INTERACTIVE PROOFS

AKHIL MATHEW

Abstract. These are notes from a Mathtable talk I gave.

You and I share two large graphs. Waving my magic wand, I discover that they are not isomorphic and rush to tell you. You don't believe me and demand proof; however, as a busy undergraduate, you insist that the proof be verifiable in polynomial time. While I could quickly convince you that two graphs are isomorphic (by exhibiting an isomorphism), it is less obvious how to prove the opposite. Nonetheless, via a suitable probabilistic interaction, I *can* efficiently convince you of the nonisomorphism of the two graphs: this is robust in that if I lie, you'll probably detect it. Moreover, with such "interactive proofs" I can efficiently convince you of the right answer for a large class of computationally difficult problems. I will attempt to convince you of this 1992 result; random bits not required.

1. Introduction

The nicest computational problems are those with an *efficient, deterministic* algorithm. While we shall not formalize the notion of algorithm, you can think of it as something that you could program in your favorite language. By a "problem," we mean a decision problem—the algorithm is to output yes or no. By "efficient," we shall adopt the following convention: an *efficient* algorithm for a problem is one that runs in time polynomial to the length of the input. By "deterministic," we mean that the program does not rely on coin flips or randomness.

1.1. Deterministic interactive proofs.

In general, a problem may have the property that verifying a solution may be much simpler than finding the answer. In other words, it may be very easy for someone to convince you of the answer, but it may be difficult to find the answer. An example is the *graph isomorphism problem*. Suppose given two graphs (represented via their adjacency matrices); the computational task is to determine whether they are isomorphic. There is no obvious way to do so efficiently. For instance, given two graphs of size n, one could look at all permutations of the adjacency matrices, and see if any matched; but this consumes an enormous amount of time. On the other hand, it is very easy for someone with the answer to convince you that two graphs are isomorphic—just present the isomorphism.

Let us now think of this as a "proof" in some sense. In any notion of a proof system that a statement S is true, we should have two conditions:

(1) If S is in fact true, there is a proof of it.
(2) If S is false, no proof of it will be valid.
(3) Checking whether a proof is valid can be done efficiently via some algorithm. However, finding the proof may be very difficult (even unsolvable!).

So we have placed ourselves in the following situation. There are two parties, the prover, whom we'll call Peggy, and the verifier, whom we'll call Victor. Peggy is trying to convince Victor of the truth of a certain statement. Victor and Peggy will exchange a sequence messages—in short, Victor will interrogate Peggy.

If Peggy lies, then Victor should see it; if Peggy tells the truth, however, Victor should nonetheless be convinced. Finally, the key condition is that Victor's computational capacity is much more limited than that of Peggy—he can only run polynomial-time algorithms. In short, Victor is secretly a time-bounded Turing machine.

Definition 1. Such a protocol is called a **deterministic interactive proof.**

Date: April 16, 2011.

1.2. **Graph nonisomorphism.** Let us now consider the following problem: given two graphs G_1, G_2, are they *nonisomorphic*? It is not known whether a purely deterministic protocol as above even exists. Peggy could convince Victor that G_1, G_2 are nonisomorphic by listing all the permutations of G_1—but Victor will get tired and give up.

So, can Victor and Peggy agree upon a better scheme?

To motivate this, let us consider the following situation. You have a red and a green apple that are otherwise identical. You, however, are color-blind, and do not have any way of seeing the difference between the two apples. (This is analogous to Victor's inability to test directly for the nonisomorphism of two graphs.) I tell you, the first one is red and the second one is green. But, it's April 1st, and you don't believe me.

Here's how you could interrogate me to find whether I am telling the truth. Hide the two apples in a separate room. Flip a coin there, and if it comes up heads, bring out the first apple; if it comes up tails, bring out the second. You then ask me what the color of the apple you have brought out is; I respond. Then, you keep doing this over and over.

Now, if I was telling the truth and the apples are different, I will be answering precisely red for the first apple, and precisely the green for the second. But if the apples were actually the same color and I was lying, then I'll have no way to anticipate what you are secretly doing, and I'll be wrong about half the time whatever my strategy.

Motivated by this, let us determine the following protocol for graph nonisomorphism.

(1) Victor and Peggy have two graphs G_1, G_2 of size n; Peggy wishes to convince Victor that they are nonisomorphic.
(2) Victor chooses a random permutation $\pi \in S_n$ and a random bit $b \in \{1, 2\}$. Victor sends $H = \pi(G_b)$ to Peggy, and asks her to tell what b was. Peggy responds.
(3) The previous step is repeated several (say, two) times.
(4) If Peggy gets b right each time, then Victor accepts. If not, Victor rejects.

This is a legitimate means for Peggy to convince Victor. If the two graphs are nonisomorphic, Peggy will be able to determine that the graph H that Victor sends is isomorphic to one of G_1, G_2 and not both; she will thus be able to determine b with certainty. However, if the two graphs are isomorphic, the choice of H provides no information at all about b, and Peggy can at best, so she will be wrong with probability $\frac{1}{2}$. In particular, we find that if Peggy is telling the truth, Victor always accepts. If Peggy lies, then Victor rejects with probability $1 - \left(\frac{1}{2}\right)^2 = \frac{3}{4}$.

Definition 2. Let $\mathcal{C}$ be some claim.

An **interactive proof** that $\mathcal{C}$ is true is an interaction between Victor (a probabilistic Turing machine) and Peggy (who has unbounded computational power). At each round, Victor sends a message to Peggy, who responds; Victor and Peggy are allowed to perform computations.

Suppose the interaction is such that:

(1) If $\mathcal{C}$ is true and Peggy follows the appropriate protocol, Victor will output "true" with probability at least $\frac{2}{3}$.
(2) If $\mathcal{C}$ is false, then no matter what Peggy does, Victor will output "false" with probability at least $\frac{2}{3}$.

Definition 3. A language $\mathcal{L}$ is in **IP** if there is an interactive proof as above for membership in $\mathcal{L}$ such that the total of Victor's computations for a given string of length n is at most polynomial in n.

2. SAT

2.1. **Boolean satisfiability.** A[1] *boolean formula* in variables $x_1, \ldots, x_n$ is just a formula using boolean connectives $(\neg, \wedge, \vee)$ on the boolean variables $x_1, \ldots, x_n$. A boolean formula thus gives a function $\phi : \{0,1\}^n \to \{0,1\}$. A computer can efficiently test whether a boolean formula is *satisfied* by a *truth assignment*, or choice of the $\{x_i\}$.

Thus the problem *Does a boolean formula ϕ have a satisfying assignment?* has an interactive proof. If Peggy wants to prove to Victor that ϕ has a satisfying assignment, she can just send him the truth assignment. (Since this is a purely deterministic interactive proof, it follows that boolean satisfiability

[1]We only consider unquantified boolean formulas here.

is an **NP**-problem—in fact, it is **NP**-complete in that any **NP** problem can be reduced to boolean satisfiability via a polynomial-time reduction.)

However, it is less obvious that Peggy can convince Victor that a boolean formula has *no* satisfying assignment. We shall show that there is an interactive proof for this, and indeed:

Theorem 1. *There is an interactive proof for the claim "ϕ is a boolean formula with k satisfying assignments."*

Note that Peggy cannot simply send the k assignments for Victor to check. There may be exponentially many of them!

2.2. **Arithmetization of boolean formula.** The key idea is that one can *arithmetize* a boolean formula. Namely, we think of a boolean formula ϕ in n variables as a function

$$\{0,1\}^n \to \{0,1\}.$$

We can express this function as a polynomial $P_\phi(x_1, \ldots, x_n)$ as follows. If the formula is one variable $x_i, 1 \leq i \leq n$, we just take x_i as the variable. In general, we can build a formula up by taking conjunctions, negations, or disjunctions of smaller formulas. We then write:

(1) $P_{\not\phi}(x_1, \ldots, x_n) = 1 - P_\phi(x_1, \ldots, x_n)$.
(2) $P_{\phi_1 \vee \phi_2} = (1 - P_{\phi_1})(1 - P_{\phi_2})$.
(3) $P_{\phi_1 \wedge \phi_2} = P_{\phi_1} P_{\phi_2}$.

Now $P_\phi(x_1, \ldots, x_n) = 0$ iff $\phi(x_1, \ldots, x_n)$ is true. So, given ϕ, Victor can easily compute P_ϕ—wait. Peggy is claiming that

$$\sum_{\{0,1\}^n} P_\phi(x_1, \ldots, x_n) = k.$$

Victor has to check Peggy's claim, though actually evaluating the sum would take far too long for him.

So we have reduced to the following situation. Given a polynomial P in n variables, Peggy needs to convince Victor of the value of $\sum_{\mathbf{x} \in \{0,1\}^n} P(\mathbf{x})$. Moreover, if p is a prime of at least n bits, then Peggy only needs to do this mod p, as this sum will be less than p. So here is the interactive protocol:

(1) Peggy sends Victor a prime p of at least n bits in length (no more than $n+1$, say; at least one exists by Bertrand's postulate).
(2) Victor verifies that it is prime.
(3) Peggy convinces Victor that $\sum_{\mathbf{x} \in \{0,1\}^n} P_\phi(\mathbf{x}) = k \mod p$.

This is clearly going to do the trick, provided that we have an interactive proof for the last claim. This is what we describe next.

2.3. **The "sumcheck" protocol.** Suppose P is a polynomial in n variables over the finite field $\mathbb{F}_p$. Here we are going to assume that P is of degree d. While P can have exponentially many coefficients, let us suppose that there is a compact representation of P.

Theorem 2. *There is an interactive protocol that can compute $\sum_{\{0,1\}^n} P(\mathbf{x})$ with error probability $(1 - d/p)^n$.*

To do this, let us define operators σ_i on the space $\mathbb{F}_p[x_1, \ldots, x_n]$ of n-variable polynomials over $\mathbb{F}_p$. We define

$$\sigma_i f = f(\cdot, \cdot, \ldots, \cdot, 0, \cdot, \ldots, \cdot) + f(\cdot, \cdot, \ldots, \cdot, 1, \cdot, \ldots, \cdot)$$

where the 0 and 1 are in the ith place. For the interactive protocol, we need a way for Peggy to convince Victor that $\sigma_1 \ldots \sigma_n P(0, 0, \ldots, 0)$ is some given quantity.

We shall describe an interactive proof for the following more general problem: If P is a polynomial that the prover can evaluate efficiently, then compute $\sigma_{i_1} \ldots \sigma_{i_k} P(a_1, \ldots, a_n)$ for some $a_1, \ldots, a_n \in \mathbb{F}_p$.

When $k = 0$, this is trivial; the verifier can just plug the value into the polynomial. In general, suppose there is an interactive protocol for computing $\sigma_{i_2} \ldots \sigma_{i_k} P(a_1, \ldots, a_n)$; then we will construct an interactive proof for computing $\sigma_{i_2} \ldots \sigma_{i_k} P(a_1, \ldots, a_n)$.

More generally, let us consider the following problem. Suppose that g is some polynomial and that, for any $a_1, \dots, a_n \in \mathbb{F}_p$, Peggy can convince Victor of the value of $g(a_1, \dots, a_n)$ with probability $\rho \simeq 1$, and using T steps.

The claim is that Peggy can convince Victor of the value of $\sigma_i g(a_1, \dots, a_n)$ (for any $a_1, \dots, a_n$, again) in time $T + O(1)$ and with probability $\rho(1 - d/p)$.

Without loss of generality, $i = 1$. One naive approach would be for Victor to ask Peggy to convince him of the values of $g(0, a_2, \dots, a_n)$ and $g(1, a_2, \dots, a_n)$, and then to add them. This won't do, however; each step will take T time, so the total time cost becomes $2T + O(1)$, where the constant depends on n and p but not on the polynomial. Since for our ultimate goal this will be done recursively n times, we will end up with an exponential-time algorithm, which is bad.

So, instead, Victor asks Peggy to send the value of $h(T) = g(T, a_2, \dots, a_n)$. This is a one-variable polynomial of degree $\leq d$, so it can be transmitted efficiently. Victor then computes $h(0) + h(1)$ and checks that it is indeed the value that Peggy claims for $\sigma_1 g(a_1, \dots, a_n)$.

But now Victor has to check that Peggy is telling the truth. It is possible that Peggy sends the wrong polynomial $h(T)$, and Victor needs to guard against that. To do this, Victor chooses a random $b \in \mathbb{F}_p$, and evaluates $h(a)$. Victor asks Peggy to convince him of the value of $g(a, a_2, \dots, a_n)$. If they are equal, Victor relaxes with confidence that Peggy's polynomial $h(T)$ was the right one, and accepts. Otherwise, he rejects.

We need to check several things:

(1) The above interactive protocol runs in time $T + O(1)$. This is clear because Peggy was asked to convince Victor of a value of g only once.
(2) If Peggy always tells the truth, Victor accepts. Indeed, if Peggy tells the truth, then Victor will be convinced of the value of $g(a, a_2, \dots, a_n)$, and Peggy will also send the right polynomial, so Victor will find that $h(a) = g(a, a_2, \dots, a_n)$ and will thus accept.
(3) If Peggy lies, then Victor rejects with probability $\rho(1 - \frac{d}{p})$. Indeed, in this case, if Peggy lies about the value of h, then with probability $1 - \frac{d}{p}$ we will have $h(a) \neq g(a, a_2, \dots, a_n)$ since these are one-variable polynomials of degree d, and can coincide at at most d input values. With probability ρ, Peggy will not be able to convince Victor that $g(a_2, \dots, a_n)$ is actually equal to the value $h(a)$. The claim follows.

3. Conclusion

We have thus seen that interactive proofs allow many one party to convince another of the truth of a statement that may be computationally difficult to test directly. Namely, we have seen that the counting problem for boolean satisfiability admits an interactive proof. By Toda's theorem, it follows that any problem in the polynomial hierarchy admits one.

The ultimate result, however, is one that we do not have time to prove:

Theorem 3 (Shamir, 1992)**. IP = PSPACE**.

Recall that a language is in **PSPACE** if there is an algorithm running in polynomial space that tests for membership. Intuitively, if there is an interactive protocol for a language, then it is possible to simulate all possible interactions in polynomial space (if we reuse space!) by a deterministic Turing machine, and thus to determine what happens in each case, and consequently the appropriate probabilities. From this membership can be decided.

The other direction is the interesting part. In fact, though, most of the work has already been done. The key point is that the problem of determining whether a *totally quantified* boolean formula is true is **PSPACE**-complete, meaning that any **PSPACE**-problem can be reduced efficiently to it. So, in other words, it suffices to give an interactive proof that a totally quantified boolean formula is true.

The arithmetization of totally quantified boolean formulas can be carried out similarly. For instance, if we have arithmetized $\phi(x_1, \dots, x_n)$ via a polynomial $P(x_1, \dots, x_n)$, then we can arithmetize $\exists x_1 \phi(x_1, \dots, x_n)$ by the polynomial $1 - (1 - P(0, x_2, \dots, x_n))(1 - P(1, x_2, \dots, x_n))$. There is a subtlety that the degrees have to be lowered, or these products will have exponentially large degree. But if we use these operations, and employ appropriate degree-lowering operators, we can keep the degrees

at each stage appropriately small. In this way, we can give an inductive and efficient protocol that Peggy can use to convince Victor of the value of these polynomials.

References

1. Sanjeev Arora and Boaz Barak, *Computational complexity: A modern approach*, Cambridge University Press, 2009.