CSE 531: Computational Complexity I **Winter 2016**

Lecture 18: Interactive Proofs

Mar 7, 2016

Lecturer: Paul Beame *Scribe: Paul Beame*

1 Background and Motivation

In the 1980's two notions interactive computation were developed. One, due to Babai, originated in generalizations of NP to allow more powerful verifiers that include probabilistic verification. The other, due to Goldwasser, Micali, and Rackoff, originated in cryptography and was a means to the end of defining zero-knowledge proofs, protocols that allow a party in a cryptographic protocol to convince another party of some property without revealing additional information. (In this course we will not discuss the zero-knowledge aspects, however.) The definitions here began research on the path to the PCP theorem.

2 Interactive Proof Preliminaries

We can view a typical NP algorithm as an interaction between an all-powerful *prover* P and a deterministic polynomial-time bounded *verifier* V. On a shared input x that the verifier wishes to prove is in L, the prover produces y, a certificate depending on x, sends the certificate y to the verifier and the verifier accepts if and only if $V(x, y) = 1$. This can be thought of as one round of interaction. The exact power of the prover is not important here but everything still must be verifiable by the deterministic polynomial time machine. So perhaps the amount of interaction is important? What if we were to allow more rounds of interaction as in the following figure?

Formally, we say that a *round* of an interactive protocol corresponds an uninterrupted sequence of communications of a single party. In this picture $y_1, y_2, \ldots, y_\ell$ denote the messages sent by the prover and $z_1, z_2, \ldots, z_{\ell-1}$ denote the messages sent in response by the verifier. We can formally define the verifier as an algorithm V and the actions of the verifier are given by by $z_1 = V(x_1, y_1)$, ..., $z_\ell = V(x, y_1, z_1, y_2, z_2, \ldots, y_\ell)$. The prover's actions can be defined similarly. The (still deterministic) polynomial time verifer accepts iff $V(x, y_1, z_1, y_2, z_2, \ldots, y_\ell) = 1$ at the end of the computation. (This is $2\ell - 1$ round computation.)

Lemma 2.1. *Interactive proofs with deterministic polynomial-time verifiers yields proofs for precisely the languages in* NP.

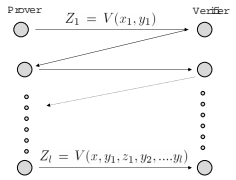

Figure 1: Prover Verifier Interaction

Proof. To see this, notice that for a multi-round proof the prover can determine in advance what the verifier will say at each response and simply send the entire sequence $(y_1, z_1, \ldots, y_\ell)$ to the verifier which yields a single round of interaction and thus NP. □

In each of the protocols it sufficed for the prover to have the power of NP in order to execute the protocol. However, our limits on the power of the verifier were what restricted the set of languages for which we had interactive proofs.

3 Arthur-Merlin Games

In order to have an interesting notion of interactive proofs we need to include randomness on the part of the verifier. We begin with a simple version of interactive protocols, in which the Prover (the all-powerful wizard Merlin) is allowed access to all the random coins used by the Verifier (the not-so smart king Arthur) so far. With this definition it suffices to assume that the strings that are sebt by the verifier simply contain the random guesses used by Arthur since it is clear that Merlin, who is all powerful, could easily have computed whatever string Arthur would have computed based on these random guesses.

The key is that Merlin is unaware of the outcomes of the coin before the Arthur sends them to him. Notice that Merlin is powerful enough to simulate all possible flips of the coin ahead of time and therefore can play optimally. The game ends with a polynomial time deterministic verification based on the input and the message exchanged. We now use r_i to denote the i-th random string that Arthur sent. The acceptance condition is BPP-like. More precisely, there is a deterministic polynomial time verifier A and a prover M such that if $(r_1, y_1, \ldots, y_\ell)$ is the sequence of communicated values on input x, where each r_i is chosen uniformly at random, then $\mathbb{P}_{r\in\{0,1\}^{|x|^{O(1)}}}[A(x, r_1, y_1, ..., y_k) = 1] > 1 - \epsilon$ if $x \in L$ and for all choices of $(y_1^*, \ldots, y_k^*)$, $\mathbb{P}_{r\in\{0,1\}^{|x|^{O(1)}}}[A(x, r_1, y_1, ..., y_k) = 1] < \epsilon$ if $x \notin L$. We count rounds in terms of half round

trips and either party is allowed to go first (so r_1 may be omitted).

3.1 Arthur-Merlin Complexity Classes

We denote the complexity class of languages accepted by Arthur-Merlin protocols by an alternating sequence of letters A and M where the number of letters equals the number of rounds. For example, AMA = $\{$ L | there is a 3 round Arthur-Merlin protocol for L in which Arthur starts$\}$.

Notice that ending at an A is like having a BPP machine to do the verification.

Consider the following view of AM:

$L \in \mathsf{AM}$ if and only if there is a polynomial-time verifier V, and a polynomial p such that $\forall x$,

$$\mathbb{P}_{r\in\{0,1\}^{p(|x|)}}[\exists y \in \{0,1\}^{p(|x|)}\ V(x,y,r)=1] \text{ is } \begin{cases} \geq 1-\varepsilon & \text{if } x \in L \\ \leq \varepsilon & \text{if } x \notin L \end{cases}$$

The first quantifier acts like a BPP machine that makes one call to an NP oracle accepting its answer; so $\mathsf{AM} = \mathsf{BP}\cdot\mathsf{NP}$. The *value* of the Arthur-Merlin game is defined to be the probability that Arthur (the verifier) is convincedm given optimal play by Merlin. We can express the value of the Arthur-Merlin game on input x using a form of quantifier, namely A as an Averaging quantifier and M as a Maximization quantifier. The value of the above AM protocol is then $Ar \in \{0,1\}^{p(|x|)} My \in \{0,1\}^{p(|x|)} y.\ V(x,y,r)=1$. This use of A and M as the quantifiers is the source of the names Arthur and Merlin.

Now consider MA. In this case after doing all the nondeterministic work, we get access to a BPP machine, so we get $\mathsf{MA} = \mathsf{N}\cdot\mathsf{BPP}$. In quantifiers with a similar verifier the game would have a value $My \in \{0,1\}^{p(|x|)}\ Ar \in \{0,1\}^{p(x)}.\ V(x,y,r)=1$.

Definition 3.1. *Define*
$\mathsf{AM}[\mathsf{k}] = \{\mathsf{L} \mid \mathsf{L}$ *has a* k *round Arthur-Merlin game with Arthur starting*$\}$.

Theorem 3.2 (Babai, Babai-Moran)**.** *For constant $k \geq 2$,* $\mathsf{AM} = \mathsf{AM}[\mathsf{k}] = \mathsf{MA}[\mathsf{k}+1]$*, moreover, for any $t(n)$.* AM*[2t(n)]* $\subseteq$ AM*[t(n)+1].*

The general idea is that with some blow-up in size (and failure probability) one can swap quantifiers. In particular AMA quantifiers can be transformed to MAM quantifiers, which can be done in parallel. Finally, MAM can be converted to AM.

Clearly $\mathsf{NP} \subseteq \mathsf{MA} \subseteq \mathsf{AM}$. By methods similar to the proof that $\mathsf{BPP} \in \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$, we get

Lemma 3.3. $\mathsf{AM} \subseteq \Pi_2^{\mathsf{P}}$ *and* $\mathsf{MA} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$.

(Indeed the result for MA is immediate from the quantifier form for BPP.)

4 Interactive Proofs

The key difference with Arthur-Merlin games is that in this case, instead of being public, the Verifier's random string r is hidden from the Prover who can only see the outputs based on r rather than r itself. We assume that The randomized verifier V runs in polynomial time as a function of the length of its inputs and that all messages are polynomial-length as a function of $|x|$.

Again acceptance is BPP-like. It is common to say that the verifier is *convinced* if it accepts the interactive computation.

Again we can define a complexity class of these interactions.

Definition 4.1. *Define* $\mathsf{IP}[\mathsf{k}(\mathsf{n})]$ *to be the set of languages* L *such that given an* $x \in L$ *there is a (polynomial-time randomized) protocol with private randomness that takes at most* $k(n)$ *rounds to convince the verifier of* x*'s membership in* L*.*

Definition 4.2. $\mathsf{IP} = \mathsf{IP}[\mathsf{n}^{\mathsf{O}(1)}]$.

The following follows immediately along the same lines as the proof that $\mathsf{BPP} \in \mathsf{PSPACE}$: one can simply try all possible runs of the protocol and take the average.

Proposition 4.3. $\mathsf{IP} \subseteq \mathsf{PSPACE}$.

We will later prove that the reverse inclusion also holds, i.e. $\mathsf{IP} = \mathsf{PSPACE}$.

It is somewhat surprising that Arthur-Merlin games and interactive proofs are essentially equivalent in power.

Theorem 4.4 (Goldwasser, Sipser)**.** $\mathsf{IP}[\mathsf{k}] \subseteq \mathsf{AM}[\mathsf{k}+2]$

Proof Sketch. The general idea is related to the kinds of arguments in Stockmeyer's Theorem, $clBPP \in \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$, and the Valiant-Vazirani Lemma. In this Arthur's random bits will be used to choose random hash functions. The Merlin convinces Arthurs that the Prover would have convinced the Verifier if the protocol would have been run.

The general idea is that Merlin proves that the set of witnesses is large by finding pre-images for randomly chosen challenge points in the output space of the random hash function that are selected by Arthur. If the set of witnesses were too small, Merlin would be unable to do this. (Merlin knows what function the Verifier would have applied to the random bits and so can send the transcript of the conversation for any fixed hidden random string of the Verifier to prove membership in the witness set. □

4.1 Graph Non-isomorphism $\in$ IP[2]

Definition 4.5. $GRAPH\text{-}NON\text{-}ISOMORPHISM = \{[G_0, G_1] \mid G_0, G_1$ *are graphs with* $|V(G_0)| = V(G_1) = n$ *and* $\forall \sigma \in S_n,\ \sigma(G_0) \neq G_1\}$.

$GRAPH\text{-}ISOMORPHISM \in$ NP and so $GRAPH\text{-}NON\text{-}ISOMORPHISM \in$ coNP.

4.1.1 Protocol

We will now give a 2 round protocol to decide $GRAPH\text{-}NON\text{-}ISOMORPHISM$. Both prover and verifier have access to G_0 and G_1.

- ($V \to P$:)

  Verifier chooses $b \in_R \{0, 1\}$, chooses $\pi \in_R S_n$.

  Verifier sends $\pi(G_b)$ to the prover.

- ($P \to V$:) The prover, which is computationally unbounded, determines to which of the two graphs this one is supposed to be isomorphic, say $G_{b'}$ for $b' \in \{0, 1\}$. (If the input graphs are isomorphic to each other the prover can do not better than choose b' randomly.) The prover send b' to the verifer.

- ([Verification:) The Verifier accepts iff $b' = b$.

The important thing to notice is that if $[G_0, G_1]$ is not in L, that is they are isomorphic graphs, then the prover has exactly a 50-50 chance of guessing which value the verifier chose for c. If $[G_0, G_1] \in L$ then the prover should never get it wrong.

This acceptance condition yields a probability only of $1/2$. We can massage this into the correct form that we can execute two (or more) copies, either sequentially of in parallel. The verifier accepts only if the prover answers correctly for each of the graphs. The prover only fools the verifier with probability $\frac{1}{4} < 1/3$ and this can reduced to 2^{-k} for any k. Notice that it is crucial that the verifier not have access to the coin flips.

Corollary 4.6. $GRAPH\text{-}NON - $ ISOMORPHISM $\in$ AM.

5 An Interactive Proof for the Permanent: Low Degree Extensions

Theorem 5.1 (Lund-Fortnow-Karloff-Nisan)**.** *There is a polynomial length interactive proof for the predicate $PERM(A) = k$.*

Before we prove this theorem we note a number of corollaries.

Corollary 5.2. *There exist polynomial length interactive proofs for all of $\#\mathsf{P}$.*

Corollary 5.3. $\mathsf{PH} \subseteq \mathsf{IP}$

This is surprising, because a constant number of alternations is equivalent to two alternations, but an unbounded number yields PH. Later, we will prove that there exist interactive proofs for everything in PSPACE.

Proof of Corollaries. These follow from the easy observation that IP is closed under polynomial-time (Turing) reduction and by Toda's theorem. □

Remark 5.4. *In the journal version of their paper, Lund, Fortnow, Karloff, and Nisan gave an alternative direct protocol proving Corollary 5.2. This proof is given in full in the text. We present the protocol for the permanent directly both because it is interesting in its own right and because the permanent problem motivates the whole approach for these proofs.*

Proof of Theorem 5.1. We perform the computation of $PERM$ over some finite field $\mathbb{F}$, where $|\mathbb{F}| \geq n^3$. We also assume that $\{1, \ldots, n\} \subset \mathbb{F}$, although we can remove this restriction by simply choosing n distinct field elements of $\mathbb{F}$ that can substitute for $\{1, \ldots, n\}$.

Recall the definition of $PERM$:

$$PERM(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i\sigma(i)}$$

where a_{ij} is the element in the i-th row and j-th column of matrix A and S_n is the set of all permutations from $\{1, \ldots, n\}$ to itself.

Note that $PERM$ is a multivariate polynomial in the inputs, with total degree n and degree at most 1 in each variable a_{ij}. Furthermore, we can define $PERM$ recursively via the following self-reduction:

Definition 5.5. *Let $A(i|j)$ be the $(n-1) \times (n-1)$ matrix equal to A with its i-th row and j-th column removed.*

By directly expanding the permanent just as one does for the determinant (without the extra bother of maintaining the signs of the terms), one obtains:

Fact 5.6. $PERM(A) = \sum_{\ell=1}^{n} a_{1\ell} PERM(A(1|\ell))$.

Essentially the term corresponds to fixing $\sigma(1) = \ell$.

To prove that PERM$(A) = k$, it suffices to prove the values of $PERM(A(1|\ell))$ for $\ell = 1, \ldots n$. Of course, a fully recursive algorithm would yield $n!$ subproblems, saving us nothing over direct computation. Instead, the prover will give *one* proof that gives allows us to recursively prove values for all $PERM(A(1|\ell))$ via a single proof rather than n separate proofs.

To do this we will use a representation of these values of the permanent as polynomials.

Basic Idea Write $B(\ell) = A(1|\ell)$. Then

$$B(\ell) = \begin{bmatrix} b_{1,1}(\ell) & \cdots & b_{1,n-1}(\ell) \\ \vdots & \ddots & \vdots \\ b_{n-1,1}(\ell) & \cdots & b_{n-1,n-1}(\ell) \end{bmatrix}$$

Each $b_{i,j}(\ell)$ is a function: $\{1, \ldots, n\} \to \mathbb{F}$. By interpolation there are degree $n-1$ polynomials $p_{i,j}(z)$ such that $p_{i,j}(\ell) = b_{i,j}(\ell)$ for $\ell = 1, \ldots n$. Write $B(z)$ for this matrix of polynomials. Then $B(\ell) = A(1|\ell)$ for $\ell = 1, \ldots, n$. The key to the interactive proof is that $B(z)$ is also defined for many values of z that are not in $\{1, \ldots, n\}$ and we can use the simple univariate case of the Schwartz-Zippel Lemma over a large enough set (the field $\mathbb{F}$ itself).

Given this matrix of polynomials $B(z)$,

$$PERM(A) = \sum_{\ell=1}^{n} a_{1\ell} PERM(B(\ell)).$$

Observe that:

1. $PERM(B(z))$ is a degree $(n-1)n$ univariate polynomial over $\mathbb{F}$.
2. Given A, both players can compute what $B(z)$ is.

Interactive protocol to prove that $PERM(A) = k$

- Prover computes $f(z) = PERM(B(z))$ and sends the $n(n-1)+1$ coefficients of f to the verifier.

- Verifier checks that $\sum_{\ell=1}^{n} a_{1\ell} f(\ell) = k$. If good, chooses $r_1 \in_R \mathbb{F}$ and sends it to the prover.
- Prover continues with a proof that $PERM(B(r_1)) = f(r_1)$.

The proof continues until matrix size is one. In that case the Verifier computes the permanent directly by checking that the single entry of the matrix is equal to the claimed value for the permanent.

Note that is very important in this protocol r_i could take on a value larger than n. In other words, $B(r_i)$ might not be a submatrix of A at all.

Clearly, if $PERM(A) = k$ then the prover can always convince the verifier.

Suppose that $PERM(A) \neq k$. At each round there is an $i \times i$ matrix A_i and an associated claimed value k_i for the permanent of A_i where $A_n = A$ and $k_n = k$. The Prover can cause the Verifier to accept only if $PERM(A_1) = k_1$. Therefore in this case there is some round i such that $PERM(A_i) \neq k_i$ but $PERM(A_{i-1}) = k_{i-1}$. Let $B(z)$ be the $(i-1) \times (i-1)$ polynomial matrix associated with A_i and $f(z)$ be the degree $i(i-1)$ polynomial sent by the Prover in this round. Either $f(z) = PERM(B(z))$ as polynomials or not.
If $f(z) = PERM(B(z))$, then $\sum_{\ell=1}^{n} a_{1\ell} f(\ell) = \sum_{\ell=1}^{n} a_{1\ell} PERM(B(\ell)) \neq k$, and the verifier will reject the proof immediately.
If $f(z) \neq PERM(B(z))$, then $f(z) - PERMB(z)) \neq 0$ and therefore $Pr_{r \in_R \mathbb{F}}[f(r) = PERM(B(r))] \leq i(i-1)/|\mathbb{F}|$. In other words, the probability that the prover can "fool" the verifier in this round is at most $i(i-1)/|\mathbb{F}|$. Therefore, the total probability that the Prover succeeds in convincing the Verifier of an incorrect value is at most $\sum_{i=2}^{n} i(i-1)/\mathbb{F} < n^3/(3|\mathbb{F}|) \leq 1/3$. for $|\mathbb{F}| \geq n^3$. □

The above proof shows that there are interactive proofs for coNP. A constant number of rounds is unlikely unless the polynomial-time hierarchy collapses.

The key idea of the above proof was *low degree polynomial extension*. The matrix of polynomials $B(z)$ had each term of degree at most $(n-1)^2$ and so we could test propereties of it for field sizes that are not much larger than the degree which is still polynomial in size.

6 IP equals PSPACE

In this section we prove the following characterization theorem for IP.

Theorem 6.1 (Shamir, Shen)**.** IP = PSPACE

(Following Shen.) We will prove the hard direction, namely that PSPACE ⊆ IP; the other direction is almost immediate as noted before.

The key idea of this proof will also involve low degree polynomial extensions but we begin with a problem that is Boolean rather than algebraic so we need also to apply *arithmetization*.

We construct an IP protocol for TQBF using low-degree polynomial extensions over a small finite field $\mathbb{F}$. Specifically, we can choose a small field $\mathbb{F}$ with $n^3m \leq |\mathbb{F}| \leq 2n^3m$, where m is the number of 3-CNF clauses and n is the number of variables in the TQBF formula $\Psi = \exists x_1 \forall x_2 \cdots Q_n x_n \varphi(x_1, \ldots, x_n)$ where φ is a 3-CNF formula with m clauses.

6.1 Arithmetization of Boolean formulas

Create multivariate polynomial extensionds for Boolean formulas as follows:

$$\begin{aligned} f \wedge g &\mapsto P_f \cdot P_g \\ x_i &\mapsto x_i \\ \neg f &\mapsto (1 - P_f) \\ (f \vee g) = \neg(\neg f \wedge \neg g) &\mapsto 1 - (1 - P_f)(1 - P_g). \end{aligned}$$

We use the notation $P_f \circledast P_g$ as a shorthand for $1 - (1 - p_f)(1 - p_g)$. Applying these operations $P_\varphi(x_1, \ldots, x_n)$ is of degree $\leq m$ in each variable, with a total degree of $\leq 3m$.

Continuing this in the obvious way we interpret $\forall$ and $\exists$ in terms of $\wedge$ and $\vee$ and so obtain that

$$P_{\forall x_n f}(x_1, \ldots, x_{n-1}) = P_f(x_1, \ldots, x_{n-1}, 0) \cdot P_f(x_1, \ldots, x_{n-1}, 1)$$

and

$$P_{\exists x_n f}(x_1, \ldots, x_{n-1}) = P_f(x_1, \ldots, x_{n-1}, 0) \circledast P_f(x_1, \ldots, x_{n-1}, 1).$$

We want to know if $P_\Psi() = 1$. The obvious analog to our proof for $PERM$ has a problem: the degree of the polynomial doubles at each quantification step and thus the univariate polynomials we will create will have exponential degree. The solution rests on the fact that our polynomial need only be correct on inputs over $\{0, 1\}$, which yields only two points per variable. Thus a polynomial of linear degree in each variable will suffice. For this purpose we introduce a new degree reduction operation L_{x_i}.

Definition 6.2.

$$\begin{aligned} P_{L_{x_i} f}(x_1, \ldots, x_n) = &x_i \cdot P_f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \\ &+ (1 - x_i) \cdot P_f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \end{aligned}$$

We now replace Ψ by the formal sequence

$$\Psi_0 = \exists x_1 L_{x_1} \forall x_2 L_{x_1} L_{x_2} \exists x_3 L_{x_1} L_{x_2} L_{x_3} \cdots Q_n x_n L_{x_1} L_{x_2} \cdots L_{x_n} \varphi(x_1, \ldots, x_n).$$

While the $\exists$ and $\forall$ operators increase the polynomial degree (squaring it), the linearization operations L_{x_i} bring it back down to at most one in each variable. Note that the total number of quantifiers is only $n + n(n-1)/2$.

We discuss the interactive protocol for PSPACE based on this conversion in Friday's class.