Computational complexity lectures

Peter Gács
Boston University
1 Introduction

1.1 The class structure

See the course homepage.

1.2 Description complexity

Showing that it is uncomputable. See the lecture linked to the course homepage.


2 Turing machines

Definition taken from the Lovász notes.

(a) $k$ doubly infinite tapes, tape symbol alphabet $\Sigma$ includes blank $\ast$. Let $\Sigma_0 = \Sigma \setminus \{\ast\}$.

(b) Read-write heads.

(c) Control unit with state space $\Gamma$, with distinguished states $\text{START}$, $\text{STOP}$.

Configuration. Transition functions

$$\alpha : \Gamma \times \Sigma^k \rightarrow \Gamma,$$
$$\beta : \Gamma \times \Sigma^k \rightarrow \Sigma^k,$$
$$\gamma : \Gamma \times \Sigma^k \rightarrow \{-1, 0, 1\}^k$$

Computing a function $f : \Sigma_0^a \rightarrow \Sigma_0^b$: input and output conventions.
Examples of Turing machines computing some simple functions.
3 Variants on the definition

3.1 Simulation

One-sided tapes, only left-right moves (no staying in place), etc.

The notion of simulation: only the input-output behavior is reproduced.

In practice: representing one data structure in another, and “programming” the update.

3.2 Simulating 2 tapes by one tape

Simulated head 1

Simulated head 2
3.3 Simulating many tapes by two tapes

3.4 2-dimensional tape

(address, content) pairs.
4 The random access machine

(a) Memory: a (potentially) infinite sequence \( x[0], x[1], x[2], \ldots \) of memory registers each containing an integer.

(b) Program store: a (potentially) infinite sequence of registers containing instructions.

\[
\begin{align*}
x[i] &:= 0; \quad x[i] := x[i] + 1; \quad x[i] := x[i] - 1; \\
x[i] &:= x[i] + x[j]; \quad x[i] := x[i] - x[j]; \\
x[i] &:= x[x[j]]; \quad x[x[i]] := x[j]; \\
\text{if } x[i] \leq 0 \text{ then goto } p.
\end{align*}
\]

Input-output conventions.

How to define running time?

Simulations between the RAM and Turing machines. There is at most a \( t \mapsto t^2 \) slowdown.
5 Universal Turing machine

Simulating $k$ tapes with $k + 2$ tapes.
6 Nondeterministic computations

6.1 Witnesses

Simple examples:

– Factoring a number.

– Connectivity of a graph.

– No short path in a graph between two given points.

Definition of a nondeterministic Turing machine.

A language recognized by a nondeterministic Turing machine.
6.2 Rewrite rules and grammars

(I did not talk about them in class, but grammars also illustrate nondeterminism.)

Let an alphabet \( \Sigma \) be fixed.

Rewrite rule (production) \( P : u \rightarrow v \) for \( u, v \in \Sigma^* \).

A rewrite process is a finite set \( \Pi \) of productions. The meaning of \( u \Rightarrow_\Pi v \) and \( u \Rightarrow^*_\Pi v \). The word problem of a rewrite process: to decide, given \( \Pi, u, v \), whether \( u \Rightarrow^*_\Pi v \).

Grammar: \( \Gamma = (\Sigma_0 \subset \Sigma, S \in \Sigma \setminus \Sigma_0) \).

The language \( L(\Gamma) = \{ w \in \Sigma^*_0 : S \Rightarrow^* w \} \).

Grammars are also more naturally related to nondeterministic computations than to deterministic ones.
6.3 Halting language

A language is called recursively enumerable if it is the halting language of a deterministic Turing machine.

**Theorem 1.** The languages recognizable by non-deterministic Turing machines are just the recursively enumerable languages.

*Proof.* Det. halting language ⇒ nondet. recognizable: this direction is easy.

Nondet. recognizable by $M$ ⇒ det. halting language: breadth-first search over all possible computations of $M$. This is also called dovetailing. \[\square\]
6.4 Witnesses and nondeterministic Turing machines

Definitions of $\text{DTIME}(f(n))$, $\text{NTIME}(f(n))$. A function $f(n)$ is well-computable if it is computable in time $O(f(n))$.

Language $L_0 \in \text{DTIME}(g(n))$ is a witness of length $f(n)$ and time $g(n)$ for language $L$ if we have $x \in L$ if and only if there is a word $y \in \Sigma_0^*$ with $|y| \leq f(|x|)$ and $x \& y \in L_0$. (Here, $\&$ is a new, separating symbol.) Sometimes, we write $x \& y$ as $(x, y)$ or $\langle x, y \rangle$ and call the witness language the witness relation instead.
Theorem 2. Assume that $f(n), g(n)$ are well-computable. Then

(a) Every language $\mathcal{L} \in \text{NTIME}(f(n))$ has a witness of length $O(f(n))$ and time $O(n)$.

(b) If language $\mathcal{L}$ has a witness of length $f(n)$ and time $g(n)$ then $\mathcal{L}$ is in $\text{NTIME}(g(n + 1 + f(n)))$.

We went through the main steps of the proof of this theorem in class,
7 Polynomial time

The invariance of $P$ with respect to machine model.

*Example 1.* PATH between points $s$ and $t$ in a graph. Breadth-first search.

The same problem, when the edges have positive integer lengths. Reducing it to PATH in the obvious way (each edge turned into a path consisting of unit-length edges) may result in an exponential algorithm (if edge lengths are large). But Dijkstra’s algorithm works in polynomial time also with large edge lengths.
7.1 Algorithms on integers

Every algorithm \((a, b) \mapsto a^b\) over positive integers is at least exponential: look at the length of the output.

Repeated squaring trick: now the number of multiplications is polynomial, but these will be performed, eventually, on very large numbers. But: this gives a polynomial algorithm for computing \((a, b, m) \mapsto a^b \mod m\).

The customary algorithm for deciding whether a number is prime, is exponential (in the length of input).

The greatest common divisor of two numbers can be computed in polynomial time, using:

Theorem 3. \(\text{gcd}(a, b) = \text{gcd}(b, a \mod b)\)

This gives rise to Euclid’s algorithm. Why polynomial-time?
7.2 Extended Euclidean algorithm

Gives us numbers \( x, y \) with

\[
gcd(a, b) = xa + yb.
\]

For this, simply maintain such a form for all numbers computed during the algorithm. If

\[
a' = x_1a + y_1b, \\
b' = x_2a + y_2b, \\
r' = a - qb' < b'
\]

then

\[
r' = (x_1 - ax_2)a + (y_1 - qy_2)b.
\]
7.3 Solving a set of linear equations: Gaussian elimination

The problem of roundoff errors.

Rational inputs, exact rational solution.

How large can the numerators and denominators grow?

Determinant, properties:

1. It is a polynomial of its entries.

2. Row operations do not change it.

3. Interpreting it as volume, hence upper bound: product of vector lengths.

Expressing entries during Gaussian elimination as a quotient of determinants. Hence, bound on the size of numerator/denominator; hence, polynomial algorithm.
8 NP problems; examples

Hamilton cycle, traveling salesman problem.

8.1 Optimization problems

Maximum independent set, minimum node cover.

Turning an optimization problem into a yes/no question (a language).

Example 2. Given graph $G$ and integer $k$, does $G$ have an independent set of size $\geq k$?
8.2 The subset sum problem

Given \(a_1, \ldots, a_n, b\), are there \(x_1, \ldots, x_n \in \{0, 1\}\) with

\[a_1x_1 + \cdots + a_nx_n = b.\]

A dynamic programming algorithm (Theorem 5.5.7). Let \(S_i\) be the set of numbers of form \(a_1x_1 + \cdots + a_ix_i\). Then

\[S_i = S_{i-1} \cup (a_i + S_{i-1}).\]

The complexity of this algorithm can be bounded by

\[(\sum_i a_k) \cdot n\]

times the cost of the algebraic operations involved.

*Is this polynomial?*
9 Reductions

9.1 Different kinds of reduction

Many-one reduction.

Example 3. Euler circuit to Hamilton circuit. This does not show that Euler circuit is as difficult as Hamilton circuit; only that Hamilton circuit is as difficult as Euler circuit.

Turing reduction.

Example 4. ONE-FACTOR: given integers $x, y$, does $x$ has a factor smaller than $y$ (and different from ±1)?

$k$-FACTOR: given integers $x, k$, can $x$ be written as $x = y_1 \cdots y_k$ with $|y_i| > 1$?
9.2 Hardness and completeness

NP-hard problems. (Some kind of reduction is understood.)

Example 5. All NP-complete problems below. Also the halting problem. Also: given a graph, does it have at least $k$ matchings? 

NP-complete problems.

We will see many examples.
10 Satisfiability

10.1 The problem

Logic formulas. Conjunctive and disjunctive normal form.

Satisfiability for conjunctive normal forms: SAT. Clauses, literals.

SAT $\leq$ 3-SAT. Reduction via logic circuits.

10.2 The NP-completeness of SAT

Describe the constraints of the space-time history of a (deterministic or nondeterministic) Turing machine computation by a conjunctive normal form.
11 Other NP-complete problems

11.1 Independent sets
Combinatorial meaning of SAT. Translate the constraints into the independent set problem of a graph.

11.2 Hamiltonian path
Follow the reduction in the Sipser book. Are the separator points needed?
12 Randomized computations

Two uses of randomness: average case and randomization. 

Example 6. Quicksort.

Important example, to come later: primality tests.
12.1 Polynomial identity

Given a function $f(x)$, is it 0 for all input values $x$?

The function may be given by a formula, or by a complicated program.

*Example 7.*

$$\det(A_1 x_1 + \cdots + A_k x_k + A_{k+1})$$

where the $A_i$ are $n \times n$ matrices.

Schwartz’s lemma, and its application to this problem.
12.2 Branching programs

(Sipser book.)

Their equivalence problem.

Read-once branching programs. Solving their equivalence problem using polynomial identity.
13 Some number theory

13.1 Division modulo $m$

**Theorem 4.** For every natural number $m > 1$ and every $u$ relatively prime to $m$, there is a $v$ with

$$uv \equiv 1 \pmod{m}.$$

This $v$ can be found in polynomial time.

**Proof.** Use the Extended Euclidean Algorithm to solve $ux + my = 1$.  

The set of remainders modulo $m$ is called $\mathbb{Z}_m$. It is a **ring** $(+,\cdot, \text{usual rules apply to these operations})$. The set of remainders modulo $m$ relatively prime to $m$ is called $\mathbb{Z}_m^*$. This is a **group** with respect to multiplication (every element has an inverse).

If $m$ is a prime then, of course, $\mathbb{Z}_m^* = \{1, \ldots, m-1\}$. In this case, we can “divide” in $\mathbb{Z}_m$ by any nonzero element, so $\mathbb{Z}_m$ is a **field**.
13.2 Fermat’s theorem

For \( a \in \mathbb{Z}_p^* \), look at the sequence \( a, a^2, \ldots \). Eventually, it begins to repeat. First, it becomes 1. The smallest number \( n \) such that \( a^n \equiv 1 \) is called the order of \( n \). It is easy to show (exercise) that if \( a^k \equiv 1 \) then the order of \( a \) divides \( k \).

**Theorem 5 (Fermat).** Let \( p \) be a prime and \( a \) a number not divisible by \( p \). Then

\[
a^{p-1} \equiv 1 \pmod{p}.
\]

**Proof.** This is true in every group. For a proof:

\[
1 \cdot 2 \cdot \cdots \cdot (p-1) \equiv (a)(2a)\cdots((p-1)a) \pmod{p}.
\]

\[\square\]
So, the order of every number in $\mathbb{Z}_p^*$ divides $(p - 1)$. Is it ever equal to $(p - 1)$?

**Theorem 6 (Primitive root).** *For every prime $p$ there is a $g$ whose order is $(p - 1)$.*

We will not prove this here. Note that $g, g^2, \ldots, g^{p-1}$ runs through all elements of $\mathbb{Z}_{p-1}^*$, that is it generates all elements of this group.
14 Prime tests

Does the Fermat theorem characterize primes? Yes: if \( a^{p-1} \equiv 1 \pmod{p} \) for every \( a \) not divisible by \( p \) then \( p \) is prime.

Does the primitive root characterize primes? Yes: if there is a \( g \) with \( g^{p-1} \equiv 1 \pmod{p} \) and \( g^k \not\equiv 1 \pmod{p} \) for every \( k < p - 1 \), then \( p \) is a prime.

Because of the “every” in these statements, these are not polynomial tests.
14.1 Fermat test

Choose a number $a \in \mathbb{Z}_m$. If $\gcd(a, m) \neq 1$ then $m$ is not a prime. Else test $a^{m-1} \equiv 1 \pmod{p}$. If this fails, reject, else accept.

Fermat-accomplices, Fermat-traitors.

Theorem 7. If there is a Fermat-traitor in $\mathbb{Z}_m^*$ then at least half of all elements of $\mathbb{Z}_m^*$ are Fermat-traitors.

Proof. The set of Fermat-accomplices is a subgroup, and the number of elements of a subgroup always divides the number of elements of the group. (See also a more direct proof.)

We call $m$ a pseudoprime if all elements of $\mathbb{Z}_m^*$ are Fermat-accomplices. We found that the Fermat test is a good probabilistic test for all numbers $m$ that are not pseudoprimes.
14.2 Miller test

Observe: if $m$ is a prime then $x^2 \equiv 1 \pmod m$ implies $x \equiv \pm 1 \pmod m$.

Let $m - 1 = 2^k M$, with $M$ odd, and

$$a_i \equiv a^{M \cdot 2^i} \pmod m.$$ 

If $m$ is prime then at least one of the following holds:

$$a_0 \equiv \pm 1, \ a_1 \equiv -1, \ a_2 \equiv -1, \ldots, \ a_k \equiv -1.$$ 

The Miller test rejects if this is false.

**Theorem 8.** *The Miller test accepts a non-prime only with probability $\leq 1/2$.*

We will not prove the theorem here: see the lecture notes.
15 Pseudorandomness

What is randomness? I prefer to postpone the general discussion: for the moment, agree that when somebody truly tosses a coin (and is not a magician) then he produces a random bit $X$:

$$P[X = 0] = P[X = 1] = 1/2.$$ 

Generally, you cannot get something for nothing. Even a pseudorandom string is somewhat random: it is computed from some seed: $x = G(s)$. But the seed is typically much shorter: $|s| \ll |x|$: for example, $|s| = n$ and $|x| = n^2$. 

15.1 Ensembles

We will compare probabilities as a function of bitstring size $n$. A collection of random strings

$$X^1, X^2, \ldots, X^n, \ldots,$$

where $X^n$ is a string of bits of length $n$, is called an ensemble. It could be “truly” random or “pseudorandom”.

Let $U^n$ be the ensemble of truly random uniformly distributed bit strings: For each $s_1, s_n \in \{0, 1\}$,

$$P(U^n = (s_1, \ldots, s_n)) = 2^{-n}.$$
15.2 Tests

An event function, or test is a function \( f(X) \) with values in \( \{0, 1\} \). Using a test, we can compare two distributions \( X \) and \( Y \) by comparing the expected values

\[
E f(X) = P[f(X) = 1], \text{ and } E f(Y) = P[f(Y) = 1].
\]

**Example 8.** Let \( X^1, X^2, \ldots \) be an ensemble such that \( X^n = (x_1, \ldots, x_n) \) was obtained by tossing a \((1/3, 2/3)\)-biased coin independently \( n \) times.

Test \( f(\cdot) \): take the majority of the first \( (n - 1) \) bits. Output 1 if it equals the last bit, and 0 otherwise.

\[
E f(U^n) = 1/2, \quad E f(X^n) \approx 2/3.
\]
We did not define a pseudorandom sequence, but it clearly will be of the form \( X^n = G(S^k) \), where \( k < n \).

**Example 9.** We define the test \( f(\cdot) \) as follows:

\[
f(Z^n) = \begin{cases} 
1 & \text{if there is an } s \text{ with } G(s) = Z^n \\
0 & \text{otherwise.}
\end{cases}
\]

Then \( f(U_n) \leq 1/2 \) and \( f(X^n) = 1 \).

So, unless we restrict tests, they will always distinguish random sequences from pseudorandom ones. From now on, we require a test to be polynomial-time computable.
We say that two ensembles \( \{X^n\} \) and \( \{Y^n\} \) are **polynomially indistinguishable** if for every polynomial-time computable event function \( f(X) \) we have

\[
\log |\mathbb{E} f(X^n) - \mathbb{E} f(Y^n)| / \log n \to -\infty
\]
as \( n \to \infty \). In other words, \( |\mathbb{E} f(X^n) - \mathbb{E} f(Y^n)| \) becomes smaller than \( 1/p(n) \) for any polynomial \( p(n) \).

An ensemble \( \{X^n\} \) is **pseudorandom** if it is polynomially indistinguishable from \( \{U^n\} \).
15.3 Unpredictability

Another natural definition of pseudorandomness is via unpredictability. We say that the ensemble $X^n$ is polynomially unpredictable if for every polynomial $p(n)$, for every polynomially computable function $F(X)$, for all large enough $n$, writing $X^n = (x_1, \ldots, x_n)$, for all $k < n$ we have

$$
P[x_k = F((x_1, \ldots, x_{k-1}))] \leq 1/2 + 1/p(n).$$

**Theorem 9.** *An ensemble is polynomially unpredictable if and only if it is pseudorandom.*

Proof in the Lovász notes.
15.4 Stretching a seed

Theorem 10. Suppose that we have a generator $G(s)$ stretching a seed $s$ of length $n$ to a sequence of length $n + 1$. Then for any polynomial $p(n)$, we have also a generator $G'(s)$ stretching a seed of length $n$ to length $p(n)$.

Proof sketch. Let $m = p(n)$, we define $G'(s) = b_1 \ldots b_m$. Let $G(x) = G_1(x)G_2(x)$, where $G_1(x)$ is one bit, and

$$x_0 = s, \quad b_{n+1} = G_1(x_n), \quad x_{n+1} = G_2(x_n).$$

For each $i$, the bit $b_i$ is unpredictable from $b_{i+1}, \ldots, b_n$. Indeed, since $G(x)$ is a generator, $b_i = G_1(x_{i-1})$ is unpredictable from $G_2(x_{i-1})$. But $b_{i+1}, \ldots, b_m$ are all computed from $G_2(x_{i-1})$, therefore $b_i$ is also unpredictable from $b_2, \ldots, b_m$. (This is only a sketch since we did not compute the probabilities.)
15.5 One-way functions

In the previous proof we only used the fact that $G_1(x)$ is unpredictable from $G_2(x)$. Of course, if we could compute $x$ from $G_2(x)$ then we could also compute $G_1(x)$. So, it is desirable that the function $G_2(x)$ is polynomially computable function but hard to invert. Precisely:

A polynomial-time function $f(x)$ is a (strong) one-way function if for any polynomial $p(n)$, for any polynomial-time algorithm $A(y)$, if we select $x$ randomly then

$$\Pr[f(A(f(x))) = f(x)] < 1/p(n).$$

Note the difference between the average-time difficulty of inverting $f(x)$ and this. The average is taken over $x$, not over $f(x)$. (Distinction not important if $f(x)$ is 1-1.)
15.6 Generators provide one-way functions

**Theorem 11.** Let $G(x)$ be a pseudo-random generator where $|G(x)| = 2|x|$. Then $f(x,y) = G(x)$ is a one-way function over $\{0,1\}^{2n}$, where $n = |x|$.

**Proof.** Suppose that there is an algorithm $A(\cdot)$ such that

$$P[f(A(G(x))) = G(x)] > 1/p(n).$$

Let us make a test function $T(z)$ distinguishing the uniform distribution $U^{2n}$ over $\{0,1\}^{2n}$ the distribution of $G(X)$:

$$T(z) = \begin{cases} 1 & \text{if } f(A(z)) = z, \\ 0 & \text{otherwise}. \end{cases}$$

Then $ET(G(X)) > 1/p(n)$, while $ET(U^{2n}) \leq 2^{-n}$. \qed
15.7 Hard-core bit

We also used not just the fact that we cannot predict $x$ from $G_2(x)$: we used the fact that we cannot predict even the bit $G_1(x)$.

Let us be given a one-way function $f$. A polynomial-time Boolean function $b(x)$ is a hard-core predicate for $f$ if for any polynomial $p(n)$, for any polynomial-time algorithm $A(y)$, if we select $(x)$ randomly then

$$\Pr[A(f(x)) = b(x)] < 1/2 + 1/p(n).$$

Use of hard-core bits:

**Theorem 12.** If $f(x)$ is a length-preserving, 1-1 one-way function and $b(x)$ is a hard-core bit for $f$, then $x \mapsto b(x)f(x)$ is a pseudorandom generator.

It is also possible to use general one-way functions for pseudorandom generation, but those results are more technical.
15.8 Existence of hard-core bit

It turns out that a hard-core bit can be constructed from every one-way function.

Theorem 13. Let $f(x)$ be a one-way function with $|f(x)| = |x|$. Then the function $g(x, y) = (f(x), y)$ is also one-way over sequences $(x, y) : |x| = |y|$, and the function $b(x, y) = x \cdot y$ (inner product) is a hard-core bit for $g(x, y)$.

The proof of this theorem is somewhat sophisticated, so we skip it.
15.9 Example candidates for one-way function and hard-core bit

Factoring $(p, q) \mapsto p \cdot q$ where $p, q$ are primes.

Discrete logarithm Given: prime $p$, generator $g$ for $p$ and $i < p$, output $(p, g, g^i \mod p)$.

Discrete square root Given positive integers $m$ and $x < m$, output $(m, x^2 \mod m)$. Solvable in probabilistic polynomial time if $m$ is a prime but is considered difficult in the general case.
15.10 Quadratic residuosity

Modulo a prime \( m \), decidable using \( x^{(m-1)/2} \mod p \). In general, it is a difficult problem unless the factorization of \( m \) is known. It is difficult even if we know that one of \( x, -x \) is quadratic.

**Blum primes.** Equivalent conditions:

- \( p \) has form \( 4k - 1 \).
- \(-1\) is not a square.
- Squaring is 1-1 over the quadratic residues, so exactly one square root is quadratic. Called *principal square root*.

Quadratic residuosity is difficult even when \( m \) is the product of two Blum primes, and it is even difficult to decide in 51% of the cases.

Modulo a product \( pq \) of Blum primes, we can find a square root if we know \((p - 1)(q - 1)\) (same as knowing \( p, q \)).
15.11 A presumed hard-core bit

Let $m$ be a product of two Blum primes. For a quadratic residue $x \pmod{m}$, let $b(x)$ be its parity. This is a hard-core bit with respect to the 1-1 one-way function $x \mapsto x^2 \mod m$ among quadratic residues.

Indeed, otherwise we could infer the parity of the principal square root and hence find which of $x, -x$ is quadratic.
15.12 Applications in cryptography

Pseudo one-time pad.

Example 10. \( x_{i+1} = x_i^2 \mod pq, \ b_i = x_i \mod 2. \)
16 Decision complexity

A simple computation model in which one can prove interesting lower bounds.

Example 11 (Sorting).

16.1 The model

Set $A$ of possible inputs, set $B$ of possible outputs. Constant $d$, the number of alternatives. Set of test functions of the form $A \rightarrow \{1, \ldots, d\}$.

Given a function $f : A \rightarrow B$, we are looking for the shallowest $d$-way decision tree deciding it. Its height is $D(f)$. 
Example 12 (Convex hull). Input: points $p_i = (x_i, y_i), i = 1, \ldots, n$, in the plane.

Output: subset of these points, the vertices of the convex hull.

Test functions, two kinds. $x_i < x_j$? Does $p_k$ lay above the line of $(p_i, p_j)$?

First, sort by $x_i$. Induction on $n$; adding $p_n$ to the convex hull. Finding and deleting all the visible points around $p_{n-1}$. Point $p_i$ is visible if $p_{i+1}$ is above the line $(p_i, p_n)$ (addition in the index is $(\text{mod } n - 1)$).

First, find by binary search an invisible point $w$: keep a pair of visible points $p_u, p_v$, one below the line $(p_{n-1}, p_n)$ and one above it. Keep halving the interval $[v, u]$ to update $u$ or $v$.

Once you found an invisible point $w$, find the uppermost visible point $p_b$ in $[n - 1, c]$ and the lowermost one, $p_a$ in $[c, n - 1]$, both by binary search. Delete $p_i$ for $i \in [a + 1, b - 1]$. \diamond
16.2 Nondeterministic decision trees

Guessing the sequence of questions. $D(f, x) =$ the minimum number of variables determining $f(x)$.

$$D_0(f) = \max\{ D(f, x) : f(x) = 0 \}, \quad D_1(f) = \max\{ D(f, x) : f(x) = 1 \}.$$

*Example 13 (Isolated points).* Inputs: graphs. Test functions: existence of a certain edge.

Output $f(G) =$: 1 if the graph has no isolated points.

$D_0(f) = n - 1, \ D_1(f) = n - 1.$

It turns out that $D(f) = \binom{n}{2}$.
Example 14. In a given graph, whether a given set is not independent. One variable for each vertex.

\( D_1(f) = 2, D_0(f) \geq n - \alpha \), where \( \alpha \) is the maximum size of an independent set. (Can you determine the exact value of \( D_0(f) \)? It is always less than \( n \).)

We will see that sometimes \( D(f) = n \). \( \diamond \)

Theorem 14. We have \( D(f) \leq D_0(f) D_1(f) \).

Proof. Every 0-witness intersects every 1-witness. \( \square \)

Formulas. Disjunctive \( k \)-normal form of a monotonic Boolean function \( f \) and \( D_1(f) \).
16.3 Information-theoretic and adversarial lower bounds

\( \log_d t \) if the range of \( f \) has size \( t \).

Weak if the range is small. Here is a different argument, using an “evasive” input.

**Proposition 15.** Assume: \( \exists \) input \( a \in A \), for all \( k \) tests, \( \varphi_1, \ldots, \varphi_k \), there is \( a' \in A \) with \( f(a') \neq f(a) \) but \( \varphi_i(a') = \varphi_i(a) \) for all \( 1 \leq i \leq k \). Then \( D(f) > k \).

(The proof is easy.)

**Example 16.** Finding the minimum of \( n \) elements.
16.4 Symmetry-based lower bounds

**Theorem 15.** Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$. Then $2^{n-D(f)}$ divides $|\{ x : f(x) = 1 \}|$.

**Example 17.** On an even number of points, the number of graphs not containing isolated points is odd. Proof: inclusion-exclusion.

Laconic Boolean functions: $D(f) = n$. 
Symmetry group of a function $f(x_1, \ldots, x_n)$: the set of those permutations $\sigma$ of $x_1, \ldots, x_n$ that leave $f$ unchanged:

$$f(x_{\sigma 1}, \ldots, x_{\sigma n}) = f(x_1, \ldots, x_n).$$

Symmetric function: if its permutation group is the set of all permutations. (These functions are very simple.)

Transitive permutation group $\Gamma$: such that for all $i, j$ there is a $\gamma \in \Gamma$ with $\gamma i = j$.

Weakly symmetric function: if its permutation group is transitive.

Example 18. Let $y = (y_{ij} : 1 \leq i < j \leq n)$ mean the presence or absence of an edge in a graph $G(y)$. Let $f(y_{1,2}, \ldots, y_{n-1,n})$ be some property of the graph: say, whether $G$ has a Hamiltonian cycle. Let $\pi$ be any permutation of the vertices $i = 1, \ldots, n$, and let

$$\pi y = (y_{\pi i, \pi j} : 1 \leq i < j \leq n).$$

Then $f(\pi y) = f(y)$: we just renamed the vertices.\hfill\triangle
**Conjecture:** Each non-constant, monotonic, weakly symmetric Boolean function is laconic.

Proved in the lecture notes, for the case that the number of variables is prime. Also known for the case of graph properties, when the number of points is prime power.
16.5 Algebraic decision trees

The test functions are bounded-degree polynomials, and we test whether the test gives 0, < 0 or > 0. Example: convex hull.

Depth lower bound based on the number of connected components of the set \( \{ x : f(x) = 1 \} \). Uses an algebraic result bounding the number of components of each set found at the leaf of the tree.
16.6 What if some answers are wrong

Game of twenty questions with a liar.

Probabilistic and adversarial error model. Error-correcting codes with feedback.

Example. I thought of a number in $\{1, \ldots, n\}$. Questions of type $x < k$ are only allowed. You must tell in advance the number $q = q(n)$ of questions you will ask. We fix a constant $r$, and I can lie in $rq$ of my answers.

Is there any $q(n)$ (even exponential) that works? (Repeating every question a lot of times does not help.)
Theorem 16. If $r < 1/3$ then you win with $r(n) = O(\log n)$. If $r \geq 1/3$ then I win, even if you are allowed arbitrary questions.

Proof for $r < 1/4$: bracketing.

Proof for $r \geq 1/3$: Let $L(x)$ = the number of lies if $x$ is the actual number. Represent this function as chips on a board.
17 Approximations

17.1 The setting

In case of NP problems, the approximation question makes sense for optimization. We will formulate it only for maximization problems, where we maximize a positive function. For object function $f(x, y)$ for $x, y \in \{0,1\}^n$, the optimum is

$$M(x) = \max_y f(x, y)$$

where $y$ runs over the possible.

For $0 < \lambda \leq 1$, an algorithm $A(x)$ is a $\lambda$-approximation if

$$f(x, A(x)) > \lambda M(x).$$

(We will see that some interesting cases are not covered by this formulation.)
17.2 Greedy algorithms

Try local improvements as long as you can.

*Example 19. MAXIMUM CUT*

Repeat: find a point on one side of the cut whose moving to the other side increases the cutsize.

**Theorem 17.** If you cannot improve anymore with this algorithm then you are within a factor 2 of the optimum.

*Proof.* The unimprovable cut contains at least half of all edges.
17.3 Less greed is sometimes better

What does the greedy algorithm for vertex cover say? The following, less greedy algorithm has better performance guarantee.

\textit{Approx\_Vertex\_Cover}(G)

\begin{verbatim}
C ← Ø
E' ← E[G]
while E' ≠ Ø do
    let (u, v) be an arbitrary edge in E'
    C ← C ∪ \{u, v\}
    remove from E' every edge incident on either u or v
return C
\end{verbatim}

\textbf{Theorem 18.} \textit{Approx\_Vertex\_Cover} has a ratio bound of 2.
17.4 Approximation classes

(1) **Fully approximable**: for every $\varepsilon$, there is a $1 - \varepsilon$-approximation. Example: TWO-MACHINE SCHEDULING (see later)

(2) **Partly approximable**: there is an constant upper bound $\lambda_{\text{max}} < 1$ on the achievable approximation ratio. Example: MAXIMUM CUT, VERTEX COVER, MAX-SAT.

(3) **Inapproximable**: Example: INDEPENDENT SET (deep result). The approximation status of this problem is different from VERTEX COVER, despite the close equivalence between the two problems.
17.5 Fully approximable version of knapsack

Given: $b > a_1 \geq a_2 \geq \ldots \geq a_n$, let $a = a_1 + \cdots + a_n$. For

$$c(x) = x_1 a_1 + \cdots + x_n a_n, \quad x_i = 0, 1$$

find

$$\max \{ c(x) : c(x) \leq b \}.$$ 

Idea for approximation: break each $a_i$ and $b$ into a smaller number of
big chunks, and use dynamic programming.

Let $r > 0, a_i'' = \lfloor a_i / r \rfloor$. Find

$$\max a_1'' x_1 + \cdots + a_n'' x_n$$

subject to $a_1 x_1 + \cdots + a_n x_n \leq b$. 
For the optimal solution, $x_1^*, \ldots, x_n^*$, estimate $c(x'')/c(x^*)$. We have

$$c(x'')/r \geq x_1'' a_1'' + \cdots + x_n'' a_n''$$

$$\geq x_1^* a_1'' + \cdots + x_n^* a_n''$$

$$\geq x_1^* a_1/r + \cdots + x_n^* a_n/r - n = c(x^*)/r - n,$$

$c(x'') \geq c(x^*) - rn$.

Let $r = \varepsilon a_1/n$, then

$$c(x'') \geq c(x^*) - \varepsilon a_1,$$

$$c(x'')/c(x^*) \geq 1 - \varepsilon a_1/c(x^*) \geq 1 - \varepsilon.$$

The amount of time is of the order of

$$na/r = n^2 a/(a_1 \varepsilon) \leq n^3 / \varepsilon,$$

which is polynomial for each fixed $\varepsilon$. 
17.6 MAX3SAT

Formula $F(x) = C_1(x) \land \cdots \land C_m(x)$, maximize the number $\max_x f(x)$ of satisfied clauses.

**Theorem 19.** If every clause has size $\geq k$ then there is a polynomial-time $1 - 2^{-k}$-approximation.

**Proof.**

1. Random choice. The expected value is easy to compute by the addition law:

   $$ Ef(X) = EC_1(X) + \cdots + EC_m(X), $$

   If clause $C_i$ has size $k_i$ then $Ef(X) = \sum_i (1 - 2^{-k_i}) = A$.

2. Note

   $$ Ef(X) = (1/2)E[f(X) \mid X_1 = 0] + (1/2)E[f(X) \mid X_1 = 1] = A. $$

   Fix $x_1 = u_1$ to make that $E[f(X) \mid X_1 = u_1] \geq A$.

3. Continue fixing $x_2, \ldots, x_n$ similarly.
17.7 Maximum independent set

**Theorem 20.** For any $\lambda_1 > \lambda_2 > 0$, if there is a $\lambda_2$-approximation then there is also a $\lambda_1$-approximation.

**Proof.** Graph product $G_1 \times G_2 = (V_1 \times V_2, E)$. There is an edge in $E$ between $(u_1, u_2)$ and $(v_1, v_2)$ iff either there is one in $E_1$ between $(u_1, v_1)$ or either there is one in $E_2$ between $(u_2, v_2)$.

Note that every maximal independent set $S$ of $G$ is of the form $S_1 \times S_2$ where $S_i$ are maximal independent in $G_i$. So, the maximum for $G^k$ ($k$th power) is $r^k$ where $r$ is the maximum for $G$. Approximation in $G^k$ to a ratio $\lambda$ gives approximation in $G$ to a ratio $\lambda^{1/k}$. \qed

This theorem will be used in a negative direction.
17.8 Negative results

It is easy to define artificial problems that are as hard to approximate as NP.

*Example* 20. Traveling salesman problem, wher a tour is a simple cycle going through all points. Hamiltonian path in $G$ to approximate traveling salesman: give huge weights to the edges not in $G$.

But it is much harder to do for natural problems.
17.9 Gap-producing reductions

Let $L \in \text{NP}$, and let us have a maximization problem, with maximum $M(x)$. Polynomial-time function $\phi : \Sigma^* \rightarrow \Sigma^*$ is a gap-producing reduction with polynomially computable factor $0 < \lambda(x) < 1$ if there is a polynomially computable $c(x) > 0$ such that

(a) if $x \in L$ then $M(\phi(x)) \geq c(x)$,

(b) if $x \notin L$ then $M(\phi(x)) < \lambda(x)c(x)$.

If there is such a reduction, and we can approximate $M(x)$ to within a factor $\lambda(x)$ (in polynomial time) then $L \in \text{P}$ in polynomial time.
17.10 Spot-checkable proofs

Let $R(x, y)$ be a polynomial witness relation for an NP language. Witness length $m = p(n)$, so $R(x, y) = 1$, $|x| = n$ implies $|y| = m$. Let $r = r(n) > 0$ be called the degree of randomization, $k = k(n) > 0$ be called the spot-check size. A function

$$\sigma : \Sigma^n \times \{0, 1\}^r \to \{1, \ldots, m\}^k.$$ 

is called a spot-selection function. For random bit string $\omega = (\omega(1), \ldots, \omega(r))$, the value

$$(i_1, \ldots, i_k) = \sigma(x, \omega)$$

is a sequence of spots to check on the witness $y$. Let $y_{\sigma(x, \omega)} = (y_{i_1}, \ldots, y_{i_k})$. A function

$$R' : \Sigma^* \times \Sigma^* \to \{0, 1\}$$

is called the spot-check function. We will write it as $R'_x(u)$. 
We say \( L \in \text{PCP}(r(n), k(n)) \) if there are a spot-selection function \( \sigma(x, \omega) \) and spot-check function \( R'_x(u) \) with degree of randomization \( O(r(n)) \) and spot-check size \( O(k(n)) \) such that

(a) If \( x \in L \) then there is a \( y \) with \( \forall \omega R'_x(y_{\sigma(x, \omega)}) = 1 \).

(b) If \( x \notin L \) then \( \forall y \mathbf{E}_\omega R'_x(y_{\sigma(x, \omega)}) < 1/2 \).

It is easy to see \( \text{PCP}(\log n, 1) \subseteq \text{NP} \) (the number of all possible random choices is polynomial).

Theorem 21. \( \text{NP} = \text{PCP}(\log n, 1) \).

The elaborate proof uses sophisticated arithmetization.
### 17.11 MAX3SAT is hard to approximate

**Theorem 22.** There is a constant $\lambda < 1$ such that MAX3SAT cannot be approximated to within $\lambda$ unless $P = NP$.

**Proof.** Let $L$ be any NP-complete language. For appropriate polynomially computable $c(x), \lambda$, we give a gap-producing reduction from $L$ to MAX3SAT. By Theorem 21 we have $L \in \text{PCP}(\log n, 1)$, so there are constants $k_1, k_2$, a spot-selection function $\sigma(x, \omega)$ and spot-check function $R'(x, u)$ with degree of randomization $k_1 \log n$ and spot-check size $k_2$ such that the desired properties hold. For each $x$, we can construct a 3-CNF $C_x(u, v)$

that is satisfiable if and only if $R'_x(u)$ is satisfiable by some $u$. Its number of clauses is some constant $K_2$ depending only on $k_2$.

The bit string $\omega = (\omega(1), \ldots, \omega(k_1 \log n))$ can take $m = n^{k_1}$ possible values $\omega_1, \ldots, \omega_m$. 


Let us form the Boolean formula

\[ F_x(y) = C_x(y_{\sigma(x,\omega_1)}, v_1) \land \cdots \land C_x(y_{\sigma(x,\omega_m)}, v_m). \]

If \( x \in L \) then it can be satisfied. Otherwise, for each possible “proof” \( y \), for at least \( m/2 \) of the values \( i \), we have \( R'_x(y_{\sigma(x,\omega_i)}) = 0 \) and hence for all possible choices of \( v_i \) at least one clause in 3-CNF \( C_x(y_{\sigma(x,\omega_i)}, v_i) \) will be false. So, we can satisfy at most a percentage \( 1 - 1/(2K_2) \) of all clauses.
17.12 From MAX3SAT to independent sets

Take the usual reduction from 3SAT to independent sets:

Each occurrence of each literal is a point. Points are connected if they are in the same clause or are negations of each other.

$k$ satisfied clauses give $k$ independent points and vice versa.

17.13 Generalization: gap-preserving reduction