# Lecture Notes on Computational Complexity

Luca Trevisan[1]

Notes written in Fall 2002, Revised May 2004

[1]Computer Science Division, U.C. Berkeley. Work supported by NSF Grant No. 9984703. Email `luca@cs.berkeley.edu`.

## Foreword

These are scribed notes from a graduate courses on Computational Complexity offered at the University of California at Berkeley in the Fall of 2002, based on notes scribed by students in Spring 2001 and on additional notes scribed in Fall 2002. I added notes and references in May 2004.

The first 15 lectures cover fundamental material. The remaining lectures cover more advanced material, that is different each time I teach the class. In these Fall 2002 notes, there are lectures on Håstad's optimal inapproximability results, lower bounds for parity in bounded depth-circuits, lower bounds in proof-complexity, and pseudorandom generators and extractors.

The notes have been only minimally edited, and there may be several errors and imprecisions.

I will be happy to receive comments, criticism and corrections about these notes.

The syllabus for the course was developed jointly with Sanjeev Arora. Sanjeev wrote the notes on Yao's XOR Lemma (Lecture 11). Many people, and especially Avi Wigderson, were kind enough to answer my questions about various topics covered in these notes.

I wish to thank the scribes Scott Aaronson, Andrej Bogdanov, Allison Coates, Kamalika Chaudhuri, Steve Chien, Jittat Fakcharoenphol, Chris Harrelson, Iordanis Kerenidis, Lawrence Ip, Ranjit Jhala, François Labelle, John Leen, Michael Manapat, Manikandan Narayanan, Mark Pilloff, Vinayak Prabhu, Samantha Riesenfeld, Stephen Sorkin, Kunal Talwar, Jason Waddle, Dror Weitz, Beini Zhou.

Berkeley, May 2004.

Luca Trevisan
luca@cs.berkeley.edu

# Contents

# Lecture 1

# Introduction, P and NP

This course assumes CS170, or equivalent, as a prerequisite. We will assume that the reader is familiar with the notions of algorithm and running time, as well as with basic notions of discrete math and probability.

A main objective of theoretical computer science is to understand the amount of resources (time, memory, communication, randomness , . . . ) needed to solve computational problems that we care about. While the design and analysis of algorithms puts upper bounds on such amounts, computational complexity theory is mostly concerned with lower bounds; that is we look for *negative results* showing that certain problems require a lot of time, memory, etc., to be solved. In particular, we are interested in *infeasible* problems, that is computational problems that require impossibly large resources to be solved, even on instances of moderate size. It is very hard to show that a particular problem is infeasible, and in fact for a lot of interesting problems the question of their feasibility is still open. Another major line of work in complexity is in understanding the relations between different computational problems and between different "modes" of computation. For example what is the relative power of algorithms using randomness and deterministic algorithms, what is the relation between worst-case and average-case complexity, how easier can we make an optimization problem if we only look for approximate solutions, and so on. It is in this direction that we find the most beautiful, and often surprising, known results in complexity theory.

Before going any further, let us be more precise in saying what a computational problem is, and let us define some important classes of computational problems. Then we will see a particular incarnation of the notion of "reduction," the main tool in complexity theory, and we will introduce **NP**-completeness, one of the great success stories of complexity theory. We conclude by demonstrating the use of diagonalization to show some separations between complexity classes. It is unlikely that such techniques will help solving the **P** versus **NP** problem.

## 1.1 Computational Problems

In a *computational problem*, we are given an *input* that, without loss of generality, we assume to be encoded over the alphabet $\{0, 1\}$, and we want to return in *output* a solution satisfying some property: a computational problem is then described by the property that the output has to satisfy given the input.

In this course we will deal with four types of computational problems: *decision* problems, *search* problems, *optimization* problems, and *counting* problems.[1] For the moment, we will discuss decision and search problem.

In a *decision* problem, given an input $x \in \{0,1\}^*$, we are required to give a YES/NO answer. That is, in a decision problem we are only asked to verify whether the input satisfies a certain property. An example of decision problem is the 3-coloring problem: given an undirected graph, determine whether there is a way to assign a "color" chosen from $\{1,2,3\}$ to each vertex in such a way that no two adjacent vertices have the same color.

A convenient way to *specify* a decision problem is to give the set $L \subseteq \{0,1\}^*$ of inputs for which the answer is YES. A subset of $\{0,1\}^*$ is also called a *language*, so, with the previous convention, every decision problem can be specified using a language (and every language specifies a decision problem). For example, if we call 3COL the subset of $\{0,1\}^*$ containing (descriptions of) 3-colorable graphs, then 3COL is the language that specifies the 3-coloring problem. From now on, we will talk about decision problems and languages interchangeably.

In a *search* problem, given an input $x \in \{0,1\}^*$ we want to compute some answer $y \in \{0,1\}^*$ that is in some relation to $x$, if such a $y$ exists. Thus, a search problem is specified by a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, where $(x,y) \in R$ if and only if $y$ is an admissible answer given $x$.

Consider for example the search version of the 3-coloring problem: here given an undirected graph $G = (V,E)$ we want to find, if it exists, a coloring $c : V \to \{1,2,3\}$ of the vertices, such that for every $(u,v) \in V$ we have $c(u) \neq c(v)$. This is different (and more demanding) than the decision version, because beyond being asked to determine whether such a $c$ exists, we are also asked to construct it, if it exists. Formally, the 3-coloring problem is specified by the relation $R_{\mathrm{3COL}}$ that contains all the pairs $(G,c)$ where $G$ is a 3-colorable graph and $c$ is a valid 3-coloring of $G$.

## 1.2 P and NP

In most of this course, we will study the *asymptotic* complexity of problems. Instead of considering, say, the time required to solve 3-coloring on graphs with $10,000$ nodes on some particular model of computation, we will ask what is the best asymptotic running time of an algorithm that solves 3-coloring on all instances. In fact, we will be much less ambitious, and we will just ask whether there is a "feasible" asymptotic algorithm for 3-coloring. Here feasible refers more to the rate of growth than to the running time of specific instances of reasonable size.

A standard convention is to call an algorithm "feasible" if it runs in polynomial time, i.e. if there is some polynomial $p$ such that the algorithm runs in time at most $p(n)$ on inputs of length $n$.

[1]This distinction is useful and natural, but it is also arbitrary: in fact every problem can be seen as a search problem

We denote by **P** the class of decision problems that are solvable in polynomial time.

We say that a search problem defined by a relation $R$ is a **NP** search problem if the relation is efficiently computable and such that solutions, if they exist, are short. Formally, $R$ is an **NP** search problem if there is a polynomial time algorithm that, given $x$ and $y$, decides whether $(x, y) \in R$, and if there is a polynomial $p$ such that if $(x, y) \in R$ then $|y| \leq p(|x|)$.

We say that a decision problem $L$ is an **NP** decision problem if there is some **NP** relation $R$ such that $x \in L$ if and only if there is a $y$ such that $(x, y) \in R$. Equivalently, a decision problem $L$ is an **NP** decision problem if there is a polynomial time algorithm $V(\cdot, \cdot)$ and a polynomial $p$ such that $x \in L$ if and only if there is a $y$, $|y| \leq p(|x|)$ such that $V(x, y)$ accepts.

We denote by **NP** the class of **NP** decision problems.

Equivalently, **NP** can be defined as the set of decision problems that are solvable in polynomial time by a non-deterministic Turing machine. Suppose that $L$ is solvable in polynomial time by a non-deterministic Turing machine $M$: then we can define the relation $R$ such that $(x, t) \in R$ if and only if $t$ is a transcript of an accepting computation of $M$ on input $x$ and it's easy to prove that $R$ is an **NP** relation and that $L$ is in **NP** according to our first definition. Suppose that $L$ is in **NP** according to our first definition and that $R$ is the corresponding **NP** relation. Then, on input $x$, a non-deterministic Turing machine can guess a string $y$ of length less than $p(|x|)$ and then accept if and only if $(x, y) \in R$. Such a machine can be implemented to run in non-deterministic polynomial time and it decides $L$.

For a function $t : N \to N$, we define by $DTIME(t(n))$ the set of decision problems that are solvable by a deterministic Turing machine within time $t(n)$ on inputs of length $n$, and by $NTIME(t(n))$ the set of decision problems that are solvable by a non-deterministic Turing machine within time $t(n)$ on inputs of length $n$. Therefore, $\mathbf{P} = \bigcup kDTIME(O(n^k))$ and $\mathbf{NP} = \bigcup kDTIME(O(n^k))$.

## 1.3 NP-completeness

### 1.3.1 Reductions

Let $A$ and $B$ be two decision problems. We say that $A$ reduces to $B$, denoted $A \leq B$, if there is a polynomial time computable function $f$ such that $x \in A$ if and only if $f(x) inB$.

Two immediate observations: if $A \leq B$ and $B$ is in **P**, then also $A \in \mathbf{P}$ (conversely, if $A \leq B$, and $A \notin \mathbf{P}$ then also $B \notin \mathbf{P}$); if $A \leq B$ and $B \leq C$, then also $A \leq C$.

### 1.3.2 NP-completeness

A decision problem $A$ is **NP**-hard if for every problem $L \in \mathbf{NP}$ we have $L \leq A$. A decision problem $A$ is **NP**-complete if it is **NP**-hard and it belongs to **NP**.

It is a simple observation that if $A$ is **NP**-complete, then $A$ is solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

### 1.3.3 An NP-complete problem

Consider the following decision problem, that we call $U$: we are given in input $(M, x, t, l)$ where $M$ is a Turing machine, $x \in \{0,1\}^*$ is a possible input, and $t$ and $l$ are integers encoded in unary[2], and the problem is to determine whether there is a $y \in \{0,1\}^*$, $|y| \leq l$, such that $M(x, y)$ accepts in $\leq t$ steps.

It is immediate to see that $U$ is in **NP**. One can define a procedure $V_U$ that on input $(M, x, t, l)$ and $y$ accepts if and only if $|y| \leq l$, and $M(x, y)$ accepts in at most $t$ steps.

Let $L$ be an **NP** decision problem. Then there are algorithm $V_L$, and polynomials $T_L$ and $p_L$, such that $x \in L$ if and only if there is $y$, $|y| \leq p_L(|x|)$ such that $V_L(x, y)$ accepts; furthermore $V_L$ runs in time at most $T_L(|x| + |y|)$. We give a reduction from $L$ to $U$. The reduction maps $x$ into the instance $f(x) = (V_L, x, T_L(|x| + p_L(|x|)), p_L(|x|))$. Just by applying the definitions, we can see that $x \in L$ if and only $f(x) \in U$.

### 1.3.4 The Problem SAT

In SAT (that stands for *CNF-satisfiability*) we are given Boolean variables $x_1, x_2, \ldots, x_n$ and a Boolean formula $\phi$ involving such variables; the formula is given in a particular format called *conjunctive normal form*, that we will explain in a moment. The question is whether there is a way to assign Boolean (True / False) values to the variables so that the formula is satisfied.

To complete the description of the problem we need to explain what is a Boolean formula in conjunctive normal form. First of all, Boolean formulas are constructed starting from variables and applying the operators $\vee$ (that stands for OR), $\wedge$ (that stands for AND) and $\neg$ (that stands for NOT).

The operators work in the way that one expects: $\neg x$ is True if and only if $x$ is False; $x \wedge y$ is True if and only if both $x$ and $y$ are True; $x \vee y$ is True if and only at least one of $x$ or $y$ is True.

So, for example, the expression $\neg x \wedge (x \vee y)$ can be satisfied by setting $x$ to False and $y$ to True, while the expression $x \wedge (\neg x \vee y) \wedge \neg y$ is impossible to satisfy.

A *literal* is a variable or the negation of a variable, so for example $\neg x_7$ is a literal and so is $x_3$. A *clause* is formed by taking one or more literals and connecting them with a OR, so for example $(x_2 \vee \neg x_4 \vee x_5)$ is a clause, and so is $(x_3)$. A *formula in conjunctive normal form* is the AND of clauses. For example

$$(x_3 \vee \neg x_4) \wedge (x_1) \wedge (\neg x_3 \vee x_2)$$

is a formula in conjunctive normal form (from now on, we will just say "CNF formula" or "formula"). Note that the above formula is satisfiable, and, for example, it is satisfied by setting all the variables to True (there are also other possible assignments of values to the variables that would satisfy the formula).

On the other hand, the formula

$$x \wedge (\neg x \vee y) \wedge \neg y$$

is not satisfiable, as it has already been observed.

[2]The "unary" encoding of an integer $n$ is a sequence of $n$ ones.

**Theorem 1 (Cook)** *SAT is NP-complete.*

## 1.4 Diagonalization

Diagonalization is essentially the only way we know of proving separations between complexity classes. The basic principle is the same as in Cantor's proof that the set of real numbers is not countable: suppose towards a contradiction that the set of real numbers in the range $[0, 1)$ where countable, and let $r_1, \ldots, r_n, \ldots$ be an enumeration. Let $r_i[j]$ be the $j$-th digit in the binary expansion of $r_i$ (that is, $r_i = sum_{j=1}^{\infty} 2^{-j} r_i[j]$). Then define the real number $r$ whose $j$-th digit is $1 - r_j[j]$, that is, $r = \sum_{j=1}^{\infty} 2^{-j}(1 - r_j[j])$. This is a well-defined real number but there can be no $i$ such that $r = r_i$, because $r$ differs from $i$ in the $i$-th digit.

Similarly, we can prove that the Halting problem is undecideble by considering the following decision problem $D$: on input $\langle M \rangle$, the description of a Turing machine, answer NO if $M(\langle M \rangle)$ halts and accepts and YES otherwise. The above problem is decidable if the Halting problem is decidable. However, suppose $D$ where decidable and let $T$ be a Turing machine that solves $D$, then $T(\langle T \rangle)$ halts and accepts if and only if $T(\langle T \rangle)$ does not halt and accept, which is a contradiction.

It is easy to do something similar with time-bounded computations, given the following result (that we will not prove).

**Theorem 2 (Efficient Universal Turing Machine)** *On input the description $\langle M \rangle$ of a Turing machine $M$, a string $x$ and an integer $t > n$, the problem of deciding whether $M$ accepts $x$ within $t$ steps is solvable in $O(t \log t)$ time.*

The $O(t \log t)$ simulation is possible if, say, both the universal machine and the simulated machine are two-tapes Turing machines. A similar theorem holds for any reasonable model of computation, with $t^{O(1)}$ in place of $O(t \log t)$.

**Theorem 3** *$DTIME(n(\log n)^3) \not\subseteq DTIME(O(n))$.*

Proof: Consider the following decision problem $D$: on input $x = (\langle M \rangle, z)$ answer NO if $M(x)$ rejects within $|x| \cdot log|x|$ steps, and YES otherwise.

The problem can be solved in $O(n(\log n)^2)$ time, which is less than $n(\log n)^3$ for all but finitely many inputs.

Suppose by contradiction that $D$ is solvable in time $\leq an + b$ on inputs of length $n$, for constants $a$ and $b$, by a machine $T$. Let $z$ be sufficiently long so that $|\langle T \rangle, z| \log(|\langle T \rangle, z|) > a|\langle T \rangle, z| + b$, then $T(\langle T \rangle, z)$ accepts if and only if it rejects, which is a contradiction. $\Box$

See the homeworks for generalizations of the above proof.

We would like to do the same for non-deterministic time, but we run into the problem that we cannot ask a non-deterministic machine to reject if and only if a non-deterministic machine of comparable running time accepts. If we could do so, then we would be able to prove **NP** $=$ *co***NP**. A considerably subtler argument must be used instead, which uses the following simple fact.

**Theorem 4** *On input the description $\langle M\rangle$ of a non-deterministic Turing machine $M$, a string $x$ and an integer $t > n$, the problem of deciding whether $M$ accepts $x$ within $t$ steps is solvable in deterministic $2^{O(t)}$ time.*

Finally, we can state and prove a special case of the *non-deterministic hierarchy theorem.*

**Theorem 5** $NTIME(n(\log n)^3) \not\subseteq NTIME(O(n))$.

PROOF: Let $f : N \to N$ be defined inductively so that $f(1) = 2$ and $f(k+1) = 2^{kf(k)}$. Consider the following decision problem $D$: on input $x = (\langle M\rangle, 1^t)$, where $M$ is non-deterministic Turing machine,

1. if $t = f(k)$ then the answer is YES if and only if the simulation of $M(\langle M\rangle, 1^{1+f(k-1)})$ as in Theorem 4 returns NO within $t$ steps,

2. otherwise answer YES if and only if $M(\langle M\rangle, 1^{t+1})$ accepts within $t \log t$ steps.

We first observe that $D$ is solvable by a non-deterministic Turing machine running in $O(n(\log n)^2)$ time, where $n$ is the input length, which is less than $n(\log n)^3$ except for finitely many values of $n$.

Suppose that $D$ were decided by a non-deterministic Turing machine $T$ running in time bounded by $an + b$, and consider inputs of the form $\langle T\rangle, 1^t$ (which are solved by $T$ in time $a't + b'$). Pick a sufficiently large $k$, and consider the behaviour of $T$ on inputs $(\langle T\rangle, 1^t)$ for $f(k-1) < t < f(k)$; since all such inputs fall in case (2), we have that $T(\langle T\rangle, 1^t) = T(\langle T\rangle, 1^{t+1})$ for all such $t$ and, in particular,

$$T(\langle T\rangle, 1^{1+f(k-1)}) = T(\langle T\rangle, 1^{f(k)}) \tag{1.1}$$

On the other hand, the input $(\langle T\rangle, 1^{f(k)})$ falls in case (2), and since $T(\langle T\rangle, 1^{1+f(k-1)})$ can be simulated deterministically in time $2^{(k-1)f(k-1)}$, if $k$ is large enough, and so the correct answer on input $(\langle T\rangle, 1^{f(k)})$ is NO if and only if $T(\langle T\rangle, 1^{1+f(k-1)})$ accepts, which is in contradiction to Equation 1.1. □

## 1.5 References

The time hierarchy theorem is proved in [HS65], which is also the paper that introduced the term "Computational Complexity." The non-deterministic hierarchy theorem is due to Cook [Coo73]. The notion of NP-completeness is due to Cook [Coo71] and Levin [Lev73], and the recognition of its generality is due to Karp [Kar72].

## Exercises

1. Show that if $\mathbf{P} = \mathbf{NP}$ for decision problems, then every $\mathbf{NP}$ search problem can be solved in polynomial time.

2. Generalize Theorem 3. Say that a monotone non-decreasing function $t : N \to N$ is time-constructible if, given $n$, we can compute $t(n)$ in $O(t(n))$ time. Show that if $t(n)$ and $t'(n)$ are two time-constructible functions such that $t'(n) > t(n) > n$ and $\lim_{n\to\infty} \frac{t(n)\log t(n)}{t'(n)} = 0$ then $DTIME(t'(n)) \not\subseteq DTIME(t(n))$.

3. The proof of Theorem 3 shows that there is a problem $D$ solvable in $O(n(\log n)^2)$ time such that every machine $M$ running in time $o(n \log n)$ fails at solving $D$ on a constant fraction of all inputs of sufficiently large length. The constant, however, is exponentially small in the representation of $M$.

   Show that there is a problem $D'$ solvable in $O(n(\log n)^2)$ time and such that for every machine $M$ running in time $o(n \log n)$ there is a constant $n_M$ such that, for every $n > n_M$, $M$ fails at solving $D'$ on at least $1/3$ of all inputs of length $n$.

   [Note: this may be an open question. If you can find an oracle relative to which the above statement is false, that would also be interesting, because it would show that the question cannot be solved via a standard diagonalization.]

# Lecture 2

# Space-Bounded Complexity Classes

## 2.1 Space-Bounded Complexity Classes

A machine solves a problem using space $s(\cdot)$ if, for every input $x$, the machine outputs the correct answer and uses only the first $s(|x|)$ cells of the tape. For a standard Turing machine, we can't do better than linear space since $x$ itself must be on the tape. So we will often consider a machine with multiple tapes: a read-only "input" tape, a read/write "work" or "memory" tape, and possibly a write-once "output" tape. Then we can say the machine uses space $s$ if for input $x$, it uses only the first $s(|x|)$ cells of the work tape.

We denote by ł the set of decision problems solvable in $O(\log n)$ space. We denote by **PSPACE** the set of decision problems solvable in polynomial space. A first observation is that a space-efficient machine is, to a certain extent, also a time-efficient one. In general we denote by **SPACE**$(s(n))$ the set of decision problems that can be solved using space at most $s(n)$ on inputs of length $n$.

**Theorem 6** *If a machine always halts, and uses $s(\cdot)$ space, with $s(n) \geq \log n$, then it runs in time $2^{O(s(n))}$.*

Proof: Call the "configuration" of a machine $M$ on input $x$ a description of the state of $M$, the position of the input tape, and the contents of the work tape at a given time. Write down $c_1, c_2, \ldots, c_t$ where $c_i$ is the configuration at time $i$ and $t$ is the running time of $M(x)$. No two $c_i$ can be equal, or else the machine would be in a loop, since the $c_i$ completely describes the present, and therefore the future, of the computation. Now, the number of *possible* configurations is simply the product of the number of states, the number of positions on the input tape, and the number of possible contents of the work tape (which itself depends on the number of allowable positions on the input tape). This is

$$O(1) \cdot n \cdot |\Sigma|^{s(n)} = 2^{O(s(n))+\log n} = 2^{O(s(n))}$$

Since we cannot visit a configuration twice during the computation, the computation must therefore finish in $2^{O(s(n))}$ steps. $\Box$

**NL** is the set of decision problems solvable by a non-deterministic machine using $O(\log n)$ space. **NPSPACE** is the set of decision problems solvable by a non-deterministic machine using polynomial space. In general we denote by $\mathbf{NSPACE}(s(n))$ the set of decision problems that can be solved by non-deterministic machines that use at most $s(n)$ bits of space on inputs of length $n$.

Analogously with time-bounded complexity classes, we could think that **NL** is exactly the set of decision problems that have "solutions" that can verified in log-space. If so, **NL** would be equal to **NP**, since there is a log-space algorithm $V$ that verifies solutions to SAT. However, this is unlikely to be true, because **NL** is contained in **P**. An intuitive reason why not all problems with a log-space "verifier" can be simulated in **NL** is that an **NL** machine does not have enough memory to keep track of all the non-deterministic choices that it makes.

**Theorem 7** $\mathbf{NL} \subseteq \mathbf{P}$.

Proof: Let $L$ be a language in **NL** and let $M$ be a non-deterministic log-space machine for $L$. Consider a computation of $M(x)$. As before, there are $2^{O(s(n))} = n^{O(1)}$ possible configurations. Consider a directed graph in which vertices are configurations and edges indicate transitions from one state to another which the machine is allowed to make in a single step (as determined by its $\delta$). This graph has polynomially many vertices, so in polynomial time we can do a depth-first search to see whether there is a path from the initial configuration that eventually leads to acceptance. This describes a polynomial-time algorithm for deciding $L$, so we're done. $\Box$

## 2.2 Reductions in NL

We would like to introduce a notion of completeness in **NL** analogous to the notion of completeness that we know for the class **NP**. A first observation is that, in order to have a meaningful notion of completeness in **NL**, we cannot use polynomial-time reductions, otherwise any **NL** problem having at least a YES instance and at least a NO instance would be trivially **NL**-complete. To get a more interesting notion of **NL**-completeness we need to turn to weaker reductions. In particular, we define *log space* reductions as follows:

**Definition 1** *Let $A$ and $B$ be decision problems. We say $A$ is log space reducible to $B$, $A \leq_{\log} B$, if $\exists$ a function $f$ computable in log space such that $x \in A$ iff $f(x) \in B$, and $B \in L$.*

**Theorem 8** *If $B \in \mathbf{L}$, and $A \leq_{\log} B$, then $A \in \mathbf{L}$.*

Proof: We consider the concatenation of two machines: $M_f$ to compute $f$, and $M_B$ to solve $B$. If our resource bound was polynomial time, then we would use $M_f(x)$ to compute $f(x)$, and then run $M_B$ on $f(x)$. The composition of the two procedures would given an algorithm for $A$, and if both procedures run in polynomial time then their composition is also polynomial time. To prove the theorem, however, we have to show that if $M_f$ and $M_B$ are log space machines, then their composition can also be computed in log space.

Recall the definition of a Turing machine $M$ that has a log space complexity bound: $M$ has one read-only input tape, one write-only output tape, and uses a log space work tape.

A naive implementation of the composition of $M_f$ and $M_B$ would be to compute $f(x)$, and then run $M_B$ on input $f(x)$; however $f(x)$ needs to be stored on the work tape, and this implementation does not produce a log space machine. Instead we modify $M_f$ so that on input $x$ and $i$ it returns the $i$-th bit of $f(x)$ (this computation can still be carried out in logarithmic space). Then we run a simulation of the computation of $M_B(f(x))$ by using the modified $M_f$ as an "oracle" to tell us the value of specified positions of $f(x)$. In order to simulate $M_B(f(x))$ we only need to know the content of one position of $f(x)$ at a time, so the simulation can be carried with a total of $O(\log |x|)$ bits of work space. □

Using the same proof technique, we can show the following:

**Theorem 9** *if $A \leq_{\log} B, B \leq_{\log} C$, then $A \leq_{\log} C$.*

## 2.3 NL Completeness

Armed with a definition of log space reducibility, we can define **NL**-completeness.

**Definition 2** *$A$ is **NL**-hard if $\forall B \in \mathbf{NL}, B \leq_{\log} A$. $A$ is **NL**-complete if $A \in \mathbf{NL}$ and $A$ is **NL**-hard.*

We now introduce a problem STCONN (s,t-connectivity) that we will show is **NL**-complete. In STCONN, given in input a directed graph $G(V, E)$ and two vertices $s, t \in V$, we want to determine if there is a directed path from $s$ to $t$.

**Theorem 10** *STCONN is **NL**-complete.*

PROOF:

1. STCONN $\in$ **NL**.

   On input *G(V,E)*, *s*,*t*, set $p$ to $s$. For $i = 1$ to $|V|$, nondeterminsitically, choose a neighboring vertex $v$ of $p$. Set $p = v$. If $p = t$, accept and halt. Reject and halt if the end of the *for* loop is reached. The algorithm only requires $O(\log n)$ space.

2. STCONN is **NL**-hard.

   Let $A \in \mathbf{NL}$, and let $M_A$ be a non-deterministic logarithmic space Turing Machine for $A$. On input $x$, construct a directed graph $G$ with one vertex for each configuration of $M(x)$, and an additional vertex $t$. Add edges $(c_i, c_j)$ if $M(x)$ can move in one step from $c_i$ to $c_j$. Add edges $(c, t)$ from every configuration that is accepting, and let $s$ be the start configuration. $M$ accepts $x$ iff some path from $s$ to $t$ exists in $G$. The above graph can be constructed from $x$ in log space, because listing all nodes requires $O(\log n)$ space, and testing valid edges is also easy.

□

## 2.4 Savitch's Theorem

What kinds of tradeoffs are there between memory and time? STCONN can be solved deterministically in linear time and linear space, using depth-first-search. Is there some sense in which this is optimal? Nondeterministically, we can search using less than linear space. Can searching be done deterministically in less than linear space?

We will use Savitch's Theorem [Sav70] to show that STCONN can be solved deterministically in $O(\log^2 n)$, and that every **NL** problem can be solved deterministically in $O(\log^2 n)$ space. In general, if $A$ is a problem that can be solved nondeterministically with space $s(n) \geq \log n$, then it can be solved deterministically with $O(s^2(n))$space.

**Theorem 11** *STCONN can be solved deterministically in $O(\log^2 n)$ space.*

PROOF: Consider a graph $G(V, E)$, and vertices $s, t$. We define a recursive function REACH$(u, v, k)$ that accepts and halts iff $v$ can be reached from $u$ in $\leq k$ steps. If $k = 1$, then REACH accepts iff $(u, v)$ is an edge. If $k \geq 2, \forall w \in V - \{u, v\}$, compute REACH$(u, w, \lfloor \mathrm{k}/2 \rfloor)$ and REACH$(w, v, \lceil \mathrm{k}/2 \rceil)$. If both accept and halt, accept. Else, reject.

Let $S(k)$ be the worst-case space use of REACH$(\cdot, \cdot, k)$. The space required for the base case $S(1)$ is a counter for tracking the edge, so $S(1) = O(\log n)$. In general, $S(k) = O(\log n) + S(k/2)$ for calls to REACH and for tracking $w$. So, $S(k) = O(\log k * \log n)$. Since $k \leq n$, the worst-case space use of REACH is $O(\log^2 n)$. $\Box$

Essentially the same proof applies to arbitrary non-deterministic space-bounded computations.

**Theorem 12 (Savitch's Theorem)** *For every function $s(n)$ computable in space $O(s(n))$,* $\mathbf{NSPACE}(s) = \mathbf{SPACE}(O(s^2))$

PROOF: We begin with a nondeterministic machine $M$, which on input $x$ uses $s(|x|)$ space. We define $REACH(c_i, c_j, k)$, as in the proof of Theorem 11, which accepts and halts iff $M(x)$ can go from $c_i$ to $c_j$ in $\leq k$ steps. We compute $REACH(c_0, c_{\text{acc}}, 2^O(s|x|))$ for all accepting configurations $c_{\text{acc}}$. If there is a call of REACH which accepts and halts, then $M$ accepts. Else, $M$ rejects. If REACH accepts and halts, it will do so in $\leq 2^{O(|x|)}$ steps.

Let $S_R(k)$ be the worst-case space used by REACH$(\cdot, \cdot, k)$: $S_R(1) = O(s(n)), S_R(k) = O(s(n)) + S_R(k/2)$. This solves $S_R = s(n) * \log k$, and, since $k = 2^O(s(n))$, we have $S_R = O(s^2(n))$. $\Box$

Comparing Theorem 11 to depth-first-search, we find that we are exponentially better in space requirements, but we are no longer polynomial in time.

Examining the time required, if we let $t(k)$ be the worst-case time used by REACH$(\cdot, \cdot, k)$, we see $t(1) = O(n + m)$, and $t(k) = n(2 * T(k/2))$, which solves to $t(k) = n^{O(\log k)} = O(n^{O(\log n)})$, which is super-polynomial. Savitch's algorithm is still the one with the best known space bound. No known algorithm achieves polynomial log space and polynomial time simultaneously, although such an algorithm is known for *undirected* connectivity.

## 2.5 Undirected Connectivity

In the undirected $s - t$ connectivity problem (abbreviated ST-UCONN) we are given an undirected graph $G = (V, E)$ and two vertices $s, t \in V$, and the question is whether that is a path between $s$ and $t$ in $G$.

While this problem is not known to be complete for **NL**, and it probably is not, ST-UCONN is complete for the class **SL** of decision problems that are solvable by *symmetric* non-deterministic machines that use $O(\log n)$ space. A non-deterministic machine is symmetric if whenever it can make a transition from a global state $s$ to a global state $s'$ then the transition from $s'$ to $s$ is also possible. The proof of **SL**-completeness of ST-UCONN is identical to the proof of **NL**-completeness of ST-CONN except for the additional observation that the transition graph of a symmetric machine is undirected.

For ST-UCONN there exists an algorithm running in polynomial time and $O(\log^2 n)$ space (but the polynomial has very high degree), due to Nisan [Nis94]. There is also an algorithm that has $O(\log^{4/3} n)$ space complexity and superpolynomial time complexity, due to Armoni, Ta-Shma, Nisan and Wigderson [ATSWZ97], improving on a previous algorithm by Nisan, Szemeredy and Wigderson [NSW92].

## 2.6 Randomized Log-space

We now wish to introduce *randomized* space-bounded Turing machine. For simplicity, we will only introduce randomized machines for solving decision problems. In addition to a read-only input tape and a read/write work tape, such machines also have a read-only *random* tape to which they have *one-way* access, meaning that the head on that tape can only more, say, left-to-right. For every fixed input and fixed content of the random tape, the machine is completely deterministic, and either accepts or rejects. For a Turing machine $M$, an input $x$ and a content $r$ of the random tape, we denote by $M(r, x)$ the outcome of the computation.

We say that a decision problem $L$ belongs to the class **RL** (for *randomized* log-space) if there is a probabilistic Turing machine $M$ that uses $O(\log n)$ space on inputs of length $n$ and such that

- For every $x \in L$, $\mathbf{Pr}_r[M(r, x) \text{ accepts }] \geq 1/2$
- For every $x \notin L$, $\mathbf{Pr}_r[M(r, x) \text{ accepts }] = 0$.

It is easy to observe that any constant bigger than 0 and smaller than 1 could be equivalently used instead of 1/2 in the definition above. It also follows from the definition that $\mathbf{L} \subseteq \mathbf{RL} \subseteq \mathbf{NL}$.

The following result shows that, indeed, $\mathbf{L} \subseteq \mathbf{SL} \subseteq \mathbf{RL} \subseteq \mathbf{NL}$.

**Theorem 13** *The problem ST-UCONN is in* **RL**.

We will not give a proof of the above theorem, but just describe the algorithm. Given an undirected graph $G = (V, E)$ and two vertices $s, t$, the algorithm performs a random walk of length $100 \cdot n^3$ starting from $s$. If $t$ is never reached, the algorithm rejects.

```
input: $G = (V, E)$, $s$, $t$

$v \leftarrow s$
for $i \leftarrow 1$ to $100 \cdot n^3$
    pick at random a neighbor $w$ of $v$
    if $w = t$ then halt and accept
    $v \leftarrow w$ reject
```

The analysis of the algorithm is based on the fact that if we start a random walk from a vertex $s$ of an undirected vertex $G$, then all vertices in the connected component of $s$ are likely to be visited at least once after $\Theta(n^3)$ steps.

The best known deterministic simulation of **RL** uses $O((\log n)^{3/2})$ space, and is due to Saks and Zhou [SZ95].

## 2.7 NL = coNL

In order to prove that these two classes are the same, we will show that there is an **NL** Turing machine which solves $\overline{\text{STCONN}}$. $\overline{\text{STCONN}}$ is the problem of deciding, given a directed graph $G$, together with special vertices $s$ and $t$, whether $t$ is *not* reachable from $s$. Note that $\overline{\text{STCONN}}$ is **coNL**-complete.

Once we have the machine, we know that **coNL** $\subseteq$ **NL**, since any language $A$ in **coNL** can be reduced to $\overline{\text{STCONN}}$, and since $\overline{\text{STCONN}}$ has been shown to be in **NL** (by the existence of our machine), so is $A$. Also, **NL** $\subseteq$ **coNL**, since if $\overline{\text{STCONN}} \in$ **NL**, by definition STCONN $\in$ **coNL**, and since STCONN is **NL**-complete, this means that any problem in **NL** can be reduced to it and so is also in **coNL**. Hence **NL** = **coNL**. This result was proved independently in [Imm88] and [Sze88].

### 2.7.1 A simpler problem first

Now all that remains to be shown is that this Turing machine exists. First we will solve a simpler problem than $\overline{\text{STCONN}}$. We will assume that in addition to the usual inputs $G$, $s$ and $t$, we also have an input $r$, which we will assume is equal to the number of vertices reachable from $s$ in $G$, including $s$.

Given these inputs, we will construct a non-deterministic Turing machine which decides whether $t$ is reachable from $s$ by looking at all subsets of $r$ vertices in $G$, halting with YES if it sees a subset of vertices which are all reachable from $s$ but do not include $t$, and halting with NO otherwise. Here is the algorithm:

```
input: $G = (V, E)$, $s$, $t$, $r$
output: YES if it discovers that $t$ is not reachable from $s$, and NO otherwise
assumption: there are exactly $r$ distinct vertices reachable from $s$

$c \leftarrow 0$
**for all** $v \in (V - \{t\})$ **do**
        non-deterministically guess if $v$ is reachable from $s$
        **if** guess = YES **then**
            non-deterministically guess the distance $k$ from $s$ to $v$
            $p \leftarrow s$
            **for** $i \leftarrow 1$ to $k$ **do**
                non-deterministically pick a neighbor $q$ of $p$
                $p \leftarrow q$
            if $p \neq v$, reject
            $c \leftarrow c + 1$
**if** $c = r$ **then** return YES, otherwise return NO
```

It is easy to verify that this algorithm is indeed in **NL**. The algorithm only needs to maintain the five variables $c, k, p, q, v$, and each of these variables can be represented with $\log |V|$ bits.

Regarding correctness, notice that, in the algorithm, $c$ can only be incremented for a vertex $v$ that is actually reachable from $s$. Since there are assumed to be exactly $r$ such vertices, $c$ can be at most $r$ at the end of the algorithm, and if it is exactly $r$, that means that there are $r$ vertices other than $t$ which are reachable from $s$, meaning that $t$ by assumption cannot be reachable form $s$. Hence the algorithm accepts if and only if it discovers that $t$ is not reachable from $s$.

### 2.7.2 Finding $r$

Now we need to provide an **NL**-algorithm that finds $r$. Let's first try this algorithm:

input: $G = (V, E)$, $s$
output: the number of vertices reachable from $s$ (including $s$ in this count)

```
$c \leftarrow 0$
for all $v \in V$ do
    non-deterministically guess if $v$ is reachable from $s$ in $k$ steps
    if guess = YES then
        $p \leftarrow s$
        for $i \leftarrow 1$ to $k$ do
            non-deterministically guess a neighbor $q$ of $p$ (possibly not moving at all)
            $p \leftarrow q$
        if $p \neq v$ reject
        $c \leftarrow c + 1$
return $c$
```

**This algorithm has a problem**. It will only return a number $c$ which is at most $r$, but we need it to return *exactly* $r$. We need a way to force it to find all vertices which are reachable from $s$. Towards this goal, let's define $r_k$ to be the set of vertices reachable from $s$ in at most $k$ steps. Then $r = r_{n-1}$, where $n$ is the number of vertices in $G$. The idea is to try to compute $r_k$ from $r_{k-1}$ and repeat the procedure $n-1$ times, starting from $r_0 = 1$. Now here is another try at an algorithm:

input: $G = (V, E)$, $s$, $k$, $r_{k-1}$
output: the number of vertices reachable from $s$ in at most $k$ steps (including $s$ in this count)
assumption: $r_{k-1}$ is the exact number of vertices reachable from $s$ in at most $k-1$ steps

```
$c \leftarrow 0$
for all $v \in V$ do
    $d \leftarrow 0$
    $flag \leftarrow FALSE$
    for all $w \in V$ do
        $p \leftarrow s$
        for $i \leftarrow 1$ to $k-1$ do
            non-deterministically pick a neighbor $q$ of $p$ (possibly not moving at all)
            $p \leftarrow q$
```

```
                if $p = w$ then
                    $d \leftarrow d + 1$
                    if $v$ is a neighbor of $w$, or if $v = w$ then
                        $flag \leftarrow TRUE$
        if $d < r_{k-1}$ reject
        if $flag$ then $c \leftarrow c + 1$
return $c$
```

Here is the idea behind the algorithm: for each vertex $v$, we need to determine if it is reachable from $s$ in at most $k$ steps. To do this, we can loop over all vertices which are a distance at most $k-1$ from $s$, checking to see if $v$ is either equal to one of these vertices or is a neighbor of one of them (in which case it would be reachable in exactly $k$ steps). The algorithm is able to force all vertices of distance at most $k-1$ to be considered because it is given $r_{k-1}$ as an input.

Now, putting this algorithm together with the first one listed above, we have shown that $\overline{\text{STCONN}} \in \mathbf{NL}$, implying that $\mathbf{NL} = \mathbf{coNL}$. In fact, the proof can be generalized to show that if a decision problem $A$ is solvable in non-deterministic space $s(n) = \Omega(\log n)$, then $\overline{A}$ is solvable in non-deterministic space $O(s(n))$.

## Exercises

1. It is possible to prove that there is an algorithm that given a Turing machine $M$ (of the type with a read/only input tape and a work tape), an input $x$ and a space bound $s$, decides whether $M(x)$ accepts and uses $\leq s$ bits of space; the algorithm uses $O(s)$ bits of space. Using the above result, argue that if $s(n) > \log n$ is a function computable in $O(s(n))$ space, then $\mathbf{SPACE}(s(n)\log n) \not\subseteq \mathbf{SPACE}(s(n))$.

2. Prove that $\mathbf{P} \neq \mathbf{SPACE}(O(n))$.

   [This is a "trick" question. It is not know how to prove $\mathbf{P} \not\subseteq \mathbf{SPACE}(O(n))$ or how to prove $\mathbf{SPACE} \not\subseteq \mathbf{P}$, so the proof has to start by assuming the classes are equal and then reach a contradiction, without explicitely showing a problem in one class that cannot belong to the other. Hint: if $A$ and $B$ are decision problems, $A \leq B$ and $B \in \mathbf{P}$ then $A \in \mathbf{P}$; what would happen if the same were true for $\mathbf{SPACE}(O(n))$?]

3. Define the class **BPL** (for *bounded-error probabilistic log-space*) as follows. A decision problem $L$ is in **BPL** if there is a log-space probabilistic Turing machine $M$ such that

   - If $x \in L$ then $\mathbf{Pr}_r[M(r,x) \text{ accepts }] \geq 2/3$;
   - If $x \notin L$ then $\mathbf{Pr}_r[M(r,x) \text{ accepts }] \leq 1/3$.

   Then

   (a) Prove that $\mathbf{RL} \subseteq \mathbf{BPL}$.

   (b) Prove that $\mathbf{BPL} \subseteq \mathbf{SPACE}(O((\log n)^2)$.

   (c) This might be somewhat harder: prove that $\mathbf{BPL} \subseteq \mathbf{P}$.

# Lecture 3

# The Polynomial Hierarchy

One way to look at the difference between **NP** and **coNP** is that a decision problem in **NP** is asking a sort of "does there exist" question, where the existence of the answer can by definition be efficiently represented. On the other hand, **coNP** asks "is it true for all" questions, which do not seem to have simple, efficient proofs.

Formally, a decision problem $A$ is in **NP** if and only if there is a polynomial time procedure $V(\cdot,\cdot)$ and a polynomial time bound $p()$ such that

$$x \in A \text{ if and only if } \exists y.|y| \leq p(|x|) \wedge V(x,y) = 1$$

and a problem $A$ is in **coNP** if and only if there is a polynomial time procedure $V(\cdot,\cdot)$ and a polynomial bound $p()$ such that

$$x \in A \text{ if and only if } \forall y : |y| \leq p(|x|), V(x,y) = 1$$

## 3.1 Stacks of quantifiers

Now suppose you had a decision problem $A$ which asked something of the following form:

$$x \in A \Leftrightarrow \exists\ y_1 \text{ s.t. } |y_1| \leq p(|x|)\ \forall\ y_2 \text{ s.t. } |y_2| \leq p(|x|)\ V(x, y_1, y_2)$$

(where $p()$ is a polynomial time bound and $V(\cdot,\cdot,\cdot)$ is a polynomial time procedure.)

In other words, an algorithm solving problem $A$ should return YES on an input $x$ if an only if there exists some string $y_1$ such that for all strings $y_2$ (both of polynomial length), the predicate $V(x, y_1, y_2)$ holds. An example of such a problem is this: given a Boolean formula $\phi$ over variables $x_1, \ldots, x_n$, is there a formula $\phi'$ which is equivalent to $\phi$ and is of size at most $k$? In this case, $y_1$ is the formula $\phi'$, $y_2$ is an arbitrary assignment to the variables $x_1, \ldots, x_n$, and $V(x, y_1, y_2)$ is the predicate which is true if and only if $x[y_2]$ and $y_1[y_2]$ are both true or both false, meaning that under the variable assignment $y_2$, $\phi$ and $\phi'$ agree. Notice that $\phi'$ is equivalent to $\phi$ if and only if it agrees with $\phi$ under all assignments of Boolean values to the variables.

As we will see, the problem $A$ is a member of the class $\Sigma_2$ in the second level of the polynomial hierarchy.

## 3.2 The hierarchy

The polynomial hierarchy starts with familiar classes on level one: $\Sigma_1 = \mathbf{NP}$ and $\Pi_1 = \mathbf{coNP}$. For all $i \geq 1$, it includes two classes, $\Sigma_i$ and $\Pi_i$, which are defined as follows:

$$A \in \Sigma_i \Leftrightarrow \exists y_1.\ \forall y_2.\ \ldots\ .Q y_i.\ V_A(x, y_1, \ldots, y_i)$$

and

$$B \in \Pi_i \Leftrightarrow \forall y_1.\ \exists y_2.\ \ldots\ .Q' y_i.\ V_B(x, y_1, \ldots, y_i)$$

where the predicates $V_A$ and $V_B$ depend on the problems $A$ and $B$, and $Q$ and $Q'$ represent the appropriate quantifiers, which depend on whether $i$ is even or odd (for example, if $i = 10$ then the quantifier $Q$ for $\Sigma_{10}$ is $\forall$, and the quantifier $Q'$ for $\Pi_{10}$ is $\exists$). For clarity, we have also omitted the $p(\cdot)$ side conditions, but they are still there.

One thing that is easy to see is that $\Pi_k = \mathrm{co}\Sigma_k$. Also, note that, for all $i \leq k-1$, $\Pi_i \subseteq \Sigma_k$ and $\Sigma_i \subseteq \Sigma_k$. These subset relations hold for $\Pi_k$ as well. This can be seen by noticing that the predicates $V$ do not need to "pay attention to" all of their arguments, and so can represent classes lower on the hierarchy which have a smaller number of them.

## 3.3 An alternate characterization

The polynomial hierarchy can also be characterized in terms of "oracle machines." The idea here is that, instead of a standard Turing machine, we consider one which is augmented with an oracle of a certain power which can be consulted as many times as desired, and using only one computational step each time. Syntactically, this can be written as follows.

Let $A$ be some decision problem and $\mathcal{M}$ be a class of Turing machines. Then $\mathcal{M}^A$ is defined to be the class of machines obtained from $\mathcal{M}$ by allowing instances of $A$ to be solved in one step. Similarly, if $\mathcal{M}$ is a class of Turing machines and $\mathcal{C}$ is a complexity class, then $\mathcal{M}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathcal{M}^A$. If $L$ is a complete problem for $\mathcal{C}$, and the machines in $\mathcal{M}$ are powerful enough to compute polynomial-time computations, then $\mathcal{M}^{\mathcal{C}} = \mathcal{M}^L$.

**Theorem 14** $\Sigma_2 = \mathbf{NP}^{3SAT}$.

Proof: Let $A \in \Sigma_2$, then for some polynomial $p()$ and polynomial-time computable $V()$ we have

$$x \in A \text{ if and only if } \exists y_1 \text{ s.t. } |y_1| \leq p(|x|).\forall y_2 \text{ s.t. } |y_2| \leq p(|x|).V(x, y_1, y_2) = 1$$

Then we define a non-deterministic machine with an **NP**-oracle as follows: on input $x$, the machine guesses a string $y_1$ of length at most $p(|x|)$, and then asks the oracle whether $\exists y_2.|y_2| \leq p(|x|) \wedge V(x, y_1, y_2) = 0$. The above question is an existential question about a polynomial-time computation, so, by Cook's theorem, it is possible to construct in polynomial time a 3SAT instance that is satisfiable if and only if the answer to the above question is YES. The machine accepts if and only if the answer from the oracle is NO. It is immediate that the machine has an accepting computation if and only if

$$\exists y_1.|y_1| \leq p(|x|).(\neg \exists y_2.|y_2| \leq p(|x|) \wedge V(x, y_1, y_2) = 0)$$

that is, the machine accepts if and only if $x \in A$.

Notice that, in the above computation, only one oracle query is made, even though the definition of $\mathbf{NP}^{3SAT}$ allows us to make an arbitrary number of oracle queries.

Let now $A \in \mathbf{NP}^{3SAT}$, and let $M$ be the oracle machine that solves $A$. We first show that there is a machine $M'$ that also solves $A$, only makes one oracle query, and accepts if and only if the answer to the oracle query is NO. On input $x$, $M'$ guesses an accepting computation of $M(x)$, that is, $M'$ guesses all the non-deterministic choices of $M(x)$, all the oracle questions, and all the answers. Then, for each question that was answered with a YES, $M'$ guesses a satisfying assignment to verify that the guess was correct. Finally, $M'$ is left with a certain set of oracle questions, say, the formulae $\phi_1, \ldots, \phi_k$, for which it has guessed that the correct oracle answer is NO. Then $M'$ asks its oracle whether (a formula equivalent to) $\phi_1 \vee \cdots \vee \phi_k$ is satisfiable, and it accepts if and only if the answer is NO.

Consider the computation of $M'(x)$ when $x \in A$: there is a valid accepting computation of $M(x)$, and $M'(x)$ can guess that computation along with the valid oracle answers; it can also guess valid assignments for all the queries for which the answer is YES; finally, it is left with unsatisfiable formulae $\phi_1, \ldots, \phi_k$, the answer to the single oracle query of $M'$ is NO, and $M'$ accepts.

Conversely, if $M'(x)$ has an accepting computation, then there must be a valid accepting computation of $M(x)$, and so $x \in A$. $\Box$

In fact, a more general result is known, whose proof works along similar lines.

**Theorem 15** *For every $i \geq 2$, $\Sigma_i = \mathbf{NP}^{\Sigma_{i-1}}$.*

## 3.4 Additional Properties

Here are some more facts about the polynomial hierarchy, which we will not prove:

1. $\Pi_i$ and $\Sigma_i$ have complete problems for all $i$.
2. A $\Sigma_i$-complete problem is not in $\Pi_j$, $j \leq i-1$, unless $\Pi_j = \Sigma_i$, and it is not in $\Sigma_j$ unless $\Sigma_j = \Sigma_i$.
3. Suppose that $\Sigma_i = \Pi_i$ for some $i$. Then $\Sigma_j = \Pi_j = \Sigma_i = \Pi_i$ for all $j \geq i$.
4. Suppose that $\Sigma_i = \Sigma_{i+1}$ for some $i$. Then $\Sigma_j = \Pi_j = \Sigma_i$ for all $j \geq i$.
5. Suppose that $\Pi_i = \Pi_{i+1}$ for some $i$. then $\Sigma_j = \Pi_j = \Pi_i$ for all $j \geq i$.

We will just prove the following special case of part (3).

**Theorem 16** *Suppose* $\mathbf{NP} = \mathbf{coNP}$. *Then, for every $i \geq 2$, $\Sigma_i = \mathbf{NP}$.*

Proof: Let us first prove that, under the assumption of the theorem, $\Sigma_2 = \mathbf{NP}$. Let $A \in \Sigma_2$ and let $M$ be the non-deterministic oracle machine that decides $A$ using oracle access to 3SAT. Let also $M'$ be the non-deterministic polynomial time Turing machine that decides the complement of the 3SAT problem. We now describe a non-deterministic polynomial time Turing machine $M''$ to decide $A$: on input $x$, $M''$ guesses an accepting computation of $M(x)$, along with oracle queries and answers; for each oracle question $\phi$ for which a YES answer has been guessed, $M''$ guesses a satisfying assignment; for each oracle question $\psi$ for which a NO answer has been guessed, $M''$ guesses an accepting computation of $M'(\psi)$. It is easy to verify that $M''(x)$ has an accepting computation if and only if $M^{3SAT}(x)$ has an accepting computation.

We can prove by induction on $i$ that $\Sigma_i = \mathbf{NP}$. We have covered the base case. Let us now suppose that $\Sigma_{i-1} = \mathbf{NP}$; then $\Sigma_i = \mathbf{NP}^{\Sigma_{i-1}} = \mathbf{NP}^{\mathbf{NP}} = \Sigma_2 = \mathbf{NP}$. $\Box$

While it seems like an artificial construction right now, in future lectures we will see that the polynomial hierarchy helps us to understand other complexity classes.

## 3.5 References

The polynomial time hierarchy was defined by Stockmeyer [Sto76]. Wrathall [Wra76] shows that every class in the polynomial hierarchy has complete problems.

## Exercises

1. Consider the following decision problem: given a directed graph $G = (V, E)$, two vertices $s, t \in V$, and an integer $k$, determine whether the shortest path from $s$ to $t$ is of length $k$. Is this problem in **NL**?

2. In the MAX SAT problem we are given a formula $\phi$ in conjunctive normal form and we want to find the assignment of values to the variables that maximizes the number of satisfied clauses. (For example, if $\phi$ is satisfiable, the optimal solution satisfies all the clauses and the MAX SAT problem reduces to finding a satisfying assignment.) Consider the following decision problem: given a formula $\phi$ in conjunctive normal form and an integer $k$, determine if $k$ is the number of clauses of $\phi$ satisfied by an optimal assignment.

    - Prove that this problem is in **NP** if and only if **NP** = *co***NP**.
      [Hint: prove that it is both **NP**-hard and *co***NP**-hard.]
    - Prove that this problem is in $\Sigma_2$.

# Lecture 4

# Circuits

This lecture is on boolean circuit complexity. We first define circuits and the function they compute. Then we consider families of circuits and the language they define. In Section 4.2, we see how circuits relate to other complexity classes by a series of results, culminating with the Karp-Lipton theorem which states that if **NP** problems can be decided with polynomial-size circuits, then **PH** $= \Sigma_2$.

## 4.1 Circuits

A circuit $C$ has $n$ inputs, $m$ outputs, and is constructed with AND gates, OR gates and NOT gates. Each gate has in-degree 2 except the NOT gate which has in-degree 1. The out-degree can be any number. A circuit must have no cycle. See Figure 4.1.

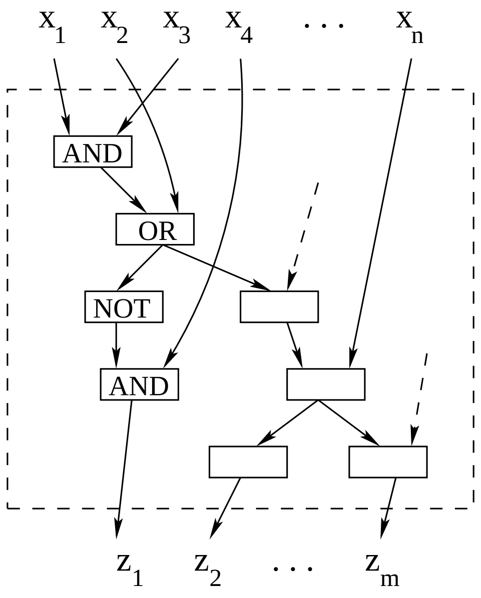


Figure 4.1: A Boolean circuit.

A circuit $C$ with $n$ inputs and $m$ outputs computes a function $f_C : \{0,1\}^n \to \{0,1\}^m$.

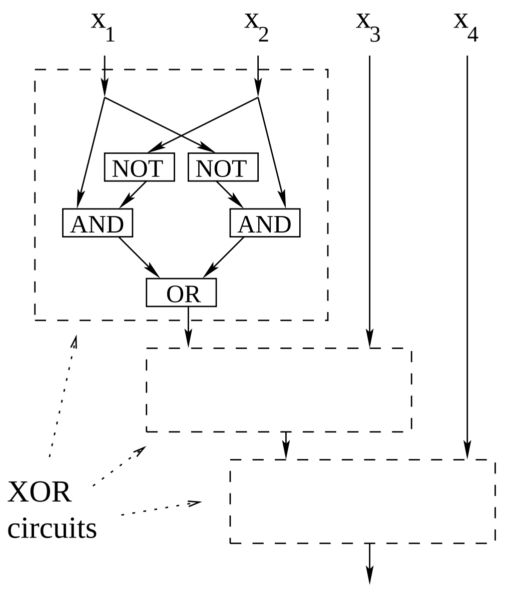


Figure 4.2: A circuit computing the boolean function $f_C(x_1x_2x_3x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

See Figure 4.2 for an example.

Define $\mathbf{SIZE}(C) = \#$ of AND and OR gates of $C$. By convention, we do *not* count the NOT gates.

To be compatible with other complexity classes, we need to extend the model to arbitrary input sizes:

**Definition 3** *A language $L$ is solved by a family of circuits $\{C_1, C_2, \ldots, C_n, \ldots\}$ if for every $n \geq 1$ and for every $x$ s.t. $|x| = n$,*

$$x \in L \;\Leftrightarrow\; f_{C_n}(x) = 1.$$

**Definition 4** *Say $L \in \mathbf{SIZE}(s(n))$ if $L$ is solved by a family $\{C_1, C_2, \ldots, C_n, \ldots\}$ of circuits, where $C_i$ has at most $s(i)$ gates.*

## 4.2 Relation to other complexity classes

**Proposition 17** *For every language $L$, $L \in \mathbf{SIZE}(O(2^n))$. In other words, exponential size circuits contain all languages.*

PROOF: We need to show that for every 1-output function $f : \{0,1\}^n \to \{0,1\}$, $f$ has circuit size $O(2^n)$.

Use the identity $f(x_1x_2\ldots x_n) = (x_1 \wedge f(1x_2\ldots x_n)) \vee (\overline{x}_1 \wedge f(0x_2\ldots x_n))$ to recursively construct a circuit for $f$, as shown in Figure 4.3.

The recurrence relation for the size of the circuit is: $s(n) = 3 + 2s(n-1)$ with base case $s(1) = 1$, which solves to $s(n) = 2 \cdot 2^n - 3 = O(2^n)$. $\Box$

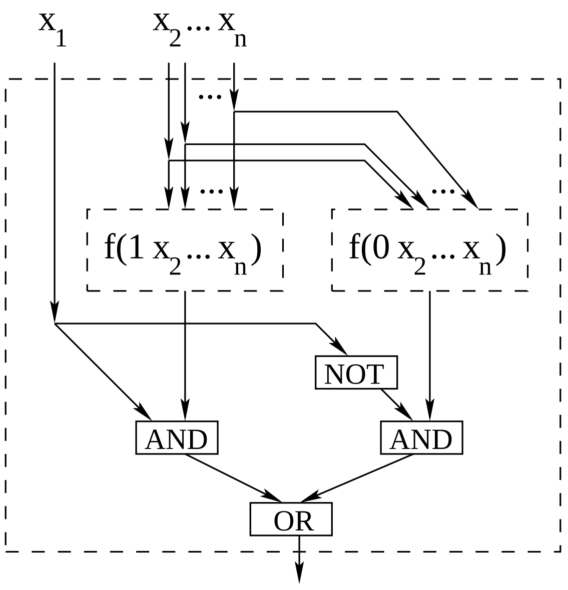


Figure 4.3: A circuit computing any function $f(x_1x_2 \ldots x_n)$ of $n$ variables assuming circuits for two functions of $n-1$ variables.

**Proposition 18** *If $L \in \mathbf{DTIME}(t(n))$, then $L \in \mathbf{SIZE}(O(t^2(n)))$.*

Proof: Let $L$ be a decision problem solved by a machine $M$ in time $t(n)$. Fix $n$ and $x$ s.t. $|x| = n$, and consider the $t(n) \times t(n)$ tableau of the computation of $M(x)$. See Figure 4.4.

Assume that each entry $(a, q)$ of the tableau is encoded using $k$ bits. By Proposition 17, the transition function $\{0,1\}^{3k} \to \{0,1\}^k$ used by the machine can be implemented by a "next state circuit" of size $k \cdot O(2^{3k})$, which is exponential in $k$ but constant in $n$. This building block can be used to create a circuit of size $O(t^2(n))$ that computes the complete tableau, thus also computes the answer to the decision problem. This is shown in Figure 4.5. □

**Corollary 19** $\mathbf{P} \subseteq \mathbf{SIZE}(n^{O(1)})$.

On the other hand, it's easy to show that $\mathbf{P} \neq \mathbf{SIZE}(n^{O(1)})$.

**Proposition 20** *There are languages $L$ such that $L \notin \mathbf{SIZE}(2^{o(n)})$. In other words, for every $n$, there exists $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^{o(n)}$.*

Proof: This is a counting argument. There are $2^{2^n}$ functions $f : \{0,1\}^n \to \{0,1\}$, and we claim that the number of circuits of size $s$ is at most $2^{O(s \log s)}$, assuming $s \geq n$. To bound the number of circuits of size $s$ we create a compact binary encoding of such circuits. Identify gates with numbers $1, \ldots, s$. For each gate, specify where the two inputs are coming from, whether they are complemented, and the type of gate. The total number of bits required to represent the circuit is

$$s(2\log(n+s)+3) \leq s(2\log 2s + 3) = s(2\log 2s + 5).$$

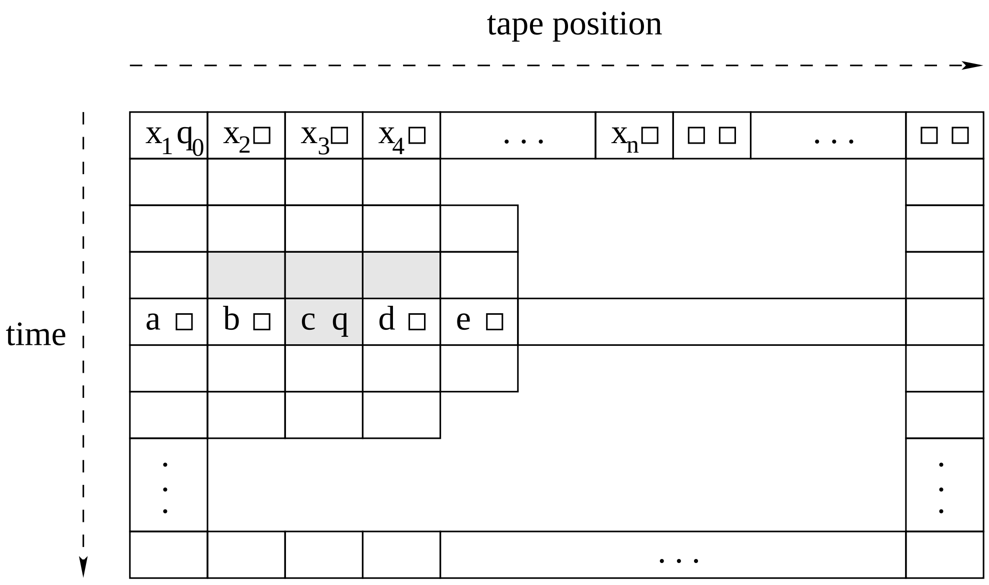


Figure 4.4: $t(n) \times t(n)$ tableau of computation. The left entry of each cell is the tape symbol at that position and time. The right entry is the machine state or a blank symbol, depending on the position of the machine head.

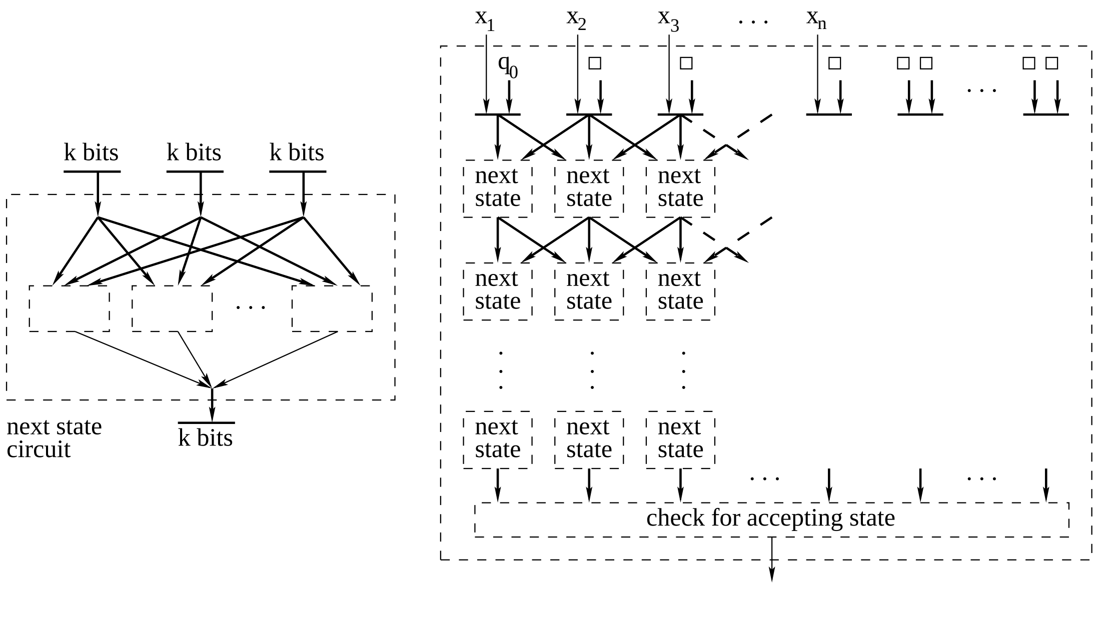


Figure 4.5: Circuit to simulate a Turing machine computation by constructing the tableau.

So the number of circuits of size $s$ is at most $2^{2s\log s+5s}$, and this is not sufficient to compute all possible functions if

$$2^{2s\log s+5s} < 2^{2^n}.$$

This is satisfied if $s = 2^{o(n)}$. $\Box$

**Theorem 21 (Karp-Lipton-Sipser)** *If* $\mathbf{NP} \subseteq \mathbf{SIZE}(n^{O(1)})$ *then* $\mathbf{PH} = \Sigma_2$. *In other words, the polynomial hierarchy would collapse to its second level.*

Before proving the above theorem, we first show a result that contains some of the ideas in the proof of the Karp-Lipton-Sipser theorem.

**Lemma 22** *If* $\mathbf{NP} \subseteq \mathbf{SIZE}(n^{O(1)})$ *then there is a family of polynomial-size circuits that on input a 3CNF formula* $\phi$ *outputs a satisfying assignment for* $\phi$ *if one such assignment exists and a sequence of zeroes otherwise.*

Proof: We define the circuits $C_n^1, \ldots, C_n^n$ as follows:

- $C_n^1$, on input a formula $\phi$ over $n$ variables outputs 1 if and only if there is a satisfying assignment for $\phi$ where $x_1 = 1$,
- $\cdots$
- $C_n^i$, on input a formula $\phi$ over $n$ variables and bits $b_1, \ldots, b_{i-1}$, outputs 1 if and only if there is a satisfying assignment for $\phi$ where $x_1 = b_1, \ldots, x_{i-1} = b_{i-1}, x_i = 1$
- $\cdots$
- $C_n^n$, on input a formula $\phi$ over $n$ variables and bits $b_1, \ldots, b_{n-1}$, outputs 1 if and only if $\phi$ is satisfied by the assignment $x_1 = b_1, \ldots, x_{n-1} = b_{n-1}, x_n = 1$.

Also, each circuit realizes an **NP** computation, and so it can be built of polynomial size. Consider now the sequence $b_1 = C_n^1(\phi)$, $b_2 = C_n^2(b_1, \phi)$, $\ldots$, $b_n C_n^n(b_1, \ldots, b_{n-1}, \phi)$. The reader should be able to convince himself that this is a satisfying assignment for $\phi$ if $\phi$ is satisfiable, and a sequence of zeroes otherwise. $\Box$

We now prove the Karp-Lipton-Sipser theorem.

Proof: [Of Theorem 21] We will show that if $\mathbf{NP} \subseteq \mathbf{SIZE}(n^{O(1)})$ then $\Pi_2 \subseteq \Sigma_2$. By a result in a previous lecture, this implies that $\mathbf{PH} = \Sigma_2$.

Let $L \in \Pi_2$, then there is a polynomial $p()$ and a polynomial-time computable $V()$ such that

$$x \in L \text{ iff } \forall y_1.|y_1| \leq p(|x|) \exists y_2.|y_2| \leq p(|x|).V(x, y_1, y_2) = 1$$

By adapting the proof of Lemma 22 (see Figure 4.6), or by using the statement of the Lemma and Cook's theorem, we can show that, for every $n$, there is a circuit $C_n$ of size polynomial in $n$ such that for every $x$ and every $y_1$, $|y_1| \leq p(|x|)$,

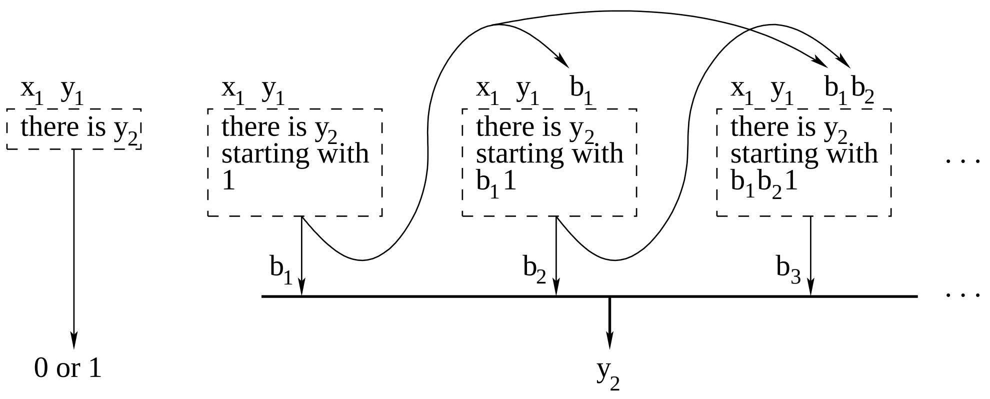

x1 y1
there is y2
0 or 1
x1 y1
there is y2 starting with 1
x1 y1 b1
there is y2 starting with b1 1
x1 y1 b1 b2
there is y2 starting with b1 b2 1
...
b1
b2
b3
...
y2

Figure 4.6: How to use decision problem solvers to find a witness to a search problem.

$$\exists y_2 . |y_2| \leq p(|x|) \wedge V(x, y_1, y_2) = 1 \text{ iff } V(x, y_1, C_n(x, y_1)) = 1$$

Let $q(n)$ be a polynomial upper bound to the size of $C_n$.

So now we have that for inputs $x$ of length $n$,

$$x \in L \text{ iff } \exists C_n . |C_n| \leq q(n) . \forall y_1 . |y_1| \leq p(n) . V(x, y_1, C_n(x, y_1)) = 1$$

which shows that $L$ is in $\Sigma_2$. □

## 4.3 References

The Karp-Lipton-Sipser theorem appears in [KL80].

## Exercises

1. Show that $\mathbf{SIZE}(n^{O(1)}) \not\subseteq \mathbf{P}$.
2. Show that there is a language in $\mathbf{SPACE}(2^{n^{O(1)}})$ that does not belong to $\mathbf{SIZE}(2^{o(n)})$.
3. Define $\mathbf{EXP} = \mathbf{DTIME}(2^{n^{O(1)}})$. Prove that if $\mathbf{EXP} \subseteq \mathbf{SIZE}(n^{O(1)})$ then $\mathbf{EXP} = \Sigma_2$.

# Lecture 5

# Probabilistic Complexity Classes

In this lecture we will define the probabilistic complexity classes **BPP**, **RP**, **ZPP** and we will see how they are related to each other, as well as to other deterministic or circuit complexity classes.

## 5.1 Probabilistic complexity classes

First we are going to describe the probabilistic model of computation. In this model an algorithm $A$ gets as input a sequence of random bits $r$ and the "real" input $x$ of the problem. The output of the algorithm is the correct answer for the input $x$ with some probability.

**Definition 5** *An algorithm $A$ is called a polynomial time probabilistic algorithm if the size of the random sequence $|r|$ is polynomial in the input $|x|$ and $A()$ runs in time polynomial in $|x|$.*

If we want to talk about the correctness of the algorithm, then informally we could say that for every input $x$ we need $\mathbf{Pr}[A(x,r) = \text{correct answer for} x] \geq \frac{2}{3}$. That is, for every input the probability distribution over all the random sequences must be some constant bounded away from $\frac{1}{2}$. Let us now define the class **BPP**.

**Definition 6** *A decision problem $L$ is in* **BPP***if there is a polynomial time algorithm $A$ and a polynomial $p()$ such that :*

$$\forall x \in L \quad \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = 1] \geq 2/3$$

$$\forall x \notin L \quad \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = 1] \leq 1/3$$

We can see that in this setting we have an algorithm with two inputs and some constraints on the probabilities of the outcome. In the same way we can also define the class **P** as:

**Definition 7** *A decision problem $L$ is in* **P***if there is a polynomial time algorithm $A$ and a polynomial $p()$ such that :*

$$\forall x \in L \quad : \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = 1] = 1$$

$$\forall x \notin L : \mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = 1] = 0$$

Similarly, we define the classes **RP**and **ZPP**.

**Definition 8** *A decision problem $L$ is in* **RP***if there is a polynomial time algorithm $A$ and a polynomial $p()$ such that :*

$$\begin{aligned} \forall x \in L \quad & \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] \geq 1/2 \\ \forall x \notin L \quad & \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = 1] \leq 0 \end{aligned}$$

**Definition 9** *A decision problem $L$ is in* **ZPP***if there is a polynomial time algorithm $A$ whose output can be $0, 1, ?$ and a polynomial $p()$ such that :*

$$\forall x \quad \mathbf{Pr}_{r \in \{0,1\}^{p(|x|)}}[A(x,r) = ?] \leq 1/2$$

$$\forall x, \forall r \quad \textit{such that} \quad A(x,r) \neq ? \quad \textit{then} \quad A(x,r) = 1 \quad \textit{if and only if} \quad x \in L$$

## 5.2 Relations between complexity classes

After defining these probabilistic complexity classes, let's see how they are related to other complexity classes and with each other.

**Theorem 23** **RP**$\subseteq$**NP**.

PROOF: Suppose we have an algorithm for **RP**. Then this algorithm is also in **NP**. If $x \in L$ then there is a random sequence $r$, for which the algorithm answers yes. On the other hand, if $x \notin L$ then there is no witness. □

Most natural probabilistic algorithms belong to the class **RP**. We can also show that the class **ZPP** is no larger than **RP**.

**Theorem 24** **ZPP**$\subseteq$**RP**.

PROOF: We are going to convert a **ZPP** algorithm into an **RP** algorithm. The construction consists of running the **ZPP** algorithm and anytime it outputs ?, the new algorithm will answer 0. In this way, if the right answer is 0, then the algorithm will answer 0 with probability 1. On the other hand, when the right answer is 1, then the algorithm will give the wrong answer with probability less than 1/2, since the probability of the **ZPP** algorithm giving the output ? is less than 1/2. □

Another interesting property of the class **ZPP** is that it's equivalent to the class of languages for which there is an average polynomial time algorithm that always gives the right answer. More formally,

**Theorem 25** *A language $L$ is in the class* **ZPP***if and only if $L$ has an average polynomial time algorithm that always gives the right answer.*

PROOF: First let us clarify what we mean by average time. For each input $x$ we take the average time of $A(x,r)$ over all random sequences $r$. Then for size $n$ we take the worst time over all possible inputs $x$ of size $|x| = n$. In order to construct an algorithm that always gives the right answer we run the **ZPP**algorithm and if it outputs a ?, then we run it again.

Suppose that the running time of the **ZPP**algorithm is $T$, then the average running time of the new algorithm is:

$$T_{avg} = \frac{1}{2} \cdot T + \frac{1}{4} \cdot 2T + \ldots + \frac{1}{2^k} \cdot kT \approx O(T)$$

Now, we want to prove that if the language $L$ has an algorithm that runs in polynomial average time $t(|x|)$, then this is in **ZPP**. What we do is run the algorithm for time $2t(|x|)$ and output a ? if the algorithm has not yet stopped. It is straightforward to see that this belongs to **ZPP**. First of all, the worst running time is polynomial, actually $2t(|x|)$. Moreover, the probability that our algorithm outputs a ? is less than 1/2, since the original algorithm has an average running time $t(|x|)$ and so it must stop before time $2t(|x|)$ at least half of the times. □

Let us now prove the fact that **RP**is contained in **BPP**.

**Theorem 26 RP⊆BPP**

PROOF: We will convert an **RP**algorithm into a **BPP**algorithm. In the case that the input $x$ does not belong to the language then the **RP**algorithm always gives the right answer, so this is definitely in **BPP**as well. In the case that the input $x$ does belong to the language then we need to boost the probability of a correct answer from at least 1/2 to at least 2/3.

More formally, let $A$ be an **RP**algorithm for a decision problem $L$. We fix some number $k$ and define the following algorithm:

```
$A^{(k)}$

input: $x$,
  pick $r_1, r_2, \ldots, r_k$
  if  $A(x, r_1) = A(x, r_2) = \ldots = A(x, r_k) = 0$ then  return 0
  else return 1
```

Let us now consider the correctness of the algorithm.In case the correct answer is 0 the output is always right, though in the case where the right answer is 1 the output is right except when all $A(x, r_i) = 0$.

$$\begin{aligned} \text{if} \quad x \notin L \quad & \mathbf{Pr}_{r_1,\ldots,r_k}[A^k(x, r_1, \ldots, r_k) = 1] = 0 \\ \text{if} \quad x \in L \quad & \mathbf{Pr}_{r_1,\ldots,r_k}[A^k(x, r_1, \ldots, r_k) = 1] \geq 1 - \left(\frac{1}{2}\right)^k \end{aligned}$$

It is easy to see that by choosing an appropriate $k$ the second probability can go arbitrarily close to 1 and therefore become larger than 2/3, which is what is required by the definition of **BPP**. In fact, by choosing $k$ to be a polynomial in $|x|$, we can make the probability exponentially close to 1. This enables us to change the definition of **RP**and instead of the bound of 1/2 for the probability of a correct answer when the input is in the language $L$, we can have a bound of $1 - \left(\frac{1}{2}\right)^{q(|x|)}$, for a fixed polynomial $q$. □

Let, now, A be a **BPP** algorithm for a decision problem $L$. Then, we fix $k$ and define the following algorithm:

```
$A^{(k)}$

input: $x$,
  pick $r_1, r_2, \dots, r_k$
  $c = \sum_{i=1}^{k} A(x, r_i)$
  **if** $c \geq \frac{k}{2}$ **then return** 1
  **else return** 0
```

In a **BPP** algorithm we expect the right answer to come up with probability more than 1/2. So, by running the algorithm many times we make sure that this slightly bigger than 1/2 probability will actually show up in the results. More formally let us define the Chernoff bounds.

**Theorem 27** *(Chernoff Bound)*
*Suppose $X_1, \dots, X_k$ are independent random variables with values in $\{0,1\}$ and for every $i$, $\mathbf{Pr}[X_i = 1] = p$. Then:*

$$\mathbf{Pr}[\tfrac{1}{k}\sum_{i=1}^{k} X_i - p > \epsilon] < e^{-\epsilon^2 \frac{k}{2p(1-p)}}$$

$$\mathbf{Pr}[\tfrac{1}{k}\sum_{i=1}^{k} X_i - p < -\epsilon] < e^{-\epsilon^2 \frac{k}{2p(1-p)}}$$

The Chernoff bounds will enable us to bound the probability that our result is far from the expected. Indeed, these bounds say that this probability is exponentially small in respect to $k$.

Let us now consider how the Chernoff bounds apply to the algorithm we described previously. We fix the input $x$ and call $p = \mathbf{Pr}_r[A(x,r) = 1]$ over all possible random sequences. We also define the independent random variables $X_1, \dots, X_k$ such that $X_i = A(x, r_i)$.

First, suppose $x \in L$. Then the algorithm $A^{(k)}(x, r_1, \dots, r_k)$ outputs the right answer 1, when $\frac{1}{k}\sum_i X_i \geq \frac{1}{2}$. So, the algorithm makes a mistake when $\frac{1}{k}\sum_i X_i < \frac{1}{2}$.

We now apply the Chernoff bounds to bound this probability.

$$\mathbf{Pr}[A^{(k)}\text{outputs the wrong answer on } x] = \mathbf{Pr}[\tfrac{1}{k}\sum_i X_i < \tfrac{1}{2}]$$
$$\leq \mathbf{Pr}[\tfrac{1}{k}\sum_i X_i - p \leq -\tfrac{1}{6}]$$

since $p \geq \frac{2}{3}$.

$$\leq e^{-\frac{k}{72p(1-p)}} = 2^{-\Omega(k)}$$

The probability is exponentially small in $k$. The same reasoning applies also for the case where $x \notin L$. Further, it is easy to see that by choosing $k$ to be a polynomial in $|x|$ instead of a constant, we can change the definition of a **BPP**algorithm and instead of the bound of $\frac{1}{3}$ for the probability of a wrong answer, we can have a bound of $2^{-q(|x|)}$, for a fixed polynomial $q$.

Next, we are going to see how the probabilistic complexity classes relate to circuit complexity classes and specifically prove that the class **BPP**has polynomial size circuits.

**Theorem 28 (Adleman) BPP⊆SIZE**$(n^{O(1)})$

Proof: Let $L$ be in the class **BPP**. Then by definition, there is a polynomial time algorithm $A$ and a polynomial $p$, such that for every input $x$

$$\mathbf{Pr}_{r\in\{0,1\}^{p(|x|)}}[A(x,r) = \text{wrong answer for} \quad x] \leq 2^{-(n+1)}$$

This follows from our previous conclusion that we can replace $\frac{1}{3}$ with $2^{-q(|x|)}$. We now fix $n$ and try to construct a family of circuits $C_n$, that solves $L$ on inputs of length $n$.

**Claim 29** *There is a random sequence $r \in \{0,1\}^{p(n)}$ such that for every $x \in \{0,1\}^n$ $A(x,r)$ is correct.*

Proof: Informally, we can see that for each input $x$ the number of random sequences $r$ that give the wrong answer is exponentially small. Therefore, even if we assume that these sequences are different for every input $x$, their sum is still less than the total number of random sequences. Formally, let's consider the probability over all sequences that the algorithm gives the right answer for all input. If this probability is greater than 0, then the claim is proved.

$$\mathbf{Pr}_r[\text{for every} \quad x, A(x,r) \quad \text{is correct}] = 1 - \mathbf{Pr}_r[\exists x, A(x,r) \quad \text{is wrong}]$$

the second probability is the union of $2^n$ possible events for each $x$. This is bounded by the sum of the probabilities.

$$\begin{aligned} &\geq 1 - \sum_{x\in\{0,1\}^n} \mathbf{Pr}_r[A(x,r)\text{is wrong}] \\ &\geq 1 - 2^n \cdot 2^{-(n+1)} \\ &\geq \frac{1}{2} \end{aligned}$$

□

So, we proved that at least half of the random sequences are correct for all possible input $x$. Therefore, it is straightforward to see that we can simulate the algorithm $A(\cdot,\cdot)$, where the first input has length $n$ and the second $p(n)$, by a circuit of size polynomial in $n$.

All we have to do is find a random sequence which is always correct and build it inside the circuit. Hence, our circuit will take as input only the input $x$ and simulate $A$ with input

$x$ and $r$ for this fixed $r$. Of course, this is only an existential proof, since we don't know how to find this sequence efficiently. □

In conclusion, let us briefly describe some other relations between complexity classes. Whether **BPP** $\subseteq$ **NP** or not is still an open question. What we know is that it's unlikely that **NP**is contained in **BPP**, since then by the previous result **NP** would have polynomial size circuits and hence by the result of Karp and Lipton the polynomial hierarchy would collapse.

## 5.3 BPP$\subseteq \Sigma_2$

This result was first shown by Sipser and Gacs. Lautemann gave a much simpler proof which we give below.

**Lemma 30** *If $L$ is in* **BPP** *then there is an algorithm $A$ such that for every $x$,*

$$\mathbf{Pr}_r(A(x,r) = \textit{right answer}) \geq 1 - \tfrac{1}{3m},$$

*where the number of random bits $|r| = m = |x|^{O(1)}$ and $A$ runs in time $|x|^{O(1)}$.*

Proof: Let $\hat{A}$ be a **BPP** algorithm for $L$. Then for every $x$, $\mathbf{Pr}_r(\hat{A}(x,r) = \text{wrong answer}) \leq \frac{1}{3}$. and $\hat{A}$ uses $\hat{m}(n)$ random bits where $n = |x|$.

Do $k(n)$ repetitions of $\hat{A}$ and accept if and only if at least $\dfrac{k(n)}{2}$ executions of $\hat{A}$ accept. Call the new algorithm $A$. Then $A$ uses $k(n)\hat{m}(n)$ random bits and $\mathbf{Pr}_r(A(x,r) = \text{wrong answer}) \leq 2^{-ck(n)}$. We can then find $k(n)$ with $k(n) = \Theta(\log \hat{m}(n))$ such that $\frac{1}{2^{ck(n)}} \leq \frac{1}{3k(n)\hat{m(n)}}$. □

**Theorem 31** **BPP**$\subseteq \Sigma_2$.

Proof: Let $L$ be in **BPP** and $A$ as in the claim. Then we want to show that

$$x \in L \Leftrightarrow \exists y_1, \ldots, y_m \in \{0,1\}^m \forall z \in \{0,1\}^m \bigvee_{i=1}^{m} A(x, y_i \oplus z) = 1$$

where $m$ is the number of random bits used by $A$ on input $x$.
Suppose $x \in L$. Then

$$\begin{aligned}
&\mathbf{Pr}_{y_1,\ldots,y_m}(\exists z A(x, y_1 \oplus z) = \cdots = A(x, y_m \oplus z) = 0) \\
&\leq \sum_{z \in \{0,1\}^m} \mathbf{Pr}_{y_1,\ldots,y_m}(A(x, y_1 \oplus z) = \cdots = A(x, y_m \oplus z) = 0) \\
&\leq 2^m \frac{1}{(3m)^m} \\
&< 1.
\end{aligned}$$

So

$$\mathbf{Pr}_{y_1,\ldots,y_m}(\forall z \bigvee_i A(x, y_i \oplus z)) = 1 - \mathbf{Pr}_{y_1,\ldots,y_m}(\exists z A(x, y_1 \oplus z) = \cdots = A(x, y_m \oplus z) = 0)$$
$$> 0.$$

So $(y_1, \ldots, y_m)$ exists.
Conversely suppose $x \notin L$. Then

$$\begin{aligned}\mathbf{Pr}_z\left(\bigvee_i A(x, y_i \oplus z)\right) &\le \sum_i \mathbf{Pr}_z\left(A(x, y_i \oplus z) = 1\right)\\ &\le m \cdot \frac{1}{3m}\\ &= \frac{1}{3}.\end{aligned}$$

So

$$\begin{aligned}\mathbf{Pr}_z(A(x, y_1 \oplus z) = \cdots = A(x, y_m \oplus z) = 0) &= \mathbf{Pr}_z\left(\bigvee_i A(x, y_i \oplus z)\right)\\ &\ge \frac{2}{3}\\ &> 0.\end{aligned}$$

So there is a $z$ such that $\bigvee_i A(x, y_i \oplus z) = 0$ for all $y_1, \ldots, y_m \in \{0,1\}^m$. $\square$

## 5.4 References

Probabilistic complexity classes were defined in [Gil77]. Adleman's proof that $\mathbf{BPP} \subseteq \mathbf{SIZE}(n^{O(1)})$ appears in [Adl78]. Sipser's proof that $\mathbf{BPP} \subseteq \Sigma_2$ appears in [Sip83], and Lautemann's proof is in [Lau83].

## Exercises

1. Prove that **ZPP**=**RP**$\cap$**coRP**
2. Show that if **NP** $\subseteq$ **BPP** then **NP** = **RP**.
3. Argue that **SPACE**$(O(n^{\log n})) \not\subseteq$ **BPP**.

Lecture 6

# A Probabilistic Algorithm

## 6.1 Branching Programs

We begin by introducing another model of computation, that of branching programs. A branching program accepts or rejects input strings $x_1, \ldots, x_n$ and can be described as a directed acyclic graph with a single root and two final states, 0 and 1. The computational path begins at the root. Each non-final vertex is labeled with some single variable $x_i$ and has two outgoing edges, one labeled 0 and the other 1. The path follows edge $b$ such that $x_i = b$, and the program accepts the input if and only if the path ends at the final state 1.

An illustrative example is the branching program that considers input string $x_1x_2x_3$ and outputs 1 if $x_1 \oplus x_2 \oplus x_3 = 1$. This is shown in figure 6.1.

Branching programs are a reasonably powerful model of computation; for example we can show that **L** can be computed by branching programs of polynomial size.

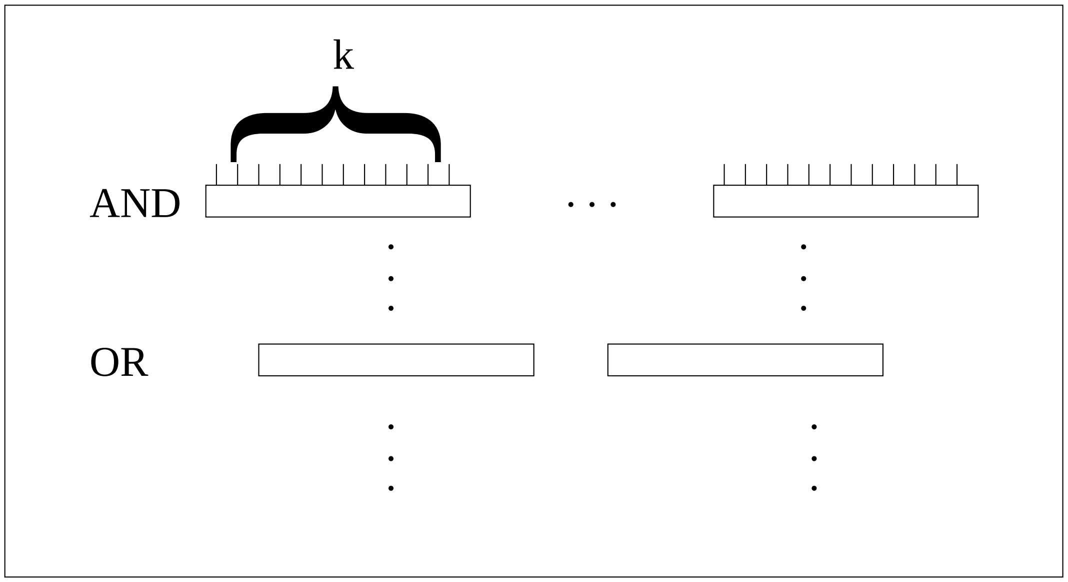


Figure 6.1: A branching program.

## 6.2 Testing Equivalence

We would like to be able to consider two branching programs and determine if they compute the same function. In general, testing equivalence for any model of computation is difficult: for Turing machines, the problem is undecidable, and for circuits, the problem is **coNP**-complete. A similar result is true for branching programs– here, testing equivalence is also **coNP**-complete, through a reduction from co-3SAT.

Happily, the problem becomes easier once we restrict it to *read-once* branching programs, in which every path from the root to a final state tests each variable at most once. We will show that in this case testing equivalence is in **coRP**. Our strategy will be as follows: Given two read-once branching programs $B_1$ and $B_2$, we will define two polynomials $p_1$ and $p_2$ of degree $n$ over $x_1, \ldots, x_n$ that are equivalent if and only if $B_1$ and $B_2$ are also equivalent. We will then use a **coRP**algorithm to test the equivalence of the two polynomials. The polynomials will be represented in compact form, but will have an exponential number of monomials and thus we cannot simply directly compare their coefficients.

Our polynomials will be determined as follows:

- We assign each vertex $v_i$ in the branching program from the root to the final states a polynomial $p_i(x_1, \ldots, x_n)$, beginning with the root vertex which gets the constant polynomial 1.
- Once a vertex has been assigned a polynomial $p(x_1, \ldots, x_n)$, its outgoing edges are then given polynomials in the following manner: If the vertex performed a test on $x_j$, the outgoing 0 edge receives the polynomial $(1 - x_j)p(x_1, \ldots, x_n)$ and the outgoing 1 edge receives $(x_j)p(x_1, \ldots, x_n)$.
- Once all of a vertex's incoming edges have been appropriately labeled, that vertex's polynomial becomes the sum of the polynomials of the incoming edges.
- The polynomial associated with the final state 1 is the polynomial of the branching program.

These rules are illustrated in figure 6.2. The idea is that the polynomial at each vertex is nonzero on a set of inputs only if that set of inputs will lead the branching program to that vertex.

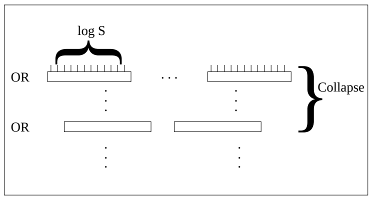


Figure 6.2: Constructing the polynomial of a branching program.

We can see that when the branching programs are read-once, the degree of a variable in any monomial in the resulting polynomial will have power no larger than 1, and hence the polynomial will be of degree at most $n$.

We now observe that given read-once branching programs $B_1$ and $B_2$, their corresponding polynomials $p_1$ and $p_2$ are equivalent if and only if $B_1$ and $B_2$ are equivalent. To see this, if any monomial $m$ in $p_1$ or $p_2$ does not contain a variable $x_i$ we replace $m$ with $mx_i + m(1 - x_i)$. We can then write $p_1$ and $p_2$ as sums of terms of the form $\prod_{i=1}^{n} y_i$, where $y_i$ is either $x_i$ or $(1 - x_i)$. Each term corresponds to a path in the branching program that ends in final state 1. From this, some extra thought reveals that equivalence of the polynomials is equivalent to equivalence of the branching programs.

Notice also that if the branching programs are polynomial size, we can evaluate the polynomials efficiently, even though they may contain an exponential number of coefficients. All we need now is a method of testing equivalence of polynomials efficiently.

## 6.3 The Schwartz-Zippel Lemma

The tool we will use for this is the Schwartz-Zippel lemma, which states:

**Lemma 32** *If $p(x_1, \ldots, x_n)$ is an $n$-variate nonzero polynomial of degree $d$ over a finite field $\mathbb{F}$, then $p$ has at most $d\mathbb{F}^{n-1}$ roots. Equivalently, $\mathbf{Pr}[p(a_1, \ldots, a_n) = 0] \leq d/\mathbb{F}$.*

Notice that we use $\mathbb{F}$ to be both the field and its size.

PROOF: The proof proceeds by induction on $n$. The base case of $n = 1$ is simply that of a univariate polynomial and follows immediately.

Now consider $p(x_1, \ldots, x_n)$ of degree at most $d$. We can write

$$p(x_1, \ldots, x_n) = p_0(x_2, \ldots, x_n) + x_1 p_1(x_2, \ldots, x_n) + x_1^2 p_2(x_2, \ldots, x_n) + \ldots + x_1^k p_k(x_2, \ldots, x_n)$$

where $k$ is the largest value for which $p_k$ is not identically zero.

We now observe that

$$|\{a_1, \ldots, a_n\} : p(a_1, \ldots, a_n) \neq 0| \geq |\{a_1, \ldots, a_n\} : p(a_1, \ldots, a_n) \neq 0 \text{ and } p_k(a_2, \ldots, a_n) \neq 0|$$

By induction, $p_k$ is nonzero on at least $\mathbb{F}^{n-1} - (d-k)\mathbb{F}^{n-2}$ points. Further, for each such point $a_2, \ldots, a_n$, $p(x_1, a_2, \ldots, a_n)$ is a nonzero univariate polynomial and hence there are at least $\mathbb{F} - k$ values of $a_1$ such that $p(a_1, \ldots, a_n) \neq 0$.

Putting this together, we can bound the number of nonzero points from below by

$$\mathbb{F}^{n-1}\left(1 - \frac{d-k}{\mathbb{F}}\right)\mathbb{F}\left(1 - \frac{k}{\mathbb{F}}\right) \geq \mathbb{F}^n\left(1 - \frac{d}{\mathbb{F}}\right) = \mathbb{F}^n - d\mathbb{F}^{n-1}$$

This finishes the proof. □

An algorithm for testing equivalence of polynomials and hence read-once branching programs is now immediate. We choose a field of size at least $3d$, and then choose random $a_1, \ldots, a_n$ from $\mathbb{F}^n$. We accept if $p_1(a_1, \ldots, a_n) = p_2(a_1, \ldots, a_n)$.

If the two polynomials are equivalent, then we always accept; otherwise reject with probability at least 2/3, making this a **coRP** algorithm.

## 6.4 References

The Schwartz-Zippel theorem is from [Sch80, Zip79]. The algorithm for testing equivalence of read-once branching programs is due to Blum, Chandra and Wegman [BCW80].

## Exercises

1. Prove that testing equivalence of general branching programs is co**NP**-complete.

2. Show that every Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a read-once branching program (possibly of exponential size).

# Lecture 7

# Unique-SAT

## 7.1 The Valiant-Vazirani Reduction

In this section we show the following: suppose there is an algorithm for the satisfiability problem that always find a satisfying assignment for formulae that have exactly one satisfiable assignment (and behaves arbitrarily on other instances): then we can get an **RP** algorithm for the general satisfiability problem, and so **NP** = **RP**.

We prove the result by presenting a randomized reduction that given in input a CNF formula $\phi$ produces in output a polynomial number of formulae $\psi_0, \ldots, \psi_n$. If $\phi$ is satisfiable, then (with high probability) at least one of the $\psi_i$ is satisfiable and has exactly one satisfying assignment; if $\phi$ is not satisfiable, then (with probability one) all $\psi_i$ are unsatisfiable.

The idea for the reduction is the following. Suppose $\phi$ is a satisfiable formula with $n$ variables that has about $2^k$ satisfying assignments, and let $h : \{0,1\}^n \to \{0,1\}^k$ be a hash function picked from a family of pairwise independent hash functions: then the average number of assignments $x$ such that $\phi(x)$ is true and $h(x) = (0, \ldots, 0)$ is about one. Indeed, we can prove formally that with constant probability there is exactly one such assignment,[1] and that there is CNF formula $\psi$ (easily constructed from $\phi$ and $h$) that is satisfied precisely by that assignment. By doing the above construction for values of $k$ ranging from 0 to $n$, we obtain the desired reduction. Details follow.

**Definition 10** *Let $H$ be a family of functions of the form $h : \{0,1\}^n \to \{0,1\}^m$. We say that $H$ is a family of pair-wise independent hash functions if for every two different inputs $x, y \in \{0,1\}^n$ and for every two possible outputs $a, b \in \{0,1\}^m$ we have*

$$\mathbf{Pr}_{h \in H}[h(x) = a \wedge h(y) = b] = \frac{1}{2^{2m}}$$

Another way to look at the definition is that for every $x \neq y$, when we pick $h$ at random then the random variables $h(x)$ and $h(y)$ are independent and uniformly distributed. In particular, for every $x \neq y$ and for every $a, b$ we have $\mathbf{Pr}_h[h(x) = a | h(y) = b] = \mathbf{Pr}_h[h(x) = a]$.

[1]For technical reasons, it will be easier to prove that this is the case when picking a hash function $h : \{0,1\}^n \to \{0,1\}^{k+2}$.

For $m$ vectors $a_1, \ldots, a_m \in \{0,1\}^m$ and $m$ bits $b_1, \ldots, b_m$, define $h_{a_1,\ldots,a_m,b_1,\ldots,b_m} : \{0,1\}^n \to \{0,1\}^m$ as $h_{\mathbf{a},b}(x) = (a_1 \cdot x + b_1, \ldots, a_m \cdot x + b_m)$, and let $H_{AFF}$ be the family of functions defined this way. Then it is not hard to see that $H_{AFF}$ is a family of pairwise independent hash functions.

**Lemma 33** *Let $T \subseteq \{0,1\}^n$ be a set such that $2^k \leq |T| < 2^{k+1}$ and let $H$ be a family of pairwise independent hash functions of the form $h : \{0,1\}^n \to \{0,1\}^{k+2}$. Then if we pick $h$ at random from $H$, there is a constant probability that there is a unique element $x \in T$ such that $h(x) = \mathbf{0}$. Precisely,*

$$\mathbf{Pr}_{h\in H}[|\{x \in T : h(x) = \mathbf{0}\}| = 1] \geq \frac{1}{8}$$

Proof: Let us fix an element $x \in T$. We want to compute the probability that $x$ is the *unique* element of $T$ mapped into $\mathbf{0}$ by $h$. Clearly,

$$\mathbf{Pr}_h[h(x) = \mathbf{0} \wedge \forall y \in T - \{x\}.h(y) \neq \mathbf{0}] = \mathbf{Pr}_h[h(x) = \mathbf{0}] \cdot \mathbf{Pr}_h[\forall y \in T - \{x\}.h(y) \neq \mathbf{0} | h(x) = \mathbf{0}]$$

and we know that

$$\mathbf{Pr}_h[h(x) = \mathbf{0}] = \frac{1}{2^{k+2}}$$

The difficult part is to estimate the other probability. First, we write

$$\mathbf{Pr}_h[\forall y \in T - \{x\}.h(y) \neq \mathbf{0} | h(x) = \mathbf{0}] = 1 - \mathbf{Pr}_h[\exists y \in T - \{x\}.h(y) = \mathbf{0} | h(x) = \mathbf{0}]$$

And then observe that

$$\begin{aligned}
& \mathbf{Pr}_h[\exists y \in T - \{x\}.h(y) = \mathbf{0} | h(x) = \mathbf{0}] \\
\leq & \sum_{y \in |T| - \{x\}} \mathbf{Pr}_h[h(y) = \mathbf{0} | h(x) = \mathbf{0}] \\
= & \sum_{y \in |T| - \{x\}} \mathbf{Pr}_h[h(y) = \mathbf{0}] \\
= & \frac{|T| - 1}{2^{k+2}} \\
\leq & \frac{1}{2}
\end{aligned}$$

Notice how we used the fact that the value of $h(y)$ is independent of the value of $h(x)$ when $x \neq y$.

Putting everything together, we have

$$\mathbf{Pr}_h[\forall y \in T - \{x\}.h(y) \neq \mathbf{0} | h(x) = \mathbf{0}] \geq \frac{1}{2}$$

and so

$$\mathbf{Pr}_h[h(x) = \mathbf{0} \wedge \forall y \in T - \{x\}.h(y) \neq \mathbf{0}] \geq \frac{1}{2^{k+3}}$$

To conclude the argument, we observe that the probability that there is a unique element of $T$ mapped into $\mathbf{0}$ is given by the sum over $x \in T$ of the probability that $x$ is the unique element mapped into $\mathbf{0}$ (all this events are disjoint, so the probability of their union is the sum of the probabilities). The probability of a unique element mapped into $\mathbf{0}$ is then at least $|T|/2^{k+3} > 1/8$. $\Box$

**Lemma 34** *There is a probabilistic polynomial time algorithm that on input a CNF formula $\phi$ and an integer $k$ outputs a formula $\psi$ such that*

- *If $\phi$ is unsatisfiable then $\psi$ is unsatisfiable.*
- *If $\phi$ has at least $2^k$ and less than $2^{k+1}$ satifying assignments, then there is a probability at least $1/8$ then the formula $\psi$ has exactly one satisfying assignment.*

PROOF: Say that $\phi$ is a formula over $n$ variables. The algorithm picks at random vectors $a_1, \ldots, a_{k+2} \in \{0,1\}^n$ and bits $b_1, \ldots, b_{k+2}$ and produces a formula $\psi$ that is equivalent to the expression $\phi(x) \wedge (a_1 \cdot x + b_1 = 0) \wedge \ldots \wedge (a_{k+2} \cdot x + b_{k+2} = 0)$. Indeed, there is no compact CNF expression to compute $a \cdot x$ if $a$ has a lot of ones, but we can proceed as follows: for each $i$ we add auxiliary variables $y_1^i, \ldots, y_n^i$ and then write a CNF condition equivalent to $(y_1^i = x_1 \wedge a_i[1]) \wedge \cdots \wedge (y_n^i = y_{n-1}^i \oplus (x_n \wedge a_i[n] \oplus b_i)))$. Then $\psi$ is the AND of the clauses in $\phi$ plus all the above expressions for $i = 1, 2, \ldots, k+2$.

By construction, the number of satisfying assignments of $\psi$ is equal to the number of satisfying assignments $x$ of $\phi$ such that $h_{a_1,\ldots,a_{k+2},b_1,\ldots,b_{k+2}}(x) = \mathbf{0}$. If $\phi$ is unsatisfiable, then, for every possible choice of the $a_i$, $\psi$ is also unsatisfiable.

If $\phi$ has between $2^k$ and $2^{k+1}$ assignments, then Lemma 33 implies that with probability at least $1/8$ there is exactly one satisfying assignment for $\psi$. $\Box$

**Theorem 35 (Valiant-Vazirani)** *Suppose there is a polynomial time algorithm that on input a CNF formula having exactly one satisfying assignment finds that assignment. (We make no assumption on the behaviour of the algorithm on other inputs.) Then* $\mathbf{NP} = \mathbf{RP}$.

PROOF: It is enough to show that, under the assumption of the Theorem, 3SAT has an **RP** algorithm.

On input a formula $\phi$, we construct formulae $\psi_0, \ldots, \psi_n$ by using the algorithm of Lemma 34 with parameters $k = 0, \ldots, n$. We submit all formulae $\psi_0, \ldots, \psi_n$ to the algorithm in the assumption of the Theorem, and accept if the algorithm can find a satisfying assignment for at least one of the formulae. If $\phi$ is unsatisfiable, then all the formulae are always unsatisfiable, and so the algorithm has a probability zero of accepting. If $\phi$ is satisfiable, then for some $k$ it has between $2^k$ and $2^{k+1}$ satisfying assignments, and there is a probability at least $1/8$ that $\psi_k$ has exactly one satisfying assignment and that the algorithm accepts. If we repeat the above procedure $t$ times, and accept if at least one iteration accepts, then if $\phi$ is unsatisfiable we still have probability zero of accepting, otherwise we have probability at least $1 - (7/8)^t$ of accepting, which is more than $1/2$ already for $t = 6$. $\Box$

## 7.2 References

The Valiant-Vazirani result is from [VV86].

## Exercises

1. Prove that $H_{AFF}$ is indeed a family of a pair-wise independent hash functions.

2. Change the assumption of Theorem 35 to having a *probabilistic* polynomial time algorithm that on input a formula with exactly one satisfying assignment finds that assignment with probability at least 1/2. Prove that it still follows that $\mathbf{NP} = \mathbf{RP}$.

# Lecture 8

# Counting Problems

## 8.1 Counting Classes

**Definition 11** *$R$ is an* **NP**-relation*, if there is a polynomial time algorithm $A$ such that $(x, y) \in R \Leftrightarrow A(x, y) = 1$ and there is a polynomial $p$ such that $(x, y) \in R \Rightarrow |y| \leq p(|x|)$.*

$\#R$ is the problem that, given $x$, asks how many $y$ satisfy $(x, y) \in R$.

**Definition 12** *$\#\mathbf{P}$is the class of all problems of the form $\#R$, where $R$ is an* **NP***-relation.*

Observe that an NP-relation $R$ naturally defines an NP language $L_R$, where $L_R = \{x : x \in R(x, y)\}$, and every NP language can be defined in this way. Therefore problems in $\#\mathbf{P}$ can always be seen as the problem of counting the number of witnesses for a given instance of an NP problem.

Unlike for decision problems there is no canonical way to define reductions for counting classes. There are two common definitions.

**Definition 13** *We say there is a* parsimonious reduction *from $\#A$ to $\#B$ (written $\#A \leq_{par} \#B$) if there is a polynomial time transformation $f$ such that for all $x$, $|\{y, (x, y) \in A\}| = |\{z : (f(x), z) \in B\}|$.*

Often this definition is a little too restrictive and we use the following definition instead.

**Definition 14** *$\#A \leq \#B$ if there is a polynomial time algorithm for $\#A$ given an oracle that solves $\#B$.*

#CIRCUITSAT is the problem where given a circuit, we want to count the number of inputs that make the circuit output 1.

**Theorem 36** *#CIRCUITSAT is $\#\mathbf{P}$-complete under parsimonious reductions.*

Proof: Let $\#R$ be in $\#\mathbf{P}$and $A$ and $p$ be as in the definition. Given $x$ we want to construct a circuit $C$ such that $|\{z : C(z)\}| = |\{y : |y| \leq p(|x|), A(x, y) = 1\}|$. We then construct $\hat{C}_n$ that on input $x, y$ simulates $A(x, y)$. From earlier arguments we know that this can be done with a circuit with size about the square of the running time of $A$. Thus $\hat{C}_n$ will have size polynomial in the running time of $A$ and so polynomial in $x$. Then let $C(y) = \hat{C}(x, y)$. $\Box$

**Theorem 37** *#3SATis #**P**-complete.*

Proof: We show that there is a parsimonious reduction from #CIRCUITSAT to #3-SAT. That is, given a circuit $C$ we construct a Boolean formula $\phi$ such that the number of satisfying assignments for $\phi$ is equal to the number of inputs for which $C$ outputs 1. Suppose $C$ has inputs $x_1, \ldots, x_n$ and gates $1, \ldots, m$ and $\phi$ has inputs $x_1, \ldots, x_n, g_1, \ldots, g_m$, where the $g_i$ represent the output of gate $i$. Now each gate has two input variables and one output variable. Thus a gate can be complete described by mimicking the output for each of the 4 possible inputs. Thus each gate can be simulated using at most 4 clauses. In this way we have reduced $C$ to a formula $\phi$ with $n+m$ variables and $4m$ clauses. So there is a parsimonious reduction from #CIRCUITSAT to #3SAT. $\square$

Notice that if a counting problem $\#R$ is #**P**-complete under parsimonious reductions, then the associated language $L_R$ is **NP**-complete, because $\#3SAT \leq_{par} \#R$ implies $3SAT \leq L_R$. On the other hand, with the less restrictive definition of reducibility, even some counting problems whose decision version is in **P** are #**P**-complete. For example, the problem of counting the number of satisfying assignments for a given 2CNF formula and the problem of counting the number of perfect matchings in a given bipartite graphs are both #**P**-complete.

## 8.2 Complexity of counting problems

We will prove the following theorem:

**Theorem 38** *For every counting problem $\#A$ in #**P**, there is an algorithm $C$ that on input $x$, compute with high probability a value $v$ such that*

$$(1-\epsilon)\#A(x) \leq v \leq (1+\epsilon)\#A(x)$$

*in time polynomial in $|x|$ and in $\frac{1}{\epsilon}$, using an oracle for **NP**.*

The theorem says that #**P** can be approximate in $\mathbf{BPP}^{\mathbf{NP}}$. We have a remark here that approximating #3SAT is **NP**-hard. Therefore, to compute the value we need at least the power of **NP**, and this theorem states that the power of **NP** and randomization is sufficient.

Another remark concerns the following result.

**Theorem 39 (Toda)** *For every $k$, $\Sigma_k \subseteq \mathbf{P}^{\#\mathbf{P}}$.*

This implies that #3SAT is $\Sigma_k$-hard for every $k$, i.e., #3SAT lies outside **PH**, unless the hierarchy collapses. Recall that **BPP** lies inside $\Sigma_2$, and hence approximating #3SAT can be done in $\Sigma_3$. Therefore, approximating #3SAT cannot be equivalent to computing #3SAT exactly, unless the polynomial hierarchy collapses.

We first make some observations so that we can reduce the proof to an easier one.

- It is enough to prove the theorem for #3SAT.

  If we have an approximation algorithm for #3SAT, we can extend it to any #A in #**P** using the parsimonious reduction from #A to #3SAT.

- It is enough to give a polynomial time $O(1)$-approximation for #3SAT.

  Suppose we have an algorithm $C$ and a constant $c$ such that

  $$\frac{1}{c}\#3\text{SAT}(\varphi) \le C(\varphi) \le c\#3\text{SAT}(\varphi).$$

  Given $\varphi$, we can construct $\varphi^k = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$. If $\varphi$ has $t$ satisfying assignments, $\varphi^k$ has $t^k$ satisfying assignments. Then, giving $\varphi^k$ to the algorithm we get

  $$\frac{1}{c}t^k \le C(\varphi^k) \le ct^k$$
  $$\left(\frac{1}{c}\right)^{1/k} t \le C(\varphi^k)^{1/k} \le c^{1/k}t.$$

  If $c$ is a constant and $k = O(\frac{1}{\epsilon})$, $c^{1/k} = 1 + \epsilon$.

- For a formula $\varphi$ that has $O(1)$ satisfying assignments, $\#3\text{SAT}(\varphi)$ can be found in $\mathbf{P}^{\mathbf{NP}}$.

  This can be done by iteratively asking the oracle the questions of the form: "Are there $k$ assignments satisfying this formula?" Notice that these are **NP** questions, because the algorithm can guess these $k$ assignments and check them.

## 8.3 An approximate comparison procedure

Consider the following approximate comparison procedure `a-comp` defined as:

$$\texttt{a-comp}(\varphi, k) = \begin{cases} \text{YES} & \text{if } \#3\text{SAT}(\varphi) \ge 2^{k+1} \\ \text{NO} & \text{if } \#3\text{SAT}(\varphi) < 2^k \\ \text{whatever} & \text{otherwise} \end{cases}$$

Given `a-comp`, we can construct an algorithm that 2-approximates #3SAT as follows:

```
Input: $\varphi$

compute:
  a-comp($\varphi$, 1)
  a-comp($\varphi$, 2)
  a-comp($\varphi$, 3)
  ⋮
  a-comp($\varphi$, $n$)

if a-comp outputs NO from the first time then
  // The value is either 0 or 1.
  // The answer can be checked by one more query to the NP oracle.
  Query to the oracle and output an exact value.
```

```
else
    Suppose that it outputs YES for $t = 1, \ldots, i-1$ and NO for $t = i$
    Output $2^i$
```

We need to show that this algorithm approximates #3SAT within a factor of 2. If `a-comp` answers NO from the first time, the algorithm outputs the right answer because it checks for the answer explicitly. Now suppose `a-comp` says YES for all $t = 1, 2, \ldots, i-1$ and says NO for $t = i$. Since `a-comp`$(\varphi, i-1)$ outputs YES, $\#3\text{SAT}(\varphi) \geq 2^{i-1}$, and also since `a-comp`$(\varphi, 2^i)$ outputs NO, $\#3\text{SAT}(\varphi) < 2^{i+1}$. The algorithm outputs $a = 2^i$. Hence,

$$\frac{1}{2}a \geq \#3\text{SAT}(\varphi) < 2 \cdot a$$

and the algorithm outputs the correct answer with in a factor of 2.

Thus, to establish the theorem, it is enough to give a $\mathbf{BPP}^{\mathbf{NP}}$ implementation of the `a-comp`.

## 8.4 Constructing `a-comp`

The procedure and its analysis is similar to the Valiant-Vazirani reduction: for a given formula $\phi$ we pick a hash function $h$ from a pairwise independent family, and look at the number of assignments $x$ that satisfy $h$ and such that $h(x) = \mathbf{0}$.

In the Valiant-Vazirani reduction, we proved that if $S$ is a set of size approximately equal to the size of the range of $h()$, then, with constant probability, exactly one element of $S$ is mapped by $h()$ into $\mathbf{0}$. Now we use a different result, a simplified version of the Leftover Hash Lemma proved by Impagliazzo, Levin, and Luby in 1989, that says that if $S$ is sufficiently larger than the range of $h()$ then the number of elements of $S$ mapped into $\mathbf{0}$ is concentrated around its expectation.

**Lemma 40 (The Leftover Hash Lemma)** *Let $H$ be a family of pairwise independent hash functions $h : \{0,1\}^n \to \{0,1\}^m$. Let $S \subset 0, 1^n$, $|S| \geq \frac{4 \cdot 2^m}{\epsilon^2}$. Then,*

$$\mathbf{Pr}_{h \in H}\left[\left| |\{a \in S : h(a) = 0\}| - \frac{|S|}{2^m} \right| \geq \epsilon \frac{|S|}{2^m}\right] \leq \frac{1}{4}.$$

From this, `a-comp` can be constructed as follows.

```
input: $\varphi, k$

if $k \leq 5$ then check exactly whether $\#3\text{SAT}(\varphi) \geq 2^k$.
if $k \geq 6$,
    pick $h$ from a set of pairwise independent hash functions $h : \{0,1\}^n \to \{0,1\}^m$,
        where $m = k - 5$
    answer YES iff there are more then 48 assignments $a$ to $\varphi$ such that
        $a$ satisfies $\varphi$ and $h(a) = 0$.
```

Notice that the test at the last step can be done with one access to an oracle to **NP**. We will show that the algorithm is in $\mathbf{BPP}^{\mathbf{NP}}$. Let $S \subseteq \{0,1\}^n$ be the set of satisfying assignments for $\varphi$. There are 2 cases.

- If $|S| \geq 2^{k+1}$, by the Leftover Hash Lemma we have:

$$\mathbf{Pr}_{h\in H}\left[\left||\{a \in S : h(a) = 0\}| - \frac{|S|}{2^m}\right| \leq \frac{1}{4}\cdot\frac{|S|}{2^m}\right] \leq \frac{3}{4},$$

(set $\epsilon = \frac{1}{2}$, and $|S| \geq \frac{4\cdot 2^m}{\epsilon^2} = 64 \cdot 2^m$, because $|S| \geq 2^{k+1} = 2^{m+6}$)

$$\mathbf{Pr}_{h\in H}\left[|\{a \in S : h(a) = 0\}| \leq \frac{3}{4}\cdot\frac{|S|}{2^m}\right] \geq \frac{3}{4},$$
$$\mathbf{Pr}_{h\in H}\left[|\{a \in S : h(a) = 0\}| \geq 48\right] \geq \frac{3}{4},$$

which is the success probability of the algorithm.

- If $|S| < 2^k$:

Let $S'$ be a superset of $S$ of size $2^k$. We have

$$\begin{aligned}\mathbf{Pr}_{h\in H}[\text{answer YES}] &= \mathbf{Pr}_{h\in H}[|\{a \in S : h(s) = 0\}| \geq 48] \\ &\leq \mathbf{Pr}_{h\in H}[\left|\left\{a \in S' : h(s) = 0\right\}\right| \geq 48] \\ &\leq \mathbf{Pr}_{h\in H}\left[\left|\left|\left\{a \in S' : h(s) = 0\right\}\right| - \tfrac{|S'|}{2^m}\right| \geq \tfrac{|S'|}{2\cdot 2^m}\right] \\ &\leq \frac{1}{4}\end{aligned}$$

(by L.H.L. with $\epsilon = 1/2, |S'| = 32 \cdot 2^m$.)

Therefore, the algorithm will give the correct answer with probability at least $3/4$, which can then be amplified to, say, $1 - 1/4n$ (so that all $n$ invocations of `a-comp` are likely to be correct).

## 8.5 The proof of the Leftover Hash Lemma

We finish the lecture by proving the Leftover Hash Lemma.

Proof: We will use Chebyshev's Inequality to bound the failure probability. Let $S = \{a_1, \dots, a_k\}$, and pick a random $h \in H$. We define random variables $X_1, \dots, X_k$ as

$$X_i = \begin{cases} 1 & \text{if } h(a_i) = 0 \\ 0 & \text{otherwise.}\end{cases}$$

Clearly, $|\{a \in S : h(a) = 0\}| = \sum_i X_i$.

We now calculate the expectations. For each $i$, $\mathbf{Pr}[X_i = 1] = \frac{1}{2^m}$ and $\mathbf{E}[X_i] = \frac{1}{2^m}$. Hence,

$$\mathbf{E}[\sum_i X_i] = \frac{|S|}{2^m}.$$

Also we calculate the variance

$$\begin{aligned}\mathbf{Var}[X_i] &= \mathbf{E}[X_i^2] - \mathbf{E}[X_i]^2 \\ &\leq \mathbf{E}[X_i^2] \\ &= \mathbf{E}[X_i] = \frac{1}{2^m}.\end{aligned}$$

Because $X_1, \ldots, X_k$ are pairwise independent,

$$\mathbf{Var}\left[\sum_i X_i\right] = \sum_i \mathbf{Var}[X_i] \leq \frac{|S|}{2^m}.$$

Using Chebyshev's Inequality, we get

$$\begin{aligned}\mathbf{Pr}\left[\left||\{a \in S : h(a) = 0\}| - \frac{|S|}{2^m}\right| \geq \epsilon\frac{|S|}{2^m}\right] &= \mathbf{Pr}\left[\left|\sum_i X_i - \mathbf{E}[\sum_i X_i]\right| \geq \epsilon\mathbf{E}[\sum_i X_i]\right] \\ &\leq \frac{\mathbf{Var}[\sum_i X_i]}{\epsilon^2\mathbf{E}[\sum_i X_i]^2} \leq \frac{\frac{|S|}{2^m}}{\epsilon^2\frac{|S|^2}{(2^m)^2}} \\ &= \frac{2^m}{\epsilon^2\,|S|} \leq \frac{1}{4}.\end{aligned}$$

□

## 8.6 Approximate Sampling

So far we have considered the following question: for an **NP**-relation $R$, given an input $x$, what is the size of the set $R_x = \{y : (x, y) \in R\}$? A related question is to be able to sample from the uniform distribution over $R_x$.

Whenever the relation $R$ is "downward self reducible" (a technical condition that we won't define formally), it is possible to prove that there is a probabilistic algorithm running in time polynomial in $|x|$ and $1/\epsilon$ to approximate within $1 + \epsilon$ the value $|R_x|$ if and only if there is a probabilistic algorithm running in time polynomial in $|x|$ and $1/\epsilon$ that samples a distribution $\epsilon$-close to the uniform distribution over $R_x$.

We show how the above result applies to 3SAT (the general result uses the same proof idea). For a formula $\phi$, a variable $x$ and a bit $b$, let us define by $\phi_{x\leftarrow b}$ the formula obtained by substituting the value $b$ in place of $x$.[1]

[1]Specifically, $\phi_{x\leftarrow 1}$ is obtained by removing each occurrence of $\neg x$ from the clauses where it occurs, and removing all the clauses that contain an occurrence of $x$; the formula $\phi_{x\leftarrow 0}$ is similarly obtained.

If $\phi$ is defined over variables $x_1, \ldots, x_n$, it is easy to see that

$$\#\phi = \#\phi_{x\leftarrow 0} + \#\phi_{x\leftarrow 1}$$

Also, if $S$ is the uniform distribution over satisfying assignments for $\phi$, we note that

$$\mathbf{Pr}_{(x_1,\ldots,x_n)\leftarrow S}[x_1 = b] = \frac{\#\phi_{x\leftarrow b}}{\#\phi}$$

Suppose then that we have an efficient sampling algorithm that given $\phi$ and $\epsilon$ generates a distribution $\epsilon$-close to uniform over the satisfying assignments of $\phi$.

Let us then ran the sampling algorithm with approximation parameter $\epsilon/2n$ and use it to sample about $\tilde{O}(n^2/\epsilon^2)$ assignments. By computing the fraction of such assignments having $x_1 = 0$ and $x_1 = 1$, we get approximate values $p_0, p_1$, such that $|p_b - \mathbf{Pr}_{(x_1,\ldots,x_n)\leftarrow S}[x_1 = b]| \leq \epsilon/n$. Let $b$ be such that $p_b \geq 1/2$, then $\#\phi_{x\leftarrow b}/p_b$ is a good approximation, to within a multiplicative factor $(1 + 2\epsilon/n)$ to $\#\phi$, and we can recurse to compute $\#\phi_{x\leftarrow b}$ to within a $(1 + 2\epsilon/n)^{n-1}$ factor.

Conversely, suppose we have an approximate counting procedure. Then we can approximately compute $p_b = \frac{\#\phi_{x\leftarrow b}}{\#\phi}$, generate a value $b$ for $x_1$ with probability approximately $p_b$, and then recurse to generate a random assignment for $\#\phi_{x\leftarrow b}$.

The same equivalence holds, clearly, for 2SAT and, among other problems, for the problem of counting the number of perfect matchings in a bipartite graph. It is known that it is **NP**-hard to perform approximate counting for 2SAT and this result, with the above reduction, implies that approximate sampling is also hard for 2SAT. The problem of approximately sampling a perfect matching has a probabilistic polynomial solution, and the reduction implies that approximately counting the number of perfect matchings in a graph can also be done in probabilistic polynomial time.

The reduction and the results from last section also imply that 3SAT (and any other NP relation) has an approximate sampling algorithm that runs in probabilistic polynomial time with an **NP** oracle. With a careful use of the techniques from last week it is indeed possible to get an *exact* sampling algorithm for 3SAT (and any other NP relation) running in probabilistic polynomial time with an NP oracle. This is essentially best possible, because the approximate sampling requires randomness by its very definition, and generating satisfying assignments for a 3SAT formula requires at least an NP oracle.

## 8.7 References

The class #**P** was defined by Valiant [Val79]. An algorithm for approximate counting within the polynomial hierarchy was developed by Stockmeyer [Sto83]. The algorithm presented in these notes is taken from lecture notes by Oded Goldreich. The left-over hash lemma is from [HILL99]. The problem of approximate sampling and its relation to approximate counting is studied in [JVV86].

# Lecture 9

# Average-Case Complexity of the Permanent

## 9.1 The Permanent Problem

The *permanent* is a number assigned to a matrix $M \in R$, for a ring $R$, according to the formula

$$\text{perm}(M) = \sum_{\pi \in S_n} \prod_{i=1}^{n} M_{i,\pi(i)}$$

where $S_n$ is the group of permutations on $n$ items. The definition of the permanent closely resembles that of the determinant; the only difference is that the determinant has a coefficient $(-1)^{\text{sgn}(\pi)}$ in front of the product. However, this resemblance is deceptive: the determinant can be calculated in polynomial time, whereas Permanent is #**P**-complete.

**Claim 41** *Let $R = \mathbb{Z}$ and let $M$ be a 0/1 matrix, which we think of as representing a bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$ as follows: number both edge sets $U$ and $V$, separately, by $1, 2, \ldots n$, and let $M_{i,j}$ be 1 if edge $(i, j) \in E$, and 0 otherwise. Then* $\text{perm}(M)$ *is the number of perfect matchings in $G$.*

Proof: Each $\pi \in S_n$ is a potential perfect matching, in that it can be thought of as a bijection between $U$ and $V$. This corresponds to an actual perfect matching iff all edges $(i, \pi(i))$ are in $E$, i.e. if $M_{i,\pi(i)} = 1$ for all $i$, which is equivalent to saying that $\prod_{i=1}^{n} M_{i,\pi(i)} = 1$. Thus we get a summand of 1 in the formula for $\text{perm}(M)$ for each perfect matching of $G$. $\Box$

Suppose we allow the entries of $M$ to be nonnegative integers. Then $M$ represents a bipartite graph where the edges have multiplicity given by the entries of $M$. It still makes sense to interpret $\text{perm}(M)$ as the number of permanent matchings, because the product of the multiplicities of the edges of a matching is the number of ways of realizing that matching.

Note that in $\mathbb{Z}_2$, $\text{perm}(M) = \det(M)$ and thus is polynomial time. But for sufficiently large moduli (or no modulus at all) it's #**P**-complete.

## 9.2 Worst-case to Average-case Equivalence

**Theorem 42** *Suppose for fixed $n$ and $\mathcal{F}$, $|\mathcal{F}| > n^{O(1)}$, we can compute* perm$(M)$ *on a fraction $1 - \frac{1}{2n}$ of the inputs $M$ in polynomial time. Then there is a (worst-case) polynomial time probabilistic algorithm for computing* perm$(M)$*.*

Rather than proving this theorem directly, we will just observe that perm$(M)$ is a degree $n$ polynomial in indeterminates $M_{11}, M_{12}, \ldots, M_{1n}, \ldots, M_{nn}$, and prove the following much more general result.

**Theorem 43** *Suppose we have oracle access to a function $f : \mathcal{F}^n \to \mathcal{F}$ and $f$ agreees with some degree $d$ polynomial $p : \mathcal{F}^n \to \mathcal{F}$ on a fraction $1 - \frac{1}{3d+3}$ of its inputs where $|\mathcal{F}| \geq d+2$. Then we can compute $f(x)$ with high probability for all $x \in \mathcal{F}^n$ in poly$(n, d)$ time, assuming field operations can be performed in poly$(n, d)$ time.*

Proof: Suppose that we input $x \in \mathcal{F}^n$ and pick $y \in \mathcal{F}^n$ uniformly at random. Consider the line $l(t) = x + ty$. Each point of $l$ is uniformly distributed except $x = l(0)$. Now,

$$\begin{aligned}\Pr[p(l(i)) = f(l(i)) \text{ for } i = 1, 2, \ldots, d+1] &= 1 - Pr[\exists i \in 1, \ldots, d+1 : p(l(i)) \neq f(l(i))] \\ &\geq 1 - \frac{d+1}{3(d+1)} = \frac{2}{3}\end{aligned}$$

Now $q(t) = p(l(t))$ is a degree $d$ polynomial in $t$, and we know $d+1$ of its values, so we can interpolate the others. Thus we have the following algorithm:

```
input: $x$,
  pick random $y \in \mathcal{F}^n$
  find univariate degree-$d$ polynomial $q()$ such that
    $q(i) = p(x + iy)$ for $i = 1, \ldots, d+1$
  **return** $(q(0))$
```

This runs in poly$(n, d)$ time, and with probability $\geq \frac{2}{3}$, it outputs $p(x)$. □

We can strengthen our result to hold even if $p(x)$ only agrees with $f(x)$ on $\frac{3}{4} + \epsilon$ of its inputs. This stronger proof depends on the following theorem, due to Berlekamp and Welch.

**Theorem 44** *Let $q : \mathcal{F} \to \mathcal{F}$ be a polynomial of degree $d$. Say we are given $m$ pairs $(a_1, b_1), \ldots, (a_m, b_m)$ such that for fewer than $\frac{m-d}{2}$ pairs $(a_i, b_i)$, $b_i \neq q(a_i)$. Then $q$ can be reconstructed with poly$(n, d)$ field operations.*

Now, as promised, we apply Berlekamp-Welch to strengthen our main theorem:

**Theorem 45** *Suppose we have oracle access to a function $f : \mathcal{F}^n \to \mathcal{F}$ and $f$ agreees with some degree $d$ polynomial $p : \mathcal{F}^n \to \mathcal{F}$ on a fraction $\frac{3}{4} + \epsilon$ of its inputs where $|\mathcal{F}| \geq 1 + d/\epsilon$. Then we can compute $p(x)$ with high probability for all $x \in \mathcal{F}^n$ in poly$(n, d)$ time, assuming field operations can be performed in poly$(n, d)$ time.*

PROOF: Consider the following algorithm:

```
input: $x$,
   pick random $y \in \mathcal{F}^n$
   find degree-$d$ univariate polynomial $q()$ such that
      $|\{t \in \mathcal{F} - \{0\} : q(t) \neq f(x+ty)\}| < (|\mathcal{F}| - 1 - d)/2$
   halt and fail if no such polynomial exists
   return $(q(0))$
```

If $q(t) = p(l(t))$ and $f(l(t))$ disagree on $< \frac{|\mathcal{F}|-d-1}{2}$ values of $t$, Berlekamp-Welch will find $q(t) \equiv p(l(t))$ and the algorithm succeeds (and in polynomial time). So, assuming only $\frac{1}{4} - \epsilon$ of our points disagree, we need to show that less than half of our runs fail.

$$\begin{aligned}
\mathbf{E}\left[\#\text{ values of } t \in \mathcal{F} - \{0\} \text{ with } p(l(t)) \neq f(l(t))\right] &= \sum_{t \neq 0} \mathbf{E}\left[\begin{cases} 0 & \text{if } p(l(t)) = f(l(t)) \\ 1 & \textit{otherwise} \end{cases}\right] \\
&\leq (|\mathcal{F}| - 1)\left(\frac{1}{4} - \epsilon\right)
\end{aligned}$$

To clean up the notation let us use $m = |\mathcal{F}| - 1$. The previous expression and Markov's inequality tell us that

$$\Pr\left[|\{t \in \mathcal{F} - \{0\} : p(l(t)) \neq f(l(t))\}| > \frac{m-d}{2}\right] \leq \frac{m(\frac{1}{4} - \epsilon)}{\frac{m-d}{2}} \leq \frac{1}{2} - \frac{\epsilon}{2} \quad \text{if } m \geq \frac{d}{\epsilon}$$

Thus, our algorithm succeeds more than half the time, so it can be repeated to yield any degree of accuracy we wish. □

## 9.3 References

The simple algorithm to reconstruct a polynomial from a faulty oracle is due to Beaver and Feigenbaum [BF90], and the more sophisticated version that uses error-correction is due to [GLR$^+$91]. The use of polynomial reconstruction algorithms in the average-case complexity of the permanent due to Lipton [Lip90]. The Berlekamp-Welch algorithm [WB86] is a polynomial time algorithm to reconstruct a univariate polynomial from faulty data.

# Lecture 10

# Average-case Complexity of Problems in PSPACE and EXP

## 10.1 Average-case Complexity in PSPACE and EXP for 7/8 Fraction of Inputs

**Lemma 46 (Multivariate Interpolation)** *Let $\mathcal{F}$ be a field, $H \subseteq \mathcal{F}$ a subset and $f : H^m \to \mathcal{F}$ an arbitrary function. Then there is a $m$-variate polynomial $p$ of degree at most $m|H|$ such that $p(x) = f(x)$ for every $x \in H^m$. Furthermore, the value $p(x)$ can be computed using time polynomial in $|\mathcal{F}|^m$ and space logarithmic in $|\mathcal{F}|^m$ given oracle access to $f$.*

Let $L$ be an **EXP**-complete problem (the same construction and reasoning can be repeated with a **PSPACE**-complete problem), and for every $n$ let $L_n : \{0,1\}^n \to \{0,1\}$ the characteristic function of $L$ on inputs of length $n$. We will construct another **EXP**-complete problem $L'$ such that for every $n$ there is a $n' = O(n)$ such that the truth-table of $L'_{n'}$ is the encoding of the truth-table of $L_n$ using a concatenation of Reed-Muller and Hadamard codes. Then we will be able to show that $L'$ is as hard in the worst case as it is hard on average.

Let us fix an input length $n$.

Let $k = 3\lceil \log n \rceil$, and let $\mathcal{F}$ be a field of size $2^k = O(n^3)$. We will identify elements of $\mathcal{F}$ with vectors of $\{0,1\}^k$ and, in particular, the zero element of $\mathcal{F}$ is mapped into $(0, \ldots, 0)$ and the one element of $\mathcal{F}$ is mapped into $(1, 0, \ldots, 0)$. Let $H \subseteq \mathcal{F}$ be a subset of size $2^{k/3}$. Let $m = 3n/k$, then $H^m$ contains at least $2^n$ elements and there is some efficiently computable injective function $h : \{0,1\}^n \to H^m$. From Lemma 46 it follows that there is a multivariate polynomial $p$ of total degree at most $O(n^2/\log n)$ such that $p(h(x)) = L_n(x)$ for all $x \in \{0,1\}^n$. Let $f : \mathcal{F}^m \times \{0,1\}^k \to \{0,1\}$ the function such that $f(x_1, \ldots, x_m, a) = p(x_1, \ldots, x_m) \cdot a$, where we abused notation a little by viewing $p(x_1, \ldots, x_m)$ as an element of $\{0,1\}^k$. With similar abuse, we can view $f$ as a function $f : \{0,1\}^{3n+k} \to \{0,1\}$; we define the language $L'$ such that $f$ is the characteristic function of $L'$ on inputs of length $3n + 3\lceil \log n \rceil$. Clearly, $L'$ is **EXP**-complete, because for every $x \in B^n$ we have $x \in L$ if and only if $p(h(x)) = 1$, which holds if and only if $(h(x), \mathbf{1}) \in L'$, where $\mathbf{1} = (1, 1, \ldots, 1)$.

**Theorem 47** *Let $p : \mathcal{F}^m \to \mathcal{F}$ be a degree $d$ polynomial, let us identify elements of $\mathcal{F}$ with elements of $\{0,1\}^k$, and let $f : \{0,1\}^{km+k} \to \{0,1\}$ be such that $f(x,a) = p(x) \cdot a$. Suppose we are given oracle access to $g : \{0,1\}^{km+k} \to \{0,1\}$ such that $f$ and $g$ agree on at least a fraction $7/8 + \epsilon$ of the inputs, where $|\mathcal{F}| \geq 1 + \max\{4/\epsilon^2, d/4\epsilon\}$. Then we can compute $p(x)$ for every $x$ in time polynomial in $m$, $|F|$ and $2^k$.*

Proof: For every $x$, define $c(x)$ to be the value that maximizes $\mathbf{Pr}_a[c(x) \cdot a = f(x,a)]$, and break ties arbitrarily if the maximum is not unique. Notice that $c(x)$ can be computed in time polynomial in $2^k$. For random $(x,a)$, $f(x,a) \neq g(x,a)$ with probability at most $1/8 - \epsilon$, so it follows that

$$\mathbf{Pr}_x[\mathbf{Pr}_a[f(x,a) \neq g(x,a)] \geq 1/4] \leq 1/2 - 4\epsilon$$

So for at least a $1/2 + 4\epsilon$ fraction of the $x$s, we have $g(x,a) = f(x,a) = p(x) \cdot a$ for more than $3/4$ of the $a$s. For each such $x$, then, $p(x) = c(x)$. The function $c()$ has then agreement $1/2 + \epsilon$ with $p()$, which means that, by previous results, we can compute $p()$ everywhere using $c()$. $\Box$

Note that the above theorem is essentially giving a poly-logarithmic time decoding procedure for the concatenation of the Reed-Muller code with the Hadamard code. The procedure first decodes the Hadamard code locally, and then simulates the procedure for decoding the Reed-Muller code using the results of the Hadamard decoding. The Reed-Muller decoding needs the number of errors to be less than 1/2, and the Hadamard decoding needs the number of errors to be less than 1/4, so the overall procedure needs the number of errors to be less than 1/8.

In order to state the application of the previous theorem to the average-case hardness of problems in **PSPACE** and **EXP**, let us define $avg\mathbf{BPTIME}_{p(n)}(t(n))$ to be the class of decision problems $L$ for which there is a probabilistic algorithm $A$ running in time $t(n)$ such that for every input length $n$

$$\mathbf{Pr}_{x \in \{0,1\}^n}[\mathbf{Pr}_r[A(x,r) = L_n(x)] \geq 3/4] \geq p(n)$$

In other words, $avg\mathbf{BPP}_{p(n)}(t(n))$ contains all problems that can be solved on at least a $p(n)$ fraction of the inputs by BPP-type algorithms in time $t(n)$. Notice that $avg\mathbf{BPP}_1(t(n)) = \mathbf{BPP}(t(n))$.

**Corollary 48** *Suppose that $\mathbf{EXP} \subseteq avg\mathbf{BPP}_{7/8+1/n}\left(n^{O(1)}\right)$, then $\mathbf{EXP} = \mathbf{BPP}$. Suppose that $\mathbf{PSPACE} \subseteq avg\mathbf{BPP}_{7/8+1/n}\left(n^{O(1)}\right)$, then $\mathbf{PSPACE} = \mathbf{BPP}$.*

## 10.2 Coding-Theoretic Perspective

The construction of the previous section can be abstracted as follows: starting from a language $L$ and an input length $n$, we encode the characteristic function $L_n : \{0,1\}^n \to \{0,1\}$ of $L$ on inputs of length $n$ as a function $f_L : \{0,1\}^{O(n)} \to \{0,1\}$, and then we argue that given a procedure that computes $g$ on a $1 - \delta$ fraction of inputs, with $\delta < 1/8$ we can reconstruct $L_n$ everywhere.

The construction can be carried over for every $L_n$, and we can see that, in order for the theorem to hold, it must be the case that if we have to different functions $L_n, L'_n : \{0,1\}^n \to \{0,1\}$ then the functions $f_L$ and $f_{L'}$ must differ in at least a $2\delta$ fraction of inputs, otherwise it would not be possible to reconstruct $f_L$ given only a function $g$ that has agreement $1-\delta$ with $f_L$. In fact, the encoding of $L_n$ into $f_L$ is a concatenation of a Reed-Muller code with a Hadamard code, and its minimum relative distance is almost $1/2$.

Now suppose we are trying to prove that **EXP** contains languages that hard to compute even on a $1/2+\epsilon$ fraction of inputs. By the above argument, we need an encoding that will map two different languages into two boolean functions that differ in at least $1-2\epsilon$ fraction of points. Unfortunately, this is impossible: there is no error-correcting code with relative minimum distance bigger than $1/2$ (except for trivial codes containing a small constant number of codewords), which means that the unique reconstruction problem is possible only if there are less than a $1/4$ fraction of errors.[1]

In order to prove stronger hardness results, we need to stop looking for unique decoding procedures, and start looking for *list decoding* procedures instead. In the list decoding problem for a code $C : \{0,1\}^n \to \{0,1\}^m$, we are given a received string $z \in \{0,1\}^m$, and we believe that less than a $\delta$ fraction of errors occurred, and we want to compute the list of all possible messages $x \in \{0,1\}^n$ such that $C(x)$ and $z$ are at distance less than $\delta$. If the list is small and it can be produced efficiently, it gives a lot of information about the original message.

We will state without proof a list-decoding result about the Hadamard code and one about the Reed-Muller code.

**Lemma 49** *Given $g : \{0,1\}^k \to \{0,1\}$ and $\epsilon > 0$, there are at most $1/4\epsilon^2$ strings $a \in \{0,1\}^k$ such that $\mathbf{Pr}_x[g(x) = a \cdot x] \geq 1/2 + \epsilon$.*

For the Hadamard code, the list can be find by brute force in time polynomial in the length of the encoding. Goldreich and Levin show that, given oracle access to $f$, the list can even be found in time polynomial in $k$ and $1/\epsilon$ (by a randomized algorithm, with high probability), although we will not need such a stronger result in our applications.

**Theorem 50 (Sudan)** *Let $\mathcal{F}$ be a field, $d$ a degree parameter, and $1 > \alpha > \sqrt{2d/m}$ be an accuracy parameter. Then there is a polynomial time algorithm that on input pairs of field elements $(a_1, b_1), \ldots, (a_m, b_m)$ finds a list of all univariate polynomials $q()$ of degree $\leq d$ such that $q(a_i) = b_i$ for at least $\alpha m$ pairs. Furthermore, the list contains at most $2/\alpha$ polynomials.*

In the next section we will see how to use Sudan's algorithm to achieve a form of list-decoding of Reed-Muller code even given a $1-\epsilon$ fraction of errors. Since we can do list-decoding of the Hadamard code with up to $1/2-\epsilon$ errors, it should be intuitive that the combination of the two procedures will give a decoding for the concatenation of the two codes that will work with $1/2 - epsilon'$ fraction of errors. Using non-uniformity, we will be able pick the right decoding from the list of possible decodings by hard-wiring the choice

[1]Note that there is still some margin of improvement, because our reconstruction procedure works only with less than a 1/8 fraction of errors.

in the circuit performing the reconstruction. This will show that if every problem in **EXP** can be solved by small circuits on $1/2 + \epsilon$ fraction of inputs, then all problems in **EXP** can be solved by small circuits on all inputs.

## 10.3 Average-case Complexity in PSPACE and EXP for 1/2 Fraction of Inputs

Let us recall a (weaker version of a) result from a past lecture. It was stated in terms of probabilistic procedures, but it holds for a stronger reason in terms of small circuits. (Because we have seen that probabilistic procedures can always be simulated by circuits.)

**Theorem 51** *Let $p : \mathcal{F}^m \to \mathcal{F}$ be a polynomial of degree $d$, $|\mathcal{F}| > 16d$, and let $f : \mathcal{F}^m \to \mathcal{F}$ be a function that agrees with $p$ on a $7/8$ fraction of inputs. Then there is an oracle circuit $C$ of size polynomial in $m$ and $|\mathcal{F}|$ such that $C^f(x) = p(x)$ for every $x$.*

Here is the multivariate list-decoding result. We state it as a non-uniform unique reconstruction procedure.

**Theorem 52** *Let $p : \mathcal{F}^m \to \mathcal{F}$ be a polynomial of degree $d$, $|\mathcal{F}| \geq 1 + 8d/\epsilon^2$, and let $f : \mathcal{F}^m \to \mathcal{F}$ be a function that agrees with $p$ on at least an $\epsilon$ fraction of inputs, with $|\mathcal{F}| \geq 1 + 8d/\epsilon^2$. Then there is an oracle circuit $C$ of size polynomial in $m$ and $|\mathcal{F}|$ such that $C^f(x) = p(x)$ for every $x$.*

Proof: We will just show the existence of a polynomial size circuit that computes $p()$ on a $7/8$ fraction of inputs, and then invoke Theorem 51.

Let us pick $x$ and $y$ at random in $\mathcal{F}^m$, we want to argue that knowing $p(y)$ will help us compute $p(x)$. The following procedure does that.

```
input: $x$, $y$, $p(y)$
  consider line $l(t) = tx + (1-t)y$ and find set $Q$ of
    all degree-$d$ univariate polynomials $q$ such that
    $|\{t \in \mathcal{F} : q(t) = f(tx + (1-t)y)\}| > \epsilon|\mathcal{F}|/2$
  if there is unique $q^* \in Q$ such that $q^*(0) = p(y)$ return $q^*(1)$
  otherwise fail
```

By our assumption on $\mathcal{F}$, $d$, and $\epsilon$, we have $\epsilon/2 > \sqrt{2d/|\mathcal{F}|}$, so the above algorithm can be implemented by a circuit $C$ of size polynomial in $m$ and $|\mathcal{F}|$. Now we want to argue that the algorithm is quite effective.

**Claim 53** $\mathbf{Pr}_{x,y}[C^f(x, y, p(y)) = p(x)] \geq 7/8$.

Before proving the claim, let us see that, once the claim is proved, we are done. By an averaging argument, there is a $y$ such that

$$\mathbf{Pr}_x[C^f(x, y, p(y)) = p(x)] \geq 7/8.$$

Then we just have to hard-wire $y$ and $p(y)$ into $C$, and we have a circuit that computes $p()$ on a $7/8$ fraction of inputs, and we can invoke Theorem 51 and we are done. Now we prove the claim.

PROOF: [Of Claim] First of all, note that when we pick $x$ and $y$ at random, the points on the line $l(t) = tx + (1-t)y$ are pairwise independent. Therefore, there is a probability at least $15/16$ then there are at least $|\mathcal{F}|\epsilon - 4\sqrt{|\mathcal{F}|}$ points on the line where $f$ agrees with $p$, and this number is at least $\epsilon/2$ by our assumptions on $\mathcal{F}$, $d$ and $\epsilon$.

So, except with probability $1/16$, the polynomial $q^*$ such that $q^*(t) = p(l(t))$ is part of the list $Q$ constructed by the algorithm. It remains to see that it is the unique one such that $q(0) = p(y)$. Notice that each other polynomial $q \in Q$ agrees with $q^*$ in at most $d$ points of the line, so, on a random point of the line, $q^*$ has a probability at least $1 - (|Q|-1)d/|\mathcal{F}|$ of having a value different from the one of all other polynomials in $Q$ on that random point. Well, $y$ was a random point, but it is not exactly a random point on $l()$: by definition, indeed, $y = l(0)$. Can we still argue that $y$ is as good as a random point of the line in distinguishing $q^*()$ from the other polynomials in $Q$?

To see that this is the case, let us pick the points $x, y$ in a different (but equivalent) way: let us first pick two random points $x', y' \in \mathcal{F}^m$, then consider the line $l'(t) = tx' + (1-t)y'$, and finally pick two random points $a, b \in \mathcal{F}$ on the line and call $x = l'(a)$ and $y = l'(b)$. Surely, $x$ and $y$ picked in this way are uniformly distributed. What about the line $l(t) = tx + (1-t)y$? It is the same line as $l'()$, just with a different parameterization: we have $l(t) = l'(t(a-b)+b)$. So for each polynomial $q \in Q$ such that $q(t)$ has agreement $\epsilon/2$ with $p(l(t))$ there is a corresponding polynomial $q'(t) = q(t(a-b)+b))$ such that $q'(t)$ has agreement $\epsilon/2$ with $p(l'(t))$ and we can call $Q'$ the list of such polynomials. Let $q'^*(t) = q^*(t(a-b)+b))$, then we have that $q^*$ is the unique polynomial in $Q$ such that $q(0) = p(y)$ if and only if $q'^*$ is the unique polynomial in $Q'$ such that $q'^*(b) = p(y)$. In the latter case, it is clear that we are looking at random point, and so the probability that $q^*$ is isolated is at least $1 - (|Q|-1)d/|\mathcal{F}|$ which is at least $15/16$.

Overall, the success probability of the algorithm is at least $7/8$ and the claim is proved. $\square$

$\square$

We can now move to the concatenated version.

**Theorem 54** *Let $p : \mathcal{F}^m \to \mathcal{F}$ be a degree $d$ polynomial, let us identify elements of $\mathcal{F}$ with elements of $\{0,1\}^k$, and let $f : \{0,1\}^{km+k} \to \{0,1\}$ be such that $f(x,a) = p(x) \cdot a$. Suppose we are given oracle access to $g : \{0,1\}^{km+k} \to \{0,1\}$ such that $f$ and $g$ agree on at least a fraction $1/2 + \epsilon$ of the inputs, where $|\mathcal{F}| \geq 8d/\epsilon^6$. Then there is an oracle circuit $C$ of size polynomial in $m$, $|F|$ and $2^k$ such that $C^g$ computes $p()$ everywhere.*

PROOF: It will be enough to show how to compute $p$ on a fraction $\epsilon^3$ of the inputs.

For every $x$, let $S(x) \subseteq \{0,1\}^k$ be the list of all strings $c$ such that $g(x,a)$ and $c\dot{a}$ agree on a fraction at least $1/2 + \epsilon/2$ of the values $a \in \{0,1\}^k$. Notice that $S(x)$ is computable in time polynomial in $2^k$ and that it contains at most $1/\epsilon^2$ elements. A Markov argument shows that

$$\mathbf{Pr}_x[\mathbf{Pr}_a[g(x,a) = p(x) \cdot a] \geq 1/2 + \epsilon/2] \geq \epsilon$$

Now, for each $x$ such that $\mathbf{Pr}_a[g(x,a) = p(x) \cdot a] \geq 1/2 + \epsilon/2$ we have $p(x) \in S(x)$ by definition. Let $c_i(x)$, for $i = 1, \ldots, 1/\epsilon^2$ be the $i$-th element of $S(x)$ (say, in lexicographic order, and if $S(x)$ has less than $i$ elements let $c_i(x)$ be $\mathbf{0}$). Again, we have that $c_i(x)$ is computable in polynomial in $2^k$ and, by an averaging argument, there must be an $i^*$ such that for at least an $\epsilon^3$ fraction of the $x$ we have $c_{i^*}(x) = p(x)$. We then let $C^f(x) = c_{i^*}(x)$, and apply Theorem 52. □

At this point, we are ready to state our main result, except that we need some more notation. Let us define $avg\mathbf{SIZE}_{p(n)}(t(n))$ to be the class of decision problems $L$ such that for every input length $n$ there is a circit $C$ of size $t(n)$ such that

$$\mathbf{Pr}_{x \in \{0,1\}^n}[C(x) = L_n(x)] \geq p(n)$$

**Corollary 55** *Suppose that* $\mathbf{EXP} \subseteq avg\mathbf{SIZE}_{1/2+1/n}\left(n^{O(1)}\right)$*, then* $\mathbf{EXP} \subseteq \mathbf{SIZE}(n^{O(1)})$*. Suppose that* $\mathbf{PSPACE} \subseteq avg\mathbf{SIZE}_{1/2+1/n}\left(n^{O(1)}\right)$*, then* $\mathbf{PSPACE} \subseteq \mathbf{SIZE}(n^{O(1)})$*.*

## 10.4 References

Polynomial encodings are used by Babai et al. [BFNW93] to prove average-case results for problems in **EXP**. The presentation in these notes is influenced by the more general perspective taken in [STV01]. Sudan's list-decoding algorithm for the Reed-Solomon code appears in [Sud97].

## Exercises

- Let $C : \{0,1\}^n \rightarrow \{0,1\}^m$ be an error-correcting code of minimum relative distance 3/4. Show that $n = 1$.
- Show that $avg\mathbf{BPP}_{1-1/n}(n^2)$ contains undecidable languages.
- This may be somewhat complicated: try to get a version of Theorem 52 that is more explicitely in terms of a list-decoding algorithm. The proof already shows that one can (withouth non-uniformity) define a list of $|\mathcal{F}|^{m+1}$ decoding algorithms (one for each choice of $y$ and $p(y)$) such that one of them is correct.
  - Show that (with a small loss in the parameters) $y$ can be chosen at random, so that we only need to enumerate over the $|\mathcal{F}|$ possible values for $p(y)$.
  - Can you modify the algorithm so that the list contains only $poly(1/\epsilon)$ elements instead of $|\mathcal{F}|$?.

# Lecture 11

# The XOR Lemma

We now see an alternative way of proving strong average-case hardness results, assuming we have as a starting point a function that is already somewhat hard on average. Part of these notes were written by Sanjeev Arora.

## 11.1 The XOR Lemma

**Definition 15 ($\delta$-hardness)** *Let $f$ be a boolean function on $n$-bit inputs, $\mathcal{D}$ a distribution on $n$ bit strings. If $\delta < 1/2$ and $g \in [n, \frac{2^n}{n}]$ then we say that $f$* is $\delta$-hard on distribution $\mathcal{D}$ for circuits of size $g$ *if for any boolean circuit $C$ with at most $g$ gates:*

$$\Pr_{x \in \mathcal{D}}[f(x) = C(x)] < 1 - \delta. \tag{11.1}$$

We want to transform $f$ into a function $f'$ that is very hard on average and that can be computed efficiently given $f$.

For a parameter $k$, consider the function $f^k : \{0,1\}^{nk} \to \{0,1\}$ defined as

$$f^k(x_1, \ldots, x_k) = f(x_1) \oplus f(x_2) \oplus \cdots \oplus f(x_k) \ .$$

Yao's "XOR-Lemma" states that the fraction on inputs on which $f^k$ can be computed by an efficient circuit tends to 1/2 for large $k$. To see why this makes sense, let us first compute how successful we can be in computing $f^k$ if we just try to compute $f()$ on each input $x_i$ and the we xor the results. In each attempt we will be successful with probability at most $\delta$ (let us say exactly $\delta$ for simplicity). Then the probability of computing $f^k$ correctly is the same as the probability that we make an even number of errors. This is the same as defining independent 0/1 random variables $Z_1, \ldots, Z_k$, where for each $i$ $\mathbf{Pr}[Z_i = 0] = \delta$ and asking what is $\mathbf{Pr}[Z_1 \oplus \cdots \oplus Z_k = 0]$. We can think of each $Z_i$ as being sampled in the following equivalent way: with probability $1 - 2\delta$ we set $Z_i = 0$, and with probability $2\delta$ we flip an unbiased coin and let $Z_i$ be equal to the outcome of the coin. Then there is a probability $(1 - 2\delta)^k$ that when we sample $Z_1, \ldots, Z_k$ we never flip a coin, we then all $Z_i$ are zero and their xor is also zero; with the remaining $1 - (1 - 2\delta)^k$ probability we flip at least one coin, and the xor of the $Z_i$ contains the value of that coin xor-ed with other independent values, and so it is unbiased. Overall, we have

$$\mathbf{Pr}[Z_1 \oplus \cdots \oplus Z_k = 0] = (1 - 2\delta)^k + \frac{1}{2}(1 - (1 - 2\delta)^k) = \frac{1}{2} + \frac{1}{2}(1 - 2\delta)^k$$

that actually tends to 1/2 for large $k$.

What happens when we consider arbitrary circuits for $f^k$? The following result by Impagliazzo allows us to get a proof of the XOR Lemma that follows quite closely the outline of the above probabilistic reasoning.

**Definition 16** *A set $H \subseteq \{0,1\}^n$ is a $\epsilon$-hard core set for a boolean function $f$ with respect to size $S$ if for any boolean circuit $C$ with at most $S$ gates:*

$$\Pr_{x \in S}[f(x) = C(x)] < \frac{1}{2} + \epsilon \tag{11.2}$$

**Lemma 56 (Impagliazzo)** *Suppose boolean function $f$ is $\delta$-hard for size $S$ on $n$-bit strings. Then for every $\epsilon > 0$ there is a set $S$ of size $\geq \delta 2^n$ such that $f$ is $\epsilon$-hard core on $S$ for circuits of size $S\epsilon^2\delta^2/4$.*

This is an extremely strong result: from the $\delta$-hardness of $f$ we only know that for every circuit $C$ there is a subset $H_C$ of $\delta 2^n$ inputs such that $C$ fails on $H_C$. Impagliazzo's result states that there is a single set $H$ of $\delta 2^n$ inputs such that every circuit $C$ fails on about half the elements of $H$. (Notice the different order of quantifiers.)

From Impagliazzo's result, it is easy to prove the XOR Lemma.

**Theorem 57 (Yao's XOR Lemma)** *Let $f$ be a boolean function on $n$-bit inputs that is $\delta$-hard for circuits of size $S$. Then the function $f^{\oplus k}$ on $nk$-bit inputs defined as*

$$f^{\oplus k}(x_1, x_2, \ldots, x_k) = f(x_1) \oplus f(x_2) \oplus \cdots \oplus f(x_k)$$

*is $(\frac{1}{2} - \epsilon - (1-\delta)^k)$-hard for circuits of size $S' = S\epsilon^2\delta^2/4$.*

Proof: The proof is by contradiction. Assume some circuit $C$ of size at most $S'$ satisfies $\Pr[C(x_1, x_2, \ldots, x_k) = f^{\oplus k}(x_1, x_2, \ldots, x_k)] \geq 1/2 + \epsilon + (1-\delta)^k$.

By Impagliazzo's Lemma, there is an $\epsilon$-hard core set $H$ of $\delta 2^n$ inputs with respect to circuits of size $S'$. We construct a circuit $C'$ of size $S'$ that predicts $f$ on at least $1/2 + \epsilon$ fraction of inputs of $H$ and thus reach a contradiction.

Partition the set of $nk$-bit strings into $S_0, S_1, S_2, \ldots, S_k$, where $S_t$ consist of those $(x_1, \ldots, x_k)$ in which exactly $t$ of the $x_i$'s lie in $H$. At most $(1-\delta)^k$ fraction of inputs lie in $S_0$, so $C$ must predict $f^{\oplus k}$ with probability at least $1/2 + \epsilon$ on inputs from the other classes. Averaging shows there is a $t \geq 1$ such that $C$ predicts $f^{\oplus k}$ with probability at least $1/2 + \epsilon$ on inputs randomly chosen from $S_t$. Using nonuniformity, assume we know such a $t$.

Let $x$ be a random input from $H$; this is the input to $C'$. Pick $a_1, a_2, \ldots, a_{t-1}$ randomly from $H$ and $b_1, \ldots, b_{k-t-1}$ randomly from the complement of $H$. Randomly permute the $k$ inputs $a_1, a_2, \ldots, a_{t-1}, x, b_1, \ldots, b_{k-t-1}$ to get an $nk$-bit string $z$; note that $z$ is a random input from $S_t$. Apply $C$ on $z$. With probability at least $(1+\epsilon)/2$ we have $C(z) = f^{\oplus k}(z)$.

By averaging, there is a way to fix $a_1, a_2, \ldots, a_{t-1}, b_1, \ldots, b_{k-t-1}$ and their random permutation such that the circuit still outputs $f^{\oplus k}(z)$ with probability $1/2 + \epsilon$. Using nonuniformity again, we hardwire these choices into circuit $C$. Thus the circuit is left with $n$ input wires, into which $x$ is being fed. Noting that

$$f^{\oplus k}(z) = f(x) \oplus f(a_1) \oplus \cdots \oplus f(a_{t-1}) \oplus f(b_1) \oplus \cdots \oplus f(b_{k-t-1}),$$

we again use nonuniformity and assume we know $b = f(a_1) \oplus \cdots \oplus f(a_{t-1}) \oplus f(b_1) \oplus \cdots \oplus f(b_{k-t-1})$. We add a NOT gate at the top of our circuit iff $b = 1$. This gives a circuit $C'$ with $S'$ gates (remember we do not count NOT gates in circuit complexity) which on input $x$ produces $C(z) \oplus b$ as output. Furthermore, for at least $1/2 + \epsilon$ fraction of $x \in H$, the circuit outputs $f^{\oplus k}(z) \oplus b$, which is $f(x)$. Thus $C'$ computes $f$ on $1/2 + \epsilon$ of inputs from $H$, a contradiction. □

## 11.2 References

Yao's XOR Lemma was stated in the conference presentation of [Yao82]. Proofs of the Lemma appeared in [Lev87, Imp95a, GNW95, IW97]. The proof given here is Impagliazzo's [Imp95a].

## Exercises

1. Prove a version of Impagliazzo's Lemma for average-case algorithms that are errorless but may fail.

   We say that a circuit $C$ is errorless for a function $f : \{0,1\}^n \to \{0,1\}$ if $C(x) \in \{0, 1, fail\}$ for each $x$, and $C(x) = f(x)$ whenever $C(x) \neq fail$.

   We say that a function $f : \{0,1\}^n \to \{0,1\}$ is $\delta$-hard for errorles circuits of size $S$ if for every errorless circuit $C$ of size $\leq S$ for $f$ we have $\mathbf{Pr}_x[C(x) = fail] \geq \delta$.

   Prove that if $f : \{0,1\}^n \to \{0,1\}$ is $\delta$-hard for errorless circuits of size $S$, then there is a subset $H \subseteq \{0,1\}^n$, $|H| \geq \delta 2^n$ such that for every errorless circuit $C$ for $f$ of size $\leq S'$ we have $\mathbf{Pr}_{x \in H}[C(x) = fail] \geq 1 - \epsilon$, where $S' = \Omega(\epsilon S/(\log(1/\delta)))$.

2. Prove a "concatenation lemma" instead of a XOR Lemma. Let $f : \{0,1\}^n \to \{0,1\}$ be $\delta$-hard for circuits of size $S$. Then argue that the function $f'\{0,1\}^{kn} \to \{0,1\}^k$ defined as $f'(x_1, \ldots, x_k) = (f(x_1), \cdots, f(x_k))$ cannot be computed on more than a $\epsilon + (1-\delta)^k$ fraction of inputs by circuits of size $\epsilon^2\delta^2 S/4$.

# Lecture 12

# Levin's Theory of Average-case Complexity

Today we introduce Levin's theory of average-case complexity.

This theory is still in its infancy: in this chapter we will introduce the notion of "distributional problems," discuss various formalizations of the notion of "algorithms that are efficient on average," introduce a reducibility that preserves efficient average-case solvability, and finally prove that there is a problem that is complete for the distributional version of **NP** under such reductions. It is still an open question how to apply this theory to the study of natural distributional problems that are believed to be hard on average.

## 12.1 Distributional Problems

**Definition 17 (Distributional Problem)** *A* distributional problem is a pair $\langle L, \mu \rangle$, where $L$ is a decision problem and $\mu$ is a distribution over the set $\{0,1\}^*$ of possible inputs.

In other settings where average-case hardness is considered (e.g. in the study of one-way functions) we normally describe the distribution of inputs as a collection of distributions $\mu_1, \ldots, \mu_n, \ldots$ where $\mu_n$ is a distribution over the set $\{0,1\}^n$ of inputs of a given input length.[1] There are various reasons why this single-distribution approach is convenient for the purposes of this chapter. We will discuss it again later, but for now the basic intuition is that we will discuss reductions where the length of the output is not a function of the length of the input, so that sampling inputs from a fixed-length distribution and passing them to a reduction does not produce a fixed-length distribution (unlike the case of cryptographic reductions).

We will restrict to the study of distributional problems where $\mu$ is "polynomial-time computable." What do we mean by that? Fo all $x \in \{0,1\}^*$, let

$$\mu(x) = \sum_{y \leq x} \Pr[y]. \qquad (12.1)$$

[1]One can always reduce the approach of distinct distributions to the approach of this chapter by assuming that $\mu$ first picks at random a certain input length $n$, and then it samples from $\mu_n$.

where '$\leq$' denotes lexicographic ordering. Then $\mu$ must be computable in poly $(|x|)$ time. Clearly this notion is at least as strong as the requirement that $\Pr[x]$ be computable in polynomial time, because

$$\Pr[x] = \mu\prime(x) = \mu(x) - \mu(x-1), \tag{12.2}$$

$x-1$ being the lexicographic predecessor of $x$. Indeed one can show that, under reasonable assumptions, there exist distributions that are efficiently computable in the second sense but not polynomial-time computable in our sense.

We can define the "uniform distribution" to be

$$\Pr[x] = \frac{1}{|x|(|x|+1)} 2^{-|x|}; \tag{12.3}$$

that is, first choose an input size at random under some polynomially-decreasing distribution, then choose an input of that size uniformly at random. It is easy to see that the uniform distribution is polynomial-time computable.

## 12.2 DistNP

We define the complexity class

$$\mathbf{DistNP} = \{\langle L, \mu\rangle : L \in NP, \mu \text{ polynomial-time computable}\}. \tag{12.4}$$

There are at least two good reasons for looking only at polynomial-time computable distributions.

1. One can show that there exists a distribution $\mu$ such that every problem is as hard on average under $\mu$ as it is in the worst case. Therefore, unless we place some computational restriction on $\mu$, the average-case theory is identical to the worst-case one.

2. Someone, somewhere, had to generate the instances we're trying to solve. If we place computational restrictions on ourselves, then it seems reasonable also to place restrictions on whoever generated the instances.

It should be clear that we need a whole *class* of distributions to do reductions; that is, we can't just parameterize a complexity class by a single distribution. This is because a problem can have more than one natural distribution; it's not always obvious what to take as the 'uniform distribution.'

## 12.3 Reductions

**Definition 18 (Reduction)** *We say that a distributional problem $\langle L_1, \mu_1\rangle$* reduces *to a distributional problem $\langle L_2, \mu_2\rangle$ (in symbols, $\langle L_1, \mu_1\rangle \leq \langle L_2, \mu_2\rangle$) if there exists a polynomial-time computable function $f$ such that:*

*1. $x \in L_1$ iff $f(x) \in L_2$.*

2. *There is a* $\epsilon > 0$ *such that, for every* $x$, $|f(x)| = \Omega(|x|^\epsilon)$.

3. *For all* $y \in \{0,1\}^*$,
$$\sum_{x:f(x)=y} \mu\prime_1(x) \leq \text{poly}(|y|)\, \mu\prime_2(y)\,. \tag{12.5}$$

The first condition is the standard condition of many-to-one reductions in complexity theory: it ensures that an algorithm that is always correct for $L_2$ can be converted into an algorithm that is always correct for $L_1$. The second condition is a technical one, that is necessary to show that the reduction preserves efficient-on-average algorithms. All known reductions satisfies this condition naturally.

The third condition is called *domination*. To motivate this condition, consider that we want reductions to preserve the existence of algorithms that are efficient on average. Suppose that we have an algorithm $A_2$ for problem $L_2$ such that, when we pick $y$ according to distribution $\mu_2$, $A(y)$ is efficient on average; if we want to solve $L_1$ under distribution $\mu_1$, then, starting from an input $x$ distributed according to $\mu_1(x)$, we compute $f(x)$ and then apply algorithm $A_2$ to $f(x)$. This will certainly be correct, but what about the running time? Intuitively, it could be the case that $A_2$ is very slow on some inputs, but such inputs are unlikely to be sampled according to distribution $\mu_2$; the domination condition ensures us that such inputs are also unlikely to be sampled when we sample $x$ according to $\mu_1$ and then consider $f(x)$.

## 12.4 Polynomial-Time on Average

Given a problem $\langle L, \mu \rangle$ and an algorithm $A$ that runs in time $t(x)$ on input $x$, what does it mean to say that $A$ solves $\langle L, \mu \rangle$ in polynomial time on average? We will consider some flawed definitions before settling on the best one and on an alternate one.

A first difficulty comes with the fact that we are dealing with a single distribution on all possible inputs. The most intuitive choice of saying that $A$ is efficient if

$$\mathbf{E}[t(x)] \text{ is small}$$

is problematic because the expectation could be infinite even if $A$ runs in worst-case polynomial time.

One can work around this difficulty by defining $A$ to be polynomial provided that for some constant $c$ and for every sufficiently large $n$,

$$\mathbf{E}[t(x)|\ |x| = n] \leq n^c$$

However we chose to define distributional problems and reducibility without separating problems by input size, and we would run into several difficulties in separating them now. Besides, it seems reasonable that there could be input lengths on which $A$ takes a long time, but that are generated with very low probability under $\mu$; in such cases $A$ may still be regarded as efficient, but this is not taken into account in the above definition.

Our next attempt folds the polynomial running time into the single distribution by defining $A$ to be polynomial on average if there is a polynomial $p$ such that

$$\mathbf{E}\left[\frac{t(x)}{p(|x|)}\right] = O(1)$$

This definition is quite appealing, but is still subject to the fatal of not being *robust*, in that: (1) reductions do not preserve this definition of polynomial solvability on average and (2) the definition is sensitive to trivial representation changes such as replacing a matrix representation of a graph by an adjacency list.

To see why these problems arise, let $\mu$ be the uniform distribution, and let

$$t(x) = 2^n \text{ if } x = \overrightarrow{0},\ t(x) = n^2 \text{ otherwise.} \tag{12.6}$$

The average running time is about $n^2$. But suppose now that $n$ is replaced by $2n$ (because of a change in representation, or because of the application of a reduction), then

$$t(x) = 2^{2n} \text{ if } x = \overrightarrow{0},\ t(x) = 4 \cdot n^2 \text{ otherwise.} \tag{12.7}$$

Similarly, if $t(x)$ is replaced by $t^2(x)$, the average running time becomes exponential.

We now come to a satisfying definition.

**Definition 19 (Polynomial on average)** *Suppose $A$ is an algorithm for a distributional problem $\langle L, \mu\rangle$ that runs in time $t(x)$ on input $x$. We say that $A$ has polynomial running time on average is there is a constant $c$ such that*

$$\mathbf{E}\left[\frac{t(x)^{1/c}}{|x|}\right] = O(1)$$

Notice, first, that this definition is satisfied by any algorithm that runs in worst-case polynomial time. If $t(x) = O(|x|^c)$, then $t(x)^{1/c} = O(|x|)$ and the sum converges. More interestingly, suppose $t()$ is a time bound for which the above definition is satisfied; then an algorithm whose running time is $t'(x) = t(x)^2$ also satisfies the definition, unlike the case of the previous definition. In fact we have the following result, whose non-trivial proof we omit.

**Theorem 58** *If $\langle L_1, \mu_1\rangle \leq \langle L_2, \mu_2\rangle$ and $\langle L_2, \mu_2\rangle$ admits an algorithm that is polynomial on average, then $\langle L_1, \mu_1\rangle$ also admits an algorithm that is polynomial on average.*

There is an additional interesting property of the definition of polynomial of average: there is a high probability that the algorithm runs in polynomial time.

Suppose that

$$\mathbf{E}\left[\frac{t(x)^{1/c}}{|x|}\right] = O(1). \tag{12.8}$$

and that we wish to compute $\Pr[t(x) \geq k \cdot |x|^c]$. Such a probability is clearly the same as

$$\Pr[t(x)^{1/c} \geq k^{1/c}|x|]$$

and by Markov's inequality this is at most $O(1/k^{1/c})$, which can be made fairly small by picking $k$ large enough. Since the algorithm runs in time at most $kn^c$ for a subset of inputs having probability $1 - k^{-1/c}$, we see that our definition gives a smooth quantitative tradeoff for how much time we need to solve an increasing fraction of inputs.

In the setting of one-way functions and in the study of the average-case complexity of the permanent and of problems in EXP (with applications to pseudorandomness), we normally interpret "average case hardness" in the following way: that an algorithm of limited running time will fail to solve the problem on a noticeable fraction of the input. Conversely, we would interpret average-case tractability as the existence of an algorithm that solves the problem in polynomial time, except on a negligible fraction of inputs. This leads to the following formal definition.

**Definition 20 (Heuristic polynomial time)** *We say that an algorithm $A$ is a heuristic polynomial time algorithm for a distributional problem $\langle L, \mu\rangle$ if $A$ always runs in polynomial time and for every polynomial $p$*

$$\sum_{x : A(x) \neq \chi_L(x)} \mu'(x) p(|x|) = O(1)$$

In other words, a polynomial time algorithm for a distributional problem is a heuristic if the algorithm fails on a negligible fraction of inputs, that is, a subset of inputs whose probability mass is bounded even if multiplied by a polynomial in the input length. It might also make sense to consider a definition in which $A$ is always correct, although it does not necessarily work in polynomial time, and that $A$ is heuristic polynomial time if there is a polynomial $q$ such that for every polynomial $p$, $\sum_{x \in S_q} \mu'(x) p(|x|) = O(1)$, where $S_q$ is the set of inputs $x$ such that $A(x)$ takes more than $q(|x|)$ time. Our definition is only more general, because from an algorithm $A$ as before one can obtain an algorithm $A$ satisfying Definition 20 by adding a clock that stops the computation after $q(|x|)$ steps.

The definition of heuristic polynomial time is *incomparable* with the definition of average polynomial time. For example, an algorithm could take time $2^n$ on a fraction $1/n^{\log n}$ of the inputs of length $n$, and time $n^2$ on the remaining inputs, and thus be a heuristic polynomial time algorithm with respect to the uniform distribution, while not beign average polynomial time with respect to the uniform distribution. On the other hand, consider an algorithm such that for every input length $n$, and for $1 \leq k \leq 2^{n/2}$, there is a fraction about $1/k^2$ of the inputs of length $n$ on which the algorithm takes time $\Theta(kn)$. Then this algorithm satisfies the definition of average polynomial time under the uniform distribution, but if we impose a polynomial clock there will be an inverse polynomial fraction of inputs of each length on which the algorithm fails, and so the definition of heuristic polynomial time cannot be met.

It is easy to see that heuristic polynomial time is preserved under reductions.

**Theorem 59** *If $\langle L_1, \mu_1\rangle \leq \langle L_2, \mu_2\rangle$ and $\langle L_2, \mu_2\rangle$ admits a heuristic polynomial time algorithm, then $\langle L_1, \mu_1\rangle$ also admits a heuristic polynomial time algorithm.*

PROOF: Let $A_2$ be the algorithm for $\langle L_2, \mu_2\rangle$, let $f$ be the function realizing the reduction, and let $p$ be the polynomial witnessing the domination property of the reduction. Let $c$ and $\epsilon$ be such that for every $x$ we have $|x| \leq c|f(x)|^{1/\epsilon}$.

Then we define the algorithm $A_1$ than on input $x$ outputs $A_2(f(x))$. Clearly this is a polynomial time algorithm, and whenever $A_2$ is correct on $f(x)$, then $A_1$ is correct on $x$. We need to show that for every polynomial $q$

$$\sum_{x:A_2(f(x))\neq\chi_{L_2}(f(x))} \mu_1'(x)q(|x|) = O(1)$$

and the left-hand side can be rewritten as

$$\begin{aligned}
& \sum_{y:A_2(y)\neq\chi_{L_2}(y)} \sum_{x:f(x)=y} \mu_1'(x)q(|x|) \\
\leq & \sum_{y:A_2(y)\neq\chi_{L_2}(y)} \sum_{x:f(x)=y} \mu_1'(x)q(c\cdot|y|^{1/\epsilon})) \\
\leq & \sum_{y:A_2(y)\neq\chi_{L_2}(y)} \mu_2'(y)p(|y|)q'(|y|) \\
= & \; O(1)
\end{aligned}$$

where the last step uses the fact that $A_2$ is a polynomial heuristic for $\langle L_2, \mu_2\rangle$ and in the second-to-last step we introduce the polynomial $q'(n)$ defined as $q(c\cdot n^{1/\epsilon})$

$\Box$

## 12.5 Existence of Complete Problems

We now show that there exists a problem (albeit an artificial one) complete for **DistNP**. Let the inputs have the form $\langle M, x, 1^t, 1^l\rangle$, where $M$ is an encoding of a Turing machine and $1^t$ is a sequence of $t$ ones. Then we define the following "universal" problem $U$.

- Decide whether there exists a string $y$ such that $|y| \leq l$ and $M(x,y)$ accepts in at most $t$ steps.

That $U$ is **NP**-complete follows directly from the definition. Recall the definition of **NP**: we say that $L \in$ **NP** if there exists a machine $M$ running in $t = \text{poly}(|x|)$ steps such that $x \in L$ iff there exists a $y$ with $y = \text{poly}(|x|)$ such that $M(x,y)$ accepts. Thus, to reduce $L$ to $U$ we need only map $x$ onto $R(x) = \langle M, x, 1^t, 1^l\rangle$ where $t$ and $l$ are sufficiently large bounds.

To give a reduction that satisfies the domination condition is indeed harded. Let $\langle L, \mu\rangle \in$ **DistNP**. Define a uniform distribution over the $\langle M, x, 1^t, 1^l\rangle$ as follows:

$$\mu'\left(\left\langle M, x, 1^t, 1^l\right\rangle\right) = \frac{1}{|M|\left(|M|+1\right)2^{|M|}} \cdot \frac{1}{|x|\left(|x|+1\right)2^{|x|}} \cdot \frac{1}{(t+l)\left(t+l+1\right)}. \qquad (12.9)$$

The trouble is that, because of the domination condition, we can't map $x$ onto $R(x)$ if $\mu\prime(x) > \text{poly}(|x|)\,2^{-|x|}$. We work around this problem by *compressing* $x$ to a shorter string if $\mu\prime(x)$ is large. Intuitively, by mapping high-probability strings onto shorter lengths, we make their high probability less conspicuous. The following lemma shows how to do this.

**Lemma 60** *Suppose $\mu$ is a polynomial-time computable distribution over $x$. Then there exists a polynomial-time algorithm $C$ such that*

1. *$C$ is injective: $C(x) \neq C(y)$ iff $x \neq y$.*
2. *$|C(x)| \leq 1 + \min\left\{|x|, \log \frac{1}{\mu\prime(x)}\right\}$.*

Proof: If $\mu\prime(x) \leq 2^{-|x|}$ then simply let $C(x) = 0x$, that is, 0 concatenated with $x$. If, on the other hand, $\mu\prime(x) > 2^{-|x|}$, then let $C(x) = 1z$. Here $z$ is the longest common prefix of $\mu(x)$ and $\mu(x-1)$ when both are written out in binary. Since $\mu$ is computable in polynomial time, so is $z$. $C$ is injective because only two binary strings $s_1$ and $s_2$ can have the longest common prefix $z$; a third string $s_3$ sharing $z$ as a prefix must have a longer prefix with either $s_1$ or $s_2$. Finally, since $\mu\prime(x) \leq 2^{-|z|}$, $|C(x)| \leq 1 + \log \frac{1}{\mu\prime(x)}$. $\square$

Now the reduction is to map $x$ onto $R_2(x) = \left\langle \overline{M}, C(x), 1^{\overline{t}}, 1^{l+|x|} \right\rangle$. Here $\overline{M}$ is a machine that on input $z, x, y$ checks that $C(x) = z$ (i.e., that $x$ is a valid decoding of $z$) and that $M(x,y)$ accepts. The running time of $\overline{M}$ is $\overline{t}$. Clearly $x \in L$ iff $\overline{M}$ accepts. To show that domination holds, let $\mu_2\prime(x) = \Pr[R_2(x)]$. Then, since the map is one-to-one, we need only show that $\mu\prime(x) \leq \text{poly}(|x|)\, \mu\prime_2(x)$. Since $\overline{t} = O(\text{poly}(t))$,

$$\begin{aligned}\mu\prime_2(x) &= \frac{1}{O\left(\left|\overline{M}\right|^2\right) 2^{\left|\overline{M}\right|}} \cdot \frac{1}{O\left(|C(x)|^2\right) 2^{|C(x)|}} \cdot \frac{1}{O(t+l+|x|)^2} \\ &\geq \text{poly}(|x|) \max\left(2^{-|x|}, \mu\prime(x)\right) \\ &\geq \text{poly}(|x|)\, \mu\prime(x)\end{aligned}$$

and we are done.

Note that, since we mapped longer inputs to shorter ones, we could not have done this reduction input-length-wise.

## 12.6 Polynomial-Time Samplability

**Definition 21 (Samplable distributions)** *We say that a distribution $\mu$ is* polynomial-time samplable if there exists a probabilistic algorithm $A$, taking no input, that outputs $x$ with probability $\mu\prime(x)$ and runs in poly$(|x|)$ time.

Any polynomial-time computable distribution is also polynomial-time samplable, provided that for all $x$,

$$\mu\prime(x) \geq 2^{-\text{poly}(|x|)} \text{ or } \mu\prime(x) = 0. \tag{12.10}$$

For a polynomial-time computable $\mu$ satisfying the above property, we can indeed construct a sampler $A$ that first chooses a real number $r$ uniformly at random from $[0,1]$, to poly$(|x|)$ bits of precision, and then uses binary search to find the first $x$ such that $\mu(x) \geq r$.

On the other hand, under reasonable assumptions, there are efficiently samplable distributios $\mu$ that are not efficiently computable.

In addition to **DistNP**, we can look at the class

$$\langle \mathbf{NP}, \mathbf{P}\text{-samplable}\rangle = \{\langle L, \mu\rangle : L \in \mathbf{NP}, \mu \text{ polynomial-time samplable}\}\,. \qquad (12.11)$$

A result due to Impagliazzo and Levin [IL90] states that if $\langle L, \mu\rangle$ is **DistNP**-complete, then $\langle L, \mu\rangle$ is also complete for the class $\langle \mathbf{NP}, \mathbf{P}\text{-samplable}\rangle$.

This means that the completeness result established in the previous section extends to the class of NP problems with samplable distributions.

## 12.7 References

Levin's theory of average-case complexity was introduced in [Lev86]. Ben-David et al. [BDCGL92] prove several basic results about the theory. Impagliazzo and Levin [IL90] show that the theory can be generalized to samplable distributions. Impagliazzo [Imp95b] wrote a very clear survey on the subject. Another good reference is a survey paper by Goldreich [Gol97].

## Exercises

1. For a parameter $c$, consider the distribution $D_{n,cn}$ over instances of 3SAT with $n$ variables generated by picking $cn$ times independently a random a clause out of the $8\binom{n}{3}$ possible clauses that can be constructed from $n$ variables. (Note that the same clause could be picked more than once.) Let $D_c$ be the distribution obtained by first picking a number $n$ with probability $1/n(n+1)$ and then sampling from $D_{n,cn}$.

   (a) Show that an instance from $D_{n,cn}$ is satisfiable with probability at least $(7/8)^{cn}$ and at most $2^n \cdot (7/8)^{cn}$.

   (b) Argue that, using the definition given in this lecture, $D_{15}$ cannot be reduced to $D_{30}$.

   [Hint: take a sufficiently large $n$, and then look at the probability of satisfiable instances of length $n$ under $D_15$ and the probability that their image is generated by $D_30$.]

# Lecture 13

# Interactive Proofs

In this lecture we will look at the notion of *Interactive Proofs* and the induced class **IP**. Intuitively, this class is obtained by adding the ingredients Interaction and Randomness to the class **NP**. As we shall see, adding either one does not make much difference but the two put together make **IP**a very powerful class.

## 13.1 Interactive Proofs

We begin be recalling the definition of **NP**. Given a language $L$, we say that: $L \in \mathbf{NP}$ iff there is an algorithm $V(\cdot, \cdot)$ running in polynomial time $T(\cdot)$ and a polynomial $p(\cdot)$ such that

$$x \in L \Leftrightarrow \exists y, |y| \leq p(|x|) \text{ and } V(x, y) \text{ accepts}$$

We can rewrite the above as two separate conditions, using elementary ideas from logic.

$$\begin{aligned} x \in L &\Rightarrow \exists y, |y| \leq p(|x|) \text{ and } V(x, y) \text{ accepts} \qquad \text{(Completeness)} \\ x \notin L &\Rightarrow \forall y, |y| \leq p(|x|) \text{ and } V(x, y) \text{ rejects} \qquad \text{(Soundness)} \end{aligned}$$

We may generalise this to the figure below 13.1 where the process of recognizing if $x \in L$ is carried out by a *Verifier* $V$ that must run in polynomial time, and a *Prover* $P$, who has unbounded time and space. The task of the prover is to convince the verifier that $x \in L$ and the verifier is a skeptic who's task is to be convinced if and only if the string actually belongs in the language.

Now the class **NP** is the set of languages $L$ such that $L \in \mathbf{NP}$ iff there is a Prover $P$ and a polynomial time verifier $V$ such that:

$$\begin{aligned} x \in L &\Rightarrow P \text{ has a strategy to convince V} \qquad \text{(Completeness)} \\ x \notin L &\Rightarrow P \text{ has no strategy to convince } V \qquad \text{(Soundness)} \end{aligned}$$

By *strategy* we mean in this case the certificate or proof that is polynomially small, (corresponding to $y$ in the figure 13.1) that $P$ supplies to $V$, while later we will generalise it to interaction *i.e.* where a sequence of messages is exchanged between the two and a strategy for $P$ is a function from the sequence of messages seen to the next message that $P$ must send to $V$.

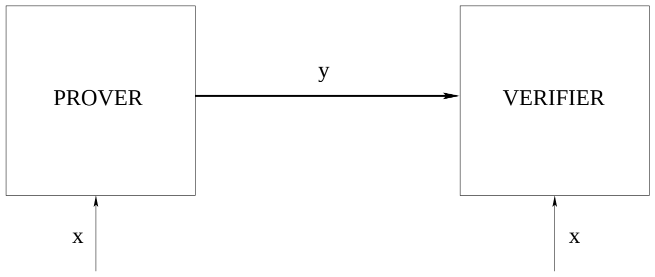


Figure 13.1: **NP**with a Prover-Verifier mechanism

Thus, we generalise the above definition to get the class **IP**. The class **IP**differs in the following two ways:

1. There is randomness, the verifier may be a randomized machine.
2. There is interaction: unlike the above where a single static proof is sent across, there are rounds of communication, where the verifier may "ask" further questions of the prover based on the messages that have been sent to it.

First, we shall see that both of the above are required.

### 13.1.1 NP+ Interaction = NP

Let the class $\overline{\mathbf{NP}}$ be the class corresponding to **NP** + interaction. That is we allow the verifier to ask questions and thus the strategy is a function from sequences of messages (the past communication) to messages (the next message that the verifier must send). We say that the class $\overline{\mathbf{NP}}$ is the set of languages $L$ such that $L \in \overline{\mathbf{NP}}$ iff there is a Prover $P$ and a polynomial time verifier $V$ such that:

$$x \in L \Rightarrow P \text{ has a strategy to convince } V$$
$$x \notin L \Rightarrow P \text{ has no strategy to convince } V$$

It is easy to check that $\mathbf{NP} = \overline{\mathbf{NP}}$. We only need to show $\overline{\mathbf{NP}} \subseteq \mathbf{NP}$, as the other inclusion is obvious. Say $L \in \overline{\mathbf{NP}}$, let $P, V$ be the prover and verifier for $L$. Define $P', V'$ the one-round prover and verifier as follows: $P'$ sends to $V'$ the entire transcript of the interaction between $P$, $V$ as the certificate. Suppose that interaction was the set of messages $y_1, \ldots, y_n$ as in the figure 13.2 below. $V'$ is essentially like $V$ only it has been given all the answers to its questions at once. Thus it takes the first message of $P$, $y_1$, its response (behaving as $V$) is exactly $y_2$, but $y_3$ is the reply to that, and so on, it "asks" the questions and is supplied all the answers on the same transcript, and it accepts iff after this exchange $V$ accepted. Thus the language accepted by $P', V'$ is the same as $L$ and so $L \in \mathbf{NP}$. Note that the entire set of messages is polynomially long as there are polynomially many messages and each is polynomially long.

Note that for the above to work, the prover must at the very beginning know what responses $V$ may make, which is possible only if $V$ is deterministic. If $V$ was randomized then to cover all possible questions in a polynomially long interaction $P'$ would have to send an exponentially large transcript.

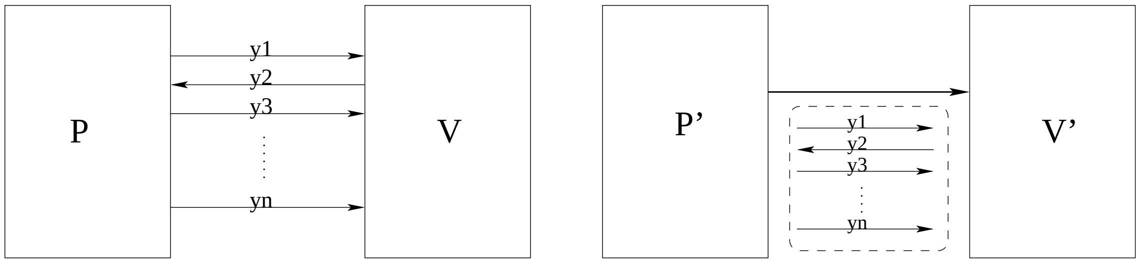


Figure 13.2: Simulating several rounds of interaction in one round

### 13.1.2 NP + Randomness

By adding randomness to the class **NP** we get the class **MA**.

**Definition 22 (MA)** *$L \in \mathbf{MA}$ iff there exists a probabilistic polynomial time machine $V$ such that:*

$$x \in L \Rightarrow \exists y.\mathbf{Pr}[V(x,y) \text{ accepts }] \geq \frac{2}{3}$$
$$x \in L \Rightarrow \forall y.\mathbf{Pr}[V(x,y) \text{ accepts }] \leq \frac{1}{3}$$

As in similar classes, the success probabilities may be boosted arbitrarily high. It is conjectured that **MA** = **NP**. it is known that if **coNP** $\subseteq$ **MA** then the polynomial heirarchy collapses.

We are now in a position to define formally the class **IP**.

## 13.2 IP

**Definition 23 (IP)** *A language $L$ is in $\mathbf{IP}(r(\cdot))$ iff there exists a probabilistic polynomial time verifier $V$ such that:*

$$x \in L \Rightarrow \exists P.\mathbf{Pr}[V \text{ interacting with } P \text{ accepts }] \geq \frac{2}{3}$$
$$x \in L \Rightarrow \forall P.\mathbf{Pr}[V \text{ interacting with } P \text{ accepts }] \leq \frac{1}{3}$$

*Where $V$ also uses at most $r(|x|)$ rounds of interaction.*

Note that by *round* we really mean a single "message" and not a question-answer sequence. That is, by two rounds we mean that the Verifier asks a question of the Prover and the Prover replies, which is exactly two messages.

A related class is **AM** which is defined as follows:

**Definition 24 (AM)** *A language $L$ is in $\mathbf{AM}(r(\cdot))$ iff $L \in \mathbf{IP}(r(\cdot))$ and at each round the verifier sends a random message, that is a message that is completely random and independent of all previous communication.*

The above is also known as *Public Coin Proof Systems*.

There are the following surprising theorems about the classes **AM** and **IP**.

**Theorem 61** $\mathbf{IP}(r(n)) \subseteq \mathbf{AM}(r(n)+2)$

**Theorem 62** $\forall r(n) \geq 2,\ \mathbf{AM}(2r(n)) \subseteq \mathbf{AM}(r(n))$.

This yields the following corollaries:

**Corollary 63** $\mathbf{AM}(O(1)) \subseteq \mathbf{AM}(2)$

That is, all constant round **AM** proofs can be done in just two rounds. And also:

**Corollary 64** **IP***(O(1))*=**AM***(O(1))*=**AM***(2)*

Finally we have the famous theorem of Shamir:

**Theorem 65** $\mathbf{IP}(\mathit{poly}(n)) = \mathbf{PSPACE}$

We know that $\mathbf{AM}(2)$ is good enough for systems with a constant number of rounds, but it would be surprising indeed if polynomially many rounds could be simulated in a constant number of round. Due to the above result, when people say **AM**they mean $\mathbf{AM}(2)$, but when people say **IP** they mean $\mathbf{IP}(\text{poly})$.

There is also the following theorem that gives an upper bound on **IP** with constant rounds.

**Theorem 66** *If* **coNP** $\subseteq \mathbf{IP}(O(1))$ *then the polynomial heirarchy collapses.*

Thus for a long time it was believed that **IP**could not capture **coNP**as it was thought that **IP**with a constant number of rounds was roughly as good as **IP**with polynomially many rounds. However, it was shown that in fact **coNP**was contained in $\mathbf{IP}(\text{poly}(n))$, the proof of which we shall shortly see and a few weeks after that result, Shamir proved that in fact **IP** was the same as **PSPACE**.

## 13.3 An Example: Graph Non-Isomorphism

To get a feel of the power of interaction and randomness let us look at an example, Graph Non-Isomorphism. The problem is the following:

- Input: Graphs $G_1 = (V_1, E_1),\ G_2 = (V_2, E_2)$
- Output: Decide whether there exists *NO* $\pi : V_1 \mapsto V_2$ such that $(u, v) \in E_1$ iff $(\pi(u), \pi(v)) \in E_2$.

This problem is not known to be in **NP**. The difficulty lies in constructing a succint proof that no permutation of the vertices is good, given that there are exponentially many permutations, let us see how this is in $\mathbf{AM}(2)$.

Suppose that the skeptic Arthur (the verifier) has the two graphs $G_1, G_2$ and wishes to know whether or not all the permutations are bad. His only resources are randomness and his wizard Merlin who is all knowing (the prover).

He does the following:

1. Picks a random $b \in \{1,2\}$.
2. Creates $G$ to be a random permutation of $G_b$.
3. Sends $G$ to Merlin and demands to know of Merlin which graph he sent him a permutation of.
4. If Merlin was correct Arthur believes that the graphs are non-isomorphic (accepts), otherwise Merlin was incorrect, Arthur believes the graphs are isomorphic (rejects).

Notice that if the graphs are really non-isomorphic, then Merlin will always be able to tell which graph was sent to him as only one of the graphs will be isomorphic to the random permutation that Merlin is given, and with his infinite resources Merlin will figure out which one and thus will reply correctly and so Arthur will accept with probability 1.

If the graphs are isomorphic then Merlin cannot tell as the permutation is a permutation of both $G_1$, $G_2$. Merlin's output depends only on $G_1, G_2, G$, say it is $i$. Now $i = b$ with probability $\frac{1}{2}$ as $b$ is picked independantly at the start. Thus in the case the graphs are isomorphic, Arthur accepts only with probability $\frac{1}{2}$. By repeating this $k$ times we can shrink this to $2^{-k}$.

Thus we have a protocol, a prover and a verifier for Graph Non-Isomorphism and so it is in $\mathbf{AM}(2)$.

**Note:** The verifier of the above has learnt nothing at all about the Graphs that he did not know before. He does not have a proof or certificate with which he may convince a third party that the Graphs are non-isomorphic. The entire power was in the interaction between it and the prover, and the one bit that the prover gave it gives it significant confidence but no *knowledge*. This was therefore a *Zero Knowledge* Proof.

**Remark 1** Graph Non-Isomorphism $\in \mathbf{IP}(2) = \mathbf{AM}(2)$. *Thus,* Graph Isomorphism *is not* **NP***-complete unless the polynomial heirarchy collapses. Thus is because, if* Graph Isomorphism *is* **NP***-complete then* Graph Non-Isomorphism *is* **coNP***-complete, and from the above result,* $\mathbf{coNP} \subseteq \mathbf{AM}(2)$ *and from the theorem 66, the polynomial heirarchy would collapse.*

## 13.4 Random Self-Reductions

It is not known how to prove worst-case to average-case equivalence for **NP**-complete problems. In this section we show that an approach similar to the one taken for #**P**, **PSPACE** and **EXP** does not work.

The worst-case to average-case equivalence results that we have seen so far rely on procedures to compute polynomials exactly given a procedure that computes the polynomial well on average. In all cases, given a point $x$ on which we want to evaluate the polynomial, we pick a random line passing through $x$ and then evaluate the good-on-average procedure on points along the line. Since each such point is uniformly distributed, we are able to get the correct value of the polynomial on all points (in the simplest case) or on a majority of points (in the procedures based on Berlekamp-Welch). The following definition captures the way such reductions work.

**Definition 25 (Random Self Reduction)** *We say that a problem $L$ has a* non-adaptive random self reduction *if there is a probabilistic polynomial time algorithm $R$ such that, given an input $x \in \{0,1\}^n$, $R(x)$ produces a sequence of inputs $y_1, \ldots, y_k \in \{0,1\}^{n'}$, where $k, n' = n^{O(1)}$, and then given $L(y_1), \ldots, L(y_k)$, $R$ decides correctly whether $x \in L$ with high probability. Furthermore, for every $x \in \{0,1\}^n$ and every $i$, the string $y_i$ is uniformly distributed in $\{0,1\}^{n'}$*

The intuition for the use of a random self-reduction is that if we have an algorithm $A$ that solves $L$ on, say, a fraction $1 - 1/4k$ of the elements of $\{0,1\}^{n'}$, then we can run $R$, generate $y_1, \ldots, y_k$, and then continue using the answers $A(y_1), \ldots, A(y_k)$, which are going to be all correct with probability at least $3/4$. So $A$ can be used to solve $L$ everywhere on inputs of length $n$.

We note that the above definition is restrictive in two ways: $R$ generates the points $y_1, \ldots, y_k$ non-adaptively, that is, it chooses $y_2$ before knowing $L(y_1)$, and so on. Secondly, it is not really necessary that all points $y_1, \ldots, y_k$ be uniformly distributed. Suppose that each point $y_i$ is just generated according to a "smooth" distribution (that is, say, no element of $\{0,1\}^{n'}$ is generated with probability larger than $c/2^{n'}$), and that we have an algorithm $A$ that succeeds on at least a fraction $1 - 1/4ck$ of $\{0,1\}^{n'}$. Then using the randomized reduction together with $A$ we still have at least a probability $3/4$ that all queries are correctly answered.

Nothing is known about the existence of adaptive and/or smooth random self-reduction, but the following theorem rules out non-adaptive uniform random self-reduction.

**Theorem 67** *Suppose an* **NP***-complete language $L$ has a random self-reduction as in Definition 25. Then the polynomial hierarchy collapses to $\Sigma_3$.*

Proof: We show that if $L \in$ **NP** and $L$ has a random self-reduction, then the complement of $L$ has an $AM[2]$ protocol (that uses some non-uniformity), and if $L$ were **NP**-complete this would imply the collapse of the hierarchy.

Let $R$ be the random self-reduction for $L$, let us define a protocol for the complement of $L$ on inputs of length $n$; let $n'$ be the length of the queries generated by $R(x)$, $k$ the number of queries, and let

$$p = \mathbf{Pr}_{y \in \{0,1\}^{n'}}[y \in L]$$

The value $p$ will be the non-uniformity used in the protocol.

We fix a parameter $m = \Theta((k \log k)^2)$. Given $x \in \{0,1\}^n$:

- The verifier runs $R(x)$ $m$ times indepedently, with fresh randomness each time, and generates queries $(y_1^1, \ldots, y_k^1), \ldots, (y_1^m, \ldots, y_k^m)$. The verifier sends all such queries to the prover.[1]
- The prover responds by telling whether each $y_j^i$ is in $L$ or not, and by certifying all the YES answers with an **NP** witness (remember that $L$ is in **NP**).
- The verifier accepts if and only if the following conditions hold:

[1]To make it an AM protocol, the verifier may just send the randomness it used to generate the $m$ runs of $R(x)$, and let the prover figure out the corresponding queries $y_j^i$.

1. $R(x)$ rejects in all the $m$ iterations, assuming the answers from the prover were correct,
2. all certificates sent by the prover are valid, and
3. for every $j = 1, \dots k$ at least $pm - m/2k$ of the queries $y_j^1, \dots, y_j^m$ are answered YES

We need to argue that if $x \notin L$ then the verifier accepts with high probability when interacting with a valid prover, and that if $x \in L$ then the verifier accepts with low probability when interacting with an arbitrary cheating prover.

If $x \notin L$ and the prover follows the protocol, then $R(x)$ rejects in all the $m$ iterations, and the verifier accepts, provided that the third condition is satisfied. Note that for each fixed $j$, the strings $y_j^1, \dots y_j^m$ are independent and uniformly distributed elements of $\{0,1\}^{n'}$, and each one has probability $p$ of being a YES instance. Overall, using Chernoff bounds, there is probability at least $1/4k$ that at least $pm - O(\sqrt{m \log k})$ of them are YES instances, and this number is larger is $pm - m/2k$ if $m \geq c(k \log k)^2$ for a sufficiently large $c$. Then there is a probability at least $3/4$ that the condition is true for all $j$.

If $x \in L$, then the verifier rejects if the prover sends valid answers for one of the $m$ runs of $R(x)$, so, if the verifier accepts, it must be because the prover sent a set of answers containing at least $m$ errors. All the erroneous answers of the prover must be YES instances for which it answers NO (if the prover tries to cheat in the other way, it would be caught because of its inability to provide a valid certificate). This means that there is some $j$ such that, among the queries $y_j^1, \dots y_j^m$, at least $m/k$ are answered NO even though they were YES instances. By the calculation in the above paragraph, very likely there were no more than $pm + O(\sqrt{m \log k})$ yes instances to start with among $y_j^1, \dots y_j^m$, and so the prover has to pretend that there are only $pm + O(\sqrt{m \log k}) - k/m$, which is less than $pm - k/2m$ by our choice of $m$, and so the verifier, with high probability, rejects when checking condition (3). $\Box$

## 13.5 References

The classes **IP** and **AM** were defined independently. The class **IP** was defined by Shafi Goldwasser, Silvio Micali and Charles Rackoff [GMR89]. They proposed the class **IP** as an intermediate step towards the definition of Zero Knowledge Proofs. The paper [GMR89] was written in 1983 but appeared first in FOCS 85. The class **AM** was defined by Lazlo Babai [Bab85] (the journal version of the paper is by Babai and Moran [BM88]), in order to characterize the complexity of a group-theoretic problem. The paper also appeared in FOCS 85, and contained theorem 62. The interactive proof for graph non-isomorphism is from [GMW91]. Theorem 61 is due to Goldreich and Sipser [GS86]. Credit for the proof that **IP** = **PSPACE** is typically shared between the work of Lund et al. [LFKN92] and of Shamir [Sha92].

The results of Section 13.4 are due to Feigenbaum and Fortnow [FF93].

## Exercises

The following exercises give a proof of Theorem 66.

1. A non-deterministic circuit $C$ has two inputs $x = x_1, \ldots, l_m$ and $y = y_1, \ldots, y_n$. $C$ accepts the string $x$ iff $\exists y. C(x, y) = 1$. Show that every problem in **MA** has non-deterministic circuits of polynomial size. Sligthly harder: show that every problem in **AM**[2] has non-deterministic circuits of polynomial size.

   [*Hint:* The proof is similar to the proof that **BPP** has polynomial sized circuits.]

2. Prove that if **coNP** has polynomial-size non-deterministic circuits, then the polynomial hierarchy collapses.

   [*Hint:* This is not too different from Karp-Lipton.]

# Lecture 14

# IP=PSPACE

In this lecture we will see that **IP** = **PSPACE**. We start by giving an interactive proof system for UNSAT, and therefore show that **coNP** $\subseteq$ **IP**. In fact, as we will see, the same proof system with minor modifications can be used for #SAT. Since #SAT is #**P**-complete, and since **PH** $\subseteq$ $\mathbf{P}^{\#\mathbf{P}}$, then **PH** $\subseteq$ **IP**. Finally, we generalize this protocol in order to solve QSAT, and thus show that **PSPACE** $\subseteq$ **IP**.

## 14.1 UNSAT $\subseteq$ IP

Let $\phi$ be a formula with $m$ clauses over the variables $x_1, \ldots, x_n$. Let $N > 2^n \cdot 3^m$ be a prime number. We translate $\phi$ to a polynomial $p$ over the field $\pmod N$ in the following way. Literals are translated as follows: $x_i \to x_i$, $\overline{x_i} \to (1 - x_i)$. A clause is translated to the sum of the (at most 3) expressions corresponding to the literals of the clause. Finally, $p$ is taken as the product of the $m$ expressions corresponding to the $m$ clauses. Notice that each literal is of degree 1, and therefore $p$ is of degree at most $m$. Furthermore, for a 0-1 assignment, $p$ evaluates to 0 if this assignment does not satisfy $\phi$, and to a non-zero number if this assignment does satisfy $\phi$ where this number can be at most $3^m$. Hence, $\phi$ is unsatisfiable iff

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \ldots, x_n) \equiv 0 \pmod N$$

Also, the translation to $p$ can be done in polynomial time, and if we take $N = 2^{O(n+m)}$ then elements in the field can be represented by $O(n+m)$ bits, and thus one evaluation of $p$ in any point in $\{0, \ldots, N-1\}^m$ can be computed in polynomial time.

We now describe a protocol that enables the prover to convince the verifier that $\phi$ is unsatisfiable (see also Figure 14.1 for a sketch of the protocol). At the first round, the prover sends $N$, a proof that it is prime, and an $m$-degree univariate polynomial $q_1(x) = \sum_{x_2, \ldots, x_n \in \{0,1\}} p(x, x_2, \ldots, x_n)$. At this point, the verifier verifies the primality proof as well as checking that $q_1(0) + q_1(1) = 0$. If any of the checks fail then the verifier rejects. Otherwise, it continues by choosing a random $r_1 \in \{0, \ldots, N-1\}$ and sending it to the prover. The prover responds by sending an $m$-degree univariate polynomial $q_2(x) = \sum_{x_3, \ldots, x_n \in \{0,1\}} p(r_1, x, x_3, \ldots, x_n)$. Now, the verifier checks if $q_2(0) + q_2(1) = q_1(r_1)$. If not, it rejects, and otherwise it continues by choosing a random $r_2 \in \{0, \ldots, N-1\}$ and

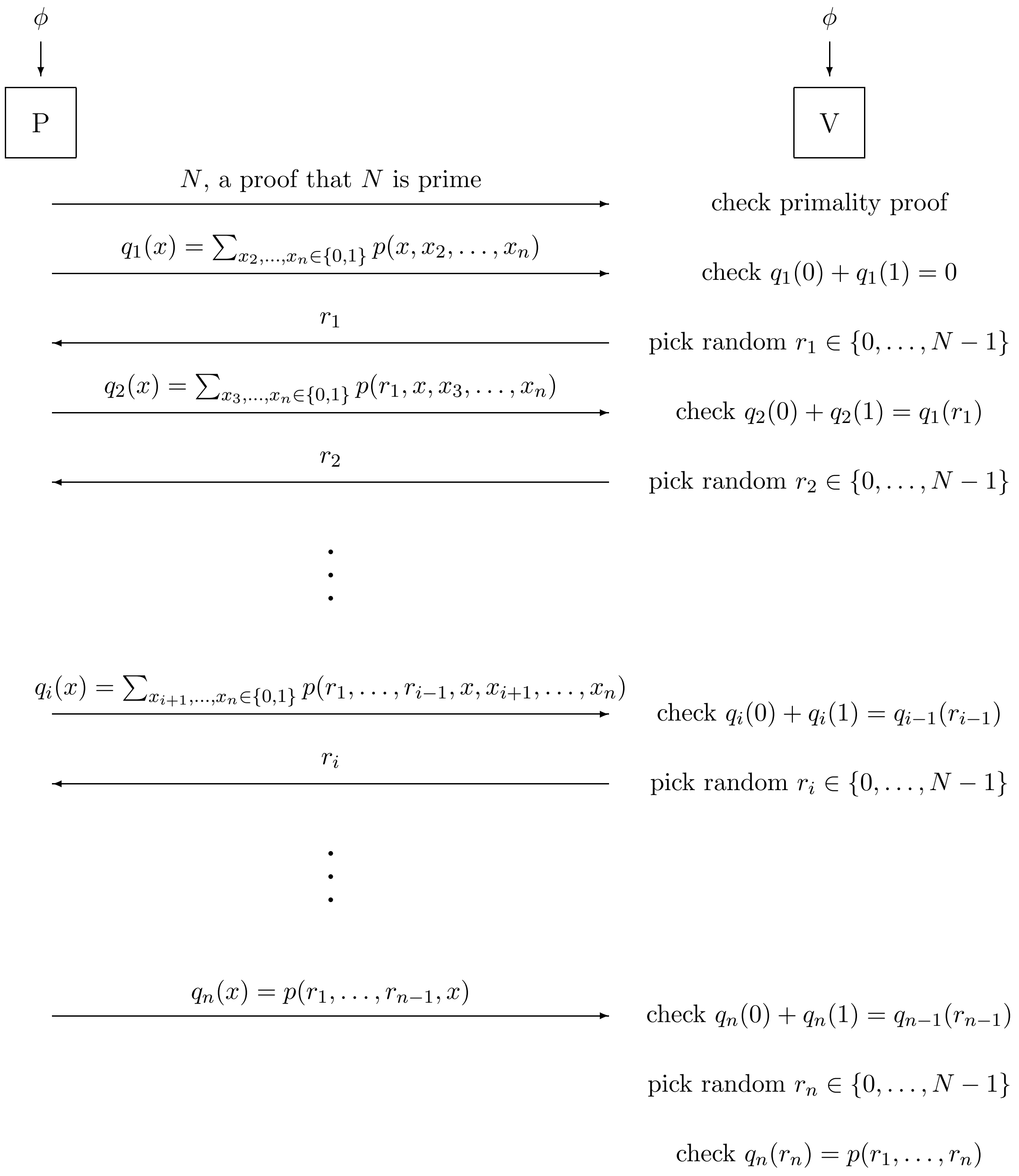

$\phi$
$\phi$
P
V
$N$, a proof that $N$ is prime
check primality proof
$q_1(x) = \sum_{x_2,\dots,x_n \in \{0,1\}} p(x, x_2, \dots, x_n)$
check $q_1(0) + q_1(1) = 0$
$r_1$
pick random $r_1 \in \{0, \dots, N-1\}$
$q_2(x) = \sum_{x_3,\dots,x_n \in \{0,1\}} p(r_1, x, x_3, \dots, x_n)$
check $q_2(0) + q_2(1) = q_1(r_1)$
$r_2$
pick random $r_2 \in \{0, \dots, N-1\}$
$q_i(x) = \sum_{x_{i+1},\dots,x_n \in \{0,1\}} p(r_1, \dots, r_{i-1}, x, x_{i+1}, \dots, x_n)$
check $q_i(0) + q_i(1) = q_{i-1}(r_{i-1})$
$r_i$
pick random $r_i \in \{0, \dots, N-1\}$
$q_n(x) = p(r_1, \dots, r_{n-1}, x)$
check $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$
pick random $r_n \in \{0, \dots, N-1\}$
check $q_n(r_n) = p(r_1, \dots, r_n)$

Figure 14.1: The protocol for proving unsatisfiability

sending it to the prover. The protocol continues in this way where the $i$th message that the prover sends is a polynomial $q_i(x) = \sum_{x_{i+1},\dots,x_n\in\{0,1\}} p(r_1,\dots,r_{i-1},x,x_{i+1},\dots,x_n)$. The verifier responds by checking that $q_i(0) + q_i(1) = q_{i-1}(r_{i-1})$, rejecting if it is not, and otherwise, choosing a random $r_i \in \{0,\dots,N-1\}$ and sending it to the prover. At the last round, the prover sends $q_n(x) = p(r_1,\dots,r_{n-1},x)$. The verifier checks that $q_n(0) + q_n(1) = q_{n-1}(r_{n-1})$, chooses a random $r_n \in \{0,\dots,N-1\}$, and checks that $q_n(r_n) = p(r_1,\dots,r_n)$. The verifier rejects if either of the checks fail, and accepts otherwise. The verifier can indeed perform the last check since as mentioned before, it can translate $\phi$ to $p$ and evaluate it at the point $(r_1,\dots,r_n)$ in polynomial time. Also, the messages are of polynomial length because elements of the field can be written with $O(m+n)$ bits, and since each polynomial has $m$ coefficients.

We now show the correctness of this proof system. If $\phi$ is unsatisfiable then the prover can make the verifier accept w.p. 1 by just following the protocol since $q_1(0) + q_1(1) = \sum_{x_1,\dots,x_n\in\{0,1\}} p(x_1,\dots,x_n) \equiv 0 \pmod N$. It is clear that the rest of the checks will succeed if the prover send the $q_i$'s accordingly. We have to show that if $\phi$ is satisfiable then no matter what polynomials the prover sends, the verifier will reject with high probability. We write $p_i(x)$ for the polynomial that a prover who follows the protocol sends in round $i$. Since $\phi$ is satisfiable then $\sum_{x_1,\dots,x_n\in\{0,1\}} p(x_1,\dots,x_n) \not\equiv 0 \pmod N$. Therefore, $p_1(0) + p_1(1) \neq 0$, and hence, if the prover sends $q_1 = p_1$ then the verifier will reject. If the prover sends $q_1 \neq p_1$ then since both are degree-$m$ polynomials then they agree on at most $m$ places. Thus, there is probability $\geq 1 - \frac{m}{N}$ that $p_1(r_1) \neq q_1(r_1)$. Suppose that indeed $p_1(r_1) \neq q_1(r_1)$. If the prover then sends $q_2 = p_2$ then the verifier will reject because $q_2(0)+q_2(1) = p_1(r_1) \neq q_1(r_1)$. Thus, the prover must send $q_2 \neq p_2$. Again, in that case, $q_2$ and $p_2$ will differ on a fraction $\geq 1 - \frac{m}{N}$ of the elements of $\{0,\dots,N-1\}$, so $p_2(r_2) \neq q_2(r_2)$ with probability $\geq 1 - \frac{m}{N}$. We continue the argument in a similar way. If $q_i \neq p_i$ then w.p. $\geq 1 - \frac{m}{N}$, $r_i$ is such that $p_i(r_i) \neq q_i(r_i)$. If so, then the prover must send $q_{i+1} \neq p_{i+1}$ in order for the verifier not to reject. At the end, if the verifier has not rejected before the last check, then w.p. $\geq 1 - (n-1)\frac{m}{N}$, $q_n \neq p_n$. If so, then w.p. $\geq 1 - \frac{m}{N}$ the verifier will reject since, again, $q_n(x)$ and $p(r_1,\dots,r_{n-1},x)$ differ on at least that fraction of the points. Thus, the total probability that the verifier will accept is at most $\frac{nm}{N}$.

## 14.2 A Proof System for #SAT

We now show that with a few minor modifications, the previous proof system can be used to prove that a formula has a given number of satisfying assignments. Suppose $\phi$ has $k$ satisfying assignments. We wish to give a protocol s.t. if the prover gives $k$ as an answer, he is able to continue with the protocol s.t. the verifier will accept w.p. 1, but if it gives another number as an answer, then no matter what messages the prover sends afterwards, the verifier will reject with high probability.

We first change the way we translate $\phi$ to the polynomial $p$. The change is only in the way we translate clauses. Instead of translating a clause to the sum of the literals, we translate the clause $(z_1 \vee z_2 \vee z_3)$ to $1-(1-z_1)(1-z_2)(1-z_3)$. Notice that if a 0-1 assignment to the variables satisfies the clause, then the value of this expression will be 1, and if the assignment does not satisfy the clause, this value will be 0. Hence, 0-1 assignments that satisfy $\phi$ will be evaluated to 1 by $p$, while as before, the 0-1 assignments that do not satisfy

$\phi$ will be evaluated to 0. Notice that now the degree of $p$ is $3m$ instead of $m$, but now the sum over all evaluations of 0-1 assignments is equal to the number of satisfying assignments of $\phi$. For the same reason, it is now enough to take $N > 2^n$.

In the current protocol, the prover starts by sending $k$, and then continues as in the previous protocol (with the updated definition of $p$). After receiving the first message, the verifier checks if $q_1(0) + q_1(1) = k$ instead of checking $q_1(0) + q_1(1) = 0$, and follows the rest of the previous protocol. If $k$ is indeed the correct number of satisfying assignments, then the prover can continue by sending $q_i = p_i$ at each round, and the verifier will accept with probability 1. If $k$ is not the correct number of satisfying assignments, then according to the same analysis as in the previous section, the verifier will accept with probability at most $\frac{3mn}{N}$.

## 14.3 A Proof System for QBF

### 14.3.1 PSPACE-Complete Language: TQBF

For a 3-$CNF$ boolean formula $\phi(x_1, x_2, \ldots, x_n)$, we may think of its satisfiability problem as determining the truth value of the statement

$$\exists x_1 \exists x_2, \ldots, \exists x_n \quad \phi(x_1, x_2, \ldots, x_n).$$

We can generalize this idea to allow universal quantifiers in additional to existential quantifiers to get formulas denoted as quantified boolean formulas. For example, $\forall x_1 \exists x_2 \quad (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$ is a quantified boolean formula. In fact, it's easy to see that it's a true quantified boolean formula. Now consider the language of all true quantified boolean formulas:

$$\text{TQBF} = \{\Phi : \quad \Phi \text{ is a true quantified boolean formula}\}.$$

It is known that TQBF is **PSPACE**-complete. Thus, if we have an interactive-proof protocol for recognizing TQBF, then we have a protocol for recognizing any language in **PSPACE**. In the rest of the lecture, we will provide a protocol for recognizing TQBF.

### 14.3.2 Arithmetization of TQBF

We will consider the following quantified boolean formula:

$$\Phi = \forall x_1 \exists x_2 \forall x_3 \ldots \forall x_n \quad \phi(x_1, x_2, \ldots, x_n),$$

where $\phi(x_1, x_2, \ldots, x_n)$ is a $3 - CNF$ boolean formula. Without loss of generality, we can assume that all the quantified boolean formulas given to us have this form. The main idea in the interactive protocol for recognizing TQBFis the same as proving $\sharp\mathbf{P} \subseteq \mathbf{IP}$ from the last lecture: we will first arithmetize the quantified boolean formula and then the prover will convince the verifier that the arithmetized expression evaluates to 1. In the following, all the random elements are drawn from a field $\{0, 1, \ldots, p-1\}$, where $p$ is sufficiently large.

**Naive Solution**

So how do we arithmetize the quantified formula? we will first show a naive solution and point out its problems. To begin with, we know from the last lecture how to arithmetize $\phi(x_1, x_2, \ldots, x_n)$ to obtain the polynomial $F(x_1, x_2, \ldots, x_n)$. Recall that $F$ has degree $3m$ in each variable ($m$ is the number of clauses in $\phi$) and agrees with $\phi$ for all $0-1$ assignments to the $n$ variables. Now we read the quantifiers from right to left and consider the expression $\forall x_n \phi(x_1, x_2, \ldots, x_n)$. This expression has $n-1$ free variables, and for each substitution of values to these variables the expression itself either evaluates to true or false. We're looking for a polynomial that has the same behavior. Using $F$ we can now write a new polynomial $P_{\forall x_n} F(x_1, x_2, \ldots, x_n)$, which is equal to

$$F(x_1, x_2, \ldots, x_{n-1}, 0) \cdot F(x_1, x_2, \ldots, x_{n-1}, 1).$$

Next consider the previous quantifier $\exists x_{n-1}$. We want to find a polynomial representation for $\exists x_{n-1} \forall x_n \phi(x_1, x_2, \ldots, x_n)$, given the polynomial representation $G(x_1, x_2, \ldots, x_{n-1})$ for $\forall x_n \phi(x_1, x_2, \ldots, x_n)$. The following polynomial, which we denote by $P_{\exists x_{n-1}} G(x_1, x_2, \ldots, x_{n-1})$, satisfies the condition:

$$1 - (1 - G(x_1, x_2, \ldots, x_{n-2}, 0)) \cdot (1 - G(x_1, x_2, \ldots, x_{n-2}, 1)).$$

In the notations introduced above, the polynomial for $\exists x_{n-1} \forall x_n \phi(x_1, x_2, \ldots, x_n)$ is

$$P_{\exists x_{n-1}} P_{\forall x_n} F(x_1, x_2, \ldots, x_n).$$

In general, we transform the quantified boolean formula $\Phi$ into an arithmetic expression using operators $P_{\forall u}$ and $P_{\exists v}$ as follows:

- Turn the 3-$CNF$ formula $\phi(x_1, x_2, \ldots, x_n)$ into the polynomial $F(x_1, x_2, \ldots, x_n)$ as done in the last lecture.
- Replace the quantifier $\exists x_i$ by the operator $P_{\exists x_i}$.
- Replace the quantifier $\forall x_i$ by the operator $P_{\forall x_i}$.

The final expression after the transformation always evaluates to 0 or 1. It evaluates to 1 if and only if the quantified boolean formula $\Phi$ is true.

We have thus arrived at a way of representing quantified boolean formulas using arithmetic operations. Now consider the arithmetic expression after the transformation

$$P_{\forall x_1} P_{\exists x_2} \ldots P_{\forall x_n} F(x_1, x_2, \ldots, x_n)$$

and it evaluates to either 0 or 1. Let us apply the idea from the last lecture to derive a protocol and see what happens. In the first step, the prover eliminates the first operator $P_{\forall x_1}$ and sends the verifier the polynomial $G_1(x_1)$ corresponding to the rest of the arithmetic expression:

$$G_1(x_1) = P_{\exists x_2} P_{\forall x_3} \ldots P_{\forall x_n} F(x_1, x_2, \ldots, x_n).$$

Upon receiving the polynomial $\hat{G_1}(x_1)$ (which the prover claims to be $G_1(x_1)$), the verifier checks that $\hat{G_1}(0) \cdot \hat{G_1}(1) = 1$. The verifier then picks a random value $r_1$, computes $\beta_1 = \hat{G_1}(r_1)$ and sends $r_1$ to the prover. The prover then needs to convince the verifier that

$$\beta_1 = P_{\exists x_2} P_{\forall x_3} \dots P_{\forall x_n} F(r_1, x_2, \dots, x_n).$$

The prover then proceeds by eliminating the operator one after another, and in the end the verifier checks that $\beta_n = F(r_1, r_2, \dots, r_n)$.

However, since each operator potentially doubles the degree of each variable in the expression, the degree of the polynomial $G_1(x_1)$ in the end can very well be exponential. This means that the prover will have to send the verifier exponentially many coefficients, but the polynomially bounded verifier will not be able to read them all.

**Revised Solution**

We will ensure that the degree of any variable in any intermediate stage of the transformation never goes above two. The arithmetization given above clearly does not satisfy this property. Yet there is a simple way to fix this. At any stage of the transformation, we have a polynomial $J(x_1, x_2, \dots, x_n)$ where some variables' degree might exceed two. Normally, we can't expect to transform $J$ into a new polynomial $J'(x_1, x_2, \dots, x_n)$ where the degree of any variable is at most two while still evaluating to the same value at all points. Yet, notice here that we only need $J'$ to agree with $J$ on all $0-1$ assignments. What $J'$ does at any other point is of no interests to us. The key observation then is that for $x = 0$ or $x = 1$, $x^k = x$ for all positive integers $k$. The desired polynomial $J'$ can thus be obtained from $J$ by erasing all exponents, that is, replacing them with 1. For example, if $J(x_1, x_2, x_3) = x_1^3 x_2^4 + 5x_1x_2^3 + x_2^6 x_3^2$, the transformed polynomial $J'$ is $6x_1x_2 + x_2x_3$. In general, we define a new operator, $Rx_i$, which when applied to a polynomial reduces the exponents of $x_i$ to 1 at all occurrences. More formally, we have

$$\begin{aligned} Rx_i J(x_1, \dots, x_n) = {} & x_i \cdot J(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) + \\ & (1 - x_i) \cdot J(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n). \end{aligned}$$

In this notation, we have

$$J'(x_1, \dots, x_n) = Rx_1 Rx_2 \dots Rx_n J(x_1, \dots, x_n).$$

In the revised arithmetization process, the only change we make is to apply the reduce operators whenever the degree of any variable can potentially be above 1 in the polynomial obtained from the previous step. In particular, we apply the reduce operators after the 3-$CNF$ formula $\phi$ is arithmetized into the polynomial $F(x_1, \dots, x_n)$. Also, since the other two operators, $P_{\forall u}$ and $P_{\forall v}$, might double the degree of some variables in the transformation, we apply the reduce operators after each application of $P_{\forall u}$ or $P_{\exists v}$. This revision will ensure that the degree of any variable at any intermediate stage never goes above 2. The degree of some variable might reach 2 after a $P_{\forall u}$ or $P_{\exists v}$ operator, but will be reduced back to 1 after the reduce operators are applied.

### 14.3.3 The Interactive Protocol

Using the revised arithmetization, we arithmetize the quantified boolean formula of the form

$$\Phi = \forall x_1 \exists x_2 \forall x_3 \ldots \forall x_n \phi(x_1, \ldots, x_n)$$

into

$$E = P_{\forall x_1} Rx_1 P_{\exists x_2} Rx_1 Rx_2 P_{\forall x_3} Rx_1 Rx_2 Rx_3 P_{\exists x_4} \ldots P_{\forall x_n} Rx_1 \ldots Rx_n F(x_1, \ldots, x_n).$$

Now the idea of the interactive protocol from the last lecture can be put to use without any trouble. The prover and the verifier communicate to eliminate one operator at each stage. In the end, no operators are left and the verifier checks the required condition.

Initially, the prover eliminates the first operator $P_{\forall x_1}$ and sends the verifier the polynomial $G_1(x_1)$ corresponding to the rest of the arithmetic expression:

$$G_1(x_1) = Rx_1 P_{\exists x_2} \ldots Rx_n F(x_1, \ldots, x_n).$$

Notice that this is a polynomial of degree 1 instead of exponential degree in the original naive arithmetization. Upon receiving the polynomial $\hat{G_1}(x_1)$(which the prover claims to be $G_1(x_1)$), the verifier checks that $P_{\forall x_1} \hat{G_1}(x_1) = \hat{G_1}(0) \cdot \hat{G_1}(1) = 1$. The verifier then picks a random value $r_1$, computes $\beta_1 = \hat{G_1}(r_1)$ and sends $r_1$ to the prover. (In this stage, the verifier actually doesn't have to send $r_1$ to the prover, but for consistency with later rounds we let the verifier send $r_1$ to the prover. The soundness of the protocol remains.) The prover then needs to convince the verifier that

$$\beta_1 = Rx_1 P_{\exists x_2} \ldots Rx_n F(r_1, x_2, \ldots, x_n).$$

The prover then eliminates the second operator $Rx_1$ and sends the verifier the polynomial $G_2(x_1)$ corresponding to the rest of the arithmetic expression:

$$G_2(x_1) = P_{\exists x_2} \ldots Rx_n F(x_1, \ldots, x_n).$$

Notice that this is a polynomial of degree at most 2. Upon receiving the polynomial $\hat{G_2}(x_1)$(which the prover claims to be $G_2(x_1)$), the verifier checks that $(Rx_1 \hat{G_2}(x_1))[r_1] = \hat{G_1}(r_1)$, where $[r_1]$ means that the polynomial in the parenthesis with free variable $x_1$ is evaluated at point $r_1$. The verifier then picks a random value $r_2$ for $x_1$, computes $\beta_2 = \hat{G_1}(r_2)$ and sends $r_2$ to the prover.

In general, when the verifier needs to be convinced that

$$\beta = (Rx_i P(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, x_i))[\alpha_i],$$

he asks for the polynomial $G(x_i) = P(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, x_i)$, and gets $\hat{G}(x_i)$. ($\alpha_j (1 \leq j \leq i)$ is the most recent random value the verifier assigned to variable $x_j$.) He verifies that $(Rx_i \hat{G}(x_i))[\alpha_i] = \beta$ and computes $\beta' = \hat{G}(\alpha_i')$ by choosing a new random value $\alpha_i'$ for $x_i$. Now the verification is reduced to whether

$$\beta' = P(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \alpha_i').$$

The case of the $P_{\forall u}$ or $P_{\exists v}$ operator is the same as in the last lecture where we proved $\sharp\mathbf{P} \subseteq \mathbf{IP}$.

The prover proceeds by eliminating the operators one after another, and in the end the verifier checks that $q(\alpha_n) = F(\alpha_1, \ldots, \alpha_n)$, where $q$ is the polynomial the prover sends to the verifier in the last round and $\alpha_i$ is the most recent random value the verifier assigned to variable $x_i$.

### 14.3.4 Analysis

**Theorem 68** *(a) If $\Phi \in TQBF$, then there exists a prover such that $Pr[Verifier\ accepts] = 1$.*
*(b) If $\Phi \notin TQBF$, then for all provers $Pr[Verifier\ accepts] \leq 1/2$.*

Proof: Part (a) is obvious. For part (b), the analysis is the same as in the case of $\sharp\mathbf{P} \subseteq \mathbf{IP}$ from the last lecture. The error can be introduced only when the verifier happens to pick a root of some low degree polynomial at some round. While dealing with the last $n$ reduce operators, the polynomials may have degree at most $3m$. For the remaining $n(n-1)/2$ reduce operators, the degree is at most 2. For every other operator ($n$ operators of the form $P_{\forall u}$ or $P_{\exists v}$) the degree of the polynomial is 1. Thus, the sum of the degrees of all the polynomials encountered in the interaction is at most $3mn + n^2$. Thus, if we choose the size of our field $p$ large enough, the probability of error is at most $(3mn + n^2)/p \leq 1/2$. $\Box$

## Exercises

1. Show that **IP** $\subseteq$ **PSPACE**.

   [Hint: it is easier to show the following stronger claim: that given a verifier $V$ and an input $x$, we can compute the maximum, over all provers' strategies $P^*$, of the probability that $V$ accepts $x$ while interacting with $P^*$.]

# Lecture 15

# Introduction to PCP

In this lecture we see the connection between **PCP** and hardness of approximation. In particular, we show that the **PCP** Theorem implies that MAX-3SAT cannot be approximated within $(1-\epsilon)$ for some $\epsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$. Also, we show that Max Clique cannot be approximated within any constant factor.

## 15.1 Probabilistically Checkable Proofs

**Definition 26** *Given functions $r(n)$, $q(n)$, we say that $L \in \mathbf{PCP}(r(n), q(n))$ if there is a polynomial time probabilistic verifier $V$, which is given $x$, oracle access to a proof $\pi$, uses $r(|x|)$ random bits, reads $q(|x|)$ bits of $\pi$ and such that*

$$x \in L \Rightarrow \exists \pi \cdot \mathbf{Pr}[V^{\pi}(x) = 1] = 1$$
$$x \notin L \Rightarrow \forall \pi \cdot \mathbf{Pr}[V^{\pi}(x) = 1] \le \tfrac{1}{2}.$$

**Theorem 69 (PCP Theorem)** $\mathbf{NP} = \mathbf{PCP}(O(\log n), O(1))$.

## 15.2 PCP and MAX-3SAT

### 15.2.1 Approximability

**Definition 27 (MAX-3SAT)** *Given a 3CNF formula $\varphi$ (i.e. with at most 3 variables per clause), find an assignment that satisfies the largest number of clauses.*

Note that MAX-3SAT generalizes 3SAT, and so cannot be solved in polynomial time unless $\mathbf{P} = \mathbf{NP}$. However, it is easy to approximate it within a factor of 2:

*Algorithm.* Output the best solution between all-true and all-false.

*Analysis.* Every clause is satisfied by at least one of the two solutions, therefore one of the solutions satisfies at least half of the clauses.

It is possible to do better. The Karloff-Zwick algorithm runs in polynomial time and satisfies $\ge \frac{7}{8}$ times the optimal number of clauses.

Some problems admit polynomial time algorithms with approximation ratio $\ge (1-\epsilon)$ $\forall \epsilon$. The **PCP** Theorem implies this is not the case for MAX-3SAT, as we see next.

### 15.2.2 Inapproximability

**Theorem 70** *The* **PCP** *Theorem implies that there is an $\epsilon > 0$ such that $(1-\epsilon)$-approximation of MAX-3SAT is* **NP***-hard.*

PROOF: Fix any **NP**-complete problem $L$. By the **PCP** Theorem, $L \in \mathbf{PCP}(O(\log n), O(1))$. Let $V$ be the verifier for $L$.

Given an instance $x$ of $L$, our plan is to construct a 3CNF formula $\varphi_x$ on $m$ variables such that (for some $\epsilon > 0$ to be determined)

$$\begin{array}{lcl} x \in L & \Rightarrow & \varphi_x \text{ is satisfiable} \\ x \notin L & \Rightarrow & \text{no more than } (1-\epsilon)m \text{ clauses of } \varphi_x \text{ are satisfiable.} \end{array} \tag{15.1}$$

Without loss of generality, assume that $V$ makes non-adaptive queries (by reading all the bits that could possibly be needed at once). This assumption is valid because the number of queries was a constant and remains a constant. Let $q$ be the number of queries.

Enumerate all random strings $R$ for $V$. The length of each string is $r(|x|) = O(\log |x|)$, so the number of such strings is polynomial in $|x|$. For each $R_i$, $V$ chooses $q$ positions $i_1, \ldots, i_q$ and a Boolean function $f_R : \{0,1\}^q \to \{0,1\}$ and accepts iff $f_R(\pi(i_1), \ldots, \pi(i_q))$.

We want a Boolean formula to simulate this. Introduce Boolean variables $x_1, \ldots, x_l$, where $l$ is the length of the proof. Now we need a correspondence between the number of satisfiable clauses and the probability that the verifier accepts.

For every $R$ we add clauses that represent $f_R(x_{i_1}, \ldots, x_{i_q})$. This can be done with $2^q$ SATclauses. We need to convert clauses of length $q$ to length 3 with additional variables. e.g. $x_2 \vee x_{10} \vee x_{11} \vee x_{12}$ becomes $(x_2 \vee x_{10} \vee y_R) \wedge (\overline{y}_R \vee x_{11} \vee x_{12})$. This requires at most $q2^q$ 3SATclauses. In general there is a series of auxiliary variables for each $R$.

If $z \in L$, then there is a proof $\pi$ such that $V^\pi(z)$ accepts for every $R$. Set $x_i = \pi(i)$ and auxiliary variables in the right way, all clauses are satisfied.

If $z \notin L$, then for every assignment to $x_1, \ldots, x_l$ (and to $y_R$'s), the corresponding proof $\pi(i) = x_i$ makes the verifier reject for half of the $R \in \{0,1\}^{r(|z|)}$. For each such $R$, one of the clauses representing $f_R$ fails. Therefore a fraction $\epsilon = \frac{1}{2}\frac{1}{q2^q}$ of clauses fails. $\Box$

It turns out if this holds for every **NP**-complete problem, then the **PCP** Theorem must be true.

**Theorem 71** *If there is a reduction as in (15.1) for some problem $L$ in* **NP***, then $L \in$* $\mathbf{PCP}(O(\log n), O(1))$*, i.e. the* **PCP** *Theorem holds for that problem.*

PROOF: We describe how to construct a verifier for $L$. $V$ on input $z$ expects $\pi$ to be a satisfying assignment for $\varphi_z$. $V$ picks $O(\frac{1}{\epsilon})$ clauses of $\varphi_z$ at random, and checks that $\pi$ satisfies all of them. Randomness is $O(\frac{1}{\epsilon}\log m) = O(\log |z|)$. $O(\frac{1}{\epsilon}) = O(1)$ bits are read in the proof.

$$\begin{aligned} z \in L &\Rightarrow \varphi_z \text{ is satisfiable} \\ &\Rightarrow \exists \pi \text{ such that } V_\pi(z) \text{ always accept.} \end{aligned}$$

$$\begin{aligned} z \notin L &\Rightarrow \forall \pi \text{ a fraction } \tfrac{1}{\epsilon} \text{ of clauses are unsatisfied} \\ &\Rightarrow \forall \pi\ V^\pi(z) \text{ will reject with probability } \geq \tfrac{1}{2} \quad \text{(details omitted).} \end{aligned}$$

□

### 15.2.3 Tighter result

What is the best known value of $\epsilon$ in Theorem 70? In the next lectures we will prove the following result:

**Theorem 72 (Håstad)** *There is a* **PCP** *verifier for* **NP***that uses* $O(\log n)$ *bits of randomness which, based on* $R$*, chooses* 3 *positions* $i_1, i_2, i_3$ *and a bit* $b$ *and accepts iff* $\pi(i_1) \oplus \pi(i_2) \oplus \pi(i_3) = b$*. The verifier satisfies*

$$z \in L \Rightarrow \exists \pi \cdot \mathbf{Pr}[V^\pi(x) = 1] \geq 1 - \epsilon \qquad (15.2)$$

$$z \notin L \Rightarrow \forall \pi \cdot \mathbf{Pr}[V^\pi(x) = 1] \leq \tfrac{1}{2} + \epsilon \qquad (15.3)$$

Note the slightly non-standard bounds for the probabilities above. If we had "$= 1$" in Equation (15.2), we could find a proof $\pi$ by solving a system of linear equations (which would imply $\mathbf{P} = \mathbf{NP}$). For every $R$, one can encode $x_{i_1} \oplus x_{i_2} \oplus x_{i_3} = b$ with 4 clauses in 3CNF.

If $z \in L$, then a fraction $\geq (1 - \epsilon)$ of clauses are satisfied.

If $z \notin L$, then for a $(\frac{1}{2} - \epsilon)$ fraction of $R$, $\frac{1}{4}$ of clauses are contradicted.

This is not producing a formula directly, but is enough to prove the hardness of approximation ratio:

$$\frac{1 - \frac{1}{4}(\frac{1}{2} - \epsilon)}{1 - \epsilon} = \frac{7}{8} + \epsilon'$$

## 15.3 Max Clique

### 15.3.1 Approximability

**Definition 28 (Max Clique)** *Given an undirected graph* $G = (V, E)$*, find the largest* $C \subseteq V$ *such that* $\forall u, v \in C$*,* $(u, v) \in E$*. When convenient, we use the complementary graph equivalent "Max Independent Set" which is the largest* $I \subseteq V$ *such that* $\forall u, v \in I$*,* $(u, v) \notin E$*.*

Finding a largest clique is **NP**-hard, so we ask about approximations. There is a simple $\frac{\log n}{n}$-approximation:

- arbitrarily partition the graph into subsets of size $\log n$
- solve exactly each subset
- report the largest clique found

Suppose there is a clique of size $\alpha n$, then one of the subset contains a clique of size $\alpha \log n$, hence the approximation. Best known is a $\frac{(\log n)^2}{n}$-approximation algorithm.

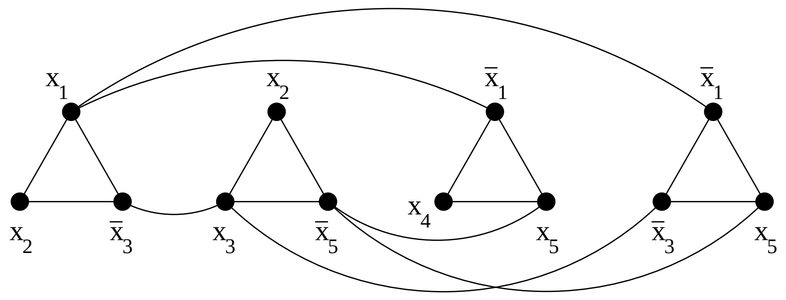


Figure 15.1: Graph construction corresponding to the 3CNF formula $\varphi = (x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_2 \vee x_3 \vee \overline{x}_5) \wedge (\overline{x}_1 \vee x_4 \vee x_5) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_5)$.

### 15.3.2 Inapproximability

We apply the **PCP** Theorem to Max Independent Set. Given a 3CNF formula $\varphi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ over variables $x_1, \ldots, x_n$ arising from Theorem 70, we build a graph using the same construction as in the **NP**-completeness proof of Independent Set in Lecture 3 (see Figure 15.1).

The graph has $3m$ vertices. Note that G has an independent set of size $m$ iff $\varphi$ is satisfiable, and G has an independent set with $k$ vertices iff $\varphi$ has an assignment that satisfies $k$ clauses. So the hardness results for MAX-3SAT translate directly to Max Independent Set. Below we show how this can be further improved.

Given $G = (V, E)$, consider $G' = (V', E')$ where $V' = V \times V$ and $((u_1, u_2), (v_1, v_2)) \in E'$ iff $(u_1, v_1) \in E$ OR $(u_2, v_2) \in E$. It can be shown that the size of the maximum independent set in $G'$ is equal to (size of max ind. set in $G$)$^2$. As a sidenote, this result is not true for cliques (a clique in $G'$ could be possibly larger than expected), unless we use a different definition of $G'$.

By a constant number of applications of the graph product operation, we get:

| graph | $G$ | $G'$ | $G''$ |
|---|---|---|---|
| size of ind. set if $\varphi$ is satisfiable | $m$ | $m^2$ | $m^4$ |
| size of ind. set if only $(1-\epsilon)m$ clauses of $\varphi$ are satisfiable | $(1-\epsilon)m$ | $(1-\epsilon)^2 m^2$ | $(1-\epsilon)^4 m^4$ |

The ratio decreases without bound, therefore there can be no constant approximation algorithm for Max Clique. In fact:

**Theorem 73 (Håstad)** *If* **NP** $\neq$ **ZPP**, *then* $\forall \epsilon > 0$, *Max Independent Set cannot be approximated within* $\frac{1}{n^{1-\epsilon}}$ *($n$ is the number of vertices).*

## 15.4 References

The PCP Theorem is from [AS98, ALM$^+$98], and the connection between proof-checking and approximability is due to [FGL$^+$96]. Karloff and Zwick [KZ97] describe a 7/8 approximation algorithm for Max 3SAT. Håstad's result on three-query PCP is from [Hås01]. Håstad's result on the approximability of the Max Clique problem is from [Hås99].

# Lecture 16

# Håstad's Proof System

In this lecture we introduce Håstad's **PCP** verifier, which yields stronger hardness results than the raw **PCP** theorem. In particular, we derive asymptotically optimal hardness of approximation results for Max E3LIN2 and Max 3SAT. We also show that Max 2SAT and Min Vertex Cover are hard to approximate within $21/22 + o(1)$ and $7/6 + o(1)$ of their optimum, respectively. Finally, we outline the connection between the **PCP** theorem and Håstad's verifier.

## 16.1 Håstad's verifier

To describe Håstad's verifier, we will need a more detailed parametrization of **PCP**. Say a language $L$ is in $\mathbf{PCP}_{c,s}[r(n), q(n)]$ if there is a probabilistic polynomial time verifier $V$ which, given input $x$ and oracle access to a proof $\pi$, uses $r(|x|)$ random bits, reads $q(|x|)$ bits of $\pi$ and such that

$$\begin{aligned} x \in L \quad &\Rightarrow \quad \exists \pi : \Pr[V^\pi(x) = 1] \geq c \\ x \notin L \quad &\Rightarrow \quad \forall \pi : \Pr[V^\pi(x) = 1] < s. \end{aligned}$$

The values $c$ and $s$ are called the *completeness* and the *soundness* of the verifier, respectively. If, in addition to these conditions, the verifier $V$ makes its queries nonadaptively, we say $L \in \mathbf{naPCP}_{c,s}[r(n), q(n)]$.

**Theorem 74 (Håstad)** *For every constant $\epsilon > 0$, $\mathbf{NP} = \mathbf{naPCP}_{1-\epsilon,1/2+\epsilon}[O(\log n), 3]$. Moreover, the verifier for any $L \in \mathbf{NP}$ is simple: $V^\pi$ generates three indices $i, j, k$ and one bit $b$ and accepts iff $\pi[i] \oplus \pi[j] \oplus \pi[k] = b$.*

We will sketch how to derive Håstad's theorem from the **PCP** theorem in the next few lectures. We remark that the theorem achieves the best possible values for its parameters in several ways. In particular, we have the following lower bounds:

1. Number of queries. (Folklore) For every $\epsilon > 0$, $\mathbf{P} = \mathbf{PCP}_{1,1-\epsilon}[O(\log n, 2)]$ and (Zwick) $\mathbf{P} = \mathbf{PCP}_{1-\epsilon,1-\epsilon^{1/3}}[O(\log n), 2]$.

2. Completeness vs. soundness. (Zwick) If $c/(1-s) \geq 2$, then $\mathbf{P} = \mathbf{PCP}_{c,1-s}[O(\log n), 3]$.

3. Completeness. (Zwick) $\mathbf{P} = \mathbf{naPCP}_{1,5/8}[O(\log n), 3]$.

However, perfect completeness can be bought by giving away soundness or nonadaptivity:

1. (Håstad) For all $\epsilon > 0$, $\mathbf{NP} = \mathbf{naPCP}_{1,3/4+\epsilon}[O(\log n), 3]$. It is not known whether this soundness constant is optimal for perfect completeness.

2. (Guruswami, Lewin, Sudan and Trevisan) For all $\epsilon > 0$, $\mathbf{NP} = \mathbf{PCP}_{1,1/2+\epsilon}[O(\log n), 3]$.

## 16.2 Applications to hardness of approximation

We show that Håstad's theorem implies the following hardness of approximation factors: $1/2 + o(1)$ for Max E3LIN2, $7/8 + o(1)$ for Max 3SAT, $21/22 + o(1)$ for Max 2SAT and $7/6 - o(1)$ for Min Vertex Cover. The proofs use approximation preserving reductions similar to the ones we saw last time.

**Linear equations modulo 2.** Max E3LIN2 is the problem of finding a maximum assignment to a linear system of equations modulo 2, with three distinct variables per equation. The value of an assignment is the number of equations satisfied by it. The problem of deciding whether a satisfying assignment of an E3LIN2 instance exists is $\mathbf{P}$—it can be solved by Gaussian elimination. Obtaining a 1/2-approximation for Max E3LIN2 is trivial: Consider the two assignments $x_i = 0$ for all $i$ and $x_i = 1$ for all $i$, and choose the one that satisfies more equations. This assignment will satisfy at least half of the equations.

Surprisingly, a 1/2 approximation is asymptotically optimal, unless $\mathbf{P} = \mathbf{NP}$. To show this, we construct a reduction $\psi$ from an arbitrary $\mathbf{NP}$ language $L$ to E3LIN2. Consider Håstad's verifier $V$ for $L$: On input $z$, $V^\pi$ generates a random string $r$ of length $R = O(\log |z|)$, computes three indices $i(r), j(r), k(r)$ and a bit $b(r)$, queries $\pi$ at positions $i, j, k$ and accepts iff $\pi[i(r)] \oplus \pi[j(r)] \oplus \pi[k(r)] = b(r)$. The reduction $\psi$ maps $z$ to the system of equations

$$\{x_{i(r)} \oplus x_{j(r)} \oplus x_{k(r)} = b(r) \mid r \in \{0,1\}^R\}.$$

This is a polynomial-time reduction, as the number of equations is $m = |\{0,1\}^R| = |z|^{O(1)}$. If $z \in L$, there exists a proof $\pi$ such that $V^\pi(x)$ accepts with probability $1 - \epsilon$ over the choice of $r$. In this case, the assignment $x_i = \pi[i]$ satisfies $(1-\epsilon)m$ of the equations $\psi(z)$. If, on the other hand, some assignment $x_i = a_i$ satisfies $(1/2+\epsilon)m$ of the equations $\psi(z)$, then the proof $\pi$ such that $\pi[i] = a_i$ makes $V^\pi(z)$ accept with probability $(1/2 + \epsilon)m$, implying $z \in L$. It follows that E3LIN2 is inapproximable within a factor of $\frac{(1/2+\epsilon)m}{(1-\epsilon)m} \leq 1/2 + 2\epsilon$, unless $\mathbf{P} = \mathbf{NP}$.

**Max 3SAT.** The $7/8 + \epsilon$ inapproximability of Max 3SAT follows by reduction from Max E3LIN2. The reduction $\psi$ from E3LIN2 to 3SAT maps each equation $x \oplus y \oplus z = b$ into a conjunction of four clauses:

$$\begin{aligned} x \oplus y \oplus z = 0 \quad &\Leftrightarrow \quad (\bar{x} \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee \bar{z}) \\ x \oplus y \oplus z = 1 \quad &\Leftrightarrow \quad (x \vee y \vee z) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z). \end{aligned}$$

Consider an instance $I$ of E3LIN and an assignment $a$ for $I$. If an equation in $I$ is satisfied by $a$, all four corresponding clauses in $\psi(I)$ are satisfied by $a$. Otherwise, exactly three of these four clauses are satisfied by $a$. Therefore, if $(1-\epsilon)m$ equations of $I$ are satisfiable, then $4(1-\epsilon)m+3\epsilon m$ clauses of $\psi(I)$ are satisfiable (by the same assignment). On the other hand, if at most $(1/2+\epsilon)m$ equations of $I$ are satisfiable, then at most $4(1/2+\epsilon)m+3(1/2-\epsilon)m$ clauses of $\psi(I)$ are satisfiable. The ratio between these two values is less than $7/8+\epsilon$.

The factor 7/8 is optimal: All instances $\psi(I)$ have exactly three literals per clause, so a random assignment is expected to satisfy $7m/8$ clauses. Even for arbitrary 3SAT instances, which may contain two literal and one literal clauses, there is a 7/8 approximation algorithm based on semidefinite programming.

**Max 2SAT.** Unlike for Max 3SAT, there is no representation of an E3LIN2 equation by a short 2CNF formula. The reduction from E3LIN2 to 2SAT uses auxiliary variables: Given an equation $x_1 \oplus x_2 \oplus x_3 = b$, there exists a 12 clause 2CNF $\phi$ over variables $x_1, x_2, x_3, y_1, y_2, y_3, y_4$ such that:

1. Every assignment of $\phi$ satisfies at most 11 clauses.
2. If $a_1 \oplus a_2 \oplus a_3 = b$, there is an assignment that satisfies 11 clauses of $\phi$ such that $x_1 = a_1$, $x_2 = a_2$ and $x_3 = a_3$.
3. If $a_1 \oplus a_2 \oplus a_3 \neq b$, every assignment of $\phi$ such that $x_1 = a_1$, $x_2 = a_2$ and $x_3 = a_3$ satisfies at most 10 clauses of $\phi$.

We reduce an E3LIN2 instance $I$ to a 2CNF instance $\psi(I)$ by applying this construction to every equation of $I$. We instantiate distinct copies of $y_1, y_2, y_3$ and $y_4$ for every equation of $I$. If $(1-\epsilon)m$ equations of $I$ are satisfiable, then $11(1-\epsilon)m$ of their corresponding clauses in $\psi(I)$ are satisfiable (by property 2). If fewer than $(1/2+\epsilon)m$ equations of $I$ are satisfiable, then at most $11(1/2+\epsilon)m+10(1/2-\epsilon)m$ equations of $\psi(I)$ are satisfiable (by properties 1 and 3). The ratio between these two values is $21/22+O(\epsilon) \approx 0.954+O(\epsilon)$. Using the same approach, one can show that Max Cut cannot be approximated within $16/17+\epsilon$, unless $\mathbf{P} = \mathbf{NP}$.

The best known approximation algorithm for Max 2SAT is based on semidefinite programming and guarantees a $\approx 0.93$ approximation.

**Min Vertex Cover.** A *vertex cover* for a graph $G$ is a set $C \subseteq V(G)$ such that every edge of $G$ has at least one endpoint in $C$. Equivalently, $C$ is a vertex cover of $G$ if $V(G) - C$ is an independent set. We show that the minimum vertex cover of $G$ cannot be approximated within 7/6, unless $\mathbf{P} = \mathbf{NP}$. Let $I$ be an instance of E3LIN2. For every equation $x_1 \oplus x_2 \oplus x_3 = b$ in $I$, we produce four vertices for $\psi(I)$. Each of these corresponds to an assignment of $x_1, x_2$ and $x_3$ that is consistent with the equation. We draw edges between all pairs of vertices that originated from the same equation, as well as between all pairs of vertices that correspond to inconsistent assignments. If $(1-\epsilon)m$ equations of $I$ are satisfiable, then $\psi(I)$ contains an independent set of size $(1-\epsilon)m$: The independent set is made of vertices corresponding to satisfied clauses consistent with the optimal satisfying assignment of $I$. The complement of this independent set is a vertex cover of size at most $(3+\epsilon)m$. If at most $(1/2+\epsilon)m$ equations of $I$ are satisfiable, then every set of $(1/2+\epsilon)m+1$ vertices of $\psi(I)$

contains an edge. Therefore the largest independent set of $V(G)$ has at most $(1/2+\epsilon)m$ vertices, so its smallest vertex cover has at least $(3\frac{1}{2}-\epsilon)m$ vertices. The ratio between these two values is $7/6 - O(\epsilon)$. The best known inapproximability ratio for this problem is $10\sqrt{5}-21 \approx 1.36$. A trivial algorithm that picks all endpoints of an arbitrary maximal matching achieves a factor 2 approximation. No algorithm is known that improves this factor asymptotically.

## 16.3 From the PCP theorem to Raz's verifier

The **PCP** theorem says that for some constant $\delta > 0$, $\mathbf{NP} = \mathbf{PCP}_{1,\delta}[O(\log n), O(1)]$. Assuming this theorem, we sketch the proof of Theorem 74. The main ingredient is a proof system for **NP** of Raz that makes only two queries, but over proofs represented in a large alphabet. To prove Theorem 74, the first step is to show that the proof system of Raz is equivalent to $\mathbf{PCP}_{1,\delta}[O(\log n), O(1)]$. The second step is to construct a simulation of the two queries performed by Raz's verifier by three queries over an appropriate binary encoding of Raz's proof. Here we cover the first step of the argument.

**Raz's proof system.** A *two-proof system* for $L$ with soundness $\gamma$ is a probabilistic polynomial-time verifier $V$ with oracle access to two proofs, $\pi_1$ and $\pi_2$ over an alphabet of size $1/\gamma^{O(1)}$ such that on input $x$, $V$ performs exactly one query into each of $\pi_1$ and $\pi_2$, and

$$\begin{aligned} x \in L &\Rightarrow \exists \pi_1, \pi_2 : \Pr[V^{\pi_1,\pi_2}(x) = 1] = 1 \\ x \notin L &\Rightarrow \forall \pi_1, \pi_2 : \Pr[V^{\pi_1,\pi_2}(x) = 1] < \gamma. \end{aligned}$$

We show that the **PCP** theorem implies the existence of a two-proof system with soundness $\delta$ for some $\delta > 0$ for any language in **NP**. First, using a canonical reduction, we can restate the **PCP** theorem in the following form: For every $L \in \mathbf{NP}$, there is a polynomial-time reduction $\psi$ that maps instances of $L$ into $3CNF$ formulas such that

$$\begin{aligned} x \in L &\Rightarrow \psi(x) \text{ is satisfiable} \\ x \notin L &\Rightarrow \text{At most } (1-\delta)m \text{ clauses of } \psi(x) \text{ are satisfiable.} \end{aligned}$$

We now construct a two-proof system for 3SAT. The proofs $\pi_1$ and $\pi_2$ are supposed to represent different encodings of an assignment for the input $\phi$. For each variable $x_i$, the value $\pi_1[i] \in \{0,1\}$ represents an assignment of $x_i$. For each clause $C_j$, the value $\pi_2[j] \in \{1,\dots,7\}$ represents one of the seven satisfying assignments for $C_j$. The verifier $V^{\pi_1,\pi_2}$ chooses a random clause $C_j$, a random variable $x_i$ that appears in $C_j$ and accepts iff the assignments $\pi_1[i]$ and $\pi_2[j]$ are consistent.

If $\phi$ is satisfiable, the proofs $\pi_1$ and $\pi_2$ encoding a satisfying assingment of $\phi$ make $V^{\pi_1,\pi_2}$ accept with probability 1. Now suppose that at most $(1-\delta)m$ clauses of $\phi$ are satisfiable. Then at least a $\delta$ fraction of entries in $\pi_2$ are inconsistent with $\pi_1$. If one of these entries in $\pi_2$ is selected by the verifier, at least one of its three literals is assigned inconsistently in $\pi_1$. Therefore an inconsistency is detected with probability $\delta/3$.

We will need a two-proof system for **NP** with much better soundness. We expect to increase soundness by "repeating" the two-proof protocol for 3SAT several times. A $k$-repetition two-proof system for 3SAT is specified as follows: The proof entries $\pi_1[i_1, \ldots, i_k] \in \{0,1\}^k$ and $\pi_2[j_1, \ldots, j_k] \in \{1, \ldots, 7\}^k$ represent partial assignments to the $k$-tuples of variables $x_{i_1}, \ldots, x_{i_k}$ and $k$-tuples of clauses $C_{j_1}, \ldots, C_{j_k}$, respectively. The verifier $V^{\pi_1,\pi_2}$ chooses a random set of $k$ clauses, a random variable in each clause and accepts iff the corresponding assignments in $\pi_1$ and $\pi_2$ are consistent. We might expect that the $k$-repetition of a two-proof system with soundness $1-\gamma$ has soundness $(1-\gamma)^k$. Surprisingly, this is not the case; in the next lecture we will see a two-proof system with soundness $1/2$ such that its 2-repetition has soundness only $5/8$. Fortunately, repetition does amplify soundness at an exponential rate:

**Theorem 75 (Raz)** *If $V$ is a two-proof verifier with soundness $\gamma$, the $k$-repetition of $V$ has soundness $2^{-\Omega(k)}$, where the constant in the $\Omega()$ notation depends only on $\gamma$ and on the alphabet size of the proof system.*

Applying this theorem (which we won't prove), we conclude that for every $\gamma > 0$ there exists a 2-proof system for 3SAT with soundness $\gamma$ with proofs over alphabets of size $1/\gamma^{O(1)}$. In the next three lectures we will show that for suitably small $\gamma = \gamma(\epsilon)$, this 2-proof system can be simulated by $\mathbf{naPCP}_{1-\epsilon,1/2+\epsilon}[O(\log n), 3]$.

## 16.4 References

Håstad's PCP result is from [Hås01]. Raz's result (Theorem 75) is from [Raz98].

# Lecture 17

# Linearity Test

As part of our preparation for an analysis of Håstad's proof system (next lecture), we need to consider the problem of testing whether a given string is a code word for some particular code. First, we will see a procedure that, given a string which may or may not be a code word for the Hadamard code, will decide if it is a code word or if it is far from being a code word. This is equivalent to a test that decides whether a Boolean function is linear or far from linear. That is, given $f : \{0,1\}^k \to \{0,1\}$,

- if $f$ is linear, i.e. $f(x_1,\ldots,x_n) = \bigoplus_{i\in S} x_i$ for some $S \subseteq \{1,\ldots,n\}$, the test accepts with probability 1;
- if the test accepts with probability at least $1/2+\delta$, then $f$ has agreement at least $1/2+\epsilon$ with a linear function.

Here is a possible test:

1. pick $x, y \in \{0,1\}^k$
2. accept if and only if $f(x)\oplus f(y) = f(x\oplus y)$.

We need to show that this test meets the requirements above. In order to analyze the test, we will introduce some concepts from Fourier analysis of Boolean functions.

## 17.1 Fourier analysis of Boolean functions

We are interested in linear functions, that is Boolean functions $l : \{0,1\}^k \to \{0,1\}$ such that there is some $a \in \{0,1\}^k$ for which $l(x) = a\cdot x$ for every $x$. There is a linear function $l_S$ for every set $S \subseteq \{1,\ldots,k\}$ and it is defined as $l_S(x_1,\ldots,x_k) = \bigoplus_{i\in S} x_i$.

If $f : \{0,1\}^k \to \{0,1\}$ is a Boolean function, we denote by $F : \{-1,1\}^k \to \{-1,1\}$ the equivalent Boolean function in the representation where we use 1 instead of 0, $-1$ instead of 1, and multiplication $(\cdot)$ instead of addition $(\oplus)$. That is,

$$f(x_1,\ldots,x_k) = \frac{1}{2} - \frac{1}{2}F(1-2x_1,\ldots,1-2x_k)$$

and

$$F(x_1, \dots, x_k) = 1 - 2f\left(\frac{1}{2} - \frac{1}{2}x_1, \dots, \frac{1}{2} - \frac{1}{2}x_k\right)$$

Note that if $l_S$ is a linear function, then $L_S(x_1, \dots, x_k) = \prod_{i \in S} x_i$ is the equivalent function in this correspondence.

Consider now the set of functions $F : \{-1, 1\}^k \to \mathbb{R}$: this is a vector space of dimension $2^k$ over the reals, and it contains all boolean functions. Define the following scaled dot product over such functions:

$$F \cdot G = \frac{1}{2^k} \sum_{x \in \{-1,1\}^k} F(x)G(x)$$

Notice that if $F : \{-1, 1\}^k \to \{-1, 1\}$, i.e. if $F$ is a Boolean function in this space, then $F \cdot F = 1$. Also notice that if $F, G : \{-1, 1\}^k \to \{-1, 1\}$ then

$$\mathbf{Pr}[f(x) = g(x)] = \frac{1}{2} + \frac{1}{2} F \cdot G$$

How do the linear Boolean functions behave with respect to this dot product? Consider two linear functions $L_S$ and $L_T$, $S \neq T$:

$$\begin{aligned} L_S \cdot L_T &= \frac{1}{2^k} \sum_{x \in \{-1,1\}^k} \left(\prod_{i \in S} x_i\right) \left(\prod_{j \in T} x_j\right) \\ &= \frac{1}{2^k} \sum_{x \in \{-1,1\}^k} \prod_{i \in S \Delta T} x_i \\ &= \mathbf{E}_{x \in \{-1,1\}^k} \prod_{i \in S \Delta T} x_i = \prod_{i \in S \Delta T} \mathbf{E}_x x_i = 0, \end{aligned}$$

where $S \Delta T = (S \setminus T) \cup (T \setminus S)$ is the symmetric difference of $S$ and $T$. To get the third line, we think of the sum as an expectation over $x \in \{-1, 1\}^k$, where $x$ is chosen randomly by setting the $i$th entry independently to $\pm 1$, and therefore it is the expectation of a product of independent variables.

So all functions $L_S$ are orthogonal to each other, that is, $L_S \cdot L_T = 0$ if $S \neq T$. Then all such functions must be linear independent, and since there are $2^k$ of them in a space of dimension $2^k$ they form an orthonormal basis. So we can write each function $F : \{-1, 1\}^k \to \mathbb{R}$ as

$$F(x) = \sum_{S \subseteq \{1, \dots, k\}} \hat{F}_S L_S(x).$$

This is known as the Fourier expansion of $F$. The Fourier coefficient $\hat{F}_S$ can be computed as $\hat{F}_S = F \cdot L_S$. Why would we write $F$ like this? We'll see that the Fourier coefficients have a useful interpretation.

If $F$ is Boolean, then

$$\begin{aligned}
\hat{F}_S &= F \cdot L_S \\
&= \mathbf{E}_x F(x) L_S(x) \\
&= \mathbf{Pr}_x[F(x) = L_S(x)] - \mathbf{Pr}_x[F(x) \neq L_S(x)] - 1 \\
&= 2\mathbf{Pr}_x[F(x) = L_S(x)] - 1 \\
&= 2\mathbf{Pr}_x[f(x) = l_S(x)] - 1.
\end{aligned}$$

Equivalently, for every $S$, we have $\mathbf{Pr}_x[f(x) = l_S(x)] = \frac{1}{2} + \frac{1}{2}\hat{F}_S$. The Fourier coefficients range between 1 and $-1$, and in a sense, they represent the bias that a Boolean function $f$ has with respect to each of the linear functions. The coefficient $\hat{F}_S$ is 1 when $f$ is exactly the corresponding linear function $l_S$, and it is -1 when $f$ is exactly the complement of the linear function.

We will see that if the linearity test for a Boolean function $f$ given at the beginning of the lecture accepts with high probability, then there is a large Fourier coefficient $\hat{F}_S$, which implies that there is a linear function $l_S$ with nontrivial agreement with $f$. First, we need a couple of results about Fourier coefficients.

**Lemma 76 (Parseval's Equality)** *For every $F : \{-1,1\}^k \to \mathbb{R}$, $\mathbf{E}_x[F^2(x)] = \sum_S \hat{F}_S^2$.*

One way to interpret this is that the set of values of $F$ and the Fourier coefficients in the expansion of $F$ have the same $l_2$ norm up to a constant.

**Corollary 77** *For every $F : \{-1,1\}^k \to \{-1,1\}$, $\sum_S \hat{F}_S^2 = 1$.*

**Lemma 78** *Let $f : \{-1,1\}^k \to \{-1,1\}$. There are at most $1/(4\epsilon^2)$ linear functions $l_S$ such that $f$ and $l_S$ have agreement at least $1/2 + \epsilon$.*

We have used the result in Lemma 78 before when we did worst case to average case reductions. We can easily prove it now using what we know about Fourier coefficients.

Proof: [Lemma 78] Suppose $l_s$ and $f$ have agreement at least $1/2 + \epsilon$. Then $\hat{F}_S \geq 2\epsilon$, which implies that $\hat{F}_S^2 \geq 4\epsilon^2$, but from Corollary 77, we know that $\sum_S \hat{F}_S^2 = 1$. So there can be at most $1/(4\epsilon^2)$ functions $l_S$ with that much agreement with $f$. □

Proof: [Lemma 76 (Parseval's Equality)]

$$\begin{aligned}
\mathbf{E}_x[F^2(x)] &= \mathbf{E}_x\left[\left(\sum_S \hat{F}_S L_S(x)\right)\left(\sum_T \hat{F}_T L_T(x)\right)\right] \\
&= \sum_S \sum_T \hat{F}_S \hat{F}_T \mathbf{E}_x[L_S(x) L_T(x)] \\
&= \sum_S \sum_T \hat{F}_S \hat{F}_T L_S \cdot L_T \\
&= \sum_S \hat{F}_S^2.
\end{aligned}$$

□

## 17.2 Using Fourier analysis to test linearity

Now consider the test to check if $f : \{0,1\}^k \to \{0,1\}$ is linear: pick at random $x, y \in \{0,1\}^k$ and accept iff $f(x) \oplus f(y) = f(x \oplus y)$. If $f$ is linear then the test accepts with probability one.

**Lemma 79** *If the test accepts $f$ with probability $p$, then there is a linear function $l_S$ that agrees with $f$ on at least a $p$ fraction of inputs.*

Proof: First observe that

$$\mathbf{Pr}[\text{ test accepts } f\ ] = \frac{1}{2} + \frac{1}{2}\mathbf{E}_{x,y}[F(x)F(y)F(xy)].$$

This follows from the same type of argument we have made before, based on the fact that $F(x)F(y)F(xy)$ is 1 when the test accepts and $-1$ when it rejects:

$$\begin{aligned}\mathbf{E}_{x,y}[F(x)F(y)F(xy)] &= \mathbf{Pr}_{x,y}[\text{ test accepts } f\ ] - \mathbf{Pr}_{x,y}[\text{ test rejects } f\ ] \\ &= 2\mathbf{Pr}_{x,y}[\text{ test accepts } f\ ] - 1.\end{aligned}$$

Now, we need to prove that when $\mathbf{E}_{x,y}[F(x)F(y)F(xy)]$ is bounded away from zero, there is a large Fourier coefficient. To do this we just use the Fourier expansion of each function and simplify.

$$\begin{aligned}\mathbf{E}_{x,y}[F(x)F(y)F(xy)] &= \mathbf{E}_{x,y}\left[\left(\sum_S \hat{F}_S L_S(x)\right)\left(\sum_T \hat{F}_T L_T(y)\right)\left(\sum_U \hat{F}_U L_U(xy)\right)\right] \\ &= \sum_S\sum_T\sum_U \hat{F}_S\hat{F}_T\hat{F}_U \mathbf{E}_{x,y}[L_S(x)L_T(y)L_U(x)L_U(y)] \\ &= \sum_S\sum_T\sum_U \hat{F}_S\hat{F}_T\hat{F}_U \left(\mathbf{E}_x L_S(x)L_U(x)\right)\left(\mathbf{E}_y L_T(y)L_U(y)\right) \\ &= \sum_S\sum_T\sum_U \hat{F}_S\hat{F}_T\hat{F}_U (L_S \cdot L_U)(L_T \cdot L_U) \\ &= \sum_S \hat{F}_S^3 \ \leq\ \max_S \hat{F}_S \sum_S \hat{F}_S^2 \ =\ \max_S \hat{F}_S.\end{aligned}$$

So the largest Fourier coefficient of $F$ is an upper bound on the expectation $\mathbf{E}_x[F(x)F(y)F(xy)]$, and the expectation tells us how likely the test is to accept $f$. We have shown so far that for every $S$, $\mathbf{Pr}_x[f(x) = l_S(x)] = \frac{1}{2} + \frac{1}{2}\hat{F}_S$, and that $\mathbf{Pr}[\text{ test accepts } f\ ] \leq \frac{1}{2} + \frac{1}{2}\max_S \hat{F}_S$. Therefore if $\mathbf{Pr}[\text{ test accepts } f\ ] \geq 1/2 + \epsilon$, then there exists an $S$ such that $\hat{F}_S \geq 2\epsilon$, i.e. there is an $S$ such that $f$ and $l_S$ have agreement on at least $1/2 + \epsilon$ fraction of inputs. $\Box$

Since not more than 1 in $4\epsilon^2$ functions can have that much agreement with $f$, if we do list decoding for $f$ then we come up with a small list.

## 17.3 Testing code words for the Long Code

So far, we have been thinking of the encoding of a message $m \in \{0,1\}^k$ in the Hadamard code as a function $l_S$, where $S = \{i : \ m_i = 1\}$. An equivalent way to think of the encoding is as the concatenation of $a_1 \cdot m, a_2 \cdot m, \dots a_{2^k} \cdot m$ where $a_1, \dots, a_{2^k}$ is an enumeration of $\{0,1\}^k$. This is how we though of the code in worst case to average case reductions. It is also equivalent to think of the encoding as the concatenation of $l_{S_1}(m), l_{S_2}(m), \dots, l_{S_{2^k}}(m)$, where $S_1, \dots, S_{2^k}$ is an enumeration of subsets of $B^k$, that is, the encoding is the concatenation of the evaluation of all possible linear functions at $m$. In fact, for any code, it is possible to think of the encoding as the concatenation of the evaluation of a series of functions at the input value.

Let us now consider the error correcting code, called the Long Code, used in Håstad's proof system. One way to think of the encoding of $m \in \{1, \dots, k\}$ in the Long Code is as a functional $M : (\{1, \dots, k\} \to \{0,1\}) \to \{0,1\}$ that given a function $f : \{1, \dots, k\} \to \{0,1\}$ returns $M(f) = f(m)$. Equivalently, the encoding of $m$ can be thought of as the concatenation of $f_1(m), f_2(m), \dots, f_{2^k}(m)$ where $f_1, \dots, f_{2^k}$ is an enumeration of *all* possible functions $f : \{1, \dots, k\} \to \{0,1\}$. Every error correcting code can be seen as a subset of this code, and no encoding can be longer than the Long Code unless two of the functions being evaluated are exactly the same.

Notice that the Long Code is doubly exponentially long: in order to transmit $\log k$ bits of information (since $m \in \{1, \dots, k\}$), we need to send an encoding that takes $2^k$ bits. In Håstad's proof system, the code is only used on inputs of a constant size, so this blow-up does not cause a problem.

Another equivalent way to think of the encoding of $m$ is as the concatenation of $a_1[m], a_2[m], \dots, a_{2^k}[m]$ where $a_1, \dots, a_{2^k}$ is an enumeration of $a \in \{0,1\}^k$, that is, it is a concatenation of the $m$th entry in each string of length $k$. Equivalently, the encoding is a function that maps strings to one of their bits: the encoding of $m$ is a function $M : \{0,1\}^k \to \{0,1\}$ that given an $x \in \{0,1\}^k$ returns $M(x) = x[m]$. From this point of view, the Long Code encodes a message $m \in \{1, \dots, k\}$ as a special linear function $l_{\{m\}}$. There is a loss of expressive power in this representation $l_{\{m\}}$ (note that it requires only $\log k$ bits of information), but it is convenient for our purposes.

### 17.3.1 Pre-Håstad verification

To get some understanding of how the Long Code test works in verification, we'll start by considering the verification model in vogue before Håstad's verifier. Given a function $A : \{0,1\}^k \to \{0,1\}$, we want the verifier to check that $A$ is actually a code word for the Long Code as follows: for some predicate $P$, the verifier should check that there is some $a \in \{1, \dots, k\}$ satisfying the predicate, $P(a) = 1$, such that given any string $x$, $A(x) = x[a]$, or in other words, $A$ corresponds to some linear function $l_{\{a\}}$.

So suppose that after performing some test, we find that $A$ has agreement $1 - \delta$ with some $l_{\{a\}}$.

Then define the vector $p \in \{0,1\}^k$ such that $p[i] = P(i)$ for every $i$, and do the following test: choose a random $x \in \{0,1\}^k$ and accept if and only if $A(p \oplus x) \oplus A(x) = 1$.

For a random $x$, $x \oplus f$ is also random, and so there is at least a $1 - 2\delta$ probability that both $A(x) = x[a]$ and $A(p \oplus x) = p[a] \oplus x[a]$. Thus, if $P(a) = 1$, the test accepts with probability at least $1 - 2\delta$, and if $P(a) = 0$ the test accepts with probability at least $2\delta$.

### 17.3.2 The Long Code test

Given $A : (\{1, \dots, k\} \to \{0,1\}) \to \{0,1\}$,

- pick $f, g : \{1, \dots, k\} \to \{0,1\}$ uniformly
- pick $h : \{1, \dots, k\} \to \{0,1\}$ such that for every $x$

$$h(x) = \begin{cases} 0 & \text{with probability } 1 - \epsilon \\ 1 & \text{with probability } \epsilon \end{cases}$$

- accept iff $A(f) \oplus A(g) = A(f \oplus g \oplus h)$

Suppose that $A = l_{\{a\}}$. Then with high probability the test will accept since with high probability $f(a) \oplus g(a) = f(a) \oplus g(a) \oplus h(a)$. If $A$ has no correlation with any linear function then this test is close enough to a test for linearity that the test will fail. If $A$ is linear but *not* a code word for the Long Code , it cannot be defined over a very large set. Suppose $A = l_S$ for some set $S$. Then the probability that

$$\bigoplus_{x \in S} f(x) \bigoplus_{x \in S} g(x) = \bigoplus_{x \in S} f(x) \bigoplus_{x \in S} g(x) \bigoplus_{x \in S} h(x)$$

gets exponentially close to 0 with the cardinality of $S$.

## 17.4 References

The linearity test studied in these notes is due to Blum, Luby and Rubinfeld [BLR93]. The analysis that we presented is due to Bellare et al. [BCH$^+$96]. The Long Code was introduced by Bellar, Goldreich and Sudan [BGS98]. The efficient test is due to Håstad [Hås01].

## Exercises

1. Show that if $f : \{0,1\}^n \to \{0,1\}$ is an arbitrary function, then $\max_S |\hat{F}_S| \geq 2^{-n/2}$.

2. Compute the Fourier coefficients of the function $f(x_1, x_2) = x_1 \vee x_2$.

3. Let $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ be arbitrary functions, and let $h : \{0,1\}^{n+m} \to \{0,1\}$ be defined as

$$h(x_1, \ldots, x_{n+m}) = f(x_1, \ldots, x_n) \oplus g(x_{n+1}, \ldots, x_{n+m})$$

   Compute the Fourier coefficients of $h$ in terms of the Fourier coefficients of $f$ and $g$.

4. Let $n$ be even and consider the function

$$f(x_1, \ldots, x_n) = (x_1 \vee x_2) \oplus (x_3 \vee x_4) \oplus \cdots (x_{n-1} \vee x_n)$$

   Prove that $\max_S |\hat{F}_S| = 2^{-n/2}$.

5. Suppose that $f : \{0,1\}^n \to \{0,1\}$ is a function that depends only on its firts $k$ variables, that is, suppose that there is a function $g : \{0,1\}^k \to \{0,1\}$ such that $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_k)$. Compute the Fourier coefficients of $f$ in terms of the Fourier coefficients of $g$.

6. Analyse the following linearity test: given oracle access to a function $f : \{0,1\}^n \to \{0,1\}$, pick uniformly at random four points $x_1, x_2, x_3, x_4 \in \{0,1\}^n$ and accept if and only if

$$f(x_1) \oplus f(x_2) \oplus f(x_3) \oplus f(x_4) = f(x_1 \oplus x_2 \oplus x_3 \oplus x_4)$$

# Lecture 18

# Testing the Long Code

## 18.1 Test for Long Code

We now consider a test to distinguish codewords of the long code. Given $A : \{0,1\}^k \rightarrow \{0,1\}$ pick $x, y \in \{0,1\}^k$ uniformly and $z \in \{0,1\}^k$ where the bits of $z$ are IID with $\Pr[z[i] = 1] = \epsilon$. Accept iff $A(x) \oplus A(y) = A(x \oplus y \oplus z)$.

If $A$ is a codeword of the long code, then $A = l_{\{a\}}$ and the test check if $x[a] \oplus y[a] = x[a] \oplus y[a] \oplus z[a]$ and succeeds iff $z[a] = 0$. Hence it succeds with probability $1 - \epsilon$.

We now show that if the test accepts with probability $1/2 + \delta$, then list decoding of the codeword produces a list with size dependent on $\delta$ rather than $k$. Intuitively this is similar to the linearity test of the last lecture.

Consider the case where $A$ is a linear function with large support: $A = l_S$, $S$ is large. Then the test checks if $l_S(x) \oplus l_S(y) = l_S(x) \oplus l_S(y) \oplus l_S(z)$ and will succeed iff $l_S(z) = 0$. The probability of this occurring is $\frac{1}{2} + \frac{1}{2}(1 - 2\epsilon)^{|S|}$. So the test cannot accept with high probability if $A$ is not linear or if $A$ is linear but has large support.

We now consider the equivalent setting where we replace 0 with 1, 1 with -1, and $\oplus$ with $\times$ so that $A : \{-1,1\} \rightarrow \{-1,1\}, x, y, z \in \{-1,1\}^k$. Here the test accepts iff $A(x)A(y)A(xyz) = 1$. Specifically, $\Pr[\text{test accepts } A] = \frac{1}{2} + \frac{1}{2}E_{x,y,z}[A(x)A(y)A(xyz)]$

where:

$$\begin{aligned}
&E_{x,y,z}[A(x)A(y)A(xyz)] \\
&= E_{x,y,z}\left[\left(\sum_S \hat{A}_S L_S(x)\right)\left(\sum_T \hat{A}_T L_T(y)\right)\left(\sum_U \hat{A}_U L_U(x)L_U(y)L_U(z)\right)\right] \\
&= \sum_{S,T,U} \hat{A}_s \hat{A}_T \hat{A}_U \overbrace{(E_x[L_S(x)L_U(x)])}^{L_S \cdot L_U} \overbrace{(E_y[L_T(y)L_U(y)])}^{L_T \cdot L_U} (E_z L_U(z)) \\
&= \sum_S \hat{A}_S^3 E_z[L_S(z)] \\
&= \sum_S \hat{A}_S^3 \prod_{i \in S} E[z_i] \\
&= \sum_S \hat{A}_S^3 (1-2\epsilon)^{|S|} \\
&\leq \left(\max_S \hat{A}_S (1-2\epsilon)^{|S|}\right) \cdot \overbrace{\sum_S \hat{A}_S^2}^{=1}
\end{aligned}$$

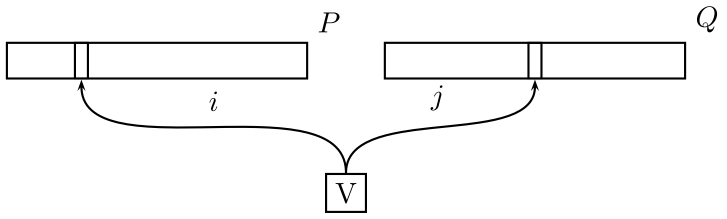
Now suppose that the probability that the test accepts is greater than $\frac{1}{2}+\delta$. Then there is some $S$ where $\hat{A}_S(1-2\epsilon)^{|S|} \geq 2\delta$. Hence:

$$\begin{aligned}
\hat{A}_S &\geq 2\delta \\
(1-2\epsilon)^{|S|} &\geq 2\delta \\
|S| \log \frac{1}{1-2\epsilon} &\leq \log \frac{1}{2\delta} \\
|S| &\leq \frac{\log \frac{1}{2\delta}}{\log \frac{1}{1-2\epsilon}}
\end{aligned}$$

This is close to the the analysis of Håstad's proof system. The primary difference is that Håstad's system looks at two strings rather than one.

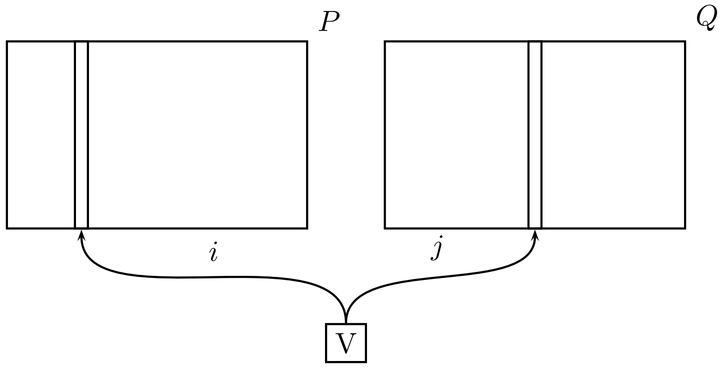
## 18.2 Håstad's verifier

First recall Raz's verifier:

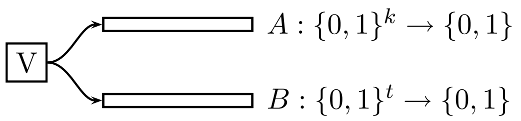


The proof $P$ is over the alphabet $\Sigma, |\Sigma| = k$ and the proof $Q$ is over the alphabet $\Gamma, |\Gamma| = t$. The verifier takes the character at position $i$ from $P$ and the character $j$ from $Q$, where $j$ is selected at random and $i$ is selected at random from a few positions of $P$. The verifier accepts iff $P[i] = \pi(Q[i])$.

Now consider Håstad's verifier:



In the proofs for Håstad's verifier, the binary strings indexed by $i, j$ are the long code encodings of the characters at those positions in Raz's Verifier. Håstad's verifer selects $i, j$ the same way as Raz's so that the problem can be modeled as:



Here the function $\pi : \{1, \ldots, t\} \to \{1, \ldots, k\}$ is as before. The verifier will accept with probability greater than $1 - \epsilon$ if $A$ is the long code of $P[i]$, $B$ is the long code of $Q[j]$ and $\pi(Q[j]) = P[i]$. We will show that if the test accepts with probability $\frac{1}{2} + \epsilon$ then $\Pr[D(A) = \pi(D(B))] \geq \text{poly}(\epsilon)$.

We digress for a moment to consider a subtle point about the long code test. What if $S = \emptyset$? This is the case where the function is 0 everywhere. Then the test passes with probability 1. This would not be a problem if the function is balanced. A function $A$ is self-dual if:

$$\begin{aligned} A(1-x) &= 1 - A(x) \text{ when } x, A(x) \in \{0,1\} \\ A(-x) &= -A(x) \text{ when } x, A(x) \in \{-1,1\} \end{aligned}$$

For example, $\oplus$ of an even number of inputs is not self-dual while $\oplus$ of an odd number of inputs is self-dual. If $A$ is self-dual then $\hat{A}_S = 0$ if $|S|$ is even. Consider the following procedure called "folding" to convert any function $A : \{0,1\}^k \to \{0,1\}$ into a self dual function $A' : \{0,1\}^k \to \{0,1\}$:

$$A'(x) = \begin{cases} A(x) & \text{if } x[1] = 0 \\ 1 - A(1-x) & \text{if } x[1] = 1 \end{cases}$$

This is self-dual since:

$$A'(1-x) = \begin{cases} 1 - A(x) = 1 - A'(x) & \text{if } x[1] = 0 \\ A(1-x) = 1 - A'(x) & \text{if } x[1] = 1 \end{cases}$$

In our case of interest, the long code, then $A' = A$. In the following analysis, all testing and decoding is performed on $A', B'$.

Given $A : \{0,1\}^k \to \{0,1\}, B : \{0,1\}^t \to \{0,1\}, \pi : \{1,\dots,t\} \to \{1,k\}$ we pick $x \in \{0,1\}^k, y \in \{0,1\}^t$ uniformly. We also pick $z \in \{0,1\}^t$ where the bits are IID with $\Pr[z[i] = 1] = \epsilon$. The test accepts iff:

$$A'(x) \oplus B'(y) = B'((x \circ \pi) \oplus y \oplus z)$$

(From now on we will stop distinguishing between $A$ and $A'$ and between $B$ and $B'$.)

Treating vectors as functions, $x \circ \pi \in \{0,1\}^t, (x \circ \pi)[i] = x[\pi(i)]$. In the case where the proof is based on a valid proof for Raz's verifier, we have $a = P[i], b = Q[j], A = l_{\{a\}}, B = l_{\{b\}}, \pi(b) = a$ and the test checks if:

$$x[a] \oplus y[b] = x[\pi(b)] \oplus y[b] \oplus z[b] = x[a] \oplus y[b] \oplus z[b]$$

And hence the test accepts with probability $1 - \epsilon$. Considering the equivalent setting of $A : \{-1,1\}^k \to \{-1,1\}, B : \{-1,1\}^t \to \{-1,1\}$. Using the Fourier analysis machinery we know that:

$$\Pr[\text{test accepts}] = \frac{1}{2} + \frac{1}{2} E_{xyz}[A(x)B(y)B((x \circ \pi)yz)].$$

Define $\pi_2(\beta) \triangleq \{i \in \{1,\dots,k\} | \text{an odd number of } j \in \beta \text{ satisfy } \pi(j) = i\}$ so that $\prod_{j\in\gamma} x[\pi(j)] = \prod_{i\in\pi_2(\gamma)} x_i$. We can then calculate:

$$\begin{aligned}
& E_{x,y,z}[A(x)B(y)B((x \circ \pi)yz)] \\
&= E_{x,y,z}\left[\left(\sum_\alpha \hat{A}_\alpha L_\alpha(x)\right)\left(\sum_\beta \hat{B}_\beta L_\beta(y)\right)\left(\sum_\gamma \hat{B}_\gamma L_\gamma(x\circ\pi) L_\gamma(y) L_\gamma(z)\right)\right] \\
&= \sum_{\alpha,\beta,\gamma} \hat{A}_\alpha \hat{B}_\beta \hat{B}_\gamma \overbrace{E_x[L_\alpha(x) L_\gamma(x\circ\pi)]}^{L_\alpha \cdot L_{\pi_2(\gamma)}} \overbrace{E_y[L_\beta(y) L_\gamma(y)]}^{L_\beta \cdot L_\gamma} \overbrace{E_z[L_\gamma(z)]}^{(1-2\epsilon)^{|\gamma|}} \\
&= \sum_\beta \hat{A}_{\pi_2(\beta)} \hat{B}_\beta^2 (1-2\epsilon)^{|\beta|}
\end{aligned}$$

Now suppose that $\Pr[\text{test accepts}] \geq \frac{1}{2} + \delta$ so that:

$$\begin{aligned}
2\delta &\leq \sum_\beta \hat{A}_{\pi_2(\beta)} \hat{B}_\beta^2 (1-2\epsilon)^{|\beta|} \\
&\leq \sum_\beta |\hat{A}_{\pi_2(\beta)}| \hat{B}_\beta^2 (1-2\epsilon)^{|\beta|} \\
&= \sum_{\beta : |\hat{A}_{\pi_2(\beta)}| \geq \delta} |\hat{A}_{\pi_2(\beta)}| \hat{B}_\beta^2 (1-2\epsilon)^{|\beta|} + \overbrace{\sum_{\beta : |\hat{A}_{\pi_2(\beta)}| < \delta} |\hat{A}_{\pi_2(\beta)}| \hat{B}_\beta^2 (1-2\epsilon)^{|\beta|}}^{<\delta}
\end{aligned}$$

Multiplying both sides by $\delta$ yields:

$$\begin{aligned}
\delta^2 &\leq \sum_{\beta:|\hat{A}_{\pi_2(\beta)}|\geq\delta} \hat{A}^2_{\pi_2(\beta)}\hat{B}^2_\beta(1-2\epsilon)^{|\beta|} \\
&\leq \sum_\beta \hat{A}^2_{\pi_2(\beta)}\hat{B}^2_\beta(1-2\epsilon)^{|\beta|}
\end{aligned}$$

Now consider the following decoding algorithm: $D(A)$ picks $\alpha \subseteq \{1,\ldots,k\}$ with probability $\hat{A}^2_\alpha$ and $a \in \alpha$ at random. Similarly $D(B)$ picks $\beta \subseteq \{1,\ldots,t\}$ with probability $\hat{B}^2_\beta$ and $b \in \beta$ at random. Using this strategy and noting that $\exists c : (1-2\epsilon)^k \leq \frac{c}{k}, c = O(\frac{1}{\epsilon})$ we find:

$$\begin{aligned}
\Pr[D(A) = \pi(D(B))] &\geq \sum_\beta \hat{A}^2_{\pi_2(\beta)}\hat{B}^2_\beta \frac{1}{|\beta|} \\
&\geq \frac{1}{c}\sum_\beta \hat{A}^2_{\pi_2(\beta)}\hat{B}^2_\beta(1-2\epsilon)^{|\beta|} \\
&\geq \frac{\delta^2}{c} = \Theta(\epsilon\delta^2)
\end{aligned}$$

This probability does not depend on $t$ or $k$, so that we can adjust the soundness parameter of Raz's verifier and not have a circular relationship of the variables.

# Lecture 19

# Circuit Lower Bounds for Parity Using Polynomials

In this lecture, we will talk about circuit lower bounds and take our first step towards proving a result about the lower bound on the size of a constant depth circuit which computes the XOR of $n$ bits.

Before we talk about bounds on the size of a circuit, let us first clarify what we mean by circuit depth and circuit size. The depth of a circuit is defined as the length of the longest path from the input to output. The size of a circuit is the number of AND and OR gates in the circuit.Note that for our purpose, we assume all the gates have unlimited fan-in and fan-out. We define $\mathbf{AC_0}$ to be the class of decision problems solvable by circuits of polynomial size and constant depth. We want to prove the result that **PARITY** is not in $\mathbf{AC_0}$.

We will try to prove this result using two different techniques. In this class, we will talk about a proof which uses polynomials; in the next class we will look at a different proof which uses random restrictions.

## 19.1 Circuit Lower Bounds for PARITY

Before we go into our proof, let us first look at a circuit of constant depth $d$ that computes **PARITY**.

**Theorem 80** *If a circuit of size $S$ and depth $d$ computes* **PARITY** *then* $S \geq 2^{\Omega(n^{\frac{1}{d-1}})}$

This bound of $2^{\Omega(n^{\frac{1}{(d-1)}})}$ is actually tight, as the following example shows.

Consider the circuit $C$ shown in Figure(19.1), which computes the **PARITY** of $n$ variables. $C$ comprises of a tree of XOR gates, each of which has fan-in $n^{\frac{1}{d-1}}$; the tree has a depth of $d-1$.

Now, since each XOR gate is a function of $n^{\frac{1}{d-1}}$ variables, it can be implemented by a CNF or a DNF of size $2^{n^{\frac{1}{d-1}}}$. Let us replace alternating layers of XOR gates in the

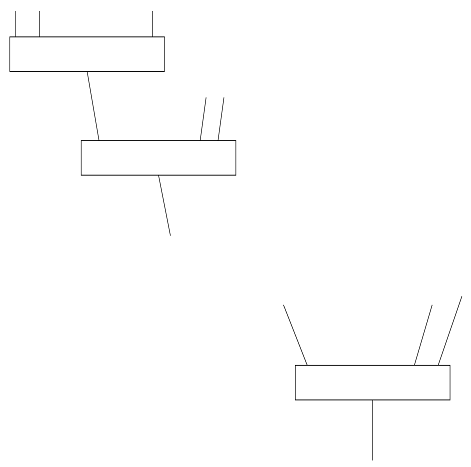


Figure 19.1: Circuit for Computing XOR of n variables; each small circuit in the tree computes the XOR of $k = n^{\frac{1}{d-1}}$ variables

tree by CNF's and DNF's - for example we replace gates in the first layer by their CNF implementation, gates in the second layer by their DNF implementation, and so on. This gives us a circuit of depth $2(d-1)$. Now we can use the associativity of OR to collapse consecutive layers of OR gates into a single layer. The same thing can be done for AND to get a circuit of depth $d$.

This gives us a circuit of depth $d$ and size $O(n2^{n^{\frac{1}{d-1}}})$ which computes **PARITY**.

## 19.2 Proof of Circuit Lower Bounds for PARITY using Polynomials

For our proof, we will utilise a property which is common to all circuits of small size and constant depth, which **PARITY** does not have. The property is that circuits of small size and constant depth can be represented by low degree polynomials, with high probability. More formally, we show that if a function $f : \{0,1\}^n \to \{0,1\}$ is computed by a circuit of size $s$ and depth $d$, then there exists a function $g : \{0,1\}^n \to \Re$ such that $\Pr_x[f(x) = g(x)] \geq \frac{3}{4}$ and $\hat{g}_\alpha \neq 0$ only for $|\alpha| \leq O((\log S)^{2d})$.

Then we will show that if a function $g : \{0,1\}^n \to \Re$ agrees with **PARITY** on more than a fraction $\frac{3}{4}$ of its inputs, then $\hat{g}_\alpha = 0$ only for $|\alpha| = \Omega(\sqrt{n})$. That is, a function which agrees with **PARITY** on a large fraction of its inputs, has to have high degree. From these two results, it is easy to see that **PARITY** cannot be computed by circuits of constant depth and small size.

**Lemma 81** *Suppose we can describe a distribution $G_{OR}$ over functions $g : \{0,1\}^n \to \{0,1\}$ such that*

- *for every $g$ in $G_{OR}$, $\hat{g}_\alpha \neq 0$ only for $|\alpha| \leq O\left((\log \frac{1}{\epsilon})(\log K)\right)$*

- *for every $x \in \{0,1\}^n$, $Pr_{g \sim G_{OR}}[g(x) = x_1 \vee x_2 \vee \cdots \vee x_k] \geq 1 - \epsilon$. Here $K$ is the maximum fan-in of each gate.*

*Also suppose we can describe a such a distribution $G_{AND}$ for AND gates. Then, given a circuit $C$ of size $S$ and depth $d$, we can find a polynomial $h$ of degree at most $O((\log S)^{2d})$, such that $\Pr_x[h(x) = C(x)] \geq \frac{3}{4}$*

Proof: The way we do this is as follows. For every gate in a circuit $C$, we pick independently an approximating function $g_i$ with parameter $\epsilon = \frac{1}{4S}$, and replace the gate by $g_i$. Then, for a given input, probability that the new circuit computes $C(x)$ correctly, that is, the probability that the results of all the gates are correctly computed, is at least $\frac{3}{4}$. Let $G_i : \{1,-1\}^n \to \{1,-1\}$ be the Fourier transformed version of function $g_i$, so that

$$g_i(x_1, \cdots, x_n) = \frac{1}{2} + \frac{1}{2}G_i(1 - 2x_1, 1 - 2x_2, \cdots, 1 - 2x_n) \tag{19.1}$$

Then $G_i$ can be written as

$$G_i = \sum_\alpha \hat{g}_{i,\alpha} \prod_{j \in \alpha} X_j \tag{19.2}$$

where $X_i = 1 - 2x_i$. Then $G_i$ has degree at most $O((\log S)^2)$, as $K$, the fan-in of any gate, is at most $S$. Therefore, if we expand out the expression for $C(x)$, we will get a multilinear polynomial of degree at most $O((\log S)^{2d})$, as $C$ has depth $d$. $\Box$

**Lemma 82** *Suppose there is a function $g : \{0,1\}^n \to \Re$ such that*

- $\Pr_x[g(x) = x_1 \oplus x_2 \oplus \cdots \oplus x_n] \geq \frac{3}{4}$ *and*
- $\hat{g}_\alpha = 0$*, for all sets $\alpha$ such that $|\alpha| > d$.*

*Then $d = \Omega(\sqrt{n})$*

Proof: Let $G : \{1,-1\}^n \to \{1,-1\}$ be the Fourier transformed version of the function $g$. That is,

$$G(x) = 1 - 2g(\frac{1}{2} - \frac{1}{2}x_1, \cdots, \frac{1}{2} - \frac{1}{2}x_n) \tag{19.3}$$

Then, $\Pr_x[G(X) = \prod_{i=1}^n X_i] \geq \frac{3}{4}$, where $X_i = 1 - 2x_i$. Define $A$ to be the set of values over which $g$ agrees with **PARITY**. In other words,

$$A = \{X : G(X) = \prod_{i=1}^n X_i\} \tag{19.4}$$

Then $|A| \geq \frac{3}{4}2^n$, by our initial assumption. Now consider the set of all functions $S = \{f : A \to \Re\}$. These form a vector space of dimension $|A|$. Any function $f$ in this set can be written as

$$f(X) = \sum_\alpha \hat{f}_\alpha \prod_{i \in \alpha} X_i \tag{19.5}$$

Over $A$, $G(X) = \prod_{i=1}^{n} X_i$. Therefore, for $X \in A$,

$$\prod_{i\in\alpha} X_i = G(X) \prod_{i\notin\alpha} X_i \tag{19.6}$$

By our initial assumption, $G(X)$ is a polynomial of degree at most $d$. Therefore, for every $\alpha$, such that $|\alpha| \geq \frac{n}{2}$, we can replace $\prod_{i\in\alpha} X_i$ by a polynomial of degree less than or equal to $d + \frac{n}{2}$. Every such function $f$ which belong to $S$ can be written as a polynomial of degree at most $d + \frac{n}{2}$. Hence the set $\left\{\prod_{i\in\alpha} X_i\right\}_{|\alpha|\leq d+\frac{n}{2}}$ forms a basis for the set $S$. As there must be at least $|A|$ such monomials, this means that

$$\sum_{k=0}^{d+\frac{n}{2}} \binom{n}{k} \geq \frac{3}{4} 2^n \tag{19.7}$$

This holds when $d = \Omega(\sqrt{n})$. $\square$

Now we need to prove that we can indeed describe such distributions $G_{OR}$ and $G_{AND}$ over functions $g_i$ for OR and AND.

## 19.3 Approximating OR

The following lemma says that we can approximately represent OR with a polynomial of degree exponentially small in the the fan-in of the OR gate. We'll use the notation that $x$ is a vector of $k$ bits, $x_i$ is the $i$th bit of $x$, and $\mathbf{0}$ is the vector of zeros (of the appropriate size based on context).

**Lemma 83** *For all $k$ and $\epsilon$, there exists a distribution $G$ over functions $g : \{0,1\}^k \to \mathbb{R}$ such that*

1. *$g$ is a degree $O((\log \frac{1}{\epsilon})(\log k))$ polynomial, and*
2. *for all $x \in \{0,1\}^k$,*

$$\Pr_{g\sim G}\left[g(x) = x_1 \vee \ldots \vee x_k\right] \geq 1 - \epsilon. \tag{19.8}$$

Proof Idea: We want a random polynomial $p : \{0,1\}^k \to \mathbb{R}$ that computes OR. An obvious choice is

$$p_{\text{bad}}(x_1, \ldots, x_k) = 1 - \prod_{i\in\{1,\ldots,k\}} (1 - x_i), \tag{19.9}$$

which computes OR with no error. But it has degree $k$, whereas we'd like it to have logarithmic degree. To accomplish this amazing feat, we'll replace the tests of all $k$ variables with just a few tests of random batches of variables. This gives us a random polynomial which computes OR with one-sided error: when $x = \mathbf{0}$, we'll have $p(x) = 0$; and when some $x_i = 1$, we'll almost always (over our choice of $p$) have $p(x) = 1$.

Proof: We pick a random collection $C$ of subsets of the bits of $x$. (That is, for each $S \in C$ we have $S \subseteq \{1, \ldots, k\}$). We'll soon see how to pick $C$, but once the choice has been made, we define our polynomial as

$$p(x_1, \ldots, x_k) = 1 - \prod_{S \in C} \left(1 - \sum_{i \in S} x_i\right). \tag{19.10}$$

Why does $p$ successfully approximate OR? First, suppose $x_1 \vee \ldots \vee x_k = 1$. Then we have $x = \mathbf{0}$, and:

$$p(0, \ldots, 0) = 1 - \prod_{S \in C} \left(1 - \sum_{i \in S} 0\right) = 0. \tag{19.11}$$

So, regardless of the distribution from which we pick $C$, we have

$$\Pr_C [p(\mathbf{0}) = 0] = 1. \tag{19.12}$$

Next, suppose $x_1 \vee \ldots \vee x_k = 1$. We have $p(x) = 1$ iff the product term is zero. The product term is zero iff the sum in some factor is 1. And that, in turn, happens iff there's some $S \in C$ which includes *exactly one* $x_i$ which is 1. Formally, for any $x \in \{0, 1\}^k$, we want the following to be true with high probability.

$$\exists S \in C. (|\{i \in S : x_i = 1\}| = 1) \tag{19.13}$$

Given that we don't want $C$ to be very large (so that the degree of the polynomial is small), we'll have to pick $C$ very carefully. In order to accomplish this, we turn to the Valiant-Vazirani reduction, which you may recall from Lecture 7:

**Lemma 84 (Valiant-Vazirani)** *Let $A \subseteq \{1, \ldots, k\}$, let $a$ be such that $2^a \leq |A| \leq 2^{a+1}$, and let $H$ be a family of pairwise independent hash functions of the form $h : \{1, \ldots, k\} \to \{0, 1\}^{a+2}$. Then if we pick $h$ at random from $H$, there is a constant probability that there is a unique element $i \in A$ such that $h(i) = \mathbf{0}$. Precisely,*

$$\Pr_{h \sim H} [|\{i \in A : h(i) = \mathbf{0}\}| = 1] \geq \frac{1}{8} \tag{19.14}$$

With this as a guide, we'll define our collection $C$ in terms of pairwise independent hash functions. Let $t > 0$ be a value that we will set later in terms of the approximation parameter $\epsilon$. Then we let $C = \{S_{a,j}\}_{a \in \{0, \ldots, \log k\}, j \in \{1, \ldots, t\}}$ where the sets $S_{a,j}$ are defined as follows.

- For $a \in \{0, \ldots, \log k\}$:
  - For $j \in \{1, \ldots, t\}$:
    - Pick random pairwise independent hash function $h_{a,j} : \{1, \ldots, k\} \to \{0, 1\}^{a+2}$
    - Define $S_{a,j} = h^{-1}(\mathbf{0})$. That is, $S_{a,j} = \{i : h(i) = \mathbf{0}\}$.

Now consider any $x \neq \mathbf{0}$ which we're feeding to our OR-polynomial $p$. Let $A$ be the set of bits of $x$ which are 1, i.e., $A = \{i : x_i = 1\}$, and let $a$ be such that $2^a \leq |A| \leq 2^{a+1}$. Then we have $a \in \{0, \ldots, \log k\}$, so $C$ includes $t$ sets $S_{a,1}, \ldots, S_{a,t}$. Consider any one such $S_{a,j}$. By Valiant-Vazirani, we have

$$\Pr_{h_{a,j} \sim H} [|\{i \in A : h_{a,j}(i) = \mathbf{0}\}| = 1] \geq \frac{1}{8} \tag{19.15}$$

which implies that

$$\Pr_{h_{a,j} \sim H} [|\{i \in A : i \in S_{a,j}\}| = 1] \geq \frac{1}{8} \tag{19.16}$$

so the probability that there is some $j$ for which $|S_{a,j} \cap A| = 1$ is at least $1 - \left(\frac{7}{8}\right)^t$, which by the reasoning above tells us that

$$\Pr_p [p(x) = x_1 \vee \ldots \vee x_k] \geq 1 - \left(\frac{7}{8}\right)^t. \tag{19.17}$$

Now, to get a success probability of $1-\epsilon$ as required by the lemma, we just pick $t = O(\log \frac{1}{\epsilon})$. The degree of $p$ will then be $|C| = t(\log k) = O((\log \frac{1}{\epsilon})(\log k))$, which satisfies the degree requirement of the lemma. $\Box$

Note that given this lemma, we can also approximate AND with an exponentially low degree polynomial. Suppose we have some $G$ which approximates OR within $\epsilon$ as above. Then we can construct $G'$ which approximates AND by drawing $g$ from $G$ and returning $g'$ such that

$$g'(x_1, \ldots, x_k) = 1 - g(1 - x_1, \ldots, 1 - x_k). \tag{19.18}$$

Any such $g'$ has the same degree as $g$. Also, for a particular $x \in \{0,1\}^k$, $g'$ clearly computes AND if $g$ computes OR, which happens with probability at least $1 - \epsilon$ over our choice of $g$.

## 19.4 References

The proof of the Parity lower bound using polynomials is due to Razborov [Raz87] and Smolensky [Smo87]. The original proof was different, and we will present it next.

# Lecture 20

# Lower Bound for Parity Using Random Restrictions

In this lecture we give an alternate proof that parity $\notin \mathbf{AC_0}$ using the technique of random restrictions. This is the original method that was used to prove parity $\notin \mathbf{AC_0}$.

## 20.1 Random Restrictions

A *restriction* fixes some inputs of a circuit to constant values, and leaves other inputs free. More formally, a restriction is a function $\rho : \{1, \ldots, n\} \to \{0, 1, *\}$. We apply a restriction $\rho$ to a circuit $C$ with $n$ inputs to obtain a restricted circuit $C_\rho$ as follows:

- For each $i \in \{1, \ldots, n\}$:
  - If $\rho(i) = 0$, set the $i$th input of $C$ to 0.
  - If $\rho(i) = 1$, set the $i$th input of $C$ to 1.
  - If $\rho(i) = *$, leave the $i$th input of $C$ as a free variable.

Now let's see how restrictions are used to prove our main theorem:

**Theorem 85** *There does not exist a constant-depth, unbounded-fanin, polynomial-size circuit which computes the parity function of its inputs.*

Proof Idea: We'll consider only circuits which at the first level consist only of ORs, at the second level consist only of ANDs, and continue alternating in this way. This is depicted in Figure 20.1. (Any circuit can be coverted to this form with at most a factor of 2 increase in the depth.)

The proof will be by contradiction. First, we show that parity circuits of depth 2 must have exponential width (Lemma 86). Next, we suppose by way of contradiction that we have an $\mathbf{AC_0}$ circuit $C$ of some constant depth $d$ which computes parity. We give a way of squashing $C$ down to depth $d-1$ while still computing parity on many variables. We can repeat the method $d-2$ times to obtain a parity circuit of depth 2 and subexponential size, which contradicts Lemma 86.

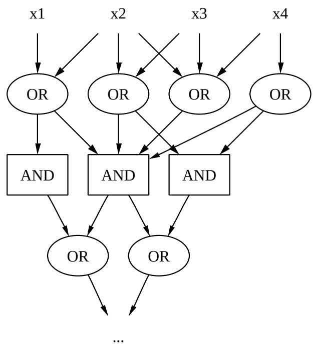


Figure 20.1: A circuit with alternating levels of ANDs and ORs.

Restrictions enter in our method of circuit squashing. We apply a random restriction $\rho_1$ to $C$'s inputs, obtaining $C_{\rho_1}$. We prove that the top level OR gates of $C_{\rho_1}$ are likely to only depend on a constant number of inputs, but a good number of variables still remain free (Corollary 88). $C_{\rho_1}$ will compute parity (or the inverse of parity, which is just as good) on those free variables. Then we show that after applying a second random restriction $\rho_2$ and obtaining $C_{\rho_1\rho_2}$, the second-level AND gates are also likely to depend on only a constant number of inputs (Lemma 89). Now we flip the first two layers, which we can do with at most an exponential size increase of those layers (Lemma 90), which is fine because the size of both layers was made constant by the two rounds of restrictions. After the flip, the second and third layers are both ORs, so we can collapse them both into a single layer, obtaining a circuit of depth $d-1$ that computes parity on many variables.

Statements and proofs of the lemmas mentioned above follow.

**Lemma 86** *If a DNF or a CNF computes parity of $n$ variables, then:*

1. *Each term includes all $n$ variables, and*
2. *There are at least $2^{n-1}$ terms.*

Proof: We will prove the lemma for CNFs, which have OR gates at their top level and a single AND of all the ORs at the second level. The proof for DNFs is quite similar.

For any CNF circuit $C$:

1. *Each term includes all $n$ variables:* Suppose by way of contradiction that $C$ has some term $t$ which does not depend on some variable $x_i$. Then when all inputs to $t$ are 0, $t$ outputs 0 and the single AND gate on the next level outputs 0, which is the output of the whole circuit. Now flip the value of $x_i$. The output of $t$ is still 0, and thus the output of $C$ has not changed. But since we've only changed one variable, the parity has flipped. Alas, we have a contradiction! So every term must depend on all variables.

2. *There are at least $2^{n-1}$ terms:* To compute parity, $C$ must output 0 on $2^{n-1}$ different settings of the input variables. $C$ outputs 0 only when one of the terms (OR gates)

outputs 0. But each OR gate outputs 0 on exactly one setting of the input variables. Thus, $C$ must have at least $2^{n-1}$ terms.

□

**Lemma 87** *Let $\rho$ be a random restriction defined as*

$$\rho(i) = \begin{cases} 0 & \textit{with probability } \frac{1}{2} - \frac{1}{\sqrt{n}} \\ 1 & \textit{with probability } \frac{1}{2} - \frac{1}{\sqrt{n}} \\ * & \textit{with probability } \frac{2}{\sqrt{n}} \end{cases} \tag{20.1}$$

*Then an OR or AND gate with $n$ inputs restricted with $\rho$ depends on at most $c$ inputs with probability at least $1 - \frac{1}{n^{c/3}}$.*

Proof Idea: We'll break the proof into two cases. (1) Large OR gates will have some input fixed at 1 w.h.p., and thus the entire gate can be replaced with a constant 1. Similarly, large AND gates can be replaced with 0 w.h.p. (2) Small gates will w.h.p. be reduced enough to depend on only $c$ inputs.

Proof:

Case 1: Suppose the gate has at least $c \log n$ inputs before applying the restriction. Then:

$$\begin{aligned}
\Pr_\rho[\text{gate depends on} \le c \text{ inputs}] &\ge \Pr_\rho[\text{gate becomes constant}] \\
&\ge \Pr_\rho[\text{some input set to 1 (OR gate) or 0 (AND gate)}] \\
&\ge 1 - \left(\frac{1}{2} + \frac{1}{\sqrt{n}}\right)^{c \log n} \\
&\ge 1 - \left(\sqrt{\frac{1}{2}}\right)^{c \log n} \quad \text{(for large enough } n\text{)} \\
&= 1 - \frac{1}{n^{c/2}} \\
&\ge 1 - \frac{1}{n^{c/3}}.
\end{aligned}$$

Case 2: Suppose the gate has at most $c \log n$ inputs. Then:

$$\begin{aligned}
\Pr_\rho[\text{gate depends on} \le c \text{ inputs}] &\ge 1 - \sum_{t=c+1}^{c \log n} \binom{c \log n}{t} \left(\frac{2}{\sqrt{n}}\right)^t \\
&\ge 1 - (c \log n)^c \left(\frac{1}{\sqrt{n}}\right)^c \\
&\ge 1 - \frac{1}{n^{c/3}} \text{ (for large enough } n\text{)}.
\end{aligned}$$

□

**Corollary 88** *If a circuit has $n^k$ gates, then there is a restriction with $\frac{\sqrt{n}}{2}$ free variables such that all top-level gates have fan-in at most $4k$.*

Proof: We apply Lemma 87 with $c = 4k$, obtaining a circuit $C_\rho$ such that

$$\Pr\left[\text{some gate of } C_\rho \text{ depends on } > 4k \text{ variables}\right] < n^k \frac{1}{n^{4k/3}} = n^{-1/3} \ll 1 \tag{20.2}$$

and

$$\Pr\left[C_\rho \text{ depends on } < \frac{\sqrt{n}}{2} \text{ variables}\right] \approx 2^{O(-\sqrt{n})} \ll 1 \tag{20.3}$$

which follows from a Chernoff bound. This guarantees the existance of a circuit which satisfies the corollary. □

**Lemma 89** *Given any c-CNF $C$ (in which each OR depends on at most $c$ variables) and any $k > 0$, there is a constant $b(c,k)$ such that after applying the random restriction $\rho$ of Lemma 87 to $C$, with probability more than $1 - \frac{1}{n^k}$, $C_\rho$ depends on at most $b(c,k)$ variables.*

Proof: The proof is by induction on $c$.

Base case: $c = 1$. The OR gates each only have 1 input, so we can replace the entire CNF with a single AND gate. Then by Lemma 87, Lemma 89 is satisfied when $b(1,k) = 3k$.

Inductive case: Suppose the lemma holds for $c - 1$; we'll prove that it's true for $c$. Let $M$ be a maximal set of disjoint clauses. That is, the clauses in $M$ share no variables (disjointedness), and every clause not in $M$ shares a variable with some clause in $M$ (maximality). (You can think of this as picking a maximal matching in the hypergraph whose nodes are the circuit's variables and whose hyperedges are the sets of variables used by each clause.) One of two cases is true:

Case 1: $|M| \geq 2^c k \log n$. Then:

$$\begin{aligned}
\Pr_\rho\left[C \text{ depends on} \leq b(c,k) \text{ inputs}\right] &\geq \Pr_\rho\left[C \text{ becomes constant}\right] \\
&\geq \Pr_\rho\left[\text{some OR gate unsatisfied}\right] \\
&\geq 1 - \left(1 - \left(\frac{1}{2} - \frac{1}{\sqrt{n}}\right)^c\right)^{|M|} \\
&\approx 1 - e^{-\frac{1}{2^c}|M|} \\
&> 1 - \frac{1}{n^k}.
\end{aligned}$$

Note that in the third inequality we use the fact that the clauses of $M$ are independent: the probability that a particular OR gate in $M$ is unsatisfied is independent of the probability that any other OR gate in $M$ is unsatisfied.

Case 2: $|M| \leq 2^c k \log n$. Let $V$ be the set of variables in the clauses of $M$, so $|V| \leq c|M| = c2^c k \log n$. We'll first show that the restriction reduces $V$ to constant size:

$$\begin{aligned}\Pr_\rho[\leq i \text{ variables of } V \text{ stay free}] &\geq 1 - \binom{2^c k \log n}{i}\left(\frac{1}{\sqrt{n}}\right)^i \\ &\geq 1 - \frac{1}{n^{i/3}} \text{ (for large enough } n).\end{aligned}$$

So to get the probability greater than $1 - \frac{1}{n^k}$ as required by the lemma, we choose $i = 4k$. But we're not yet done: we've shown that the gates of $M$ shrink to constant size, but we have said nothing about the gates not in $M$. To do that, recall that every clause not in $M$ shares a variable with a clause in $M$, so if we restrict the remaining $4k$ free inputs of $M$, obtaining $C'_\rho$, then every clause in $C'_\rho$ will have some input fixed. $C'_\rho$ is thus a $(c-1)$-CNF and we can apply the inductive hypothesis: $C'_\rho$ depends on at most $b(c-1, k+1)$ variables. Now, there are $2^{4k}$ ways to set the $4k$ variables, so $C_\rho$ depends on at most

$$b(c,k) = 4k + 2^{4k} b(c-1, k+1) \tag{20.4}$$

variables.

□

**Lemma 90 (The Switching Lemma)** *A CNF or DNF of $c$ variables can be represented as a DNF or a CNF, respectively, of size at most $2^c$.*

We omit the proof of this lemma, but note that we can actually prove a stronger fact: *any* function of $c$ variables can be written as a DNF or as a CNF of size at most $2^c$.

## 20.2 Håstad's Result

This section is devoted to a sketch of Håstad's tight result for the complexity of parity with bounded depth circuits. We begin by stating the key theorem:

**Theorem 91** *(Switching Lemma) Suppose $f$ is a $k$-CNF*[1] *over the variables $x_1, \ldots, x_n$. Pick at random a restriction that leaves a fraction $p$ of the variables unfixed. For each of the $n(1-p)$ variables that are fixed, independently hardwire 0 or 1 as that variable's value. Then for every $t$,*

$$\mathbf{Pr}[\textit{after the restriction } f \textit{ can be expressed as a } t\textit{-CNF and a } t\textit{-DNF}] \geq 1 - (7pk)^t.$$

The theorem states that, after making the specified sort of restriction, we get a formula that can be expressed as a simpler DNF (if the original $f$ was a CNF) or a simpler CNF (if the original $f$ was a DNF) with exponentially high probability.

We state one more useful result before getting on to the analysis:

[1]We can take $f$ to be a $k$-DNF as well by taking complements.

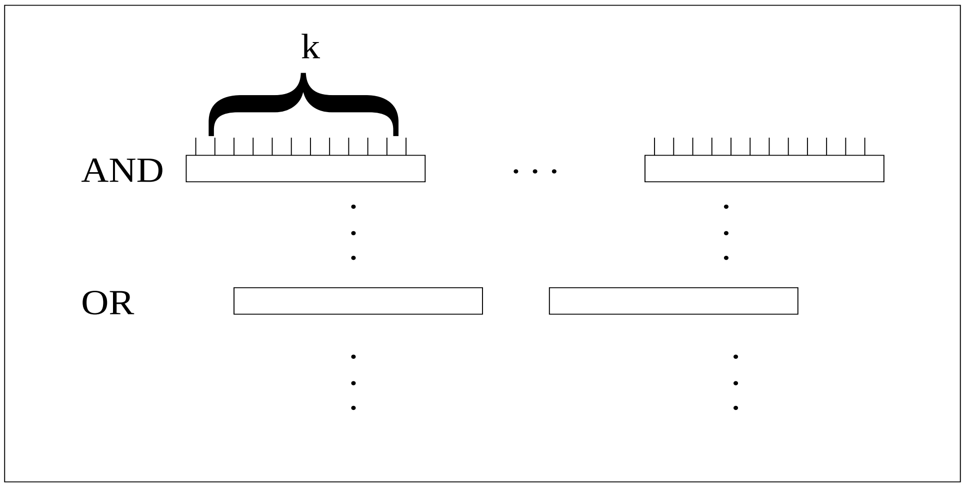


Figure 20.2: The first two levels of the circuit $C$. By the Switching Lemma, we can transform the AND gates of the top level to OR gates with high probability by making a random restriction.

**Lemma 92** *If $f$ is a $k$-CNF (or a $k$-DNF) for $x_1 \oplus x_2 \oplus \cdots \oplus x_m$, then every term has size $n$.*

Now let $C$ be a depth $d$ circuit for parity of size $S$. Each level of $C$ will consist entirely of AND gates or of OR gates, starting with AND gates at the top level. Each of the AND gates at the top level is a 1-CNF, each clause being a single variable. We can apply the Switching Lemma with $t = \log S$ and $p = 1/14$ so that each of these top-level gates becomes a $\log S$-DNF with probability at least $1 - 1/S$ (see Figure 1).

Since the AND gates of the top level have been switched to ORs, we can collapse the first two levels into one level of ORs to get a circuit that computes the parity of $n/14$ inputs such that the top gates have fan-in at most $\log S$ (see Figure 2).

Next we apply the Switching Lemma to the (new) top level of gates, each of which is a $\log S$-DNF (so $k = \log S$), with $p = 1/(14 \log S)$ and $t = \log S$ (see Figure 3). Thus each of these OR gates can be computed by a $\log S$-CNF with probability at least

$$\begin{aligned} 1 - \left(7 \frac{\log S}{14 \log S}\right)^{\log S} &= 1 - \left(7 \frac{1}{14}\right)^{\log S} \\ &= 1 - \frac{1}{2^{\log S}} \\ &= 1 - \frac{1}{S}, \end{aligned}$$

and after collapsing the two consecutive levels of ANDs, we get a depth $d-1$ circuit that computes the parity of $n/(14(14 \log s))$ inputs such that the top gates have fan-in at most $\log S$.

Now we continue applying this procedure recursively until we get a depth 2 circuit that computes the parity of $n/(14(14 \log S)^{d-2})$ inputs such that the top gates have fan-in at most $\log S$. By the lemma, the fan-in at the top must be at least the number of variables,

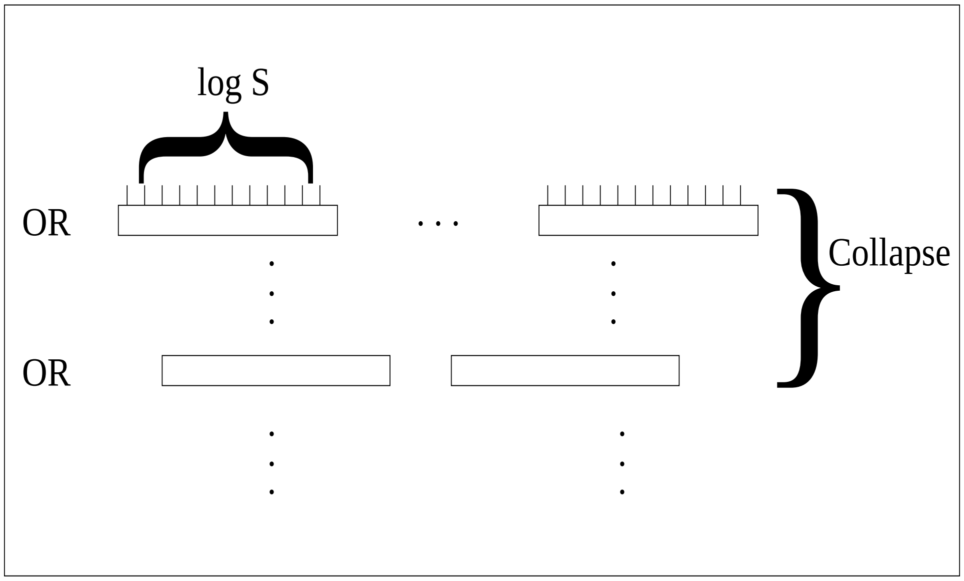


Figure 20.3: Applying the Switching Lemma to the AND gates transforms them to OR gates, so we can collapse the first two levels to form one level of ORs.

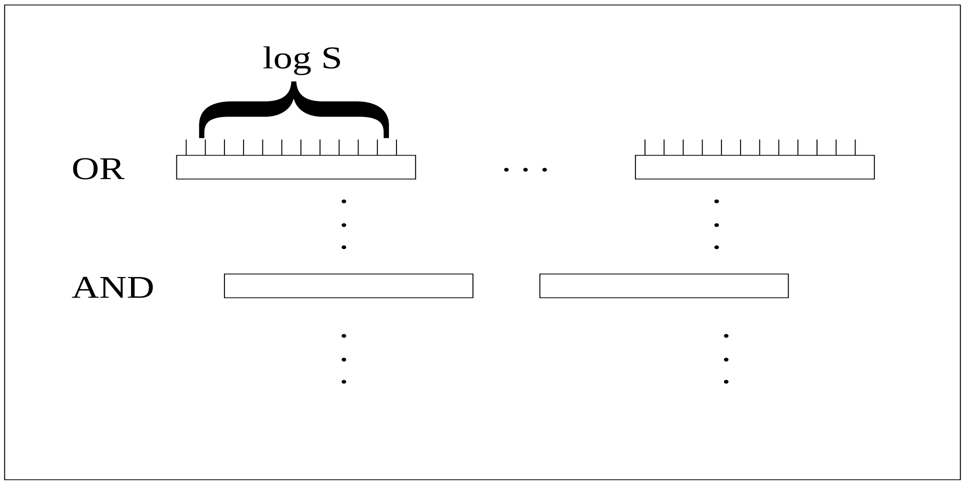


Figure 20.4: We use the Switching Lemma to transform the (new) first level of ORs to ANDs, and then we collapse the two consecutive levels of ANDs to one new level.

so

$$\log S \geq \frac{n}{14(14 \log S)^{d-2}}$$

or

$$S \geq 2^{\frac{1}{14} n^{1/d-1}}.$$

## 20.3 References

The idea of using random restrictions, as well as the first proof that Parity is in $\mathbf{AC_0}$, is due to Furst, Saxe and Sipser [FSS84]. The lower bound was improved to exponential by Yao [Yao85], and the optimal lower bound is due to Håstad [Hås86].

# Lecture 21

# Proof Complexity

In this lecture and in the next one we are going to discuss proof complexity, which is related to the question of **NP** vs. **coNP** in much the same way that circuit complexity is related to the question of **P** vs. **NP**.

Recall that **NP** = **coNP** if and only if there is some **coNP**-complete problem—the complement of an **NP**-complete problem—that is in **NP**. In particular, equality will hold if **UNSAT** is in **NP**, where **UNSAT** is the problem of determining if a Boolean formula $\varphi$ is a contradiction—that is, if $\varphi(x_1, \dots, x_n) = 0$ for all truth assignments to the variables $x_1, \dots, x_n$ of $\varphi$. If **UNSAT** is in **NP**, then we can make the following precise statements:

1. There is a polynomial-time verifier $V$ and a polynomial bound $p$ such that for all unsatisfiable formulas $\varphi$, there exists a proof $\pi$, with $|\pi| \leq p(|\varphi|)$, such that $V(\varphi, \pi)$ accepts, and

2. For all satisfiable formulas $\varphi$ and all $\pi$, $V(\varphi, \pi)$ rejects (even if the length of $\pi$ is not bounded by some polynomial in $|\varphi|$).

These statements follow immediately from the definition of **NP**.

For the purposes of studying proof complexity, we will consider a situation in which the requirement of a polynomial bound $p$ in (1) is discarded. Thus, we say that $V$ is a **sound and complete** proof system for **UNSAT** if

1. (**Soundness**) If $\varphi$ is a satisfiable formula, then for all $\pi$, $V(\varphi, \pi)$ rejects,

2. (**Completeness**) If $\varphi$ is an unsatisfiable formula, then there is a $\pi$ such that $V(\varphi, \pi)$ accepts, and

3. (**Efficiency**) $V$ runs in polynomial time.

We can now state a necessary and sufficient condition for **NP** to be different from **coNP** in terms of these sound and complete proof systems: **NP** $\neq$ **coNP** if and only if for every sound and complete proof system for **UNSAT**, there is an infinite sequence of unsatisfiable formulas whose shortest proof sizes grow superpolynomially.

What we are going to do now is to fix a natural proof system for **UNSAT** and then attempt to show unconditionally that there are formulas requiring superpolynomial proofs. This general approach can be likened to the approach physicists take when investigating physical theory: a physical theory offers some predictions about phenomena that can be observed in nature, and we check to see that those predictions are actually correct. If they are, we take that to be evidence in support of the theory. In this case, the "theory" is that **NP** $\neq$ **coNP**, and this theory predicts that for every proof system for **UNSAT**, there will be formulas requiring proofs of superpolynomial length.

Notice that any algorithm $A$ for **SAT**, whether it runs in polynomial time or not, induces a proof system for **UNSAT**. The proof that $\varphi$ is unsatisfiable is a description of a rejecting computation of $A$ on input $\varphi$. Hence a lower bound on the proof length of a proof in this system automatically gives a lower bound on the running time of $A$.

## 21.1 Resolution Proof System

The proof system for **UNSAT** that we will investigate is Resolution. Given a CNF formula $\varphi = C_1 \wedge C_2 \ldots \wedge C_m$, where each $C_i$ is a clause, a proof of the unsatisfiability of $\varphi$ (also called a "refutation" of $\varphi$) is a sequence of clauses $(D_1, D_2, \ldots, D_t)$ such that

1. Each $D_i$ is either
   (a) A clause of the original formula $\varphi$ (i.e., $C_j$ for some $j$ between 1 and $m$) or
   (b) A clause obtained from $D_j$ and $D_h$, $j < i$ and $h < i$, using the **Resolution rule**
   $$\frac{x \vee D,\ \overline{x} \vee D'}{D \vee D'}, \quad \text{and}$$
2. $D_t$ is $\underline{\mathbf{0}}$ (the false empty clause).

As an example of how Resolution works, consider the formula

$$\varphi = x_1 \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge \overline{x_3};$$

we can construct a proof as follows (see Figure 4):

1. Let $D_1 = x_1$, $D_2 = \overline{x_1} \vee x_2$, $D_3 = \overline{x_2} \vee x_3$, and $D_4 = \overline{x_3}$, so the first four clauses in the refutation are just the original clauses of $\varphi$.
2. From $D_1 = x_1$ and $D_2 = \overline{x_1} \vee x_2$, using the resolution rule we can deduce $D_5 = x_2$.
3. From $D_3 = \overline{x_2} \vee x_3$ and $D_5 = x_2$, using the resolution rule we can deduce $D_6 = x_3$.
4. From $D_4 = \overline{x_3}$ and $D_6 = x_3$, using the resolution rule we deduce the empty clause $D_7 = \underline{\mathbf{0}}$.

The proof that $\varphi$ is unsatisfiable is then just the sequence of clauses $(D_1, D_2, \ldots, D_7)$. The reason that this is a proof is that the Resolution rule is sound: any assignment to the variables of $\varphi$ that satisfies $\varphi$ will also satisfy any clauses derived using the Resolution rule. Thus, if we derive a clause that cannot be satisfied by any assignment, it follows that $\varphi$ can likewise not be satisfied by any assignment to its variables. This reasoning shows that the proof system is sound.

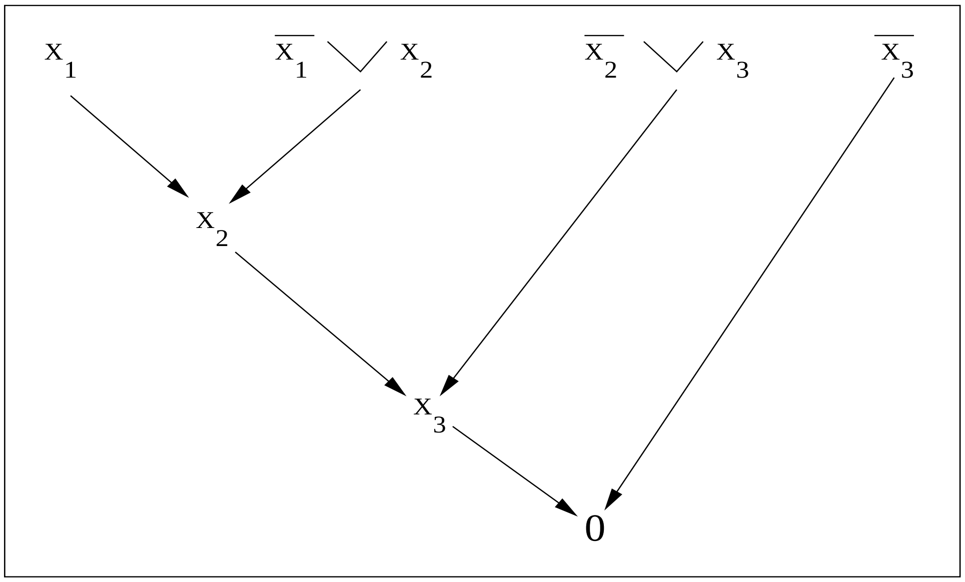


Figure 21.1: A tree-like Resolution proof.

In this particular example, the proof can be pictured as a tree (again see Figure 4), so the Resolution proof for $\varphi$ is called **tree-like**. In general, a proof can be represented as a directed acyclic graph, with the sequence $(D_1, D_2, \ldots, D_t)$ being a topological sort of that graph.

Now we want to show that Resolution is complete (i.e., every unsatisfiable formula has a refutation). In doing so, it will be useful to allow the use of the **weakening rule**

$$\frac{D}{D \vee x}.$$

We lose no generality in adding the weakening rule because, as it turns out, any proof that uses the rule can be converted to a proof that doesn't without adding to the proof's length:

**Lemma 93** *If there is a proof for $\varphi$ of length*[1] *$k$ using the Resolution and weakening rules, then there is a proof for $\varphi$ of length at most $k$ using the Resolution rule only.*

We will not prove this result here.

Now let $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be an unsatisfiable formula over the $n$ variables $x_1, x_2, \ldots, x_n$. Using the weakening rule, construct all clauses of length $n$ that can be obtained from some $C_i$, $i = 1, 2, \ldots, m$, by adding literals. We claim that we can derive all length $n$ clauses this way.

To see this, suppose that the clause

$$(x_1 = a_1) \vee (x_2 = a_2) \vee \ldots \vee (x_n = a_n) \qquad (*)$$

[1]The number of clauses in the proof

cannot be derived using weakening. Clearly the assignment

$$\left.\begin{array}{l} x_1 = 1 - a_1 \\ x_2 = 1 - a_2 \\ \dots \\ x_n = 1 - a_n \end{array}\right\} \qquad (**)$$

contradicts $(*)$. Now take a clause $C_i \in \varphi$ over the variables $x_{i_1}, x_{i_2}, \dots, x_{i_l}$, where each $x_{i_j}$ is an $x_k$ for some $k$. A clause is contradicted by exactly one assignment, so if $C_i$ is contradicted by the restriction of $(**)$ to $x_{i_1}, x_{i_2}, \dots, x_{i_l}$, it follows that $C_i$ is a subset of $(*)$, but this is impossible since we assumed that $(*)$ cannot be derived from any $C_j$ by adding literals. Hence $(**)$ satisfies every clause of $\varphi$, but this is a contradiction since we assumed that $\varphi$ is unsatisfiable. (In other words, a length $n$ clause that is not derivable using weakening describes a satisfying assignment for $\varphi$).

Having derived all length $n$ clauses, we can use the Resolution rule to derive all length $n-1$ clauses. Continuing this process, we eventually derive the clauses $x$ and $\overline{x}$ for a variable $x \in \{x_1, x_2, \dots, x_n\}$, and one more application of the Resolution rule gives $\underline{\mathbf{0}}$, showing that $\varphi$ has a refutation and that the proof system is indeed complete.

## 21.2 Width of a Resolution Proof

If $(D_1, D_2, \dots, D_t)$ is a Resolution proof for $\varphi$, then $t$ is the **length**[2] of the proof and the size of the shortest clause in the derivation is the **width** of $\varphi$ (we only count clauses that are derived—not clauses that are in $\varphi$). It turns out to be easier to prove width lower bounds than length lower bounds directly. That this approach can still lead to strong lower bounds for length is the content of the following theorem:

**Theorem 94** *Fix an unsatisfiable k-CNF formula $\varphi$ over n variables. Let w be the minimal width of a proof for $\varphi$, let $L_T$ be the length of the shortest tree-like Resolution proof, and let L be the length of the shortest general Resolution proof. Then*

$$w \leq \log L_T + k$$
$$w \leq O(\sqrt{n \log L}) + k.$$

We first state a lemma that will be useful in the proof of the theorem.

**Lemma 95** *Let $\varphi$ be as above, $x$ a variable in $\varphi$, and $a \in \{0, 1\}$. Suppose $\varphi_{x=a}$ is a k-CNF, the width of $\varphi_{x=1-a}$ is at most $d - 1$, and the width of $\varphi_{x=a}$ is at most $d$. The $\varphi$ has width no greater than* $\max d, k$.

We will prove the lemma and the second part of this theorem in the next lecture. For now, we focus on the first part of the theorem.

PROOF: The strategy of the proof is to show that if $\varphi$ has a short proof, then it has a proof of small width. We will show this by induction on $n$, the number of variables, and $b$, where

[2]We will also refer to the length of a proof as its **size** since the length is the number of nodes in the proof DAG.

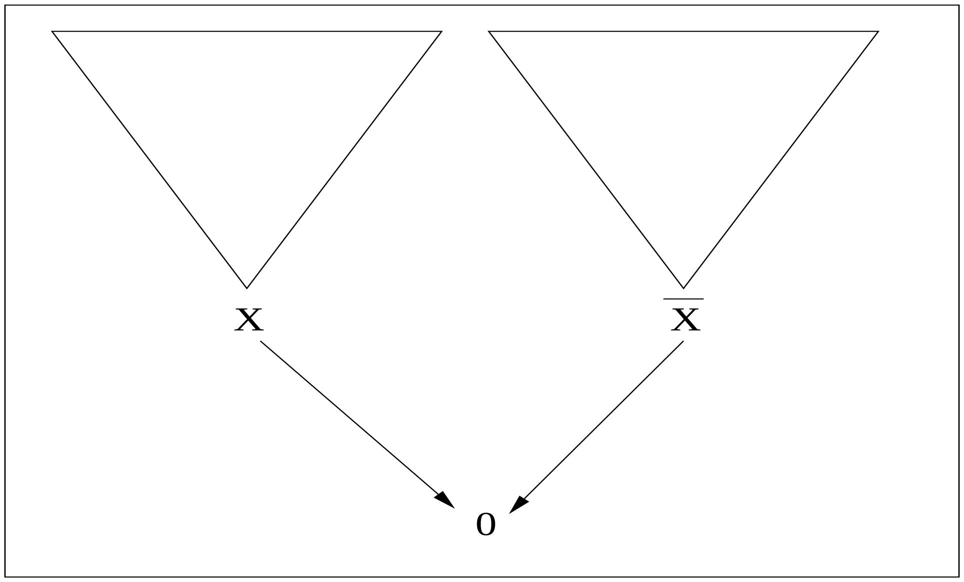


Figure 21.2: The tree-like Resolution proof of $\varphi$ consists of two subtrees which are derivations of $x$ and $\overline{x}$. Since the size of the proof is at most $2^b$, one of these subtrees must have size at most $2^{b-1}$.

$b$ is an integer such that $\varphi$ has a tree-like proof of size bounded above by $2^b$. The claim is that if $\varphi$ has a tree-like proof of size at most $2^b$, then it has a proof of width at most $b+k$.

If $b=0$, then the proof that $\varphi$ is unsatisfiable is ($\underline{\mathbf{0}}$), in which case the claim is obvious.

If $n=1$, then the only possible literals are $x$ and $\overline{x}$, so any proof will have width 0 (or 1, depending on the conventions one chooses). Again the claim is immediate.

Now suppose $\varphi$ has a tree-like proof of size $2^b$. The last step of the proof must by a derivation of $\underline{\mathbf{0}}$ from $x$ and $\overline{x}$, where $x$ is a variable in $\varphi$ (see Figure 5). Since the total size of the proof is $2^b$, the tree-like derivation of $x$ or the tree-like derivation of $\overline{x}$ must have size no greater than $2^{b-1}$; without loss of generality, assume that the derivation of $x$ is the small one.

Suppose that we substitute $x=0$ into all the clauses of $\varphi$ and all the clauses we derived from $\varphi$. The derivation is still correct since the Resolution rule still applies after we make this substitution, so the tree-like derivation of $x$ becomes a derivation of 0, i.e. a proof for $\varphi_{x=0}$. Hence $\varphi_{x=0}$ has a tree-like proof of size at most $2^{b-1}$, and we can apply the inductive hypothesis to conclude that there is also a proof of $\varphi_{x=0}$ of width at most $b+k-1$.

Next make the substitution $x=1$ into all the clauses of $\varphi$ and all the clauses derived from $\varphi$. Then, as above, we get a proof of $\varphi_{x=1}$ of size at most $2^b$. But since there is one less variable in $\varphi_{x=1}$, by induction on $n$ there is a proof for $\varphi_{x=1}$ of width at most $b+k$.

The desired conclusion now follows from the lemma. $\square$

## 21.3 References

The first superpolynomial lower bounds for resolution proof length were proved by Tseitin [Tse70]. The first exponential lower bounds are due to Haken [Hak85]. The notion of width is due to Ben-Sasson and Wigderson [BSW01].

# Lecture 22

# Lower Bounds on Resolution Proof Length

## 22.1 Width and Proof Complexity

### 22.1.1 The Main Theorem

In general, a derivation $(\varphi \rightarrow D)$ can have different proofs and by its width $w(\varphi \rightarrow D)$, we mean the width of its minimum-width proof. The following theorem is about $w(\varphi \rightarrow 0)$ (written sometimes as simply $w(\varphi)$), where 0 is the empty false clause and $\varphi$ is unsatisfiable. A proof that derives $(\varphi \rightarrow 0)$ is called a refutation of $\varphi$.

**Theorem 96** (Main Width-Length Theorem) *Let $\varphi$ be a unsatisfiable $k$-CNF formula on $n$ variables. Let $L_T$ and $L$ be the length of the shortest tree-like and shortest general refutation of $\varphi$ respectively. Then, the width of the minimum-width refutation of $\varphi$, $w(\varphi)$ is related to $L_T$ and $L$ as:*

$$w(\varphi) \leq log\, L_T + k \tag{22.1}$$

$$w(\varphi) \leq O(\sqrt{n\, log\, L}) + k \tag{22.2}$$

This theorem is an attempt at simplifying the study of proof complexity [BSW01]. Instead of working directly with $L_T$ and $L$, we work on proving linear lower bounds on $w(\varphi)$ to get exponential lower bounds on $L_T$ and $L$. That is, if $w(\varphi) = \Omega(n)$ and $k = o(n)$, then by Relations (22.1) and (22.2), $L_T$ and $L = 2^{\Omega(n)}$. If we replace $n$ with $n^{\epsilon}$, we can't say anything about $L$ using the sub-linear lower bound on $w(\varphi)$, but we can still get a sub-exponential lower bound on $L_T$.

We proved Relation (22.1) last time, using a simple induction argument on the structure of the tree-like refutation. We can't mimic that proof for Relation (22.2) though, since a single clause in a general refutation can be used later in more than one derivation, leading to a *dag* structure. We prove Relation (22.2) using the following technical lemma.

**Lemma 97** *Let $\pi_d$ be the set of clauses of size strictly greater than d in a general refutation $\pi$, of a unsatisfiable k-CNF formula $\varphi$ on n variables. If $|\pi_d| < (1 - \frac{d}{2n})^{-b}$ for some parameter b, then the width of $\pi$ is atmost $d + b + k$.*

The clauses in $\pi_d$ are in a sense the problematic clauses in $\pi$, but if there are not too many of them, we can safely bound the width of $\pi$. This lemma is true for any value of $d$ and the proof of Relation (22.2) uses the value of $d$ that optimizes this bound.

Proof: (of Relation (22.2) of Width-Length Theorem) Let $\pi$ be the shortest general refutation of $\varphi$ with length $|\pi| = L$. We obtain the result by using Lemma 97 on the trivial bound $|\pi_d| < L$.

$$\begin{aligned}
w(\varphi) &\leq w(\pi) \\
&\leq d + b + k && \text{(where b is from, } L = (1 - \tfrac{d}{2n})^{-b}) \\
&= d + log_{\frac{1}{1-\frac{d}{2n}}} L + k \\
&\leq d + log_{1+\frac{d}{2n}} L + k && \text{(using, } \tfrac{1}{1-\epsilon} \approx 1{+}\epsilon) \\
&\leq d + O(\tfrac{2n}{d} \, log \, L) + k && \text{(using, } log_{1+\epsilon} x \approx \tfrac{1}{\epsilon} \, log \, x) \\
&= d + O(\tfrac{n}{d} \, log \, L) + k
\end{aligned}$$

Therefore, when $d = \frac{n}{d} \, log \, L$ (i.e., $d = \sqrt{n \, log \, L}$) in the relation above, we get the desired upper bound on $w(\varphi)$. □

### 22.1.2 Proof of the technical lemmas

In this section, we prove two technical lemmas used in the proof of Width-Length theorem. Lemma 98 was used in proving Relation (22.1) last time and it will also be used in the proof of Lemma 97 this time. We need more notations before proceeding. For a variable $x$ in $\varphi$, let $x^1$ and $x^0$ be aliases for the literals $x$ and $\overline{x}$ respectively. Also, let $\varphi_{x=a}$ be the restriction of $\varphi$ to $x = a$, for $a \in \{0, 1\}$. That is, $\varphi_{x=a}$ is the set of clauses obtained from $\varphi$ after removing all clauses that have $x^a$ in them and removing $x^{1-a}$ from all clauses that have this literal. The restriction of a clause $D$ to $x = a$, denoted by $D_{x=a}$, is obtained similarly.

**Lemma 98** *If for $a \in \{0, 1\}$, $\varphi_{x=a}$ is a k-CNF formula,*

$$w(\varphi_{x=1-a}) \leq d - 1 \qquad \textit{and} \qquad w(\varphi_{x=a}) \leq d,$$

*then $w(\varphi) \leq max(d, k)$.*

Proof: We will first prove the claim: If $w(\varphi_{x=1-a}) \leq d{-}1$, then $w(\varphi \to x^a) \leq d$. We assume $a = 1$ for notational ease (the proof for general $a$ extends easily). Let $\pi$ be a refutation of $\varphi_{x=0}$ with width atmost $d{-}1$ and let $\phi$ denote the set of clauses in $\varphi$ that have the literal $x$ in them. Then $\phi' = \phi_{x=0} \cap \pi$ denotes the restricted clauses of $\phi$ that were used in the proof $\pi$. Obtain a new proof $\pi'$ by copying all clauses in $\pi$, and adding $x$ back to the clauses in $\phi'$ and its subsequent derivations (to preserve validity). The new proof $\pi'$ derives $x$ directly if $\phi'$ was not an empty set or indirectly via a weakening rule from the final 0 clause otherwise. Thus, $\pi'$ is a valid proof of $(\varphi \to x)$ with width atmost $w(\pi) + 1 = d$.

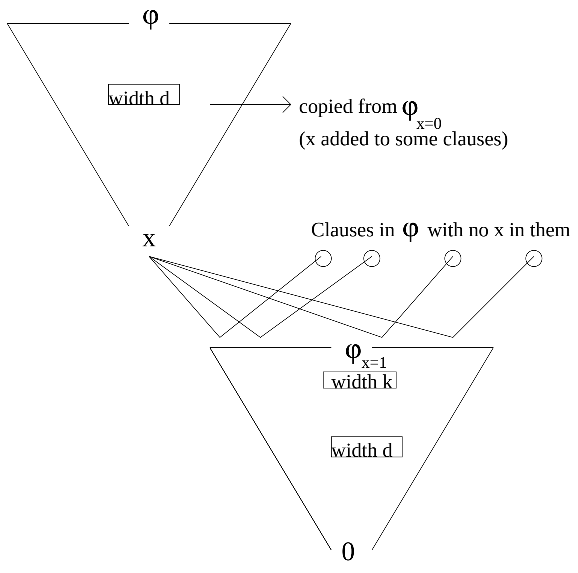


Figure 22.1: Composing $(\varphi \to x)$ with $(\varphi_{x=1} \to 0)$, to obtain a refutation for $\varphi$.

Now we turn to the proof of the lemma, again assuming $a = 1$. We use the claim above to derive $(\varphi \to x)$. We then resolve this derived $x$ with all clauses of $\varphi$ that doesn't contain $x$, to obtain $\varphi_{x=1}$. This is finally fed into the refutation of $\varphi_{x=1}$. See Figure 22.1 for a depiction of this composition. Since $\varphi_{x=1}$ is a $k$-CNF and the width of both refutations used in the composition is atmost d, we get a refutation of $\varphi$ with width atmost $max(d,k)$. □

We now give the proof of Lemma 97, which was used in proving Relation (22.2) of Width-Length theorem.

Proof: (of Lemma 97) The proof is by induction on $b$ and $n$. When $b = 0$, $|\pi_d| < 1$ and the lemma is trivially true for any value of $n$. Similarly for $n = 1$, the lemma is trivially true for any value of $b$, as there are only two literals in the formula. For the induction step, we need a literal that hits many clauses in $\pi_d$ and we will indeed have such a literal by the Pigeon Hole Principle (on viewing the $2n$ literals as holes and the sum-total of the sizes of all clauses in $\pi_d$ as the number of pigeons). Specifically, there must be a literal, say $x$ wlog, that occurs in atleast $\frac{d}{2n}|\pi_d|$ clauses in $\pi_d$ (another way to look at this result is to pick one out of the $2n$ literals in random and see the expected number of clauses in $\pi_d$ that this literal hits).

Using this $x$, we obtain valid refutations of $\varphi_{x=0}$ and $\varphi_{x=1}$ by restricting $\pi$ to $x = 0$ and $x = 1$. The induction step then follows from applying Lemma 98 on these new refutations. By induction on n, we already have $w(\varphi_{x=0}) \leq b + d + k$. So, we just need to prove that $w(\varphi_{x=1}) \leq b + d + k - 1$. Here, we are rescued by the fact that $x$ appears in many clauses in $\pi_d$ and hence the set of clauses that are still wider than $d$ in the restricted refutation $\varphi_{x=1}$

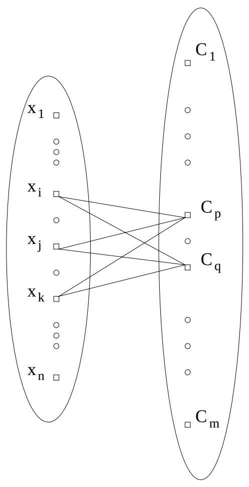


Figure 22.2: Graph $G_\varphi$ associated with a formula $\varphi$. Clause $C_p$ could be $x_i \vee x_j \vee x_k$ and $C_q$ could be $\overline{x_i} \vee x_j \vee \overline{x_k}$.

(call this set $\pi'_d$) is small enough:

$$\begin{aligned} \left|\pi'_d\right| &\leq (1 - \tfrac{d}{2n})\ |\pi_d| \\ &< (1 - \tfrac{d}{2n})^{-(b-1)} && \text{(substituting for } \pi_d\text{)} \end{aligned}$$

Since the parameter is now $(b-1)$, $w(\varphi_{x=1}) \leq b-1+d+k$ by induction on $b$ and we are done with the proof. $\square$

## 22.2 Width Lower Bounds

### 22.2.1 The Setting

We establish lower bounds on width by relating it to some natural combinatorial property. In our case, we will use the expansion property of the bipartite graph associated with a CNF formula. We first define this expansion property and state the main theorem relating width and expansion. Next we use this theorem to show that random $k$-CNFs require exponential-length refutations (whp, due to their expansion properties).

**Definition 29** (Graph associated with a formula) *Let $\varphi$ be a CNF formula on $m$ clauses $C_1, C_2, \ldots, C_m$ and $n$ variables $x_1, x_2, \ldots, x_n$. The graph associated with $\varphi$ (see Figure 22.2) is the undirected bipartite graph $G_\varphi = (L, R, E)$, defined by*

$$\begin{aligned} L &= \{x_1, x_2, \ldots, x_n\}\,, \\ R &= \{C_1, C_2, \ldots, C_m\} \ \textit{and} \\ E &= \{(x_i, C_j) \mid \textit{clause } C_j \textit{contains one of the literals } x_i \textit{or } \overline{x_i}\}\,. \end{aligned}$$

**Definition 30** (Bipartite Expansion) *A CNF formula $\varphi$ on $m$ clauses $C_1, C_2, \ldots, C_m$ is $(r, c)$-expanding if and only if*

$$\forall S \subseteq \{C_1, C_2, \ldots, C_m\} \text{ and } |S| \leq r, \quad |N(S)| \geq (1+c)\, r$$

*where $N(S)$ refers to the neighbourhood of $S$ in $G_\varphi$.*

**Theorem 99** (Main Width-LB (LowerBound) Theorem) *For positive r and c, if $\varphi$ is (r,c)-expanding and unsatisfiable, then*

$$w(\varphi) \geq \frac{rc}{2}.$$

The proof of this theorem is quite interesting and is given in the next two sections. The application of this theorem lies in the context of Width-Length theorem (Theorem 96), since a linear lower bound on $rc$ is now enough to show a exponential lower bound on the length of any refutation of $\varphi$. Infact, the next theorem gives such a lower bound for random $k$-CNFs.

**Theorem 100** *Let $\varphi$ be a random $k$-CNF formula on $n$ variables and $\Delta n$ clauses, constructed by picking each clause with $k$ variables independently at random, from all possible $\binom{n}{k} 2^k$ clauses with repetition. Then, whp $\varphi$ is $\left(\frac{n}{\Delta^{\frac{1}{k-2+\epsilon}}}, \epsilon\right)$-expanding, where the clause density $\Delta$ and the parameter $\epsilon$ with $0 \leq \epsilon \leq \frac{1}{2}$ are constants.*

The proof of this theorem follows from counting arguments and is left as an exercise. For a specific value of $\epsilon$, say $\frac{1}{2}$, we see that random $k$-CNFs are $(\Omega(n), \frac{1}{2})$-expanding and hence by Width-LB and Width-Length theorem, they require exponential-length refutations whp (note that they are also unsatisfiable whp for proper choice of $\Delta$ and $k$).

### 22.2.2 Matching and Satisfiability

This section prepares us for the proof of Width-LB theorem. Consider a subformula S of $\varphi$. If we have a matching of $S$ in $G_\varphi$, then we can satisfy $S$ by assigning proper truth values to the unique non-conflicting variable that each of its clauses is matched to. This simple observation along with Hall's Matching theorem, leads to an interesting result called the Tarsi's theorem. We look at Hall's theorem first.

**Theorem 101** (Hall's Matching Theorem) *For a bipartite graph $G = (L, R, E)$, there exists a matching of vertices $R' \subseteq R$, if and only if*

$$\forall R'' \subseteq R', \; \left|N(R'')\right| \geq \left|R''\right|$$

*where $N(R'')$ refers to the neighbourhood of $R''$ in $G$.*

If there is a matching of vertices in $R'$, then the condition in the theorem is clearly true. But the theorem's strength lies in the sufficiency of the condition, inspite of possible overlaps between the neighbourhood of different $R''$s. Sufficiency of the condition leads to the following two corollaries of Hall's theorem. The first corollary follows from the absence of a matching (in $G_\varphi$) of the entire formula $\varphi$. The second one follows from the presence of a matching (in $G_\varphi$) of a subformula of size atmost $r$ (by virtue of $\varphi$'s bipartite expansion property).

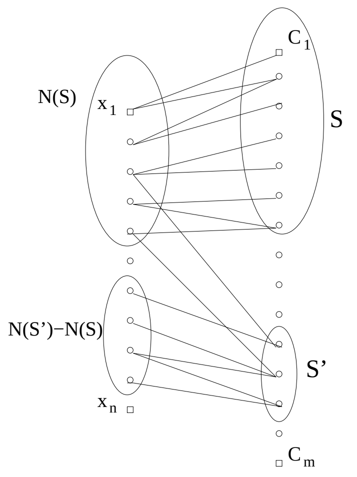


Figure 22.3: $S'$ is an arbitrary subset of $\{C_1, C_2, \ldots, C_m\} - S$ in the proof of Tarsi's theorem. $|S'| < |N(S') - N(S)|$, because otherwise we could add $S'$ to the already maximal $S$, resulting in a contradiction.

**Corollary 102** *If $\varphi$ is unsatisfiable, then there exists a subset $S$ of the clauses of $\varphi$ for which $|S| > |N(S)|$.*

**Corollary 103** *For a positive $r$ and $c$, if $\varphi$ is $(r, c)$-expanding, then every subformula of $\varphi$ having atmost $r$ clauses is satisfiable.*

We now present Tarsi's theorem and its corollary, which will be used in the proof of Width-LB theorem in the next section.

**Theorem 104** (Tarsi's theorem) *For a minimally unsatisfiable CNF formula $\varphi$ on $m$ clauses and $n$ variables,*

$$m > n.$$

*(Note that minimal unsatisfiability refers to minimality w.r.t inclusion of clauses. That is, if we we take away any clause from $\varphi$, it becomes satisfiable).*

Proof: Since $\varphi$ is a unsatisfiable formula (on clauses $C_1, C_2, \ldots, C_m$ and variables $x_1, \ldots, x_n$), by Corollary 102 of Hall's theorem, we have a subset of clauses $S$ for which $|S| > |N(S)|$. Let's make sure that $S$ is the maximal such set. If $S$ is the entire formula, we are done with the proof as $|S| = m$ and $|N(S)| = n$. So, let's assume otherwise (i.e., $|S| < m$) and let $S'$ be an arbitrary subset of $\{C_1, C_2, \ldots, C_m\} - S$. By maximality of $S$, we have $|S'| < |N(S') - N(S)|$ (see Figure 22.3). By Hall's theorem, this result basically translates to a matching in $G_\varphi$ (and hence satisfiability), of all the clauses in $\{C_1, C_2, \ldots, C_m\} - S$ using only the variables in $\{x_1, \ldots, x_n\} - N(S)$. Since $\varphi$ is minimally unsatisfiable, $S$ can also be satisfied (ofcourse using only the variables in $N(S)$), leading to the satisfiability of the entire formula $\varphi$. The only way to resolve this contradiction requires $S$ to be the entire formula, the case that we have already proved. □

**Corollary 105** *If a CNF formula $\varphi$ on $m$ clauses and $n$ variables, minimally implies a clause $D$ of size $k$, then*

$$m > n - k \quad \text{or equivalently,} \quad k > n - m.$$

*(Note that "minimally implies" again refers to minimality w.r.t inclusion of clauses. That is, $D$ can be derived from $\varphi$ but not from any subformula of $\varphi$).*
Proof: Recall that $x^1$ and $x^0$ are aliases for $x$ and $\overline{x}$ respectively. If $D$ is $(x_1^{a_1} \vee \ldots \vee x_k^{a_k})$, the unique assignment that contradicts $D$ is then $(x_1{=}1{-}a_1, \ldots, x_k{=}1{-}a_k)$. Restricting $\varphi$ to this assignment should yield a unsatisfiable formula $\varphi_{(x_1=1-a_1,\ldots,x_k=1-a_k)}$ on $(n-k)$ variables and atmost $m$ clauses. It is also minimally unsatisfiable because $\varphi$ minimally implied $D$ in the first place. We obtain the result by direct application of Tarsi's theorem. □

### 22.2.3 Putting them together

Equipped with the tools of last section, we are ready for the proof of Width-LB theorem (Theorem 99). Intuitively, $\varphi$ can't be refuted using only $r$ of its clauses and we should be able to find a clause that can be derived from a "fairly big" but not "too big" subformula of $\varphi$. The proof follows from arguing about the size of such a clause in the minimum-width refutation.

Proof: (of Width-LB Theorem) Let $\pi$ be the minimum-width refutation of $\varphi$, a $(r, c)$-expanding formula on $m$ clauses. Define a measure $\mu$ (a number between 1 and $m$) for all clauses $D \in \pi$ as:

$$\mu(D) = |\text{ smallest subformula of } \varphi \text{ that minimally implies } D \;|.$$

This measure has the following easily verifiable properties.

1. For any clause $C$ in $\pi$ that is also in $\varphi$, $\mu(C) = 1$.
2. For the final 0 clause in $\pi$, $\mu(0) \geq r$. (from Corollary 103 of Hall's theorem).
3. For a clause $D$ derived from $D'$ and $D''$ in $\pi$, $\mu(D) \leq \mu(D') + \mu(D'')$. In other words, $\mu$ is a sub-additive measure.

The above properties suggest that there exists a clause in $\pi$ (say $D^*$), whose measure is such that $r/2 \leq \mu(D^*) \leq r$. To see this, start from the final 0 clause and pick the higher measure clause of the two clauses from which it is derived. The measure of the picked clause is atleast $r/2$ and if it is also atmost $r$ stop; Otherwise continue this process recursively. Since $\mu(C) = 1$ for a clause $C$ in $\varphi$, we will ultimately succeed due to sub-additivity of the measure.

Now, let $S$ be the smallest subformula of $\varphi$ that minimally implies $D^*$. Then using the fact that $r/2 \leq |S| \leq r$ and the corollary of Tarsi's theorem, we get: (note how we use the fact that $S$ is "fairly big" but not "too big")

$$\begin{aligned}
w(\varphi) &\geq \text{Size of } D^* \\
&\geq |N(S)| - |S| && \text{(by Corollary 105 of Tarsi's theorem)} \\
&\geq (1+c)\,|S| - |S| && \text{(by expansion property of } \varphi\text{, since } |S| \leq r) \\
&= c\,|S| \\
&\geq rc/2.
\end{aligned}$$

□

## Exercises

1. Prove Theorem 100 on the expansion property of random $k$-CNFs.
2. Show that Relation (22.1) of Width-Length theorem (proved last time) and Lemma 97, can be proven by induction on $n$ alone. (The current proofs use induction on two variables).
3. Provide a family of examples, for which the upper bound on $w(\varphi)$ in Lemma 98 becomes tight. (Note that a tighter bound is $min(d+1, max(d, k))$ because there is a composition slightly different from the one in Figure 22.1, that yields a bound of $d+1$ on $w(\varphi)$).

## 22.3 References

All the results in this lecture are due to Ben-Sasson and Wigderson [BSW01].

# Lecture 23

# Pseudorandomness and Derandomization

*These notes follow quite literally the notes that I wrote for the PCMI school on computational complexity in Summer 2000. There is some redundancy between these notes and the content of past lectures.*

## 23.1 Probabilistic Algorithms versus Deterministic Algorithms

A probabilistic algorithm $A(\cdot,\cdot)$ is an algorithm that takes two inputs $x$ and $r$, where $x$ is an instance of some problem that we want to solve, and $r$ is the output of a *random source*. A random source is an idealized device that outputs a sequence of bits that are uniformly and independently distributed. For example the random source could be a device that tosses coins, observes the outcome, and outputs it. A probabilistic algorithm $A$ is good if it is efficient and if, say, for every $x$,

$$\mathbf{Pr}_r[A(x,r) = \text{ right answer for } x\ ] \geq \frac{3}{4}$$

We will typically restrict to the case where $A$ solves a decision problem (e.g. it tests whether two read-once branching programs are equivalent). In this case we say that a language $L$ is in **BPP** if there is a polynomial time algorithm $A(\cdot,\cdot)$ (polynomial in the length of the first input) such that for every $x$

$$\mathbf{Pr}_r[A(x,r) = \chi_L(x)] \geq \frac{3}{4}$$

or, said another way,

$$x \in L \Rightarrow \mathbf{Pr}_r[A(x,r) = 1] \geq \frac{3}{4}$$

and

$$x \notin L \Rightarrow \mathbf{Pr}_r[A(x,r) = 1] \leq \frac{1}{4}\ .$$

The choice of the constant $3/4$ is clearly quite arbitrary. For any constant $1/2 < p < 1$, if we had defined **BPP** by requiring the probabilistic algorithm to be correct with probability st least $p$, we would have given an equivalent definition. In fact, for any polynomial $p$, it would have been equivalent to define **BPP** by asking the algorithm to be correct with probability at least $1/2+1/p(n)$, where $n$ is the size of the input, and it would have also been equivalent if we had asked the algorithm to be correct with probability at least $1-1/2^{p(n)}$. That is, for any two polynomials $p$ and $q$, if for a decision problem $L$ we have a probabilistic polynomial time $A$ that solves $L$ on every input of length $n$ with probability at least $1/2+1/p(n)$, then there is another probabilistic algorithm $A'$, still running in polynomial time, that solves $L$ on every input of length $n$ with probability at least $1-2^{-q(n)}$.

For quite a few interesting problems, the only known polynomial time algorithms are probabilistic. A well-known example is the problem of testing whether two multivariate low-degree polynomials given in an implicit representation are equivalent. Another example is the problem of extracting "square roots" modulo a prime, i.e. to find solutions, if they exist, to equations of the form $x^2 = a \pmod{p}$ where $p$ and $a$ are given, and $p$ is prime. More generally, there are probabilistic polynomial time algorithms to find roots of polynomials modulo a prime. There is no known deterministic polynomial time algorithm for any of the above problems.

It is not clear whether the existence of such probabilistic algorithms suggests that probabilistic algorithms are inherently more powerful than deterministic ones, or that we have not been able yet to find the best possible deterministic algorithms for these problems. In general, it is quite an interesting question to determine what is the relative power of probabilistic and deterministic computations. This question is the main motivations for the results described in this lecture and the next ones.

### 23.1.1 A trivial deterministic simulation

Let $A$ be a probabilistic algorithm that solves a decision problem $L$. On input $x$ of length $n$, say that $A$ uses a random string $r$ of length $m = m(n)$ and runs in time $T = T(n)$ (note that $m \leq T$).

It is easy to come up with a deterministic algorithm that solves $L$ in time $2^{m(n)}T(n)$. On input $x$, compute $A(x, r)$ for every $r$. The correct answer is the one that comes up the majority of the times, so, in order to solve our problem, we just have to keep track, during the computation of $A(x, r)$ for every $r$, of the number of strings $r$ for which $A(x, r) = 1$ and the number of strings $r$ for which $A(x, r) = 0$.

Notice that the running time of the simulation depends exponentially on the number of random bits used by $A$, but only polynomially on the running time of $A$. In particular, if $A$ uses a logarithmic number of random bits, then the simulation is polynomial. However, typically, a probabilistic algorithm uses a linear, or more, number of random bits, and so this trivial simulation is exponential. As we will see in the next section, it is not easy to obtain more efficient simulations.

### 23.1.2 Exponential gaps between randomized and deterministic procedures

For some computational problems (e.g. approximating the size of a convex body) there are probabilistic algorithms that work even if the object on which they operate is exponentially big and given as a black box; in some cases one can prove that deterministic algorithms cannot solve the same problem in the same setting, unless they use exponential time. Let us see a particularly clean (but more artificial) example of this situation.

Suppose that there is some function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ that is given as an oracle; we want to devise an algorithm that on input $x$ finds an approximation (say, to within an additive factor 1/10) to the value $\mathbf{Pr}_y[f(x,y) = 1]$. A probabilistic algorithm would pick $O(1)$ points $y_1, \ldots, y_t$ at random, evaluate $f(x, y_i)$, and then output the fraction of $i$ such that $f(x, y_i) = 1$. This will be an approximation to within 1/10 with good probability. However a deterministic subexponential algorithm, given $x$, can only look at a negligible fraction of the values $f(x, y)$. Suppose that $f$ is zero everywhere. Now consider the function $g(x, y)$ that is equal to $f$ on all the points that our algorithm queries, and is 1 elsewhere (note that, by this definition, the queries of the algorithm on input $x$ will be the same for $f$ and $g$). If the algorithm takes sub-exponential time, $g$ is almost everywhere one, yet the algorithm will give the same answer as when accessing $f$, which is everywhere zero. If our algorithm makes less than $2^{n-1}$ oracle queries, it cannot solve the problem with the required accuracy.

## 23.2 De-randomization Under Complexity Assumptions

It is still not known how to improve, in the general case, the deterministic simulation of Section 23.1.1, and the observation of Section 23.1.2 shows one of the difficulties in achieving an improvement. If we want to come up with a general way of transforming probabilistic procedures into deterministic sub-exponential procedures, the transformation cannot be described and analyzed by modeling in a "black box" way the probabilistic procedure.[1] If we want to deterministically and sub-exponentially simulate **BPP** algorithms, we have to exploit the fact that a **BPP** algorithm $A(\cdot, \cdot)$ is not an arbitrary function, but an efficiently computable one, and this is difficult because we still have a very poor understanding of the nature of efficient computations.

The results described in these notes show that it is indeed possible to deterministically simulate probabilistic algorithms in sub-exponential (or even polynomial) time, provided that certain complexity-theoretic assumptions are true. It is quite usual in complexity theory that, using reductions, one can show that the answer to some open question is implied by (or even equivalent to) the answer to some other question, however the nature of the results of these notes is somewhat unusual. Typically a reduction from a computational problem $A$ to a problem $B$ shows that if $B$ has an efficient algorithm then $A$ has also an efficient algorithm, and, by counterpositive, if $A$ is intractable then $B$ is also intractable. In general, using reductions one shows that algorithmic assumptions imply algorithmic consequences, and intractability assumptions imply intractability consequences. In these notes we will see instead that the existence of efficient derandomized algorithms is implied by the *intractability* of some other problem, so that a hardness condition implies an algorithm

[1]More precisely, it is impossible to have a sub-exponential time deterministic "universal derandomization procedure" that given $x$ and oracle access to an arbitrary function $A(\cdot, \cdot)$ outputs 1 when $\mathbf{Pr}_r[A(x,r) = 1] \geq 3/4$ and outputs 0 when $\mathbf{Pr}_r[A(x,r) = 1] \leq 1/4$. In fact, more generally, it is impossible to give sub-exponential time algorithms for all **BPP** problems by using "relativizing" techniques. It is beyond the scope of these notes to explain what this means, and why it is more general. "Relativizations" are discussed in [Pap94], where it is possible to find pointers to the relevant literature.

consequence.

In the next section we will introduce some notation about computational problems and complexity measures, and then we will state some results about conditional de-randomization.

### 23.2.1 Formal Definitions of Complexity Measures and Complexity Classes

For a decision problem $L$ and an integer $n$ we denote by $L_n$ the restriction of $L$ to inputs of length $n$. It will be convenient to think of $L_n$ as a Boolean function $L_n : \{0,1\}^n \to \{0,1\}$ (with the convention that $x \in L_n$ if and only if $L_n(x) = 1$).

For a function $f : \{0,1\}^n \to \{0,1\}$, consider the size of the smallest circuit that solves $f$; denote this number by $CC(f)$. By definition, we have that if $C$ is a circuit with $n$ inputs of size less than $CC(f)$ then there exists an $x \in \{0,1\}^n$ such that $C(x) \neq f(x)$.

Consider now, for every $n$, what is the largest $s$ such that for every circuit $C$ of size $\leq s$ we have $\mathbf{Pr}_{x\in\{0,1\}^n}[C(x) = f(x)] \leq 1/2 + 1/s$; denote this number by $H(f)$.

Recall that $DTIME(T(n))$ is the class of decision problems that can be solved by deterministic algorithms running in time at most $T(n)$ on inputs of length $n$. We have the classes $\mathbf{E} = DTIME(2^{O(n)})$ and $\mathbf{EXP} = DTIME(2^{n^{O(1)}})$.

### 23.2.2 Hardness versus Randomness

From our previous arguments, we have $\mathbf{BPP} \subseteq \mathbf{EXP}$. Since there are settings where probabilistic procedures require exponential time to be simulated, one would conjecture that $\mathbf{BPP} \not\subseteq 2^{n^{o(1)}}$; on the other hand, $\mathbf{BPP}$ seems to still represent a class of feasible computations, and it would be *very* surprising if $\mathbf{BPP} = \mathbf{EXP}$. As we will see in a moment, something is wrong with the above intuition. Either $\mathbf{BPP} = \mathbf{EXP}$, which sounds really impossible, or else it must be the case that $\mathbf{BPP}$ has sub-exponential time deterministic algorithms (that will work well only on average, but that would be quite remarkable enough).

**Theorem 106 ([IW98])** *Suppose* $\mathbf{BPP} \neq \mathbf{EXP}$*; then for every* $\mathbf{BPP}$ *language* $L$ *and every* $\epsilon > 0$ *there is a deterministic algorithm* $A$ *that works in time* $2^{n^\epsilon}$ *and, for infinitely many* $n$*, solves* $L$ *on a fraction* $1 - 1/n$ *of the inputs of length* $n$*.*

This gives a non-trivial simulation of $\mathbf{BPP}$ under an uncontroversial assumption. We can also get an optimal simulation of $\mathbf{BPP}$ under an assumption that is much stronger, but quite believable.

**Theorem 107 ([IW97])** *Suppose there is a problem* $L$ *in* $\mathbf{E}$ *and a fixed* $\delta > 0$ *such that, for all sufficiently large* $n$*,* $CC(L_n) \geq 2^{\delta n}$*; then* $\mathbf{P} = \mathbf{BPP}$*.*

We will call the statement "there is a problem $L$ in $\mathbf{E}$ and a fixed $\delta > 0$ such that, for all sufficiently large $n$, $CC(L_n) \geq 2^{\delta n}$" the "IW assumption." Note that if the IW assumption is true, then it is true in the case where

$$L = \{(M, x, 1^k) : \text{ machine } M \text{ halts within } 2^k \text{ steps on input } x\ \}$$

Notice also that $L$ cannot be solved by algorithms running in time $2^{o(n)}$, and so it would be a little bit surprising if it could be solvable by circuits of size $2^{o(n)}$, because it would mean that, for general exponential time computations, non-uniformity buys more than a polynomial speed-up. In fact it would be very surprising if circuits of size $2^{.99n}$ existed for $L$.

The two theorems that we just stated are the extremes of a continuum of results showing that by making assumptions on the hardness of problems in **E** and **EXP** it is possible to devise efficient deterministic algorithms for all **BPP** problems. The stronger the assumption, the more efficient the simulation.

Notice that the assumption in Theorem 107 is stronger than the assumption in Theorem 106 in two ways, and that, similarly, the conclusion of Theorem 107 is stronger than the conclusion in Theorem 106 in two ways. On the one hand, the assumption in Theorem 107 refers to circuit size, that is, to a non-uniform measure of complexity, whereas the assumption in Theorem 106 uses a uniform measure of complexity (running time of probabilistic algorithms). This difference accounts for the fact that the conclusion of Theorem 107 gives an algorithm that works for all inputs, while the conclusion of Theorem 106 gives an algorithm that works only for most inputs. The other difference is that Theorem 107 assumes exponential hardness, while Theorem 107 assumes only super-polynomial hardness. This is reflected in the running time of the consequent deterministic simulations (respectively, polynomial and sub-exponential).

When one makes the non-uniform assumption that there is a problem in **E** that requires circuits of size $s(n)$, then the consequence is a deterministic simulation of **BPP** in time roughly $2^{s^{-1}(n^{O(1)})}$ [ISW99, SU01, Uma02]. So if one assumes that **E** requires super-polynomial circuits, **BPP** can be simulated in time $2^{n^{o(1)}}$, if one assumes that **E** requires circuits of size $2^{\Omega(n)}$ then the simulation runs in time $n^{O(1)}$, if one assumes that **E** requires circuits of size $n^{\log n}$ then the simulation runs in time $2^{2^{O(\sqrt{\log n})}}$, and so on. The result of [IW98] does not scale up so well when one is willing to make stronger uniform assumptions. In particular, the following is an open question.

**Conjecture 1** *Suppose* $\mathbf{E} \not\subseteq \bigcap_{\delta>0} BPTIME(2^{\delta n})$*; then for every* **BPP** *language* $L$ *there is a deterministic polynomial time algorithm* $A$ *that, for infinitely many* $n$*, solves* $L$ *on a fraction* $1-1/n$ *of the inputs of length* $n$*.*

## 23.3 Pseudorandom Generators

We say that a function $G:\{0,1\}^t \to \{0,1\}^m$ is a $(s,\epsilon)$-pseudorandom generator if for every circuit $D$ of size $\leq s$ we have

$$|\mathbf{Pr}_r[D(r)=1] - \mathbf{Pr}_z[D(G(z))=1]| \leq \epsilon$$

Suppose that we have a probabilistic algorithm $A$ such that for inputs $x$ of length $n$ the computation $A(x,\cdot)$ can be performed by a circuit of size $s(n)$; suppose that for every $x$ we have $\mathbf{Pr}_r[A(x,r) = \text{ right answer }] \geq 3/4$, and suppose that we have a $(s,1/8)$ pseudorandom generator $G:\{0,1\}^{t(n)} \to \{0,1\}^{m(n)}$. Then we can define a new probabilistic algorithm $A'$ such that $A'(x,z) = A(x,G(z))$. It is easy to observe that for every $x$ we have

$$\mathbf{Pr}_z[A'(x,z) = \text{ right answer }] \geq 5/8$$

and that, using the trivial derandomization we can get a deterministic algorithm $A''$ that always works correctly and whose running time is $2^t$ times the sum of the running time of $A$ plus the running time of $G$.

If $t$ is logarithmic in $m$ and $s$, and if $G$ is computable in poly$(m, s)$ time, then the whole simulation runs in deterministic polynomial time. Notice also that if we have a $(s, \epsilon)$-pseudorandom generator $G : \{0,1\}^t \to \{0,1\}^m$, then for every $m' \leq m$ we also have, for a stronger reason, a $(s, \epsilon)$ pseudorandom generator $G' : \{0,1\}^t \to \{0,1\}^{m'}$ ($G'$ just computes $G$ and omits the last $m - m'$ bits of the output). So there will be no loss in generality if we consider only generators for the special case where, say, $s = 2m$. (This is not really necessary, but it will help reduce the number of parameters in the statements of theorems.) We have the following easy theorem.

**Theorem 108** *Suppose there is a family of generators $G_m : \{0,1\}^{O(\log m)} \to \{0,1\}^m$ that are computable in poly$(m)$ time and that are $(2m, 1/8)$-pseudorandom; then* $\mathbf{P} = \mathbf{BPP}$.

Of course this is only a sufficient condition. There could be other approaches to proving (conditionally) $\mathbf{P} = \mathbf{BPP}$, without passing through the construction of such strong generators. Unfortunately we hardly know of any other approach, and anyway the (arguably) most interesting results are proved using pseudorandom generators.[2]

## 23.4 The two main theorems

### 23.4.1 The Nisan-Wigderson Theorem

**Theorem 109 (Special case of [NW94])** *Suppose there is $L \in \mathbf{E}$ and $\delta > 0$ such that, for all sufficiently large $n$, $H(L_n) \geq 2^{\delta n}$; then there is a family of generators $G_m : \{0,1\}^{O(\log m)} \to \{0,1\}^m$ that are computable in poly$(m)$ time and that are $(2m, 1/8)$-pseudorandom (in particular,* $\mathbf{P} = \mathbf{BPP}$*).*

Notice the strength of the assumption. For almost every input length $n$, our problem has to be so hard that even circuits of size $2^{\delta n}$ have to be unable to solve the problem correctly on more than a fraction $1/2 + 2^{-\delta n}$ of the inputs. A circuit of size 1 can certainly solve the problem on a fraction at least $1/2$ of the inputs (either by always outputting 0 or by always outputting 1). Furthermore, a circuit of size $2^n$ always exist that solves the problem on every input. A circuit of size $2^{\delta n}$ can contain, for example, the right solution to our problem for every input whose first $(1 - \delta)n$ bits are 0; the circuit can give the right answer on these $2^{\delta n}$ inputs, and answer always 0 or always 1 (whichever is better) on the other inputs. This way the circuit is good on about a fraction $1/2 + 2^{-(1-\delta)n}$ of the inputs.

[2]Some exceptions are discussed below. Andreev et al. [ACR98] show that in order to deterministically simulate probabilistic algorithms it is enough to construct *hitting set generators*, a seemingly weaker primitive than a pseudorandom generator. The complicated proof of [ACR98] was simplified in subsequent work [ACRT99, BF99, GW99]. Andreev et al. [ACR99] also show how to construct hitting set generators, but only under very strong complexity assumptions. Miltersen and Vinodchandran [MV99] give a very elegant construction of hitting set generators, but it also requires a stronger hardness assumption than in [IW97]. On the other hand, [MV99] also gets a stronger conclusion, and, in particular, it is not known how to prove the main result of [MV99] (about the "derandomization" of two-rounds interactive proofs) using pseudorandom generators.

So, in particular, for every problem, there is a circuit of size $2^{n/2}$ that solves the problem on a fraction $1/2 + 2^{-n/2}$ of the inputs. It is somewhat more tricky to show that there is in fact even a circuit of size $2^{(1/3+o(1))n}$ that solves the problem on a fraction $1/2 + 2^{-(1/3+o(1))n}$ of the inputs, and this is about best possible for general problems [ACR97].

### 23.4.2 Worst-case to Average-case Reduction

**Theorem 110 ([BFNW93, Imp95a, IW97])** *Suppose there is $L \in \mathbf{E}$ and $\delta > 0$ such that, for all sufficiently large $n$, $CC(L_n) \geq 2^{\delta n}$; Then there is $L' \in \mathbf{E}$ and $\delta' > 0$ such that, for all sufficiently large $n$, $H(L'_n) \geq 2^{\delta' n}$.*

This is quite encouraging: the (believable) IW assumption implies the (a priori less believable) NW assumption. Notice how Theorem 107 follows from Theorems 109 and 110.

The results on polynomial reconstruction from Lecture 10 imply Theorem 110, although this is not the way it was originally proved. Our proof followed [STV01].

# Lecture 24

# The Nisan-Wigderson Generator

*These notes follow quite literally the notes that I wrote for the PCMI school on computational complexity in Summer 2000. There is some redundancy between these notes and the content of past lectures.*

In this lecture we prove Theorem 109.

## 24.1 The Nisan-Wigderson Construction

The Nisan-Wigderson generator is based on the existence of a decision problem $L$ in $\mathbf{E}$ such that for almost every input length $l$ we have $H(L_l) \geq 2^{\delta l}$, yet there is a uniform algorithm that solves $L_l$ in $2^{O(l)}$ time. Our goal is to use these assumptions on $L_l$ to build a generator whose input seed is of length $O(l)$, whose output is of length $2^{\Theta(l)}$ and indistinguishable from uniform by adversaries of size $2^{\Theta(l)}$, and the generator should be computable in time $2^{O(l)}$.

As we will see in a moment, it is not too hard to construct a generator that maps $l$ bits into $l+1$ bits, and whose running time and pseudorandomness are as required. We will then present the Nisan-Wigderson construction, and present its analysis.

### 24.1.1 Impredictability versus Pseudorandomness

Let $f : \{0,1\}^l \to \{0,1\}$ be a function such that $H(f) \geq s$, and consider the pseudorandom generator $G : \{0,1\}^l \to \{0,1\}^{l+1}$ defined as $G(x) = x \cdot f(x)$, where '$\cdot$' is used to denote concatenation. We want to argue that $G$ is a $(s-3, 1/s)$-pseudorandom generator.

The argument works by contradiction, and consists in the proof of the following result.

**Lemma 111** *Let $f : \{0,1\}^l \to \{0,1\}$. Suppose that there is a circuit $D$ of size $s$ such that*

$$|\mathbf{Pr}_x[D(x \cdot f(x)) = 1] - \mathbf{Pr}_{x,b}[D(x \cdot b) = 1]| > \epsilon$$

*then there is a circuit $A$ of size $s+3$ such that*

$$\mathbf{Pr}_x[A(x) = f(x)] > \frac{1}{2} + \epsilon$$

Proof: First of all, we observe that there is a circuit $D'$ of size at most $s+1$ such that

$$\mathbf{Pr}_z[D'(x \cdot f(x)) = 1] - \mathbf{Pr}_{x,b}[D'(x \cdot b) = 1] > \epsilon \tag{24.1}$$

This is because Expression (24.1) is satisfied either by taking $D = D'$ or by taking $D = \neg D'$. A way to interpret Expression (24.1) is to observe that when the first $l$ bits of the input of $D'()$ are a random string $x$, $D'$ is more likely to accept if the last bit is $f(x)$ than if the last bit is random (and, for a stronger reason, if the last bit is $1 - f(x)$). This observation suggests the following strategy in order to use $D'$ to predict $f$: given an input $x$, for which we want to compute $f(x)$, we guess a value $b$, and we compute $D'(x,b)$. If $D'(x,b) = 1$, we take it as evidence that $b$ was a good guess for $f(x)$, and we output $b$. If $D'(x,b) = 0$, we take it as evidence that $b$ was the wrong guess, and we output $1-b$. Let $A_b$ be the procedure that we just described. We claim that

$$\mathbf{Pr}_{x,b}[A_b(x) = f(x)] > \frac{1}{2} + \epsilon \tag{24.2}$$

The claim is proved by the following derivation

$$\begin{aligned}
& \mathbf{Pr}_{x,b}[A_b(x) = f(x)] \\
= \;& \mathbf{Pr}_{x,b}[A_b(x) = f(x) | b = f(x)]\mathbf{Pr}_{x,b}[b = f(x)] \\
& + \mathbf{Pr}_{x,b}[A_b(x) = f(x) | b \neq f(x)]\mathbf{Pr}_{x,b}[b \neq f(x)] \\
= \;& \frac{1}{2}\mathbf{Pr}_{x,b}[A_b(x) = f(x) | b = f(x)] + \frac{1}{2}\mathbf{Pr}_{x,b}[A_b(x) = f(x) | b \neq f(x)] \\
= \;& \frac{1}{2}\mathbf{Pr}_{x,b}[D'(x,b) = 1 | b = f(x)] + \frac{1}{2}\mathbf{Pr}_{x,b}[D'(x,b) = 0 | b \neq f(x)] \\
= \;& \frac{1}{2} + \frac{1}{2}\mathbf{Pr}_{x,b}[D'(x,b) = 1 | b = f(x)] - \frac{1}{2}\mathbf{Pr}_{x,b}[D'(x,b) = 1 | b \neq f(x)] \\
= \;& \frac{1}{2} + \mathbf{Pr}_{x,b}[D'(x,b) = 1 | b = f(x)] \\
& - \frac{1}{2}\left(\mathbf{Pr}_{x,b}[D'(x,b) = 1 | b = f(x)] + \mathbf{Pr}_{x,b}[D'(x,b) = 1 | b \neq f(x)]\right) \\
= \;& \frac{1}{2} + \mathbf{Pr}_x[D'(x, f(x)) = 1] - \mathbf{Pr}_{x,b}[D'(x,b) = 1] \\
> \;& \frac{1}{2} + \epsilon
\end{aligned}$$

From Expression (24.2) we can observe that there must be a $b_0 \in \{0,1\}$ such that

$$\mathbf{Pr}_x[A_{b_0}(x) = f(x)] > \frac{1}{2} + \epsilon$$

And $A_{b_0}$ is computed by a circuit of size at most $s+3$ because $A_{b_0}(x) = b_0 \oplus (\neg D'(x, b_0))$, which can be implemented with two more gates given a circuit for $D'$. □

### 24.1.2 Combinatorial Designs

Consider a family $(S_1, \ldots, S_m)$ of subsets of an universe $U$. We say the family is a $(l, \alpha)$-design if, for every $i$, $|S_i| = l$, and, for every $i \neq j$, $|S_i \cap S_j| \leq \alpha$.

**Theorem 112** *For every integer $l$, fraction $\gamma > 0$, there is an $(l, \log m)$ design $(S_1, \ldots, S_m)$ over the universe $[t]$, where $t = O(l/\gamma)$ and $m = 2^{\gamma l}$; such a design can be constructed in $O(2^t t m^2)$ time.*

We will use the following notation: if $z$ is a string in $\{0,1\}^t$ and $S \subset [t]$, then we denote by $z_{|S}$ the string of length $|S|$ obtained from $z$ by selecting the bits indexed by $S$. For example if $z = (0,0,1,0,1,0)$ and $S = \{1,2,3,5\}$ then $z_{|S} = (0,0,1,1)$.

### 24.1.3 The Nisan-Wigderson Generator

For a Boolean function $f : \{0,1\}^l \to \{0,1\}$, and a design $\mathcal{S} = (S_1, \ldots, S_m)$ over $[t]$, the Nisan-Wigderson generator is a function $NW_{f,\mathcal{S}} : \{0,1\}^t \to \{0,1\}^m$ defined as follows:

$$NW_{f,\mathcal{S}}(z) = f(z_{|S_1}) \cdot f(z_{|S_2}) \cdots f(z_{|S_m})$$

## 24.2 The Reduction from Distinguishing to Predicting

The following lemma implies Theorem 109.

**Lemma 113** *Let $f : \{0,1\}^l \to \{0,1\}$ be a Boolean function and $\mathcal{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$. Suppose $D : \{0,1\}^m \to \{0,1\}$ is such that*

$$|\mathbf{Pr}_r[D(r) = 1] - \mathbf{Pr}_z[D(NW_{f,\mathcal{S}}(z)) = 1]| > \epsilon \ .$$

*Then there exists a circuit $C$ of size $O(m^2)$ such that*

$$|\mathbf{Pr}_x[D(C(x)) = f(x)] - 1/2| \geq \frac{\epsilon}{m}$$

Proof: The main idea is that if $D$ distinguishes $NW_{f,\mathcal{S}}(\cdot)$ from the uniform distribution, then we can find a bit of the output of the generator where this distinction is noticeable. On such a bit, $D$ is distinguishing $f(x)$ from a random bit, and such a distinguisher can be turned into a predictor for $f$. In order to find the "right bit", we will use the *hybrid argument.* At this level of abstraction, the analysis is the same as the analysis of the Blum-Micali-Yao generator, however, as the analysis unfolds, we will see major differences.

Let us start with the hybrid argument. We define $m+1$ distributions $H_0, \ldots, H_m$; $H_i$ is defined as follows: sample a string $v = NW_{f,\mathcal{S}}(z)$ for a random $z$, and then sample a string $r \in \{0,1\}^m$ according to the uniform distribution, then concatenate the first $i$ bits of $v$ with the last $m - i$ bits of $r$. By definition, $H_m$ is distributed as $NW_{f,\mathcal{S}}(y)$ and $H_0$ is the uniform distribution over $\{0,1\}^m$.

Using the hypothesis of the Lemma, we know that there is a bit $b_0 \in \{0,1\}$ such that

$$\mathbf{Pr}_y[D'(NW_{f,\mathcal{S}}(y)) = 1] - \mathbf{Pr}_r[D'(r)] > \epsilon$$

where $D'(x) = b_0 \oplus D(x)$.

We then observe that

$$\begin{aligned}
\epsilon &\leq \mathbf{Pr}_z[D'(NW_{f,\mathcal{S}}(z)) = 1] - \mathbf{Pr}_r[D'(r)] \\
&= \mathbf{Pr}[D'(H_m) = 1] - \mathbf{Pr}[D'(H_0) = 1] \\
&= \sum_{i=1}^{m} (\mathbf{Pr}[D'(H_i) = 1] - \mathbf{Pr}[D'(H_{i-1}) = 1])
\end{aligned}$$

In particular, there exists an index $i$ such that

$$\mathbf{Pr}[D'(H_i) = 1] - \mathbf{Pr}[D'(H_{i-1}) = 1] \geq \epsilon/m \tag{24.3}$$

Now, recall that

$$H_{i-1} = f(z_{|S_1}) \cdots f(z_{|S_{i-1}}) r_i r_{i+1} \cdot r_m$$

and

$$H_i = f(z_{|S_1}) \cdots f(y_{|S_{i-1}}) f(y_{|S_i}) r_{i+1} \cdot r_m \ .$$

We can assume without loss of generality (up to a renaming of the indices) that $S_i = \{1, \ldots, l\}$. Then we can see $z \in \{0,1\}^t$ as a pair $(x, y)$ where $x = z_{|S_i} \in \{0,1\}^l$ and $y = z_{|[t]-S_i} \in \{0,1\}^{t-l}$. For every $j < i$ and $z = (x, y)$, let us define $f_j(x, y) = f(z_{|S_j})$: note that $f_j(x, y)$ depends on $|S_i \cap S_j| \leq \log m$ bits of $x$ and on $l - |S_i \cap S_j| \geq l - \log m$ bits of $y$. With this notation we have

$$\begin{aligned}
&\mathbf{Pr}_{x,y,r_{i+1},\ldots,r_m}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), f(x), \ldots, r_m) = 1] \\
&-\mathbf{Pr}_{x,y,r_{i+1},\ldots,r_m} D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1] > \epsilon/m
\end{aligned}$$

That is, when $D'$ is given a string that contains $f_j(x, y)$ for $j < i$ in the first $i - 1$ entries, and random bits in the last $m - i$ entries, then $D'$ is more likely to accept the string if it contains $f(x)$ in the $i$-th entry than if it contains a random bit in the $i$-th entry. This is good enough to (almost) get a predictor for $f$. Consider the following algorithm:

```
Algorithm $A$
Input: $x \in \{0,1\}^l$
Pick random $r_i, \ldots, r_m \in \{0,1\}$
Pick random $y \in \{0,1\}^{t-l}$
Compute $f_1(x,y), \ldots, f_{i-1}(x,y)$
If $D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1$ output $r_i$
Else output $1 - r_i$
```

Let us forget for a moment about the fact that the step of computing $f_1(x, y), \ldots, f_{i-1}(x, y)$ looks very hard, and let us check that $A$ is good predictor. Let us denote by $A(x, y, r_1, \ldots, r_m)$

the output of $A$ on input $x$ and random choices $y, r_1, \ldots, r_m$.

$$\begin{aligned}
& \mathbf{Pr}_{x,y,r}[A(x,y,r) = f(x)] \\
= \; & \mathbf{Pr}_{x,y,r}[A(x,y,r) = f(x) | r_i = f(x)] \mathbf{Pr}_{x,r_i}[r_i = f(x)] \\
& + \mathbf{Pr}_{x,y,r}[A(x,y,r) = f(x) | r_i \neq f(x)] \mathbf{Pr}_{x,r_i}[r_i \neq f(x)] \\
= \; & \frac{1}{2} \mathbf{Pr}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) = r_i] \\
& + \frac{1}{2} \mathbf{Pr}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 0 | f(x) \neq r_i] \\
= \; & \frac{1}{2} + \frac{1}{2} \big( \mathbf{Pr}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) = b] \\
& - \mathbf{Pr}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) \neq b] \big) \\
= \; & \frac{1}{2} + \mathbf{Pr}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) = b] \\
& - \frac{1}{2} \big( \mathbf{Pr}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) = b] \\
& + \mathbf{Pr}_{x,y,r}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1 | f(x) \neq b] \big) \\
= \; & \frac{1}{2} + \mathbf{Pr}[D'(H_i) = 1] - \mathbf{Pr}[D'(H_{i-1}) = 1] \\
\geq \; & \frac{1}{2} + \frac{\epsilon}{m}
\end{aligned}$$

So $A$ is good, and it is worthwhile to see whether we can get an efficient implementation. We said we have

$$\mathbf{Pr}_{x,y,r_i,\ldots,r_m}[A(x,y,r) = f(x)] \geq \frac{1}{2} + \frac{\epsilon}{m}$$

so there surely exist fixed values $c_i, \ldots, c_m$ to give to $r_i, \ldots, r_m$, and a fixed value $w$ to give to $y$ such that

$$\mathbf{Pr}_{x,r}[A(x, w, c_i, c_{i+1}, \ldots, c_m) = f(x)] \geq \frac{1}{2} + \frac{\epsilon}{m}$$

At this point we are essentially done. Since $w$ is fixed, now, in order to implement $A$, we only have to compute $f_j(x,w)$ given $x$. However, for each $j$, $f_j(x,w)$ is a function that depends only on $\leq \log m$ bits of $x$, and so is computable by a circuits of size $O(m)$. Even composing $i-1 < m$ such circuit, we still have that the sequence $f_1(x,w), \ldots, f_{i-1}(x,w), c_i, c_{i+1}, \ldots, c_m$ can be computed, given $x$, by a circuit $C$ of size $O(m^2)$. Finally, we notice that at this point $A(x,w,c)$ is doing the following: output the xor between $c_i$ and the complement of $D'(C(x))$. Since $c_i$ is fixed, either $A(x,w,c)$ always equals $D(C(x))$, or one is the complement of the other. In either case the Lemma follows. □

At this point it should not be too hard to derive Theorem 109.

# Lecture 25

# Extractors and Pseudorandom Generators

## 25.1 Use of Weak Random Sources

Suppose that we have a probabilistic algorithm $A(\cdot,\cdot)$ that on inputs of length $n$ runs in time $T(n)$ and uses $m(n)$ random bits. Instead of a perfect source of randomness, we assume that we have a source that produces an output containing some "impredictability," but that can still be very far from uniform. A very general way of modeling this source is to assume that on input $1^N$ it outputs a string in $\{0,1\}^N$, and that the output string has "some randomness" (a notion that we will formalize and quantify in a moment). Typically, a good way to quantify the amount of randomness, or impredictability, in a distribution, is to compute its (Shannon) entropy. For a random variable $X$ whose range is $\{0,1\}^N$, its entropy is defined as $\mathcal{H}(X) = \sum_{a\in\{0,1\}^N} \mathbf{Pr}[X=a]\log(1/\mathbf{Pr}[X=a])$,

Shannon entropy is a very good measure in cases where one is allowed to take multiple samples from the distribution, but in our setting this is not the best measure. Consider for example a distribution $X$ such that $X=(0,0,\cdots,0)$ with probability $1-1/\sqrt{N}$, and it is uniform with probability $1/\sqrt{N}$. Then its Shannon entropy is about $\sqrt{N}$, which is quite high, yet it is almost always a useless string of zeroes. It is a good intuition to think that the amount of randomness contained in the outcome $a$ of a random variable $X$ is $\log 1/\mathbf{Pr}[X=a]$. If $X$ has Shannon entropy $k$, then *on average*, when we sample from $X$ we get a value of "randomness" $k$, however it can be the case that with very high probability we get almost zero randomness, and with low probability we get high randomness. We would rather have a measure of randomness that guarantees to have almost always, or, even better, always, high randomness. This motivates the definition of *min-entropy*: a random variable $X$ has min-entropy at least $k$ if for every $a$ in the range of $X$ we have $\mathbf{Pr}[X=a] \leq 1/2^k$. That is, the min-entropy of $X$ is $\min_a\{\log 1/\mathbf{Pr}[X=a]\}$.

**Definition 31** *A random variable with range $\{0,1\}^N$ having min-entropy at least $k$ will be called a $(N,k)$-source.*

Given one access to a $(N,k)$ source, we would like to be able to simulate any probabilistic algorithm that uses $m$ random bits, where $m$ is close to $k$. If the simulation is "black box" and takes time $T$, one can argue that $m \leq k + O(\log T)$. We will not define formally what a black-box simulation is, but we will develop simulations that are black box, so it will come as no surprise that our simulations will work only for $m$ smaller than $k$, in fact only for $m$ smaller than $k^{1/3}$. (This is partly due to oversimplifications in the analysis; one could get $k^{.99}$ with almost the same proof.)

## 25.2 Extractors

An extractor is a function that transforms a $(N, k)$ source into an almost uniform distribution. The transformation is done by using a (typically very small) number of additional random bits.

Formally, we have the following definition.

**Definition 32** *For two random variables $X$ and $Y$ with range $\{0,1\}^m$, their variational distance is defined as $||X - Y|| = \max_{S \subseteq \{0,1\}^m}\{|\mathbf{Pr}[X \in S] - \mathbf{Pr}[Y \in S]|\}$. We say that two random variables are $\epsilon$-close if their variational distance is at most $\epsilon$.*

**Definition 33** *A function $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is a $(k, \epsilon)$ extractor if for any $(N, k)$ source $X$ we have that $Ext(X, U_l)$ is $\epsilon$-close to uniform, where $U_l$ is the uniform distribution over $\{0,1\}^l$.*

Equivalently, if $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is a $(k, \epsilon)$ extractor, then for every distribution $X$ ranging over $\{0,1\}^N$ of min-entropy $k$, and for every $S \subseteq \{0,1\}^m$, we have

$$|\mathbf{Pr}_{a \in X, z \in \{0,1\}^t}[Ext(a, z) \in S] - \mathbf{Pr}_{r \in \{0,1\}^m}[r \in S]| \leq \epsilon$$

## 25.3 Applications

See [Nis96] for an extensive survey. Here we present only one application. Another notable application is the construction of expanders.

Suppose that $A(\cdot, \cdot)$ is a probabilistic algorithm that on an input of length $n$ uses $m(n)$ random bits, and suppose that for every $x$ we have $\mathbf{Pr}_r[A(x, r) = \text{ right answer }] \geq 3/4$. Suppose $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ is a $(k, 1/4)$-extractor.

Consider the following algorithm $A'$: on input $x \in \{0,1\}^n$ and weakly random $a \in \{0,1\}^N$, $A'$ computes $A(x, Ext(a, z))$ for every $z \in \{0,1\}^t$, and then it outputs the answer that appears the majority of such $2^t$ times. We want to argue that $A'$ is correct with high probability if $a$ is sampled from a weak random source of entropy slightly higher than $k$. Let us fix the input $x$. Consider the set $B$ of strings $a \in \{0,1\}^N$ for which the algorithm $A'$ makes a mistake:

$$B = \{a : \mathbf{Pr}_{z \in \{0,1\}^t}[A(x, Ext(a, z)) = \text{ right answer }] < 1/2\}$$

Consider the random variable $X$ that is uniformly distributed over $B$ (clearly, $X$ has min-entropy $\log B$). Then we have

$$\mathbf{Pr}_{a \in X, z \in \{0,1\}^t}[A(x, Ext(a, z)) = \text{ right answer }] < 1/2$$

and so

$$|\mathbf{Pr}_{a\in X,z\in\{0,1\}^t}[A(x,Ext(a,z)) = \text{ right answer }] - \mathbf{Pr}_r[A(x,r) = \text{ right answer }]| > 1/4$$

and then it follows form the property of $Ext$ that $X$ must have min-entropy less than $k$, that is $|B| \leq 2^k$.

Let now $X$ be a $(N, k+2)$-source, and let us execute algorithm $A'$ using $X$. Then

$$\mathbf{Pr}_{a\in X,z\in\{0,1\}^t}[A(x,Ext(a,z)) = \text{ right answer }] = 1 - \mathbf{Pr}_{a\in X}[a \in B] \geq 3/4$$

More generally

**Theorem 114** *Suppose $A$ is a probabilistic algorithm running in time $T_A(n)$ and using $m(n)$ random bits on inputs of length $n$. Suppose we have for every $m(n)$ a construction of a $(k(n), 1/4)$-extractor $Ext_n : \{0,1\}^N \times \{0,1\}^{t(n)} \to \{0,1\}^{m(n)}$ running in $T_E(n)$ time. Then $A$ can be simulated in time $2^t(T_A + T_E)$ using one sample from a $(N, k+2)$ source.*

## 25.4 An Extractor from Nisan-Wigderson

This is a simplified presentation of results in [Tre01] (see also [RRV99, ISW00]).

Let $\mathcal{C} : \{0,1\}^N \to \{0,1\}^{\bar{N}}$ be a polynomial time computable error-correcting code such that any ball of radius at most $1/2 - \delta$ contains at most $1/\delta^2$ codewords. Such a code exists with $\bar{n} = \text{poly}(n, 1/\delta)$.

For a string $x \in \{0,1\}^{\bar{N}}$, let $< x >: \{0,1\}^{\log \bar{N}} \to \{0,1\}$ be the function whose truth table is $x$. Let $l = \log \bar{N}$, and let $\mathcal{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$. Then consider the procedure $ExtNW : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ defined as

$$ExtNW_{\mathcal{C},\mathcal{S}}(x,z) = NW_{<\mathcal{C}(x)>,\mathcal{S}}(z) \ .$$

That is, $ExtNW$ first encodes its first input using an error-correcting code, then views it as a function, and finally applies the Nisan-Wigderson construction to such a function, using the second input as a seed.

**Lemma 115** *For sufficiently large $m$ and for $\epsilon > 2^{-m^2}$, $ExtNW_{\mathcal{C},\mathcal{S}}$ is a $(m^3, 2\epsilon)$-extractor.*

Proof: Fix a random variable $X$ of min-entropy $m^3$ and a function $D : \{0,1\}^m \to \{0,1\}$; we will argue that

$$|\mathbf{Pr}[D(r) = 1] - \mathbf{Pr}_{a\in X,z\in\{0,1\}^t}[D(ExtNW(a,z)) = 1]| \leq 2\epsilon$$

Let us call a value $a$ bad if it happens that

$$|\mathbf{Pr}[D(r) = 1] - \mathbf{Pr}_{z\in\{0,1\}^t}[D(ExtNW(a,z)) = 1]| > \epsilon$$

and let us call $B$ the set of bad $a$. When $a$ is bad, it follows that there is a circuit $C$ of size $O(m^2)$ such that either $D(C())$ or its complement agrees with $a$ on a fraction $1/2 + \epsilon/m$ of its entries. Therefore, $a$ is totally specified by $D$, $C$, and $2\log(m/\epsilon)$ additional bits (once we have $D$ and $C$, we know that the encoding of $a$ sits in a given sphere of radius $1/2 - \epsilon/m$, together with at most other $(m/\epsilon)^2$ codewords). Therefore, for a fixed $D$, the size of $B$ is upper bounded by the number of circuits of size $O(m^2)$, that is $2^{O(m^2 \log m)}$, times $(m/\epsilon)^2$, times 2. The total is $2^{O(m^2 \log m)}$. The probability that an element $a$ taken from $X$ belongs to $B$ is therefore at most $2^{-m^3} \cdot 2^{O(m^2 \log m)} < \epsilon$ for sufficiently large $m$. We then have

$$\begin{aligned}
& |\mathbf{Pr}[D(r) = 1] - \mathbf{Pr}_{a \in X, z \in \{0,1\}^t}[D(ExtNW(a,z)) = 1]| \\
\leq\ & \sum_a \mathbf{Pr}[X = a] \left|\mathbf{Pr}[D(r) = 1] - \mathbf{Pr}_{z \in \{0,1\}^t}[D(ExtNW(a,z)) = 1]\right| \\
\leq\ & \mathbf{Pr}[X \in B] + \epsilon \leq 2\epsilon
\end{aligned}$$

□

**Theorem 116** *Fix a constant $\epsilon$; for every $N$ and $k = N^{\Omega(1)}$ there is a polynomial-time computable $(k, \epsilon)$-extractor $Ext : \{0,1\}^N \times \{0,1\}^t \to \{0,1\}^m$ where $m = k^{1/3}$ and $t = O(\log N)$.*

# Bibliography

[ACR97] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. Optimal bounds for the approximation of boolean functions and some applications. *Theoretical Computer Science*, 180:243–268, 1997.

[ACR98] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. A new general derandomization method. *Journal of the ACM*, 45(1):179–213, 1998.

[ACR99] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. Worst-case hardness suffices for derandomization: A new method for hardness vs randomness trade-offs. *Theoretical Computer Science*, 221:3–18, 1999.

[ACRT99] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and BPP simulations. *SIAM Journal on Computing*, 28(6):2103–2116, 1999. Preliminary version in *Proc of FOCS'97*.

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, pages 75–83, 1978.

[ALM$^+$98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in *Proc. of FOCS'92*.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in *Proc. of FOCS'92*.

[ATSWZ97] R. Armoni, A. Ta-Shma, A. Wigderson, and S. Zhou. $SL \subseteq L^{\frac{4}{3}}$. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 230–239, 1997.

[Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 421–429, 1985. See also [BM88].

[BCH$^+$96] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan. Linearity testing over characteristic two. *IEEE Transactions on Information Theory*, 42(6):1781–1795, 1996.

[BCW80] Manuel Blum, Ashok K. Chandra, and Mark N. Wegman. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10(2):80–82, 1980.

[BDCGL92] Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193–219, 1992.

[BF90] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of STACS'90*, pages 37–48, 1990.

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In *STACS'99*, pages 100–109, 1999.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BGS98] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCP's and non-approximability – towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998. Preliminary version in *Proc. of FOCS'95.*

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993. Preliminary version in *Proc. of STOC'90.*

[BM88] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988.

[BSW01] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow: Resolution made simple. *Journal of the ACM*, 48(2), 2001.

[Coo71] S.A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[Coo73] Stephen A Cook. A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7(4):343–353, 1973.

[FF93] Joan Feigenbaum and Lance Fortnow. On the random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22:994–1005, 1993.

[FGL$^+$96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. Preliminary version in *Proc. of FOCS91.*

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6:675–695, 1977.

[GLR$^+$91] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 32–42, 1991.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Preliminary version in *Proc of STOC'85*.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity. *Journal of the ACM*, 38(3), 1991.

[GNW95] O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR lemma. Technical Report TR95-50, Electronic Colloquium on Computational Complexity, 1995.

[Gol97] Oded Goldreich. Notes on Levin's theory of average-case complexity. Technical Report TR97-058, Electronic Colloquium on Computational Complexity, 1997.

[GS86] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the 18th ACM Symposium on Theory of Computing*, pages 59–68, 1986.

[GW99] O. Goldreich and A. Wigderson. Improved derandomization of BPP using a hitting set generator. In *RANDOM'99*, pages 131–137, 1999.

[Hak85] A. Haken. The intractibility of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th ACM Symposium on Theory of Computing*, pages 6–20, 1986.

[Hås99] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HS65] J. Hartmanis and R.E. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965.

[IL90] Russell Impagliazzo and Leonid Levin. No better ways to generate hard NP instances than picking uniformly at random. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 812–821, 1990.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[Imp95a] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 538–545, 1995.

[Imp95b] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the 10th IEEE Conference on Structure in Complexity Theory*, pages 134–147, 1995.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 1999.

[ISW00] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudo-random generators with optimal seed length. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 1–10, 2000.

[IW97] R. Impagliazzo and A. Wigderson. $P = BPP$ unless $E$ has sub-exponential circuits. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[IW98] R. Impagliazzo and A. Wigderson. Randomness versus time: De-randomization under a uniform assumption. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 734–743, 1998.

[JVV86] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KL80] R.M. Karp and R.J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 302–309, 1980.

[KZ97] B. Karloff and U. Zwick. A $(7/8-\epsilon)$-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 406–415, 1997.

[Lau83] C. Lautemann. BPP and the Polynomial Hierarchy. *Information Processing Letters*, 17:215–217, 1983.

[Lev73] L. A. Levin. Universal search problems. *Problemi Peredachi Informatsii*, 9:265–266, 1973.

[Lev86] Leonid Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[Lev87] Leonid Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. Preliminary version in *Proc of FOCS'90.*

[Lip90] Richard J. Lipton. Efficient checking of computations. In *Proceedings of STACS'90*, pages 207–215, 1990.

[MV99] P.B. Miltersen and N.V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 71–80, 1999.

[Nis94] N. Nisan. RL $\subseteq$ SC. *Computational Complexity*, 4(1), 1994.

[Nis96] N. Nisan. Extracting randomness: How and why. In *Proceedings of the 11th IEEE Conference on Computational Complexity*, pages 44–58, 1996.

[NSW92] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 24–29, 1992.

[NW94] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994. Preliminary version in *Proc. of FOCS'88.*

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Raz87] A.A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition. *Matematicheskie Zametki*, 41:598–607, 1987.

[Raz98] R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998. Preliminary version in *Proc. of STOC'95.*

[RRV99] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 149–158, 1999.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.

[Sha92] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992. Preliminary version in *Proc of FOCS'90.*

[Sip83] M. Sipser. A complexity theoretic apprach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 330–335, 1983.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 77–82, 1987.

[Sto76] L.J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.

[Sto83] L.J. Stockmeyer. The complexity of approximate counting. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 118–126, 1983.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[Sud97] M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997. Preliminary version in *Proc. of FOCS'96*.

[SZ95] M. Saks and S. Zhou. RSPACE($S$) $\subseteq$ DSPACE($S^{3/2}$). In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 344–353, 1995.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tre01] L. Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

[Tse70] G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1970.

[Uma02] C. Umans. Pseudo-random generators for all hardnesses. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 627–634, 2002.

[Val79] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[VV86] L.G. Valiant and V.V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[WB86] L.R. Welch and E.R. Berlekamp. Error-correction for algebraic block codes. US Patent No. 4,633,470, 1986.

[Wra76] C. Wrathall. Complete sets for the polynomial hierarchy. *Theoretical Computer Science*, 3:23–34, 1976.

[Yao82] A.C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[Yao85] Andrew C Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, pages 216 – 226. Springer-Verlag, 1979.