COMP 531

# Advanced Complexity Theory

Lecture Notes

Hamed Hatami

# 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 theory of computation: Turing Machines, Non-determinism, Decidability, Undecidability, and Turing reductions. We will also assume that the reader is familiar with basic complexity theory: the complexity classes $\mathsf{P}$, $\mathsf{NP}$, $\mathsf{coNP}$, the definition of $\mathsf{NP}$ through efficient verifiers, the notion of $\mathsf{NP}$-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 review some of these concepts.

## Decidable and Undecidable Languages:

An alphabet $\Sigma$ is just a finite set, where we think of its elements 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 emphasize 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, naturally: 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 *decidable* if a Turning Machine decides it. The Turing Machines that always halt are called *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 *undecidable*.

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

$$x \in L \iff \exists y, V(x, y) = \textsc{Accept}.$$

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 *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 $\mathsf{NP}$ is the set of languages with efficient verifiers. A canonical problem in $\mathsf{NP}$ is the $\mathsf{SAT}$ 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.

**Definition 1.1.** A language $A$ is *polynomial time mapping reducible*, or *simply polynomial time reducible* to a language $B$, written as $A \leqslant_p B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every $w$,

$$w \in A \iff f(w) \in B.$$

A **Boolean formula** over variables $x_1, \ldots, x_n$ is a formula consisting of variables and logical operators $\wedge, \vee, \neg$ corresponding to AND, OR, and negation, respectively. For example

$$\phi(x_1, x_2, x_3) := (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$$

is a Boolean formula over the variables $x_1, x_2, x_3$, and one can easily check that $\phi$ evaluates to $\textsc{True}$ if and only if majority of the variables $x_1, x_2, x_3$ are $\textsc{True}$. We will identify 1 with $\textsc{True}$ and 0 with $\textsc{False}$; 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 **literals**. For a CNF formula

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

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

For example

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3)$$

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

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

$$\mathsf{SAT} = \{\phi \mid \phi \textit{ is a CNF, and there exists } y \textit{ such that } \phi(y) = \textsc{True}\},$$

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

It is easy to see that $\mathsf{SAT}$ is in $\mathsf{NP}$. To see this, note that a $y$ that satisfies $\phi(y) = \textsc{True}$ 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

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

The interesting and challenging part of the Cook-Levin theorem is the proof that every language in $\mathsf{NP}$ reduces to $\mathsf{SAT}$. 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 $\mathsf{NP}$ 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 $2\mathsf{SAT} \in \mathsf{P}$ where $2\mathsf{SAT}$ is $\mathsf{SAT}$ restricted to CNF's such that every clause is of size exactly 2.

**Exercise 1.4.** Prove that $3\mathsf{SAT}$ is $\mathsf{NP}$-complete by showing that $\mathsf{SAT} \leqslant_p 3\mathsf{SAT}$.

# Chapter 2

# 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. Many decidable problems remain intractable as 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 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 $\mathsf{P}$ and $\mathsf{NP}$, it can become an issue for other notations we will define soon. For example, suppose we define $\mathsf{DTIME}(n)$ 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. 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 different tape alphabets can change the 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

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

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 on 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 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 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 2.1.** Let $T : \mathbb{N} \to \mathbb{N}$ be a function.

- $\mathsf{DTIME}(T(n))$ is the class of languages $L$ that be decided in deterministic time $O(T(n))$.

- $\mathsf{NTIME}(T(n))$ is the class of languages $L$ that be decided in nondeterministic time $O(T(n))$.

- $\mathsf{P} = \bigcup_{k\in\mathbb{N}} \mathsf{DTIME}(n^k)$ is the class of the languages that can be decided in polynomial deterministic time.

- $\mathsf{NP} = \bigcup_{k\in\mathbb{N}} \mathsf{NTIME}(n^k)$ is the class of the languages that can be decided in polynomial nondeterministic time.

- $\mathsf{EXP} = \bigcup_{k\in\mathbb{N}} \mathsf{DTIME}(2^{n^k})$ is the class of the languages that can be decided in exponential deterministic time.

- $\mathsf{NEXP} = \bigcup_{k\in\mathbb{N}} \mathsf{NTIME}(2^{n^k})$ is the class of the languages that can be decided in exponential nondeterministic time.

The most famous question of complexity theory is whether $\mathsf{P} = \mathsf{NP}$. 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 $\mathsf{NP}$-completeness. See the introduction of Aaronson's survey [Aar16] for a discussion on the importance of this question. It is conjectured that $\mathsf{P} \neq \mathsf{NP}$, but nobody knows how to prove this. In fact, the question seems to be well beyond the reach of the currently known techniques in complexity theory and mathematics. Even though we seem very far from solving this question, it plays a central and fruitful role in complexity theory as it provides a direction for the area. 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 with 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 solve a problem $X$ using a 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 $\mathsf{NP}$-complete problems compiled over the few decades since the proof of the Cook-Levin theorem and Karp's subsequent seminal paper [Kar72].

- **Counting arguments:** Sometimes, we can show that there are problems in some class $Y$ but not in $X$ by counting arguments. That is by proving 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 does not belong to a complexity class. For example $\mathsf{P} \neq \mathsf{NP}$ is equivalent to the statement that $\mathsf{SAT} \notin \mathsf{P}$. However, as we shall see, we cannot even show that $\mathsf{SAT}$ does not belong to much smaller classes than $\mathsf{P}$.

- 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 few known results in complexity theory that separate complexity classes.

# Time Hierarchy theorems

The first theorem we prove in this course uses the diagonalization technique to separate $\mathsf{P}$ from $\mathsf{EXP}$. 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 2.2.** 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 is that simple operations such as updating the location of the tape-head of $M$ on its input require time and space $\Omega(\log(n))$.

Let us start with an easier task:

- Input: Description of a Turing Machine $M$;
- Goal: Design a Turing Machine $N$ such that
  - $N$ runs in time $O(2^n)$, and
  - if $M$ is a polynomial time algorithm, then $L(N) \neq L(M)$.

Note that 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 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 $N$

```
Data: On input $w$
Set $n = |w|$, and $t = 2^{n/2}$;
Simulate $M$ on $w$ for at most $t$ steps;
if $M$ halts and accepts $w$ in these $t$ steps then
    REJECT;
else
    ACCEPT;
end
```

By remark 2.2 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 $L(M) \neq L(N)$.

Above, 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 $\mathsf{P} \neq \mathsf{EXP}$, we need to construct a TM $N$ that runs in exponential time and furthermore $L(N) \neq L(M)$ *for every* $M$ that runs in polynomial time. This would show $L = L(N) \in \mathsf{EXP}$ but $L \notin \mathsf{P}$. This is not difficult as there are infinitely many possible inputs $w$. We just need to dedicate an infinite sequence of inputs to each Turing Machine $M$ and use those to make sure that $L(M) \neq L(N)$. We can do this, for example, by looking at the first part of a string $w$, and if it describes a Turing Machine $M$, then dedicating $w$ to $M$. More precisely if $w = \langle M, x\rangle$, where $M$ is a description of a Turing Machine, dedicate such $w$ to $M$.

**Theorem 2.3.** *We have* $\mathsf{P} \neq \mathsf{EXP}$.

*Proof.* Consider the following Turing Machine:

**Algorithm 2:** Turing Machine $N$

```
Data: On input $w = \langle M, x\rangle$, where $M$ is a description of a TM.
Let $n = |w|$, and $t = 2^{n/2}$.
Simulate $M$ on $w = \langle M, x\rangle$ for at most $t$ steps.
if $M$ halts and accepts $w$ in these $t$ steps then
    REJECT;
else
    ACCEPT;
end
```

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 $L(N) \in \mathsf{EXP}$ but $L(N) \notin \mathsf{P}$. □

We can easily generalize Theorem 2.3 to other running times. However one must be careful. In the description of the Turing Machine $N$ in the proof of Theorem 2.3, 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 our setting to functions that are computable efficiently.

**Definition 2.4** (Time-constructible function)**.** A function $T : \mathbb{N} \to \mathbb{N}$ is called *time-constructible* if it is non-decreasing, $T(n) \geqslant n\log(n)$ for every $n$, and there is a Turing Machine that computes $x \to 1^{T(|x|)}$ in time $O(T(|x|))$.

**Theorem 2.5** (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* $\mathsf{DTIME}(T_1(n)) \neq \mathsf{DTIME}(T_2(n))$.

*Proof.* The log factor in the statement is an artifact of Remark 2.2. We leave the details to the reader. □

# Exercises

**Exercise 2.1.** 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 2.2.** Prove Theorem 2.5.

**Exercise 2.3.** 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 $\mathsf{NTIME}(T_1(n)) \neq \mathsf{NTIME}(T_2(n))$.

- 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. (**Hint:** Use a *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 3

# Polynomial Hierarchy

In order to motivate the definition of the polynomial hierarchy, it is useful to rewrite the definitions of $\mathsf{P}$, $\mathsf{NP}$, and $\mathsf{coNP}$ in the following manner:

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

These definitions suggest certain generalizations of these classes:

**Definition 3.1.** 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
$$x \in L \Leftrightarrow \exists y_1 \forall y_2 \exists y_3 \ldots y_k,\ M(x, y_1, \ldots, y_k) = \textsc{Accept}, \tag{3.1}$$
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
$$x \in L \Leftrightarrow \forall y_1 \exists y_2 \forall y_3 \ldots y_k,\ M(x, y_1, \ldots, y_k) = \textsc{Accept}, \tag{3.2}$$
where the quantifiers are alternating

Note that $\mathsf{P} = \Sigma_0 = \Pi_0$, $\mathsf{NP} = \Sigma_1$ and $\mathsf{coNP} = \Pi_1$, and we have the inclusions $\Sigma_k \cup \Pi_k \subseteq \Sigma_{k+1} \cap \Pi_{k+1}$. Note further that $\Pi_k = \text{co}\Sigma_k$ 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 (3.1) holds. Then consider the algorithm $M'$ that runs $M$ and flips its output. That is

$$M'(x, y_1, \ldots, y_k) = \textsc{Accept} \Leftrightarrow M(x, y_1, \ldots, y_k) = \textsc{Reject}.$$

Note that by negating (3.1) we have

$$x \notin L \Leftrightarrow \forall y_1 \exists y_2 \forall y_3 \ldots y_k,\ M'(x, y_1, \ldots, y_k) = \textsc{Accept},$$

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

**Example 3.2.** Define the language Unique SAT (denoted by $\mathsf{!SAT}$) as

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

Note that $\mathsf{!SAT} \in \Sigma_2$. 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

$$\phi \in \mathsf{!SAT} \Leftrightarrow \exists y_1 \forall y_2, \ M(\phi, y_1, y_2) = \textsc{Accept},$$

which shows that $\mathsf{!SAT} \in \Sigma_2$.

Finally, the polynomial hierarchy is defined as

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

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

**Theorem 3.3.** *We have* $\mathsf{PH} \subseteq \mathsf{EXP}$*.*

*Proof.* Since $\mathsf{PH} = \bigcup_{k=1}^{\infty} \Sigma_k$, it suffices to show that for every $k \in \mathbb{N}$, we have $\Sigma_k \subseteq \mathsf{EXP}$. Consider $L \in \Sigma_k$, and let $M$ be a fast algorithm such that (3.1) 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 (3.2) holds. If it holds, we accept $x$. Otherwise, we reject.

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

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

**Proposition 3.4.** *If* $\mathsf{P} = \mathsf{NP}$*, then* $\mathsf{P} = \Sigma_2$*.*

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

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

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

and $M$ runs in polynomial time $p(|x|)$. Since $M$ runs in time $p(|x|)$ we can require further that the strings $y_1$ and $y_2$ are of size at most $p(|x|)$:

$$x \in L \Leftrightarrow \exists y_1 \forall y_2, \ |y_1|, |y_2| \leqslant p(|x|) \text{ and } M(x, y_1, y_2) = \textsc{Accept}.$$

Define

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

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

Now note that

$$x \in L \Leftrightarrow \exists y_1, |y_1| \leqslant p(|x|) \text{ and } (x, y_1) \in \tilde{L}_1,$$

which, together with the above discussion, shows that $L \in \mathsf{NP}$. □

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 3.5.** *If* $\mathsf{P} = \mathsf{NP}$*, then* $\mathsf{P} = \mathsf{PH}$*. More generally if* $\Sigma_k = \Sigma_{k+1}$*, then* $\Sigma_k = \mathsf{PH}$*.*

# Exercises

**Exercise 3.1.** Establish the inclusion $\Sigma_k \cup \Pi_k \subseteq \Sigma_{k+1} \cap \Pi_{k+1}$ for all $k$.

**Exercise 3.2.** Complete the proof of Theorem 3.5.

**Exercise 3.3.** Prove that under polynomial-time (mapping) reductions, $\mathsf{QSAT}_k$ is a complete problem for $\Sigma_k$. Here $\mathsf{QSAT}_k$ is the set of true statements of the form

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

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 $\mathsf{QSAT}_1 = \mathsf{SAT}$.

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

**Exercise 3.4.** Give the pseudo-code of the algorithm described in the proof of Theorem 3.3. What is the space complexity of this algorithm?

**Exercise 3.5.** Let $\mathsf{QSAT}$ 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 $\mathsf{PH} = \cup_{k=1}^{\infty}\Sigma_k$. Can we conclude that $\mathsf{QSAT} \in \mathsf{PH}$? Why?

**Exercise 3.6.** Consider the following problem. We are given an arrangement of an $n \times n$ chess board with chess pieces as input. We are asked "Can white win in at most 10 moves if she starts first?". More precisely, "if white starts, can she always win in at most 10 moves, no matter how black plays?"

Which complexity class in the polynomial hierarchy captures this problem?

**Exercise 3.7.** Show that if $\mathsf{PH}$ does not collapse, that is $\mathsf{PH} \neq \Sigma_k$ for any $k$, then it does not have a complete problem under the polynomial-time reductions.

# Chapter 4

# Space complexity

In the previous lectures, we discussed several *time* complexity classes, deterministic and non-deterministic. Next, we turn our attention to space complexity. Recall that in our computational model, we used a separate tape for the workspace 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 $S : \mathbb{N} \to \mathbb{N}$, where $S(n)$ is the maximum space complexity of $M$ over inputs of size $n$.

**Definition 4.1.** Let $S : \mathbb{N} \to \mathbb{N}$ be a function.

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

Similar to the time hierarchy theorem, one can use a simple diagonalization argument to prove a space hierarchy theorem. As with the time hierarchy theorem, we need to assume that the space function can be computed using a reasonable amount of resources.

**Definition 4.2** (Space-constructible function)**.** A function $S : \mathbb{N} \to \mathbb{N}$ is called *space-constructible* if it is non-decreasing, $S(n) \geqslant \log(n)$, and there is a Turing Machine that computes $x \to 1^{S(|x|)}$ in space $O(S(|x|))$.

**Theorem 4.3** (Space Hierarchy Theorem)**.** *Let* $S_1, S_2 : \mathbb{N} \to \mathbb{N}$ *be such that* $S_2$ *is space-constructible, and* $S_1(n) = o(S_2(n))$*. Then* $\mathsf{SPACE}(S_1(n)) \neq \mathsf{SPACE}(S_2(n))$.

Note that unlike Theorem 2.5, we do not need to multiply $S_1$ by $\log(S_1)$. Indeed as it is mentioned in Remark 2.2, 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

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

## Time and Space

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

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

How large is the class $\mathsf{PSPACE}$? Where does $\mathsf{PSPACE}$ fit in the above picture? Does it contain $\mathsf{P}$? 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

$$\mathsf{DTIME}(T(n)) \subseteq \mathsf{SPACE}(T(n)).$$

How about $\mathsf{NP}$, or $\mathsf{PH}$, or even $\mathsf{EXP}$? We will show that

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

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

First let us establish $\mathsf{PSPACE} \subseteq \mathsf{EXP}$. 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. Suppose the space is bounded by $S(n)$. In that case, 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 *configuration*, 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 return to the same configuration repeatedly and thus will never halt. This is a contradiction as $M$ is an algorithm.

**Theorem 4.4.** *Let $S : \mathbb{N} \to \mathbb{N}$ be a space-constructible function. Then* $\mathsf{SPACE}(S) \subseteq \cup_{k=1}^{\infty} \mathsf{DTIME}(2^{kS})$.

Note that another consequence of Theorem 4.4 is that $\mathsf{L} \subseteq \mathsf{P}$. We conclude

**Theorem 4.5.** *We have*

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}.$$

*Proof.* The inclusions $\mathsf{L} \subseteq \mathsf{P}$, and $\mathsf{PSPACE} \subseteq \mathsf{EXP}$ are immediate from Theorem 4.4. The inclusion $\mathsf{PH} \subseteq \mathsf{PSPACE}$ follows from the proof of Theorem 3.3. It is not difficult to see that the algorithm presented in the proof of Theorem 3.3 uses a polynomial amount of space (See the Exercise 3.4 in the previous lecture). □

**Remark 4.6.** The only known separations in Theorem 4.5 are $\mathsf{L} \neq \mathsf{PSPACE}$ and $\mathsf{P} \neq \mathsf{EXP}$.

Perhaps the most famous example of a language in $\mathsf{PSPACE}$ is $\mathsf{QSAT}$. This language is defined similarly to $\mathsf{SAT}$, 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 $\mathsf{SAT}$ problem, which asks whether there exists a truth assignment that satisfies the CNF formula. However, $\mathsf{QSAT}$ allows using $\forall$ quantifiers for some variables, making the problem more general and probably more difficult.

**Definition 4.7** ($\mathsf{QSAT}$)**.** Let $\mathsf{QSAT}$ 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

$$\exists x_1 \forall x_2 \forall x_3 \ (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \in \mathsf{QSAT},$$

while

$$\forall x_1 \exists x_2 \forall x_3 \ (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \notin \mathsf{QSAT}.$$

It is easy to see that $\mathsf{QSAT}$ is in $\mathsf{PSPACE}$. See Exercise 4.5, where the reader is also asked to prove that $\mathsf{QSAT}$ is $\mathsf{PSPACE}$-complete under polynomial-time reductions.

## Logarithmic Space $\mathsf{L}$ and $\mathsf{NL}$

In the previous section, we saw that $\mathsf{PSPACE}$ 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 $\mathsf{P}$ already belongs to $\mathsf{PSPACE}$, and we deem the languages that are not in $\mathsf{P}$ as intractable for time-related reasons. Hence, when discussing space-efficient algorithms, the class $\mathsf{L}$ is arguably a better choice. It captures the set of problems that can be solved efficiently in terms of space. It is easy to see that the two basic arithmetic operations +, × fall into this class. It is also true, but much harder to prove, that the division ÷ also belongs to $\mathsf{L}$. This has been established in 2001 by Chiu, Davida, and Litow [CDL01]. The class $\mathsf{L}$ 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 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 stream size. As we shall see later in the course, the class $\mathsf{L}$ 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 $X \leqslant_p Y$ and $Y \in \mathsf{L}$, then $X \in \mathsf{L}$. Instead, we need the refiner notion of *Logspace mapping reductions*:

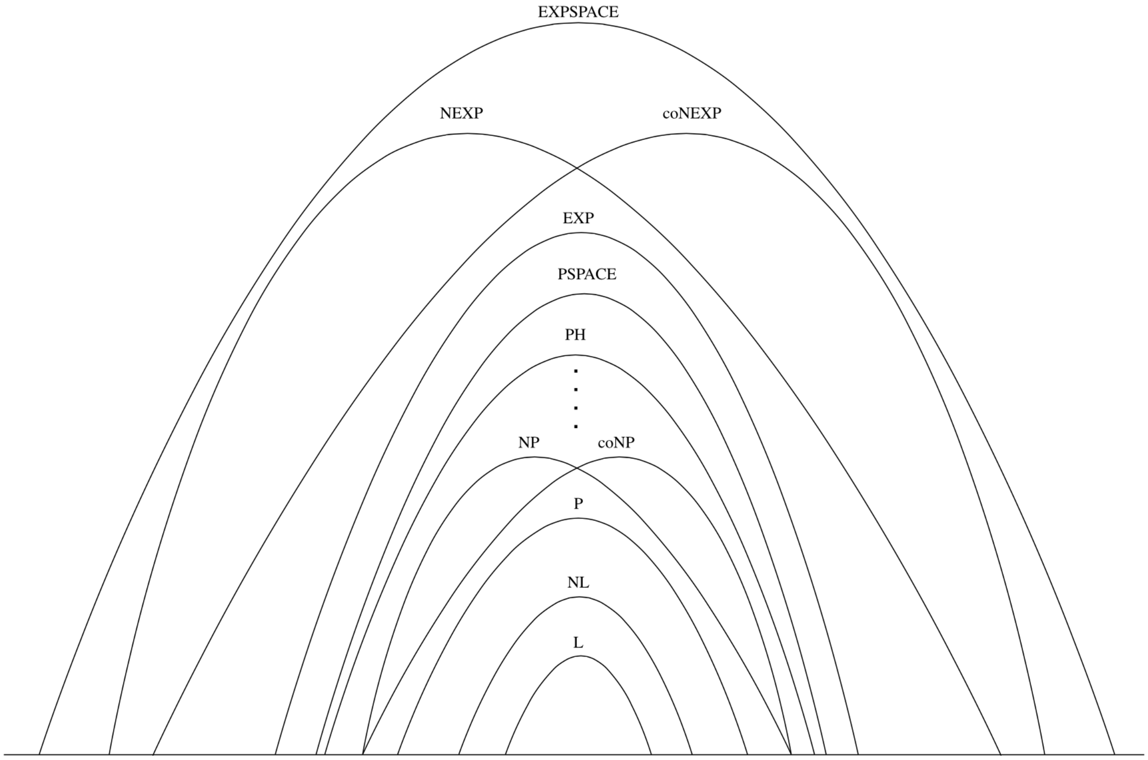

Figure 4.1: A summary of the inclusion relation between some complexity classes

**Definition 4.8.** We say that $X$ is Logspace mapping reducible to $Y$, and write $X \leqslant_\ell Y$ if there exists a map $f : \Sigma^* \to \Sigma^*$ that can be computed in logarithmic space, and it satisfies

$$x \in X \Leftrightarrow f(x) \in Y.$$

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 $X \leqslant_\ell X$ given by $f : w \mapsto w$, we can clearly compute $f$ in logarithmic space, but $f(|w|) = |w| = \omega(\log(|w|))$. This might seem an obstacle to proving that if $X \leqslant_\ell Y$ and $Y \in \mathsf{L}$, then $X \in \mathsf{L}$, 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 4.9.** *If $X \leqslant_\ell Y$ and $Y \in \mathsf{L}$, then $X \in \mathsf{L}$.*

*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 4.4 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. □

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

Figure 4.1 summarizes the inclusion relation between the complexity classes we have seen so far.

## Exercises

**Exercise 4.1.** 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 4.2.** Prove that the following language is in $\mathsf{L}$.

$$\{0^n 1^n : n \in \mathbb{N}\}.$$

**Exercise 4.3.** Prove that the following language is in $\mathsf{L}$.

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

where $w^R$ is the reverse of $w$.

**Exercise 4.4.** Prove that the addition function is in $\mathsf{L}$. That is $f(x, y) = x + y$, where $x$, $y$ and $x + y$ are in binary basis.

**Exercise 4.5.** First, write a pseudo-code which shows $\mathsf{QSAT} \in \mathsf{PSPACE}$. Next, prove that $\mathsf{QSAT}$ is $\mathsf{PSPACE}$-complete under polytime reductions.

# Chapter 5

# Savitch's Theorem

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

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

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 $\in \mathsf{NL}$.

**Theorem 5.1.** *The language* PATH *is in* $\mathsf{NL}$.

*Proof.* To show that PATH $\in \mathsf{NL}$, 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, we accept the input; otherwise, we reject it. Obviously, if there is an $s - t$ path, there will be a sequence of non-deterministic choices that lead to Accept. On the other hand, if there are no $s \to t$ paths, then no matter how we make the non-deterministic choices, the input will 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 $\mathsf{NL}$. The following theorem shows that every language in $\mathsf{NL}$ can be log-space reduced to PATH.

**Theorem 5.2.** *The language* PATH *is* $\mathsf{NL}$*-complete under the log-space reductions.*

*Proof.* We have already shown that PATH $\in \mathsf{NL}$. It remains to prove completeness. Consider an arbitrary $X \in \mathsf{NL}$, 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 to have 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 $s \to t$ 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. To list the nodes, 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, we output $(c, c')$ as an edge and otherwise output it as a non-edge.

□

It is unknown whether PATH is in $\mathsf{L}$ or not. By Theorem 5.2 if PATH $\in \mathsf{L}$, then $\mathsf{NL} = \mathsf{L}$. This is a very interesting question, as unlike the $\mathsf{P}$ versus $\mathsf{NP}$ question, there are fewer reasons to believe that $\mathsf{L}$ and $\mathsf{NL}$ are necessarily different. Towards answering these questions, in 2005, Reingold, in a breakthrough paper [Rei08] showed that the undirected version of PATH belongs to $\mathsf{L}$. That is UPATH $\in \mathsf{L}$, where

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

# Savitch's theorem

In this section, we discuss the relationship between 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 $\mathsf{NP}$-complete problems such as $\mathsf{SAT}$ 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 more efficiently.

**Theorem 5.3** (Savitch's Theorem)**.** *We have*

$$\mathsf{NL} \subseteq \mathsf{L}^2 \stackrel{\text{def}}{=} \mathsf{SPACE}(\log^2(n)).$$

*Proof.* By Theorem 5.2 the statement is equivalent to showing that PATH $\in \mathsf{SPACE}(\log^2(n))$. 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 5.1, 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 $\langle G, s, t\rangle \in$ 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 $s \to w$ path of length at most $m/2$, and a $w \to t$ path of length at most $m/2$. If they both exist, then we know there is an $s \to t$-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 for each level, we need to remember one vertex. So, 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 $u \to v$ path of length at most $k$.

```
Data: On input $w = \langle G, u, v, k\rangle$.
if $k \leqslant 1$ then
    if $uv \in E$ or $u = v$ then
        Accept;
    else
        Reject;
    end
else
    for $w \in V$ do
        if Path$(G, u, w, \lceil k/2\rceil)$ = Accept and Path$(G, w, v, \lceil k/2\rceil)$ = Accept then
            Accept;
        end
    end
    Reject;
end
```

□

**Corollary 5.4.** *We have* $\mathsf{PSPACE} = \mathsf{NPSPACE}$.

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

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

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

What is the running time of the algorithm of Theorem 5.3? Note that the running time corresponds to the recursion $T(m) \leqslant 2mT(m/2)$ 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 a 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 5.5** (Open Problem)**.** *Is there an algorithm for* PATH *that runs in time $n^{O(1)}$ and uses space $\log^{O(1)}(n)$?*

## Exercises

**Exercise 5.1.** Prove that UPATH $\leqslant_\ell$ PATH.

**Exercise 5.2.** 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. In other words, for every cell, a fair coin is flipped independently to decide the cell's value.

Let $\mathsf{C}$ be the class of problems that can be solved with a deterministic *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 $\mathsf{C} = \mathsf{NL}$.

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

**Exercise 5.3.** Recall that a Boolean formula is an expression involving Boolean variables and Boolean operations $\wedge, \vee, \neg$ (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 MinFormula $\in \mathsf{PSPACE}$.
- Prove that if $\mathsf{P} = \mathsf{NP}$, then MinFormula $\in \mathsf{P}$.

**Exercise 5.4.** Prove that the following language is in $\mathsf{PSPACE}$.

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

**Exercise 5.5.** Prove that the set of properly nested parenthesis is in $\mathsf{L}$. For example (()()) is properly nested, while ())( is not.

**Exercise 5.6.** Prove that the following is in $\mathsf{L}$.

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

**Exercise 5.7.** An $\mathsf{NL}$-certifier is an 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 $L \in \mathsf{NL}$ if and only if there is an $\mathsf{NL}$-certifier $M$ such that

$$x \in \mathsf{L} \Leftrightarrow \exists y,\ M(x, y) = \text{Accept}.$$

**Exercise 5.8.** Prove that if in the definition of the $\mathsf{NL}$-certifier if relax the read-once condition on the certificate-tape to an ordinary read-only tape, then we obtain the class $\mathsf{NP}$.

# Chapter 6

# NL versus coNL: Immerman-Szelepcsenyi's Theorem

In the previous lecture, we showed that $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{L}^2$, where $\mathsf{L}^2 = \mathsf{SPACE}(\log^2(n))$. 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-Szelepcsenyi 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 $\text{PATH}^c$ is coNL-complete. Hence, in order to show that NL = coNL, it suffices to prove $\text{PATH}^c \in \mathsf{NL}$. 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 difficultly 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-Szelepcsenyi's Theorem

To present the proof of the Immerman-Szelepcsenyi Theorem, it is useful to think of the NL algorithms in the language of verifiers. We refer the reader to Exercise 5.7, where the reader is asked to prove that $L \in \mathsf{NL}$ if and only if there is an NL-certifier $M$ such that

$$x \in \mathsf{L} \Leftrightarrow \exists y,\ M(x,y) = \textsc{Accept},$$

where an NL-certifier is an 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 Szelepcsenyi [Sze88] independently proved that NL = coNL.

**Theorem 6.1.** *We have* NL = coNL.

*Proof.* By NL-completeness of PATH, it suffices to prove that

$$\text{NonPATH} = \{\langle G, s, t\rangle : G \text{ is a directed graph with } \textit{no } s \to t \text{ paths}\} \in \mathsf{NL}.$$

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

$$A_k = \{u\ :\ \exists su\text{-path of length} \leqslant k\}.$$

Now let us try to build a "certificate" for *not* containing any $st$-paths. For each $k = 0, \dots, n$, define

$$P_k = [B_1^k, \dots, B_n^k],$$

where the blocks are defined as

$$B_i^k = \begin{cases} [\text{an } su_i\text{-path of length} \leqslant k] & u_i \in A_k \\ [\ ] & u_i \notin A_k \end{cases}$$

That is, when $u_i \in A_k$, then the evidence is provided to us, whereas if $u_i \notin A_k$, then $B_i^k$ 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 $c_k$, $k$, $P_k$, $v$. ;                 // Below, only remember $c, k, c_k, i, v, B$
Initialize $c := 0$, and $B := \langle v \notin A_{k+1} \rangle$
for $i = 1, \dots, n$ do
    if $\langle u_i \in A_k \rangle$ according to $P_k$ then
        Verify the claim by following the provided path;     // requires $O(\log(m))$ space
        Increase $c$;
        if $u_i v \in E$ or $v = u_i$ then
            $B := \langle v \in A_{k+1} \rangle$
        end
    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 can only 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:

$$P = [\underbrace{P_0 \dots P_0}_{m \text{ times}}, \underbrace{P_1 \dots P_1}_{m \text{ times}}, \dots, \underbrace{P_m \dots P_m}_{m \text{ times}}].$$

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 $k = 0, \dots, m-1$, 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 $sv$-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 6.1.** Prove that the following is in NL.

$$\{\langle G \rangle \ : \ G \text{ is a bipartite graph}\}.$$

**Exercise 6.2.** Where is the error in the following incorrect proof that NonPATH $\in \mathsf{NL}$?

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 $\langle G, s, t\rangle \in$ 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 $G$:

$$C = [\underbrace{x, x, \ldots, x}_{k \text{ times}}].$$

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 $\langle G, s, t\rangle \in$ NonPATH.

# Chapter 7

# 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]. Curiously, most natural problems in the class NP seem to be either NP-complete or fall into P. A partial explanation for this phenomenon 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 $\mathsf{P} \neq \mathsf{NP}$, then there are problems in NP that are neither NP-complete nor belong to P.

**Theorem 7.1** (Ladner's theorem [Lad75]). *If $\mathsf{P} \neq \mathsf{NP}$, then there is a problem $L \in \mathsf{NP} \setminus \mathsf{P}$ 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 $\mathsf{P} \neq \mathsf{NP}$. In fact, to this day, no such decision problem is known, and there are very few candidates for such problems 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, $\{\langle \phi, 1^{f(|\phi|)}\rangle \mid \phi \in \mathsf{SAT}\}\rangle$ for some carefully chosen relatively large $f(|\phi|)$.

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

We first give 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 $\mathsf{P} \neq \mathsf{NP}$, 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

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

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 $n$.

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

- $L \notin \mathsf{P}$: Note that if $L \in \mathsf{P}$, then given a 3CNF $\phi$, we can decide in time $2^{O(\log^2 n)}$ whether $\phi \in 3\mathsf{SAT}$ 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 $\mathsf{NP}$-complete: If it is, there is a polynomial time reduction from $3\mathsf{SAT}$ to $L$. In particular there exists a polynomial time computable $g : \Sigma^* \to \Sigma^*$ that given a 3CNF $\phi$ of size $n$ produces $\langle \psi, 1^{h(|\psi|)} \rangle = g(\phi)$ such that

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

  Since this is a polynomial-time reduction, we have $|g(\phi)| = n^{O(1)}$, which shows that $\psi$ must be small enough to satisfy $h(|\psi|) \leqslant |g(\phi)| = n^{O(1)}$. Recalling $h(k) = 2^{\log^2 k}$, we conclude $|\psi| \leqslant 2^{\sqrt{\log n}} = o(n)$, which is in fact much smaller than $n$. Note that now to see whether $\phi \in 3\mathsf{SAT}$ we can try all the truth assignments to the much smaller 3CNF $\psi$. This will require time $O(2^{2^{\sqrt{n}}}) = 2^{o(n)}$, 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 $\mathsf{P} \neq \mathsf{NP}$ only tells us that every such algorithm must run in super-polynomial without proving us with any concrete functions.

*Proof of Theorem 7.1.* Note that for every polynomial $p(n)$, there exists a constant $k \in \mathbb{N}$ such that $p(n) \leqslant n^k + k$ for all $n \geqslant 0$. 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 for every language $L \in \mathsf{P}$, there is some $k$ such that $M_k$ decides $L$.

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

$$\mathsf{SAT}_H = \{\langle \phi, 1^{|\phi|^{H(|\phi|)} - |\phi|} \rangle \ : \ \langle \phi \rangle \in \mathsf{SAT}\},$$

so that a CNF $\phi$ is padded to a length $|\phi|^{H(|\phi|)}$ input for $\mathsf{SAT}_H$. Provided that $H$ can be computed efficiently, this blow-up in the input size will make the problem easier than $\mathsf{SAT}$ because the running time is always measured with resepct to the input length. On the other hand, we have to choose $H$ carefully so that $\mathsf{SAT}_H \notin \mathsf{P}$ if $\mathsf{SAT} \notin \mathsf{P}$.

We will define the function $H : \mathbb{N} \to \mathbb{N}$ as in the following. Define $H(n) = 1$ for all $n \leqslant 100$. For $n > 100$, we define $H(n)$ to be the smallest $k < \log \log n$ such that $M_k$ decides $\mathsf{SAT}_H$ 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 $h(1), ..., h(\log(n))$.
  2. For each $k = 1$ to $\log \log(n)$ do
     (a) $flag = True$.
     (b) For every $x$ of length $|x| \leqslant \log(n)$:
         i. Decide whether $x \in \mathsf{SAT}$ by checking that $x = \langle \phi, 1^{|\phi|^{h(|\phi|)} - |\phi|} \rangle$ for some $\phi \in \mathsf{SAT}$
            with $|\phi| \leqslant \log(n)$, 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 $x \in \mathsf{SAT}_H$ then set
             $flag = False$.
     (c) If $flag = True$, then return $k$ and terminate.
  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 7.2.** *$H(n)$ 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 $|\phi| \leqslant \log n$), 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 $O(\log(n))^k) \leqslant O((\log(n)^{\log\log(n)}) = o(n)$. Finally, observing that there are $O(n)$ choices of $x$, with $|x| \leqslant \log n$, we arrive at the recursion $T(n) \leqslant T(1) + \ldots + T(\log(n)) + n^{O(1)}$ which solves to $T(n) = n^{O(1)}$. □

**Claim 7.3.** *We have $\mathsf{SAT}_H \in \mathsf{NP}$.*

*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 $w = \langle \phi, 1^{|\phi|^{H(|\phi|)} - |\phi|}\rangle$, and then use the fact that $\mathsf{SAT} \in \mathsf{NP}$. □

**Claim 7.4.** *If $\mathsf{P} \neq \mathsf{NP}$ then $\mathsf{SAT}_H \notin \mathsf{P}$, and $\lim_{n\to\infty} H(n) = \infty$.*

*Proof.* Suppose for contradiction that $\mathsf{SAT}_H \in \mathsf{P}$. Then there exists a $k < \infty$ such that $M_k$ decides $\mathsf{SAT}_H$ correctly on all inputs. This means that for every $n$ with $\log\log(n) \geqslant k$, we have $H(n) \leqslant k = O(1)$. Then the mapping

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

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

Now suppose for contradiction that $\lim_{n\to\infty} H(n) < \infty$. 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 $\mathsf{SAT}_H$ for all inputs. Since $M_k$ is a polynomial-time algorithm, this contradicts the previous assertion that $\mathsf{SAT}_H \notin \mathsf{P}$. □

**Claim 7.5.** *If $\mathsf{P} \neq \mathsf{NP}$ then $\mathsf{SAT}_H$ is not $\mathsf{NP}$-complete.*

*Proof.* Suppose that $\mathsf{SAT}_H$ is $\mathsf{NP}$-complete. Then $\mathsf{SAT} \leqslant_p \mathsf{SAT}_H$, and hence there is a polynomial-time computable map $f$ such that

$$\phi \in \mathsf{SAT} \iff f(\phi) \in \mathsf{SAT}_H.$$

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

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

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

$$\phi \in \mathsf{SAT} \iff \pi(\phi) \in \mathsf{SAT}.$$

Since $\lim_{n\to\infty} H(n) = \infty$, 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 $\mathsf{SAT}$, which contradicts our assumption $\mathsf{NP} \neq \mathsf{P}$. □

The above claims imply Theorem 7.1. □

# Exercises

**Exercise 7.1.** Prove that if $\mathsf{P} \neq \mathsf{NP}$, then there is an infinite sequence of languages $L_1, L_2, \ldots$ in $\mathsf{NP}$ such that neither of them is in $\mathsf{P}$, and moreover $L_{i+1} \leqslant_p L_i$ but $L_i \not\leqslant_p L_{i+1}$, for all $i$.

# Chapter 8

# 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 to a problem. What a step is 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

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

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 "$\langle M, w\rangle \in \mathrm{HALT}_{\mathrm{TM}}$?" 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 even undecidable problems, such as the halting problem, can be used to define oracle Turing Machines.

**Definition 8.1.** An *oracle* for a language $B$ is a device capable of reporting whether any string $w$ is a member of $B$. An oracle Turing machine $M^B$ is a modified Turing Machine with the additional capability of querying an oracle for $B$. This is implemented by an additional *write-only* query tape and a special state 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 8.2.** 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 8.3.** 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 $A \leqslant_T B$) 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 $A \leqslant_T^p B$) if there is a polynomial time oracle Turing Machine $M^B$ that decides $A$.

Note that if $A \leqslant_T B$, 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 $A \leqslant_T^p B$, and $B \in \mathsf{P}$, then $A \in \mathsf{P}$, 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 can we 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 8.1, 8.2, 8.3 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 8.4** (Semi-formal)**.** Let $\mathsf{C}$ be the class of languages that can be decided by Turing Machines with certain resources. Then $\mathsf{C}^B$ is the class of languages that can be solved by $B$-oracle Turing Machines with the same resources. More generally for a set $\mathcal{B}$ of languages, we define $\mathsf{C}^{\mathcal{B}} = \bigcup_{B \in \mathcal{B}} \mathsf{C}^B$.

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

$$\mathsf{P}^B := \{A : A \leqslant_T^p B\}\,.$$

**Remark 8.5** (Warning)**.** Note that Definition 8.4 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. As a result, if a complexity class has two different definitions (say $\mathsf{NL} = \mathsf{coNP}$), then depending on what definition one chooses, one might end up in two different complexity classes. In other words it is possible that $\mathsf{C} = \mathsf{D}$ while $\mathsf{C}^B \neq \mathsf{D}^B$ for some oracle $B$. See Exercise 8.8.

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

We have $\mathsf{P}^{\mathsf{SAT}} = \mathsf{P}^{\mathsf{NP}}$, and generally, the following proposition is easy to prove.

**Proposition 8.6.** *Suppose that a language $B$ is complete for some class $\mathsf{D}$ under some notion of reduction. Then $\mathsf{C}^B = \mathsf{C}^{\mathsf{D}}$ provided that the machines in the definition of $\mathsf{C}$ can execute that notion of reduction.*

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

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

**Proposition 8.7.** *We have $\mathsf{NP}^{\mathsf{NP}} = \Sigma_2$.*

*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 $B \in \mathsf{NP}$ and furthermore

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

This shows that $L \in \mathsf{NP}^B$, and consequently $\Sigma_2 \subseteq \mathsf{NP}^{\mathsf{NP}}$. We leave proof of the opposite direction as an exercise. See Exercise 8.5.

□

Proposition 8.7 hints at an oracle-based definition of the polynomial hierarchy.

**Definition 8.8** (Oracle Definition of the Polynomial Hierarchy)**.** Define $\Sigma_0 = \Pi_0 = \Delta_0 = \mathsf{P}$, and for every $i \geqslant 0$, let

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

In Exercise 8.6 you are asked to prove that this definition of $\Sigma_i^p$ and $\Pi_i^p$ is consistent with the earlier definition of these classes involving quantifiers and polynomial-time verifiers. Note 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 $M^B$ is countable, and thus for every $B$, some languages are not recognizable by any oracle Turing Machine $M^B$.

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 $M^B$ as well. Hence the same diagonalization argument implies similar hierarchy theorems: e.g. $\mathsf{P}^B \neq \mathsf{EXP}^B$ and $\mathsf{PSPACE}^B \neq \mathsf{EXPSPACE}^B$ 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 $\mathsf{P}^B \neq \mathsf{EXP}^B$. While we can celebrate the broadness of such results, we suspect that, unfortunately, this robustness may be a sign of the 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 $\mathsf{P}$ versus $\mathsf{NP}$ question. Does it relativize? Baker, Gill, and Solovay [BGS75] proved that the answer is negative.

**Theorem 8.9** (Baker, Gill, and Solovay [BGS75])**.** *There exist languages $A$ and $B$ such that $\mathsf{P}^A = \mathsf{NP}^A$, and $\mathsf{P}^B \neq \mathsf{NP}^B$.*

*Proof.* Note that for every $L$, we have $\mathsf{P}^L \subseteq \mathsf{NP}^L$, thus the statement is equivalent to ensuring that $\mathsf{NP}^A \subseteq \mathsf{P}^A$ and $\mathsf{NP}^B \not\subseteq \mathsf{P}^B$.

The first part is easy. We will show that for any $\mathsf{PSPACE}$-complete language $A$ (such as $\mathsf{TQBF}$), we have $\mathsf{P}^A = \mathsf{NP}^A = \mathsf{PSPACE}$. It suffices to show $\mathsf{NP}^A \subseteq \mathsf{PSPACE}$, as the inclusions $\mathsf{PSPACE} \subseteq \mathsf{P}^A$ is obvious from the completeness of $A$. Let $L \in \mathsf{NP}^A$. Then there exists a polynomial time *non-deterministic* oracle Turing Machine $M^A$ that decides $L$. Note that with an additional polynomial amount of space, we can go over all the computational paths of $M^A$, and for each occurring query, run the $\mathsf{PSPACE}$ 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 $\mathsf{NP}^B \not\subseteq \mathsf{P}^B$. To achieve this, we need to guarantee the existence of a language $L_B \in \mathsf{NP}^B$ such that $L_B \notin \mathsf{P}^B$. A good candidate is

$$L_B = \{1^n \mid \exists y \in B, |y| = n\},$$

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 $L_B$ is always in $\mathsf{NP}^B$, as to accept $1^n$, we simply need to non-deterministically guess an $x$ of length $n$, and use the oracle to verify that $x \in B$.

The set $L_B$ is a good candidate for a problem in $\mathsf{NP}^B \backslash \mathsf{P}^B$ because the membership of $1^n$ depends on the membership situation of $2^n$ 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 $M^B$ can decide $L_B$, but any oracle Turing machine deciding $L_B$ must essentially query all the $2^n$ strings of size $n$.

**Claim 8.10.** *There exists a language $B$ such that every deterministic oracle Turing Machine $M^B$ that decides $L_B$ must have a running time of at least $2^n$ on infinitely many inputs $w = 1^n$.*

*Proof.* We will construct $B$ using an enumeration $M_1, M_2, \ldots$ of the descriptions of all oracle Turing Machines (recall Remark 8.3). We will assume that every description is repeated infinitely many times in this enumeration[1]. We will construct $B$ and a sequence of integers $0 < n_1 < n_2 < \ldots$ such that $M_k^B$ cannot correctly decide $1^{n_k} \in L_B$ in time that is less than $2^{n_k}$. Since every Turing Machine appears infinitely many times in the sequence, we conclude that any $M^B$ that does not have running time $\geqslant 2^n$ for infinitely many $n$ will eventually fail on some $n_k$.

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 $w \in \{0,1\}^{n_k}$.*

[1] Using say $1, 1, 2, 1, 2, 3, 1, 2, 3, 4, \ldots$.

In the process, we will keep track of a set $\mathcal{S}$ of all the strings $w$ whose memberships/non-memberships to $B$ are already decided.

Initialize $\mathcal{S} = \emptyset$. For $k = 1, 2, \ldots$, let $n_k = 1 + \max_{w \in \mathcal{S}} |w|$, so that no element in $\{0,1\}^{n_k}$ belongs to $\mathcal{S}$. 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 $w \in \mathcal{S}$, 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 $\mathcal{S}$. 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 $y \notin \mathcal{S} \cup \{y_0\}$ with $|y| \leqslant \max_{w \in S} |w|$ we set $y \notin B$, and update $\mathcal{S}$. 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 $\mathcal{S}$. 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 $\mathcal{S}$. 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 non-relativization of P vs NP mean?** Theorem 8.9 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 8.10 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 $\mathsf{IP}^B \neq \mathsf{PSPACE}^B$. 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 by 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 $\mathsf{IP}^B \neq \mathsf{PSPACE}^B$ 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 8.1.** Prove that if $A \leqslant_m B$, then $A \leqslant_T B$, and if $A \leqslant_p B$, then $A \leqslant_T^p B$.

**Exercise 8.2.** Give an example of languages $A$ and $B$ such that $A \leqslant_T B$ but $A \nleqslant_m B$. Similarly show that if NP $\neq$ coNP, then there are languages $A$ and $B$ such that $A \leqslant_T^p B$ but $A \nleqslant_p B$.

**Exercise 8.3.** Prove that there are languages $A, B \in$ EXP such that $A \leqslant_T^p B$ but $A \nleqslant_p B$. (Hint: Construct $A$ and $B$ such that $n \in A$ if and only if either $2n \in B$ or $2n+1 \in B$.)

**Exercise 8.4.** Prove that $\mathsf{NP} \cup \mathsf{coNP} \subseteq \Delta_2$, and that under the assumption $\mathsf{NP} \neq \mathsf{coNP}$, this inclusion is proper.

**Exercise 8.5.** Prove that $\mathsf{NP}^{\mathsf{NP}} \subseteq \Sigma_2$ by showing that if $L \in \mathsf{NP}^B$ and $B \in \mathsf{NP}$, then $L \in \Sigma_2$.

**Exercise 8.6.** Prove that Definition 8.8 is consistent with the usual definition of the polynomial hierarchy.

**Exercise 8.7.** Does the proof of $\mathsf{NL} = \mathsf{coNL}$ relativize? In other words, is it true that for every $B$ we have $\mathsf{NL}^B = \mathsf{coNL}^B$?

**Exercise 8.8.** Let $A$ be an oracle such that $\mathsf{P}^A = \mathsf{NP}^A$. Prove that there is an oracle such that $\mathsf{P}^{A,B} \neq \mathsf{NP}^{A,B}$.

# Chapter 9

# Randomized Complexity Classes

> The law of averages, if I have got this right, means that if six monkeys were thrown up in the air for long enough they would land on their tails about as often as they would land on their heads.
>
> *Rosencrantz and Guildenstern Are Dead* by Tom Stoppard

A randomized algorithm receives, in addition to its input data, a stream of random bits that it can use to make 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.

Two principal reasons are 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 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, such that 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 input if there is a sequence of choices that leads to halting at the 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 9.1.** A *probabilistic Turing machine* is a type of Turing machine with 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 cell's value.

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 $\{0,1\}^{\mathbb{N}}$. 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, $\perp$, the latter corresponding to an infinite loop. Let us consider an example to demonstrate the power of randomized algorithms.

**Example 9.2** (Matrix Multiplication)**.** Let $L$ be the set of all triples $\langle A, B, C\rangle$, where $A, B, C$ are $n \times n$ matrices with integer entries 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 $AB = C$

```
Data: Three $n \times n$ matrices $A, B, C$
Select $u \in \{0,1\}^n$ uniformly at random;
if $ABu \neq Cu$ then
    Reject
else
    Accept
end
```

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[1] of the vectors $u \in \{0,1\}^n$, we have $Du \neq \mathbf{0}$. 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 a probability of 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 $(1/2)^{1000} \leqslant 10^{-300}$ 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 9.3** (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 $f = g$.

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.

[1] If $D_{ij} \neq 0$, then no matter what the other coordinates of $u = (u_1, \ldots, u_n)$ are, always at least one of the two choices $u_j \in \{0,1\}$ leads to $Du \neq \mathbf{0}$

**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 specify $F$. 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 field size is larger than the degree of $f$ and $g$.

**The inputs:** To study efficient algorithms for PIT, we need to specify how the algorithm is given access to the input polynomials $f$ and $g$. The obvious 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 compare the two polynomials syntactically in linear time (in input length). However, two other natural settings where the problem becomes interesting:

1. We are given $f, g$ as *Arithmetic circuits*: An arithmetic circuit is a directed acyclic graph with a single sink node labelled as the output node, the nodes of in-degree 0 are labelled by input variables $x_1, ..., x_n$ 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. This input form makes the problem more interesting because 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 while a simple linear-sized circuit computes 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 we use to measure the algorithm's efficiency. When $F$ is finite, it is reasonable to demand polynomial-time algorithms in $n \log |F|$. Otherwise, we can ask for a 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 9.4** (Schwarz-Zippel)**.** *Let $f(x_1, ..., x_n)$ 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*

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

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 9.4, 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 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 variation 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. When the field is small, this can be done efficiently. However, over $\mathbb{Z}$, $f(x_1, \ldots, x_n)$ can be as large as $(3 \cdot 2^m)^{2^m}$, which takes exponential time and space to even write down. The way around this issue is to pick a random prime $p \leqslant 2^{2m}$ and do all the arithmetic modules that prime. That is, we compute $f(x_1, \ldots, x_n) \mathrm{mod}\, p$ and compare it to $g(x_1, \ldots, x_n) \mathrm{mod}\, p$, and we reject if they are different. Note that this can be done in 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 $f(x_1, \ldots, x_n) \not\equiv_p g(x_1, \ldots, x_n)$.

The key here is to use the Prime Number Theorem, which guarantees that there are at least $2^{2m}/4m$ prime numbers $p \leqslant 2^{2m}$. Moreover, since $\log |f(x_1, \ldots, x_n) - g(x_1, \ldots, x_n)| \leqslant \log(3 \cdot 2^m)^{2^m} = O(m \cdot 2^m) = o(2^{2m}/4m)$, 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 $1 - o(1)$, $f(x_1, \ldots, x_n) \not\equiv_p g(x_1, \ldots, x_n)$.

## 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 RP

The complexity class 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 $\geqslant 1/2$.

In particular, the only type of error allowed is 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 accepts and rejects, we also consider the class 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 coRP as both the algorithms make no error when $x \in L$, and make an error with probability at most $1/2$ when $x \notin L$.

**Error-reduction in RP:** The constant $1/2$ in the definition of 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 RP with a milder requirement of

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

**How large is 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 NP, we are interested in the existence of advice that leads to acceptance, whereas, in the probabilistic classes such as RP, we demand certain bounds on the probability that random advice leads to a correct answer. In other words, NP can be thought of as a probabilistic class: $L \in \mathsf{NP}$ 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 $\mathsf{RP}$ is contained in $\mathsf{NP}$, that is, nondeterminism is at least as powerful as one-sided error randomness.

**Theorem 9.5.** *We have* $\mathsf{RP} \subseteq \mathsf{NP}$, *and* $\mathsf{coRP} \subseteq \mathsf{coNP}$.

*Proof.* It suffices to prove $\mathsf{RP} \subseteq \mathsf{NP}$. Consider $L \in \mathsf{RP}$. 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 $\geqslant 1/2$.

Then obviously

$$x \in L \Longleftrightarrow \exists r \quad M_r(x) = \text{Accept},$$

which establishes $L \in \mathsf{NP}$. □

## The complexity class ZPP

The complexity class $\mathsf{ZPP}$ (*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 $x$,
$$\Pr_R[M_R(x) = \text{Do Not Know}] \leqslant \frac{1}{2}.$$
- 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

$$\Pr_R[\text{Do Not Know}] \leqslant \left(\frac{1}{2}\right)^k = 2^{-n^{O(1)}}.$$

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

$$\Pr_R[M_R(x) = \text{Do Not Know}] \leqslant 1 - n^{-c},$$

for any fixed constant $c > 0$. Repeating this $k = n^c$ would have resulted[2] to

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

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

**Theorem 9.6.** *We have* $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$.

*Proof.* First consider $L \in \mathsf{ZPP}$. Then there is a polynomial time probabilistic Turing Machine $M_R$ such that it always outputs Accept, Reject, or Do Not Know, such that for every input

$$\Pr_R[M_R = \text{Do Not Know}] \leqslant \frac{1}{2}.$$

To convert this to an $\mathsf{RP}$ algorithm, we can simply output Reject if $M_R(x) \in \{\text{Reject}, \text{Do Not Know}\}$, and otherwise output 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 accept and reject gives shows similarly $\mathsf{ZPP} \subseteq \mathsf{coRP}$.

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

[2] Using the inequality $1 + x \leqslant e^x$ that holds for all $x$.

- If $x \notin L$, then $\mathbf{Pr}[M_R(x) = \textsc{Reject}] = 1$, and $\mathbf{Pr}[N_R(x) = \textsc{Reject}] \geqslant 1/2$.
- If $x \in L$, then $\mathbf{Pr}[M_R(x) = \textsc{Accept}] \geqslant 1/2$, and $\mathbf{Pr}[N_R(x) = \textsc{Accept}] = 1$.

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

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

It is straightforward to see that this is a ZPP algorithm. □

## The complexity class BPP

The complexity class 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 $\leqslant 1/3$.

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

**Example 9.7.** Define $L$ as the language consisting of triples of polynomials $(p_1, p_2, p_3)$ such that exactly two of the polynomials are identical. We leave it as an exercise to show that $L \in \mathsf{BPP}$. It is not known whether $L$ is in RP or 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 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 $M_R(x)$ outputs accept. If $X \geqslant k/2$ then accept and otherwise reject.

Every single run of the algorithm is biased towards the correct answer, and hence one expects (by the law of large numbers) that the majority of outputs are correct with a large probability. 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 $\mathbb{E}[X_i] \geqslant \frac{1}{2} + \varepsilon$, and hence $\mu := \mathbb{E}[\sum X_i] \geqslant \frac{k}{2} + \varepsilon k$. Note that our majority count algorithm makes an error if

$$\sum X_i < \frac{k}{2} \leqslant \mu(1-\varepsilon).$$

Applying Chernoff bound[3] we obtain

$$\mathbf{Pr}[\text{Error}] \leqslant \mathbf{Pr}\left[\sum X_i \leqslant (1-\varepsilon)\mu\right] \leqslant e^{-\frac{\mu\varepsilon^2}{2}} \leqslant e^{-\frac{k\varepsilon^2}{4}}.$$

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

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

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

[3] Chernoff bound says that if $X$ is the sum of $k$ i.i.d. Bernoulli random variables, then denoting $\mu = \mathbb{E}[X]$, for every $\delta > 0$, we have

$$\mathbf{Pr}[X < \mu(1-\delta)] \leqslant e^{-\frac{\delta^2\mu}{2}} \text{ and } \mathbf{Pr}[X > \mu(1+\delta)] \leqslant e^{-\frac{\delta^2\mu}{2+\delta}}.$$

**How large is $\mathsf{BPP}$?** 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 9.8.** *We have* $\mathsf{BPP} \subseteq \Sigma_2 \cap \Pi_2$.

*Proof.* Since $\mathsf{BPP}$ is closed under taking complements, it suffices to show that $\mathsf{BPP} \subset \Sigma_2$. Let $L \in \mathsf{BPP}$. By definition, there is a polynomial time probabilistic algorithm that correctly decides every membership query with a probability of 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 $|A_x| \geqslant 2^\ell \left(1 - \frac{1}{2^n}\right)$;
- If $x \notin L$, then $|A_x| < 2^\ell \left(\frac{1}{2^n}\right)$;

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. We will only need such strong bounds for the case $x \notin L$. These bounds will allow us to extract some structure about $A_x$ to describe $L$ as a $\Sigma_2$ language.

For any subset $A \subseteq \{0,1\}^\ell$, and any $t \in \{0,1\}^r$, 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 $|A| \leqslant 2^{-n}2^\ell$, 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. How well can we do in the other case where $A_x$ is almost everything?

**Claim 9.9.** *If $A$ is a subset of $\{0,1\}^\ell$ with $|A| \geqslant \left(\frac{2}{3}\right) 2^\ell$, then there are $t_1, \dots, t_\ell \in \{0,1\}^\ell$ such that $\{0,1\}^\ell = \bigcup_{i=1}^\ell A \oplus t_i$.*

*Proof.* Pick $t_1, \dots, t_\ell$ 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 $\mathbf{Pr}[y_0 \in A \oplus t] = |A|/2^\ell$. Hence the probability that $y_0$ is not covered by any of our random shifts is bounded by

$$\mathbf{Pr}\left[y_0 \notin \bigcup_{i=1}^{\ell} A \oplus t_i\right] = \left(1 - \frac{|A|}{2^\ell}\right)^\ell \leqslant (1/3)^\ell.$$

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

$$\mathbf{Pr}\left[\exists y \notin \bigcup_{i=1}^{\ell} A \oplus t_i\right] \leqslant 2^\ell (1/3)^\ell < 1.$$

Since this probability is less than 1, there must be at least one choice of shifts such that a $y \notin \bigcup_{i=1}^\ell A \oplus t_i$ does not exist. □

Since $\ell = p(n)$ is a polynomial, there exists a constant $n_0$ such that for every $n \geqslant n_0$, $2^n > \ell$. From the above discussion we conclude that for every $x$ with $|x| \geqslant n_0$, the set $\{0,1\}^\ell$ can be covered with $\ell$ shifts of $A_x$ if and only if $x \in L$. Hence

$$x \in L \Longleftrightarrow \exists t_1, \dots t_\ell \in \{0,1\}^\ell \ \forall r \in \{0,1\}^\ell, \ \bigvee_{i=1}^{\ell} M_{r \oplus t_i}(x) = \text{Accept}.$$

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

$$(x, [t_1, \dots, t_\ell], r),$$

if $|x| \geqslant n_0$, runs $M_{r \oplus t_i}$ on $x$ for $i = 1, \dots, \ell$, 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 polynomial in $|x|$, and

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

□

Theorem 9.8 shows that BPP is contained in the second level of the polynomial hierarchy. It is not known whether BPP $\subseteq$ NP, but it is conjectured that BPP = P. One reason to believe this conjecture was given by the "hardness vs randomness" line of work that, in essence, shows that the existence of an explicit difficult function implies P = BPP. To be more precise, as an example, Impagliazzo and Wigderson [IW97] proved that P = BPP if E requires exponential size *circuits* (see Chapter 10 and Conjecture 10.6), where 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 were known to be in BPP but were not known to be in 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 P.

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

## The complexity class 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 PP refers to *probabilistic polynomial time.* As discussed above, an error probability of 1/2 can be achieved by outputting ACCEPT/REJECT by tossing a fair coin. The condition in the definition of PP asks for beating this obvious bound.

Since every 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 $\frac{1}{2} + 2^{-n^c}$ 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 BPP, where the error reduction argument allowed us to assume the probability of correctness of $\frac{1}{2} + n^{-O(1)}$.

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

**Theorem 9.10.** *The class* 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* $\mathbf{Pr}[\text{ACCEPT}] > \frac{1}{2}$.
- *If $x \notin L$, then* $\mathbf{Pr}[\text{REJECT}] \geqslant \frac{1}{2}$.

*Proof.* See Exercise 9.3 □

**Example 9.11.** 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 MAJSAT $\in$ PP. We can pick a truth assignment uniformly at random, accept if it satisfies $\phi$ and reject otherwise. Note that if $\phi \in$ MAJSAT, then it will be accepted with probability $> \frac{1}{2}$, and if $\phi \notin$ MAJSAT, then it will be rejected with probability $\leqslant \frac{1}{2}$. Hence by Theorem 9.10 MAJSAT is in PP. In Exercise 9.4 you are asked to prove that MAJSAT is in fact PP-complete under polynomial-time many-one reductions.

**How large is PP?** With Theorem 9.6, and a glance at the definitions, we see

$$\mathsf{P} \subseteq \mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP} \subseteq \mathsf{RP} \cup \mathsf{coRP} \subseteq \mathsf{BPP} \subseteq \mathsf{PP}.$$

How large is PP, the largest of these classes?

**Theorem 9.12.** *We have* NP $\subseteq$ PP $\subseteq$ PSPACE.

*Proof.* First we show PP $\subseteq$ PSPACE. Consider a 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 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 runs that have resulted in ACCEPT. This will require a space of $O(p(n))$. By the definition of $\mathsf{PP}$, if $x \in L$, then this count must be strictly larger than $\left|\{0,1\}^{p(n)}\right|/2 = 2^{p(n)-1}$, and if $x \notin L$, then it must be strictly less than $2^{p(n)-1}$. Hence we can deduce whether $x \in L$ is from the count.

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

$$x \in L \Longleftrightarrow \exists y V(x,y) = \textsc{Accept}.$$

Without loss of generality we can assume $|y| = p(n)$. We construct a $\mathsf{PP}$ 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 $V(x,y) = \textsc{Accept}$. Hence in this case

$$\mathbf{Pr}[\textsc{Accept}] = 2^{-p(n)} + \frac{1}{2}(1 - 2^{-p(n)}) \geqslant \frac{1}{2} + 2^{-p(n)-1} > \frac{1}{2}.$$

On the other hand, if $x \notin L$, then

$$\mathbf{Pr}[\textsc{Reject}] \geqslant \frac{1}{2}.$$

The proof now follows from Theorem 9.10.

□

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

**Theorem 9.13** (Toda's theorem)**.** *We have* $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$.

Since MAJSAT is $\mathsf{PP}$-complete (see Exercise 9.4), this theorem is equivalent to the statement that given oracle access to MAJSAT, one can solve any problem in $\mathsf{PH}$ in polynomial time. In Exercise 9.6 shows that if $\mathsf{PP} \subseteq \mathsf{PH}$, then the polynomial hierarchy collapses.

## Exercises

**Exercise 9.1.** Prove that $\mathsf{ZPP}$ can be alternatively defined as the class of languages for which a probabilistic Turing machine exists that with probability 1 returns the correct $\textsc{Accept}/\textsc{Reject}$ answer, and furthermore for every input $x$, the running time is polynomial in *expectation* for every input.

- For every $x$,
$$\mathop{\mathbf{Pr}}_{R}[M_R(x) \text{ is correct}] = 1.$$
- There is a polynomial such that for every $x$,
$$\mathbb{E}_R[\text{Running time of } M_R \text{ on } x] \leqslant p(|x|).$$

**Exercise 9.2.** Prove that the language in Example 9.7 is in $\mathsf{BPP}$.

**Exercise 9.3.** Prove Theorem 9.10.

**Exercise 9.4.** Prove that MAJSAT is $\mathsf{PP}$-complete under polynomial time mapping reductions.

**Exercise 9.5.** Prove that

$$\mathsf{NP} \subseteq \mathsf{BPP} \Longleftrightarrow \mathsf{NP} = \mathsf{RP}.$$

**Exercise 9.6.** Prove that if MAJSAT $\in \mathsf{PH}$ then there is some $k$ such that $\mathsf{PH} \subseteq \Sigma_k$.

# Chapter 10

# 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 labelled 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 labelled 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-in*s and *fan-out*s. Most standard circuits are restricted to have gates $\neg$, $\wedge$, and $\vee$, where $\wedge$ and $\vee$ 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 10.1.** Figure 10.1 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 counted towards the circuit's size.

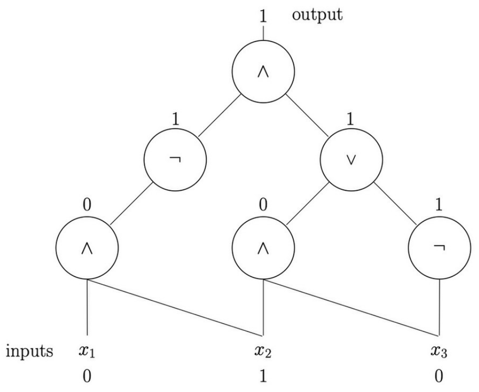

Figure 10.1: A circuit with 3 inputs and six gates.

It is sometimes useful to study sets of gates other than ¬, ∧, and ∨. 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 $B$.

- $\{\neg, \wedge, \vee\}$ is a universal basis, where ∧ and ∨ are binary AND and OR functions. To see this recall that every function $f : \{0,1\}^n \to \{0,1\}$ can be expressed as a DNF:

  $$f(x) = \bigvee_{y:f(y)=1} \left( \left( \bigwedge_{i:y_i=1} x_i \right) \wedge \left( \bigwedge_{i:y_i=0} \overline{x_i} \right) \right).$$

  The right-hand side can be broken further to use only binary ∧ and ∨'s.

- For every $k \geqslant 2$, the set $\mathcal{B}_k$ of all $2^{2^k}$ functions $g : \{0,1\}^k \to \{0,1\}$ is a universal basis.

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

- $\{\wedge, \vee\}$ 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.

- $\{\wedge, \oplus\}$ is a universal basis where $\oplus$ is the binary XOR function.

**Definition 10.2.** Let $\mathcal{B}$ be a basis. The corresponding circuit complexity of a function $f : \{0,1\}^n \to \{0,1\}$, denoted by $\text{size}_{\mathcal{B}}(f)$, is the size of the smallest circuit with gates from $\mathcal{B}$ that computes $f$. If such a circuit does not exist, then we define $\text{size}_{\mathcal{B}}(f)$ to be infinity.

One important feature of Definition 10.2 is that as long as *finite* universal bases 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 $\mathcal{B}$ and $\mathcal{B}'$ are two finite universal bases, then since we can replace the gates in $\mathcal{B}'$ with $\mathcal{B}$-circuits and vice versa, there are constants $c_1, c_2 > 0$ such that

$$\text{size}_{\mathcal{B}}(f) \leqslant c_1 \text{size}_{\mathcal{B}'}(f) \leqslant c_2 \text{size}_{\mathcal{B}}(f),$$

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 $\mathcal{B}_2$, 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 10.3.** *Let $M$ be a deterministic Turing Machine with a polynomial running time that decides a language $L$. There is a* poly$(n)$*-time algorithm that given $n$, outputs a $\mathcal{B}_2$-circuit $\phi_n$ such that for every $x$ with $|x| = n$,*

$$x \in L \Longleftrightarrow \phi_n(x) = 1.$$

Proposition 10.3 gives an alternative equivalent definition for the class $\mathsf{P}$.

**Definition 10.4** (Alternative Definition of $\mathsf{P}$)**.** A language $L$ is in $\mathsf{P}$ if and only if there is a poly$(n)$-time algorithm $M_L$ that given $n$, outputs a $\mathcal{B}_2$-circuit $\phi_n$ (of size poly$(n)$) such that for every $x$ with $|x| = n$,

$$x \in L \Longleftrightarrow \phi_n(x) = 1.$$

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 $\mathsf{P}$ called $\mathsf{P/Poly}$.

**Definition 10.5.** $\mathsf{P/Poly}$ is the class of languages $L$ such that for every $n$, there is a $\mathcal{B}_2$-circuit $\phi_n$ of size poly$(n)$ such that for every $x$ with $|x| = n$,

$$x \in L \Longleftrightarrow \phi_n(x) = 1.$$

Of course, $\mathsf{P} \subseteq \mathsf{P/Poly}$, and one immediately wonders whether the two classes are the same. The answer is a strong "No!"; in fact, $\mathsf{P/Poly}$ contains some languages that are not even Turing recognizable. For example, every language over the unary alphabet $\Sigma = \{1\}$ belongs to $\mathsf{P/Poly}$, 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 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 $\mathsf{EXP}$/Exponential, analogous to $\mathsf{P/Poly}$, 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 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 $\mathsf{P} \neq \mathsf{NP}$ , which is equivalent to the statement $\mathsf{SAT} \notin \mathsf{P}$. The reason we believe that $\mathsf{SAT} \notin \mathsf{P}$ 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 believe that for every large $n$ it is not possible to solve $\mathsf{SAT}$ with polynomially many elementary operations. In other words we believe that $\mathsf{SAT} \notin \mathsf{P/Poly}$. Obviously if we succeed in proving $\mathsf{SAT} \notin \mathsf{P/Poly}$, then we would establish $\mathsf{P} \neq \mathsf{NP}$. This allows us to focus on the circuit complexity of $\mathsf{SAT}$, a simpler object to study than the time complexity.

**Conjecture 10.6.** $\mathsf{SAT}$ *has circuit complexity* $2^{\Omega(n)}$.

# Karp-Lipton Theorem

In the previous section, we discussed that it is reasonable to believe that $\mathsf{SAT} \notin \mathsf{P/Poly}$, which would imply $\mathsf{P} \neq \mathsf{NP}$. However, it is unclear from that discussion whether the statement $\mathsf{SAT} \in \mathsf{P/Poly}$ would have any unexpected implications. Is it possible that $\mathsf{SAT} \in \mathsf{P/Poly}$, and yet $\mathsf{P} \neq \mathsf{NP}$? As far as it is known, this is a possibility. However, Karp and Lipton [KL80] showed that if $\mathsf{SAT} \in \mathsf{P/Poly}$, then the polynomial hierarchy collapses. The original proof of Karp and Lipton collapses $\mathsf{PH}$ to $\Sigma_3$, but Sipser improved it to $\Sigma_2$ in the following theorem.

**Theorem 10.7** (Karp-Lipton-Sipser)**.** *If* $\mathsf{SAT} \in \mathsf{P/Poly}$*, or equivalently* $\mathsf{NP} \subseteq \mathsf{P/Poly}$*, then* $\mathsf{PH} = \Sigma_2$.

*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, we use $A$ again to see if $\phi|_{x_1=0}$ is still satisfactory. 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 $\mathsf{SAT} \in \mathsf{P/Poly}$, 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 $\mathsf{PH} = \Sigma_2$ (why?).

Consider the $\Pi_2$-complete problem of $L = \forall\exists\mathsf{SAT}$. Here, for CNF formulas $\phi(y, z)$ with two sets of variables $y_i$s and $z_i$s,

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

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

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

This shows $L \in \Sigma_2$. □

# Shannon-Muller Lower bound

We saw that to prove $\mathsf{P} \neq \mathsf{NP}$ it suffices to prove a super-polynomial circuit lower bound for $\mathsf{SAT}$. 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 $f : \{0,1\}^n \to \{0,1\}$ compared to the number of small circuits.

**Theorem 10.8** (Shannon)**.** *Almost every Boolean function $f : \{0,1\}^n \to \{0,1\}$ requires $\mathcal{B}_2$ circuits of size $2^n/20n$.*

*Proof.* There are exactly $2^{2^n}$ Boolean functions $f : \{0,1\}^n \to \{0,1\}$. The number of circuits with $t$ gates can be upper-bounded as follows: Since $|\mathcal{B}_2| = 16$, there are $16^t$ choices assigning gates to nodes. There 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 $\mathcal{B}_2$ is at most

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

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

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

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

On the other hand, we know from the DNF representation that every function $f : \{0,1\}^n \to \{0,1\}$ 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 10.8.

Theorem 10.8 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 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 $\mathsf{NP}$).

**Problem 10.9.** *Find an explicit function $f : \{0,1\}^n \to \{0,1\}$ with circuit complexity $\omega(n)$.*

# Adleman's theorem and advice: $\mathsf{BPP} \subseteq \mathsf{P/Poly}$

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

**Theorem 10.10** (Adleman [Adl78, BG81])**.** $\mathsf{BPP} \subseteq \mathsf{P/Poly}$.

*Proof.* Suppose $L \in \mathsf{BPP}$. Consider any $k \in \mathbb{N}$. By applying the error reduction for $\mathsf{BPP}$, we can obtain a polynomial-time randomized algorithm $M_R^k$ such that for every $x$,

$$\mathop{\mathbf{Pr}}_R[M_R^k(x) \text{ is incorrect}] < \frac{1}{2^k}.$$

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

$$\mathop{\mathbf{Pr}}_R[\exists x,\ |x| = n \text{ and } M_R^n(x) \text{ is incorrect}] < 1.$$

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

We can convert $M_r^n$ to a polynomial-size circuit $\phi_n$. Hence $L \in \mathsf{P/Poly}$. □

The above proof suggests a different definition of $\mathsf{P/Poly}$ based on "advice" strings.

**Definition 10.11** (Alternative definition of $\mathsf{P/Poly}$)**.** A language $L$ is in $\mathsf{P/Poly}$ 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 $x$,

$$x \in L \Longleftrightarrow M(r_{|x|}, x) = \textsc{Accept}.$$

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 $\mathsf{P/Poly}$, 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 10.1.** Prove that the 2-bit NAND is a universal basis.

**Exercise 10.2.** Prove that the following language is $\mathsf{P}$-complete with respect to $\leqslant_\ell$ reductions:

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

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

**Exercise 10.3.** Suppose that 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

$$\mathop{\mathbf{Pr}}_{R\sim S}[M_R(x) \text{ is incorrect}] \leqslant \varepsilon + \delta,$$

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

**Exercise 10.4.** Prove that for every constant $k$, we have $\Sigma_3 \not\subseteq \mathsf{Size}(n^k)$, where $\mathsf{Size}(n^k)$ is the set of languages with circuit complexity at most $O(n^k)$.

**Exercise 10.5.** Let $\mathcal{C} \subseteq \{0,1\}^n$, and $f : \mathcal{C} \to \{0,1\}$ be known to us. In the terminology of machine learning, $\mathcal{C}$ is called a concept class, and $f(x)$ is called the label of the concept $x \in \mathcal{C}$. The goal is to determine the label of an unknown $x$ by making the smallest possible 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 $\mathbf{sign}(t - \sum_{i=1}^n w_i x_i)$. Consider a query model $Q$ (e.g., the threshold query model).

The deterministic query complexity of $\mathcal{C}$, denoted by $\mathrm{dt}^Q(\mathcal{C})$, is the smallest $k$ such that the label of every point $x \in \mathcal{C}$ 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 11.10).

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

$$\mathrm{dt}^Q(\mathcal{C}) = O(\mathrm{rdt}^Q(\mathcal{C}) \log |\mathcal{C}|).$$

# Chapter 11

# $\mathsf{AC}^0$: 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 $\mathsf{NC}$ family is defined by such circuits, with $\wedge$, $\vee$ gates of fan-in 2.

**Definition 11.1** (Nick's class ($\mathsf{NC}$))**.** For every $i \geqslant 0$, $\mathsf{NC}^i$ 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 $\mathsf{NC} = \cup_{i=0}^{\infty} \mathsf{NC}^i$.

Since the 2-bit gates $\wedge$ and $\vee$ together with $\neg$ form a universal basis, we could have defined the classes $\mathsf{NC}^i$ equivalently by allowing only these gates.

Due to the bound on fan-in of the gates, $\mathsf{NC}^0$ 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 11.2** ($\mathsf{AC}$)**.** For every $i \geqslant 0$, $\mathsf{AC}^i$ consists of languages $L \subseteq \{0,1\}^*$ that can be decided by polynomial-sized circuit families of depth $O(\log^i n)$ with $\wedge$, $\vee$ gates of arbitrary fan-in and $\neg$ gates.

Define $\mathsf{AC} = \cup_{i=0}^{\infty} \mathsf{AC}^i$.

The name $\mathsf{AC}$ was inspired by $\mathsf{NC}$, with the 'A' standing for "alternating" as will be clear later. It is easy to see

$$\mathsf{NC}^0 \subseteq \mathsf{AC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \cdots .$$

Moreover, since each $\wedge$ and $\vee$ gate of polynomial fan-in can be simulated by a simple $\mathsf{NC}^1$ circuit, for every $i$ we have

$$\mathsf{NC}^i \subseteq \mathsf{AC}^{i+1},$$

and thus

$$\mathsf{AC} = \mathsf{NC} \subseteq \mathsf{P/Poly}.$$

Unfortunately, our understanding of circuit lower bounds does not even extend to $\mathsf{NC}^1$. We do not know any explicit function that is not provably in $\mathsf{NC}^1$. In fact, even a super linear lower bound for $\mathsf{NC}^1$ is considered a major open problem in circuit complexity.

**Problem 11.3.** *Give an explicit function that requires size $\omega(n)$ for circuits of depth $O(\log(n))$ and fan-in 2.*

## Lower-bounds for $\mathsf{AC}$ circuits

We are interested in proving lower bounds for bounded-depth circuits. Since $\mathsf{NC}^0$ circuits only depend on a constant portion of their input variables, it is easy to prove lower-bounds for $\mathsf{NC}^0$. In particular, any function that depends on all its input variables is not computable by $\mathsf{NC}^0$ circuits. For example simple functions such as $x_1 \wedge x_2 \wedge \cdots \wedge x_n$ do not have constant-depth $\mathsf{NC}^0$ circuits. On the other hand, $\mathsf{AC}^0$ 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 about the structure of the $\mathsf{AC}^0$ circuits.

1. Note that by De Morgan's laws
$$\neg(p_1 \vee \ldots \vee p_k) = (\neg p_1) \wedge \ldots \wedge (\neg p_k),$$
and
$$\neg(p_1 \wedge \ldots \wedge p_k) = (\neg p_1) \vee \ldots \vee (\neg p_k).$$
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 $\wedge$ and $\vee$ gates form alternating levels with edges only between each level and the immediate level above it. To achieve the latter, we can add the dummy $\wedge$ or $\vee$ 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 will at most, increase the size by a power of the depth of the circuit (see Exercise 11.1).

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 11.4.** 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 $\mathsf{AC}^i$ for any $i \geqslant 0$.

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 11.5** (Conjunctive Normal Form, $\wedge$ of $\vee$)**.** A formula is in conjunctive normal form, abbreviated to CNF, if it is a conjunction (i.e. $\vee$) of clauses, where a clause is a disjunction (i.e $\vee$) 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 $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$ is a formula in conjunctive normal form. It corresponds to an alternating circuit of depth 2 with 3 gates.

**Definition 11.6** (Disjunctive Normal Form, $\vee$ of $\wedge$)**.** A formula is in disjunctive normal form, abbreviated to DNF, if it is a disjunction (i.e. $\vee$) of conjunctive clauses (i.e $\vee$ of literals). A $t$-DNF is a DNF, where each clause consists of at most $t$ literals.

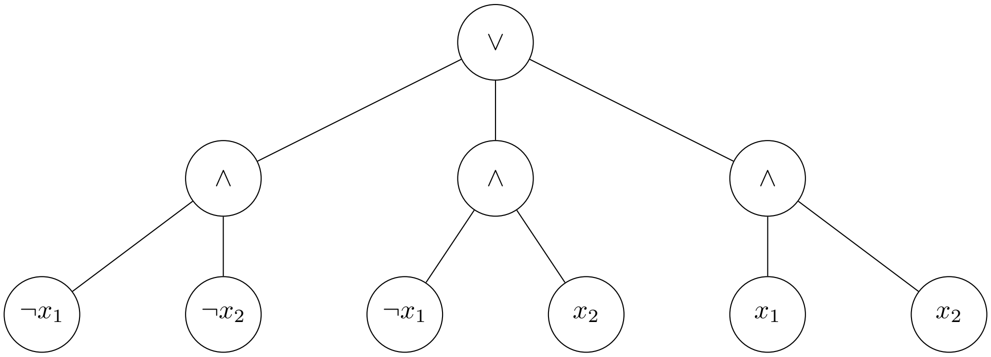

Figure 11.1: Depth-2 alternating circuit satisfying (1),(2), and (3) for DNF $(\neg x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \vee (x_1 \wedge x_2)$

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
$$\left(\bigwedge_{i \in T} x_i\right) \wedge \left(\bigwedge_{i \notin T} \neg x_i\right)$$

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 $\vee$ of these clauses, we create a DNF formula that represents the function $f$.

**Example 11.7.** 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

$$(\neg x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \vee (x_1 \wedge x_2)$$

from Fig. 11.1 represents $f$.

By changing the role of 0's and 1's and $\wedge$ and $\vee$, 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 11.8.** *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 $\mathsf{AC}^0$ is one of the few complexity classes that are relatively well understood. We will show that the parity function $\textsc{Parity} : \{0,1\}^n \to \{0,1\}$ defined as

$$\textsc{Parity}(x) = x_1 \oplus \ldots \oplus x_n,$$

or equivalently

$$\textsc{Parity}(x) = x_1 + \ldots + x_n \bmod 2,$$

does not belong to $\mathsf{AC}^0$. 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 $\textsc{Parity} \notin \mathsf{AC}^0$ 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 Håstad [Has86] further strengthened and simplified this argument and obtained near-optimal exponential lower bounds.

**Theorem 11.9** ([Has86])**.** *Any depth $d$ alternating circuit that computes* $\textsc{Parity}$ *is of size $2^{\Omega(n^{1/d})}$.*

**Proof strategy:** Consider an $\mathsf{AC}^0$ circuit of depth $d = O(1)$, and let us assume that the first level (i.e., the gates directly connected to inputs) consists of $\wedge$ gates, the second level consists of the $\vee$ 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 $\mathsf{AC}^0$ circuit greatly. Consider one of the $\wedge$ 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 $\wedge$ gate only needs one 0 input to be set to 0. Note that the bottom two circuit levels are a collection of DNF's ($\vee$ of $\wedge$'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. We apply a second random restriction, setting values to more randomly chosen variables. A vital element of the proof, known as the 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 *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 $\wedge$ 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 $\textsc{Parity}$, as no matter how many variables we set (and the values we set them to), the $\textsc{Parity}$ function will still depend on all of the remaining variables.

### Decision tree complexity

A key complexity measure used in the proof of $\textsc{Parity} \notin \mathsf{AC}^0$ is the decision tree complexity of a Boolean function.

**Definition 11.10.** A *decision tree* over variables $x_1, \dots, x_n$ is a binary tree where each internal node has two children, left and right. Moreover, each internal node is labelled with a variable, and each leaf is labelled with a value of 0 or 1. To evaluate a decision tree at a point $x = (x_1, \dots, x_n)$, we start from the root, and at each internal node with label $x_i$ we *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 $\mathrm{dt}(f)$ be the *depth of the smallest decision tree* computing $f$.

Obviously, for every Boolean function $f$, we have $\mathrm{dt}(f) \leqslant n$. And as a result, the parity function is as hard as it gets for decision trees.

**Observation 11.11.** $\mathrm{dt}(\textsc{Parity}_n) = n$.

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

$$\left(\bigwedge_{i\in T} x_i\right) \wedge \left(\bigwedge_{i\in F} \neg x_i\right).$$

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

**Proposition 11.12.** *Let $f$ be a function with decision tree complexity $\mathrm{dt}(f)$. Then $f$ is computable by a $\mathrm{dt}(f)$-DNF and a $\mathrm{dt}(f)$-CNF with the number of clauses bounded by the number of the leaves.*

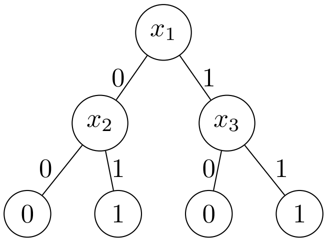

Figure 11.2: A depth 2 Decision Tree computing the function $f : \{0,1\}^3 \to \{0,1\}$ that can be represented by 2-DNF $(\neg x_1 \wedge x_2) \vee (x_1 \wedge x_3)$ and 2-CNF $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_3)$.

### Håstad's switching lemma

As we mentioned earlier, Håstad obtained near-optimal bounds for the size of a depth $d$ circuit that computes $\textsc{Parity}$. 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 11.13.** Let $X = \{x_1, \dots, x_n\}$ be the input variables to a circuit $C$ computing a function $f$. A *restriction* $\rho$ is an element[1] 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 11.14** (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 $\frac{1-p}{2}$. Then for every $s \in \mathbb{N}$,*

$$\mathbf{Pr}[\mathrm{dt}(f_\rho) > s] \leqslant (5pt)^s.$$

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

$$\mathbf{Pr}[\mathrm{dt}(f_\rho) > s] \leqslant 2^{-s}.$$

[1] $\{0,1,*\}^X$ means the set of functions $\rho : X \to \{0,1,*\}$

**Remark 11.15.** 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 $(x_1 \vee \ldots \vee x_t)$.

- (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.
- (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 $\mathrm{dt}(f_\rho) = 0$.
- (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 $(5pt)^{s-|T|}$. 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 bout $(5pt)^{|T|}$.

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 11.16** (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 $F : \{0,1\}^n \to \{0,1\}$, we have*

$$\mathbf{Pr}[\mathrm{dt}(f_\rho) > s | F_\rho \equiv 1] \leqslant (5pt)^s, \tag{11.1}$$

*where $F_\rho \equiv 1$ is the event when $F_\rho$ is the constant 1 function.*

*Proof.* Set $\alpha := 5pt$, and suppose that $f = \wedge_{i=1}^m C_i$ 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 $T \subseteq \{1, \ldots, n\}$, $|T| \leqslant t$. First, we split the LHS of Eq. (11.1) into two terms based on whether $C_1$ receives a 1 from the restriction:

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

Hence in order to prove (11.1) it suffices to prove both

$$\mathbf{Pr}[\mathrm{dt}(f_\rho) > s | F_\rho \equiv 1,\ \rho_T \notin \{0,*\}^T] \leqslant \alpha^s, \tag{11.2}$$

and

$$\mathbf{Pr}[\mathrm{dt}(f_\rho) > s | F_\rho \equiv 1,\ \rho_T \in \{0,*\}^T] \leqslant \alpha^s, \tag{11.3}$$

as then we would have

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

To prove (11.2) note that for $g = \wedge_{i=2}^m C_i$ (which has only $m-1$ clauses),

$$\text{L.H.S of (11.2)} \quad = \quad \mathbf{Pr}[\mathrm{dt}(g_\rho) > s | F_\rho \equiv 1,\ \rho_T \notin \{0,*\}^T] = \mathbf{Pr}[\mathrm{dt}(g_\rho) > s \mid (F \wedge C_1)_\rho \equiv 1] \leqslant \alpha^s,$$

where in the last inequality we used the induction hypothesis applied to $g$ and $F \wedge C_1$. It remains to prove (11.3). We break (11.3) into $2^{|T|}$ terms based on which coordinates in $T$ are $*$'s and which ones are 0's:

$$\begin{aligned}
\text{L.H.S of (11.3)} &= \sum_{Y \subseteq T} \mathbf{Pr}[\mathrm{dt}(f_\rho) > s,\ \rho_Y = *,\ \rho_{T-Y} = \mathbf{0} \mid F_\rho \equiv 1,\ \rho_T \in \{0,*\}^T] \\
&\leqslant \sum_{Y \subseteq T} \mathbf{Pr}[\rho_Y = *, \rho_{T-Y} = \mathbf{0} \mid F_\rho \equiv 1,\ \rho_T \in \{0,*\}^T] \times \\
&\qquad\qquad \mathbf{Pr}[\mathrm{dt}(f_\rho) > s \mid F_\rho \equiv 1,\ \rho_Y = *,\ \rho_{T-Y} = \mathbf{0},\ \rho_T \in \{0,*\}^T] \\
&\leqslant \sum_{Y \subseteq T} \mathbf{Pr}\left[\rho_Y = * \,\middle|\, F_\rho \equiv 1, \rho_T \in \{0,*\}^T\right] \times \mathbf{Pr}\left[\mathrm{dt}(f_\rho) > s \,\middle|\, F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = \mathbf{0}\right].
\end{aligned}$$

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

$$\text{L.H.S of (11.3)} \leqslant \sum_{\substack{Y \subseteq T \\ Y \neq \emptyset}} \mathbf{Pr}\left[\rho_Y = * \,\middle|\, F_\rho \equiv 1, \rho_T \in \{0,*\}^T\right] \times \mathbf{Pr}\left[\mathrm{dt}(f_\rho) > s \,\middle|\, F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = \mathbf{0}\right]. \tag{11.4}$$

We bound the two terms in the product separately.

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

$$\mathbf{Pr}[F_\rho \equiv 1 \mid \rho_Y = *,\ \rho_T \in \{0,*\}^T] \leqslant \mathbf{Pr}[F_\rho \equiv 1 \mid \rho_T \in \{0,*\}^T],$$

Hence using $\mathbf{Pr}[A|B]\,\mathbf{Pr}[B] = \mathbf{Pr}[A \wedge B]$ we have

$$\begin{aligned}
\mathbf{Pr}[\rho_Y = * \mid F_\rho \equiv 1,\ \rho_T \in \{0,*\}^T] &= \frac{\mathbf{Pr}[F_\rho \equiv 1 \mid \rho_Y = *,\ \rho_T \in \{0,*\}^T]}{\mathbf{Pr}[F_\rho \equiv 1 \mid \rho_T \in \{0,*\}^T]} \mathbf{Pr}[\rho_Y = * \mid \rho_T \in \{0,*\}^T] \\
&\leqslant \mathbf{Pr}[\rho_Y = * \mid \rho_T \in \{0,*\}^T] = \left(\frac{2p}{1+p}\right)^{|Y|} \leqslant (2p)^{|Y|}
\end{aligned}$$

**Second observation: (bounding $\mathbf{Pr}\left[\mathrm{dt}(f_\rho) > s \,\middle|\, F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = \mathbf{0}\right]$):** 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 $\mathrm{dt}(f_{\sigma\rho}) \leqslant s - |Y|$, then $\mathrm{dt}(f_\rho) \leqslant s$. Indeed to verify this, note that we can always build a decision tree of depth at most $|Y| + \max_\sigma \mathrm{dt}(f_{\sigma\rho})$ 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 $\mathrm{dt}(f_{\sigma\rho})$ afterwards. Hence for $Y \neq \emptyset$, recalling that $g = \wedge_{i=2}^m C_i$, we have

$$\begin{aligned}
\mathbf{Pr}[\mathrm{dt}(f_\rho) > s \mid F_\rho \equiv 1, \rho_Y = *, \rho_{T-Y} = \mathbf{0}] &\leqslant \mathbf{Pr}\left[\exists \sigma \in \{0,1\}^{|Y|},\ \mathrm{dt}(f_{\sigma\rho}) > s - |Y| \,\middle|\, F_\rho \equiv 1, \rho_{T-Y} = \mathbf{0}\right] \\
&\leqslant \sum_{\sigma \in \{0,1\}^{|Y|}} \mathbf{Pr}[\mathrm{dt}(f_{\sigma\rho}) > s - |Y| \mid F_\rho \equiv 1, \rho_{T-Y} = \mathbf{0}] \\
&= \sum_{\sigma \in \{0,1\}^{|Y|}} \mathbf{Pr}[\mathrm{dt}(f_{\sigma\rho}) > s - |Y| \mid (F \wedge \wedge_{i \in T \setminus Y} \overline{x_i})_\rho \equiv 1] \\
&= \sum_{\sigma \in \{0,1\}^{|Y|}} \mathbf{Pr}[\mathrm{dt}(g_{\sigma\rho}) > s - |Y| \mid (F \wedge \wedge_{i \in T \setminus Y} \overline{x_i})_\rho \equiv 1] \\
&\leqslant \sum_{\sigma \in \{0,1\}^{|Y|}} \alpha^{s-|Y|} \leqslant 2^{|Y|}\alpha^{s-|Y|}.
\end{aligned}$$

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

Combining the two observations with (11.4), we finish the proof:

$$\begin{aligned}
\text{L.H.S of (11.3)} &\leqslant \sum_{\substack{Y \subseteq T \\ Y \neq \emptyset}} 2^{|Y|}\alpha^{s-|Y|}(2p)^{|Y|} = \alpha^s \sum_{\substack{Y \subseteq T \\ Y \neq \emptyset}} \left(\frac{4p}{\alpha}\right)^{|Y|} = \alpha^s \left(\left(1 + \frac{4p}{\alpha}\right)^{|T|} - 1\right) \\
&= \alpha^s \left(\left(1 + \frac{4}{5t}\right)^t - 1\right) \leqslant \alpha^s (e^{\frac{4}{5}} - 1) \leqslant \alpha^s.
\end{aligned}$$

□

**Remark 11.17.** 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 11.16.

**Corollary 11.18.** *Let $f$ be a Boolean function computed by an* AC *circuit of size $M$ and depth $d$. Choose a random restriction $\rho$ by setting every variable independently to $*$ with probability $p = \frac{1}{10^d s^{d-1}}$, and to $0$ and $1$ each with probability $\frac{1-p}{2}$. Then*

$$\mathbf{Pr}[\mathrm{dt}(f_\rho) > s] \leqslant M2^{-s}.$$

*Proof.* We sample the restriction $\rho$ by first sampling a random restriction $\rho_0$ with $\mathbf{Pr}[*] = 1/10$, and then sampling $d-1$ consecutive restrictions $\rho_1, \ldots, \rho_{d-1}$ each with $\mathbf{Pr}[*] = \frac{1}{10s}$.

Assume without loss of generality that the bottom gates are $\vee$. We claim, With high probability, after the restriction $\rho_0$, all the remaining bottom fan-ins are at most $s$. To see this, consider two cases for each gate at the bottom level of the original circuit:

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, no input to this gate got assigned a 1 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 $\binom{2s}{s}(1/10)^s \leqslant 2^{-s}$.

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 $\mathbf{Pr}[*] = \frac{1}{10s}$. After each of these, we use Håstad's switching lemma (see Remark 11.17) 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 $\mathrm{dt}(g_{\rho_i}) > s$, is hence bounded by $(5\frac{1}{10s}s)^s \leqslant 2^{-s}$. The probability that a particular gate fails to satisfy the desired property is no more than $2^{-s}$. Since the top gate is $\wedge$, 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 $\mathrm{dt}(f_\rho) \leqslant s$. 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 contributes $2^{-s}$ to the probability of failure, and hence applying the union bound yields the desired bound. □

Using the above statement, if $M2^{-s}$ is bounded away from 1, and $s \ll pn$, 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 $\textsc{Parity}_\rho$ requires decision trees of full depth (Observation 11.8), which is $> s$. Comparing the parameters proves Theorem 11.9.

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

$$\mathbf{Pr}[f_\rho \textit{ does not have a CNF with fan-in} \leqslant s] \leqslant M2^{-s}.$$

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

In the next section, we will show that this improvement implies a better lower bound of $2^{\Omega(n^{1/d-1})}$ for $\textsc{Parity}$, 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 11.20** (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,

$$I_i(f) = \mathbf{Pr}[f(x) \neq f(x \oplus e_i)],$$

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 $0 \leqslant I_i(f) \leqslant 1$ and $0 \leqslant I_f \leqslant n$. 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 $f$ as a function of 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 $\int \left|\frac{\partial h}{\partial x_i}\right| = \int \lim_{\delta \to 0} \frac{|h(x+\delta e_i)-h(x)|}{t}$.

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

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

linearity of expectation shows

$$I_f = \mathbb{E}[s_f(x)], \tag{11.5}$$

and thus sometimes $I_f$ is called the *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.

**Lemma 11.21.** *Let $f$ be a CNF or a DNF formula where all the clauses are of size at most $t$. Then $I_f \leqslant 2t$.*

*Proof.* We prove the lemma for the DNF case, and the CNF case follows by replacing $f$ with $1-f$. For $x \in \{0,1\}^n$, let $s_{1\to 0}(x)$ denote the number of coordinates $i \in [n]$ such that $f(x) = 1$ and $f(x \oplus e_i) = 0$. Trivially if $f(x) = 0$, then $s_{1\to 0}(x) = 0$. One can define $s_{0\to 1}(x)$ similarly so that $s_f(x) = s_{1\to 0}(x) + s_{0\to 1}(x)$ for every $x$. Note that

$$\mathbb{E}\left[s_{0\to 1}(x)\right] = \mathbb{E}\left[s_{1\to 0}(x)\right],$$

because for every $i$,

$$\mathbf{Pr}[f(x) = 0 \wedge f(x \oplus e_i) = 1] = \mathbf{Pr}[f(x) = 1 \wedge f(x \oplus e_i) = 0].$$

In particular,

$$I_f = \mathbb{E}[s_f(x)] = 2\mathbb{E}\left[s_{1\to 0}(x)\right].$$

Finally, note that since $f$ is represented by a DNF with clauses of size at most $t$, we have $s_{1\to 0}(x) \leqslant t$ for every $x$ as $f(x) = 1, f(x \oplus e_i) = 0$ is only possible if $x_i$ is involved in all satisfying clauses. We conclude

$$I_f = \mathbb{E}[s_f(x)] = 2\mathbb{E}\left[s_{1\to 0}(x)\right] \leqslant 2t.$$

□

Boppana [Bop97] proved that small-size, low-depth AC circuits have small total influences.

**Theorem 11.22** (Boppana [Bop97])**.** *Let $f$ be a Boolean function computed by an AC circuit of depth $d$ and size $M$ (including the input gates), then*

$$I_f \leqslant (20 \log M)^d.$$

*Proof.* Applying Corollary 11.18 with $s = 2 \log M$ and $p = \frac{1}{10^d s^{d-1}}$, and combining it with the easy fact that $I_f \leqslant \mathrm{dt}(f)$, we conclude that

$$\mathbf{Pr}[I_{f_\rho} \geqslant s] \leqslant M 2^{-s} \leqslant \frac{1}{M} \leqslant \frac{1}{n}.$$

Here we are using $n \leqslant M$ by counting the input gates in the size of the circuit. Hence

$$\mathbb{E}_\rho[I_{f_\rho}] \leqslant \mathbf{Pr}[I_{f_\rho} > s]n + s \leqslant \frac{1}{n}n + s \leqslant s + 1 \leqslant 2s.$$

On the other hand, note that for every $i$,

$$\mathbf{Pr}_{\rho,x}[f_\rho(x) \neq f_\rho(x \oplus e_i)] = p \,\mathbf{Pr}_{x}[f(x) \neq f(x \oplus e_i)],$$

where $p$ is the probability that the $i$th variable is not fixed by $\rho$. Hence

$$\mathbb{E}_\rho[I_{f_\rho}] = pI_f.$$

We conclude

$$I_f \leqslant \frac{2s}{p} \leqslant 2s \cdot (10s)^{d-1} \leqslant (20 \log M)^d.$$

□

One can improve the bound slightly by using Corollary 11.19 and Lemma 11.21 instead of Corollary 11.18.

**Theorem 11.23** (Boppana [Bop97])**.** *Let $f$ be a Boolean function computed by an AC circuit of depth $d$ and size $M$, then*

$$I_f \leqslant (20 \log M)^{d-1}.$$

The majority function $\textsc{Maj}$ is defined as $\textsc{Maj}(x) = 1$ if and only if $\sum x_i \geqslant n/2$. As we saw above, the total influence of $\textsc{Parity}$ is $n$. It is also straightforward to see (Exercise 11.2) that the total influence of $\textsc{Maj}$ is $\Theta(\sqrt{n})$. We conclude

**Corollary 11.24.** *The functions $\textsc{Parity}$ and $\textsc{Maj}$ are not in $\mathrm{AC}^0$. In particular, $\textsc{Parity}$ and $\textsc{Maj}$ require AC circuits of depth $d$ and size at least $2^{\Omega(n^{\frac{1}{d-1}})}$ and $2^{\Omega(n^{\frac{1}{2(d-1)}})}$ respectively.*

To this day, Håstad's bound for parity remains the strongest explicit known lower bound against small-depth circuits for any function, even in the case of $d = 3$. The special case of depth-3 has received significant attention as one of the simplest restricted models where our understanding is lacking. The following open problem is one of the frontiers of circuit complexity.

**Problem 11.25.** *Find an explicit function $f : \{0,1\}^n \to \{0,1\}$ that requires circuit size $2^{\omega(\sqrt{n})}$ for AC circuits of depth 3.*

Valiant proved that a construction with a stronger lower bound of $2^{\Omega(n)}$ for Problem 11.25 would imply a solution to Problem 11.3. More precisely, if $f : \{0,1\}^n \to \{0,1\}$ requires circuit size $2^{\omega(\sqrt{n})}$ for AC circuits of depth 3, then it requires circuit size $\omega(n)$ for fan-in 2 circuits of depth $O(\log(d))$.

## Exercises

**Exercise 11.1.** 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 11.2.** Prove that the total influence of $\textsc{Maj} : \{0,1\}^n \to \{0,1\}$ is $\Theta(\sqrt{n})$.

**Exercise 11.3.** 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 11.4.** Prove that

$$\mathrm{Var}[f] \leqslant I_f.$$

**Exercise 11.5.** 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 $\deg(f)$ is bounded by $\mathrm{dt}(f)$. Is it true that $\deg(f) \leqslant t$ if $f$ has a $t$-CNF representation?

**Exercise 11.6.** Let $\mathrm{size}_{\mathrm{dt}}(f)$ 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 $\mathbf{Pr}[*] = p$), we have

$$\mathbb{E}_\rho[\mathrm{size}_{\mathrm{dt}}(f_\rho)] \leqslant \mathrm{size}_{\mathrm{dt}}(f)^{\log(1+p)}.$$

Give an example where the bound is sharp.

# Chapter 12

# $\mathsf{AC}^0$ with parity gates: Razborov-Smolensky

We have seen that $\mathsf{AC}$ circuits of bounded depth cannot compute Parity and Maj. A natural way to strengthen such circuits is to allow gates capable of computing such hard functions. In particular, two classes $\mathsf{TC}^0$ and $\mathsf{ACC}^0$ are defined as follows.

For $i \geqslant 0$, let $\mathsf{TC}^i$ be the class of Boolean functions computable by Boolean threshold circuits with a polynomial number of gates and depth $O(\log^i n)$. Threshold circuits are Boolean circuits with $\wedge$, $\vee$, $\neg$, and Maj gates. Since $\textsc{Maj} \in \mathsf{NC}^1 \subseteq \mathsf{AC}^1$, we have

$$\mathsf{AC}^i \subseteq \mathsf{TC}^i \subseteq \mathsf{NC}^{i+1} \subseteq \mathsf{AC}^{i+1}.$$

Our understanding of the class $\mathsf{TC}^0$ is very limited. It is not known whether $\mathsf{P} \not\subseteq \mathsf{TC}^0$.

On the other hand, define $\mathsf{AC}^i[m]$ 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, $\mathsf{AC}^i[m]$ circuits are allowed gates $\wedge, \vee, \neg$ in addition to $\mathrm{mod}_m(x)$ gates which evaluate to 1 if and only if $\sum x_i \equiv 0 \mod m$. Define $\mathsf{ACC}^i = \cup_m \mathsf{AC}^i[m]$. It is easy to verify that

$$\mathsf{AC}^i \subseteq \mathsf{ACC}^i \subseteq \mathsf{TC}^i.$$

It is conjectured that $\textsc{Maj} \notin \mathsf{AC}^0[6]$, however, it is not yet ruled out whether $\mathsf{P}/\mathrm{poly} \not\subseteq \mathsf{AC}^0[6]$. It is not even ruled out whether $\mathsf{EXP} \not\subseteq \mathsf{AC}^0[6]$. The strongest lower-bounds known for $\mathsf{ACC}^0$ are due to recent work of Murray and Williams [MW18] who built on a breakthrough of Williams [Wil14] to prove that $\mathsf{NQP} \not\subseteq \mathsf{ACC}^0$. Here $\mathsf{NQP}$ refers to nondeterministic quasipolynomial time.

The case of $\mathsf{AC}^0[2]$ (also denoted $\mathsf{AC}^0[\oplus]$) 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 Parity and Maj, the former belongs to $\mathsf{AC}^0[\oplus]$, Razborov [Raz87] and Smolensky [Smo87] proved that the latter, Maj, is hard even for $\mathsf{AC}^0[\oplus]$. Their proof has an important implication that every $\mathsf{AC}^0$ 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 $f : \{0,1\}^n \to \mathbb{R}$ has a unique representation as a multivariate polynomial in $\mathbb{R}(x_1, \dots, x_n)$ 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 12.1.** We have

$$\textsc{Maj}(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_1x_3 - 2x_1x_2x_3.$$

$$\textsc{Parity}(x_1, x_2, x_3) = x_1 + x_2 + x_3 - 2x_1x_2 - 2x_2x_3 - 2x_1x_3 + 4x_1x_2x_3.$$

$$\wedge(x_1, \dots, x_n) = x_1x_2 \dots x_n,$$

and

$$\vee(x_1, \dots, x_n) = 1 - (1-x_1)(1-x_2)\dots(1-x_n) = \sum_{\emptyset \subsetneq S \subseteq [n]} (-1)^{|S|+1} \prod_{i \in S} x_i.$$

Note that even the simplest $\mathsf{AC}^0$ functions such as $\wedge$ and $\vee$ have a full degree as a polynomial. However, the following theorem shows that every $\mathsf{AC}^0$ function can be approximated well with a low-degree polynomial.

**Theorem 12.2** ([Raz87], [Smo87])**.** *Let $f : \{0,1\}^n \to \{0,1\}$ 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 $\leqslant (s \log M)^d$ such that*

$$\Pr_x[f(x) \neq g(x)] \leqslant \left(1 - \frac{1}{2e}\right)^s M,$$

*where $x$ is chosen randomly and uniformly from $\{0,1\}^n$. In particular, taking $s = 100\log(M)$, there is a polynomial of degree $\leqslant (100 \log M)^{2d}$, such that*

$$\Pr_x[f(x) \neq g(x)] \leqslant 0.01.$$

*Proof.* The key is approximating the $\wedge$ and $\vee$ gates with low-degree polynomials. The function $g$ is constructed inductively. We will show how to make a step with an $\wedge$ gate. Since the whole construction is symmetric concerning 0 and 1, the step also holds for an $\vee$ gate. Let

$$f = \wedge_{i=1}^k f_i$$

where $k < M$. For convenience, let us assume that $k = 2^\ell$ is a power of 2. For every $j = 1, \ldots, \ell$, pick $s$ 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 $t := s\ell \leqslant s \log M$. 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 $\leqslant (s \log M)^{d-1}$. Hence, the degree of $f$ is bounded by $t(s \log M)^{d-1} \leqslant (s \log M)^d$. 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. Consider any $x$ such that $g_j(x) = f_j(x)$ for all $j$. We have

$$\Pr_{S_1,\ldots,S_t}[f(x) \neq g(x)] = \Pr_{S_1,\ldots,S_t}\left[\prod_{i=1}^t \left(1 - |S_i| + \sum_{j \in S_i} f_j\right) \neq \prod_{j=1}^k f_j\right].$$

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 $1 \leqslant z \leqslant k$ be the number of zeros among $f_1(x), \ldots, f_k(x)$, and $\alpha \in \mathbb{Z}$ be such that $2^\alpha \leqslant z < 2^{\alpha+1}$. 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

  $$zp(1-p)^{z-1} \geqslant \frac{1}{2}(1-p)^{\frac{1}{p}-1} > \frac{1}{2e}.$$

  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 $\leqslant M$ 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 $x \in \{0,1\}^n$,

$$\Pr_g[f(x) \neq g(x)] \leqslant M\left(1 - \frac{1}{2e}\right)^s.$$

Since this holds for every $x$, we have

$$\Pr_{g,x}[f(x) \neq g(x)] \leqslant M\left(1 - \frac{1}{2e}\right)^s,$$

which shows that there is a fixed polynomial $g_0$ such that

$$\Pr_x[f(x) \neq g_0(x)] \leqslant M\left(1 - \frac{1}{2e}\right)^s,$$

as desired.

□

## The class $\mathsf{AC}^0[\oplus]$: Algebraic techniques

In this section, we study the class $\mathsf{AC}^0[\oplus]$ and ultimately show that $\textsc{Maj} \notin \mathsf{AC}^0[\oplus]$. 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 12.3.** *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 $i \leqslant j$ implies $|S_i| \leqslant |S_j|$.

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 $y_i = 1 \Leftrightarrow i \in S$. With this notation, note that

$$\phi_S(S) = 1,$$

while

$$\phi_T(S) = 0,$$

for all $T \neq S$ with $|T| \geqslant |S|$.

This shows that

$$\phi_{S_i} \notin \mathrm{span}_{\mathbb{F}_2}\{\phi_{S_j} : j > i\}.$$

Thus $\phi_{S_1}, \phi_{S_2}, \ldots, \phi_{S_{2^n}}$ are linearly independent. □

Since there are exactly $2^n$ monomials $x^S$, which is equal to 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 12.3 has the following corollaries that we will use later in the lecture.

**Corollary 12.4.** *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 12.5.** *For every $A \subseteq \mathbb{F}_2^n$, the set of monomials $\prod_{i \in [n] \setminus S}(1 - x_i)$ for $S \in A$ is a linear basis for the $\mathbb{F}_2$-vector space of all functions $f : A \to \mathbb{F}_2$.*

## Razborov-Smolensky over $\mathbb{F}_2$

In Theorem 12.2, 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 12.2.

**Theorem 12.6** (Theorem 12.2 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 $\leqslant (s \log M)^d$ such that*

$$\mathop{\mathbf{Pr}}_{x}[f(x) \neq g(x)] \leqslant \left(1 - \frac{1}{2e}\right)^s M,$$

*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 $\wedge$ and $\vee$ gates we can use the same approximation as in Theorem 12.2 but simplify everything mod 2, or equivalently carry all the operations over $\mathbb{F}_2$. □

Finally, we will show that $\textsc{Maj} : \mathbb{F}_2^n \to \mathbb{F}_2$ cannot be approximated by any polynomial of degree at most $\sqrt{n}$, and thus conclude that $\textsc{Maj} \notin \mathsf{AC}^0[\oplus]$.

**Theorem 12.7.** *Let $p \in \mathbb{F}_2(x_1, \ldots, x_n)$ be a polynomial of degree $t$. Then*

$$\mathop{\mathbf{Pr}}_{x}[\textsc{Maj}(x) = p(x)] \leqslant \frac{1}{2} + O(t/\sqrt{n}).$$

*Proof.* The proof is a clever algebraic proof based on the notion of rank. We will start by proving the following claim.

**Claim 12.8.** *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*

$$f(x) = g(x) + \textsc{Maj}(x) \cdot h(x).$$

*Proof.* Let $A_0, A_1 \subseteq \mathbb{F}_2^n$ be $A_0 = \textsc{Maj}^{-1}(0)$ and $A_1 = \textsc{Maj}^{-1}(1)$. Applying Corollary 12.4 for $A_0$, there is a linear combination $g(x)$ of $x^S$ for $S \in A_0$ such that

$$f(x) = g(x) \qquad \forall x \in A_0.$$

Since $A_0 = \textsc{Maj}^{-1}(0)$, every $S \in A_0$ satisfies $|S| \leqslant n/2$ and thus $\deg(g) \leqslant n/2$.

To handle $A_1$, consider $\tilde{f}(x) = f(x) - g(x)$. Applying Corollary 12.5 to $A_1$, we obtain a polynomial $h$ of degree at most $n/2$ such that

$$\tilde{f}(x) = h(x) \qquad \forall x \in A_1.$$

It is easy to check that, $f(x) = g(x) + \textsc{Maj}(x)h(x)$: For $x \in A_0$,

$$g(x) + \textsc{Maj}(x)h(x) = g(x) = f(x).$$

And for $x \in A_1$,

$$g(x) + \textsc{Maj}(x)h(x) = g(x) + h(x) = g(x) + \tilde{f}(x) = f(x).$$

□

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

$$f(x) = g(x) + \textsc{Maj}(x) \cdot h(x) \qquad \forall x \in \mathbb{F}_2^n.$$

Let

$$T = \{x \ : p(x) = \textsc{Maj}(x)\},$$

so that

$$f(x) = g(x) + p(x) \times h(x) \qquad \forall x \in T.$$

Note that $\deg(g + ph) \leqslant \frac{n}{2} + t$. 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,

$$|T| \leqslant \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{\frac{n}{2}+t} \leqslant \frac{2^n}{2} + \sum_{k=\frac{n}{2}}^{\frac{n}{2}+t} \binom{n}{k} \leqslant 2^n \left(\frac{1}{2} + O\left(\frac{t}{\sqrt{n}}\right)\right),$$

using the fact that $\binom{n}{k} \leqslant \binom{n}{n/2} \sim \frac{2^n}{\sqrt{n\pi}} = O\left(\frac{2^n}{\sqrt{n}}\right)$, for all $k$.

□

**Theorem 12.9.** *Let $C$ be an $\mathsf{AC}^0[\oplus]$ circuit of size $M$ and depth $d$ such that*

$$\mathop{\mathbf{Pr}}_x[\textsc{Maj}(x) \neq C(x)] \leqslant \frac{1}{3}.$$

*Then $M \geqslant 2^{\Omega(n^{\frac{1}{4d}})}$.*

*Proof.* See Exercise 12.1. □

## Concluding Remarks

By this point, we have proved that small depth circuits are limited in their power, and even uncomplicated functions such as Maj and Parity require exponentially many gates if we limit the depth to 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

$$\mathop{\mathbf{Pr}}_x[C(x) \neq p(x)].$$

In the case of the proofs by random restrictions of Chapter 11, this is less apparent, but as we will see in Chapter 16, Linial, Mansour, and Nisan [LMN93] use Corollary 11.18 to show the existence of a low-degree polynomial $p(x)$ such that the so-called $L_2$ error term

$$\mathbb{E}_x \left|C(x) - p(x)\right|^2$$

is small.

The proof of the Razoborov-Smolensky theorem 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 12.10** (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*

$$\mathop{\mathbf{Pr}}_x[p(x) = f(x)] \leqslant \frac{1}{2} + \frac{1}{n}.$$

For an in-depth look into the importance of such correlation bounds for polynomials, we refer the reader to [Vio09].

## Exercises

**Exercise 12.1.** Prove Theorem 12.9 using Theorem 12.7 and Theorem 12.6.

**Exercise 12.2.** In this exercise we consider Boolean functions as $f : \{-1,1\}^n \to \{-1,1\}$. Note that in this setting $\textsc{Parity}(x) = \prod_{i\in\{1,\ldots,n\}} x_i$.

Consider the vector space of all functions $f : \{-1,1\}^n \to \mathbb{R}$. Prove that every such function can be written as

$$f = g + \textsc{Parity} \times h,$$

where $g$ and $h$ are polynomials of degree at most $n/2$.

**Exercise 12.3.** Let $f : \{0,1\}^n \to \{0,1\}$, and let $g$ be a random polynomial of degree $\leqslant d$ such that for every $x$, we have

$$\mathop{\mathbf{Pr}}_{g}[f(x) = g(x)] \geqslant \frac{1}{2} + \varepsilon.$$

Prove that for every $k$, there is a random polynomial $h$ of degree $\leqslant kd$ such that for every $x$, we have

$$\mathop{\mathbf{Pr}}_{h}[f(x) \neq h(x)] \leqslant e^{-k\varepsilon^2/4}.$$

**Exercise 12.4.** 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 $\vee_{i=1}^n x_i = 0$, then $\mathbf{Pr}_R[T_R(x) = 0] = 1$, and if $\vee_{i=1}^n x_i = 1$, then

$$\mathop{\mathbf{Pr}}_{R}[T_R(x) = \vee_{i=1}^n x_i] \geqslant \frac{1}{2}.$$

**Exercise 12.5.** 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 $\sum x_i \geqslant k$. 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 $\mathbf{Pr}_R[T_R(x) = 0] = 1$, and if $t_k(x) = 1$, then

$$\mathop{\mathbf{Pr}}_{R}[T_R(x) = t_k(x)] \geqslant \frac{1}{2}.$$

Note that $k = 1$ corresponds to the previous exercise.

**Exercise 12.6.** Deduce an analogue of Theorem 12.7 for the Parity function from the Switching lemma methods: There are constants $c_1, c_2 > 0$ such that for every $\mathsf{AC}$ circuit $C$ of depth $d$ and size at most $2^{c_1 n^{1/d}}$,

$$\mathbf{Pr}[C(x) = \textsc{Parity}(x)] \leqslant \frac{1}{2} + 2^{-c_2 n^{1/d}}.$$

**Exercise 12.7.** In this exercise our goal is to prove that unlike majority, *approximate majority* can be solved by an $\mathsf{AC}^0$ circuit. More precisely we want to construct a polynomial size circuit $C$ of depth 3 such that if $|x| := \sum x_i \geqslant \frac{3n}{4}$, then $C(x) = 1$, and if $|x| < \frac{n}{4}$, then $C(x) = 0$. For $\frac{n}{4} \leqslant |x| < \frac{3n}{4}$, 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 $C_0$.
- $C_2 = \vee_{i=1}^{n^{15}} D_i$, 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

$$\mathop{\mathbf{Pr}}_{C_3}[C_3(x) = 1] < 2^{-n^2},$$

and if $|x| \geqslant \frac{3n}{4}$, then

$$\mathop{\mathbf{Pr}}_{C_3}[C_3(x) = 0] < 2^{-n^2}.$$

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 $|x| \geqslant \frac{3n}{4}$, $C(x) = 1$.

**Exercise 12.8** (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 $d : X \times X \to \mathbb{R}^{\geqslant 0}$ satisfies the following properties:

- $d(x, y) = 0$ implies $x = y$.
- $d(x, y) = d(y, x)$ for all $x, y$.
- $d(x, y) + d(y, z) \geqslant d(x, z)$ 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 $x, y$:

$$d(x, y) \leqslant \|\psi(x) - \psi(y)\| \leqslant O(\log(n)) \times d(x, y).$$

Hint: Pick subsets $S_1, \dots, S_d \subseteq X$ randomly and according to carefully chosen parameters for an appropriate value of $d$, and define $\phi(x) = (d(x, S_1), \dots, d(x, S_d))$, where $d(x, S) := \min_y d(x, S)$. Obtain $\psi$ by scaling $\phi$ so that $d(x, y) \leqslant \|\psi(x) - \psi(y)\|$.

# Chapter 13

# Razborov's monotone circuit lower-bound

We continue our study of the computational power of restricted classes of circuits. In Chapters 11 and 12, we considered bounded depth circuits. In this chapter, we turn our attention to monotone circuits with no depth restriction. Since we allow arbitrary depth, we can assume that all gates have fan-in 2, as breaking the larger fan-in $\wedge$ and $\vee$ gates into binary gates does not increase the circuit size by much.

Many natural functions, such as 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 $\wedge$ and $\vee$ 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 13.1.** Let $\textsc{Clique}_k : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ 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 there is an edge between the vertices $i$ and $j$.

Obviously, $\textsc{Clique}_k$ is a monotone function and can be represented by a monotone circuit of size $O(n^k) = O(2^{k \log(n)})$ as

$$\bigvee_{\substack{S \subseteq \{1,\ldots,n\} \\ |S|=k}} \bigwedge_{\substack{i,j \in S \\ i \neq j}} x_{ij}. \tag{13.1}$$

□

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 13.2.** *[Raz85a, AB87] For sufficiently large $n$, $\textsc{Clique}_{n^{1/6}}$ requires monotone circuits of size at least $2^{n^{1/12}}$.*

Note that Theorem 13.2 looks incredibly impressive, as unlike the AC lower-bounds of Chapters 11 and 12, it does not restrict the depth of the circuit. Indeed at the time, many people believed this was very close to proving a general circuit lower-bound, hence establishing $\mathsf{P} \neq \mathsf{NP}$. 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 $\textsc{Clique}_k$. However, as later research showed, monotone circuits are very different from general circuits as they cannot simulate general algorithms. Indeed we are still nowhere near proving $\mathsf{P} \neq \mathsf{NP}$.

Let $V = \{1, \ldots, n\}$ denote the vertex set. For every $S \subseteq V$, define $C_S(G)$ as 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 (13.1) as

$$\textsc{Clique}_k(G) = \bigvee_{S \in \binom{V}{[k]}} C_S(G),$$

where $\binom{V}{[k]}$ denotes the collection of all subsets of size $k$ of $V$.

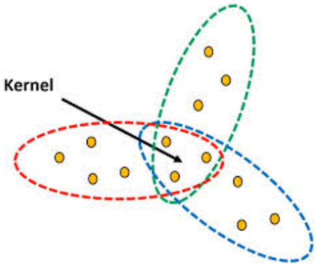

Figure 13.1: A sunflower with 3 petals.

Razborov's idea is to show that every small monotone circuit that tries to compute the clique function can be "approximated" by a DNF of the form $\bigvee_{S \in \mathcal{T}} C_S(G)$ for a small collection $\mathcal{T}$ of subsets of $V$. To achieve this, he applies an important combinatorial result due to Erdös and Rado [ER60] called the sunflower lemma: A collection of sets $Z_1, \ldots, Z_p$ is called a *sunflower* if $Z_i \cap Z_j = \cap_{k=1}^{p} Z_k$ for all $i \neq j$. The set $\cap_{k=1}^{p} Z_k$ is called the *core* of the sunflower, and the sets $Z_i$ are called its *petals*. See Figure 13.1.

**Theorem 13.3** (Sunflower lemma [ER60])**.** *Every collection of $(p-1)^\ell \ell!$ sets of sizes at most $\ell$ contains a sunflower with $p$ petals.*

**Remark 13.4.** Erdös and Rado [ER60] conjectured that their $(p-1)^\ell \ell! \approx \Theta\left(p\ell\right)^\ell$ 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 13.3 to $\Theta(p\log(\ell))^\ell$. However, for our application, the bound in Theorem 13.3 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

$$\Pr_{G \sim \mu_{\text{YES}}}[C(G) = \text{YES}] = 1 \qquad \text{and} \qquad \Pr_{G \sim \mu_{\text{NO}}}[C(G) = \text{NO}] = 1.$$

Solving CLIQUE$_k$ on either $\mu_{\text{YES}}$ or $\mu_{\text{NO}}$ is very easy. For $\mu_{\text{YES}}$, it suffices always to output YES, and for $\mu_{\text{NO}}$, one has always to output NO. However, Razborov's proof shows that no sub-exponential size monotone circuit can simultaneously succeed 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 $\widetilde{C}(G) = \bigvee_{S \in \mathcal{T}} C_S(G)$ satisfies

$$\Pr_{G \sim \mu_{\text{YES}}}[\widetilde{C}(G) = \text{YES}] \geqslant 0.9 \qquad \text{and} \qquad \Pr_{G \sim \mu_{\text{NO}}}[\widetilde{C}(G) = \text{NO}] \geqslant 0.9.$$

In other words, $\widetilde{C}$ 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 $\mathcal{T}$ for $\widetilde{C}$ to do well on the distributions $\mu_{\text{YES}}$ and $\mu_{\text{NO}}$.

# The preliminary lemmas

Throughout the proof, we set the following parameters:

$$k = n^{1/6}, \qquad \ell = \sqrt{k}, \qquad p = 10\sqrt{k}\log k, \qquad t = k^{2\ell} \geqslant (p\ell)^{\ell} \geqslant (p-1)^{\ell}\ell!. \tag{13.2}$$

We show that a single $C_S$ is a very poor approximation of $\textsc{Clique}_k$.

**Lemma 13.5.** *Let $\ell$ and $t$ be as in (13.2), and consider a set $S \subseteq V$.*

*(I) If $|S| \leqslant \ell$, then*

$$\mathop{\mathbf{Pr}}_{G\sim\mu_{\text{No}}}[C_S(G) = \textsc{Yes}] \geqslant \frac{1}{2}.$$

*(II) If $|S| \geqslant \ell$, then*

$$\mathop{\mathbf{Pr}}_{G\sim\mu_{\text{Yes}}}[C_S(G) = \textsc{Yes}] \leqslant t^{-2}2^{-2\ell}.$$

*Proof.* First consider $|S| \leqslant \ell$. Let $c : V \to \{1, \dots, k-1\}$ be chosen uniformly at random. For distinct $i, j \in S$, the probability that $c(i) = c(j)$ is $\frac{1}{k-1}$. 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

$$\binom{|S|}{2}\frac{1}{k-1} = \binom{\ell}{2}\frac{1}{k-1} < 0.5.$$

Hence with probability at least $\frac{1}{2}$, the set $S$ is a clique in $G$.

Next, consider $|S| \geqslant \ell$, 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 $n - \ell + 1 \geqslant n/2$, and $4k^5 < n$,

$$\mathbf{Pr}[C_S(G) = 1] = \frac{\binom{n-\ell}{k-\ell}}{\binom{n}{k}} = \frac{k(k-1)\dots(k-\ell+1)}{n(n-1)\dots(n-\ell+1)} \leqslant \left(\frac{2k}{n}\right)^{\ell} \leqslant t^{-2}2^{-\ell}.$$

□

The next lemma shows that replacing a sunflower with its core never damages the circuit on $\textsc{Yes}$ instances, and it is very unlikely to damage the output of the circuit on a $G$ drawn from $\mu_{\text{No}}$.

**Remark 13.6.** 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 $\mu_{\text{No}}$ it is fine if our approximation turns the output of a gate from 1 to 0. These cannot change the output of the circuit erroneously.

**Lemma 13.7.** *Let $Z_1, \dots, Z_p$ be a sunflower with core $R$ and sets of sizes at most $\ell$. Then $\bigvee_{i=1}^{p} C_{Z_i}(G) \leqslant C_R(G)$ for all $G$, and furthermore for $G \sim \mu_{\text{No}}$,*

$$\mathop{\mathbf{Pr}}_{G\sim\mu_{\text{No}}}\left[\bigvee_{i=1}^{p} C_{Z_i}(G) \neq C_R(G)\right] \leqslant 2^{-p}.$$

*Proof.* It is obvious that $\bigvee_{i=1}^{p} C_{Z_i} \leqslant C_R$, and the inequality is strict only if $C_R$ detects a clique on $R$, while all $Z_1, \dots, Z_p$ have missing edges. We will show that this happens with a small probability if $G \sim \mu_{\text{No}}$. We want to bound

$$\mathop{\mathbf{Pr}}_{G\sim\mu_{\text{No}}}[C_R(G) = 1 \text{ and } \vee_{i=1}^{p} C_{Z_i}(G) = 0] = \mathop{\mathbf{Pr}}_{\mu_{\text{No}}}[C_R(G) = 1]\mathop{\mathbf{Pr}}_{\mu_{\text{No}}}[\vee_{i=1}^{p} C_{Z_i}(G) = 0 | C_R(G) = 1].$$

We have

$$\mathop{\mathbf{Pr}}_{\mu_{\text{No}}}[\vee_{i=1}^{p} C_{Z_i}(G) = 0 | C_R(G) = 1] = \prod_{i=1}^{p} \mathop{\mathbf{Pr}}_{\mu_{\text{No}}}[C_{Z_i}(G) = 0 | C_R(G) = 1] \leqslant \prod_{i=1}^{p} \mathop{\mathbf{Pr}}_{\mu_{\text{No}}}[C_{Z_i}(G) = 0] \leqslant \left(\frac{1}{2}\right)^{p},$$

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 13.5. □

## Approximation by a small number of cliques

**Lemma 13.8.** *Consider the parameters in (13.2). 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 $\widetilde{C}(G) = \bigvee_{S\in\mathcal{T}} C_S(G)$ satisfies*

$$\Pr_{G\sim\mu_{\text{Yes}}}[\widetilde{C}(G) = \text{No}] \leqslant \frac{M2^{-\sqrt{k}}}{4} \qquad \textit{and} \qquad \Pr_{G\sim\mu_{\text{No}}}[\widetilde{C}(G) = \text{Yes}] \leqslant \frac{M2^{-\sqrt{k}}}{4}.$$

We will prove the lemma by induction, traversing the circuit from the bottom to the top and replacing each gate by OR's of $C_S$. 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 we will ensure that the replaced gate approximates the original gate with an error probability of at most $2^{-\sqrt{k}-2}$ 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 $M2^{-\sqrt{k}-2}$.

**Handling the OR gates:** Consider $f \vee g$ where $f = \bigvee_{S\in\mathcal{T}_1} C_S$ and $g = \bigvee_{S\in\mathcal{T}_2} C_S$ with $|\mathcal{T}_1|, |\mathcal{T}_2| \leqslant t$. Obviously $f \vee g = \bigvee_{S\in\mathcal{T}} C_S$ 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 $t$. We cannot afford to double the size of $\mathcal{T}$ at each gate. 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 13.7 we can replace the sunflower with its core creating an error of at most $2^{-p}$ for $\mu_{\text{No}}$, and no error for $\mu_{\text{Yes}}$. Since $|\mathcal{T}| \leqslant |\mathcal{T}_1| + |\mathcal{T}_2| \leqslant 2t$, repeating this process at most $t$ times will reduce the size to the desired $t$, and create an error probability of at most

$$t2^{-p} \leqslant 2^{-2\ell} \leqslant 2^{-\sqrt{k}-2},$$

for either of the distributions.

**Handling the AND gates:** Note that for every $G$ we have

$$C_{S_1}(G) \wedge C_{S_2}(G) \geqslant C_{S_1\cup S_2}(G),$$

and for $G \sim \mu_{\text{Yes}}$, since $G$ is a clique, we have

$$C_{S_1}(G) \wedge C_{S_2}(G) = C_{S_1\cup S_2}(G).$$

Consider $f \wedge g$ 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 $f \wedge g$ 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 $|\mathcal{T}| \leqslant |\mathcal{T}_1||\mathcal{T}_2| \leqslant t^2$, we need to apply Lemma 13.7 at most $t^2 - t$ times, replacing the sunflowers with their cores. This will create an error probability of at most
$$t^2 2^{-p} \leqslant 2^{-2\ell}.$$
- 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 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 13.5 (II), this creates an error with probability of at most
$$|\mathcal{T}|t^{-2}2^{-2\ell} \leqslant 2^{-2\ell}.$$

Adding these two cases, we see that the probability of error is at most

$$2^{-2\ell} + 2^{-2\ell} = 2^{-2\ell+1} \leqslant 2^{-\sqrt{k}-2}.$$

**Proof Theorem 13.2:** Note that $n^{1/12} = \sqrt{k}$. If there is a monotone circuit of size $M \leqslant 2^{n^{1/12}}$ that computes $\text{Clique}_k$, then by Lemma 13.8 there exists a collection $\mathcal{T}$ of at most $t$ subsets of $V$, each of size at most $\ell$, such that $\widetilde{C}(G) = \bigvee_{S\in\mathcal{T}} C_S(G)$ satisfies

$$\Pr_{G\sim\mu_{\text{Yes}}}[\widetilde{C}(G) = \text{No}] \leqslant \frac{1}{4} \qquad \text{and} \qquad \Pr_{G\sim\mu_{\text{No}}}[\widetilde{C}(G) = \text{Yes}] \leqslant \frac{1}{4}.$$

By Lemma 13.5 (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 $\mathsf{P}$, 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 $\mathsf{P}$. 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 $\mathsf{P}$ 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 $\mathsf{P}$ via semidefinite programming.

## Exercises

**Exercise 13.1.** Consider a circuit of the form $C(G) = \vee_{S \in \mathcal{T}} C_S(G)$ for solving the $k$-clique problem. Let $\mu_{\text{Yes}}$ and $\mu_{\text{No}}$ be defined as in the lecture. Prove that if

$$\mathop{\mathbf{Pr}}_{G\sim\mu_{\text{Yes}}}[C(G) = \text{No}] \leqslant \frac{1}{4} \qquad \text{and} \qquad \mathop{\mathbf{Pr}}_{G\sim\mu_{\text{No}}}[C(G) = \text{Yes}] \leqslant \frac{1}{4},$$

then $|\mathcal{T}| \geqslant \Omega\left(\left(\frac{n}{2k}\right)^{\sqrt{k}}\right)$.

**Exercise 13.2.** Optimize the parameters in the proof of Theorem 13.2 to show that as long as $k \leqslant n^{\frac{1}{3}-\varepsilon}$ 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 13.3.** 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

$$\mathop{\mathbf{Pr}}_{x}[\phi(x) \neq \psi(x)] \leqslant \varepsilon.$$

**Exercise 13.4.** 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 $\psi(x) \geqslant \phi(x)$ for all $x$, and

$$\mathop{\mathbf{Pr}}_{x}[\phi(x) \neq \psi(x)] \leqslant \varepsilon.$$

**Exercise 13.5.** Obviously, $\text{Maj} \in \mathsf{P}$ 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 $\mathbf{Pr}[B_i = 1] = p = \frac{1}{2} + \delta$ for $\delta \in [-0.5, 0.5]$. Prove that

$$\mathbf{Pr}[\text{Maj}_3(B_1, B_2, B_3) = 1] = p + \frac{1}{2}(1 - 4\delta^2)\delta.$$

   Note $1 - 4\delta^2 \geqslant 0$, 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 $\mathbf{Pr}[B = 1] \geqslant \frac{1}{2}$ if $\text{Maj}(x) = 1$, and otherwise $\mathbf{Pr}[B = 1] \leqslant \frac{1}{2} - \frac{1}{n}$. Construct a random polynomial-size $O(\log(n))$-depth circuit $C$, using only $\text{Maj}_3$ gates such that for all $x$,

$$\mathop{\mathbf{Pr}}_{C}[\text{Maj}(x) = C(x)] > 1 - \frac{1}{2^n}.$$

3. Conclude that there is a polynomial size monotone circuit $C$ of depth $O(\log(n))$ such that for all $x$, we have $\text{Maj}(x) = C(x)$.

# Chapter 14

# Razborov-Rudich: Natural Proof Barrier

We have already seen that *relativization* is a barrier to answering the question of $\mathsf{P} \neq \mathsf{NP}$. 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 $\mathsf{P}$ and $\mathsf{NP}$, as there are oracles $A$ and $B$ such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$.

The non-uniform approach to circuit complexity overcomes this barrier. Polynomial-time algorithms can be converted to polynomial size $(\neg, \vee, \wedge)$-circuits, which 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 of this rapid progress, curiously, they, and all subsequent results, fell short of obtaining any nontrivial lower bounds for general circuits, particularly proving that $\mathsf{P} \neq \mathsf{NP}$.

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 $\mathsf{P} \neq \mathsf{NP}$. They dubbed such proofs as *natural proofs.* Loosely speaking, a lower-bound proof is natural if applied 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 $\mathsf{P} \neq \mathsf{NP}$ under reasonable complexity-theoretic assumptions.

## What are *natural proofs*?

Almost all known circuit lower bounds follow the following approach:

- We come up with a simple-to-analyze *property* $\mathcal{P}$ (e.g., having large total influence, not approximable by a low-degree polynomial, not simplifying under random restrictions, etc.), which we interpret as a notion of being complex.
- We show that our function of choice, say $\mathsf{SAT}$, satisfies this property and thus is complex.
- We show that functions computable by circuits in the class $C$ are not complex (i.e., do not satisfy $\mathcal{P}$) and thus cannot include our function.

Here a *property* $\mathcal{P}$ just means a subset of all Boolean functions $f : \{0,1\}^n \to \{0,1\}$, and when we say $f$ satisfies $\mathcal{P}$, we mean $f \in \mathcal{P}$. Obviously, for the above approach to be sensible, $\mathcal{P}$ must be simpler and easier to understand than the property of not 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 $\mathcal{P}$ 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 random function has a non-negligible chance of being complex: Formally, there exists a constant $c > 0$ such that for every sufficiently large $n$, a randomly selected function $f : \{0,1\}^n \to \{0,1\}$ satisfies
$$\mathbf{Pr}[f \in \mathcal{P}] \geqslant c.$$

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.*

**Example 14.1.** If $\mathcal{P} := \{f \ : \ I_f \geqslant n/4\}$, then $\mathcal{P}$ is constructive because we can compute $I_f$ in $2^{O(n)}$, and $\mathcal{P}$ is large because $\mathbb{E}_f[I_f] = n/2$, and thus $\mathbf{Pr}_f[I_f \geqslant n/4] \geqslant \frac{1}{2}$.

As we have seen earlier in Theorem 11.22, every function that can be computed with an $\mathsf{AC}$ 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, $\textsc{Parity}_n \in \mathcal{P}$, and thus does not belong to the set of functions that have $\mathsf{AC}$ circuits of size $n^{O(1)}$ and depth $O(1)$.

The above proof is a *natural proof* for the $\textsc{Parity} \notin \mathsf{AC}^0$. □

## The Razborov-Rudich Theorem

In [RR97] Razborov and Rudich state:

> 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 $\mathsf{SAT}$, or some other function in $\mathsf{NP}$, has “high” discrepancy.
>
> Such 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 prove that natural proofs of general circuit lower bounds are unlikely in the following sense. Any natural proof of a general circuit lower-bound implies 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 all cryptography 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 14.2.** A function $f : \{0,1\}^n \to \{0,1\}^m$ is a one-way function if

- Given $x$, $f(x)$ can be computed in $\mathrm{poly}(|x|)$ time.
- For all probabilistic polynomial time algorithms $M_R$, we have
$$\mathop{\mathbf{Pr}}_{\substack{x\in\{0,1\}^n\\ R}}[M_R(f(x)) \in f^{-1}(f(x))] \leqslant \frac{1}{n^{\omega(1)}}.$$

Cryptography is based on the assumption that functions such as integer factorization are one-way functions. Recall that $\mathrm{size}(n^k)$ denotes the set of functions that can be computed by a circuit of size $n^k$.

**Theorem 14.3** (Razborov-Rudich [RR97])**.** *Assuming that one-way functions exist, no natural proof can show*
$$\mathsf{NP} \not\subseteq \mathrm{size}(n^k),$$
*for all* $k \in \mathbb{N}$.

## 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 $G : \{0,1\}^\ell \to \{0,1\}^m$ 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

$$\Pr_{R\in\{0,1\}^m}[M_R(x) = \textsc{Accept}] \approx \Pr_{y\in\{0,1\}^\ell}[M_{G(y)}(x) = \textsc{Accept}].$$

On the left-hand side, we are using a truly random $R$, whereas on the right-hand side, we are using an artificially produced 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 14.4** (Pseudo-Random Generator)**.** A function $G : \{0,1\}^\ell \to \{0,1\}^m$ is an $\varepsilon$-PRG against a class $C$ if for every $T \in C$,

$$\left| \Pr_{y\in\{0,1\}^\ell}[T(G(y)) = 1]] - \Pr_{R\in\{0,1\}^m}[T(R) = 1] \right| \leqslant \varepsilon.$$

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 14.5** ([HsILL99])**.** *If one-way functions exist, there is a sequence of pseudo-random generators $G_k : \{0,1\}^k \to \{0,1\}^{2k}$, computable in polynomial time in $k$ such that for all $c > 0$, all circuits $T$ of polynomial size $S = k^c$ satisfy*

$$\left| \Pr_{y\in\{0,1\}^k}[T(G(y)) = 1]] - \Pr_{R\in\{0,1\}^{2k}}[T(R) = 1] \right| \leqslant \frac{1}{S},$$

*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 $2^{O(m^\varepsilon)}$ that, in addition to its usual randomness, also has oracle access to a random string $R : \{0,1\}^m \to \{0,1\}$. Here, we interpret $R : \{0,1\}^m \to \{0,1\}$ as an additional (randomness) string of length $2^m$. Instead of providing $R$ to the algorithm on a tape, we allow the algorithm to have oracle access to $R$ and query any coordinate of $R$ without having to traverse the randomness tape. During its computation, $A$ can query the values of $R(x)$ for different points $x \in \{0,1\}^m$. We want to construct a small set of functions $R_s$ such that $A$ cannot distinguish between a completely random string $R : \{0,1\}^m \to \{0,1\}$ and a string selected randomly from our much smaller set of strings $R_s : \{0,1\}^m \to \{0,1\}$. Goldreich, Goldwasser, and Micali [GGM85] show that this is possible.

**Theorem 14.6** (Goldreich, Goldwasser, Micali [GGM85])**.** *If one-way functions exist, there exists a collection of functions $R_s : \{0,1\}^m \to \{0,1\}$ for $s \in \{0,1\}^m$ such that*

- *Given $s, x \in \{0,1\}^m$, we can compute $R_s(x)$ in* $\mathrm{poly}(m)$ *time.*

- *For any $2^{O(m^\varepsilon)}$-time probabilistic algorithm $A$, where $\varepsilon > 0$ is an absolute constant, we have*

$$\left| \mathop{\mathbf{Pr}}_{s\in\{0,1\}^m}\left[A^{R_s} = \textsc{Accept}\right] - \mathop{\mathbf{Pr}}_{R:\{0,1\}^m\to\{0,1\}}\left[A^R = \textsc{Accept}\right]\right| = o(1),$$

*where the probabilities are also over the randomness in $A$.*

Note that in Theorem 14.6, on the left-hand side, we randomly choose a string from only $2^m$ strings $R_s$, while on the right-hand side, we are choosing $R$ out of all the $2^{2^m}$ possible $R : \{0,1\}^m \to \{0,1\}$.

# The proof of Razborov-Rudich Theorem

Finally, we are ready to prove Theorem 14.3. Consider a *natural property* $\mathcal{P}$ that is useful for proving that some function, such as $\mathsf{SAT}$, does not belong to the circuit classes $\mathrm{size}(n^k)$ for all $k \in \mathbb{N}$. We have

- **Constructivity**: It can be decided in time $2^{O(n)}$ whether $f : \{0,1\}^n \to \{0,1\}$ has the property.
- **Largeness**:
$$\mathop{\mathbf{Pr}}_f[f \in \mathcal{P}] \geqslant \delta.$$
- **Usefulness**: For every $k$, for sufficiently large $n$,
$$\mathcal{P} \cap \mathrm{size}(n^k) = \emptyset.$$

Let $n = m^{\varepsilon/2} < m$, where $\varepsilon$ is as in Theorem 14.6, and consider the functions $R_s : \{0,1\}^m \to \{0,1\}$ provided by the theorem. For $s \in \{0,1\}^m$, define

$$g_s : \{0,1\}^n \to \{0,1\} \qquad \text{as} \qquad g_s(x) = R_s(x0^{m-n}).$$

Note that given $s \in \{0,1\}^m$ and $x \in \{0,1\}^n$, by the first part of Theorem 14.6 the value of $g_s(x) = R_s(x0^{m-n})$ can be computed in time $m^{O(1)} = n^{O(1)}$. Consequently, there is a $\mathrm{size}(n^{O(1)})$ circuit that given $x, s$ computes $g_s(x)$. Hence, we have the following by the usefulness of $\mathcal{P}$.

$$g_s \notin \mathcal{P} \qquad \text{for every } s \in \{0,1\}^m, \tag{14.1}$$

while by the largeness property

$$\mathop{\mathbf{Pr}}_g[g \in \mathcal{P}] \geqslant \delta \qquad \text{where } g : \{0,1\}^n \to \{0,1\} \text{ is chosen randomly.} \tag{14.2}$$

This suggests a probabilistic algorithm for distinguishing a randomly selected $R_s : \{0,1\}^m \to \{0,1\}$ from a randomly selected function from all the functions $R : \{0,1\}^m \to \{0,1\}$, and thus contradicting the assertion of Theorem 14.3.

Namely, given a function $R : \{0,1\}^m \to \{0,1\}$ from an unknown distribution which is either uniform over the set of all $R_s$ or uniform over the set of all $\{0,1\}^m \to \{0,1\}$, we look at the function $g : \{0,1\}^n \to \{0,1\}$ defined as $g(x) = R(x0^{m-n})$. By the constructivity 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 $R$ was selected from the set $\{R_s \ : \ s \in \{0,1\}^m\}$, then by (14.1)

$$\mathbf{Pr}[A^R = \textsc{Accept}] = 0.$$

On the other hand, if $R$ was selected from all of the functions $\{0,1\}^m \to \{0,1\}$, then note that $g : \{0,1\}^n \to \{0,1\}$ is also a completely random function, and thus by (14.2), we have

$$\mathbf{Pr}[A^R = \textsc{Accept}] \geqslant \delta.$$

In other words, we have algorithm $A$ that runs in time $2^{O(m^{\varepsilon/2})}$ such that

$$\left| \mathop{\mathbf{Pr}}_{s\in\{0,1\}^m}\left[A^{R_s} = \textsc{Accept}\right] - \mathop{\mathbf{Pr}}_{R:\{0,1\}^m\to\{0,1\}}\left[A^R = \textsc{Accept}\right]\right| \geqslant \delta,$$

contradicting Theorem 14.6.

## Concluding Remarks

The natural proof framework 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 is through the arithmetization technique on interactive proofs. This technique is used to prove some lower bounds on interactive proof classes, for example, Santhanam [San09] and Vindochandran [Vin04]. To curb the power of this technique, Aaronson and Wigderson [AW08] defined the algebrization barrier, a generalization of relativization that incorporates proofs that use arithmetization. They show that proofs that algebrize are still too weak to resolve the $\mathsf{P} \neq \mathsf{NP}$ 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 delivered relatively few lower bounds for rather weak circuit classes.

## Exercises

**Exercise 14.1.** Prove that the Razborov-Smolensky proof for $\textsc{Maj} \notin \mathsf{AC}^0[\oplus]$ is a natural proof.

# Chapter 15

# Fourier analysis and Polynomial Representations

Consider a Boolean function

$$f : \{\textsc{False}, \textsc{True}\}^n \to \{\textsc{False}, \textsc{True}\}.$$

Identifying $\{\textsc{False}, \textsc{True}\}$ 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 Ableian 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 $S \subseteq \{1,\dots,n\}$, 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:

$$\langle x^S, x^T\rangle = \mathbb{E}\left[x^S x^T\right] = \mathbb{E}\left[x^{S\Delta T}\right] = \begin{cases} 1 & S = T \\ 0 & S \neq T \end{cases},$$

where $S\Delta T = (S\setminus T)\cup(T\setminus S)$ 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

$$f(x) = \sum_{S\subseteq\{1,\dots,n\}} \widehat{f}(S)x^S,$$

where the coefficients $\widehat{f}(S) \in \mathbb{R}$ are given by

$$\widehat{f}(S) = \langle f, x^S\rangle = \mathbb{E}[f(x)x^S].$$

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

$$\sum_{S\subseteq\{1,\dots,n\}} \widehat{f}(S)x^S = \sum_{S\subseteq\{1,\dots,n\}} \widehat{f}(S)\prod_{i\in S}(2y_i - 1) = \sum_{S\subseteq\{1,\dots,n\}} \left(\sum_{T\supseteq S} 2^{|S|}(-1)^{|T\setminus S|}\widehat{f}(T)\right) y^S,$$

and conversely

$$\sum_{S\subseteq\{1,\dots,n\}} a_S y^S = \sum_{S\subseteq\{1,\dots,n\}} a_S \prod_{i\in S}\frac{x_i+1}{2} = \sum_{S\subseteq\{1,\dots,n\}} \left(\sum_{T\supseteq S} 2^{-|T|}a_T\right) x^S,$$

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 $\widehat{f}(S) \neq 0$ 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:

$$\langle f, g\rangle := \sum_S \widehat{f}(S)\widehat{g}(S).$$

Note that

$$\mathbb{E}[f] = \widehat{f}(\emptyset),$$

and by the Parseval identity

$$\mathrm{Var}[f] = \mathbb{E}[f^2] - \mathbb{E}[f]^2 = \langle f, f\rangle - \widehat{f}(\emptyset)^2 = \sum_{S\neq\emptyset} \widehat{f}(S)^2,$$

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 $f : \{-1,1\}^n \to \{0,1\}$, we have

$$I_i(f) = \mathbf{Pr}[f(x) \neq f(x \oplus e_i)] = \mathbb{E}\,|f(x) - f(x \oplus e_i)|^2 = \mathbb{E}\left(\sum_{S:i\in S} 2\widehat{f}(S)x^S\right)^2 = \sum_{S:i\in S} 4\widehat{f}(S)^2,$$

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 $f$

$$I(f) = \sum_S 4|S||\widehat{f}(S)|^2.$$

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 15.1.** *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 $\{\textsc{False}, \textsc{True}\}^n$. Hence Boolean functions $f : \{\textsc{False}, \textsc{True}\}^n \to \{0, 1\}$ 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

$$\chi : G \to \mathbb{T}, \tag{15.1}$$

where $\mathbb{T}$ is the unit complex circle.

**Theorem 15.2.** *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 $\widehat{G}$.*

*Proof.* The principal character 1 is the identity of $\widehat{G}$. 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 $\chi^{-1} := \frac{1}{\chi} = \overline{\chi}$ is also a character. □

The group $\widehat{G}$ 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 A) 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 15.3.** 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 15.4.** *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 15.5.** *The characters of a finite Abelian group $G$ are orthonormal functions in $L_2(G)$.*

*Proof.* It follows from (15.1) that every $\chi\in\widehat{G}$ satisfies

$$\|\chi\|_2^2=\mathbb{E}\left[|\chi(x)|^2\right]=\mathbb{E}[1]=1.$$

So characters are unit vectors in $L_2(G)$. It remains to verify the orthogonality. Let $\chi\neq\xi$ be two different characters. Then $\chi\overline{\xi}=\chi\xi^{-1}$ is a non-principal character of $G$ (why?). Hence by Lemma 15.4, we have

$$\langle\chi,\xi\rangle=\mathbb{E}\left[\chi(x)\overline{\xi}(x)\right]=\mathbb{E}\left[\chi\overline{\xi}(x)\right]=0.$$

□

So far we have discussed the Pontryagin dual of $G$ in an abstract manner. Since finite Abelian groups have simple structures (Theorem 15.1), 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

$$\chi_a:x\mapsto e^{\frac{2\pi i}{N}ax}.$$

Let us verify that $\chi_a$ is actually a character. Indeed $\chi_a(0)=e^{\frac{2\pi i}{N}0}=e^0=1$, and since $e^{2\pi i}=1$, we have

$$\chi_a(x)\chi_a(y)=e^{\frac{2\pi i}{N}ax}e^{\frac{2\pi i}{N}ay}=e^{\frac{2\pi i}{N}a(x+y\pmod N)}=\chi_a(x+y).$$

Note that $L_2(G)$ is $|G|$-dimensional, and hence by Lemma 15.5, $G$ 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 $\widehat{G}$ 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

$$\chi_a:x\mapsto\prod_{j=1}^{k}e^{\frac{2\pi i}{N_j}a_jx_j}.$$

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 15.5 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 15.6.** *If $G$ is a finite Abelian group, then the characters of $G$ form an orthonormal basis for $L_2(G)$. Furthermore we have $\widehat{G}\cong G$.*

Theorem 15.6 shows that $G$ is isomorphic to its dual $\widehat{G}$, 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 $f=\sum_{a\in G}\widehat{f}(a)\chi_a$. The corresponding coefficients $\widehat{f}(a)\in G$ are referred to as the Fourier coefficients. This leads to the definition of the Fourier transform.

**Definition 15.7.** The Fourier transform of a function $f:G\to\mathbb{C}$ is the unique function $\widehat{f}:\widehat{G}\to\mathbb{C}$ defined as

$$\widehat{f}(\chi)=\langle f,\chi\rangle=\mathbb{E}f(x)\overline{\chi(x)}.$$

We will often use the notation $\widehat{f}(a)$ to denote $\widehat{f}(\chi_a)$.

Let us state a simple example of the Fourier transform of a function on $\mathbb{Z}_2^n$.

**Example 15.8.** Let $f : \mathbb{Z}_2^n \to \mathbb{C}$ be the parity function $f : x \mapsto \sum_{i=1}^n x_i (\text{mod } 2)$. Then

$$\widehat{f}(0) = \mathbb{E} f(x)\chi_0 = \mathbb{E} f(x) = \frac{1}{2}.$$

We also have

$$\widehat{f}(1,\dots,1) = \mathbb{E} f(x)(-1)^{\sum_{j=1}^n x_j} = -\frac{1}{2},$$

as $f(x) = 1$ if and only if $\sum_{j=1}^n x_j = 1(\text{mod } 2)$. Next consider $a \in \mathbb{Z}_2^n$ with $a \neq (1,\dots,1)$ and $a \neq 0$. Let $j_0, j_1$ be such that $a_{j_0} = 0$ and $a_{j_1} = 1$. We have (why?)

$$\widehat{f}(a) = \mathbb{E} f(x)\chi_a(x) = \frac{1}{2}\mathbb{E}\left[f(x)\chi_a(x) + f(x + e_{j_0} + e_{j_1})\chi_a(x + e_{j_0} + e_{j_1})\right],$$

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 $\widehat{f}(a) = 0$ for every $a \in \mathbb{Z}_2^n$ satisfying $a \neq (1,\dots,1)$ and $a \neq 0$.

The Fourier transform is a linear operator: $\widehat{\lambda f + g} = \lambda \widehat{f} + \widehat{g}$, and we have the following easy observation.

**Lemma 15.9.** *The Fourier transform considered as an operator from $L_1(G)$ to $L_\infty(\widehat{G})$ is norm decreasing:*

$$\|\widehat{f}\|_\infty \leqslant \|f\|_1.$$

*Proof.* By (15.1) for every $a \in G$, we have

$$|\widehat{f}(a)| = \left|\mathbb{E} f(x)\overline{\chi_a(x)}\right| \leqslant \mathbb{E}|f(x)||\overline{\chi_a(x)}| = \mathbb{E}|f(x)| = \|f\|_1.$$

□

The Fourier coefficient $\widehat{f}(0)$ is of particular importance as

$$\widehat{f}(0) = \mathbb{E}[f(x)].$$

So if $\mathbf{1}_A$ is the indicator function of a subset $A \subseteq G$, then $\widehat{\mathbf{1}_A}(0) = \frac{|A|}{|G|}$ corresponds to the density of $A$.

It follows from the fact that the characters from an orthonormal basis for $L_2(G)$ that

$$f = \sum_{a \in G} \widehat{f}(a)\chi_a,$$

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 15.10.** *If $H$ is a subgroup of $G$, then for every $a \in G$, we have*

$$\widehat{\mathbf{1}_H}(a) = \begin{cases} |H|/|G| & a \in H^\perp \\ 0 & a \notin H^\perp \end{cases}$$

*Proof.* If $a \in H^\perp$, then

$$\widehat{\mathbf{1}_H}(a) = \langle \mathbf{1}_H, \chi_a \rangle = \mathbb{E}\mathbf{1}_H(x)\overline{\chi_a(x)} = \mathbb{E}\mathbf{1}_H(x) = |H|/|G|.$$

On the other hand if $a \notin H^\perp$, then there exists $y \in H$ such that $\chi_a(y) \neq 1$. Then

$$\sum_{z \in H} \overline{\chi_a(z)} = \chi_a(y) \sum_{z \in H} \overline{\chi_a(z - y)} = \chi_a(y) \sum_{z \in H} \overline{\chi_a(z)},$$

which shows that $\sum_{z \in H} \overline{\chi_a(z)} = 0$. Hence

$$\widehat{\mathbf{1}_H}(a) = \mathbb{E}\mathbf{1}_H(x)\overline{\chi_a(x)} = \frac{1}{|G|}\sum_{z \in H} \mathbb{E}\mathbf{1}_H(z) = 0.$$

□

**Remark 15.11.** It follows from Lemma 15.10 that if $A = y + H$ is a coset of $H$ in $G$ (i.e. $H$ is a subgroup of $G$ and $y \in G$), then for every $a \in G$,

$$\begin{aligned}\widehat{\mathbf{1}_A}(a) &= \mathbb{E}\mathbf{1}_A(x)\overline{\chi_a(x)} = \mathbb{E}\mathbf{1}_H(x-y)\overline{\chi_a(x)} = \mathbb{E}\mathbf{1}_H(x)\overline{\chi_a(x+y)} = \overline{\chi(y)}\widehat{\mathbf{1}_H}(a) \\ &= \begin{cases} \overline{\chi(y)}|H|/|G| & a \in H^\perp \\ 0 & a \notin H^\perp \end{cases}\end{aligned}$$

**Example 15.12.** Let us revisit Example 15.8 in light of Remark 15.11. Note that $H = \{x \in \mathbb{Z}_2^n : \sum_{i=1}^n x_i = 0 \pmod 2\}$ is a subgroup of $\mathbb{Z}_2^n$. Now the function $f$ defined in Example 15.8 is the indicator function of $A = e_1 + H$. Note that

$$H^\perp = \{a : (-1)^{\sum_{i=1}^n x_i a_i} = 1 \ \forall x \in H\} = \{(0,\ldots,0),(1,\ldots,1)\}.$$

Hence

$$\widehat{f}(a) = \widehat{\mathbf{1}_A}(a) = \begin{cases} \overline{\chi_a(e_1)}|H|/|G| & a \in H^\perp \\ 0 & a \notin H^\perp \end{cases}$$

We conclude that $\widehat{f}(0) = 1/2$ and $\widehat{f}(1,\ldots,1) = -1/2$, and $\widehat{f}(a) = 0$ for every $a \in \mathbb{Z}_2^n$ satisfying $a \neq (1,\ldots,1)$ and $a \neq 0$.

Next we prove the Parseval's identity, a very simple but extremely useful fact in Fourier analysis.

**Theorem 15.13** (Parseval)**.** *For every $f \in L_2(G)$,*

$$\|f\|_2^2 = \sum_{a \in G} |\widehat{f}(a)|^2.$$

*Proof.* We have

$$\|f\|_2^2 = \langle f, f\rangle = \left\langle \sum_{a\in G} \widehat{f}(a)\chi_a, \sum_{b \in G}\widehat{f}(b)\chi_b \right\rangle = \sum_{a,b\in G} \widehat{f}(a)\overline{\widehat{f}(b)}\langle \chi_a, \chi_b\rangle.$$

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*:

$$\langle f, g\rangle = \sum_{a \in G} \widehat{f}(a)\overline{\widehat{g}(a)}.$$

As the first example of an application of the Parseval identity, let us show that for every subgroup $H$ of $G$, we have

$$|H||H^\perp| = |G|. \tag{15.2}$$

Indeed by Lemma 15.10, we have

$$\frac{|H|}{|G|} = \mathbb{E}\mathbf{1}_H = \mathbb{E}\mathbf{1}_H^2 = \langle \mathbf{1}_H, \mathbf{1}_H\rangle = \|\mathbf{1}_H\|_2^2 = \sum_{a\in G} |\widehat{\mathbf{1}_H}(a)|^2 = \sum_{a \in H^\perp} (|H|/|G|)^2 = \frac{|H|^2|H^\perp|}{|G|^2}$$

which simplifies to (15.2).

Next we introduce the important notion of *convolution*.

**Definition 15.14.** Let $G$ be a finite Abelian group. For two functions $f, g : G \to \mathbb{C}$, we define their convolution $f * g : G \to \mathbb{C}$ as

$$f * g(x) = \mathbb{E}_{y \in G}[f(x-y)g(y)].$$

Note that $f * g(x)$ 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 $f * \mathbf{1}_A(x)$ 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 $f * \mathbf{1}_A(x)$ 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 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 $\{x \in \mathbb{Z}_2^n : \sum_{i=1}^n x_i \leqslant r\} \subseteq \mathbb{Z}_2^n$.

**Lemma 15.15.** *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*

$$\operatorname{supp}(f * g) \subseteq \operatorname{supp}(f) + \operatorname{supp}(g).$$

**(e)** *We have*

$$\|f * g\|_\infty \leqslant \|f\|_1 \|g\|_\infty.$$

**(f)** *More generally, if $p$ and $q$ are conjugate exponents, then*

$$\|f * g\|_\infty \leqslant \|f\|_p \|g\|_q.$$

**(g)** *We have*

$$\|f * g\|_1 \leqslant \|f\|_1 \|g\|_1.$$

*Proof.* **(a)** For every $x \in G$, we have

$$f * g(x) = \mathbb{E}_y[f(x-y)g(y)] = \mathbb{E}_y[f(x-y)g(x-(x-y))] = \mathbb{E}_z[f(z)g(x-z)] = g * f(x).$$

**(b)** By Part **(a)**,

$$\begin{aligned} (f * g) * h(x) &= (g * f) * h(x) = \mathbb{E}_z \mathbb{E}_y [g(x-z-y)f(y)]h(z) = \\ &= \mathbb{E}_{y,z} g(x-z-y)f(y)h(z) = (h * g) * f(x) = f * (g * h)(x). \end{aligned}$$

**(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

$$|f * g(x)| \leqslant \mathbb{E}_{y \in G}|f(x-y)||g(y)| \leqslant \left(\mathbb{E}|f(x-y)|^p\right)^{1/p} \left(\mathbb{E}|g(y)|^q\right)^{1/q} = \left(\mathbb{E}|f(y)|^p\right)^{1/p} \|g\|_q = \|f\|_p \|g\|_q.$$

**(g)** We have

$$\|f * g\|_1 = \mathbb{E}_x |f * g(x)| \leqslant \mathbb{E}_{x,y}|f(x-y)||g(y)| = \mathbb{E}_{z,y}|f(z)||g(y)| = \mathbb{E}_z|f(z)|\mathbb{E}_y|g(y)| = \|f\|_1 \|g\|_1.$$

□

The relevance of the Fourier transform to convolution lies in the following lemma.

**Lemma 15.16.** *If $f, g : G \to \mathbb{C}$, then*

$$\widehat{f * g} = \widehat{f} \cdot \widehat{g}.$$

*Proof.* We have

$$\begin{aligned} \widehat{f * g}(a) &= \mathbb{E}_x f * g(x)\overline{\chi_a(x)} = \mathbb{E}_x \left(\mathbb{E}_y f(x-y)g(y)\right) \overline{\chi_a(x)} = \mathbb{E}_{x,y} f(x-y)g(y)\overline{\chi_a(x-y)\chi_a(y)} \\ &= \mathbb{E}_{z,y} f(z)g(y)\overline{\chi_a(z)\chi_a(y)} = \mathbb{E}_z f(z)\overline{\chi_a(z)}\mathbb{E}_y g(y)\overline{\chi_a(y)} = \widehat{f}(a) \cdot \widehat{g}(a). \end{aligned}$$

□

Note that Lemma 15.16 in particular shows that

$$\mathbb{E}f(x)\mathbb{E}g(x) = \widehat{f}(0)\widehat{g}(0) = \widehat{f * g}(0) = \mathbb{E}f * g(x).$$

We also have the dual version of Lemma 15.16,

$$\widehat{f \cdot g}(x) = \sum_{y \in G} \widehat{f}(x-y)\widehat{g}(y), \tag{15.3}$$

which converts point-wise product back to convolution.

For a function $h : G \to \mathbb{C}$, define $\tilde{h} : G \to \mathbb{C}$ as $h : x \mapsto \overline{h(-x)}$. Note that $\tilde{h} = \sum_{a \in G} \overline{\widehat{h}(a)}\chi_a$. Hence it follows from the Parseval identity and Lemma 15.16 that for $f, g, h : G \to \mathbb{C}$, we have

$$\langle f * h, g\rangle = \langle f, g * \tilde{h}\rangle = \sum_{a \in G} \widehat{f}(a)\widehat{h}(a)\overline{\widehat{g}(a)}.$$

## Infinite Ableian 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 $\widehat{G}$ that is called the Pontryagin dual of $G$.

In the case of $\mathbb{R}$, the characters are

$$\chi_r : x \mapsto e^{-2\pi rx},$$

for $r \in \mathbb{R}$. Note that $\chi_0$ is the principal character, and $\chi_t\chi_t = \chi_{r+t}$, and as a result $\widehat{\mathbb{R}} \cong \mathbb{R}$. Other examples are $\widehat{\mathbb{Z}} \cong \mathbb{T}$, and $\widehat{\mathbb{T}} \cong \mathbb{Z}$. 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:

$$\widehat{f}(\chi) = \int f(x)\chi(x)d\mu_G(x).$$

Note that this integral is well-defined precisely because

$$\int f(x)\chi(x)d\mu_G(x) \leqslant \int |f(x)||\chi(x)|d\mu_G = \int |f(x)|d\mu_G < \infty.$$

So it is important that $f \in L_1(G)$. This gives us the Fourier transform

$$\mathcal{F} : L_1(G) \to L_1(\widehat{G}),$$

that maps the function $f$ to its Fourier coefficients:

$$\mathcal{F} : f \mapsto \widehat{f}.$$

For example in the case of $\mathbb{R}$, we get the familiar formula

$$\widehat{f}(r) = \int_{-\infty}^{\infty} f(x)e^{-2\pi rx}dx.$$

The Fourier inversion theorem says that we have

$$f = \int_{-\infty}^{\infty} \widehat{f}(\chi)\chi(x)d\nu_{\widehat{G}}(\chi),$$

where $\nu_{\widehat{G}}$ is the Haar measure of the dual group $\widehat{G}$. For example, for $\mathbb{R}$, we get the familar Fourier inversion formula

$$f(x) = \int_{-\infty}^{\infty} \widehat{f}(r)e^{-2\pi rx}dr,$$

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 homorphisms 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 $\mathbb{Z}_2^n$ 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 $\mathbb{Z}_2^n$ 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 $\mathbb{Z}_2^n$ 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

$$f(x) := \sum_{S \subseteq [n]} \widehat{f}(S) x^S.$$

## Exercises

**Exercise 15.1.** Prove the Identity (15.3).

**Exercise 15.2.** Suppose that for $f : \mathbb{Z}_2^n \to \{0,1\}$ we have $\widehat{f}(S) = 0$ for all $|S| \geqslant 2$ (that is $\deg(f) \leqslant 1$). 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 15.3.** Let $G$ be a finite Abelian group, and $H$ be a subgroup of $G$. Prove that

$$\left(H^\perp\right)^\perp = H.$$

**Exercise 15.4.** Given two subsets $\mathcal{A}, \mathcal{B} \subseteq 2^{[n]}$. For $S \subseteq [n]$, let

$$\textsc{parity}_S(\mathcal{A}) := \{A \in \mathcal{A} \mid |A \cap S| \equiv 0 \bmod 2\},$$

and

$$\textsc{parity}_S(\mathcal{B}) := \{B \in \mathcal{B} \mid |B \cap S| \equiv 0 \bmod 2\}.$$

- Prove that if $\textsc{parity}_S(\mathcal{A}) = \textsc{parity}_S(\mathcal{A})$ for every $S$, then $\mathcal{A} = \mathcal{B}$.
- Suppose $|\textsc{parity}_S(\mathcal{A}) - \textsc{parity}_S(\mathcal{A})| \leqslant \delta$ for every $S$. Bound $|\,(\mathcal{A} \setminus \mathcal{B}) \cup (\mathcal{B} \setminus \mathcal{A})\,|$ in terms of $\delta$.

**Exercise 15.5.** 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

$$\mathbb{E}_{x \in H} f(x) = \sum_{a \in H^\perp} \widehat{f}(\chi_a).$$

**Exercise 15.6.** Let $G$ be a finite Abelian group and $f, g : G \to \mathbb{C}$. Show that for every positive integer $m$,

$$\|f * g\|_m \leqslant \|f\|_1 \|g\|_m.$$

**Exercise 15.7.** Consider a function $f : \mathbb{Z}_2^n \to \{0,1\}$ and its Fourier expansion $f = \sum_{S \subseteq [n]} \widehat{f}(S) \chi_S$. 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 15.8.** Suppose that for $f : \mathbb{Z}_2^n \to \{0,1\}$ we have $\widehat{f}(S) = 0$ for all $|S| > k$ (that is $\deg(f) \leqslant k$). 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 16

# Learning and $\mathsf{AC}^0$ circuits

As we saw in Chapter 15, every function $f : \{-1,1\}^n \to \mathbb{R}$ has a unique representation as a polynomial in variables $x_1, \ldots, x_n$. That is

$$f(x) = \sum_{S \subseteq [n]} \widehat{f}(S) x^S,$$

where $x^S = \prod_{i \in S} x_i$, and $\widehat{f}(S)$ 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 15. In this chapter we will study the Fourier expansion of the functions in $\mathsf{AC}^0$, and see an application of this to the area of theoretical machine learning.

## PAC learning from uniform samples

Consider a class $\mathcal{C}$ of Boolean functions $f : \{-1,1\}^n \to \{0,1\}$. For example $\mathcal{C}$ could be the set of all such $f$ with $\mathrm{dt}(f) \leqslant k$, where $\mathrm{dt}(f)$ refers to the smallest height of a decision-tree computing $f$. Now suppose that we are facing the task of learning a function $f \in \mathcal{C}$ 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 $\mathcal{C}$ but nevertheless it is a good approximation of $f$. That is

$$\Pr_y[h(y) \neq f(y)] \leqslant \varepsilon.$$

Let us see what kind of information about $f$ can be learned from uniform samples. Consider a fixed $S \subseteq \{1, \ldots, n\}$, and recall that

$$\widehat{f}(S) = \langle f, x^S \rangle = \mathbb{E}_x[f(x)x^S].$$

Note that $|f(x)x^S| \leqslant 1$, and thus by Chernoff bound, with high probability, a few samples would suffice to give us a good approximation of this expected value. More precisely, if $x_1, \ldots, x_m \in \{-1,1\}^n$ are sampled uniformly and independently, then with high probability the empirical estimate

$$\widetilde{f}(S) = \frac{1}{m} \sum_{i=1}^{m} f(x_i) x_i^S,$$

will be very close to the actual expected value $\widehat{f}(S) = \mathbb{E}_x[f(x)x^S]$.

**Lemma 16.1.** *Given access to random samples from $f : \{-1,1\}^n \to \{0,1\}$, there is a randomized algorithm which takes as input $S \subseteq \{1, \ldots, n\}$ and $0 \leqslant \delta, \varepsilon \leqslant \frac{1}{2}$, and using at most $O(\log(1/\delta)\varepsilon^{-2})$ samples, outputs an estimate $\widetilde{f}(S)$ for $\widehat{f}(S)$ such that*

$$\Pr[|\widetilde{f}(S) - \widehat{f}(S)| > \varepsilon] \leqslant \delta.$$

Lemma 16.1 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 $\mathcal{C}$. Namely, there is a small set $\mathcal{S} \subseteq \mathcal{P}(\{1,\dots,n\})$ such that most of the mass of the Fourier coefficients is concentrated in $\mathcal{S}$:

$$\sum_{S \notin \mathcal{S}} |\widehat{f}(S)|^2 \leqslant \varepsilon.$$

Compare this to

$$\sum_{S} |\widehat{f}(S)|^2 = \mathbb{E}[|f(y)|^2] = \mathbb{E}f.$$

If this is the case, then we can focus on estimating only those $\widehat{f}(S)$ for $S \in \mathcal{S}$, and obtain a good estimate of $f$ as

$$\sum_{S \in \mathcal{S}} \widetilde{f}(S) x^S \approx f.$$

**Theorem 16.2.** *Suppose that there there is a subset $\mathcal{S} \subseteq \mathcal{P}(\{1,\dots,n\})$ such that for every $f \in \mathcal{C}$, we have*

$$\sum_{S \notin \mathcal{S}} |\widehat{f}(S)|^2 \leqslant \varepsilon.$$

*There is a randomized algorithm that uses at most $O(|\mathcal{S}|\log(|\mathcal{S}|)\varepsilon^{-1})$ samples from an unknown $f \in \mathcal{C}$, and outputs a Boolean function $h : \{0,1\}^n \to \{0,1\}$ such that with probability at least $1-\delta$,*

$$\Pr_x[f(x) \neq h(x)] \leqslant 8\varepsilon.$$

*Proof.* Let $K = |\mathcal{S}|$. We use $O\left(\log(K/\delta)\left(\frac{\sqrt{K}}{\sqrt{\varepsilon}}\right)^2\right) = O\left(\log(K/\delta)K\varepsilon^{-1}\right)$ samples, to estimate $\widehat{f}(S) \approx \widetilde{f}(S)$ for each $S \in \mathcal{S}$. Then for every $S$, we have

$$\Pr\left[|\widehat{f}(S) - \widetilde{f}(S)| > \frac{\sqrt{\varepsilon}}{\sqrt{K}}\right] \leqslant \frac{\delta}{K}.$$

Hence, by the union bound,

$$\Pr\left[|\widehat{f}(S) - \widetilde{f}(S)| > \frac{\sqrt{\varepsilon}}{\sqrt{K}} \text{ for some } S \in \mathcal{S}\right] \leqslant \delta.$$

Let

$$g = \sum_{S \in \mathcal{S}} \widetilde{f}(S) x^S,$$

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 the Parseval identity, we have

$$\mathbb{E}_x[|f(x) - g(x)|^2] = \sum_{S \in \mathcal{S}} |\widehat{f}(S) - \widetilde{f}(S)|^2 + \sum_{S \notin \mathcal{S}} |\widehat{f}(S)|^2 \leqslant K\left(\frac{\sqrt{\varepsilon}}{\sqrt{K}}\right)^2 + \varepsilon \leqslant 2\varepsilon.$$

There is only one issue left to resolve, and that is $g$ is not a Boolean function. However, this can be easily fixed. 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 $|f(x) - h(x)| \leqslant 2|f(x) - g(x)|$. Hence

$$\Pr_x[f(x) \neq h(x)] = \mathbb{E}_x|f(x) - h(x)|^2 \leqslant 4\mathbb{E}_x|f(x) - g(x)|^2 \leqslant 8\varepsilon.$$

□

# Decision Trees

Let $\mathcal{C}$ be the class of Boolean functions $f : \{-1,1\}^n \to \{0,1\}$ 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 $d$.

Therefore, for this class $\mathcal{C}$, we can take $\mathcal{S} = \{S \mid |S| \leqslant d\}$, which is a set of size at most $n^d$. We conclude that this class can be learned in the sense of Theorem 16.2 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 $\mathsf{AC}^0$ circuit is likely to simplify to a function with small decision tree complexity. The small decision tree, in particular, implies a small degree as a polynomial. This observation suggests that the functions in $\mathsf{AC}^0$ must not have a large weight on the coefficients of the monomials of large degrees. Linial, Mansour, and Nisan [LMN93] made this speculation a theorem.

For a positive integer $k$, and a function $f : \{-1,1\}^n \to \mathbb{R}$, we define

$$f^{=k} := \sum_{S:|S|=k} \widehat{f}(S)x^S,$$

and $f^{\leqslant k}$, $f^{\geqslant k}$, $f^{<k}$ and $f^{>k}$ are defined similarly. Note that by the Parseval identity,

$$\|f\|_2^2 = \sum_{k=0}^{n} \|f^{=k}\|_2^2.$$

Since the degree of a decision tree is bounded by its depth, we have the following corollary to the Switching Lemma.

**Corollary 16.3.** *Let $f$ be a Boolean function computed by an* AC *circuit of size $M$ and depth $d$. Choose a random restriction $\rho$ by setting every variable independently to $*$ with probability $p = \frac{1}{10^d s^{d-1}}$, and to 0 and 1 each with probability $\frac{1-p}{2}$. Then*

$$\mathbf{Pr}[\deg(f_\rho) > s] \leqslant M2^{-s}.$$

Finally, we prove the main theorem of this section.

**Theorem 16.4** (Linial, Mansour, Nisan [LMN93])**.** *Let $f$ be a Boolean function computed by an AC circuit of depth $d$ and size $M$, and let $t$ be any integer. Then*

$$\sum_{|S|>t} |\widehat{f}(S)|^2 \leqslant 2M2^{-t^{1/d}/20}.$$

*In particular for $t = (40\log(M/\varepsilon))^d$, we have*

$$\sum_{|S|>t} |\widehat{f}(S)|^2 \leqslant \varepsilon.$$

*Proof.* Consider a random restriction $\rho \in \{-1,1,*\}^n$ with $\mathbf{Pr}[*] = p \leqslant \frac{1}{10^d k^{d-1}}$ for a value of $k$ 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

$$x^S = (x_T)^{S\cap T}(x_{\overline{T}})^{S\setminus T}.$$

Note that $f_{x_T} : \{-1,1\}^{\overline{T}} \to \{0,1\}$ and since

$$f(x) = \sum_{S\subseteq[n]} \widehat{f}(S)x^S = \sum_{S\subseteq[n]} \widehat{f}(S)(x_T)^{S\cap T}(x_{\overline{T}})^{S\setminus T} = \sum_{A\subseteq\overline{T}} \left(\sum_{B\subseteq T} \widehat{f}(A\cup B)x_T^B\right) x_{\overline{T}}^A,$$

we have

$$\widehat{f_{x_T}}(A) = \sum_{B\subseteq T} \widehat{f}(A\cup B)x_T^B,$$

for every $A \subseteq \overline{T}$. Hence, by the Parseval identity, we have

$$\mathbb{E}_{x_T} \left|\widehat{f_{x_T}}(A)\right|^2 = \sum_{B\subseteq T} |\widehat{f}(A\cup B)|^2,$$

which shows that

$$\mathbb{E}_{x_T} \left\|f_{x_T}^{>k}\right\|_2^2 = \mathbb{E}_{x_T} \sum_{\substack{A\subseteq\overline{T}\\ |A|>k}} \left|\widehat{f_{x_T}}(A)\right|^2 = \sum_{\substack{A\subseteq\overline{T}\\ |A|>k}} \sum_{B\subseteq T} |\widehat{f}(A\cup B)|^2 = \sum_{S:|S\cap\overline{T}|>k} |\widehat{f}(S)|^2.$$

Now we use the randomness in $T$. Since $f_{x_T}^{>k} = 0$ if $\deg(f_\rho) \leqslant k$, and that always $\|f_{x_T}^{>k}\|_2^2 \leqslant \|f_{x_T}\|_2^2 \leqslant 1$, we have

$$\mathbb{E}_T \left[ \sum_{S:|S\cap\overline{T}|>k} |\widehat{f}(S)|^2 \right] = \mathbb{E}_T \mathbb{E}_{x_T} \left\| f_{x_T}^{>k} \right\|_2^2 = \mathbb{E}_\rho \left\| f_\rho^{>k} \right\|_2^2 \leqslant \mathbf{Pr}[\deg(f_\rho) > k] \leqslant M2^{-k}, \tag{16.1}$$

where the last inequality follows from Corollary 16.3 as we chose $\mathbf{Pr}[*] = p \leqslant \frac{1}{10^d k^{d-1}}$. Also we can bound the left-hand size of (16.1) from below:

$$\text{L.H.S. of (16.1)} = \sum_{S\subseteq[n]} \mathbf{Pr}[|S\cap\overline{T}| > k] |\widehat{f}(S)|^2 \geqslant \sum_{|S|>t} \mathbf{Pr}[|S\cap\overline{T}| > k] |\widehat{f}(S)|^2.$$

Taking $p = \frac{1}{10t^{(d-1)/d}}$ and $k = t^{1/d}/20$, satisfies $p \leqslant \frac{1}{10^d k^{d-1}}$, and by the Chernoff bound for $|S| > t$, the probability of $|S \cap \overline{T}| > k = pt/2$ is at least $1 - 2e^{\frac{-pt}{12}} \geqslant \frac{1}{2}$. Hence by (16.1), we have

$$\sum_{S:|S|>t} \frac{1}{2} |\widehat{f}(S)|^2 \leqslant M2^{-t^{1/d}/20}.$$

□

## Mansour-Nisan conjecture

Theorem 16.4 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 $\binom{n}{\leqslant t} \leqslant n^t$ 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 16.5** (Mansour [Man95])**.** *Let $f$ be computable by a DNF with at most $t$ terms. For every $\varepsilon$, there exists a subset $\mathcal{S} \subseteq \mathcal{P}(\{1,\dots,n\})$ of size $t^{O(\log 1/\varepsilon)}$ such that*

$$\sum_{S\notin\mathcal{S}} |\widehat{f}(S)|^2 \leqslant \varepsilon.$$

## Exercises

**Exercise 16.1.** Prove that s every $f$ computable by a decision tree with $L$ leaves, we have

$$\sum_S |\widehat{f}(S)| \leqslant L.$$

**Exercise 16.2.** Let $f : \{-1,1\}^n \to \{0,1\}$ be a Boolean function such that

$$\sum_S |\widehat{f}(S)| \leqslant K.$$

Prove that there exists a set $\mathcal{S}$ of size at most $K^2/\varepsilon$ such that

$$\sum_{S\notin\mathcal{S}} \widehat{f}(S)^2 \leqslant \varepsilon.$$

**Exercise 16.3.** Prove that for every monotone $f$ we have

$$\sum_{S:|S|\geqslant I_f/\varepsilon} \widehat{f}(S)^2 \leqslant \varepsilon.$$

**Exercise 16.4.** Prove that for every *monotone* $f$ we have

- $2\widehat{f}(\{i\}) = I_i(f)$ for every $i$.
- We have $I_f \leqslant 2\sqrt{n}$.

# Chapter 17

# 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 $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ defines a communication problem. An input $x \in \mathcal{X}$ is given to Alice, and an input $y \in Y$ 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 the 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, it is labelled with a function $a_v : \mathcal{X} \to \{0,1\}$, which prescribes the bit sent by Alice at this node as a function of her input. Similarly, Bob's nodes are labelled with Boolean functions on $\mathcal{Y}$. Each leaf is labelled 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 $\mathrm{Cost}_\pi(x,y)$. This is exactly the path length 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 $\mathrm{Cost}_\pi(x,y)$ over all inputs $(x,y)$.

$$\mathrm{CC}(\pi) := \max_{x,y} \mathrm{Cost}_\pi(x,y).$$

Every such protocol $\pi$ computes a function $\mathcal{X} \times \mathcal{Y} \to \{0,1\}$, 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 $\mathrm{D}(f)$, is the smallest communication cost of a protocol that computes $f$.

**Example 17.1** (Equality Function, Identity Matrix)**.** Consider the function $\mathrm{EQ} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined as $\mathrm{EQ}(x,y) = 1$ if and only if $x = y$. Note that this function corresponds to the $2^n \times 2^n$ identity matrix. We obviously have $\mathrm{D}(\mathrm{EQ}) \leqslant n+1$ as Alice can send $x$ to Bob by transmitting $n$ bits, and then Bob knowing both $x$ and $y$, can send $\mathrm{EQ}(x,y)$ to Alice. In the next section, we will prove that $\mathrm{D}(\mathrm{EQ}) = n+1$.

Note that the protocol described above works for every function. We have the following observation.

**Proposition 17.2.** *For every function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, we have*

$$\mathrm{D}(f) \leqslant n+1.$$

**Example 17.3** (Parity function)**.** Consider the function $\textsc{Parity} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined as

$$\textsc{Parity}(x,y) = \sum_{i=1}^n x_i + \sum_{i=1}^n y_i \bmod 2.$$

We have $\mathrm{D}(\textsc{Parity}) = 2$ as Alice can send the one bit $\textsc{Parity}(x)$ to Bob, and then Bob can compute $\textsc{Parity}(x,y)$, and send it to Alice.

## Monochromatic rectangles, and Fooling sets

Consider a deterministic communication protocol for computing a function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$. A useful insight into communication complexity is that a bit sent by Alice at a node $v$ corresponds to a partition of the rows into two parts

$a_v^{-1}(0)$ and $a_v^{-1}(1)$, and every bit sent by Bob corresponds to a partition of the columns. Every time Alice sends a bit, we restrict it to a subset of the rows and proceed with the created submatrix. Similarly, Bob's communicated bits restrict the columns. As this process continues, we see that every leaf $\ell$ of the protocol corresponds to a submatrix $\mathcal{X}_\ell \times \mathcal{Y}_\ell$ of $f$. In the context of communication complexity, submatrices are often called *combinatorial rectangles* or simply *rectangles*. Note that if the protocol computes $f$ correctly, then the value of $f$ on all the points in $\mathcal{X}_\ell \times \mathcal{Y}_\ell$ must match the output of the protocol. In particular, $f$ can take only one value on the rectangle. Such rectangles are called *monochromatic rectangles*. We conclude that the leaves of the protocol partition $\mathcal{X} \times \mathcal{Y}$ into monochromatic rectangles.

**Proposition 17.4.** *Every deterministic protocol with $m$ leaves that computes $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, partitions $\mathcal{X} \times \mathcal{Y}$ into $m$ monochromatic rectangles. Consequently, a $c$-bit protocol induces a partition of the matrix $f$ to at most $2^c$ monochromatic rectangles.*

Almost all lower bounds in communication rely on the existence of these monochromatic rectangles. Let us start with the fooling set technique for proving lower bounds on $\mathrm{D}(f)$.

**Definition 17.5** (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 17.6** (Fooling Set Lower-bound)**.** *If $S$ is a $b$-fooling set $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, 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 17.7.** *We have* $\mathrm{D}(\mathrm{EQ}) = n+1$ *for the equality function* $\mathrm{EQ} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$.

*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 $\mathrm{D}(\mathrm{EQ}) \geqslant \lceil \log 2^n + 1 \rceil = n+1$. □

## 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 17.8** (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 labelled with* 1.

**Example 17.9.** Consider the inner product function $\mathrm{IP} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined as

$$\mathrm{IP}(x,y) := \sum_{i=1}^{n} x_i y_i \bmod 2.$$

Let $\mu$ be the uniform probability distribution on $\mathrm{IP}^{-1}(0)$. 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

$$\mathrm{IP}(x,y) = \langle x, y \rangle,$$

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 $\dim(S') + \dim(T') \leqslant n$, which shows that $|R| \leqslant |S'||T'| \leqslant 2^n$, and thus $\mu(R) \leqslant \frac{2^n}{2^{2n-1}} = 2^{1-n}$. We conclude that

$$\mathrm{D}(\mathrm{IP}) \geqslant \log 2^{n-1} = n-1.$$

## 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 $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, then

$$f = \sum_{\ell \in L_1} \mathbf{1}_{R_\ell},$$

where $R_\ell$ is the 1-monochormatic rectangle corresponding to $\ell$. It follows that $\mathbf{rk}(f) \leqslant |L_1|$. We conclude the following theorem.

**Theorem 17.10.** *For every function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, we have*

$$\mathrm{D}(f) \geqslant \log \mathbf{rk}(f).$$

On the other hand, it is not difficult to show that $\mathrm{D}(f) \leqslant \mathbf{rk}(f) + 1$. A major open problem in communication complexity is whether $\mathrm{D}(f)$ and $\log \mathbf{rk}(f)$ are polynomially related.

**Conjecture 17.11** (Long-rank Conjecture)**.** *Is there a constant $c$ such that*

$$\mathrm{D}(f) \leqslant O(\mathbf{rk}(f)^c)?$$

## Non-deterministic Communication complexity

## Exercises

**Exercise 17.1.** 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 17.8 would imply that the number of 1-leaves is at least $|S|$.

**Exercise 17.2.** Prove that $\mathrm{D}(f) \leqslant \mathbf{rk}(f) + 1$.

# Chapter 18

# Randomized Communication complexity

There are two different natural models to define a probabilistic communication protocol. In the *private coin* model, each party has its private 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. In the protocol tree, Alice's nodes are labelled by functions of $R_A$ and $x$, and Bob's nodes are labelled 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 support of this distribution:

$$\mathrm{CC}(\pi_R) = \max_r \mathrm{CC}(\pi_r) = \max_r \max_{x,y} \mathrm{Cost}_{\pi_r}(x,y).$$

In the probabilistic models of computation, three types of error are often considered.

- **Two-sided error (as in BPP):** This is the most important notion of randomized communication complexity. For every $x, y$, we require

  $$\Pr_R[\pi_R(x,y) \neq f(x,y)] \leqslant \varepsilon,$$

  where $\varepsilon$ is a fixed constant that is strictly less than 1/2. Note that $\varepsilon = 1/2$ is achievable by outputting a random bit; hence $\varepsilon$ in the definition must be strictly less than 1/2. It is common to take $\varepsilon = \frac{1}{3}$. 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 $\mathrm{R}_\varepsilon(f)$ and is defined as the smallest communication cost $\mathrm{CC}(\pi_R)$ of a public-coin probabilistic protocol that computes $f$ with two-sided error at most $\varepsilon$. We set $\varepsilon = 1/3$ as the standard error and denote

  $$\mathrm{R}(f) = \mathrm{R}_{\frac{1}{3}}(f).$$

- **One-sided error (as in RP):** 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

  $$\Pr_R[\pi_R(x,y) \neq f(x,y)] \leqslant \varepsilon.$$

  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 outputting 1 only when at least one of the repetitions outputs 1. We denote by $\mathrm{R}^1_\varepsilon(f)$ the smallest $\mathrm{CC}(\pi_R)$ over all public-coin protocols $\pi_R$ with one-sided error of at most $\varepsilon$. We set $\varepsilon = 1/3$ as the standard error and denote

  $$\mathrm{R}^1(f) = \mathrm{R}^1_{\frac{1}{3}}(f).$$

- **Zero error:** In this case, the protocol is not allowed to make any errors. However, it can output $\perp$, indicating that the communication was inconclusive. For every $x, y$, we must have $\mathbf{Pr}_R[\pi_R(x,y) = \perp] \leqslant \frac{1}{3}$. We denote
$$\mathrm{R}_0(f) = \inf \mathrm{CC}^{\mathrm{avg}}(\pi_R),$$
where the infimum is over every such protocol.

Note that one can convert a zero-error protocol $\pi$ to a one-sided error protocol by outputting, say, 1 instead of $\perp$. We conclude
$$\mathrm{R}(f) \leqslant \mathrm{R}^1(f) \leqslant \mathrm{R}_0(f) \leqslant \mathrm{D}(f).$$

**Example 18.1** (Equality Function, Identity Matrix)**.** Consider the equality function $\mathrm{EQ} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. 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 probability can be reduced to below $\frac{1}{3}$ by repeating this procedure twice. We conclude
$$\mathrm{R}(\mathrm{EQ}) = O(1) \qquad \text{and} \qquad \mathrm{R}^1(\overline{\mathrm{EQ}}) = O(1),$$
where $\overline{\mathrm{EQ}} = 1 - \mathrm{EQ}$.

# Yao's Min-Max Theorem

Let $\mu$ be a probability distribution over $\mathcal{X} \times \mathcal{Y}$. The distributional communication complexity of $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ 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, $\mathrm{D}^{\mu}_{\varepsilon}(f)$ is the smallest cost of a deterministic protocol $\pi$ such that
$$\mathop{\mathbf{Pr}}_{(x,y)\sim\mu}[\pi(x,y) \neq f(x,y)] \leqslant \varepsilon.$$

Note that in contrast to the previous notions of communication complexity, here we care about the average error, as the probability of error is measured for an input that is chosen randomly. Let us start with the following easy observation.

**Proposition 18.2.** *For every $\mu$, we have*
$$\mathrm{D}^{\mu}_{\varepsilon}(f) \leqslant \mathrm{R}_{\varepsilon}(f).$$

*Proof.* Consider a randomized protocol $\pi_R$ of cost $c$. Then for every $(x, y)$, we have
$$\mathop{\mathbf{Pr}}_{R}[\pi_R(x,y) \neq f(x,y)] \leqslant \varepsilon.$$

Since this holds for every $(x, y)$, we can average it with $\mu$ weights.
$$\mathop{\mathbf{Pr}}_{\substack{(x,y)\sim\mu \\ R}}[\pi_R(x,y) \neq f(x,y)] \leqslant \varepsilon.$$

Now there must exist an $r$ in the support of $R$ such that the deterministic protocol $\pi_r$ satisfies
$$\mathop{\mathbf{Pr}}_{(x,y)\sim\mu}[\pi_r(x,y) \neq f(x,y)] \leqslant \varepsilon.$$

Therefore, $\mathrm{D}^{\mu}_{\varepsilon}(f) \leqslant c$. □

Proposition 18.2 suggests that to prove a lower-bound for $\mathrm{R}_{\varepsilon}(f)$, 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 this lower-bound technique is without loss as there is always a distribution $\mu$ that matches $\mathbb{R}_{\varepsilon}(f)$. That is
$$\max_{\mu} \mathrm{D}^{\mu}_{\varepsilon}(f) = \mathrm{R}_{\varepsilon}(f).$$

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 $B$: There are two sets $S = \{s_1, \ldots, s_m\}$ and $T = \{t_1, \ldots, t_m\}$ corresponding to the possible strategies that players $A$ and $B$ can take, respectively. Let $M$ be the payoff matrix: $M = [m_{ij}]_{m \times n}$ 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 game theory terminology, 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 $\{t_1, \ldots, t_n\}$. These mixed strategies mean 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?

$$\mathbb{E}_{\substack{s_i \sim \mu \\ t_j \sim \nu}}[m_{ij}] = \mu^t M \nu,$$

where on the right-hand-side $\mu$ and $\nu$ are represented as vectors. Player $A$ wants to maximize the pay-off, and Player $B$ wants to minimize it.

Suppose we ask Player $A$ to choose her mixed strategy $\mu$ first and then allow Player $B$ to choose his strategy accordingly. Assuming that they 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 $B$ at a great advantage since he can tailor his strategy according to $\mu$. Let us examine this case more carefully. 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 minimized by a $\nu$ that assigns all its mass to the $j$ with the smallest $\sum_{i=1}^{m} \mu_i m_{ij}$. In other words, $\nu$ is a basically some pure strategy $t_j$:

$$\max_\mu \min_\nu \mu^t M \nu = \max_\mu \min_j \sum_{i=1}^{m} \mu_i m_{ij}.$$

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 the rationality of the players, the payoff will be

$$\min_\nu \max_\mu \mu^t M \nu.$$

In this case, it seems that Player $A$ is at a great advantage. Also, again note that Player $A$ can use a pure strategy without affecting the optimum payoff.

$$\min_\nu \max_\mu \mu^t M \nu = \min_\nu \max_i \sum_{j=1}^{n} m_{ij} \nu_j.$$

Perhaps surprisingly, it turns out that it does not matter which player chooses their strategy first.

**Theorem 18.3** (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 18.4.** *For every $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$, we have*

$$\max_\mu \mathrm{D}_\varepsilon^\mu(f) = \mathrm{R}_\varepsilon(f).$$

*Proof.* We have already seen that $\max_\mu \mathrm{D}_\varepsilon^\mu(f) \leqslant \mathrm{R}_\varepsilon(f)$.

To prove the opposite direction, let $c = \max_\mu \mathrm{D}_\varepsilon^\mu(f)$, 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 a randomized communication protocol of cost at most $c$. The pay-off here is

$$\mathop{\mathbf{Pr}}_{\substack{(x,y)\sim\mu\\ \pi\sim\nu}}[f(x,y) \neq \pi(x,y)]$$

By the minimax theorem

$$\min_{\nu}\max_{\mu}\mathop{\mathbf{Pr}}_{\substack{(x,y)\sim\mu\\ \pi\sim\nu}}[f(x,y) \neq \pi(x,y)] = \max_{\mu}\min_{\nu}\mathop{\mathbf{Pr}}_{\substack{(x,y)\sim\mu\\ \pi\sim\nu}}[f(x,y) \neq \pi(x,y)] = \max_{\mu}\min_{\pi\in T}\mathop{\mathbf{Pr}}_{(x,y)\sim\mu}[f(x,y) \neq \pi(x,y)] \leqslant \varepsilon,$$

where in the last inequality we used to assumption that $\max_\mu \mathrm{D}_\varepsilon^\mu(f) = c$. Hence, we conclude that

$$\min_{\nu}\max_{(x,y)}\mathop{\mathbf{Pr}}_{\substack{(x,y)\sim\mu\\ \pi\sim\nu}}[f(x,y) \neq \pi(x,y)] = \min_{\nu}\max_{\mu}\mathop{\mathbf{Pr}}_{\substack{(x,y)\sim\mu\\ \pi\sim\nu}}[f(x,y) \neq \pi(x,y)] \leqslant \varepsilon,$$

which shows that there exists a $\nu$ such that for every $(x,y)$, we have

$$\mathop{\mathbf{Pr}}_{\pi\sim\nu}[f(x,y) \neq \pi(x,y)] \leqslant \varepsilon.$$

Consequently,

$$\mathrm{R}_\varepsilon(f) \leqslant c.$$

Since $\pi \sim \nu$ is a randomized protocol of cost at most $c$.

□

Note that Theorem 18.4 is quite robust, and it applies to any other situation where randomized protocols are defined as a probability distribution over deterministic protocols (e.g. query complexity, randomized algorithms, randomized decision trees, etc).

# Exercises

**Exercise 18.1.** Prove the minimax theorem using linear programming duality.

# Chapter 19

# 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 20, we prove that these notions also arise as two fundamental concepts in theoretical machine learning.

**Question 19.1.** *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 $\mathrm{R}_{\frac{1}{2}}(f) = 0$ 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 $\frac{1}{2}$ and ask for the smallest communication complexity that can achieve an error that is strictly smaller than $\frac{1}{2}$. As we shall see, this question is only interesting in the private coin model.

**Question 19.2.** *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 $\frac{1}{2}$?*

## Discrepancy

It turns out that the answer to Question 19.1 is the so-called *discrepancy*. This notion is 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 $\pm 1$-valued functions.

Let $\mathcal{X}, \mathcal{Y}$ be finite sets. The discrepancy of a function $f : \mathcal{X} \times \mathcal{Y} \to \{-1,1\}$ with respect to a measure $\mu$ on $\mathcal{X} \times \mathcal{Y}$ is defined as

$$\mathrm{Disc}^{\mu}(f) = \max_{A \times B \subseteq \mathcal{X} \times \mathcal{Y}} \left|\mathbb{E}_{xy \sim \mu}\left[f(x,y) 1_A(x) 1_B(y)\right]\right|,$$

and the discrepancy of $f$ is defined with respect to the "hardest" distribution $\mu$:

$$\mathrm{Disc}(f) = \min_{\mu} \mathrm{Disc}^{\mu}(f),$$

where the minimum is over probability distributions $\mu$ on $\mathcal{X} \times \mathcal{Y}$.

Note that if $A \times B$ is a monochromatic rectangle, then $|\mathbb{E}_{xy \sim \mu}[f(x,y)1_A(x)1_B(y)]| = \mu(A \times B)$. So a small discrepancy under $\mu$, in particular, implies that all monochromatic rectangles have small measures, and thus the discrepancy method is a generalization of the rectangle bound. Roughly speaking, the discrepancy of a function under $\mu$ is small if combinatorial rectangles are balanced: the difference between the measure of 1's and $-1$'s in every rectangle is small. By the concentration of probability, it is not difficult to see that a typical random function will always have a very small discrepancy.

First, we show that discrepancy can be used to achieve a protocol of cost 2 that has error guarantee $\frac{1}{2} - \frac{\mathrm{Disc}(f)}{2}$.

**Theorem 19.3.** *For $\delta = \mathrm{Disc}(f)$, we have*

$$\mathrm{R}_{\frac{1}{2} - \frac{\delta}{2}}(f) \leqslant 2.$$

*Proof.* Suppose that

$$\mathrm{Disc}(f) = \min_{\mu} \max_{A \times B \subseteq \mathcal{X} \times \mathcal{Y}} \left|\mathbb{E}_{xy \sim \mu}\left[f(x,y) 1_A(x) 1_B(y)\right]\right| = \delta.$$

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 $(x, y) \notin A \times B$, 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

$$\mathrm{Disc}(f) = \min_{\mu} \max_{(A \times B, b)} \mathbb{E}_{xy\sim\mu} \left[ f(x, y) b 1_A(x) 1_B(y) \right] = \delta.$$

By applying the minimax principle,

$$\max_{\nu} \min_{xy} \mathbb{E}_{(A\times B,b)\sim\nu} \left[ f(x, y) b \mathbf{1}_A(x) \mathbf{1}_B(y) \right] = \delta.$$

In particular, there exists a distribution $\nu$ such that

$$\min_{xy} \mathbb{E}_{(A\times B,b)\sim\nu} \left[ f(x, y) b \mathbf{1}_A(x) \mathbf{1}_B(y) \right] \geqslant \delta.$$

Think of $\nu$ as a probabilistic protocol. We choose a random rectangle $(A, B)$ according to $\nu$, together with a predicted value $b \in \{-1, 1\}$ for the points in the rectangle. For the points outside the rectangle, we make no predictions.

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 $(x, y)$,

$$\mathbb{E}_R \left[ \pi_R(x, y) f(x, y) \right] = \mathbb{E}_{(A\times B,b)\sim\nu} \left[ f(x, y) b \mathbf{1}_A(x) \mathbf{1}_B(y) \right] \geqslant \delta,$$

which shows that

$$\mathop{\mathbf{Pr}}_{R}[\pi_R(x, y) \neq f(x, y)] \leqslant \frac{1}{2} - \frac{\delta}{2}.$$

□

The following theorem shows that discrepancy implies lower bounds for randomized communication complexity.

**Theorem 19.4.** *Let $f : \mathcal{X} \times \mathcal{Y} \to \{-1, 1\}$, let $\varepsilon \in (0, 1/2)$. We have*

$$\log \left( \frac{2\varepsilon}{\mathrm{Disc}(f)} \right) \leqslant \mathrm{R}_{\frac{1}{2}-\varepsilon}(f).$$

*Proof.* Let $c = \mathrm{R}_{\frac{1}{2}-\varepsilon}(f)$. For every probability distribution $\mu$ on $\mathcal{X} \times \mathcal{Y}$, we have

$$\mathrm{D}^{\mu}_{\frac{1}{2}-\varepsilon}(f) \leqslant c,$$

and thus, there is a deterministic protocol $\pi$ such that

$$\frac{1}{2} + \varepsilon \leqslant \mathop{\mathbf{Pr}}_{xy\sim\mu}[\pi(x, y) = f(x, y)],$$

which translates to

$$\mathbb{E}_{xy\sim\mu}[\pi(x, y) f(x, y)] \geqslant 2\varepsilon.$$

Since the leaves of the protocol provide a partition of $\mathcal{X} \times \mathcal{Y}$ into at most $2^c$ rectangles $A_\ell \times B_\ell$, we can write $\pi(x, y) = \sum_\ell \mathbf{1}_{A_\ell}(x) \mathbf{1}_{B_\ell}(y) \pi(\ell)$, where $\pi(\ell) \in \{-1, 1\}$ is the output of the protocol at leave $\ell$.

Hence, there exists a combinatorial rectangle $A \times B$ such that

$$\frac{2\varepsilon}{2^c} \leqslant \mathbb{E}_{xy\sim\mu}[f(x, y) \pi(x, y) \mathbf{1}_A(x) \mathbf{1}_B(y)] \leqslant \left| \mathbb{E}_{xy\sim\mu}[f(x, y) \mathbf{1}_A(x) \mathbf{1}_B(y)] \right|.$$

We conclude

$$\mathrm{Disc}(f) \geqslant \frac{2\varepsilon}{2^c}.$$

□

Note that by Theorem 19.3 and Theorem 19.4, we have the following corollary that answers Question 19.1.

**Corollary 19.5** (Answer to Question 19.1)**.** *Let $f : \mathcal{X} \times \mathcal{Y} \to \{-1, 1\}$. Then*

$$\frac{\mathrm{Disc}(f)}{2} \leqslant \sup\{\varepsilon : \mathrm{R}_{\frac{1}{2}-\varepsilon}(f) \leqslant 2\} \leqslant 2\mathrm{Disc}(f).$$

### Eigen-value method

### Discrepancy and Grothendieck's inequality

We defined the discrepancy $\mathrm{Disc}^{\mu}(f)$ according to correlations with combinatorical rectangles. That is the largest possible

$$|\langle f, 1_A(x)1_B(y)\rangle_\mu| = |\mathbb{E}_\mu f(x,y)1_A(x)1_B(y)|.$$

Sometimes it is useful to relax this slightly, and instead of rectangles $1_A(x)1_B(y)$, use more general rank-1 functions $u(x)v(y)$, where $u : \mathcal{X} \to [-1,1]$ and $v : \mathcal{Y} \to [-1,1]$. Let us define

$$\mathrm{Disc}^{\mu}_{\pm}(f) = \max_{\substack{u:\mathcal{X}\to[-1,1]\\ v:\mathcal{Y}\to[-1,1]}} \mathbb{E}_{xy\sim\mu}\left[f(x,y)u(x)v(y)\right].$$

Since this definition is a relaxation of the definition of $\mathrm{Disc}^{\mu}(f)$, we have

$$\mathrm{Disc}^{\mu}(f) \leqslant \mathrm{Disc}^{\mu}_{\pm}(f).$$

On the other hand, note that $\mathbb{E}_{xy\sim\mu}\left[f(x,y)u(x)v(y)\right]$ is a linear function in each $u(x)$ and each $v(y)$. It follows that in the definition of $\mathrm{Disc}^{\mu}_{\pm}(f)$, we can actually assume that the range of $u$ and $v$ are the discrete set $\{-1,1\}$, and define equivalently:

$$\mathrm{Disc}^{\mu}_{\pm}(f) = \max_{\substack{u:\mathcal{X}\to\{-1,1\}\\ v:\mathcal{Y}\to\{-1,1\}}} \mathbb{E}_{xy\sim\mu}\left[f(x,y)u(x)v(y)\right].$$

Note that $u(x)v(y)$ can be written as a sum and subtractions of four rectangles:

$$u(x)v(y) = 1_A(x)1_B(y) + 1_{A^c}(x)1_{B^c}(y) - 1_{A^c}(x)1_B(y) - 1_A(x)1_{B^c}(y),$$

where $A = u^{-1}(1)$ and $B = v^{-1}(1)$. We conclude that the two notions are equivalent:

$$\mathrm{Disc}^{\mu}(f) \leqslant \mathrm{Disc}^{\mu}_{\pm}(f) \leqslant 4\mathrm{Disc}^{\mu}(f).$$

Let us relax the definition even further. Let $S^{d-1}$ denote the unit sphere in dimension $\mathbb{R}^d$.

$$\mathrm{Disc}^{\mu}_{\gamma_2}(f) = \max_{d} \max_{\substack{u:\mathcal{X}\to S^{d-1}\\ v:\mathcal{Y}\to S^{d-1}}} \mathbb{E}_{xy\sim\mu}\left[f(x,y)\langle u(x), v(y)\rangle\right].$$

Again we could alternatively define $\mathrm{Disc}^{\mu}_{\gamma_2}(f)$ using the unit ball instead of the unit sphere, and the two definitions would be equivalent.

Note that when $d = 1$ the only points on $S^1$ are $\{-1,1\}$, and we recover the definition of $\mathrm{Disc}^{\mu}_{\pm}(f)$. Hence, this is a relaxation, and we have

$$\mathrm{Disc}^{\mu}_{\pm}(f) \leqslant \mathrm{Disc}^{\mu}_{\gamma_2}(f).$$

We can also bound $\mathrm{Disc}^{\mu}_{\gamma_2}(f)$ by $O(\mathrm{Disc}^{\mu}_{\pm}(f))$ using the so-called Grothendieck inequality.

**Theorem 19.6** (Grothendieck inequality)**.** *Let $M = [m_{ij}]$ be an $m \times n$ matrix with real entries, and let $d \in \mathbb{N}$. Then*

$$\max_{u_i, v_j \in S^{d-1}} \sum_{i,j} m_{ij}\langle u_i, v_j\rangle \leqslant K \max_{u_i, v_j \in \{-1,1\}} \sum_{i,j} m_{ij} u_i v_j,$$

*where $K = \frac{\pi}{2\ln(1+\sqrt{2})} \leqslant 1.78$.*

We conclude that the three notions $\mathrm{Disc}^{\mu}(f), \mathrm{Disc}^{\mu}_{\pm}(f), \mathrm{Disc}^{\mu}_{\gamma^2}(f)$ are equivalent:

$$\mathrm{Disc}^{\mu}(f) \leqslant \mathrm{Disc}^{\mu}_{\pm}(f) \leqslant \mathrm{Disc}^{\mu}_{\gamma^2}(f) \leqslant 8\mathrm{Disc}^{\mu}(f).$$

## Unbounded-Error model

As we have seen in the discussion about the complexity class $\mathsf{PP}$, achieving an error of $1/2$ for a randomized protocol is trivial. Namely, for any Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$ 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-coin protocols or private-coin protocols? First, we argue that this model is not very interesting for public randomness.

**Proposition 19.7.** *For any $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$ there is a public-coin protocol $\pi_R$ of cost 2 such that*

$$\Pr_R[\pi_R(x,y) \neq f(x,y)] < \frac{1}{2}.$$

*Proof.* Alice and Bob use the shared randomness to sample 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

$$\mathbf{Pr}[\text{Error}] = \frac{1}{2}\,\mathbf{Pr}[(x,y) \neq (x',y')] < \frac{1}{2}.$$

□

So, we focus our attention from now on to private-coin protocols.

**Definition 19.8** (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

$$\mathbf{Pr}[\pi(x,y) \neq f(x,y)] < \frac{1}{2} \qquad \forall x, y.$$

## 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 19.9** (Sign-rank)**.** The sign-rank of a matrix $A = [a_{ij}]$, denoted by $\mathbf{rk}_\pm(A)$ is the smallest rank of a matrix $B = [b_{ij}]$ such that $a_{ij}b_{ij} > 0$ for all $i, j$.

**Theorem 19.10** (Paturi and Simon [PS86])**.** *For every $f : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$, we have*

$$U(f) = \log \mathbf{rk}_\pm(f) \pm O(1).$$

*Proof.* We first prove that $\log \mathbf{rk}_\pm(f) \leqslant U(f) + O(1)$. 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 $\mathbf{rk}_\alpha \pm (f) \leqslant \mathbf{rk}(A)$. 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 $\mathbf{rk}(A) \leqslant |L| \leqslant 2^c$ as desired.

We next prove that $U(f) \leqslant \log \mathbf{rk}_{\pm 1} + O(1)$. Assume that $\mathbf{rk}_\pm(f) = r$. Equivalently, there exist vectors $u_x, v_y \in \mathbb{R}^r$ such that $\mathbf{sign}(\langle u_x, v_y \rangle) = f(x,y)$ 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 $a_i = \mathbf{sign}((u_x)_i) \in \{-1,1\}$ and sends it to Bob.
- Bob, on receiving $i \in [r]$, $a_i \in \{-1,1\}$ from Alice, knows $u_x(i)$. He samples a random bit $b \in \{-1,1\}$ such that $\mathbb{E}[b] = u_x(i)v_y(i)$, and output it.

Note that

$$\mathbb{E}[b] = \sum_{i=1}^r u_x(i)v_y(i) = \langle u_x, v_y \rangle,$$

which together with $\mathbf{sign}(\langle u_x, v_y \rangle) = f(x,y)$ shows that the error probability of $\pi$ is strictly less than $\frac{1}{2}$. □

In light of Theorem 19.10, we can forget about communication complexity, and focus on the mathematical notion of sign-rank. In Chapter 20, 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

# Chapter 20

# Theory of Learning: Dimension and Margin

In Chapter 19, we saw that the randomized communication complexity in the unbounded-error model is essentially equivalent to a fundamental matrix parameter called sign-rank. This section studies sign-rank and a closely related parameter called margin. These notions occupy a central place in present-day machine learning in theory and practice. The performance of classifiers such as Support Vector Machines (a.k.a SVM) relies on low-dimension high-margin representations of data. As we shall see, these coincide with the sign rank and the margin of the corresponding matrix.

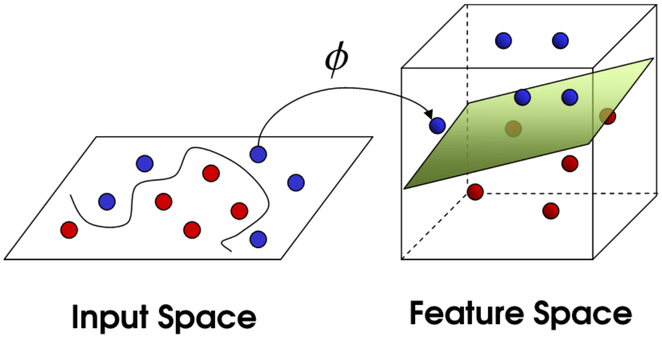

Figure 20.1: In the Support Vector Machine algorithm, the data is represented in a low dimensional space (feature space), and generated hypothesis is the hyperplane with the highest margin on the training data.

## Sign-rank in Machine Learning: Dimension

Let $X$ 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* $\mathcal{C} = \{f_1, \dots, f_m\}$ is simply a set of functions $f_j : A \to \mathbb{R}$. We will be interested in the Boolean case, where concepts take ±1 values. Such concepts, for example, could correspond to likes/dislikes of a person, e.g $f_j(a) = 1$ means that the person $j$ likes the item $a$, and $f_j(a) = -1$ corresponds to disliking the item.

The goal in machine learning is to observe the value of an unknown $f \in \mathcal{C}$ 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 a person's preferences 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 helpful to represent the concepts and the data items geometrically as points in $\mathbb{R}^d$ for some $d$. For example, for medical research, a person can be represented by their age, height, weight, etc., each corresponding to a coordinate of $\mathbb{R}^d$. The space $\mathbb{R}^d$ is called the feature space, and every coordinate of $\mathbb{R}^d$ is a feature of the input.

Every concept $f : X \to \{-1, 1\}$ is a partition of the input points into two sets. It is desirable to define these partitions via mathematically nice objects such as hyperplanes.

**Definition 20.1.** We say that the concept class $\mathcal{C}$ is *realized* by *unit* vectors $u_1, \ldots, u_m \in \mathbb{R}^d$ and $x_1, \ldots, x_n \in \mathbb{R}^d$ if

$$f_j(x_i) = \mathbf{sign}(\langle u_j, x_i \rangle),$$

where

$$\mathbf{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}.$$

Note that

$$\mathbf{sign}(\langle u_j, x_i \rangle) = \mathbf{sign}\left(\left\langle \frac{u_j}{\|u_j\|}, \frac{x_i}{\|x_i\|} \right\rangle\right), \tag{20.1}$$

and thus the assumption that the vectors $u_j$ and $x_i$ are unit vectors is without loss of generality.

In such a realization, items are represented by points $x_i$ on the unit sphere, 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 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 best 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. For such algorithms to succeed, it is desirable to make sure that the data points are far away from the boundary of these half-spaces. This is captured by the *margin* of the representation, which is defined as

$$\gamma = \min_{i,j} |\langle u_j, x_i \rangle|.$$

Note that $\gamma$ is the closest distance of any point $x_i$ to one of the hyperplanes defined by $u_i$.

Two important questions arise naturally:

- *What is the smallest dimension $d$ such that $\mathcal{C}$ can be realized in $\mathbb{R}^d$?*
- *What is the largest (more accuratly, supremum of) $\gamma$ such that there is a realization of $\mathcal{C}$ with margin $\gamma$?* This quantity is called the margin of the concept class and is denoted by $\mathrm{mgin}(\mathcal{C})$.

It turns out that sign-rank is the answer to the first question, and the discrepancy is the answer to the second question!

For the reader's convenience, we recall the definition of the sign-rank.

**Definition 20.2** (Sign-rank)**.** The sign-rank of a matrix $A = [a_{ij}]$, denoted by $\mathbf{rk}_{\pm}(A)$ is the smallest rank of a matrix $B = [b_{ij}]$ such that $a_{ij}b_{ij} > 0$ for all $i, j$.

**Proposition 20.3.** *Let $\mathcal{C} = \{f_1, \ldots, f_m\}$ be a concept class of functions $f_j : X \to \{-1, 1\}$, 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 $\mathcal{C}$ can be realized in $\mathbb{R}^d$.*

*Proof.* First note that if there is a realization of $\mathcal{C}$ in $\mathbb{R}^d$ with vectors $u_1, \ldots, u_m$ and $v_1, \ldots, v_n$, then $M = \mathbf{sign}(UV)$, 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 $\mathbf{rk}_{\pm}(F) \leqslant \mathbf{rk}(UV) \leqslant d$.

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 $\mathbf{rk}(B) = d$. 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 $u_j \in \mathbb{R}^d$ 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

$$u'_j := \frac{1}{\sqrt{1 + t_j^2}} \begin{bmatrix} u_j \\ -t_j \end{bmatrix} \qquad \tilde{x}'_i = \frac{1}{\sqrt{2}} \begin{bmatrix} \tilde{x}_i \\ 1 \end{bmatrix}.$$

Note that

$$\langle u'_j, \widetilde{x}'_i \rangle > 0 \Leftrightarrow \langle u_j, \widetilde{x}_i \rangle > t_j,$$

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:

$$x \mapsto \mathbf{sign}(\langle u, x \rangle) = \mathbf{sign}\left(\sum u_i x_i\right).$$

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 20.4.** *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 $S_{i,j} = \mathbf{sign}(p_j(x_i))$. Then $\mathbf{rk}_\pm(S) = O(d^e)$.*

*Proof.* We use linearization. Let $\mathcal{I}_e$ denote the set of all $\alpha = (\alpha_1, \ldots, \alpha_d) \in \mathbb{Z}_+^d$ such that $\sum_{i=1}^d \alpha_i \leqslant e$. For $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, and $\alpha \in \mathcal{I}_e$, 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 $p_i(x) = \sum_{\alpha \in \mathcal{I}_e} \lambda_{i,\alpha} x^\alpha$. For each $p_i$ define the vector $u_i = (\lambda_{i,\alpha} : \alpha \in \mathcal{I}_e) \in \mathbb{R}^{|\mathcal{I}_e|}$, and for every $x \in \mathbb{R}^d$ define the vector $v_x = (x^\alpha : \alpha \in \mathcal{I}_e)$. Now note that $p_i(x) = \langle u_i, v_x \rangle$. Hence

$$\mathbf{rk}_\pm(S) = |\mathcal{I}_e| = O(d^e).$$

□

## Margin and Discrepancy

**Theorem 20.5.** *Let $f : \mathcal{X} \times \mathcal{Y} \to \{-1, 1\}$. We have*

$$\mathrm{mgin}(f) = \mathrm{Disc}_{\gamma_2}(f).$$

*Proof.* Let $\delta = \mathrm{Disc}_{\gamma^2}(M)$. Then for every $\varepsilon > 0$, by applying Yao's minimax principle, there exists a $d$ such that

$$\delta - \frac{\varepsilon}{2} < \max_{\nu} \min_{xy} \mathbb{E}_{(u,v)\sim\nu} \left[ f(x,y) \langle u(x), v(y) \rangle \right],$$

where $\nu$ is a probability distrubtions over pairs $(u, v)$ with $u : \mathcal{X} \to S^{d-1}$, and $v : \mathcal{Y} \to S^{d-1}$. In paticular, there exists a number $k \in \mathbb{N}$ such that for every $x, y$, we have

$$\delta - \varepsilon < f(x,y) \left( \sum_{i=1}^k \nu(i) \langle u_i(x), v_i(y) \rangle \right), \tag{20.2}$$

where $u_i : \mathcal{X} \to S^{d-1}$, and $v_i : \mathcal{Y} \to S^{d-1}$.

Define the*unit* vectors $u_x, v_y \in \mathbb{R}^{kd}$ as

$$u_x = \left( \sqrt{\nu(1)} u_1(x), \sqrt{\nu(2)} u_2(x), \ldots, \sqrt{\nu(k)} u_k(x) \right) \in \mathbb{R}^{kd},$$

and similarly

$$v_y = \left( \sqrt{\nu(1)} v_1(y), \sqrt{\nu(2)} v_2(y), \ldots, \sqrt{\nu(k)} v_k(y) \right) \in \mathbb{R}^{kd}.$$

Note that Eq. (20.2) now means that

$$\delta - \varepsilon \leqslant f(x,y) \langle u_x, v_y \rangle,$$

and thus, we have a realization of $f$ in $\mathbb{R}^{kd}$ with margin at least $\delta - \varepsilon$. Taking the limit of $\varepsilon \to 0$ proves that

$$\mathrm{mgin}(f) \geqslant \mathrm{Disc}_{\gamma_2}(f).$$

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

$$\eta \leqslant f(x,y)\langle u_x, v_y\rangle.$$

Then for every distribution $\mu$ on the inputs

$$\mathrm{Disc}^{\mu}_{\gamma_2}(f) \geqslant \mathbb{E}_{\mu} f(x,y)\langle u_x, v_y\rangle \geqslant \eta,$$

which shows that

$$\mathrm{Disc}_{\gamma_2}(f) \geqslant \mathrm{mgin}(f).$$

□

## Bounding discrepancy by spectral norm

One improtant method to bound the discrepancy is to show that the spectral norm of the matrix is small. Recall that the spectral norm of a matrix $M$ is defined as

$$\|M\| = \sup \frac{\|Mu\|_2}{\|u\|_2} = \sigma_{\max}(M),$$

where $\sigma_{\max}$ is the largest singular value of $M$. Equivalently $\sigma_{\max} = \sqrt{\lambda_{max}}$, where $\lambda_{\max}$ is the largest eigenvalue of $MM^T$. A good rule of thumb is that sign-matrices that have small spectral norm are random-looking, and they have high complexity.

**Lemma 20.6.** *For every $m \times n$ matrix $M$, we have*

$$\mathrm{Disc}^{\mu}_{\gamma_2}(M) \leqslant \frac{\|M\|}{\sqrt{mn}},$$

*where $\mu$ is the uniform measure over $[m] \times [n]$.*

*Proof.* For all unit vectors $u_i, v_j \in \mathbb{R}^d$, we have

$$\begin{aligned} mn \times \mathrm{Disc}^{\mu}_{\gamma_2}(M) &= \sum_{i,j} m_{ij}\langle u_i, v_j\rangle = \sum_{r=1}^{d}\sum_{i,j} m_{ij} u_i(r) v_j(r) = \sum_{r=1}^{d} \langle u^{(r)}, Mv^{(r)}\rangle \leqslant \sum_{r=1}^{d} \|u^{(r)}\|\|M\|\|v^{(r)}\| \\ &\leqslant \|M\| \sqrt{\sum_{r=1}^{d} \|u^{(r)}\|^2}\sqrt{\sum_{r=1}^{d} \|v^{(r)}\|^2} = \sqrt{mn}\|M\|. \end{aligned}$$

□

## 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 a 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 20.1 t 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 $\mathbf{sign}(\langle u_i, v_j\rangle)$.

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 $u_i$'s are orthonormal, simply because $X$ might be much larger than $d$. Instead, this even distribution is captured by what is called isotropic position.

**Definition 20.7.** 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 $e_1, \dots, e_d$ is in isotropic position. Also note that the union (with repretition) of two isotropic set of vectors is also isotropic. Another way to obtain a set of points in isotropic positions is to take 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 20.8** (Forster [For02])**.** *Let $u_1, \dots, u_m \in \mathbb{R}^d$ be in general positions, meaning that every $d$ of them are linearly independent. There exists an invertible linear transformation $T : \mathbb{R}^d \to \mathbb{R}^d$ such that the unit vectors $\frac{Tu_1}{\|Tu_1\|}, \dots, \frac{Tu_m}{\|Tu_m\|} \in \mathbb{R}^d$ are in isotropic position.*

The assumption in Theorem 20.8 that the vectors $u_i$ are in general position is not an issue for us. Indeed if $u_1, \dots, u_m \in \mathbb{R}^d$ and $v_1, \dots, v_n \in \mathbb{R}^d$ 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 20.8 allows us to assume, without loss of generality, that in a realization of $M \in \{-1,1\}^{m\times n}$ with unit vectors $u_1, \dots, u_m \in \mathbb{R}^d$ and $v_1, \dots, v_n \in \mathbb{R}^d$, the vectors $u_i$ are in isotrpic position. Indeed we can apply Theorem 20.8, and replace the original vectors with the unit vectors $\frac{Tu_i}{\|Tu_i\|} \in \mathbb{R}^d$ and $\frac{T^{-1}v_j}{\|T^{-1}v_j\|} \in \mathbb{R}^d$. Note that

$$\langle Tu_i, T^{-1}v_j\rangle = \langle u_i, v_j\rangle,$$

which shows

$$\mathbf{sign}\left\langle \frac{Tu_i}{\|Tu_i\|}, \frac{T^{-1}v_i}{\|T^{-1}v_j\|}\right\rangle = \mathbf{sign}\langle u_i, v_j\rangle.$$

We are ready to prove Forster's lower-bound on sign-rank.

**Theorem 20.9** (Forster's Theorem)**.** *Every matrix $M \in \{-1,1\}^{m\times n}$ satisfies*

$$\mathbf{rk}_{\pm}(M) \geqslant \frac{mn}{\|M\|_{\gamma_2^*}} \geqslant \frac{\sqrt{mn}}{\|M\|}.$$

*Proof.* Consider a realization of $M \in \{-1,1\}^{m\times n}$ with unit vectors $u_1, \dots, u_m \in \mathbb{R}^d$ and $v_1, \dots, v_n \in \mathbb{R}^d$. By Theorem 20.8, we can assume without loss of generality that $u_1, \dots, u_m \in \mathbb{R}^d$ are in isotropic position.

Since $|\langle u_i, v_j\rangle| \leqslant 1$, and the fact that $u_i$ are in isotropic position, we have

$$\sum_{i,j} |\langle u_i, v_j\rangle| \geqslant \sum_{j=1}^{n}\sum_{i=1}^{m} |\langle u_i, v_j\rangle|^2 = \sum_{j=1}^{n} \frac{m\|v_j\|^2}{d} = \frac{mn}{d}.$$

On the other hand, by Lemma 20.6

$$\sum_{i,j} |\langle u_i, v_j\rangle| = \sum_{ij} m_{ij}\langle u_i, v_j\rangle \leqslant mn\mathrm{Disc}^{\mu}_{\gamma^2}(M) \leqslant \sqrt{mn}\|M\|.$$

Together, they imply

$$d \geqslant \frac{\sqrt{mn}}{\|M\|}.$$

□

## An example

Consider the $\pm1$ version of the inner product function: $\mathrm{IP} : \{0,1\}^n \times \{0,1\}^n \to \{-1,1\}$ defined as

$$\mathrm{IP}(x,y) := (-1)^{\sum_{i=1}^n x_i y_i} = \chi_x(y),$$

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 $\mathrm{IP}(x,y)$.

By the orthogonality of Fourier characters, we have

$$HH^T(x,z) = \sum_y H(x,y)H(x,z) = \sum_y \chi_x(y)\chi_z(y) = \left\{ \begin{array}{ll} 0 & x \neq z \\ 2^n & x = z \end{array} \right. .$$

Hence $HH^T = 2^n \mathbf{I}_{2^n}$, where $\mathbf{I}_{2^n}$ is the identity matrix. Thus all the eigvenvalues of $HH^T$ are $2^n$. Consequently, all the signular values of $H$ are $2^{n/2}$, and thuus $\|H\| = \sqrt{2^n} = 2^{n/2}$. We conclude that

$$\mathrm{Disc}_{\gamma_2}(H) \leqslant \frac{2^{n/2}}{\sqrt{2^n \times 2^n}} = 2^{-n/2},$$

which in particular implies

$$\mathrm{R}(\mathrm{IP}) = \Omega(n).$$

Furthermore by Theorem 20.9

$$\mathbf{rk}_{\pm}(\mathrm{IP}) \geqslant 2^{n/2},$$

which shows that, we even have

$$U(\mathrm{IP}) = \Omega(n).$$

# Exercises

**Exercise 20.1.** Prove that $\mathrm{Tr}(AB) = \sum_{i,j} a_{ij} b_{ij}$.

**Exercise 20.2.** Let $M = [m_{ij}]$ be an $m \times n$ real-valued matrix. We have

$$\|M\|_{\gamma_2} = \sup \left\{ \|uMv^t\|_{\mathrm{Tr}} \ : \ u \in \mathbb{R}^m, v \in \mathbb{R}^n, \|u\|, \|v\| \leqslant 1 \right\},$$

and in particular

$$\|M\|_{\gamma_2} \geqslant \frac{\|M\|_{\mathrm{Tr}}}{\sqrt{mn}}.$$

**Exercise 20.3.** Prove

$$\|M\|_{\gamma_2} = \sup_{N : \|N\|_{\gamma_2^*=1}} \langle M, N \rangle$$

**Exercise 20.4.** Prove that in the definitions of $\|M\|_{\gamma_2}$ and $\|M\|_{\gamma_2^*}$, one can restrict to $d \leqslant \min m, n$.

**Exercise 20.5.** Grothedieck

**Exercise 20.6.** Box norm.

**Exercise 20.7.** Convert ±1 and $\alpha$ to Boolean.

# Bibliography

[Aar16] Scott Aaronson, *P vs NP*, Open problems in mathematics, Springer, 2016, pp. 1–122. 8

[AB87] N. Alon and R. B. Boppana, *The monotone circuit complexity of Boolean functions*, Combinatorica **7** (1987), no. 1, 1–22. MR 905147 73

[Adl78] Leonard Adleman, *Two theorems on random polynomial time*, 19th Annual Symposium on Foundations of Computer Science (Ann Arbor, Mich., 1978), IEEE, Long Beach, Calif., 1978, pp. 75–83. MR 539832 52

[Ajt83] M. Ajtai, $\Sigma_1^1$*-formulae on finite structures*, Ann. Pure Appl. Logic **24** (1983), no. 1, 1–48. MR 706289 57, 79

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, *PRIMES is in P*, Ann. of Math. (2) **160** (2004), no. 2, 781–793. MR 2123939 46

[AKS19] ———, *Errata: PRIMES is in P*, Ann. of Math. (2) **189** (2019), no. 1, 317–318. MR 3898710 46

[ALWZ20] Ryan Alweiss, Shachar Lovett, Kewen Wu, and Jiapeng Zhang, *Improved bounds for the sunflower lemma*, Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (New York, NY, USA), STOC 2020, Association for Computing Machinery, 2020, p. 624–630. 74

[AW08] Scott Aaronson and Avi Wigderson, *Algebrization: a new barrier in complexity theory*, STOC'08, ACM, New York, 2008, pp. 731–740. MR 2582924 83

[BG81] Charles H. Bennett and John Gill, *Relative to a random oracle A*, $\mathbf{P}^A \neq \mathbf{NP}^A \neq \mathrm{co}-\mathbf{NP}^A$ *with probability* 1, SIAM J. Comput. **10** (1981), no. 1, 96–113. MR 605606 46, 52

[BGS75] Theodore P. Baker, John Gill, and Robert Solovay, *Relativizations of the P =? NP question*, SIAM J. Comput. **4** (1975), no. 4, 431–442. 35

[BM82] Manuel Blum and Silvio Micali, *How to generate cryptographically strong sequences of pseudorandom bits*, 23rd annual symposium on foundations of computer science (Chicago, Ill., 1982), IEEE, New York, 1982, pp. 112–117. MR 780388 81

[Bop97] Ravi B. Boppana, *The average sensitivity of bounded-depth circuits*, Inform. Process. Lett. **63** (1997), no. 5, 257–261. MR 1475339 (98f:68093) 62, 63

[CDL01] Andrew Chiu, George Davida, and Bruce Litow, *Division in logspace-uniform* $\mathrm{NC}^1$, Theor. Inform. Appl. **35** (2001), no. 3, 259–275. MR 1869217 18

[CIKK16] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova, *Learning algorithms from natural proofs*, 31st Conference on Computational Complexity, LIPIcs. Leibniz Int. Proc. Inform., vol. 50, Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2016, pp. Art. No. 10, 24. MR 3540811 83

[Coo71] Stephen A. Cook, *The complexity of theorem-proving procedures*, IN STOC, ACM, 1971, pp. 151–158. 6

[ER60] P. Erdős and R. Rado, *Intersection theorems for systems of sets*, J. London Math. Soc. **35** (1960), 85–90. MR 111692 74

[For94] Lance Fortnow, *The role of relativization in complexity theory*, Bulletin of the European Association for Theoretical Computer Science **52** (1994), 52–229. 36

[For02] Jürgen Forster, *A linear lower bound on the unbounded error probabilistic communication complexity*, vol. 65, 2002, Special issue on complexity, 2001 (Chicago, IL), pp. 612–625. MR 1964645 118, 119

[FSS84] Merrick Furst, James B. Saxe, and Michael Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory **17** (1984), no. 1, 13–27. MR 738749 57, 79

[FV98] Tomás Feder and Moshe Y Vardi, *The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory*, SIAM Journal on Computing **28** (1998), no. 1, 57–104. 29

[GGM85] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, Theory of algorithms (Pécs, 1984), Colloq. Math. Soc. János Bolyai, vol. 44, North-Holland, Amsterdam, 1985, pp. 161–189. MR 872307 81

[Gil77] John Gill, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput. **6** (1977), no. 4, 675–695. MR 464691 39

[GJ79] Michael R. Garey and David S. Johnson, *Computers and intractability*, W. H. Freeman and Co., San Francisco, Calif., 1979, A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences. MR 519066 29

[Has86] Johan Hastad, *Almost Optimal Lower Bounds for Small Depth Circuits*, Proceedings of the 18th Annual ACM Symposium on Theory of Computing, ACM, 1986, pp. 6–20. 57, 79

[HN90] Pavol Hell and Jaroslav Nešetřil, *On the complexity of $H$-coloring*, J. Combin. Theory Ser. B **48** (1990), no. 1, 92–110. MR 1047555 29

[HsILL99] Johan Hå stad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput. **28** (1999), no. 4, 1364–1396. MR 1681085 81

[Imm88] Neil Immerman, *Nondeterministic space is closed under complementation*, SIAM J. Comput. **17** (1988), no. 5, 935–938. MR 961049 25

[IW97] Russell Impagliazzo and Avi Wigderson, *P= BPP if E requires exponential circuits: Derandomizing the xor lemma*, Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 1997, pp. 220–229. 46

[Kar72] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972), 1972, pp. 85–103. MR 0378476 9, 29

[KL80] Richard M. Karp and Richard J. Lipton, *Some connections between nonuniform and uniform complexity classes*, Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA (Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, eds.), ACM, 1980, pp. 302–309. 51

[Lad75] Richard E. Ladner, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach. **22** (1975), 155–171. MR 464698 29

[Lev73] L. A. Levin, *Universal enumeration problems*, Problemy Peredači Informacii **9** (1973), no. 3, 115–116. MR 0340042 6

[LG14] François Le Gall, *Powers of tensors and fast matrix multiplication*, ISSAC 2014—Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2014, pp. 296–303. MR 3239939 40

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. Assoc. Comput. Mach. **40** (1993), no. 3, 607–620. MR 1370363 (96h:68074) 69, 99

[Lok08] Satyanarayana V. Lokam, *Complexity lower bounds using linear algebra*, Found. Trends Theor. Comput. Sci. **4** (2008), no. 1-2, front matter, 1–155 (2009). MR 2539154 119

[Man95] Yishay Mansour, *An $O(n^{\log \log n})$ learning algorithm for DNF under the uniform distribution*, vol. 50, 1995, Fifth Annual Workshop on Computational Learning Theory (COLT) (Pittsburgh, PA, 1992), pp. 543–550. MR 1339562 100

[MW18] Cody Murray and Ryan Williams, *Circuit lower bounds for nondeterministic quasi-polytime: an easy witness lemma for np and nqp*, Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, 2018, pp. 890–901. 65

[PS86] Ramamohan Paturi and Janos Simon, *Probabilistic communication complexity*, vol. 33, 1986, Twenty-fifth annual symposium on foundations of computer science (Singer Island, Fla., 1984), pp. 106–123. MR 864082 112

[Raz85a] A. A. Razborov, *Lower bounds on the monotone complexity of some Boolean functions*, Dokl. Akad. Nauk SSSR **281** (1985), no. 4, 798–801. MR 785629 73, 79

[Raz85b] Alexander A. Razborov, *Lower bounds on monotone complexity of the logical permanent*, Mathematical notes of the Academy of Sciences of the USSR **37** (1985), 485–493. 77

[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). 65, 66, 79

[Raz95] Alexander A. Razborov, *Bounded arithmetic and lower bounds in Boolean complexity*, Feasible mathematics, II (Ithaca, NY, 1992), Progr. Comput. Sci. Appl. Logic, vol. 13, Birkhäuser Boston, Boston, MA, 1995, pp. 344–386. MR 1322282 74

[Rei08] Omer Reingold, *Undirected connectivity in log-space*, J. ACM **55** (2008), no. 4, Art. 17, 24. MR 2445014 21

[RR97] Alexander A. Razborov and Steven Rudich, *Natural proofs*, vol. 55, 1997, 26th Annual ACM Symposium on the Theory of Computing (STOC '94) (Montreal, PQ, 1994), pp. 24–35. MR 1473047 79, 80

[San09] Rahul Santhanam, *Circuit lower bounds for Merlin-Arthur classes*, SIAM J. Comput. **39** (2009), no. 3, 1038–1061. MR 2538849 83

[Sav70] Walter J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci. **4** (1970), 177–192. MR 266702 22

[Sha49] Claude E. Shannon, *The synthesis of two-terminal switching circuits*, Bell System Tech. J. **28** (1949), 59–98. MR 29860 49

[Sha92] Adi Shamir, *IP = PSPACE*, J. Assoc. Comput. Mach. **39** (1992), no. 4, 869–877. MR 1187216 36

[Smo87] Roman Smolensky, *Algebraic methods in the theory of lower bounds for boolean circuit complexity*, STOC, 1987, pp. 77–82. 65, 66, 79

[Sze88] Róbert Szelepcsényi, *The method of forced enumeration for nondeterministic automata*, Acta Inform. **26** (1988), no. 3, 279–284. MR 975334 25

[Tar88] Éva Tardos, *The gap between monotone and non-monotone circuit complexity is exponential*, Combinatorica **8** (1988), 141–142. 77

[Tod91] Seinosuke Toda, *PP is as hard as the polynomial-time hierarchy*, SIAM J. Comput. **20** (1991), no. 5, 865–877. MR 1115655 47

[Vin04] N. V. Vinodchandran, $\mathrm{AM}_{\mathrm{exp}} \not\subseteq (\mathrm{NP} \cap \mathrm{coNP})/\mathrm{poly}$, Inform. Process. Lett. **89** (2004), no. 1, 43–47. MR 2025889 83

[Vio09] Emanuele Viola, *Guest column: correlation bounds for polynomials over {0 1}*, ACM SIGACT News **40** (2009), no. 1, 27–44. 69

[Wig93] A. Wigderson, *The fusion method for lower bounds in circuit complexity*, Combinatorics, Paul Erdős is eighty, Vol. 1, Bolyai Soc. Math. Stud., János Bolyai Math. Soc., Budapest, 1993, pp. 453–468. MR 1249727 83

[Wig19] Avi Wigderson, *Mathematics and computation: A theory revolutionizing technology and science*, 2019. 7

[Wil14] Ryan Williams, *Nonuniform ACC circuit lower bounds*, J. ACM **61** (2014), no. 1, Art. 2, 32. MR 3167918 65, 83

[Yao85] A. C. Yao, *Separating the polynomial-time hierarchy by oracles*, 26th Annual Symposium on Foundations of Computer Science (sfcs 1985), 1985, pp. 1–10. 57

[Zhu20] Dmitriy Zhuk, *A proof of the csp dichotomy conjecture*, Journal of the ACM (JACM) **67** (2020), no. 5, 1–78. 29

# Appendix A

# Background: Basic Analysis

This chapter aims 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 AM-GM inequality.

**Theorem A.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*

$$(a_1 \ldots a_n)^{1/n} \leqslant \frac{a_1 + \ldots + a_n}{n}.$$

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 $t \leqslant [0,1]$,

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

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

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

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

**Theorem A.2.** *If $f : D \to \mathbb{R}$ is a concave function, then for every $x_1, \ldots, x_n \in D$ and $t_1, \ldots, t_n \geqslant 0$ with $\sum_{i=1}^n t_i = 1$ we have*

$$t_1 f(x_1) + \ldots + t_n f(x_n) \leqslant f(t_1 x_1 + \ldots + t_n x_n).$$

*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 A.3** (Cauchy-Schwarz). *If $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are complex numbers, then*

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

Hölder's inequality is an important generalization of the Cauchy-Schwarz inequality.

**Theorem A.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*

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

The numbers $p$ and $q$ appearing in Theorem A.4 are called *conjugate exponents*. In fact, 1 and $\infty$ are also called conjugate exponents, and Hölder's inequality in this case becomes:

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

The next theorem is called Minkowski's inequality.

**Theorem A.5** (Minkowski's inequality)**.** *If $p \geqslant 1$ is a real number, and $x_1, \ldots, x_n$ are complex numbers, then*

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

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

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

## 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 any countable union of its members. That is, if $A_1, A_2, \ldots$ belong to $\mathcal{F}$, then $\cup_{i=1}^{\infty} A_i \in \mathcal{F}$.

**Example A.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 A.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 $\{E_i\}_{i \in \mathcal{I}}$ is a countable set of *pairwise disjoint sets* in $\mathcal{F}$, then

$$\mu(\cup_{i \in \mathcal{I}} E_i) = \sum_{i \in \mathcal{I}} \mu(E_i).$$

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

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

For many natural measure spaces $\mathcal{M} = (\Omega, \mathcal{F}, \mu)$, 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 A.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 A.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 any 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 $E_1, \ldots, F_m \in \mathcal{A}$, we have

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

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 A.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 A.12.** Let $\mathcal{A}$ be the algebra on $\mathbb{R}$, defined in Example A.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 any countable union of its members. Let $\mathcal{F} \times \mathcal{G}$ 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 $\mathcal{F} \times \mathcal{G}$ 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 A.11 shows that $\mu \times \nu$ extends uniquely from $A$ to a measure over all of $\mathcal{F} \times \mathcal{G}$. 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

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

**Example A.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

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

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

**Theorem A.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,*

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

*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: $d(x, z) \leqslant d(x, y) + d(y, z)$, for every $x, y, z \in M$.

A sequence $\{x_i\}_{i=1}^{\infty}$ 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 $m, n \geqslant N_\varepsilon$, we have $d(x_m, x_n) \leqslant \varepsilon$. 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 A, let us state the Hölder's and Minkowski's inequalities in a more general form.

**Theorem A.15** (Hölder's inequality)**.** *Consider a measure space $\mathcal{M} = (\Omega, \mathcal{F}, \mu)$, 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*

$$\left|\int f(x)\overline{g(x)}d\mu(x)\right| \leqslant \left(\int |f(x)|^p d\mu(x)\right)^{1/p} \left(\int |g(x)|^q d\mu(x)\right)^{1/q}.$$

**Theorem A.16** (Minkowski's inequality)**.** *Consider a measure space $\mathcal{M} = (\Omega, \mathcal{F}, \mu)$, a real $p \geqslant 1$, and two measurable functions $f, g : \Omega \to \mathbb{C}$ such that $|f|^p$ and $|g|^p$ are both integrable. Then*

$$\left(\int |f(x) + g(x)|^p d\mu(x)\right)^{1/p} \leqslant \left(\int |f(x)|^p d\mu(x)\right)^{1/p} + \left(\int |g(x)|^p d\mu(x)\right)^{1/p}.$$

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

**Definition A.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 $x, y \in V$, $\|x + y\| \leqslant \|x\| + \|y\|$.

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

$$\|T\| := \sup_{x\neq 0} \frac{\|Tx\|_Y}{\|x\|_X} < \infty. \tag{A.1}$$

The set of all bounded operators from $X$ to $Y$ is denoted by $B(X, Y)$. Note that the *operator norm* defined in (A.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 $\mathbf{B}_X := \{x : \|x\| \leqslant 1\}$ is called the *unit ball* of $X$. Note that by the triangle inequality, $\mathbf{B}_X$ 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 $\mathbf{B}_X$ 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

$$K^\circ = \{x \in \mathbb{R}^n : \sum x_i y_i \leqslant 1, \ \forall y \in K\}. \tag{A.2}$$

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 A.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 A.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:

$$0 \leqslant \langle x + \lambda y, x + \lambda y \rangle = \langle x, x \rangle + |\lambda|^2 \langle y, y \rangle + \lambda \overline{\langle x, y \rangle} + \overline{\lambda} \langle x, y \rangle.$$

Now taking $\lambda := \sqrt{\frac{\langle x,x \rangle}{\langle y,y \rangle}} \times \frac{\langle x,y \rangle}{|\langle x,y \rangle|}$ will show that

$$0 \leqslant 2\langle x, x \rangle \langle y, y \rangle - 2\sqrt{\langle x, x \rangle \langle y, y \rangle} |\langle x, y \rangle|,$$

which leads to the triangle inequality. □

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

**Example A.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

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

It is easy to see that $u^{(1)}, u^{(2)}, \ldots$ 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 $1 \leqslant p < \infty$, 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 A.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:

$$\begin{aligned}
\|f+g\|_p^p &= \int |f(x)+g(x)|^p d\mu(x) = \int |f(x)+g(x)|^{p-1}|f(x)+g(x)| d\mu(x) \\
&\leqslant \int |f(x)+g(x)|^{p-1}|f(x)| d\mu(x) + \int |f(x)+g(x)|^{p-1}|g(x)| d\mu(x) \\
&\leqslant \left( \int |f(x)+g(x)|^p d\mu(x) \right)^{\frac{p-1}{p}} \|f\|_p + \left( \int |f(x)+g(x)|^p d\mu(x) \right)^{\frac{p-1}{p}} \|g\|_p \\
&= \|f+g\|_p^{p-1}(\|f\|_p + \|g\|_p),
\end{aligned}$$

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 A.22.** *Let $\mathcal{M} = (\Omega, \mathcal{F}, \mu)$ be a probability space, $1 \leqslant p \leqslant q \leqslant \infty$ be real numbers, and $f \in L_q(\mathcal{M})$. Then*

$$\|f\|_p \leqslant \|f\|_q.$$

*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

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

□

Note that Theorem A.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 decreasing.

# Exercises

**Exercise A.1.** Let $x = \langle x_1, \ldots, x_n \rangle$ and $y = \langle y_1, \ldots, y_n \rangle$ be complex vectors. By studying the derivative of $\langle x + ty, x + ty \rangle$ with respect to $t$, prove Theorem A.3.

**Exercise A.2.** Deduce Theorem A.5 from Hölder's inequality.

**Exercise A.3.** Let $1 \leqslant p \leqslant q \leqslant \infty$. Show that for every $f \in \ell_p$, we have $\|f\|_q \leqslant \|f\|_p$.

**Exercise A.4.** Recall that by Hölder's inequality, if $p, q \geqslant 1$ are conjugate exponents and $a_1, \ldots, a_n, b_1, \ldots, b_n$ are complex numbers, then

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

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

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

**Exercise A.5.** Let $X$ be a probability space, and $p, q \geqslant 1$ 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 A.6.** Suppose that $(X, \mu)$ is a measure space and $\frac{1}{p} + \frac{1}{q} + \frac{1}{r} = 1$, for $p, q, r \geqslant 1$. Show that if $f \in L_p(X)$, $g \in L_q(X)$, and $h \in L_r(X)$, then

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

**Exercise A.7.** Suppose that $X$ is a measure space and $\frac{1}{p} + \frac{1}{q} = \frac{1}{r}$, for $p, q, r \geqslant 1$. Show that if $f \in L_p(X)$ and $g \in L_q(X)$, then

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

**Exercise A.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 $p, q \geqslant 1$, and every linear operator $T : L_2(X) \to L_2(X)$, we have

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