6.841/18.405J: Advanced Complexity Theory April 1, 2002

Lecture 12

Lecturer: Madhu Sudan *Scribe: Grant Wang*

In this lecture, we introduce the concept of interaction in computation, discuss the motivation, and enumerate a few results.

1 Motivation for interaction

The motivation for interaction came from the field of cryptography. Consider the following scenario:

Suppose a user X wishes to log into a computer C remotely, where he has stored private data. Here, the user wishes to convince C that he is indeed X, and C wishes to verify that the user is X. The computer and the user talk back and forth until C is either sure the user is X or not. In such a situation, the user and the computer desire the following property: even if an eavesdropper E has overhead the conversation between X and C, E cannot later succeed in logging on as user X.

Note that typical password security mechanisms fail to uphold such a property, e.g. `telnet`.

We can view such an interaction as a *proof of knowledge*: User X wants to *prove* her identity to C, yet does not want to reveal any other knowledge about herself. The challenge/response scheme that is the conversation between X and C is an *interactive proof.*

2 Definition of Interaction

More concretely, an interactive proof consists of two parties, a prover P and a verifier V, and a string w. The verifier is trying to verify some fact of w, and the prover P is attempting to prove this to the verifier about w. The verifier V is allowed to flip random coins that only he can see, which we model here by a random string R. Initially, he computes a function of both w, R denoted by $q_1 = q_1(w, R)$, which he sends to the prover P. The prover receives q_1, and computes its own function of w and q_1, denoted by $a_1 = a_1(w, q_1)$. This continues for k rounds, until V receives a_k from the prover P. V then computes a verdict function of the common knowledge w, random string R, and the answers $a_1 \ldots a_k$ from the prover, i.e. V computes $verdict(w, R, a_1 \ldots a_k) = 0/1$. If the verifier is convinced, the verdict will be 1, and if the verifier is not convinced, the verdict will be 0.

Formally, a verifier is a collection of functions $(q_1(\cdot), \ldots q_k(\cdots))$, with a verdict function $verdict(\cdot)$, where each of the functions can be computed in probabilistic polynomial time, and a prover is a collection of functions $(a_1(\cdot) \ldots a_k(\cdot))$, which are each computationally unbounded. We can think of the language accepted by a fixed verifier V. We say that V accepts a language L with completeness c and soundness s, if:

$$
\begin{aligned}
w \in L &\Rightarrow \exists P \text{ s.t. } \Pr_R[verdict(P \leftrightarrow V) = 1] \geq c \\
w \notin L &\Rightarrow \forall P \text{ s.t. } \Pr_R[verdict(P \leftrightarrow V) = 1] \leq s
\end{aligned}
$$

3 Interactive Proofs for Graph Non-isomorphism

Let $\pi : \{1 \ldots n\} : \{1 \ldots n\}$ be a permutation relabeling the vertices of a graph G. We write such a relabeling as $\pi(G)$. We say that a graph G is isomorphic to H if there exists a permutation π such that $\pi(G) = H$, i.e. G and H are really the same graph under a different labeling.

Clearly, it is easy to prove that a graph G is isomorphic to a graph H: I can give you the permutation π, and you can check that $\pi(G) = H$. But how can I prove to you that G and H are non-isomorphic? There is a clever interactive proof for graph non-isomorphism.

Given graphs G and H, the verifier picks a random permutation π, and chooses F to be G or H randomly. Then, he applies $\pi(F)$ and sends this to the prover. The prover wants to show the verifier that G and H are non-isomorphic. The prover sends back $F' \in \{G, H\}$, which is his guess as to which graph F the verifier picked. The verifier accepts if $F = F'$.

Note that the completeness is 1. If the graphs are non-isomorphic, the prover that correctly identifies F to be isomorphic to either G or H will force the verifier to always accept. Note that the prover is computationally unbounded, so he can achieve this by trying all permutations π and applying them to F and determining if $\pi(F) = G$ or $\pi(F) = H$.

The soundness is $\frac{1}{2}$. If the graphs are isomorphic, the prover cannot identify F to be G or H since $F = \pi(G) = \pi'(H)$. The best he can do is guess G or H at random, which will be correct $\frac{1}{2}$ of the time.

4 Resources for Interactive Proofs

What are the resources that affect the computational power of interactive proofs? We have already stated that V consists of probabilistic, polynomial-time computable functions, and that P is computationally unbounded. We also have other resources:

- Number of rounds of interaction: 1, constant, polynomially many rounds?
- One-sided error vs. two-sided error
- Secrecy: private coins vs. public coins

Of these three resources, secrecy seems to be essential. Consider the interactive proof for graph nonisomorphsim above. If the verifier allowed his random choice of G, H to be known to the public, the prover could always answer correctly. Later, we will see the surprising result that public coins are equivalent to private coins.

5 Interactive Proofs and Arthur-Merlin games

The notion of interaction was developed independently by Goldwasser, Micali, and Rackoff (here it was called interactive proofs) and Babai (here known as Arthur-Merlin games). Whereas the motivation for Goldwasser, Micali, and Rackoff was cryptographic, the motivation for Babai was complexity-theoretic.

The key difference between Arthur-Merlin games and interactive proofs (as defined by Goldwasser, Micali and Rackoff) is public coins vs. private coins. In Arthur-Merlin games, all the coins that the verifier (Arthur) flips randomly are known to the prover (Merlin).

For interactive proofs, we write IP$[k(\cdot)]$ to be the class of languages whose verifier has $k(n)$ rounds of interaction with completeness $\frac{2}{3}$ and soundness $\frac{1}{3}$. Definitionally, IP=IP[poly$(\cdot)$].

For Arthur-Merlin games, we write AM to be the class of languages decided when Arthur flips some random coins, Merlin computes an answer a, and Arthur chooses to accept or reject deterministically. We write AM$[k(\cdot)]$ to be $k(n)$ rounds where Arthur flips random coins, and Merlin computes an answer a dependent on those random coin flips. We write MA to stand for the class of languages accepted when Merlin first speaks, and then Arthur decides to accept or reject when he can flip coins. Similarly define AMAMA, and AM.

6 Relations between complexity classes

It turns out that in interactive proofs (and Arthur-Merlin games), one round of interaction is equivalent in power to any constant number of rounds. We believe that poly(n) number of rounds is greater in power, because if poly(n) rounds was equivalent to a constant number of rounds, the polynomial hierarchy collapses.

The surprising result, due to Goldwasser and Sipser, is that private coins are equivalent to public coins, i.e. AM[$k(\cdot)$] = IP[$k(\cdot)$]. Definitionally, however, we refer to AM as interactive proofs with public coins in one-round, and IP as interactive proofs with private coins in polynomially many rounds.

7 Results for Arthur-Merlin games

Lemma 1 *AM[k] = AM, for all constants k.*

Proof Idea The proof idea behind this is one we've already seen. We can think of Arthur-Merlin games in terms of operators on complexity classes, as we did in the proof of Toda's theorem. That is, $AM = BP \cdot \exists \cdot P$, and $AMAM = BP \cdot \exists \cdot BP \cdot \exists \cdot P$. If we can show that $\exists \cdot BP \cdot \exists \cdot P \subseteq BP \cdot \exists \cdot \exists \cdot P$, we will have that AMAM $\subseteq$ AM, and proceeding by induction, we can prove the claim. ■

Lemma 2 *AM[$k(n)$] = AM[$\frac{k(n)}{c}$], for all constants c*

This is analogous to the speedup theorem for Turing machines: we can always decrease the number of rounds by a constant factor without losing any power.

Lemma 3 *AM with 2-sided error = AM with 1-sided error*

8 IP vs. AM

We have the following results relating IP and AM:

Theorem 4 (Goldwasser-Sipser) *AM[$k(\cdot)$] = IP[$k(\cdot)$], for all $k(\cdot)$*

This is the surprising result that public coins are equivalent to private coins.

Theorem 5 *IP $\subseteq$ PSPACE*

Theorem 6 *IP = $\overline{IP}$*

Theorem 7 *PSPACE $\subseteq$ IP*

Note that by IP = PSPACE, we get that IP = $\overline{\text{IP}}$, since we know PSPACE is closed under complement. It is not known how to prove that IP is closed under complement without IP=PSPACE. The result that PSPACE $\subseteq$ IP was obtained in the 90's in two papers, one by Lund, Fortnow, and Nisan and the other by Shamir. As a consequence of IP=PSPACE, Condon, Feigenbaum, Lund, and Shor show that playing (in a solitaire manner) Mah-Jong (where tiles are stacked up randomly, and one can win by removing all tiles through the single action of removing matching pairs of tiles) is equivalent in difficulty to playing interactively the optimal GO player.

We will see these results in the next few lectures.