

Interactive Computation

Slides based on *S.Aurora, B.Barak. Complexity Theory: A Modern Approach.*

Ahto Buldas

Ahto.Buldas@ut.ee

“What is intuitively required from a theorem-proving procedure? First, that it is possible to prove a true theorem. Second, that it is impossible to prove a false theorem. Third, that communicating the proof should be efficient, in the following sense. It does not matter how long must the prover compute during the proving process, but it is essential that the computation required from the verifier is easy.”

Goldwasser, Micali, Rackoff 1985.

Motivation

The standard notion of a mathematical proof follows the *certificate* definition of NP. That is, to prove that a statement is true one provides a sequence of symbols that can be written down in a book or on paper, and a valid sequence exists only for true statements.

However, people often use more general ways to convince one another of the validity of statements: they *interact* with one another, with the person verifying the proof (henceforth the *verifier*) asking the person providing it (henceforth the *prover*) for a series of explanations before he is convinced.

It seems natural to try to understand the power of such interactive proofs. For example, can one prove that a given formula is not satisfiable?

The surprising answer is yes!! Indeed, interactive proofs turned out to have unexpected powers and applications.

Interactive proofs with deterministic verifier I

What happens when we introduce interaction into the NP scenario. The prover and verifier *interact* with the verifier asking questions and the prover is responding, where at the end the verifier decides whether or not to accept the input.

Both verifier and prover can keep state during the interaction, or equivalently, the message a party sends at any point in the interaction can be a function of all messages sent and received so far. Formally:

Interactive proofs with deterministic verifier II

Def. (Interaction of deterministic functions): Let $f, g: \{0, 1\}^* \rightarrow \{0, 1\}^*$ be functions. A k -round interaction of f and g on input $x \in \{0, 1\}^*$, denoted by $\langle f, g \rangle(x)$ is the sequence of strings $a_1, \dots, a_k \in \{0, 1\}^*$ define by:

$$\begin{aligned} a_1 &= f(x), \\ a_2 &= g(x, a_1), \\ &\dots \\ a_{2i+1} &= f(x, a_1, \dots, a_{2i}) \\ a_{2i+2} &= g(x, a_1, \dots, a_{2i+1}) . \end{aligned}$$

The output of f (resp. g) at the end of the interaction denoted $\text{out}_f \langle f, g \rangle(x)$ (resp. $\text{out}_g \langle f, g \rangle(x)$) is defined as $f(x, a_1, \dots, a_k)$ (resp. $g(x, a_1, \dots, a_k)$).

Deterministic proof systems

Def.: We say that a language L has a k -round deterministic interactive proof system if there's a deterministic TM V that on input x, a_1, \dots, a_i runs in time polynomial in $|x|$, satisfying:

(Completeness) $x \in L \Rightarrow \exists P : \{0, 1\}^* \rightarrow \{0, 1\}^* : \text{out}_V \langle V, P \rangle(x) = 1$

(Soundness) $x \notin L \Rightarrow \forall P : \{0, 1\}^* \rightarrow \{0, 1\}^* : \text{out}_V \langle V, P \rangle(x) = 0$

The class **dIP** contains all languages with a $k(n)$ -round deterministic interactive proof systems with $k(n)$ polynomial in n .

Limits of deterministic verifiers

It turns out this actually does not change the class of languages we can prove:

Theorem: $\text{dIP} = \text{NP}$.

Proof: Clearly, every NP language has a 1-round proof system. Now we prove that if a L has an interactive proof system of this type then $L \in \text{NP}$. The certificate for membership is just the transcript (a_1, a_2, \dots, a_k) causing the verifier to accept. To verify this transcript, check that indeed $V(x) = a_1$, $V(x, a_1, a_2) = a_3$, ... , and $V(x, a_1, \dots, a_k) = 1$. If $x \in L$ then there indeed exists such a transcript. If there exists such a transcript (a_1, \dots, a_k) then we can define a prover function P to satisfy $P(x, a_1) = a_2$, $P(x, a_1, a_2, a_3) = a_4$, etc. We see that $\text{out}_V \langle V, P \rangle(x) = 1$ and hence $x \in L$.

The class IP

We need to let the verifier be probabilistic. Also, the verifier will be allowed to come to a wrong conclusion (e.g., accept a proof for a wrong statement) with some small probability. However, as in the case of probabilistic algorithms, this probability is over the verifier's coins and the verifier will reject proofs for a wrong statement with good probability regardless of the strategy the prover uses.

It turns out that the combination of interaction and randomization has a huge effect: as we will see, the set of languages which have interactive proof systems now jumps from NP to $PSPACE$.

Example: Distinguishing Pepsi and Coke

Consider the following scenario: Marla claims to Arthur that she can distinguish between the taste of Coke and Pepsi.

To verify this statement, Marla and Arthur repeat the following experiment 50 times:

Marla turns her back to Arthur, as he places Coke in one unmarked cup and Pepsi in another, choosing randomly whether Coke will be in the cup on the left or on the right. Then Marla tastes both cups and states which one contained which drinks.

While, regardless of her tasting abilities, Marla can answer correctly with probability $\frac{1}{2}$ by a random guess, if she manages to answer correctly for all the 50 repetitions, Arthur can indeed be convinced that she can tell apart Pepsi and Coke.

Formal definition of IP

To model an interaction between f and g where f is probabilistic, we add an additional m -bit input r to the function f in (1), that is having $a_1 = f(x, r)$, $a_3 = f(x, r, a_1, a_2)$, etc. The interaction $\langle f, g \rangle(x)$ is now a random variable over $r \in \{0, 1\}^m$. Similarly the output $\text{out}_f \langle f, g \rangle(x)$ is also a random variable.

Def. (IP): Let $k: \mathbb{N} \rightarrow \mathbb{N}$ be some function with $k(n)$ computable in $\text{poly}(n)$ time. A language L is in $\mathbf{IP}[k]$ if there is a Turing machine V such that on inputs x, r, a_1, \dots, a_i , V runs in time polynomial in $|x|$ and such that:

$$\begin{aligned} \text{(Completeness)} \quad x \in L &\Rightarrow \exists P : \Pr[\text{out}_V \langle V, P \rangle(x) = 1] \geq \frac{3}{4} \\ \text{(Soundness)} \quad x \notin L &\Rightarrow \forall P : \Pr[\text{out}_V \langle V, P \rangle(x) = 0] \leq \frac{1}{4} . \end{aligned}$$

We define $\mathbf{IP} = \cup_{c>0} \mathbf{IP}[n^c]$.

Some properties of IP

Allowing the prover to be probabilistic (i.e., the answer function a_i depends upon some random string used by the prover) does not change the class **IP**. The reason is that for any language L , if a probabilistic prover P results in making verifier V accept with some probability, then averaging implies there is a deterministic prover which makes V accept with the same probability.

Since the prover can use an arbitrary function, it can in principle use unbounded computational power (or even compute undecidable functions). However, one can show that given any verifier V , we can compute the optimum prover (which, given x , maximizes the verifiers acceptance probability) using $\text{poly}(|x|)$ space (and hence $2^{\text{poly}(|x|)}$ time). Thus **IP** \subseteq **PSPACE**.

The probabilities of correctly classifying an input can be made arbitrarily close to 1 by using the same boosting technique we used for **BPP**: to replace $\frac{3}{4}$ by $1 - \exp(-m)$, sequentially repeat the protocol m times and take the majority answer. In fact, using a more complicated proof, it can be shown that we can decrease the probability without increasing the number of rounds using parallel repetition (i.e., the prover and verifier will run m executions of the protocol in parallel). We note that the proof is easier for the case of public coin proofs, which will be defined below.

Replacing the constant $\frac{3}{4}$ in the completeness requirement by 1 does not change **IP**. This is a nontrivial fact.

In contrast replacing the constant $\frac{3}{4}$ by 1 in the soundness condition is equivalent to having a deterministic verifier and hence reduces the class **IP** to **NP**.

We emphasize that the prover functions do not depend upon the verifiers random strings, but only on the messages/questions the verifier sends. In other words, the verifiers random string is private. Often these are called *private coin interactive proofs*. Later we will also consider the model where all the verifiers questions are simply obtained by tossing coins and revealing them to the prover, this is known as *public coins* or *Arthur-Merlin proofs*.

Proving that graphs are not isomorphic.

Well now see an example of a language in **IP** that is not known to be in **NP**. Recall that the usual ways of representing graphs adjacency lists, adjacency matrices involve a numbering of the vertices. We say two graphs G_1 and G_2 are *isomorphic* if they are the same up to a renumbering of vertices. In other words, if there is a permutation π of the labels of the nodes of G_1 such that $\pi(G_1) = G_2$.

If G_1 and G_2 are isomorphic, we write $G_1 \cong G_2$. The GI problem is the following: given two graphs G_1, G_2 (say in adjacency matrix representation) decide if they are isomorphic. Note that clearly $GI \in \mathbf{NP}$, since a certificate is simply the description of the permutation π .

The graph isomorphism problem is important in a variety of fields and has a rich history. Along with the factoring problem, it is the most famous **NP**-problem that is not known to be either in **P** or **NP**-complete. The results

of this section show that GI is unlikely to be **NP**-complete, unless the polynomial hierarchy collapses. This will follow from the existence of the following proof system for the complement of GI: the problem GNI of deciding whether two given graphs are not isomorphic.

Protocol: Private-coin Graph Non-isomorphism:

- V : pick $i \leftarrow \{1, 2\}$ uniformly randomly. Randomly permute the vertices of G_i to get a new graph H . Send H to P .
- P : identify which of G_1, G_2 was used to produce H . Let G_j be that graph. Send j to V .
- V : accept if $i = j$; reject otherwise.

To see that the IP definition is satisfied by the above protocol, note that if $G_1 \not\cong G_2$ then there exists a prover such that $\Pr[V \text{ accepts}] = 1$, because

if the graphs are non-isomorphic, an all-powerful prover can certainly tell which one of the two is isomorphic to H .

On the other hand, if $G_1 \cong G_2$ the best any prover can do is to randomly guess, because a random permutation of G_1 looks exactly like a random permutation of G_2 . Thus in this case for every prover, $\Pr[V \text{ accepts}] \leq \frac{1}{2}$. This probability can be reduced to $\frac{1}{4}$ by sequential or parallel repetition

Public coins and AM

Allowing the prover full access to the verifier's random string leads to the model of interactive proofs with public-coins.

Def. (AM, MA): For every k we denote by $AM[k]$ the class of languages that can be decided by a k -round interactive proof in which each verifier's message consists of sending a random string of polynomial length, and these messages comprise of all the coins tossed by the verifier. A proof of this form is called a public coin proof (it is sometimes also known as an *Arthur Merlin proof*)

We define by AM the class $AM[2]$. That is, AM is the class of languages with an interactive proof that consists of the verifier sending a random string, the prover responding with a message, and where the decision to accept

is obtained by applying a deterministic poly-time function to the transcript. The class \mathbf{MA} denotes the class of languages with 2-round public coins interactive proof with the prover sending the first message. That is, $L \in \mathbf{MA}$ if there's a proof system for L that consists of the prover first sending a message, and then the verifier tossing coins and applying a poly-time predicate to the input, the provers message and the coins.

AM and IP

Note that clearly for every k , $\mathbf{AM}[k] \subseteq \mathbf{IP}[k]$. The interactive proof for GNI seemed to crucially depend upon the fact that P cannot see the random bits of V . If P knew those bits, P would know i and so could trivially always guess correctly. Thus it may seem that allowing the verifier to keep its coins private adds significant power to interactive proofs, and so the following result should be quite surprising:

Theorem: For every $k: \mathbb{N} \rightarrow \mathbb{N}$ with $k(n)$ computable in $\text{poly}(n)$,

$$\mathbf{IP}[k] \subseteq \mathbf{AM}[k + 2] .$$

The central idea of the proof of Theorem 8.8 can be gleaned from the proof for the special case of GNI:

Theorem: $\text{GNI} \in \mathbf{AM}[k]$ for some constant $k \geq 2$.

