

CSC 5450 Randomness and Computation

Week 13: Complexity Theory

- Plan
- Interactive proofs: graph non-isomorphism, non-satisfiability
 - Probabilistically checkable proofs, hardness of approximation
-

Proof

A theorem proving procedure should satisfy three properties:

- ① Completeness: there is a proof for a true theorem.
- ② Soundness: there is no proof for a false theorem.
- ③ Efficiency: one can check the proof efficiently.

Note that the efficiency requirement is only for the verifier, but not the prover (it does not matter how long it takes to find a proof, but it is important that the proof can be verified efficiently).

Recall that NP is the class of problems for which there is a short proof for its YES instances.

For example, for SAT, to prove that a formula is satisfiable, one just needs to provide a satisfying assignment (short proof), and it satisfies the above three properties. Note that checking a proof is easy, but finding a proof may be not, and this is the P vs NP problem.

What about problems not known to be in NP? Can we still have an efficient theorem-proving procedure?

For example, how to prove that a formula is not satisfiable? This problem is not known to be in NP, and we don't expect that there exists a short proof for every non-satisfiable formula as in the case for NP problems.

However, we are going to see that using randomness and interaction, there is an efficient theorem-proving procedure for very general problems (including non-satisfiability).

Also, in the second half, we will see that using randomness one can check the proofs for NP problems much more efficiently.

Interactive Proof

[1, Chapter 8]

The new element here is to allow interactions between the prover and the verifier (where for

NP problems there is no interaction between them: the prover just gave a short proof for the verifier to check).

In the interactive proof setting, there is a prover and a verifier, and they can communicate with each other interactively.

A problem is said to be in IP if there is a communication protocol with the following three properties:

- ① Completeness: for a YES-instance, there exists a (honest) prover that can convince the verifier that it is a YES-instance with probability $\geq 2/3$.
- ② Soundness: for a NO-instance, no prover can convince the verifier that it is a YES-instance with probability $\geq 1/3$ (in other words, any prover will succeed with probability $\leq 1/3$).
- ③ Efficiency: there are at most a $\text{poly}(n)$ rounds of interactions, and in each round the verifier can be done in $\text{poly}(n)$ time. Verifier can access to random bits.

Remarks: - Randomness is crucial; if not this class is equal to NP.

- There is no assumption on the prover's computational power, so even an all-powerful prover cannot fool the verifier.
- The probabilities $2/3$ and $1/3$ are not important. In fact, one can change $2/3$ to 1 (non-trivial) and $1/3$ to an arbitrarily small constant.

An Example Suppose someone shows you two bank notes, which look identical to you, but he claims that they are different (e.g. one is true while another is fake) and he can distinguish them.

Can you think of a way to tell whether he is lying?

There is a simple way using some randomness.

Put the two notes behind your back. Flip a coin. If head then give the prover the first note; if tail give the prover the second note. Ask the prover whether it is the first or the second note.

If the notes are different, then an honest prover can always answer the question correctly, because the prover can distinguish them.

On the other hand, if the notes are the same, then since the prover can't distinguish them and doesn't know your random bit, the prover can only answer this question correctly with probability $1/2$.

Therefore, if we repeat this procedure many times, then we can tell whether the prover is lying or not with high probability.

Graph Non-Isomorphism

Given two graphs G_1 and G_2 , if they are isomorphic, then there is a short proof of this fact by showing a bijection of their vertices.

However, if they are not isomorphic, we don't know of a general method to provide a short proof of this fact that is efficiently verifiable.

There is a simple interactive proof for graph non-isomorphism, using the above idea:

The verifier flips a coin and sends a random permutation of G_i (G_1 if head, G_2 if tail) to the prover, and ask whether it was G_1 or G_2 .

If G_1 and G_2 are different, then an honest prover can always answer the question correctly.

While if G_1 and G_2 are the same, since the prover doesn't know the randomness, the prover can answer correctly with probability $1/2$.

Zero knowledge Proof This is an example where the prover can convince someone that two graphs are non-isomorphic, while not giving out any other information (e.g. what properties are different). This is an idea that is very useful in cryptography (e.g. not to leak commercial secrets).

Private Coin The correctness of the above protocol depends crucially on the assumption that the prover can't see the verifier's private coin flips.

Can we design an interactive proof protocol without this assumption?

This looks impossible, but surprisingly it can be done.

Public Coin

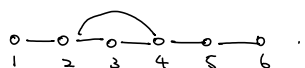
In this setting, the verifier can still use randomness but the prover can also see the random bits.

There is still an interactive proof protocol for graph non-isomorphism.

For this, we take a more global view of the problem.

The idea is to consider $S = \{H : H \cong G_1 \text{ or } H \cong G_2\}$.

For simplicity, suppose G_1 and G_2 have $n!$ ^{different} equivalent graphs, i.e. any permutation gives a different graph, e.g.



Then $|S| = 2n!$ when G_1 and G_2 are non-isomorphic and $|S| = n!$ if they are isomorphic.

(More generally, consider $S = \{(H, \pi) : H \cong G_1 \text{ or } H \cong G_2 \text{ and } \pi \text{ is an automorphism of } H\}$.)

Then, we have the same conclusion that $|S| = 2n!$ if G_1 and G_2 non-isomorphic; otherwise $|S| = n!$.

Now, the problem reduces to the problem of checking whether an (unknown) set is large or small.

We can do this by hashing. Suppose we choose a hash family \mathcal{H} with $\sim 2n^2$ bins.
pairwise independence

Protocol

The verifier sends a random hash function h from the hash family \mathcal{H} , and a random bin number y .

And the verifier request the prover to send an element $x \in S$ such that $h(x)=y$ and a proof that $x \in S$.

Analysis (sketch)

- If $|S|=n^2$, then the prover succeeds with probability $\leq \frac{1}{2}$.
- If $|S|=2n^2$, then it can be shown that the prover succeeds with probability $\geq \frac{3}{4}$, by using pairwise independence and the inclusion-exclusion principle (page 155 of [1]).

Therefore, by repeating this process a few times (in parallel), we can reduce the error probability.

Finally, note that the verifier can be implemented efficiently: a hash function is easy to generate and transmit, and checking whether $h(x)=y$ and $x \in S$ (by checking an isomorphism) can be done easily.

Complexity of Graph Isomorphism (page 156-157 of [1])

The fact that graph non-isomorphism has an interactive proof of the above form (public coins, one round; also known as an Arthur-Merlin proof) is not only interesting on its own, but also has some nontrivial complexity implication.

In particular, it implies that if graph isomorphism is NP-complete (so every NP problem has such a protocol), then the polynomial hierarchy will collapse to the second level (which is believed to be not the case). So, this gives a strong evidence that graph isomorphism is NOT NP-complete.

#SAT in IP [MR 7.7] [1, chapter 8.3]

Can we design an interactive proof protocol for non-satisfiability?

This is non-trivial, but surprisingly there is an interactive proof for an even harder problem, #SAT, the problem of counting the number of satisfying assignments of a boolean formula.

A key idea is arithmetization: encoding a formula as a polynomial.

First, we represent any truth assignment by a 0-1 vector (0 for false, 1 for true).

We would like to represent the formula as a polynomial so that a satisfying assignment will evaluate to one and otherwise zero.

For each clause with, say, three variables, we replace it by a degree three polynomial $1 - \{i\}\{j\}\{k\}$.

For each clause with, say, three variables, we replace it by a degree three polynomial $1 - z_i z_j z_k$,

where $z_i = 1 - x_i$ if x_i is a variable and $z_i = x_i$ if \bar{x}_i is a negated variable.

Then, the polynomial for the formula is $P(x_1, x_2, \dots, x_n) = \prod_{i=1}^m (1 - z_i z_j z_k)$.

For example, if the formula is $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4)$, then $P(x_1, x_2, x_3, x_4) = (1 - (1 - x_1)(1 - x_2)x_3)(1 - x_1x_3(1 - x_4))$.

Given a formula ϕ , let $\# \phi = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \dots \sum_{x_n \in \{0,1\}} P(x_1, x_2, \dots, x_n)$.

Suppose the prover tries to convince the verifier that $\# \phi = k$.

The problem is reduced to a problem about polynomial.

We pick a prime p between $2^n + 1$ and 2^{2^n} , and compute $\# \phi \bmod p$, which is the same as $\# \phi$ since $\# \phi \leq 2^n$.

Let d be the degree of the polynomial. Then $d \leq 3m$ for 3-SAT, and $d \leq mn$ in general.

Protocol

Base case: when $n=1$, we can directly compute whether $\# \phi = k$, and we are done.

So, we assume $n \geq 2$.

Inductive step: The idea is to reduce the problem of checking whether

$k = \sum_{x_1 \in \{0,1\}} \dots \sum_{x_n \in \{0,1\}} P(x_1, \dots, x_n)$ of an n -variable polynomial P to

checking whether $k' = \sum_{x_1 \in \{0,1\}} \dots \sum_{x_{n-1} \in \{0,1\}} P'(x_1, \dots, x_{n-1})$ of an $(n-1)$ -variable polynomial P' .

Let $Q(y) = \sum_{x_1 \in \{0,1\}} \dots \sum_{x_{n-1} \in \{0,1\}} P(x_1, x_2, \dots, x_{n-1}, y)$ be an one-variable polynomial on y .

Observe that $Q(0) + Q(1) = k$.

Note that $Q(y)$ is just a degree d polynomial with one variable, where all the coefficients are at most $p \leq 2^{2^n}$, and so this polynomial can be represented in polynomial space.

Here is the protocol:

① The verifier asks the prover to send $Q(y)$.

The prover then sends the verifier some polynomial $S(y)$ (if the prover is honest, then $S(y) = Q(y)$).

② The verifier checks if $S(0) + S(1) = k$; if not, reject immediately. Otherwise,

the verifier recursively asks the prover to prove that $S(r) = \sum_{x_1 \in \{0,1\}} \dots \sum_{x_{n-1} \in \{0,1\}} P(x_1, \dots, x_{n-1}, r)$, where r is a random element in $\{0, \dots, p-1\}$.

Analysis

If the prover is honest, then $k = \sum \dots \sum P(\dots)$, and in each step the prover just needs

to send the polynomial $Q(y)$, and the prover will succeed with probability one.

So, we focus on the case when the prover is not honest, i.e. $k \neq \sum_{x_1 \in \{0,1\}} \dots \sum_{x_n \in \{0,1\}} P(x_1, \dots, x_n)$.

Then, the prover can not send $Q(y)$ to the verifier, because $Q(0) + Q(1) \neq k$.

Suppose the prover sends some polynomial $S(y)$ such that $S(y) \neq Q(y)$ but $S(0) + S(1) = k$, so that the prover won't be rejected by the verifier right away.

Since $S(y) \neq Q(y)$ and they are just degree d polynomial, by picking a random element r from $\{0, \dots, p-1\}$, $S(r) \neq Q(r)$ with probability at least $1 - \frac{d}{p}$.

Now, the verifier asks the prover to prove $S(r) = \sum_{x_1 \in \{0,1\}} \dots \sum_{x_{n-1} \in \{0,1\}} P(x_1, \dots, x_{n-1}, r)$.

The correct answer for that sum should be $Q(r)$.

So, if $S(r) \neq Q(r)$, then the prover faces the same problem of proving a wrong answer, while the number of variables of the polynomial is decreased by one.

Eventually, if the number of variables is one (the base case) and the prover is still trying to prove a wrong answer, then the verifier will find out.

So, the prover can "cheat" the verifier only if the prover is lucky that $S(r) = Q(r)$ although $S(y) \neq Q(y)$, but this happens with probability at most $n \cdot (\frac{d}{p}) \leq nm/2^{2n}$, which is very small.

Finally, observe that there are at most n rounds, and in each round the verifier can be implemented in polynomial time (since $Q(y)$ is of polynomial size as argued before).

So, this is an interactive protocol for counting satisfying assignments of a boolean formula.

Power of Interactive Proof

Extending the ideas in the above proof, one can show that $IP = PSPACE$, the class of problems computable in polynomial space, which is believed to be a much bigger class than NP .

So, by allowing randomness and interaction, the power of the proof system is significantly increased.

By allowing multiple provers with no communications between them (e.g. police asking different suspects simultaneously), one can prove that $MIP = NEXP$, the class of problems computable in nondeterministic exponential time.

Interestingly, the ideas used in proving $MIP = NEXP$ turns out to be useful in giving a new characterization of NP , which is what we'll discuss in the following.

Probabilistically Checkable Proofs [1, chapter 11]

For 3-SAT, if an instance is satisfiable, the prover can send a satisfying assignment to the verifier, and the verifier can check whether it indeed satisfies all the clauses in linear time.

Can we do it more efficiently using randomness?

Surprisingly, there is a way to write a proof of satisfiability so that the verifier only needs to read three bits of the proof while being reasonably confident that the proof is correct.

More precisely, given an instance of 3-SAT, the verifier can ask the prover to provide a proof in a specific format of $\text{poly}(n)$ bits with the following properties:

- ① Completeness: if the formula is satisfiable, then there exists a prover such that the verifier accepts with probability at least 0.99
- ② Soundness: if the formula is unsatisfiable, then the verifier accepts with probability at most 0.51 (regardless of the prover).
- ③ Efficiency: the verifier only needs to read three bits of the proof (regardless the number of variables and clauses in the formula). The verifier can access to random bits.

(There are different variants, e.g. perfect completeness, arbitrarily good soundness, constant number of bits.)

Hardness of Approximation

The PCP theorem has important implications in hardness of approximation.

It is actually equivalent to the hardness of approximating, say, MAX-3-SAT.

To prove hardness of MAX-3-SAT, one starts from an NP-complete problem, say, 3-SAT, and do a polynomial time transformation from 3-SAT to MAX-3-SAT so that:

- if 3SAT is YES, then the resulting instance of MAX-3-SAT is $\geq 1-\epsilon$ satisfiable.
- if 3SAT is NO, then the resulting instance of MAX-3-SAT is $\leq \frac{7}{8} + \epsilon$ satisfiable.

If such a "gap amplification" reduction exists, it implies that approximating MAX-3-SAT to a factor better than $(\frac{7}{8} + \epsilon)$ is NP-hard, because one can use such an approximation algorithm to determine whether the original instance of 3SAT is satisfiable or not.

Claim There exists such a gap amplification of 3SAT if and only if there exists a PCP (ignoring the exact parameters up to a constant factor).

proof

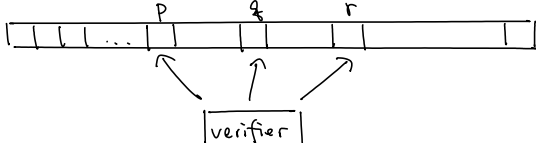
We focus on the direction that PCP \Rightarrow hardness of 3-SAT, since the implication is more important.

Suppose a PCP exists.

Then there is a specific format that the verifier will ask the prover to fill in the proof of $\text{poly}(n)$ bits. Then the verifier, based on the three bits read, will decide to accept or reject.

In the construction of PCP, we can assume that randomness is only used to decide which three bits to read, and after that the decision is deterministic based on the content of the three bits.

(This can be justified since we can improve the completeness to one, while reading more bits.)

Pictorially,  proof with $\text{poly}(n)$ bits.

So, for the positions (p, q, r) , the verifier has a truth table to decide whether to accept or not.

Thus, each query of three bits is just a constraint on the three bits.

The prover knows the strategy of the verifier.

So, for the prover to succeed with the highest probability, it amounts to assigning the bits in the proof carefully, so that the maximum number of queries can be satisfied (since the prover doesn't know the randomness, and each query is equally likely to be asked).

So, the prover is solving a constraint satisfaction problem to solve, where each constraint is on three bits.

The fact that it is a PCP means that if 3SAT is NO, then there is no way to satisfy $\frac{7}{8}$ -fraction of the queries.

And this just means that the 3-CSP is hard to approximate.

Each 3-CSP can be transformed to 3-SAT, by using a constant number of 3-SAT constraints for one 3-CSP constraint (as each truth table can be written using 3-CNF).

This implies that 3-SAT is also hard to approximate for some constant (say 0.9).

This proves one direction.

The other direction is simpler; having such a hard to approximate 3-SAT formula directly implies a PCP, just randomly pick a clause and check whether it is satisfied. \square

Having such a gap for 3-SAT, other hardness of approximation follows, e.g. $\Omega(\log n)$ hardness for minimum set cover, and many other results.

Exponential Size PCP

To give some idea how to prove the PCP theorem, we show a weaker result that there is a way for

the prover to provide an exponential length proof such that the verifier only needs to read three bits. Very roughly, the verifier asks the prover to encode the truth assignment in a very long way, so that it only requires to read a few bits to determine whether it is a codeword.

Walsh-Hadamard Code

Let $u \in \{0,1\}^n$ be n bits.

The Walsh-Hadamard encoding is to represent these n bits by the truth table of the function

$$x \mapsto u \odot x, \text{ where for } x, y \in \{0,1\}^n \text{ we define } x \odot y = \sum_{i=1}^n x_i y_i \pmod{2}.$$

This is very inefficient, since n bits is represented by 2^n bits.

The following simple fact will be used throughout the proof.

Fact If $u, v \in \{0,1\}^n$ and $u \neq v$, then $\Pr(u \odot x \neq v \odot x) = \frac{1}{2}$ if x is a uniform random string of n bits.

Let $WH(u)$ be the Walsh-Hadamard encoding of u .

Then the fact implies that $WH(u)$ and $WH(v)$ differ in 2^{n-1} bits if u and v are different.

So, this is an encoding with very large Hamming distance between codewords.

Local testing of Walsh-Hadamard code

The set of codewords is precisely the set of linear functions from $\{0,1\}^n$ to $\{0,1\}$, i.e.

each codeword u represents a function f such that $f(x+y) = f(x) + f(y)$ for all $x, y \in \{0,1\}^n$, and all linear functions can be represented this way.

To check whether $f \in \{0,1\}^{2^n}$ is a Walsh-Hadamard codeword, a natural test is to pick random $x, y \in \{0,1\}^n$, and check whether $f(x+y) = f(x) + f(y)$.

This is known as the linearity test, and it has the following performance guarantee.

Theorem Let $f \in \{0,1\}^{2^n}$ be such that $\Pr(f(x+y) = f(x) + f(y)) \geq p$, then f is p -close to a linear function, i.e. changing at most $(1-p)$ -fraction of the entries of f gives a linear function. (Think of p as a large constant, say $p = 0.99$.)

By repeating the test a constant number of times, we can get p to be a constant arbitrarily close to one.

This is an important result in property testing. The proof is also very interesting, using basic Fourier analysis in an elegant way. Interested reader is referred to [1, Chapter 22.5].

Local decoding of Walsh-Hadamard code

Local decoding of Walsh-Hadamard code

Suppose that $f \in \{0,1\}^{2^n}$ passes the above linearity test, then f is $(1-\varepsilon)$ -close to a linear function \tilde{f} . We can actually "decode" $\tilde{f}(x)$ with high probability for any $x \in \{0,1\}^n$.

Given $x \in \{0,1\}^n$, pick a random $x' \in \{0,1\}^n$, compute $\tilde{f}(x) := f(x') + f(x+x')$.

Note that with probability $1-2\varepsilon$ (union bound), both $f(x') = \tilde{f}(x')$ and $f(x+x') = \tilde{f}(x+x')$

because f is $(1-\varepsilon)$ -close to \tilde{f} and x' and $x+x'$ are uniformly distributed in $\{0,1\}^n$.

Therefore, any bit of \tilde{f} can be reconstructed with high probability (think of ε a small constant).

PCP Construction

First, we introduce the problem for which we construct a PCP.

The problem is called QUADEQ, a system of quadratic equations over the field of size two, i.e. $\{0,1\} \bmod 2$.

For example,

$$\begin{aligned} u_1 u_2 + u_3 u_4 + u_1 u_5 &= 1 \\ u_2 u_3 + u_1 u_4 &= 0 \\ u_1 u_4 + u_3 u_5 + u_3 u_4 &= 1 \end{aligned}$$

Let m be the number of equations. Then we can write an $m \times n^2$ matrix A in which the i -th row corresponds to the coefficients of the i -th equation.

To find a satisfying assignment to this system, it is equivalent to finding an n^2 -dimensional vector U such that

① $AU = b$ where b is the m -dimensional vector corresponding to the constants on the right hand side;

② $U = u \otimes u$ where \otimes denotes the tensor product of u , i.e. $U_{ij} = u_i \cdot u_j$.

The Verifier

The verifier asks the prover to provide two things:

① a Walsh-Hadamard encoding f of u .

② a Walsh-Hadamard encoding g of $u \otimes u$.

So, the encoding length is $2^n + 2^{n^2}$.

Step 1 Run the linearity test to make sure that both f and g are both 0.999 -close to linear functions \tilde{f} and \tilde{g} . This only requires constant number of bits.

Then, we will use local decoding to read constant number of bits of \tilde{f} and \tilde{g} . Since we only need constant number of bits, with high probability all the bits are computed correctly.

Note that an honest prover will pass with probability one.

Step 2 Check that $g = u \otimes u$, while $f = u$.

The verifier will pick random $r, r' \in \{0,1\}^n$ and check whether $f(r) \cdot f(r') = g(r \otimes r')$; if not, reject immediately.

Let $u = (u_1, u_2, \dots, u_n)$.

In a correct proof, $f(r) \cdot f(r') = \left(\sum_{i=1}^n u_i r_i \right) \left(\sum_{i=1}^n u_i r'_i \right) = \sum_{1 \leq i, j \leq n} u_i u_j r_i r'_j = (u \otimes u) \cdot (r \otimes r')$.

Suppose the prover gives a n^2 -dimensional vector $w \neq u \otimes u$.

We claim that each test catches it with constant probability.

Let W be the $n \times n$ matrix where $W_{ij} = w_{ij}$, and U be the $n \times n$ matrix where $U_{ij} = u_i \cdot u_j$.

By our assumption, $W \neq U$.

Then $g(r \otimes r') = \sum_{1 \leq i, j \leq n} w_{ij} r_i r'_j = r^T W r'$, and

$$f(r)f(r') = (u \otimes r)(u \otimes r') = \left(\sum_{i=1}^n u_i r_i \right) \left(\sum_{i=1}^n u_i r'_i \right) = \sum_{i,j} u_i u_j r_i r'_j = r^T U r'.$$

Since $W \neq U$, by the fact, $r^T W \neq r^T U$ with probability $1/2$, and hence $r^T W r' \neq r^T U r'$ with prob. $1/4$.

Therefore, repeating constant number of times, an incorrect proof will be rejected with prob > 0.9 .

This only requires constant number of bits, as $f(r)$, $f(r')$ and $g(r \otimes r')$ are just three bits of the proof (using the Walsh-Hadamard encoding).

Step 3 Finally, we will check that $g = u \otimes u$ encodes a satisfying assignment.

By passing the first two steps, we assume $g = u \otimes u$ for some u .

To check whether a constraint $\sum_{i,j} a_{ij} u_i u_j = b_\ell$ is satisfied, note that we just need to read one bit, i.e. by looking at the bit of $u \otimes u$ that corresponds to the row $(a_{11}, a_{12}, \dots, a_{1n}, \dots, a_{nn})$.

But checking one bit for each constraint requires m bits, too many.

The final observation is that if one constraint is not satisfied so that $A(u \otimes u) \neq b$, then

by the fact $r^T A(u \otimes u) \neq r^T b$ with probability $1/2$, and $(r^T A)(u \otimes u)$ can be obtained by reading one bit of the encoding of $u \otimes u$.

Therefore by reading a constant number of bits, an unsatisfying assignment will be rejected with high probability.

Overall, by reading a constant number of bits, an incorrect proof will be rejected with high probability, while a correct proof will be accepted with probability one.

Remark: It is much more work to get the PCP theorem, but this weaker result is one component of it.

References

- [1] Arora, Barak. Computational Complexity: A Modern Approach.