CS 6743 **Lecture 18** [1] **Fall 2007**

1 Interactive proof systems and graph non-isomorphism

The *interactive protocols (IP)* are also protocols between a probabilistic polytime Verifier and an all-powerful Prover, where after a certain number of rounds of communication, the Verifier accepts or rejects.

In interactive protocols, the Verifier moves first. An IP protocol with k rounds of communication is the protocol where at most k messages are exchanged in a conversation between Verifier and Prover.

We say that a language $L \in \mathsf{IP}[k]$ if there is a probabilistic polytime verifier such that, for every $x \in L$, there is a Prover that convinces the Verifier to accept with probability $\geq 3/4$ after at most k-rounds; and for every $x \notin L$, each Prover can convince the Verifier to accept with probability at most $1/4$ after k rounds.

Here's an example of an IP protocol for the Graph Nonisomorphism Problem (NISO). Define $NISO = \{(G_1, G_2) \mid G_1 \text{ and } G_2 \text{ are not isomorphic}\}$.

Theorem 1. *$NISO \in \mathsf{IP}[2]$*

Proof. Here's a protocol. Given an input (G_1, G_2) of two graphs on n vertices, the Verifier will randomly pick $i \in \{1, 2\}$, and a random permutation π of the set $\{1, 2, \ldots, n\}$. The verifier will send $\pi(G_i)$ to the Prover (i.e., the Verifier sends a randomly permuted copy of a randomly chosen graph in (G_1, G_2)). The Prover sends back $j \in \{1, 2\}$. The Verifier accepts iff $j = i$.

Analysis (1) If G_1 and G_2 are non-isomorphic, the computationally unbounded Prover can always find a correct $j = i$ by checking which of G_1 and G_2 is isomorphic to the graph received from the Verifier. So, in this case, the Verifier can be made to accept with probability 1.

(2) If G_1 and G_2 are isomorphic, then a graph sent to the Prover by the Verifier in case $i = 1$ is from the same distribution as the graph sent in the case $i = 2$. Hence, the Prover has no way of determining i, and his j will be equal to i with probability $1/2$. (This can be shown formally after some simple probability calculations; it's left as an exercise.)

So, if the graphs are non-isomorphic, the Verifier accepts with probability 1. If the graph are isomorphic, the Verifier accepts with probability at most $1/2$. (It is possible to reduce the error probability to $1/4$.) $\square$

2 IP=PSPACE

The less interesting part of the proof is to show that $IP \subseteq PSPACE$. For that, we show how to simulate the interaction in the IP protocol in PSPACE.

[1] Parts of this lecture are a modification of notes by Valentine Kabanets

Suppose that in the string of messages all odd-numbered messages starting from the first message m_1 belong to the Verifier and all even-numbered messges to the Prover. The PSPACE algorithm tries to reconstruct the message string recursively starting from the last message. Intuitively, it tries to calculate the moves made by the Prover which has the highest probability of making Verifier accept the given string. The depth of the recursion is the number of messages, so it is possible to achieve this in polynomial space. Please see Sipser's book for more details.

3 Counting complexity classes

Before we proceed proving $PSPACE \subseteq IP$, we will give a variant of this proof, showing that a smaller class $\#P$ is in IP. $\#P$ is one of the family of *counting* complexity classes, where we are concerned with the *number* of accepting/rejecting paths of a non-deterministic Turing machine, rather than their mere existence. In contrast with complexity classes that we have seen before, this is a functional class: that is, the answer is the number rather than a boolean value. You can check that this value can be computed using polynomial space. A "boolean version" of $\#P$ could be considered $P^{\#P}$, which contains polynomial-time hierarchy (this result is called Toda's theorem).

One problem known to be complete for $\#P$ is the Permanent of a matrix (which is the same as determinant, but with no signs on permutations). This is a surprising fact, since determinant itself can be computed in a (believed to be) much smaller complexity class NC^2. For the proof in the next lecture, we will use another natural $\#P$ problem called $\#SAT$, in which we are required to compute the number of satisfying assignments of a given formula.