CS 761 : Randomized Algorithms , Spring 2023, Waterloo

Lecture 19: Interactive proofs

In general, probabilistic techniques are very useful in checking things (e.g. string equality).

Today, we will see interesting "interactive proofs" of checking graph non-isomorphism and non-satisfiability.

Proof

As argued by Goldwasser, Micali and Rackoff, a proof 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 <u>short</u> 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 .

Interactive Proof

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 <u>prover</u> and a <u>verifier</u>, and they can communicate with each other <u>interactively</u>.

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 poly(n) rounds of interactions, and in each round the verifier can be done in poly(n) time. Verifier can access to random bits.

Remarks : - Randomess 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

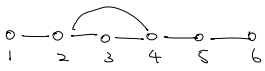
In this setting, the verifier can still use randomness but the prover can also see the random bits.

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

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

Set Lower Bound

Now, the graph non-isomorphism problem is reduced to the problem of checking whether an unknown set is large or not, i.e. whether $|S| \geq 2n!$ or $|S| \leq n!$

The idea is to use <u>hashing</u>: If we have a hash table of size $\sim 2n!$, then an honest prover will succeed with good probability, while if $|S| = n!$ then no one will succeed with probability $\geq \frac{1}{2}$.

Protocol Let $K := 2n!$. Think of K as the set size that the prove would like to establish.

- Let L be an integer such that $\frac{1}{4}2^L \le K \le \frac{1}{2}2^L$. ($2^L$ will be the size of the hash table.)
- The verifier chooses a random function $h: S \to \{0,1\}^L$ from a 2-universal hash family.
- The verifier also chooses a random L-bit string y, and asks the prover to send $x \in S$ such that $h(x)=y$ and a certificate that $x \in S$.
- The verifer then receives x and the certificate that $x \in S$ from the prover, and accepts iff $h(x)=y$ and $x \in S$.

In the graph non-isomorphism problem, the verifier asks the prover to send (H, π) where H is a graph and π is an automorphism of H, and also a certificate that $H \cong G_1$ or $H \cong G_2$ by sending $i \in \{1,2\}$ and a permutation σ such that $\sigma(H) = G_i$.

Analysis: Let $p = K/2^L$, intuitively the "probability" that a cell/bin of the hash table is non-empty.

We will prove that: (i) if $|S| \ge K$, then the accepting probability is at least $3p/4$;

(ii) if $|S| \le K/2$, then the accepting probability is at most $p/2$.

Point (ii) is trivial, as there could be at most $|S| \le K/2 = (p/2)2^L$ non-empty cells in the hash table.

To prove point (i), we show that for any $y \in \{0,1\}^L$, $\Pr_h(\exists x \in S,\ h(x)=y) \ge 3p/4$.

For any $x \in S$, let E_x be the event that $h(x)=y$, and so

$$\Pr(\exists x \in S,\ h(x)=y) = \Pr\left(\bigcup_{x \in S} E_x\right) \ge \sum_{x \in S} \Pr(E_x) - \frac{1}{2}\sum_{x \ne x' \in S} \Pr(E_x \cap E_{x'}) = \frac{|S|}{2^L} - \binom{S}{2}\frac{1}{2^{2L}} \approx \frac{|S|}{2^L}\left(1 - \frac{|S|}{2^{L+1}}\right) \ge \frac{3}{4}p,$$

where the first inequality is by the inclusion-exclusion principle (with truncation), and we have $\Pr(E_x) = \frac{1}{2^L}$ and $\Pr(E_x \cap E_{x'}) = \frac{1}{2^{2L}}$ as the hash family is 2-universal. □

Complexity: We can simply use linear hashing $h_{a,b}(x) = (ax+b) \bmod 2^L$.

So, the verifier can be implemented efficiently, as the hash function can be easily generated and transmitted, and checking whether $h(x)=y$ and $x \in S$ (by verifying the isomorphism and automorphism sent by the prover) can be done easily.

Complexity of Graph Isomorphism

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.

A much stronger evidence is Babai's recent quasi-polynomial time $n^{O(\text{polylog}(n))}$ - algorithm for graph isomorphism.

#SAT in IP

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

This seems very hard, 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 <u>arithmetization</u>: 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 $1-z_iz_jz_k$, 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 $P(x_1,x_2,x_3,x_4) = \big(1-(1-x_1)(1-x_2)x_3\big)\big(1-x_1x_3(1-x_4)\big)$.

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

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^{2n}, and compute $\#\phi \bmod p$, which is the same as $\#\phi$ since $\#\phi \le 2^n$.

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

<u>Protocol</u>

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

So, we assume $n \ge 2$.

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

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

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

Let $Q(y) = \sum_{x_1\in\{0,1\}} \cdots \sum_{x_{n-1}\in\{0,1\}} P(x_1, x_2, \dots x_{n-1}, y)$ 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 $p \le 2^{2n}$, 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 $S(r) = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_{n-1} \in \{0,1\}} P(x_1, \ldots, x_{n-1}, r)$, where r is a random element in $\{0, \ldots, p-1\}$.

Analysis

If the prover is honest, then $k = \sum_{x_1} \sum_{x_2} \cdots \sum_{x_n} P(x_1, \ldots, x_n)$, 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. $k \neq \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} P(x_1, \ldots, x_n)$.

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 $S(y) \not\equiv Q(y)$ but $S(0)+S(1) = k$, so that the prover won't be rejected by the verifier right away.

Since $S(y) \not\equiv Q(y)$ and they are just degree d polynomial, by picking a random element r from $\{0, \ldots, p-1\}$, $S(r) \neq Q(r)$ with probability at least $1 - \frac{d}{p}$.

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

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 $S(y) \not\equiv Q(y)$, but this happens with probability at most $n \cdot (\frac{d}{p}) \le n^2 m / 2^{2n}$, 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, the celebrated PCP (probabilistically checkable proof) theorem, that there is a way to write the certificate that a YES-instance can be checked by reading only 3 bits.

Hope we will have time to discuss the PCP theorem.

References: Arora, Barak. Computational Complexity, chapter 8.

Motwani, Raghavan. Randomized algorithms, chapter 7.