ideas from Fourier analysis, but it also suggests certain generalizations of the proofs. Some results about Boolean functions can be generalized to other Abelian groups, or even non-Abelian groups such as the group of all the permutations, called the symmetric group. In the setting of non-Abelian groups, representation theory replaces Fourier analysis.

In the previous chapters, in particular in the proof of the Razborov-Smolensky’s theorem, we have already seen how (I) and (II) can be useful. In this chapter we will focus on (III) and (IV).
Polynomial representations of \( f : \{-1, 1\}^n \to \mathbb{R} \)

As it is mentioned in the introduction to this chapter, it is sometimes useful to represent Boolean functions as \( f : \{-1, 1\}^n \to \mathbb{R} \) rather than \( f : \{0, 1\}^n \to \mathbb{R} \). Consider the vector space of all functions \( f : \{-1, 1\}^n \to \mathbb{C} \) endowed with the inner product

\[
\langle f, g \rangle := \mathbb{E} \left[ f(x) \overline{g(x)} \right].
\]

For every \( S \subseteq \{1, \ldots, n\} \), consider the monomial \( x^S = \prod_{i \in S} x_i \), which is a function from \( \{-1, 1\}^n \) to \( \{-1, 1\} \). We define \( x^\emptyset \equiv 1 \) as the constant monomial. Note that

\[
\mathbb{E}[x^S] = \begin{cases} 1 & S = \emptyset \\ 0 & S \neq \emptyset \end{cases}.
\]

This implies the orthonormality of these monomials with respect to the inner product defined above:

\[
\langle x^S, x^T \rangle = \mathbb{E} [x^S x^T] = \mathbb{E} [x^{S \triangle T}] = \begin{cases} 1 & S = T \\ 0 & S \neq T \end{cases},
\]

where \( S \triangle T = (S \setminus T) \cup (T \setminus S) \) denotes the symmetric difference between \( S \) and \( T \). Since there are \( 2^n \) such monomials, which matches the dimension, we conclude that the monomials \( x^S \) form an orthonormal basis for the vector space of all functions \( f : \{-1, 1\}^n \to \mathbb{R} \). Hence in this case, not only we have the linear independence of the monomials as in the case of the functions \( f : \{0, 1\}^n \to \mathbb{R} \), but we have the additional geometric structure coming from the orthonormality. It follows from the orthonormality that every function \( f : \{-1, 1\}^n \to \mathbb{R} \) has a unique polynomial representation

\[
f(x) = \sum_{S \subseteq \{1, \ldots, n\}} \hat{f}(S) x^S,
\]

where the coefficients \( \hat{f}(S) \in \mathbb{R} \) are given by

\[
\hat{f}(S) = \langle f, x^S \rangle = \mathbb{E}[f(x)x^S].
\]

Note that this representation, and the coefficients are different from that of the polynomial representation when \( f \) is considered as a function \( \{0, 1\}^n \to \mathbb{R} \). Indeed the change of variable \( y_i = \frac{x_i + 1}{2} \) converts a \(-1, 1\)-valued variable \( x_i \) to a \(\{0, 1\}\)-valued variable \( y_i \), and thus

\[
\sum_{S \subseteq \{1, \ldots, n\}} \hat{f}(S) x^S = \sum_{S \subseteq \{1, \ldots, n\}} \hat{f}(S) \prod_{i \in S} (2y_i - 1) = \sum_{S \subseteq \{1, \ldots, n\}} \left( \sum_{T \supseteq S} 2^{|S|} (-1)^{|T\setminus S|} \hat{f}(T) \right) y^S,
\]

and conversely

\[
\sum_{S \subseteq \{1, \ldots, n\}} a_S y^S = \sum_{S \subseteq \{1, \ldots, n\}} a_S \prod_{i \in S} \frac{x_i + 1}{2} = \sum_{S \subseteq \{1, \ldots, n\}} \left( \sum_{T \supseteq S} 2^{-|T|} a_T \right) x^S,
\]

gives the relations between the coefficients in these two different representations. One important fact about these representations is that, whether we choose \( f : \{-1, 1\}^n \to \mathbb{R} \) or \( f : \{0, 1\}^n \to \mathbb{R} \), the degree of the polynomial remains the same: The largest \( S \) such that \( a_S \neq 0 \) also satisfies \( \hat{f}(S) \neq 0 \) and vice versa. We call this quantity simply the degree of the Boolean function \( f \) and denote it by \( \deg(f) \).

Of course depending on the application, sometimes it might be convenient to work with the polynomial representation over \( \{0, 1\}^n \), but when the choice \( \{-1, 1\}^n \) is appropriate, the orthogonality of the monomials can be extremely useful. For example, we have the Parseval identity:

\[
\langle f, g \rangle := \sum_S \hat{f}(S) \hat{g}(S).
\]

Note that

\[
\mathbb{E}[f] = \hat{f}(\emptyset),
\]

and by the Parseval identity

\[
\text{Var}[f] = \mathbb{E}[f^2] - \mathbb{E}[f]^2 = \langle f, f \rangle - \hat{f}(\emptyset)^2 = \sum_{S \neq \emptyset} \hat{f}(S)^2,
\]

92
showing that both the expected value and the variance can be easily expressed in terms of the coefficients. Let us add to these the influences: for \( f : \{-1,1\}^n \rightarrow \{0,1\} \), we have

\[
I_i(f) = \Pr[f(x) \neq f(x \oplus e_i)] = \mathbb{E} \left| f(x) - f(x \oplus e_i) \right|^2 = \mathbb{E} \left( \sum_{S \subseteq S} 2 \hat{f}(S)x^S \right)^2 = \sum_{S \subseteq S} 4 \hat{f}(S)^2,
\]

where again we used the Parseval’s identity (Here by an abuse of notation we used \( x \oplus e_i \) to denote the vector that is obtained from \( x \) by changing the sign of the \( i \)-th coordinate). This also gives us a formula for the total influences of \( f \)

\[
I(f) = \sum_S 4 |S| |\hat{f}(S)|^2.
\]

Finally as we will see in the next section, this polynomial representation of a Boolean function coincides with the Fourier expansion of \( f \) when it is considered as a function from the Abelian group \( \mathbb{Z}_2^n \) to \( \mathbb{R} \).
Fourier analysis of Finite Abelian Groups

In this section we develop the basic Fourier analysis of finite Abelian groups. Recall that the cyclic group $\mathbb{Z}_N$ is the Abelian group with elements $\{0, 1, \ldots, N-1\}$, where the group product is defined as $a + b := a + b \pmod{N}$. Finite Abelian groups can be characterized as the products of cyclic groups:

**Theorem 16.1.** Every finite Abelian group $G$ is isomorphic to the group $\mathbb{Z}_{N_1} \times \ldots \times \mathbb{Z}_{N_k}$ for some positive integers $N_1, \ldots, N_k$.

We will be mostly interested in the group $\mathbb{Z}_2^N$ as it can be identified with the set $\{\text{FALSE, TRUE}\}^n$. Hence Boolean functions $f : \{\text{FALSE, TRUE}\}^n \to \{0, 1\}$ can be identified with functions $f : \mathbb{Z}_2^n \to \{0, 1\}$, and this shall allow us to use the Fourier analysis of $\mathbb{Z}_2^n$ to study Boolean functions. The reason we use the notation $\mathbb{Z}_2$ instead of $\mathbb{F}_2$ is that here we only care about the group structure of the set (with $+$).

**Basic Fourier Theory**

Let $G$ be a finite Abelian group. A function $\chi : G \to \mathbb{C} \setminus \{0\}$ mapping the group to the non-zero complex numbers is called a character of $G$ if it is a group homomorphism. That is, $\chi(a + b) = \chi(a)\chi(b)$ for all $a, b \in G$, and $\chi(0) = 1$, where $0$ is the identity of $G$. Note that the constant function $1$ is always a character and it is called the principal character of $G$.

Let $\chi$ be a character of $G$, and consider an element $a \in G$. Since $G$ is a finite group, $a$ is of some finite order $n$ (that is $na = 0$ where $na$ refers to adding $a$ to itself $n$ times). Hence $1 = \chi(0) = \chi(na) = \chi(a)^n$ which shows that $\chi(a)$ is an $n$-th root of unity. In particular, every character $\chi$ of $G$ satisfies

$$\chi : G \to \mathbb{T},$$  

(16.1)

where $\mathbb{T}$ is the unit complex circle.

**Theorem 16.2.** If $G$ is a finite Abelian group, then the characters of $G$ together with the usual point-wise product of complex valued functions form a group $\hat{G}$.

**Proof.** The principal character $1$ is the identity of $\hat{G}$. Note that if $\chi$ and $\xi$ are characters of $G$, then $\chi\xi$ is also a character. Indeed $\chi(ab)\xi(ab) = \chi(a)\xi(a)\chi(b)\xi(b)$, and $\chi(0)\xi(0) = 1 \times 1 = 1$. To check the existence of the inverse elements, note that if $\chi$ is a character, then $\chi^{-1} := \frac{1}{\chi} = \bar{\chi}$ is also a character. \qed

The group $\hat{G}$ is called the Pontryagin dual of $G$. Fourier analysis is based on expressing functions $f : G \to \mathbb{C}$ as linear combinations of characters. It will be convenient to treat the set of these functions as a Hilbert space: Let $L_2(G)$ denote the set of functions $f : G \to \mathbb{C}$, where here $G$ is endowed with the uniform probability measure. Recall (see Section 2) that $L_2(G)$ is a Hilbert space with the inner product

$$\langle f, g \rangle = \mathbb{E}_{x \in G} f(x)\overline{g(x)} = \frac{1}{|G|} \sum_{x \in G} f(x)\overline{g(x)}.$$  

In the sequel, we will often consider $G$ as a probability space, and $\mathbb{E}_{x \in G}$ shall always mean that $x$ is a random variable that takes values in $G$ uniformly at random. To simplify the notation we usually abbreviate $\mathbb{E}_{x \in G}$ to simply $\mathbb{E}$. For a function $f : G \to \mathbb{C}$, the notation $\mathbb{E}[f]$ means $\mathbb{E}_{x \in G}[f(x)]$ (which is equal to $\frac{1}{|G|} \sum_{x \in G} f(x)$).

**Example 16.3.** Consider the group $\mathbb{Z}_2^n$. For every $a = (a_1, \ldots, a_n) \in \mathbb{Z}_2^n$, one can construct a corresponding character $\chi_a$ that maps $x$ to $\prod_{i:a_i=1}(-1)^{x_i} = (-1)^{\sum_{i:a_i=1} x_i}$. The principal character is $\chi_0 \equiv 1$ where the $0$ in the index refers to $(0, \ldots, 0)$, the identity element of the group. It is easy to verify that these are indeed all the characters of $\mathbb{Z}_2^n$. Note that in this case the characters are actually real-valued (they only take values 1 and $-1$), but as we shall see below for all other Abelian groups there are characters that take non-real values.

Since the coordinates of $a \in \mathbb{Z}_2^n$ are 0 or 1, we will sometimes identify $a$ with the set $S = \{i : a_i = 1\} \subseteq \{1, \ldots, n\}$, and denote the characters as $\chi_S$ for $S \subseteq \{1, \ldots, n\}$. This notation is sometimes more intuitive as

$$\chi_S(x) = (-1)^{\sum_{i \in S} x_i},$$

and this is consistent with the polynomial representation considered earlier.

Our next goal will be to prove that the characters form an orthonormal basis for $L_2(G)$. First let us prove a simple lemma.
Lemma 16.4. Let $G$ be a finite Abelian group, and $\chi$ be a non-principal character of $G$. Then $\sum_{x \in G} \chi(x) = 0$.

Proof. Suppose to the contrary that $\sum_{x \in G} \chi(x) \neq 0$. Consider an arbitrary $y \in G$, and note

$$\chi(y) \sum_{x \in G} \chi(x) = \sum_{x \in G} \chi(y + x) = \sum_{x \in G} \chi(x)$$

which shows that $\chi(y) = 1$. Since $y$ was arbitrary, we conclude that $\chi$ must be the principal character which is a contradiction.

Now we can prove the orthogonality of the characters.

Lemma 16.5. The characters of a finite Abelian group $G$ are orthonormal functions in $L_2(G)$.

Proof. It follows from (16.1) that every $\chi \in \hat{G}$ satisfies

$$\|\chi\|^2 = E[|\chi(x)|^2] = E[1] = 1.$$

So characters are unit vectors in $L_2(G)$. It remains to verify the orthogonality. Let $\chi \neq \xi$ be two different characters. Then $\chi \xi = \chi \xi^{-1}$ is a non-principal character of $G$ (why?). Hence by Lemma 16.4, we have

$$\langle \chi, \xi \rangle = E[\chi(x)\xi(x)] = E[\chi\xi(x)] = 0.$$

So far we have discussed the Pontryagin dual of $G$ in an abstract manner. Since finite Abelian groups have simple structures (Theorem 16.1), it is quite easy to describe the characters of $G$. We start with the basic case of $G = \mathbb{Z}_N$. For every $a \in \mathbb{Z}_N$, define $\chi_a \in L_2(G)$ as

$$\chi_a : x \mapsto e^{2\pi i ax}.$$

Let us verify that $\chi_a$ is actually a character. Indeed $\chi_a(0) = e^{2\pi i 0} = e^0 = 1$, and since $e^{2\pi i} = 1$, we have

$$\chi_a(x)\chi_a(y) = e^{2\pi i ax}e^{2\pi i ay} = e^{2\pi i a(x+y) \mod N} = \chi_a(x+y).$$

Note that $L_2(G)$ is $|G|$-dimensional, and hence by Lemma 16.5, $G$ has at most $|G|$ characters. It follows that $\{\chi_a : a \in G\}$ are all the characters of $G$. The principal character is $\chi_0 \equiv 1$. Also $\chi_a\chi_b = \chi_{a+b}$ which shows that the dual group $\hat{G}$ is isomorphic to $G$. As we shall see below this is in general true for all finite Abelian groups.

Now let us consider the general case of $G = \mathbb{Z}_{N_1} \times \ldots \times \mathbb{Z}_{N_k}$ for some positive integers $N_1, \ldots, N_k$. For every $a = (a_1, \ldots, a_k) \in G$, define $\chi_a \in L_2(G)$ as the product of the characters $\chi_{a_1}, \ldots, \chi_{a_k}$ of the groups $\mathbb{Z}_{N_1}, \ldots, \mathbb{Z}_{N_k}$ applied to the coordinates of $x \in G$ respectively. More precisely

$$\chi_a : x \mapsto e^{2\pi i a_j x_j}.$$

As in the case of $\mathbb{Z}_N$, it is straightforward to verify that $\chi_a$ is a character by showing that $\chi_a(0) = 1$, and $\chi_a(x+y) = \chi_a(x)\chi_a(y)$. Again Lemma 16.5 shows that $\{\chi_a : a \in G\}$ are all the characters of $G$. We also have the identify $\chi_a\chi_b = \chi_{a+b}$ which implies the following theorem.

Theorem 16.6. If $G$ is a finite Abelian group, then the characters of $G$ form an orthonormal basis for $L_2(G)$. Furthermore we have $\hat{G} \cong G$.

Theorem 16.6 shows that $G$ is isomorphic to its dual $\hat{G}$, and so it shall be convenient to identify the two groups in the sequel, and denote the characters by $\chi_a$ where $a \in G$. Since the characters form an orthonormal basis for $L_2(G)$, every function $f : G \to \mathbb{C}$ can be expressed in a unique way as a linear combination of the characters $f = \sum_{a \in G} \hat{f}(a)\chi_a$.

The corresponding coefficients $\hat{f}(a) \in G$ are referred to as the Fourier coefficients. This leads to the definition of the Fourier transform.

Definition 16.7. The Fourier transform of a function $f : G \to \mathbb{C}$ is the unique function $\hat{f} : \hat{G} \to \mathbb{C}$ defined as

$$\hat{f}(\chi) = \langle f, \chi \rangle = E f(x)\overline{\chi(x)}.$$

We will often use the notation $\hat{f}(a)$ to denote $\hat{f}(\chi_a)$.
Let us state a simple example of the Fourier transform of a function on \( \mathbb{Z}_2^n \).

**Example 16.8.** Let \( f : \mathbb{Z}_2^n \to \mathbb{C} \) be the parity function \( f : x \mapsto \sum_{i=1}^{n} x_i \pmod{2} \). Then
\[
\hat{f}(0) = \mathbb{E} f(x) \chi_0 = \mathbb{E} f(x) = \frac{1}{2}.
\]
We also have
\[
\hat{f}(1, \ldots, 1) = \mathbb{E} f(x)(-1)^{\sum_{j=1}^{n} x_j} = -\frac{1}{2},
\]
as \( f(x) = 1 \) if and only if \( \sum_{j=1}^{n} x_j = 1 \pmod{2} \). Next consider \( a \in \mathbb{Z}_2^n \) with \( a \neq (1, \ldots, 1) \) and \( a \neq 0 \). Let \( j_0, j_1 \) be such that \( a_{j_0} = 0 \) and \( a_{j_1} = 1 \). We have (why?)
\[
\hat{f}(a) = \mathbb{E} f(x) \chi_a(x) = \frac{1}{2} \mathbb{E} [f(x) \chi_a(x) + f(x + e_{j_0} + e_{j_1}) \chi_a(x + e_{j_0} + e_{j_1})],
\]
where \( e_j \) denotes the vector in \( \mathbb{Z}_2^n \) which has 1 at its \( j \)th coordinate and 0 everywhere else. Note that \( f(x) = f(x + e_{j_0} + e_{j_1}) \) and furthermore \( \chi_a(x) = -\chi_a(x + e_{j_0} + e_{j_1}) \). We conclude that \( \hat{f}(a) = 0 \) for every \( a \in \mathbb{Z}_2^n \) satisfying \( a \neq (1, \ldots, 1) \) and \( a \neq 0 \).

The Fourier transform is a linear operator: \( \lambda \hat{f} + g = \lambda \hat{f} + \hat{g} \), and we have the following easy observation.

**Lemma 16.9.** The Fourier transform considered as an operator from \( L_1(G) \) to \( L_\infty(\hat{G}) \) is norm decreasing:
\[
\| \hat{f} \|_\infty \leq \| f \|_1.
\]
**Proof.** By (16.1) for every \( a \in G \), we have
\[
|\hat{f}(a)| = |\mathbb{E} f(x) \chi_a(x)| \leq \mathbb{E} |f(x)||\chi_a(x)| = \mathbb{E} |f(x)| = \| f \|_1.
\]
\[\square\]

The Fourier coefficient \( \hat{f}(0) \) is of particular importance as
\[
\hat{f}(0) = \mathbb{E} |f(x)|.
\]
So if \( 1_A \) is the indicator function of a subset \( A \subseteq G \), then \( \hat{1}_A(0) = |A|/|G| \) corresponds to the density of \( A \).

It follows from the fact that the characters from an orthonormal basis for \( L_2(G) \) that
\[
f = \sum_{a \in G} \hat{f}(a) \chi_a,
\]
and that this expansion of \( f \) as a linear combination of characters is unique. This formula is called the Fourier inversion formula as it shows how the functions \( f \) can be reconstructed from its Fourier transform.

If \( A \subseteq G \), then the orthogonal complement of \( A \) is defined as
\[
A^\perp = \{ a \in G : \chi_a(x) = 1 \ \forall x \in A \}.
\]
It follows from the identities \( \chi_0 = 1 \) and \( \chi_a \chi_b = \chi_{a+b} \) that \( S^\perp \) is a subgroup of \( G \). The Fourier transform of the indicator function of a subgroup of \( G \) has a simple form:

**Lemma 16.10.** If \( H \) is a subgroup of \( G \), then for every \( a \in G \), we have
\[
\hat{1}_H(a) = \begin{cases} 
|H|/|G| & a \in H^\perp \\
0 & a \notin H^\perp
\end{cases}
\]
**Proof.** If \( a \in H^\perp \), then
\[
\hat{1}_H(a) = \langle 1_H, \chi_a \rangle = \mathbb{E} 1_H(x) \chi_a(x) = \mathbb{E} 1_H(x) = |H|/|G|.
\]
On the other hand if \( a \notin H^\perp \), then there exists \( y \in H \) such that \( \chi_a(y) \neq 1 \). Then
\[
\sum_{z \in H} \chi_a(z) = \chi_a(y) \sum_{z \in H} \chi_a(z - y) = \chi_a(y) \sum_{z \in H} \chi_a(z),
\]
which shows that \( \sum_{z \in H} \chi_a(z) = 0 \). Hence
\[
\hat{1}_H(a) = \mathbb{E} 1_H(x) \chi_a(x) = \frac{1}{|G|} \sum_{z \in H} \mathbb{E} 1_H(z) = 0.
\]
\[\square\]
Definition 16.14. Let \( f = y + H \) be a coset of \( H \) in \( G \) (i.e. \( H \) is a subgroup of \( G \) and \( y \in G \)), then for every \( a \in G \),

\[
\hat{1}_A(a) = \mathbb{E}_a 1_A = \mathbb{E}_a 1_H(x - y)\chi_a(x) = \mathbb{E}_a 1_H(x)\chi_a(x + y) = \chi(y)\hat{1}_H(a)
\]

Indeed by Lemma 16.10, we have

Example 16.12. Let us revisit Example 16.8 in light of Remark 16.11. Note that \( \hat{f} \) is a subgroup of \( G \).

Example 16.11. is a subgroup of \( \mathbb{Z}^n_2 \).

Hence \( H^\perp = \{ a : (-1)^{\sum_{i=1}^n x_i a_i} = 1 \ \forall x \in H \} = \{(0, \ldots, 0), (1, \ldots, 1)\} \).

Next we prove the Parseval’s identity, a very simple but extremely useful fact in Fourier analysis.

Theorem 16.13 (Parseval). For every \( f \in L_2(G) \),

\[
\|f\|^2 = \sum_{a \in G} |\hat{f}(a)|^2.
\]

Proof. We have

\[
\|f\|^2 = \langle f, f \rangle = \sum_{a \in G} \hat{f}(a)\chi_a \sum_{b \in G} \hat{f}(b)\chi_b = \sum_{a,b \in G} \hat{f}(a)\hat{f}(b)\langle \chi_a, \chi_b \rangle.
\]

The identify now follows from orthonormality of characters:

\[
\langle \chi_a, \chi_b \rangle = \begin{cases} 0 & a \neq b; \\ 1 & a = b. \end{cases}
\]

The proof of the Parseval identity, when applied to two different functions \( f, g \in L_2(G) \), implies the Plancherel theorem:

\[
\langle f, g \rangle = \sum_{a \in G} \hat{f}(a)\overline{\hat{g}(a)}.
\]

As the first example of an application of the Parseval identity, let us show that for every subgroup \( H \) of \( G \), we have

\[
|H||H^\perp| = |G|.
\]

Indeed by Lemma 16.10, we have

\[
\frac{|H|}{|G|} = \mathbb{E}_H 1 = \mathbb{E}_H 1_H = \langle 1_H, 1_H \rangle = \|1_H\|^2 = \sum_{a \in G} |\hat{1}_H(a)|^2 = \sum_{a \in H^\perp} (|H||G|)^2 = \frac{|H|^2|H^\perp|}{|G|^2}
\]

which simplifies to (16.2).

Next we introduce the important notion of convolution.

Definition 16.14. Let \( G \) be a finite Abelian group. For two functions \( f, g : G \to \mathbb{C} \), we define their convolution

\( f \ast g : G \to \mathbb{C} \) as

\[
f \ast g(x) = \mathbb{E}_{y \in G} [f(x - y)g(y)].
\]

Note that \( f \ast g(x) \) is the average of \( f(a)f(b) \) over all pairs \( a, b \) with \( a + b = x \). This gives a combinatorial nature to convolution which makes it very useful in dealing with certain discrete problems. Consider a set \( A \subseteq G \). Then \( f \ast 1_A(x) \) is the average of \( f \) over the set \( x - A := \{ x - y : y \in A \} \). For example if \( A \) is the Hamming ball\(^1\) of radius \( r \) around 0 in \( \mathbb{Z}_2^n \), then \( f \ast 1_A(x) \) is the average of \( f \) over the Hamming ball of radius \( r \) around \( x \). These types of averaging operators usually “smooth” \( f \), and makes it more similar to a constant functions. This smoothing property of the convolution is one of the main tools in harmonic analysis and this course.

Next let us list some basic facts about the convolution. We define the support of \( f : G \to \mathbb{C} \), denoted by \( \text{supp}(f) \), to be the set of the points \( x \in G \) with \( f(x) \neq 0 \).

\(^1\)The Hamming ball of radius \( r \) around 0 is defined as \( \{ x \in \mathbb{Z}_2^n : \sum_{i=1}^n x_i \leq r \} \subseteq \mathbb{Z}_2^n \).
Lemma 16.15. Consider three functions \( f, g, h : G \to \mathbb{C} \).

(a) We have \( f * g = g * f \).

(b) We have \( (f * g) * h = f * (g * h) \).

(c) We have \( f * (\lambda h + g) = \lambda f * h + f * g \).

(d) We have \( \text{supp}(f * g) \subseteq \text{supp}(f) + \text{supp}(g) \).

(e) We have \( \|f * g\|_\infty \leq \|f\|_1 \|g\|_\infty \).

(f) More generally, if \( p \) and \( q \) are conjugate exponents, then
\[
\|f * g\|_\infty \leq \|f\|_p \|g\|_q.
\]

(g) We have \( \|f * g\|_1 \leq \|f\|_1 \|g\|_1 \).

Proof. (a) For every \( x \in G \), we have
\[
f * g(x) = E_y[f(x - y)g(y)] = E_y[f(x - y)g(x - (x - y))] = E_z[f(z)g(x - z)] = g * f(x).
\]

(b) By Part (a),
\[
(f * g) * h(x) = (g * f) * h(x) = E_z E_y[g(x - z - y)f(y)]h(z) = E_y E_z g(x - z - y)f(y)h(z) = (h * g) * f(x) = f * (g * h)(x).
\]

(c) is trivial.

(d) follows from the fact that \( f(x) \) is the average of \( f(a)g(b) \) over all pairs of points \( a, b \in G \) with \( a + b = x \).

(e) is a special case of (f).

(f) Note that for every \( x \in G \), by Hölder's inequality we have
\[
|f * g(x)| \leq E_{y \in G}|f(x - y)||g(y)| \leq (E|f(x - y)|^p)^{1/p} (E|g(y)|^q)^{1/q} = (E|f(y)|^q)^{1/q} = \|f\|_p \|g\|_q.
\]

(g) We have \( \|f * g\|_1 = E_x |f * g(x)| \leq E_x E_{x,y}|f(x - y)||g(y)| = E_{z,y}|f(z)||g(y)| = E_{z} E_{x,y}|f(y)| = \|f\|_1 \|g\|_1 \).

The relevance of the Fourier transform to convolution lies in the following lemma.

Lemma 16.16. If \( f, g : G \to \mathbb{C} \), then
\[
\hat{f} \ast \hat{g} = \hat{f} \cdot \hat{g}.
\]

Proof. We have
\[
\hat{f} \ast \hat{g}(a) = E_x f * g(x) \overline{\chi_a(x)} = E_x (E_y f(x - y)g(y)) \overline{\chi_a(x)} = E_{x,y} f(x - y)g(y) \overline{\chi_a(x - y)} \overline{\chi_a(y)} = E_{x,y} f(z)g(y) \overline{\chi_a(z)} \overline{\chi_a(y)} = \hat{f}(a) \cdot \hat{g}(a).
\]
Note that Lemma 16.16 in particular shows that
\[ \mathbb{E}f(x)\mathbb{E}g(x) = \hat{f}(0)\hat{g}(0) = \hat{f} \ast g(0) = \mathbb{E}f \ast g(x). \]

We also have the dual version of Lemma 16.16,
\[ \hat{f} \cdot g(x) = \sum_{y \in G} \hat{f}(x - y)\hat{g}(y), \quad (16.3) \]
which converts point-wise product back to convolution.

For a function \( h : G \to \mathbb{C} \), define \( \tilde{h} : G \to \mathbb{C} \) as \( \tilde{h} : x \mapsto h(-x) \). Note that \( \tilde{h} = \sum_{a \in G} \overline{h(a)}\chi_a \). Hence it follows from the Parseval identity and Lemma 16.16 that for \( f, g, h : G \to \mathbb{C} \), we have
\[ \langle f \ast h, g \rangle = \langle f, g \ast \tilde{h} \rangle = \sum_{a \in G} \hat{f}(a)\overline{\hat{h}(a)}\hat{g}(a). \]
Infinite Abelian groups and beyond

Some readers might be familiar with the Fourier analysis over reals. Note that \( \mathbb{R} \) is a topological group: It is a group with a topology on it such that both the group’s binary operation \((x, y) \mapsto x + y\) and the function mapping group elements to their respective inverses \((x \mapsto -x)\) are continuous functions with respect to the topology. Furthermore \( \mathbb{R} \) is locally compact. Every element has a compact neighbourhood. For example for every \( x \in \mathbb{R} \), the neighbourhood \([x - 1, x + 1]\) is compact.

It turns out that for every locally compact group \( G \), there is a unique (Borel) measure on \( G \) that is invariant under group action. That is \( \mu(Sa) = \mu(S) = \mu(aS) \) for every \( a \in G \) and measurable \( S \subseteq G \). As an interesting side note, the proof of the existence of this measure uses the Hall’s marriage theorem about perfect matching.

For example, the counting measure on \( \mathbb{Z} \), the uniform probability measure on a finite group, or the usual Borel measure on \( \mathbb{R} \) are all Haar measures of those groups. Now with this measure \( \mu_G \) one can define the inner product of \( f, g : G \to \mathbb{C} \) as

\[
\langle f, g \rangle = \int f(x) \overline{g(x)} \, d\mu_G(x).
\]

For example, for functions \( f, g : \mathbb{R} \to \mathbb{C} \), we have

\[
\langle f, g \rangle = \int f(x) \overline{g(x)} \, dx.
\]

We can also define the characters of the locally compact abelian group \( G \) as the set of continuous group homomorphisms \( \chi : G \to \mathbb{T} \), where \( \mathbb{T} \) is the unit complex circle. As in the finite case, it turns that characters of \( G \) from a (locally compact) Abelian group \( \hat{G} \) that is called the Pontryagin dual of \( G \).

In the case of \( \mathbb{R} \), the characters are

\[
\chi_r : x \mapsto e^{-2\pi r x},
\]

for \( r \in \mathbb{R} \). Note that \( \chi_0 \) is the principal character, and \( \chi_r \chi_t = \chi_{r+t} \), and as a result \( \hat{\mathbb{R}} \cong \mathbb{R} \). Other examples are \( \hat{\mathbb{Z}} \cong \mathbb{T} \), and \( \hat{\mathbb{T}} \cong \mathbb{Z} \). So unlike the finite case, in the infinite case it is not necessarily true that the dual group is isomorphic to \( G \).

Now consider \( f \in L^1(G) \). That is \( f : G \to \mathbb{C} \) and \( ||f||_1 = \int |f| \, d\mu_G < \infty \). Then for every character \( \chi \) of the group, we can define a corresponding Fourier coefficient:

\[
\hat{f}(\chi) = \int f(x) \chi(x) \, d\mu_G(x).
\]

Note that this integral is well-defined precisely because

\[
\int f(x) \chi(x) \, d\mu_G(x) \leq \int |f(x)||\chi(x)| \, d\mu_G = \int |f(x)| \, d\mu_G < \infty.
\]

So it is important that \( f \in L_1(G) \). This gives us the Fourier transform

\[
F : L_1(G) \to L_1(\hat{G}),
\]

that maps the function \( f \) to its Fourier coefficients:

\[
F : f \mapsto \hat{f}.
\]

For example in the case of \( \mathbb{R} \), we get the familiar formula

\[
\hat{f}(r) = \int_{-\infty}^{\infty} f(x) e^{-2\pi r x} \, dx.
\]

The Fourier inversion theorem says that we have

\[
f = \int_{-\infty}^{\infty} \hat{f}(\chi) \chi(x) \, d\nu_\hat{G}(\chi),
\]

where \( \nu_\hat{G} \) is the Haar measure of the dual group \( \hat{G} \). For example, for \( \mathbb{R} \), we get the familiar Fourier inversion formula

\[
f(x) = \int_{-\infty}^{\infty} \hat{f}(r) e^{-2\pi r x} \, dr,
\]

that holds for every \( f \in L_1(\mathbb{R}) \).

Finally let us remark that one cannot apply Fourier analyses to non-Abelian groups simply because the characters are defined as homomorphisms into the Abelian group \( \mathbb{T} \), and for a non-Abelian group \( G \) there might not be any such non-trivial homomorphisms. In those cases the Fourier analysis is replaced with the representation theory where one considers the homomorphisms from \( G \) to the non-Abelian group of linear transformations of a vector space.
Concluding remarks:

We finish this chapter by remarking that the Fourier expansion on $\mathbb{Z}_n^2$ and the polynomial representation of functions $f : \{-1, 1\}^n \to \mathbb{R}$ are essentially identical.

As we saw above the characters of the group $\mathbb{Z}_n^2$ are of the form $\chi_S(x) = (-1)^{\sum_{i \in S} x_i}$ for all $S \subseteq [n]$. Note that changing the domain from $\mathbb{Z}_n^2$ to $\{-1, 1\}^n$, will convert the character $\chi_S$ to the monomial $x^S := \prod_{i \in S} x_i$. Hence the Fourier expansion turns into the polynomial representation

$$f(x) := \sum_{S \subseteq [n]} \hat{f}(S)x^S.$$

Exercises

Exercise 16.1. Prove the Identity (16.3).

Exercise 16.2. Suppose that for $f : \mathbb{Z}_n^2 \to \{0, 1\}$ we have $\hat{f}(S) = 0$ for all $|S| \geq 2$ (that is $\deg(f) \leq 1$). Show that either $f \equiv 0$, $f \equiv 1$, $f(x) = x_i$, or $f(x) = 1 - x_i$ for some $i \in [n]$.

Exercise 16.3. Let $G$ be a finite Abelian group, and $H$ be a subgroup of $G$. Prove that

$$(H^\perp)^\perp = H.$$ 

Exercise 16.4. Given two subsets $A, B \subseteq 2^{[n]}$. For $S \subseteq [n]$, let

$$\text{parity}_S(A) := \{A \in A \mid |A \cap S| \equiv 0 \mod 2\},$$

and

$$\text{parity}_S(B) := \{B \in B \mid |B \cap S| \equiv 0 \mod 2\}.$$

- Prove that if $\text{parity}_S(A) = \text{parity}_S(A)$ for every $S$, then $A = B$.
- Suppose $|\text{parity}_S(A) - \text{parity}_S(A)| \leq \delta$ for every $S$. Bound $|(A \setminus B) \cup (B \setminus A)|$ in terms of $\delta$.

Exercise 16.5. Let $G$ be a finite Abelian group, and $H$ be a subgroup of $G$. Prove that for every $f : G \to \mathbb{C}$, we have

$$\mathbb{E}_{x \in H} f(x) = \sum_{a \in H^\perp} \hat{f}(\chi_a).$$

Exercise 16.6. Let $G$ be a finite Abelian group and $f, g : G \to \mathbb{C}$. Show that for every positive integer $m$,

$$\|f * g\|_m \leq \|f\|_1 \|g\|_m.$$ 

Exercise 16.7. Consider a function $f : \mathbb{Z}_n^2 \to \{0, 1\}$ and its Fourier expansion $f = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S$. Define the discrete derivative of $f$ in direction $i \in \{1, \ldots, n\}$ as $\Delta_i f : x \mapsto f(x + e_i) - f(x)$. Write the Fourier expansion of $\Delta_i f$ in terms of the Fourier coefficients of $f$.

Exercise 16.8. Suppose that for $f : \mathbb{Z}_n^2 \to \{0, 1\}$ we have $\hat{f}(S) = 0$ for all $|S| > k$ (that is $\deg(f) \leq k$). Show that all the Fourier coefficients of $f$ are of the form $\frac{r}{2^k}$ where $r \in \{0, \pm 1, \ldots, \pm 2^k\}$. Conclude that every such function depends on at most $k2^{2k}$ coordinates.
Chapter 17

Learning and \(\text{AC}^0\) circuits

As we saw in Chapter 16, every function \(f : \{-1,1\}^n \rightarrow \mathbb{R}\) has a unique representation as a polynomial in variables \(x_1, \ldots, x_n\). That is

\[
f(x) = \sum_{S \subseteq [n]} \hat{f}(S)x^S,
\]

where \(x^S = \prod_{i \in S} x_i\), and \(\hat{f}(S)\) are real numbers. Furthermore, this representation is sometimes referred to as the Fourier expansion of \(f\), related to the Fourier analysis of the group \(\mathbb{Z}_2^n\), as explained in Chapter 16. In this chapter we will study the Fourier expansion of the functions in \(\text{AC}^0\), and see an application of this to the area of theoretical machine learning.

PAC learning from uniform samples

Consider a class \(C\) of Boolean functions \(f : \{-1,1\}^n \rightarrow \{0,1\}\). For example \(C\) could be the set of all such \(f\) with \(dt(f) \leq k\), where \(dt(f)\) refers to the smallest height of a decision-tree computing \(f\). Now suppose that we are facing the task of learning a function \(f \in C\) that is unknown to us by observing uniformly sampled points \(x\). In other words, we will be given a certain number of observations of the form \((x, f(x))\) where \(x\) are sampled independently and uniformly at random from \(-1,1\)^n, and from these samples, we want to guess \(f(y)\) every value of \(y\). In other words, we will produce a hypothesis \(h\) that is not necessarily in \(C\) but nevertheless it is a good approximation of \(f\). That is

\[
\Pr_y[h(y) \neq f(y)] \leq \varepsilon.
\]

Lemma 17.1. Given access to random samples from \(f : \{-1,1\}^n \rightarrow \{0,1\}\), there is a randomized algorithm which takes as input \(S \subseteq \{1, \ldots, n\}\), and \(0 \leq \delta, \varepsilon \leq \frac{1}{2}\), and using at most \(O(\log(1/\delta)\varepsilon^{-2})\) samples, outputs an estimate \(\tilde{f}(S)\) for \(\hat{f}(S)\) such that

\[
\Pr[|\tilde{f}(S) - \hat{f}(S)| > \varepsilon] \leq \delta.
\]

Lemma 17.1 shows that each individual coefficient can be estimated very well from few samples. However since there are \(2^n\) coefficients \(S\), estimating all of them will not be computationally efficient and moreover the errors in these estimates can accumulate to a large error. Indeed one should not expect to learn a generic function from a few random samples unless the function \(f\) is restricted to have some structure.
Now suppose that we know something about the Fourier expansion of the functions in the class \(C\). Namely, there is a small set \(S \subseteq \mathcal{P}([1, \ldots, n])\) such that most of the mass of the Fourier coefficients is concentrated in \(S\):

\[
\sum_{S \in S} |\hat{f}(S)|^2 \leq \varepsilon.
\]

Compare this to

\[
\sum_{S} |\hat{f}(S)|^2 = \mathbb{E}[|f(y)|^2] = \mathbb{E}f.
\]

If this is the case, then we can focus on estimating only those \(\hat{f}(S)\) for \(S \in S\), and obtain a good estimate of \(f\) as

\[
\sum_{S \in S} \hat{f}(S)x^S \approx f.
\]

**Theorem 17.2.** Suppose that there is a subset \(S \subseteq \mathcal{P}([1, \ldots, n])\) such that for every \(f \in C\), we have

\[
\sum_{S \in S} |\hat{f}(S)|^2 \leq \varepsilon.
\]

There is a randomized algorithm that uses at most \(O(|S| \log(|S|) \varepsilon^{-1})\) samples from an unknown \(f \in C\), and outputs a Boolean function \(h : \{0, 1\}^n \rightarrow \{0, 1\}\) such that with probability at least \(1 - \delta\),

\[
\Pr_x[f(x) \neq h(x)] \leq 8\varepsilon.
\]

**Proof.** Let \(K = |S|\). We use \(O \left( \log(K/\delta) \left( \frac{\sqrt{2}}{\sqrt{K}} \right)^2 \right) = O(\log(K/\delta)K\varepsilon^{-1})\) samples, to estimate \(\hat{f}(S) \approx \tilde{f}(S)\) for each \(S \in S\). The for every \(S\), we have

\[
\Pr \left[ |\hat{f}(S) - \tilde{f}(S)| > \frac{\sqrt{\varepsilon}}{\sqrt{K}} \right] \leq \frac{\delta}{K}.
\]

Hence by union bound,

\[
\Pr \left[ |\hat{f}(S) - \tilde{f}(S)| > \frac{\sqrt{\varepsilon}}{\sqrt{K}} \text{ for some } S \in S \right] \leq \delta.
\]

Let

\[
g = \sum_{S \in S} \tilde{f}(S)x^S,
\]

and note that if our estimates are successful, which happens with probability at least \(1 - \delta\), then \(g\) is a good estimate of \(f\). Indeed by Parseval

\[
\mathbb{E}_x[|f(x) - g(x)|^2] = \sum_{S \in S} |\hat{f}(S) - \tilde{f}(S)|^2 + \sum_{S \in S} |\tilde{f}(S)|^2 \leq K \left( \frac{\sqrt{\varepsilon}}{\sqrt{K}} \right)^2 + \varepsilon \leq 2\varepsilon.
\]

There is only one issue left to resolve, and that is \(g\) is not a Boolean function. However, this can be easily addressed. Let \(h\) be the Boolean rounding of \(g\) defined as \(h(x) = 0\) if \(g(x) < 1/2\) and otherwise \(h(x) = 1\). Note that since \(f\) is Boolean, we always have \(|f(x) - h(x)| \leq 2|f(x) - g(x)|\). Hence

\[
\Pr_x[f(x) \neq h(x)] = \mathbb{E}_x[|f(x) - h(x)|^2] \leq 4\mathbb{E}_x[|f(x) - g(x)|^2] \leq 8\varepsilon.
\]

\(\square\)

**Decision Trees**

Let \(C\) be the class of Boolean functions \(f : \{-1, 1\}^n \rightarrow \{0, 1\}\) computable by a decision tree of depth at most \(d\). It is easy to see that the degree of every such \(f\) as a polynomial is at most \(d\). Indeed if \(L_0\) and \(L_1\) are respectively the sets of the 0-leaves and 1-leaves of the decision tree, then

\[
f = \sum_{\ell \in L_1} c_\ell(x),
\]

where \(c_\ell(x) = 1\) if and only if \(x\) follows the path to \(\ell\) in the decision tree. Note that \(c_\ell\) only depends on the values of at most \(d\) variables in \(x\), and thus is of degree at most \(d\).

This shows that for this class \(C\) we can take \(S = \{S \mid |S| \leq d\}\), which is a set of size at most \(n^d\). We conclude that this class can be learned in the sense of Theorem 17.2 from at most \(O(n^d \log n^d)\) samples.
Linial-Mansour-Nisan

The switching lemma shows that under random restriction a function computable by an $\text{AC}^0$ circuit is likely to simplify to a function with small decision tree complexity. The small decision tree in particular implies small degree as a polynomial. This suggests that the functions in $\text{AC}^0$ must not have a large weight on the coefficients of the monomials of large degree. Linial, Mansour, and Nisan [LMN93] made this speculation into a theorem.

For a positive integer $k$, and a function $f : \{-1, 1\}^n \to \mathbb{R}$, we define

$$f^k := \sum_{S : |S| = k} \hat{f}(S)x^S,$$

and $f^{\leq k}$, $f^{\geq k}$, $f^{< k}$ and $f^{> k}$ are defined similarly. Note that by the Parseval identity,

$$\|f\|^2 = \sum_{k=0}^{n} \|f^k\|^2.$$

Since the degree of a decision tree is bounded by its depth, we have the following corollary to the Switching Lemma.

**Corollary 17.3.** Let $f$ be a Boolean function computed by an $\text{AC}$ circuit of size $M$ and depth $d$. Choose a random restriction $\rho$ by setting every variable independently to $*$ with probability $p = \frac{1}{10^{d^2 \cdot d}}$, and to 0 and 1 each with probability $\frac{1-p}{2}$. Then

$$\Pr[\deg(f_\rho) > s] \leq M^{2^{-s}}.$$

Now we are at the point to prove the main theorem of this section.

**Theorem 17.4** (Linial, Mansour, Nisan [LMN93]). Let $f$ be a Boolean function computed by an $\text{AC}$ circuit of depth $d$ and size $M$, and let $t$ be any integer. Then

$$\sum_{|S| > t} |\hat{f}(S)|^2 \leq 2M^{2^{-d/20}}.$$

In particular for $t = (40 \log(M/\varepsilon))d$, we have

$$\sum_{|S| > t} |\hat{f}(S)|^2 \leq \varepsilon.$$

**Proof.** Consider a random restriction $\rho \in \{-1, 1, *\}^n$ with $\Pr[*] = p = \frac{1}{10^{d^2 \cdot d}}$, for values of $k$ and $p$ to be determined later. We sample $\rho$ in two steps. First we pick $T \subseteq [n]$ corresponding to the positions that are not assigned a $*$. Then we pick $x_T \in \{-1, 1\}^T$ uniformly at random, and $\rho$ is defined as $\rho := (x_T, *)$. Set $f_{x_T} := f_\rho = f(x_T, \cdot)$. Since $x^S = \prod_{i \in S} x_i$, we can decompose it as

$$x^S = (x_T)^{S \cap T}(x_T)^{S \setminus T}.$$

Note that $f_{x_T} : \{-1, 1\}^T \to \{0, 1\}$ and since

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S)x^S = \sum_{S \subseteq [n]} \hat{f}(S)(x_T)^{S \cap T}(x_T)^{S \setminus T} = \sum_{A \subseteq T} \left( \sum_{B \subseteq T} \hat{f}(A \cup B)x_T^B \right) x_T^A,$$

we have

$$\hat{f}_{x_T}(A) = \sum_{B \subseteq T} \hat{f}(A \cup B)x_T^B,$$

for every $A \subseteq T$. Hence by the Parseval identity

$$\mathbb{E}_{x_T} \left| \hat{f}_{x_T}(A) \right|^2 = \sum_{B \subseteq T} \left| \hat{f}(A \cup B) \right|^2,$$

which shows that

$$\mathbb{E}_{x_T} \|f^{|A| > k}\|^2 = \mathbb{E}_{x_T} \sum_{A \subseteq T \atop |A| > k} \left| \hat{f}_{x_T}(A) \right|^2 = \sum_{A \subseteq T} \sum_{B \subseteq T} \left| \hat{f}(A \cup B) \right|^2 = \sum_{S : |S \cap T| > k} \left| \hat{f}(S) \right|^2.$$

105
Now we use the randomness in $T$. Since $f_{xT}^k = 0$ if $\deg(f) \leq k$, and that always $\|f_{xT}^k\|_2^2 \leq \|f_x\|_2^2 \leq 1$, we have

$$E_T \left[ \sum_{S:|S\cap T| > k} |\hat{f}(S)|^2 \right] = E_T E_x \|f_{xT}^k\|_2^2 = E_x \|f_{xT}^k\|_2^2 \leq \Pr[\deg(f) > k] \leq M 2^{-k},$$

(17.1)

where the last inequality follows from Corollary 17.3 as we chose $\Pr[*] = p \leq \frac{1}{10^d t}$. Also we can bound the left-hand size of (17.1) from below:

$$L.H.S. \text{ of } (17.1) = \sum_{S \subseteq [n]} \Pr[|S \cap T| > k] |\hat{f}(S)|^2 \geq \sum_{|S| > t} \Pr[|S \cap T| > k] |\hat{f}(S)|^2.$$

Taking $p = \frac{1}{10^d t}$ and $k = t^{1/d}/20$, satisfies $p \leq \frac{1}{10^d k^{20}}$, and by the Chernoff bound for $|S| > t$, the probability of $|S \cap T| > k = pt/2$ is at least $1 - 2e^{-pt^2} \geq \frac{1}{2}$. Hence by (17.1), we have

$$\sum_{S:|S| > t} \frac{1}{2} |\hat{f}(S)|^2 \leq M 2^{-t^{1/d}/20}.$$

\[ \square \]

**Mansour-Nisan conjecture**

Theorem 17.4 shows that the Fourier coefficients of every function computable by an AC circuit of polynomial size and constant depth are highly concentrated on the first $t = \log^{O(1)}(n)$ levels. There are roughly $n^t$ such coefficients. While this is not an exponential number, it is still super-polynomial. The following tantalizing conjecture due to Mansour speculates that for DNF’s one can pinpoint the significant coefficients to a polynomial size set.

**Conjecture 17.5** (Mansour [Man95]). Let $f$ be computable by a DNF with at most $t$ terms. For every $\varepsilon$, there exists a subset $S \subseteq \mathcal{P}([1, \ldots, n])$ of size $t^{O(\log 1/\varepsilon)}$ such that

$$\sum_{S \notin S} |\hat{f}(S)|^2 \leq \varepsilon.$$

**Exercises**

**Exercise 17.1.** Prove that for every $f$ computable by a decision tree with $L$ leaves, we have

$$\sum_S |\hat{f}(S)| \leq L.$$

**Exercise 17.2.** Let $f : \{-1, 1\}^n \to \{0, 1\}$ be a Boolean function such that

$$\sum_S |\hat{f}(S)| \leq K.$$

Prove that there exists a set $S$ of size at most $K^2/\varepsilon$ such that

$$\sum_{S \notin S} \hat{f}(S)^2 \leq \varepsilon.$$

**Exercise 17.3.** Prove that for every monotone $f$ we have

$$\sum_{S:|S| > t} \hat{f}(S)^2 \leq \varepsilon.$$

**Exercise 17.4.** Prove that for every monotone $f$ we have

- $2\hat{f}(\{i\}) = I_i(f)$ for every $i$.
- We have $I_f \leq 2\sqrt{n}$. 

106
Chapter 18

Communication complexity

The field of communication complexity, which is formally defined in 1979 in a paper by Yao [?], studies the amount of communication required to solve a problem of computing discrete functions when the input is split between two parties. Every Boolean function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$ defines a communication problem. An input $x \in \mathcal{X}$ is given to Alice, and an input $y \in \mathcal{Y}$ is given to Bob. Together, they should both compute the entry $f(x, y)$ by exchanging bits of information in turn, according to a previously agreed-on protocol. There is no restriction on their computational power; the only measure we care to minimize is the number of exchanged bits.

A deterministic protocol $\pi$ specifies for each of the two players, the bit to send next, as a function of their input and history of the communication so far. A protocol naturally corresponds to a binary tree as follows. Every internal node is associated with either Alice or Bob. If an internal node $v$ is associated with Alice, then it is labeled with a function $a_v : \mathcal{X} \rightarrow \{0, 1\}$, which prescribes the bit sent by Alice at this node as a function of her input. Similarly, Bob’s nodes are labeled with Boolean functions on $\mathcal{Y}$. Each leaf is labeled by 0 or 1 which corresponds to the output of the protocol. We denote the number of bits exchanged on the input $(x, y)$ by $\text{Cost}_\pi(x, y)$. This is exactly the length of the path from the root to the corresponding leaf. The communication cost of the protocol is simply the depth of the protocol tree, which is the maximum of $\text{Cost}_\pi(x, y)$ over all inputs $(x, y)$.

$$\text{CC}(\pi) := \max_{x, y} \text{Cost}_\pi(x, y).$$

Every such protocol $\pi$ computes a function $\mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$, which we also denote by $\pi$. Namely $\pi(x, y)$ is the label of the leaf reached by the path corresponding to the players communication on the input $(x, y)$. We say that $\pi$ computes $f$ if $\pi(x, y) = f(x, y)$ for all $x, y$. The deterministic communication complexity of $f$, denoted by $D(f)$, is the smallest communication cost of a protocol that computes $f$.

Example 18.1 (Equality Function, Identity Matrix). Consider the function $\text{EQ} : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ defined as $\text{EQ}(x, y) = 1$ if and only if $x = y$. Note that this function corresponds to the $2^n \times 2^n$ identity matrix. We obviously have $D(\text{EQ}) \leq n + 1$ as Alice can send $x$ to Bob by transmitting $n$ bits, and then Bob knowing both $x$ and $y$, can send $\text{EQ}(x, y)$ to Alice. In the next section, we will prove that $D(\text{EQ}) = n + 1$.

Note that the protocol described above works for every function. We have the following observation.

Proposition 18.2. For every function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$, we have

$$D(f) \leq n + 1.$$

Example 18.3 (Parity function). Consider the function $\text{Parity} : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ defined as

$$\text{Parity}(x, y) = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i \mod 2.$$

We have $D(\text{Parity}) = 2$ as Alice can send the one bit $\text{Parity}(x)$ to Bob, and then then Bob can compute $\text{Parity}(x, y)$, and send it to Alice.

Monochromatic rectangles, and Fooling sets

Consider a deterministic communication protocol for computing a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$. A useful insight in communication complexity is that a bit sent by Alice at a node $v$ corresponds to a partition of the rows into two parts
Definition 18.5 (Fooling sets). Let \( b \in \{0, 1\} \). A \( b \)-fooling set for a function \( f: \mathcal{X} \times \mathcal{Y} \to \{0, 1\} \) is a set \( S \subseteq \mathcal{X} \times \mathcal{Y} \) such that

- \( f \) is equal to \( b \) on all points in \( S \).
- For every distinct \((x_1, y_1), (x_2, y_2) \in S\), either \( f(x_1, y_2) \neq b \) or \( f(x_2, y_1) \neq b \).

The motivation behind the definition of fooling sets is that no two elements in \( S \) can belong to the same monochromatic rectangle. This follows from the fact that if \((x_1, y_1), (x_2, y_2) \in R\) for a rectangle \( R \), then \((x_1, y_2), (x_2, y_1) \in R\), and thus \( R \) cannot be monochromatic. We conclude the following proposition.

**Theorem 18.6** (Fooling Set Lower-bound). If \( S \) is a \( b \)-fooling set for \( f: \mathcal{X} \times \mathcal{Y} \to \{0, 1\} \), then every deterministic protocol for \( f \) must have at least \( |S| \) leaves that are labeled with \( b \).

Now we are ready to prove that the deterministic communication complexity of \( EQ \) is \( n + 1 \).

**Corollary 18.7.** We have \( D(EQ) = n + 1 \) for the equality function \( EQ: \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\} \).

**Proof.** Note that the diagonal \( \{(x, x) : x \in \{0, 1\}^n\} \) is a 1-fooling set of size \( 2^n \) for \( EQ \). Adding any non-empty 0-fooling set to this shows that every deterministic protocol for \( EQ \) must have at least \( 2^n + 1 \) leaves, and thus \( D(EQ) \geq \lfloor \log 2^n + 1 \rfloor = n + 1 \). \( \square \)

### Rectangle Size Bounds

This is another technique for proving lower bounds on communication complexity. In fact, the fooling set technique is a special case of the rectangle bound technique. Our basic strategy is to prove that the size of every monochromatic rectangle is small, and thus that many monochromatic rectangles are needed to partition \( \mathcal{X} \times \mathcal{Y} \).

Let \( f: \mathcal{X} \times \mathcal{Y} \to \{0, 1\} \). We define a probability distribution \( \mu \) on \( f^{-1}(1) \), and argue that \( \mu(R) \) is small for any rectangle \( R \) consisting of only 1’s. Alternatively, we can make the same proof by using rectangles consisting only of 0’s.

**Theorem 18.8** (Rectangle Size Bound). Let \( f: \mathcal{X} \times \mathcal{Y} \to \{0, 1\} \), and let \( \mu \) be a probability measure on \( \mathcal{X} \times \mathcal{Y} \) such that every 1-monochromatic rectangle has measure at most \( \delta \). Every deterministic protocol for \( f \) has at least \( 1/\delta \) leaves that are labeled with 1.

**Example 18.9.** Consider the inner product function \( IP: \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\} \) defined as

\[
IP(x, y) := \sum_{i=1}^{n} x_i y_i \mod 2.
\]

Let \( \mu \) be the uniform probability distribution on \( IP^{-1}(0) \). Note that \( \mu \) is supported on \( 4^n/2 = 2^{2n-1} \) points. Let \( R = S \times T \) be any 0-monochromatic rectangle. Identify \( \{0, 1\} \) with \( \mathbb{F}_2 \), and analogously \( \{0, 1\}^n \) with the \( n \)-dimensional vector space \( \mathbb{F}_2^n \). Then

\[
IP(x, y) = \langle x, y \rangle,
\]

where the inner product is defined through \( \mathbb{F}_2^n \). Let \( S' \) and \( T' \) be the linear spans of \( S \) and \( T \) over \( \mathbb{F}_2 \). Note that since \( \langle x, y \rangle = 0 \) for every \( (x, y) \in S \times T \), we also have that \( \langle x, y \rangle = 0 \) for every \( (x, y) \in S' \times T' \). Consequently \( \dim(S') + \dim(T') \leq n \), which shows that \( |R| \leq |S'| ||T'|| \leq 2^n \), and thus \( \mu(R) \leq \frac{2^n}{2^{2n-1}} = 2^{1-n} \). We conclude that

\[
D(IP) \leq \log 2^{n-1} = n - 1.
\]
Rank Lower-Bound

Note that every monochromatic rectangle corresponds to a rank-1 matrix. In particular if \( L_1 \) are all the 1-leafs of a protocol that computes a function \( f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\} \), then

\[
f = \sum_{\ell \in L_1} 1_{R_\ell},
\]

where \( R_\ell \) is the 1-monochromatic rectangle corresponding to \( \ell \). It follows that \( \text{rk}(f) \leq |L_1| \). We conclude the following theorem.

**Theorem 18.10.** For every function \( f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\} \), we have

\[
D(f) \geq \log \text{rk}(f).
\]

On the other hand it is not difficult to show that \( D(f) \leq \text{rk}(f) + 1 \). A major open problem in communication complexity is whether \( D(f) \) and \( \log \text{rk}(f) \) are polynomially related.

**Conjecture 18.11** (Long-rank Conjecture). Is there a constant \( c \) such that

\[
D(f) \leq O(\text{rk}(f)^c) ?
\]

Non-deterministic Communication complexity

**Exercises**

**Exercise 18.1.** Prove that the fooling set technique is a special case of the rectangle bound technique. More precisely, given a 1-fooling set \( S \), introduce an appropriate measure \( \mu \) such that Theorem 18.8 would imply that the number of 1-leaves is at least \( |S| \).

**Exercise 18.2.** Prove that \( D(f) \leq \text{rk}(f) + 1 \).
Chapter 19

Randomized Communication complexity

There are two different natural models to define a probabilistic communication protocol. In the private coin model, each party has its own random coin to flip. Alice cannot see Bob’s coin flips, and vice versa. Formally, Alice has access to a random string $R_A$ of arbitrary length, and similarly Bob has access to a random string $R_B$. The two strings are chosen independently, according to some probability distribution. Now in the protocol tree Alice’s nodes are labeled by functions of $R_A$ and $x$, and Bob’s nodes are labeled by functions of $R_B$ and $y$.

In the public coin model however, there is one common random string $R$ that is given to both Alice and Bob. We will mainly focus on this model, and note that in the public coin model, a probabilistic protocol $\pi_R$ is simply a distribution over deterministic protocols. In this notation $R$ is a random variable, and every fixation of $R$ to a particular value $r$ leads to a deterministic protocol $\pi_r$. We define the communication cost of a probabilistic protocol $\pi_R$ as the maximum cost of any protocol $\pi_r$ in the support of this distribution:

$$CC(\pi_R) = \max_r CC(\pi_r) = \max_r \max_{x,y} \text{Cost}_{\pi_r}(x,y).$$

We also define the average cost of such a protocol as the expected number of exchanged bits over the worst input $(x, y)$:

$$CC_{avg}(\pi_R) = \max_{x,y} E_R[\text{Cost}_{\pi_R}(x, y)].$$

In the probabilistic models of computation, three types of error are often considered.

- **Two-sided error**: This is the most important notion of randomized communication complexity. For every $x, y$, we require

$$\Pr_R[\pi_R(x, y) \neq f(x, y)] \leq \varepsilon,$$

where $\varepsilon$ is a fixed constant that is strictly less than 1/2. Note that $\varepsilon = 1/2$ can be easily achieved by outputting a random bit; hence it is crucial that $\varepsilon$ in the definition is strictly less than 1/2. It is common to take $\varepsilon = 1/3$. Indeed, the choice of $\varepsilon$ is not important so long as $\varepsilon \in (0, 1/2)$, since the probability of error can be reduced to any constant $\varepsilon' > 0$ by repeating the same protocol independently for some $O(1)$ times, and outputting the most frequent output.

The two-sided error communication complexity is simply called the randomized communication complexity. It is denoted by $R_\varepsilon(f)$ and is defined as the smallest communication cost $CC(\pi_R)$ of a probabilistic protocol that computes $f$ with two-sided error at most $\varepsilon$. We set $\varepsilon = 1/3$ as the standard error, and denote

$$R(f) = R_{1/3}(f).$$

- **One-sided error**: In this setting the protocol is only allowed to make an error if $f(x, y) = 1$. In other words, for every $x, y$ with $f(x, y) = 0$, we have

$$\Pr_R[\pi_R(x, y) = 0] = 1,$$

and for every $x, y$ with with $f(x, y) = 1$, we have

$$\Pr_R[\pi_R(x, y) \neq f(x, y)] \leq \varepsilon.$$

Again the choice of $\varepsilon$ is not important so long as $\varepsilon \in (0, 1)$ because the probability of error can be reduced from $\varepsilon$ to $\varepsilon^k$ by repeating the same protocol independently $k$ times and outputing 1 only when at least one of the
repetitions outputs 1. We denote by $R^1(f)$ the smallest $\text{CC}(\pi_R)$ over all protocols $\pi_R$ with one-sided error of at most $\varepsilon$. We set $\varepsilon = 1/3$ as the standard error, and denote

$$R^1(f) = R^1_{1/3}(f).$$

- **Zero error:** In this case the protocol is not allowed to make any errors. For every $x, y$, we must have $\Pr_{R}[\pi_R(x, y) \neq f(x, y)] = 0$. In this setting, $\text{CC}^{\text{avg}}(\cdot)$ is considered, as $\text{CC}(\cdot)$ leads to the same notion of complexity as the deterministic communication complexity. We denote

$$R_0(f) = \inf \text{CC}^{\text{avg}}(\pi_R),$$

over all such protocols.

Note that one can convert a zero-error protocol $\pi$ with average cost $c$ to a one-sided error protocol $\pi'$ with cost $3c$, by terminating the protocol after at most $3c$ steps, and outputting 0 in the case where the protocol is terminated prematurely. The protocol $\pi'$ clearly does not make any errors on 0-inputs. Furthermore, since the average cost of $\pi$ is $c$, by Markov's inequality, the probability that the protocol $\pi'$ is terminated prematurely is at most $1/3$. We conclude

$$R(f) \leq R^1(f) \leq 3R_0(f).$$

Obviously, $R(f), R^1(f), R_0(f)$ are all bounded by $D(f)$.

**Example 19.1 (Equality Function, Identity Matrix).** Consider the equality function $\text{EQ} : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$. Suppose that Alice and Bob use their common randomness to select a subset $S \subseteq \{0, 1\}^n$ uniformly at random and communicate to each other whether their inputs belong to $S$. If $x, y \in S$ or $x, y \notin S$, then they declare $x = y$, and otherwise declare $x \neq y$. Note that if $x = y$, they will always be correct, and if $x \neq y$, they will be correct with probability $1/2$. The error can be reduced obviously by repeating this procedure. We conclude

$$R(\text{EQ}) = O(1) \quad \text{and} \quad R^1(\overline{\text{EQ}}) = O(1),$$

where $\overline{\text{EQ}} = 1 - \text{EQ}$.

**Yao’s Min-Max Theorem**

Let $\mu$ be a probability distribution over $\mathcal{X} \times \mathcal{Y}$. The distributional communication complexity of $f$ is the cost of the best deterministic protocol that gives the correct answer for $f$ on at least a $(1 - \varepsilon)$-fraction of all inputs in $\mathcal{X} \times \mathcal{Y}$, weighted by $\mu$. More formally, $D^\mu_\varepsilon(f)$ is the smallest cost of a deterministic protocol $\pi$ such that

$$\Pr_{(x,y) \sim \mu} [\pi(x, y) \neq f(x, y)] \leq \varepsilon.$$

Let us start by the following easy observation.

**Proposition 19.2.** For every $\mu$, we have

$$D^\mu_\varepsilon(f) \leq R_\varepsilon(f).$$

*Proof.* Consider a randomized protocol $\pi_R$ of cost $c$. Then for every $(x, y)$, we have

$$\Pr_R[\pi_R(x, y) \neq f(x, y)] \leq \varepsilon.$$

Since this holds for every $(x, y)$, we can average it with $\mu$ weights.

$$\Pr_{(x,y) \sim \mu} [\pi_R(x, y) \neq f(x, y)] \leq \varepsilon.$$

Now there must exist an $r$ in the support of $R$ such that the deterministic protocol $\pi_r$ satisfies

$$\Pr_{(x,y) \sim \mu} [\pi_r(x, y) \neq f(x, y)] \leq \varepsilon,$$

and thus $D^\mu_\varepsilon(f) \leq c$. \qed

112
Proposition 19.2 suggests that in order to prove a lower-bound for $R_\varepsilon(f)$, one could come up with a distribution $\mu$ that is difficult for deterministic communication complexity, in the sense that every deterministic protocol of low cost will fail on $\varepsilon$ fraction (according to $\mu$ weights) of the inputs.

A natural question arises! What is the best that one can achieve with such lower-bounds? A well-known game theoretic argument shows that there is always a distribution $\mu$ that matches $R_\varepsilon(f)$. That is

$$\max_{\mu} D_\varepsilon^\mu(f) = R_\varepsilon(f).$$

Before proving this statement, let us recall the basic game theoretic result required for the proof.

Consider a zero-sum game between two players $A$ and $B$. That is there are two sets $S = \{s_1, \ldots, s_m\}$ and $T = \{t_1, \ldots, t_n\}$ corresponding to the possible strategies that players $A$ and $B$ can take, respectively. Let $M$ be the payoff matrix, that is $M = [m_{ij}]_{m \times n}$ is a real matrix, where $m_{ij}$ is the gain of Player $A$ (and loss of Player $B$) if they choose to use strategies $s_i$ and $t_j$. The term zero-sum here refers to the assumption that the gain of one player equals to the loss of the other player. In zero-sum games, this value is called the payoff.

In the terminologies of game theory the strategies $s_i$ and $t_j$ are called pure strategies. On the other hand a mixed strategy for Player $A$ is a probability distribution $\mu$ over $\{s_1, \ldots, s_m\}$, and a mixed strategy for Player $B$ is a probability distribution over $\{t_1, \ldots, t_n\}$. This means that Player $A$ picks a strategy $s_i$ according to $\mu$ and player $B$ picks a strategy $t_j$ according to $\nu$. What is the expected payoff?

$$E_{s_i \sim \mu, t_j \sim \nu}[m_{ij}] = \mu^t M \nu,$$

where on the right-hand-side $\mu$ and $\nu$ are represented as vectors. Obviously Player $A$ wants to maximize the pay-off and Player $B$ wants to minimize it.

Suppose that we first ask the Player $A$ to choose her mixed strategy $\mu$, and then allow Player $B$ to choose his strategy accordingly. Assuming that they are rational and play according to their best interests, they should arrive at the payoff of

$$\max_\mu \min_\nu \mu^t M \nu.$$

It might seem that this puts the Player $B$ in great advantage, since he can tailor his strategy according to $\mu$. Let us examine this case more carefully. Note that in this case, the pay-off is

$$\langle \mu^t M, \nu \rangle = \sum_{j=1}^n \nu_j \left( \sum_{i=1}^m \mu_i m_{ij} \right).$$

This is obviously minimized by a $\nu$ that assigns all its mass to the $j$ with the smallest $\sum_{i=1}^m \mu_i m_{ij}$. That is $\nu$ is a basically a pure strategy $t_j$:

$$\max_\mu \min_\nu \mu^t M \nu = \min_j \sum_{i=1}^m \mu_i m_{ij}.$$

Now let us reverse the order of the player, and force the Player $B$ to choose his strategy $\nu$ first, and then allow player $A$ to choose her strategy accordingly. This time, assuming rationality, the payoff will be

$$\min_\nu \max_\mu \mu^t M \nu,$$

and now it seems that Player $A$ is in great advantage. Also again note that Player $A$ can use a pure strategy without affecting the optimum payoff.

$$\min_\nu \max_\mu \mu^t M \nu = \max_j \sum_{i=1}^n m_{ij} \nu_j.$$

Perhaps surprisingly it turns out that it does not matter which player chooses their strategy first.

**Theorem 19.3** (Minimax theorem).

$$\max_\mu \min_\nu \mu^t M \nu = \min_\nu \max_\mu \mu^t M \nu.$$

We will not prove this theorem, but those readers who are familiar with linear programming should notice that it follows easily from the linear programming duality. Now let us apply this theorem to communication complexity.

**Theorem 19.4.** For every $f: \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$, we have

$$\max_\mu \mu D_\varepsilon^\mu(f) \leq R_\varepsilon(f).$$
Proof. We have already seen that \( \max_\mu D_\mu^\epsilon(f) \leq R_\epsilon(f) \).

To prove the opposite direction, let \( c = \max_\mu D_\mu^\epsilon(f) \), and let \( S = \mathcal{X} \times \mathcal{Y} \), and let \( T \) be the set of all deterministic protocols with communication cost at most \( c \). Consider the two player zero-sum game with pure strategies \( S \) and \( T \), where the payoff of \( (x, y) \in S \) and \( \pi \in T \) is 1 if \( \pi(x, y) \neq f(x, y) \), and 0 otherwise. Now a mixed strategy for player \( A \) is a distribution \( \mu \) on \( \mathcal{X} \times \mathcal{Y} \), and a mixed strategy for player \( B \) is a distribution \( \nu \) on \( T \), which is basically a randomized communication protocol of cost at most \( c \). The pay-off here is

\[
\Pr_{(x, y) \sim \mu, \pi \sim \nu} [f(x, y) \neq \pi(x, y)]
\]

By the minimax theorem

\[
\min_\nu \max_\mu \Pr_{(x, y) \sim \mu, \pi \sim \nu} [f(x, y) \neq \pi(x, y)] = \max_\mu \min_\nu \Pr_{(x, y) \sim \mu, \pi \sim \nu} [f(x, y) \neq \pi(x, y)] \leq \epsilon,
\]

where in the inequality we used to assumption that \( \max_\mu D_\mu^\epsilon(f) = c \). Hence we conclude that

\[
\min_\nu \max_\mu \Pr_{(x, y) \sim \mu, \pi \sim \nu} [f(x, y) \neq \pi(x, y)] \leq \epsilon,
\]

which shows that there exists a \( \nu \) such that for every \( (x, y) \), we have

\[
\Pr_{\pi \sim \nu} [f(x, y) \neq \pi(x, y)] \leq \epsilon.
\]

Consequently

\[
R_\epsilon(f) \leq c,
\]

since \( \pi \sim \nu \) is a randomized protocol of cost at most \( c \).

\( \square \)

Note that Theorem 19.4 is quite robust, and can be applied to any other situation where randomized protocols are defined as a probability distribution over deterministic protocols (e.g. query complexity, randomized algorithms, etc).

Exercises

Exercise 19.1. Prove the minimax theorem using linear programming duality.
Chapter 20

Discrepancy and Sign-rank

In this chapter we look at two extreme settings of communication complexity, and see that they lead to two important mathematical notions of discrepancy and sign-rank. Later in Chapter 21, we prove that these notions also arise as two fundamental concepts in theoretical machine learning.

Question 20.1. Given a function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, what is the best error that can be achieved by a randomized protocol of cost 2?

The reason we consider 2 bits of communication is to allow both Alice and Bob to speak. Note that $R_{1/2}(f) = 0$ as to achieve an error of 1/2 it suffices for Alice and Bob to agree on a random bit. Hence the question is: How much improvement can we gain over 1/2 if we are only allowed 2 bits of communication?

The second extreme setting is to be content with any improvement over the obvious error of 1/2, and ask for the smallest communication complexity that can achieve an error that is strictly smaller than 1/2. As we shall see, this question is only interesting in the private coin model.

Question 20.2. Given a function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, what is the smallest communication cost of a private-coin protocol that achieves an error that is strictly less than 1/2?

Discrepancy

It turns out that the answer to Question 20.1 is the so called discrepancy. This notion was introduced by Chor and Goldreich [CG88], and has become one of the most commonly used measures in communication complexity to prove lower bounds for randomized protocols. To define this parameter, it is more convenient to switch to ±1-valued functions.

Let $\mathcal{X}, \mathcal{Y}$ be finite sets. The discrepancy of a function $f : \mathcal{X} \times \mathcal{Y} \to \{-1,1\}$ is defined as

$$\text{Disc}(f) = \min_{\mu} \max_{A \times B \subseteq \mathcal{X} \times \mathcal{Y}} |\mathbb{E}_{x,y \sim \mu} [f(x,y)1_A(x)1_B(y)]|,$$

where the minimum is over probability distributions $\mu$ on $\mathcal{X} \times \mathcal{Y}$. The following theorem shows that discrepancy can be used to establish lower-bounds for randomized communication complexity, and furthermore it is the answer to Question 20.1.

Theorem 20.3. Let $f : \mathcal{X} \times \mathcal{Y} \to \{-1,1\}$, let $\varepsilon \in (0,1/2)$, and $\delta := \text{Disc}(f)$. We have

$$\log \left( \frac{\delta}{2\varepsilon} \right) \leq R_{1/2-\varepsilon}(f),$$

and moreover,

$$R_{1/2-\delta}(f) \leq 2.$$

Proof. First suppose that there is a randomized communication protocol with $R_{1/2-\varepsilon}(f) \leq c$. By Yao’s minimax theorem, for every probability distribution $\mu$ on $\mathcal{X} \times \mathcal{Y}$, we have

$$D^\mu_{1/2-\varepsilon}(f) \leq c,$$
and thus there is a deterministic protocol $\pi$ such that
\[ \frac{1}{2} + \varepsilon \leq \Pr_{x,y \sim \mu} [\pi(x, y) = f(x, y)], \]
which translates to
\[ \mathbb{E}_{x,y \sim \mu} [\pi(x, y)f(x, y)] \geq 2\varepsilon. \]
Since the leaves of the protocol provide a partition of $X \times Y$ into at most $2^c$ rectangles, there exists a combinatorial rectangle $A \times B$ such that
\[ \frac{2\varepsilon}{2^c} \leq \mathbb{E}_{x,y \sim \mu} [f(x, y)\mathbb{1}_A(x)\mathbb{1}_B(y)] \leq |\mathbb{E}_{x,y \sim \mu} [f(x, y)\mathbb{1}_A(x)\mathbb{1}_B(y)]|. \]
We conclude
\[ \text{Disc}(f) \geq \frac{2\varepsilon}{2^c}. \]

Next suppose that
\[ \text{Disc}(f) = \min_{\mu} \max_{A \times B \subseteq X \times Y} |\mathbb{E}_{x,y \sim \mu} [f(x, y)\mathbb{1}_A(x)\mathbb{1}_B(y)]| = \delta. \]
Consider the following two-player zero-sum game: Player 1 chooses $(A \times B, b)$, where $b \in \{-1, 1\}$, and Player 2 chooses an input $(x, y)$. If $(x, y) \not\in A \times B$, then the payoff is zero, and otherwise the payoff is 1 if $b = f(x, y)$, and $-1$ otherwise. Now we can rephrase the discrepancy as
\[ \text{Disc}(f) = \min_{\mu} \max_{(A \times B, b)} \mathbb{E}_{x,y \sim \mu} [f(x, y)b\mathbb{1}_A(x)\mathbb{1}_B(y)] = \delta. \]
By applying the minimax principle,
\[ \max_{\nu} \min_{x,y} \mathbb{E}_{(A \times B, b) \sim \nu} [f(x, y)b\mathbb{1}_A(x)\mathbb{1}_B(y)] = \delta. \]
In particular, there exists a distribution $\nu$ such that
\[ \min_{x,y} \mathbb{E}_{(A \times B, b) \sim \nu} [f(x, y)b\mathbb{1}_A(x)\mathbb{1}_B(y)] \geq \delta. \]
Consider the following randomized protocol $\pi_R$ of communication cost 2: Alice and Bob choose $(A \times B, b)$ according to the distribution $\nu$, they check to see if $x \in A$ and $y \in B$, and in that case they agree on the output $b$, and otherwise they agree on a random $\pm 1$ bit as their output.

Note that for every $(x, y)$,
\[ \mathbb{E}_R [\pi_R(x, y)f(x, y)] = \mathbb{E}_{(A \times B, b) \sim \nu} [f(x, y)b\mathbb{1}_A(x)\mathbb{1}_B(y)] \geq \delta, \]
which shows that
\[ \Pr_R[\pi_R(x, y) \neq f(x, y)] \leq \frac{1}{2} - \frac{\delta}{2}. \]

Note that by Theorem 20.3, we have the following corollary that answers Question 20.1.

**Corollary 20.4** (Answer to Question 20.1). Let $f : X \times Y \to \{-1, 1\}$. Then
\[ \inf \{\varepsilon : R_{\frac{1}{2} - \varepsilon}(f) \leq 2\} = \Theta(\text{Disc}(f)). \]

Recall that the spectral norm of a matrix $M$ is defined as
\[ \|M\| = \sup \frac{\|Mu\|_2}{\|u\|_2} = \sigma_{\text{max}}(M), \]
where $\sigma_{\text{max}}$ is the largest singular value of $M$. Let us also show how discrepancy can be bounded by the spectral norm of a matrix.

**Theorem 20.5.** For every $m \times n$ sign matrix $M$, we have
\[ \text{Disc}(A) \leq \frac{\|M\|}{\sqrt{mn}}. \]
Example 20.6. Consider the ±1 version of the inner product function: \( IP : \{0,1\}^n \times \{0,1\}^n \rightarrow \{-1,1\} \) defined as

\[
    IP(x, y) := (-1)^{\sum_{i=1}^n x_i y_i} = \chi_x(y),
\]

where \( \chi_x \) is the Fourier character corresponding to \( x \). Let \( H \) be the \( 2^n \times 2^n \) matrix with \( xy \)-entry being equal to \( IP(x, y) \).

By the orthogonality of Fourier characters, we have

\[
    HH^T(x, z) = \sum_y H(x, y)H(x, z) = \sum_y \chi_x(y)\chi_z(y) = \begin{cases} 0 & x \neq z \\ 2^n & x = z \end{cases}.
\]

Hence \( HH^T = 2^n I \), and thus \( \|H\| = \sqrt{2^n} = 2^{n/2} \). We conclude that

\[
    \text{Disc}(H) \leq \frac{2^{n/2}}{\sqrt{2^n \times 2^n}} = 2^{-n/2}.
\]

In particular

\[
    \text{R}(IP) = \Omega(n).
\]

Other formulations of discrepancy

In the proof of Theorem 20.3, we saw that

\[
    \text{Disc}(f) = \min_{\mu} \max_{A \times B \subseteq X \times Y} \| \mathbb{E}_{x,y \sim \mu} [f(x,y)1_A(x)1_B(y)] \| = \max_{\nu} \min_{A \times B, b} \mathbb{E}_{(x,y) \sim \nu} [f(x,y)b1_A(x)1_B(y)],
\]

where \( \mu \) is a distribution on \( X \times Y \), and \( \nu \) is a distribution over signed rectangles \( (A \times B, b) \), where \( b \in \{-1,1\} \).

Discrepancy is defined through combinatorial rectangles \( A \times B \) which correspond to \( 1_A \otimes 1_B \). Let us define a closely related variant of discrepancy where instead of \( \{0,1\} \)-valued vectors \( 1_A, 1_B \), we use \( \pm 1 \)-valued vectors.

\[
    \text{Disc}_{\pm}(f) = \min_{\mu} \max_{u : X \rightarrow \{-1,1\}, \nu : Y \rightarrow \{-1,1\}} \| \mathbb{E}_{x,y \sim \mu} [f(x,y)u(x)v(y)] \| = \max_{\nu} \min_{x,y} \mathbb{E}_{(u,v) \sim \nu} [f(x,y)u(x)v(y)], \tag{20.1}
\]

where \( \nu \) is a distribution over \( (u,v) \), where \( u : X \rightarrow \{-1,1\} \) and \( v : Y \rightarrow \{-1,1\} \). Since \( u(x)v(y) \) can be written as a \( \pm 1 \) sum of four rectangles, it is straightforward to see that

\[
    \text{Disc}(f) \leq \text{Disc}_{\pm}(f) \leq 4\text{Disc}(f),
\]

so the two notions are essentially equivalent.

Unbounded-Error model

As we have seen in the discussion about the complexity class \( \text{PP} \), achieving an error of \( 1/2 \) for a randomized protocol is trivial. Namely, for any Boolean function \( f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{-1,1\} \) and any input \( (x, y) \), the players can output a random bit \( b \in \{-1,1\} \), and they will be correct with probability \( 1/2 \). An unbounded-error protocol has a better success probability, but we do not care by how much.

A question arises: Should we consider public randomness protocols or private randomness protocols? First, we argue that this model is not very interesting for public randomness.
Proposition 20.7. For any $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$ there is a public randomness protocol $\pi_R$ of cost $O(1)$ such that
\[
\Pr_R[\pi_R(x,y) \neq f(x,y)] < \frac{1}{2}.
\]

Proof. Alice and Bob treat the public randomness as sampling an input $(x', y')$. They check if $x = x'$ and $y = y'$. If both agree, they can compute $f(x,y) = f(x', y')$. Otherwise, they output a random bit. Note that
\[
\Pr[\text{Error}] = \frac{1}{2} \Pr[(x,y) \neq (x', y')] < \frac{1}{2}.
\]

So, we restrict our attention from now on to private randomness protocols.

Definition 20.8 (unbounded-error communication complexity). The unbounded-error communication complexity of a function $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$, denoted by $U(f)$, is the smallest cost of a private-coin protocol $\pi$ such that
\[
\Pr[\pi(x,y) \neq f(x,y)] < \frac{1}{2} \quad \forall x,y.
\]

Sign-rank

Recall that the log-rank conjecture speculates that for deterministic protocols, the communication complexity is polynomially related to log of the rank of the corresponding matrix. A similar connection is known to be true for unbounded-error protocols, except that rank is replaced by sign rank.

Definition 20.9 (Sign-rank). The sign-rank of a matrix $A = [a_{ij}]$, denoted by $\text{rk}_\pm(A)$ is the smallest rank of a matrix $B = [b_{ij}]$ such that $a_{ij}b_{ij} > 0$ for all $i,j$.

Theorem 20.10 (Paturi and Simon [PS86]). For every $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$, we have
\[
U(f) = \log \text{rk}_\pm(f) \pm O(1).
\]

Proof. We first prove that $\log \text{rk}_\pm(f) \leq U(f) + O(1)$. Assume that $U(f) = c$. That is, there is an unbounded-error protocol $\pi$ for $f$ with cost $c$. Define $A(x,y) = \mathbb{E}_{R_A,R_B}[\pi(x,y)]$, which has the same sign as $f(x,y)$. Thus $\text{rk}_\pm(f) \leq \text{rk}(A)$. We next bound the rank of $A$. We can view $\pi$ as a protocol tree of depth $c$, where at every node $v$ there is a probability assigned to each of its children. If $v$ is an Alice’s node, then its left child is followed with probability $a_v(x) \in [0,1]$, and its right child with probability $1 - a_v(x)$. If $v$ is a Bob’s node, then the same thing holds, except that now the probabilities are $b_v(y)$ and $1 - b_v(y)$, respectively. Take a leaf $\ell$. The probability that we reach $\ell$ is the product of probabilities that we take the path leading to it, which is a product of $a_v(x)$, $1 - a_v(x)$, $b_v(y)$, or $1 - b_v(y)$ over the nodes $v$ in the path from the root to $\ell$. Putting them together, the probability that $\pi$ reaches a leaf $\ell$ can be succinctly written as $p_\ell(x)q_\ell(y)$, for some functions $p_\ell : \mathcal{X} \to [0,1]$, and $q_\ell : \mathcal{Y} \to [0,1]$. Hence denoting by $L$ the set of all leaves, we have
\[
A(x,y) = \sum_{\ell \in L} p_\ell(x)q_\ell(y),
\]
which shows that $\text{rk}(A) \leq |L| \leq 2^c$ as desired.

We next prove that $U(f) \leq \log \text{rk}_\pm(f) + O(1)$. Assume that $\text{rk}(f) = r$. Equivalently, there exist vectors $u_x, v_y \in \mathbb{R}^r$ such that $\text{sign}(u_x, v_y) = f(x,y)$ for all $(x,y) \in \mathcal{X} \times \mathcal{Y}$. Recall that for a vector $u$, we have $\|u\|_1 = \sum_{i=1}^r |u_i|$. We may assume that $\|u_x\|_1 = \|v_y\|_1$, for all $x,y$ as this does not change the premise. The benefit is that we can treat the entries $|(u_x)_1|, \ldots, |(u_x)_r|$ as a probability distribution over $[r]$, and similarly for $v_y$. Consider the following protocol:

- Alice samples an element $i \in [r]$ with probability $|(u_x)_i|$ and sends $i$ to Bob.
- In addition, Alice computes $a_i = \text{sign}((u_x)_i) \in \{-1,1\}$ and sends it to Bob.
- Bob, on receiving $i \in [r]$, $a_i \in \{-1,1\}$ from Alice, samples a random bit $b_i \in \{-1,1\}$ such that $\mathbb{E}[b_i] = (v_y)_i$, and send it to Alice.
- They agree on the output $a_ib_i$.  

118
Note that
\[ E[a_ib_i] = \sum_{i=1}^{r} |(u_x)_i| \text{sign}((u_x)_i)(v_y)_i = \sum_{i=1}^{r} (u_x)_i(v_y)_i = \langle u_x, v_y \rangle, \]
which together with \( \text{sign}(\langle u_x, v_y \rangle) = f(x, y) \) shows that the error probability of \( \pi \) is strictly less than \( \frac{1}{2} \). \( \square \)

In light of Theorem 20.10, we can forget about communication complexity, and focus on the mathematical notion of sign-rank. In Chapter 21, we will prove a celebrated theorem of Forster, which shows that similar to discrepancy, the spectral norm can be used to establish a lower-bound on sign-rank.

**Exercises**

**Exercise 20.1.** Prove that in
Chapter 21

Machine Learning: Dimension and Margin

In Chapter 20, we saw that the randomized communication complexity in the unbounded-error model is essentially equivalent to a basic matrix parameter called sign-rank. In this section, we study sign-rank and a closely related parameter called margin. Both these notions occupy a central place in the present-day machine learning in both theory and practice. For example the performance of large margin classifiers such as Support Vector Machines (a.k.a SVM) rely on low dimension, high margin representations of data. As we shall see, these precisely correspond to sign-rank and the margin of the corresponding matrix.

Sign-rank in Machine Learning: Dimension

Let $X = \{x_1, \ldots, x_n\}$ be a set of data items, e.g. a list of books, movies, songs, restaurants. In the terminology of machine learning, every function $f : X \to \mathbb{R}$ is called a concept. A concept class $C = \{f_1, \ldots, f_m\}$ is simply a set of functions $f_j : A \to \mathbb{R}$. We will be interested in the Boolean case, where concepts take $\pm 1$ values. Such concepts, for example, could correspond to likes/dislikes of a person, e.g $f_j(x_i) = 1$ means that the person $j$ likes the item $x_i$, and $f_j(x_i) = -1$ corresponds to disliking the item.

The goal in machine learning is to observe the value of an unknown $f \in C$ only on a few samples from $X$ (called training data), and accordingly produce a hypothesis $h : X \to \{-1, 1\}$ that predicts the value of $f$ on every item in $X$. For example, by observing the preferences of a person on a few books, a few songs, etc, we want to predict whether the person likes or dislikes the other items in our list.

It is very useful to represent the concepts and the data items geometrically as points in $\mathbb{R}^d$ for some $d$. For example, for a medical research, a person can be represented by their age, height, weight, etc, each corresponding to a coordinate of $\mathbb{R}^d$. In machine learning, the host space $\mathbb{R}^d$ is called the feature space, and every coordinate of $\mathbb{R}^d$ is thought of as a feature of the input.

Every concept $f : X \to \{-1, 1\}$, is a partition of the input points (which are now represented as points in $\mathbb{R}^d$) into two sets. Obviously it is desirable if these partitions are defined via mathematically nice objects such as hyperplanes.
**Definition 21.1.** We say that the concept class $C$ is realized by unit vectors $u_1, \ldots, u_m \in \mathbb{R}^d$ and $\tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^d$ if

$$f_j(x_i) = \text{sign}(\langle u_j, \tilde{x}_i \rangle),$$

where

$$\text{sign}(x) = \begin{cases} 
1 & x > 0 \\
0 & x = 0 \\
-1 & x < 0 
\end{cases}.$$  

Note that

$$\text{sign}(\langle u_j, \tilde{x}_i \rangle) = \text{sign} \left( \frac{\langle u_j \parallel u_j \parallel \parallel \tilde{x}_i \rangle}{\parallel \tilde{x}_i \parallel} \right),$$

and thus the assumption that the vectors $u_j$ and $\tilde{x}_i$ are unit vectors is without loss of generality.

In such a realization, the items are represented by the unit points $\tilde{x}_i$ in the space, and the concepts $f_j$ are represented by the half-space $H_j$ defined by the normal vectors $u_j$:

$$H_j = \{ x \in \mathbb{R}^d : \langle u_j, x \rangle > 0 \}.$$  

Such representations are very useful both for theoretical and practical reasons. For example, in Support Vector Machines, having observed the value of a few points, one tries to generate a hypothesis that is a hyperplane that well-separates the +1 points from −1 points. Intuitively a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point. Obviously for such algorithms to succeed, it is desirable to make sure that the data points are far away from boundary of these half-spaces. This is captured by the margin of the representation, which is defined as

$$\gamma = \min_{i,j} |\langle u_j, \tilde{x}_i \rangle|.$$  

Note that $\gamma$ is the closest distance of any point $\tilde{x}_i$ to one of the hyperplanes defined by $u_i$.

Two important questions arise naturally:

- **What is the smallest dimension $d$ such that $C$ can be realized in $\mathbb{R}^d$?**
- **What is the largest $\gamma$ such that there is a realization of $C$ with margin $\gamma$?** This quantity is called the margin of the concept class and is denoted by $\text{mgn}(C)$.

It turns out that sign-rank is the answer to the first question, and discrepancy is the answer to the second question! For the convenience of the reader, we recall the definition of the sign-rank.

**Definition 21.2 (Sign-rank).** The sign-rank of a matrix $A = [a_{ij}]$, denoted by $\text{rk}_\pm(A)$ is the smallest rank of a matrix $B = [b_{ij}]$ such that $a_{ij}b_{ij} > 0$ for all $i,j$.

**Proposition 21.3.** Let $C = \{ f_1, \ldots, f_m \}$ be a concept class of functions $f_j : \{ x_1, \ldots, x_n \} \rightarrow \{-1, 1\}$, and let $M = [m_{ij}]_{m \times n}$ be the matrix with entries $m_{ij} = f_i(x_j)$. The sign-rank of $M$ is the smallest $d$ such that $C$ can be realized in $\mathbb{R}^d$.

**Proof.** First note that if there is a realization of $C$ in $\mathbb{R}^d$ with vectors $u_1, \ldots, u_m$ and $v_1, \ldots, v_n$, then $M = \text{sign}(UV)$, where $U$ is the $m \times d$ matrix with rows $u_i$, and $V$ is the $d \times n$ matrix with columns $v_j$. Thus $\text{rk}_\pm(F) \leq \text{rk}(UV) \leq d$.

On the other hand, suppose that $M$ can be sign-represented by a matrix $B$ of rank $d$. That is $b_{ij}m_{ij} > 0$ for all $i,j$ and $\text{rk}(B) = d$. Then $B$ has a decomposition $B = UV$ where $U$ is an $m \times d$ matrix, and $V$ is a $d \times n$ matrix. Note that we can normalize the rows of $U$, and the columns of $V$ to turn them into unit vectors, without affecting the signs of the entries of $UV$. This concludes the proposition.  

**Non-homogeneous hyper-planes:** In the definition of the realization in $\mathbb{R}^d$, we used homogeneous hyperplanes, i.e. the ones that pass through the origin. One might wonder if we can gain some advantage by allowing half-spaces that are defined through general hyperplanes: For $u_i \in \mathbb{R}^d$ and $t_j \in [-1, 1]$:

$$H_j = \{ x \in \mathbb{R}^d : \langle u_j, x \rangle > t_j \}.$$  

It turns out allowing non-homogeneous polynomials does not bring much advantage neither regarding the dimension nor the margin. Indeed we can convert a non-homogeneous realization in $\mathbb{R}^d$ to a homogeneous realization in $\mathbb{R}^{d+1}$ by using the vectors

$$u_j' := \frac{1}{\sqrt{1 + t_j^2}} \begin{bmatrix} u_j \\ -t_j \end{bmatrix}, \quad \tilde{x}_i' = \frac{1}{\sqrt{2}} \begin{bmatrix} \tilde{x}_i \\ 1 \end{bmatrix}. $$
Note that
\[ \langle u_j', \bar{x}_i \rangle > 0 \Leftrightarrow \langle u_j, \bar{x}_i \rangle > t_j, \]
and furthermore the margin of this new realization is only by a factor of at most \( \frac{1}{2} \) worse than the old margin. Hence essentially without loss of generality we can work with homogeneous half-spaces.

**Low-degree polynomials:** Half-spaces are defined through degree one polynomials:
\[ x \mapsto \text{sign}(\langle u, x \rangle) = \text{sign} \left( \sum u_i x_i \right). \]

One way to generalize half-spaces is to use higher degree polynomials. Indeed such polynomial realizations are common in machine learning, and in the terminology of machine learning, those polynomials are called kernels. The following lemma shows that using low-degree polynomials do not bring us significant advantage regarding the degree.

**Lemma 21.4.** Let \( p_1, \ldots, p_n : \mathbb{R}^d \to \mathbb{R} \) be polynomials of degree \( e \). Let \( x_1, \ldots, x_n \in \mathbb{R}^d \). Define a sign matrix \( S \) given by \( S_{i,j} = \text{sign}(p_j(x_i)) \). Then \( \text{rk}_\pm(S) = O(d^\epsilon) \).

**Proof.** We use linearization. Let \( I_e \) denote the set of all \( \alpha = (\alpha_1, \ldots, \alpha_d) \in \mathbb{Z}_+^d \) such that \( \sum_{i=1}^d \alpha_i \leq e \). For \( x = (x_1, \ldots, x_d) \in \mathbb{R}^d \), and \( \alpha \in I_e \), we shorthand \( x^\alpha = \prod_{i=1}^d x_i^{\alpha_i} \). Note that these are all the monomials of degree at most \( e \).

Assume \( p_i(x) = \sum_{\alpha \in I_e} \lambda_{i,\alpha} x^\alpha \). For each \( p_i \) define the vector \( u_i = (\lambda_{i,\alpha} : \alpha \in I_e) \in \mathbb{R}^{|I_e|} \), and for every \( x \in \mathbb{R}^d \) define the vector \( v_x = (x^\alpha : \alpha \in I_e) \). Now note that \( p_i(x) = \langle u_i, v_x \rangle \). Hence
\[ \text{rk}_\pm(S) = |I_e| = O(d^\epsilon). \]

\[ \square \]

**Margin and Discrepancy**

**Theorem 21.5.** Let \( f : \mathcal{X} \times \mathcal{Y} \to \{-1, 1\} \). We have
\[ \text{Disc}_\pm(f) \leq \text{mgin}(f) \leq 2\text{Disc}_\pm(f). \]

**Proof.** Let \( \delta = \text{Disc}_\pm(M) \). By Equation 20.1, we have
\[ \delta = \max_{\nu \sim \mu} E_{(u,v) \sim \nu} \left[ f(x,y)u(x)v(y) \right], \]
where max is over distributions \( \mu \) over pairs \( (u,v) \) where \( u : \mathcal{X} \to \{-1, 1\} \), and \( v : \mathcal{Y} \to \{-1, 1\} \). In particular there exists a \( d \in \mathbb{N} \), and a distribution \( \nu \) over \( \{1, \ldots, d\} \) such that
\[ \delta = \min_{x,y} \sum_{i=1}^d \nu(i)f(x,y)u_i(x)v_i(y), \]
where \( u_i : \mathcal{X} \to \{-1, 1\} \) and \( v_i : \mathcal{Y} \to \{-1, 1\} \).

Define the unit vectors \( u_x, v_y \in \mathbb{R}^d \) as
\[ u_x = (\sqrt{\nu(1)}u_1(x), \ldots, \sqrt{\nu(d)}u_d(x)), \quad \text{and} \quad v_y = (\sqrt{\nu(1)}v_1(y), \ldots, \sqrt{\nu(d)}v_d(y)). \]

Note that for every \( (x,y) \in \mathcal{X} \times \mathcal{Y} \),
\[ \langle u_x, v_y \rangle = \sum_{i=1}^d \nu(i)u_i(x)v_i(y), \]
which by above satisfies
\[ \delta \leq f(x,y)\langle u_x, v_y \rangle. \]
Hence we obtained a realization of \( f \) with margin \( \delta \).

On the other hand, consider a realization \( u_x, v_y \in \mathbb{R}^d \) of \( f \) with margin \( \eta \). That is \( u_x, v_y \in \mathbb{R}^d \) are unit vectors, and for every \( x, y \) we have
\[ \eta \leq f(x,y)\langle u_x, v_y \rangle. \]
Pick a unit vector \( r \in \mathbb{R}^d \) uniformly at random, and define \( u_r : \mathcal{X} \to \{-1, 1\} \) and \( v_r : \mathcal{Y} \to \{-1, 1\} \) as
\[
 u_r(x) = \text{sign}(r, u_x) \quad \text{and} \quad v_r(y) = \text{sign}(r, v_y).
\]

We need to understand how \( \mathbb{E}_r u_r(x)v_r(y) \) compares to \( \langle u_x, v_y \rangle \). Note that we can focus on the 2 dimensional plane that contains \( u_x \) and \( v_y \). In this case, if the line that is perpendicular to \( r \) goes between \( u_x \) and \( v_y \), then \( u_r(x)v_r(y) = -1 \), and otherwise \( u_r(x)v_r(y) = 1 \). The angle between \( u_x \) and \( v_x \) is exactly \( \arccos(u_x, v_y) \), and thus we have
\[
 \mathbb{E}_r u_r(x)v_r(y) = \frac{\pi - \arccos(u_x, v_y)}{\pi} \times 1 + \frac{\arccos(u_x, v_y)}{\pi} \times (-1) = 2\left(\frac{\pi - \arccos(u_x, v_y)}{\pi}\right) = 2\arcsin(u_x, v_y).
\]

After some straightforward calculus, we obtain
\[
 f(x, y)\mathbb{E}_r u_r(x)v_r(y) = \frac{2f(x, y)\arcsin(u_x, v_y)}{\pi} \geq \frac{2\ln(1 + \sqrt{2})}{\pi}f(x, y)\langle u_i, v_j \rangle \geq \frac{\eta}{1.8}.
\]

\[ \square \]

The Matrix Theory Background

Before proving bounds on the sign-rank and margin, we need to recall some important concepts from matrix theory.

The inner product of two real \( m \times n \) matrices \( A = [a_{ij}] \) and \( B = [b_{ij}] \) is defined as
\[
 \langle A, B \rangle = \sum_{i,j} a_{ij}b_{ij} = \text{Tr}(AB^T),
\]
where \( \text{Tr} \) denotes the trace operator.

The space of matrices is endowed with several norms. We will focus on the so-called Schatten norms. If \( \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(m,n)} \geq 0 \) are the singular values of an \( m \times n \) matrix \( M \), then for \( p \in [1, \infty) \)
\[
 \|M\|_{S_p} = \left( \sum_i \sigma_i^p \right)^{1/p}.
\]

The \( p = \infty \) case coincides with the spectral norm of \( M \):
\[
 \|M\|_{S_\infty} = \sigma_1 = \|M\|_{\ell_2 \to \ell_2}.
\]

Another important case is \( p = 1 \), which coincides with the trace norm.
\[
 \|M\|_{S_1} = \sum_i \sigma_i = \|M\|_{\ell_\infty} = \text{Tr} \left( \sqrt{MM^T} \right),
\]
where \( B = \sqrt{MM^T} \), means that \( B \) is the unique positive semi-definite matrix that satisfies \( BB^T = MM^T \).

These norms in many ways behave similar to the \( \ell_p \) norms. For example, we have the Hölder inequality, which states that for \( p, q \in [1, \infty] \) with \( \frac{1}{p} + \frac{1}{q} = 1 \), we have
\[
 \langle A, B \rangle \leq \|A\|_p \|B\|_q.
\]

The case \( \cdot \|_{S_2} \) is particularly well-behaved, and we have a Parseval type equality:
\[
 \|M\|_{S_2}^2 = \sum_i \sigma_i^2 = \langle M, M \rangle = \sum_{i,j} m_{ij}^2.
\]

The \( \gamma_2 \) and \( \nu \) norms Next we discuss an important norm defined originally by Grothendieck. There are various equivalent ways to define this norm.

**Definition 21.6 (Definition of \( \|M\|_{\gamma_2} \).** Let \( M = [m_{ij}] \) be an \( m \times n \) real-valued matrix. Then \( \|M\|_{\gamma_2} \) is the smallest \( \lambda \) such that there exists \( d \in \mathbb{N} \), and vectors \( u_i, v_j \in \mathbb{R}^d \) with \( m_{ij} = \langle u_i, v_j \rangle \), and
\[
 \max_{i,j} \|u_i\| \|v_j\| = \lambda.
\]

\[ 124 \]
The corresponding dual norm $\|M\|_{\gamma_2}^*$ is defined as

$$\|M\|_{\gamma_2}^* = \sup \sum_{i,j} m_{ij} \langle u_i, v_j \rangle,$$

where the supremum is over all $d \in \mathbb{N}$, and unit vectors $u_i, v_j \in \mathbb{R}^d$.

It is easy to see that indeed $\| \cdot \|_{\gamma_2}^*$ is the dual norm of $\| \cdot \|_{\gamma_2}$:

$$\|M\|_{\gamma_2} = \sup_{N: \|N\|_{\gamma_2} = 1} \langle M, N \rangle.$$

As a side note, notice that $\|M\|_{\gamma_2}$ can be computed in polynomial time by solving the following semi-definite program:

$$\min \sum_{i,j} m_{ij} \langle u_i, v_j \rangle \quad \text{s.t.} \quad \langle u_i, u_i \rangle = 1 \quad \forall i \in [m]$$

$$\langle v_j, v_j \rangle = 1 \quad \forall j \in [n].$$

Similarly one can write a semi-definite program to compute $\|M\|_{\gamma_2}$. Note that in the definitions of $\|M\|_{\gamma_2}$ and $\|M\|_{\gamma_2}^*$, we allowed to use vectors in $\mathbb{R}^d$ for any $d$. It is not difficult to see that it suffices to restrict to $d \leq \min(m, n)$.

What if we require $d = 1$, or in other words instead of vectors, use numbers for $u_i$ and $v_j$? This leads to the definition of the norms $\|M\|_{\nu}$ and its dual $\|M\|_{\nu^*}$.

**Definition 21.7.** Let $M = [m_{ij}]$ be an $m \times n$ real-valued matrix. The norm $\|M\|_{\nu}$ is the smallest $\lambda$ such that there exists $u_i, v_j \in \mathbb{R}$ such that $m_{ij} = u_i v_j$, and

$$\max_{i,j} |u_i| |v_j| = \lambda.$$

Its dual is defined as

$$\|M\|_{\nu^*} = \sup \sum_{i,j} m_{ij} u_i v_j = \sup u^t M v,$$

where the supremum is over $u \in \{-1,1\}^m$ and $v \in \{-1,1\}^n$.

Since $\| \cdot \|_{\nu}$ is defined by restricting the definition of $\| \cdot \|_{\gamma_2}$ to $d = 1$, we obviously have

$$\|M\|_{\gamma_2} \leq \|M\|_{\nu} \quad \text{and} \quad \|M\|_{\nu^*} \leq \|M\|_{\gamma_2}^*.$$

How larger can $\|M\|_{\gamma_2}^*$ be compared to $\|M\|_{\nu^*}$? Equivalently how larger can $\|M\|_{\gamma_2}$ be compared to $\|M\|_{\nu}$? The answer to this question is the subject of the famous Grothendeick’s inequality as explained below.

Consider the unit vectors $u_i, v_j$ in $\mathbb{R}^d$ that maximize $\sum m_{ij} \langle u_i, v_j \rangle$. Now consider a uniform random unit vector $r \in \mathbb{R}^d$. This vector defines a hyperplane in the space, and let us round the points that are above the hyper-plane to 1, and the points below to -1. In other words, define $\tilde{u}_i = \text{sign}(\langle u_i, r \rangle)$, and $\tilde{v}_j = \text{sign}(\langle v_j, r \rangle)$. Applying the same proof as we saw earlier we get

$$\mathbb{E}_r \tilde{u}_i \tilde{v}_j \geq \frac{2 \ln(1 + \sqrt{2})}{\pi} \langle u_i, v_j \rangle \geq \frac{1}{1.8} \langle u_i, v_j \rangle.$$

We conclude Grothendeick’s inequality:

**Theorem 21.8 (Grothendeick’s inequality).** There exists a universal constant $K_G \leq \frac{\pi}{2 \ln(1 + \sqrt{2})} \leq 1.8$ such that for every matrix $M$,

$$\|M\|_{\nu^*} \leq \|M\|_{\gamma_2} \leq K_G \|M\|_{\nu^*}$$

and

$$\|M\|_{\gamma_2} \leq \|M\|_{\nu} \leq K_G \|M\|_{\gamma_2}^*.$$

This shows that the norms $\|M\|_{\gamma_2}$ and $\|M\|_{\nu}$ are equivalent. The following simple lemma relates $\| \cdot \|_{\nu^*}$ and $\| \cdot \|_{\gamma_2}$ to the usual spectral norm $\|M\|$.

**Lemma 21.9.** For every $m \times n$ matrix $M$, we have

$$\|M\|_{\gamma_2} \leq \sqrt{mn} \|M\|.$$
Proof. The upper bound: For all unit vectors \( u_i, v_j \in \mathbb{R}^d \), we have

\[
\sum_{i,j} m_{ij}(u_i, v_j) = \sum_{r=1}^{d} \sum_{i,j} m_{ij} u(r) v_j(r) = \sum_{r=1}^{d} (u(r), M v(r)) \leq \sum_{r=1}^{d} \|u(r)\|\|M\|\|v(r)\| \\
\leq \|M\| \left( \sqrt{\sum_{r=1}^{d} \|u(r)\|^2} \right) \left( \sqrt{\sum_{r=1}^{d} \|v(r)\|^2} \right) = \sqrt{mn} \|M\|.
\]

\[\square\]

Forster’s Sign-rank Lower-bound

Proving a strong lower-bound on the sign-rank of an explicit sign-matrix was open for 15 years until finally Forster in a breakthrough [For02] used geometric ideas to establish strong lower bounds on the sign-rank of Hadamard matrices and, more generally, all sign matrices with small spectral norm.

Consider a sign-matrix \( M \in \{-1,1\}^{m \times n} \), say with small \( \|M\| \). We want to prove a lower-bound on the dimension \( d \) of any realization of \( M \) in \( \mathbb{R}^d \) by vectors \( u_i, v_j \in \mathbb{R}^d \). First of all, as we discussed in Remark 21.11 we can always assume that these vectors are unit vectors, as replacing them with the unit vectors \( \frac{u_i}{\|u_i\|} \) and \( \frac{v_j}{\|v_j\|} \), does not change the \( \text{sign}(\langle u_i, v_j \rangle) \).

The key idea of Forster was to show that one can further assume that, similar to the standard bases \( e_1, \ldots, e_d \), the vectors \( u_i \) are evenly distributed in all the directions. Of course we cannot assume that \( u_i \) are orthonormal, simply because \( X \) might be much larger than \( d \). Instead this even distribution is captured by what is called isotropic position.

Definition 21.10. A set of vectors \( u_1, \ldots, u_m \in \mathbb{R}^d \) is said to be in isotropic position, if

\[
\sum_{i \in [m]} u_i^t u_i = \frac{m}{d} I_d,
\]

where \( I_d \) is the identity matrix. Equivalent, for every vector \( v \), we have \( \sum_{i \in [m]} \langle u_i, v \rangle^2 = \frac{m}{d} \|v\|^2 \).

Note that an orthonormal basis such as \( e_1, \ldots, e_d \) is in isotropic position, but perhaps the best way to visualize a set of points in isotropic positions is to imagine a large set of points that are well distributed on a sphere.

The following theorem shows how to transform any set of vectors in general position into a set in isotropic position. We will not prove the theorem. For the proof, either the original paper of Forster [For02] or the book of Lokam [Lok08] are good sources.

Theorem 21.11 (Forster [For02]). Let \( u_1, \ldots, u_m \in \mathbb{R}^d \) be in general positions, meaning that every \( d \) of them are linearly independent. There exists an invertible linear transformation \( T : \mathbb{R}^d \rightarrow \mathbb{R}^d \) such that the unit vectors \( \frac{T u_1}{\|T u_1\|}, \ldots, \frac{T u_m}{\|T u_m\|} \in \mathbb{R}^d \) are in isotropic position.

The assumption in Theorem 21.11 that the vectors \( u_i \) are in general position is not an issue for us. Indeed if \( u_1, \ldots, u_m \in \mathbb{R}^d \) and \( v_1, \ldots, v_n \in \mathbb{R}^d \) realizes a matrix \( M \in \{-1,1\}^{m \times n} \), then we can simply perturb the vectors \( u_i \) by adding a small random noise to them. This will not change the signs of \( \langle u_i, v_j \rangle \), but it will guarantee that with probability one, they are in general positions. Theorem 21.11 allows us to assume, without loss of generality, that in a realization of \( M \in \{-1,1\}^{m \times n} \) with unit vectors \( u_1, \ldots, u_m \in \mathbb{R}^d \) and \( v_1, \ldots, v_n \in \mathbb{R}^d \), the vectors \( u_i \) are in isotropic position. Indeed we can apply Theorem 21.11, and replace the original vectors with the unit vectors \( \frac{T u_i}{\|T u_i\|} \in \mathbb{R}^d \) and \( \frac{T^{-1} v_j}{\|T^{-1} v_j\|} \in \mathbb{R}^d \). Note that

\[
\langle T u_i, T^{-1} v_j \rangle = \langle u_i, v_j \rangle,
\]

which shows

\[
\text{sign} \left( \frac{T u_i}{\|T u_i\|} \times \frac{T^{-1} v_j}{\|T^{-1} v_j\|} \right) = \text{sign} \langle u_i, v_j \rangle.
\]

We are ready to prove Forster’s lower-bound on sign-rank.

Theorem 21.12 (Forster’s Theorem). Every matrix \( M \in \{-1,1\}^{m \times n} \) satisfies

\[
\text{rk}_{\pm}(M) \geq \frac{mn}{\|M\|_2^2} \geq \frac{\sqrt{mn}}{\|M\|}.
\]

126
Proof. Consider a realization of $M \in \{-1,1\}^{m \times n}$ with unit vectors $u_1, \ldots, u_m \in \mathbb{R}^d$ and $v_1, \ldots, v_n \in \mathbb{R}^d$. By Theorem 21.11, we can assume without loss of generality that $u_1, \ldots, u_m \in \mathbb{R}^d$ are in isotropic position.

Since $|\langle u_i, v_j \rangle| \leq 1$, and the fact that $u_i$ are in isotropic position, we have

$$\sum_{i,j} |\langle u_i, v_j \rangle| \geq \sum_{j=1}^{n} \sum_{i=1}^{m} |\langle u_i, v_j \rangle|^2 = \sum_{j=1}^{n} \frac{m\|v_j\|^2}{d} = \frac{mn}{d}.$$  

On the other hand,

$$\sum_{i,j} |\langle u_i, v_j \rangle| = \langle M, B \rangle \leq \|M\|_{\gamma_2} \|B\|_{\gamma_2} \leq \|M\|_{\gamma_2}.$$  

□

Exercises

Exercise 21.1. Prove that $\text{Tr}(AB) = \sum_{i,j} a_{ij}b_{ij}$.

Exercise 21.2. Let $M = [m_{ij}]$ be an $m \times n$ real-valued matrix. We have

$$\|M\|_{\gamma_2} = \sup \left\{ \|uMv^t\|_{\text{Tr}} : u \in \mathbb{R}^m, v \in \mathbb{R}^n, \|u\|, \|v\| \leq 1 \right\},$$

and in particular

$$\|M\|_{\gamma_2} \geq \frac{\|M\|_{\text{Tr}}}{\sqrt{mn}}.$$

Exercise 21.3. Prove

$$\|M\|_{\gamma_2} = \sup_{N : \|N\|_{\gamma_2} = 1} \langle M, N \rangle$$

Exercise 21.4. Prove that in the definitions of $\|M\|_{\gamma_2}$ and $\|M\|_{\gamma_2}^*$, one can restrict to $d \leq \min m, n$.

Exercise 21.5. Grothendieck


Exercise 21.7. Convert ±1 and $\alpha$ to Boolean.
Bibliography


129


[Raz87] A.A. Razborov, Lower bounds on the size of bounded depth circuits over a complete basis with logical addition, Mathematical notes of the Academy of Sciences of the USSR 41 (1987), no. 4, 333–338 (English). 71, 72, 85


[Vin04] N. V. Vinodchandran, AM$_{\exp}$ \not\subset (NP $\cap$ coNP)/poly, Inform. Process. Lett. 89 (2004), no. 1, 43–47. MR 2025889 88

[Vio09] Emanuele Viola, Guest column: correlation bounds for polynomials over \{0,1\}, ACM SIGACT News 40 (2009), no. 1, 27–44. 75


