Lecture 19:
Interactive Proofs and the PCP Theorem

Valentine Kabanets

November 29, 2016

1 Interactive Proofs

In this model, we have an all-powerful Prover (with unlimited computational prover) and a polytime Verifier. Given a common input x, the prover and verifier exchange some messages, and at the end of this interaction, Verifier accepts or rejects.

Ideally, we want the verifier accept an input in the language, and reject the input not in the language. For example, if the language is SAT, then Prover can convince Verifier of satisfiability of the given formula by simply presenting a satisfying assignment. On the other hand, if the formula is unsatisfiable, then any "proof" of Prover will be rejected by Verifier. This is nothing but the reformulation of the definition of NP.

We can consider more complex languages. For instance, UNSAT is the set of unsatisfiable formulas. Can we design a protocol so that if the given formula is unsatisfiable, Prover can convince Verifier to accept; otherwise, Verifier will reject? We suspect that no such protocol is possible if Verifier is restricted to be a deterministic polytime algorithm. However, if we allow Verifier to be a *randomized* polytime algorithm, then indeed such a protocol can be designed.

In fact, we'll show that every language in PSPACE has an interactive protocol. But first, we consider a simpler problem #SAT of counting the number of satisfying assignments to a given formula.

1.1 Interactive Proof Protocol for #SAT

The problem is #SAT: Given a cnf-formula $\phi(x_1, \ldots, x_n)$, compute the number of satisfying assignments of ϕ.

We want to construct an interactive proof protocol for this language. It will turn out very useful to "arithmetize" the formula ϕ. Given a cnf $\phi(x_1, \ldots, x_n)$, we can construct a multivariate polynomial $p(x_1, \ldots, x_n)$ of total degree at most $|\phi(x_1, \ldots, x_n)|$ such that, on any Boolean string $a_1, \ldots, a_n$, we have

$$p(a_1, \ldots, a_n) = \begin{cases} 1 & \text{if } \phi(a_1, \ldots, a_n) \text{ is True,} \\ 0 & \text{otherwise.} \end{cases}$$

The construction of such a polynomial is by induction. For the formula x, we define the corresponding polynomial to be x. For $\bar{x}$, we define the polynomial $1 - x$. For $\alpha \wedge \beta$, where α, β are formulas with corresponding polynomials p_a and p_b, we define the polynomial $p_a p_b$ (the product of

the two polynomials). For $\bar{\alpha}$, we define the polynomial $1 - p_a$. Finally, for $\alpha \vee \beta$, we define the polynomial $1 - (1 - p_a)(1 - p_b)$ (to see why this works, use de Morgan's laws).

Claim 1. *For a given formula $\phi(x_1, \ldots, x_n)$, the above transformation produces a polynomial $p(x_1, \ldots, x_n)$ such that*

1. *for every $\vec{a} \in \{0,1\}^n$, $p(\vec{a}) = \phi(\vec{a})$ (where we equate "True" with 1, and "False" with 0), and*
2. *the degree of p is at most the size of ϕ.*

Proof. The proof of each item is by induction on the formula structure. □

Say that $p(x_1, \ldots, x_n)$ is such a translation of cnf ϕ into a polynomial. Then the number of satisfying assignments for ϕ is exactly $\sum p(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ range over all assignments in $\{0,1\}^n$.

In our protocol, we'll ensure that an honest prover always convinces the verifier to accept (by behaving honestly in each round). On the other hand, a cheating prover will be forced to cheat in each round. But the instances will get easier with each round, so that in the last round a cheating prover will be found out even by our randomized polynomial-time verifier.

We want to design a protocol for the following problem: Given a polynomial $p(x_1, \ldots, x_n)$ of degree at most $d \leq n^{O(1)}$, compute

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_n=0}^{1} p(x_1, x_2, \ldots, x_n).$$

The prover will send the verifier some number k, claiming it to be the correct answer for the summation problem above.

We'll describe a verifier that, after interacting with the prover,

- will accept with probability 1 if the prover is honest,
- will reject with high probability if the prover is dishonest.

More precisely, in the first round, the Prover will send the verifier some information, and the verifier will respond with the new challenge for the prover: prove that $k' = \sum p'(x_2, \ldots, x_n)$, for some new k' and polynomial p'.

After the first round, the following will be true. If $k = \sum p(x_1, \ldots, x_n)$ was true before this round, then $k' = \sum p'(x_2, \ldots, x_n)$ is true with probability 1 after this round. If $k = \sum p(x_1, \ldots, x_n)$ was false before this round, then $k' = \sum p'(x_2, \ldots, x_n)$ is false with *very high probability* after this round.

In other words, true equalities will be preserved with probability 1. But, false equalities, with high probability $(1 - \epsilon)$, will be replaced with new false equalities (here, ϵ is the error probability).

Let's fix a prime number $q > 2^n$. All our computation will be modulo q.

Define the polynomial

$$f_1(x_1) = \sum_{x_2} \cdots \sum_{x_n} p(x_1, x_2, \ldots, x_n).$$

Clearly, the degree of f_1 is at most that of p.

In the first round of the protocol, Prover is supposed to send over the coefficient of the polynomial $f_1(x_1)$. In reality, Prover may or may not send the actual polynomial f_1. We denote by $g_1(x_1)$ the polynomial actually sent by Prover.

Verifier will check if $g_1(x_1)$ is of low degree. If not, the Verifier rejects.

Then Verifier checks if $g_1(0) + g_1(1) = k$. If not, Verifier rejects.

Finally, Verifier picks a random number r_1 from the set Z_q, and sends this r_1 to Prover.
In the next stage, Verifier wants to be convinced that

$$g_1(r_1) = \sum_{x_2} \cdots \sum_{x_n} p(r_1, x_2, \ldots, x_n).$$

Let's analyze this protocol. First, consider the case of an Honest Prover, the prover that follows the correct protocol. The prover starts off by giving the verifier the correct value k. Then, this prover sends over to Verifier the correct function $g_1 = f_1$. Since $k = f_1(0) + f_1(1)$, the honest prover will pass the checks of the verifier. Also, since $f_1 = g_1$, and $f_1(r_1) = \sum_{x_2} \cdots \sum_{x_n} p(r_1, x_2, \ldots, x_n)$ by definition of f_1, the honest prover will be faced with a correct equality in the next round. So, the honest prover will always be able to convince the verifier to accept.

Now consider the case of a Dishonest Prover. In the beginning, this prover gives a wrong value for k. That is, the prover gives some number m which is *not* equal to $\sum_{x_1} \cdots \sum_{x_n} p(x_1, \ldots, x_n)$. Then the prover sends over a polynomial g_1. The verifier checks if $g_1(0) + g_1(1) = m$. If not, the verifier rejects. Otherwise, we know that $g_1(0) + g_1(1) \neq f_1(0) + f_1(1)$, and so $g_1 \neq f_1$. By picking a random r_1, the verifier creates a new instance

$$g_1(r_1) = \sum_{x_2} \cdots \sum_{x_n} p(r_1, x_2, \ldots, x_n).$$

The right-hand side is $f_1(r_1)$. Since g_1 and f_1 are distinct low degree polynomial, the probability that they agree on a random point from Z_q is at most (the degree of $f_1)/q$, which is $o(1)$ for large enough n. Thus, with probability $1 - o(1)$,

$$g_1(r_1) \neq \sum_{x_2} \cdots \sum_{x_n} p(r_1, x_2, \ldots, x_n),$$

and so the dishonest prover is faced with an incorrect equality in the new round. Thus, after n rounds, with high probability, the dishonest prover will claim that two distinct numbers are equal, which will be caught by the verifier, and the verifier will reject.

This concludes the description of the interactive proof protocol for #SAT.

1.2 Generalizing to PSPACE

We formally define a new complexity class IP (Interactive Proof) as follows: a language $L \in \mathsf{IP}$ if there is a randomized polytime verifier V such that, for every input $x \in \{0,1\}^n$,

- if $x \in L$, then there is a Prover that convinces the verifier V to accept with probability 1,
- if $x \notin L$, then every prover will be rejected by the verfieir V with probability at least $1/2$.

We have the following result.

Theorem 1 (Lund,Fortnow,Karloff,Nisan'92; Shamir'92)**.** $\mathsf{IP} = \mathsf{PSPACE}$

The direction IP $\subseteq$ PSPACE is fairly straightforward: we can evaluate in PSPACE a "game tree" of the interaction between the Prover and the Verifier, computing the maximum acceptance probability of the Verifier (over all possible prover strategies). (See the textbook for more details.)

A more interesting is the direction PSPACE $\subseteq$ IP. We will just outline some ideas of the proof below.

First recall a PSPACE-complete problem TQBF: Given a quantified Boolean formula

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

decide if it's true. We can arithmetize ϕ as before, getting a polynomial $p(x_1, \ldots, x_n)$ that agrees with ϕ on all Boolean n-tuples. Then we can define an arithmetic expression:

$$\sum_{x_1=0}^{1} \prod_{x_2=0}^{1} \sum_{x_3=0}^{1} \ldots p(x_1, x_2, x_3, \ldots, x_n).$$

It's not hard to argue that the original qbf is true iff the above arithmetic expression evaluates to a non-zero.

We can try to run a protocol similar to #SAT, replacing the "sum check" $(g_1(0) + g_1(1) = m)$ with the "product check" $(g_1(0) * g_1(1) = m)$ for the $\prod$ sign.

One important technical difficulty is that if we do it naively, the degrees of univariate polynomials that our Prover will be asked to send us will be too big! There is a trick (to modify the arithmetic expression above, by introducing extra variables — see, e.g., the text for the full proof) that will ensure that all these polynomials are of low degree. Thereafter, our "sum check/product check" protocol will work.

Thus, it can be shown that every PSPACE-problem can be solved by an interactive proof protocol, using randomized polytime verifier and a polynomial number of rounds of interaction with the prover!

1.3 Multiple Provers

We can generalize our Interactive Proofs by allowing 2 Provers. These provers may agree in advance on their joint strategy, but cannot communicate with each other during the actual protocol (interaction) with the Verifier. (As before, the verifier is polytime randomized algorithm, and the number of rounds is polynomially bounded.)

Clearly, the new class of 2-prover interactive proofs (denoted 2IP or MIP) should contain PSPACE (as it is a generalization of the class IP). It turns out that

Theorem 2 (Babai, Fortnow, Lund, 1991)**.** MIP = NEXP.

Here NEXP is nondeterministic exponential time (an exponential-time version of NP).

2 PCP Theorem

Scaling the above result that MIP = NEXP "down to NP", one gets the PCP Theorem; here "PCP" stands for Probabilistically Checkable Proofs. Rather than multiple provers, we have a single proof π that is written down before the Verifier starts the verification task. The Verifier has limited access to that proof, though: it is allowed to read only a constant number of symbols of the proof,

albeit the verifier is free to choose which positions it wants to see. The verifier uses random coin flips to decide on the positions in the proof to see.

The PCP Theorem basically says the following:

Theorem 3 (PCP Theorem)**.** *Every language L in* NP *has a verifier V such that*

- *V is randomized polytime algorithm,*
- *V reads only a constant number of symbols in a given "proof",*

and such that, for every $x \in \{0,1\}^n$,

- *if $x \in L$, then there is a proof π such that $V^\pi(x)$ accepts with probability at least $2/3$ (where V^π means that V has random access to the proof π: V can read any chosen symbol of the proof, but remember that V is allowed to read only a constant number, say 3, of symbols of π),*
- *if $x \notin L$, then every candidate proof π is such that $V^\pi(x)$ rejects with probability at least $2/3$.*

The PCP Theorem was proved by Arora, Safra, and Arora, Lund, Motwani, Sudan, Szegedy in two papers from 1998 that in turn built upon rich body of work on interactive proof systems done earlier by many researchers. Later, the authors of these two papers have been honored with the Gödel prize for their contribution to computer science.

An important application of the PCP Theorem is to proving that certain optimization problems are not only NP-hard to solve in the worst case, but also remain NP-*hard to approximate* to within certain approximation ratios.

2.1 Hardness of Approximation

Recall that a Vertex Cover Minimization problem is: Given a graph $G = (V, E)$ find a smallest-size set $S \subseteq V$ that covers all the edges of G (i.e., every edge in G shares at least one of its endpoints with the set S). We know that the decision version of Vertex Cover is NP-complete. Hence, the Minimization version is NP-hard. Assuming that $P \neq NP$, this means that there is no polytime algorithm that finds an optimum-size vertex cover in any given graph.

If we can't hope to find a smallest vertex cover, perhaps we can efficiently find a set cover which is "almost" optimal. We say that the Vertex Cover Minimization problem can be efficiently *c-approximated*, for some $c \geq 1$, if there is a polytime algorithm that, on any input graph G, returns a vertex cover S of G such that $|S_{opt}| \leq |S| \leq c|S_{opt}|$, where S_{opt} is a smallest-size vertex cover for G.

For some c, such an approximation algorithm exists.

Theorem 4. *VC is efficiently 2-approximable.*

Proof. Let $G = (V, E)$ be any input graph. Here's the algorithm for finding a vertex cover S of G.

Initially set $S = \emptyset$. Until there are no edges left, repeat the following: Take any edge $e = \{u, v\}$. Add vertices u and v to S. Remove e from G, as well as remove all edges that touch e (at u or v).

Clearly, the described algorithm runs in polytime, and produces a vertex cover S. Next we argue that $|S| \leq 2|S_{opt}|$. To this end, observe that the set S corresponds to (the endpoints of) a set of mutually disjoint edges of G. Any vertex cover of G (including an optimal S_{opt}) must cover each of these $|S|/2$ edges; so every vertex cover must contain at least one vertex for each of these edges. This means that $|S_{opt}| \geq |S|/2$, as promised. □

Now that we have a 2-approximating algorithm for VC, we may ask for a 3/2-approximating algorithm, or more generally, for a $(1+\epsilon)$-approximating algorithm for any $\epsilon > 0$. If such an algorithm exists, then it's almost as good as being able to solve VC optimally. Does it exists???

It turns out that the answer is "No, unless $\mathsf{P} = \mathsf{NP}$". In other words, it is NP-hard to approximate VC to within some constant $1 < \alpha \leq 2$ (currently, the best known value of α is $10\sqrt{5} - 21 \approx 1.3607$, due to a result by Dinur and Safra from 2001). This result (as well as many similar results for other optimization problems) is proved using the machinery of PCPs (and is technically quite involved).

Note that it is hopeless to try to show *unconditionally* that VC cannot be γ-approximated. This is because if $\mathsf{P} = \mathsf{NP}$, then certainly VC can be 1-approximated. Thus, proving that VC is NP-hard to approximate is the next best thing.

Remark This can be viewed as a generalization of NP-hardness from exact optimization problems to approximation problems.