1 Recap

At the end of last week’s lecture, we were in the middle of proving:

**Theorem 1** \[ \text{GNI} \subseteq \text{AM}[2] \]

We defined \( \text{GNI} = \{(G_1, G_2) : G_1 \not\sim G_2\} \), the set of all pairs of non-isomorphic graphs. To prove the theorem, it suffices to give a two-round public-coin interactive proof protocol for GNI. Earlier in the lecture, we saw a simple private-coin protocol that demonstrates \( \text{GNI} \subseteq \text{IM}[2] \).

For the public-coin protocol, we observed that it would suffice for the prover to demonstrate to the verifier that the following set is large:

\[ S = \{(H, \pi) : H \text{ is isomorphic to } G_1 \text{ or } G_2 \text{ and } \pi \text{ is not an automorphism of } H\} \]

Given a graph \( G = ([n], E) \) and a permutation \( \pi : [n] \to [n] \), we defined \( \pi(G) = ([n], E') \), where each edge \((i, j)\) is in \( E' \) if and only if \((\pi(i), \pi(j))\) is in \( E \).

**Definition 2** Given graphs \( G \) and \( H \) on \( n \) vertices, a permutation \( \pi : [n] \to [n] \) is an **isomorphism** between \( G \) and \( H \) if \( \pi(G) = H \). If such a map exists, then \( G \) and \( H \) are **isomorphic**, and we write \( G \cong H \). If \( \pi(G) = G \), then \( \pi \) is an **automorphism** of \( G \).

With these definitions in mind, we observed that the size of \( S \) is \( n! \) if \( G_1 \cong G_2 \) and \( 2n! \) if \( G_1 \not\sim G_2 \). We asked how the prover could convince the verifier that \( S \) is large.

2 Set lower bound protocol

Putting aside the specifics of the GNI problem for now, we consider a general protocol for interactive proofs that sets are large.

Suppose we are given a set \( S \subseteq \{0,1\}^m \) such that membership in \( S \) is easy to certify. This means that if \( x \in S \), then the prover \( P \) can convince the verifier \( V \) to accept, and if \( x \not\in S \), then the \( V \) will reject no matter what the \( P \) does.

**Question:** Can the \( P \) convince \( V \) of the approximate size of \( S \)?

In particular supposing that the prover claims \( |S| = k \), we would like a protocol with the following property:

**Goal:** If \( |S| \geq k \), then the verifier will accept with high probability. If \( |S| \leq \frac{k}{2} \), then the verifier will reject w.h.p., regardless of what the prover does.

If such a protocol exists, then it completes our proof that \( \text{GNI} \subseteq \text{AM}[2] \).

First consider two cases where this goal is not difficult to achieve:

**Simple case 1:** Suppose that \( k = 2^m \). Then the verifier simply samples \( x \) uniformly at random from \( \{0,1\}^m \) and asks the prover to prove that \( x \in S \). If \( |S| = 2^m \), then the prover can always succeed.
If $|S| \leq 2^{m-1}$, then with probability at least $\frac{1}{2}$, $x \notin S$ and the prover fails.

**Simple case 2:** Suppose that $k \geq 2^m/10$. Then $P$’s claim is that a $k/2^m$ fraction of all points are in $S$. Omitting the details, $V$ can check this claim by sending many points $x_1, ..., x_r \in_U \{0,1\}^m$, asking $P$ to prove membership in $S$ for as many of the $x_i$ as it can.

But what happens when $k$ is exponentially smaller than $2^m$? $V$ cannot send exponentially many query points to $P$, so the above method won’t work. We need a new approach.

### 2.1 Main tool: Pairwise independent hash functions

**Definition 3** Let $\mathcal{H}_{n,k}$ be a collection of functions from $\{0,1\}^n$ to $\{0,1\}^k$. $\mathcal{H}_{n,k}$ is a **pairwise independent** family if for all $x, x' \in \{0,1\}^n$ such that $x \neq x'$, and for all $y, y' \in \{0,1\}^k$, we have

$$\Pr_{h \in \mathcal{H}_{n,k}}[h(x) = y \text{ and } h(x') = y'] = 2^{-2k}$$

Consider the family $\mathcal{H}_{n,n} = \{h_{a,b}\}_{a,b \in F_2^n}$, where $h_{a,b} : F_2^n \to F_2^n$ is given by $h_{a,b}(x) = ax + b$.

**Claim 4** This is a pairwise independent hash family.

The proof is left as an exercise.

**Corollary 5** This also gives a pairwise independent hash family $\mathcal{H}_{n,k}$ for $k \leq n$ by deleting the last $n - k$ bits.

### 2.2 The protocol

We are now ready to give our protocol for lower bounding the size of $S$.

**Single round of the Set lower bound protocol:**

- Choose $r$ such that $2^{r-2} \leq k \leq 2^{r-1}$.
- $V$ picks $h$ uniformly at random from pairwise independent hash family $\mathcal{H}_{m,r}$ and sends $h$ to $P$.
- $V$ samples $y \in_R \{0,1\}^r$ and asks $P$ whether $y$ is in the image of $S$ under $h$, i.e.,

$$\exists x \text{ s.t. } x \in S \text{ and } h(x) = y$$

$V$ expects $P$ to send some $x \in S$ such that $x \in h^{-1}(y)$.

Following claim says the the image of $S$ under a random hash function from pairwise independent hash family $\mathcal{H}_{m,r}$ is roughly of same size as $S$.

**Claim 6** $S \subseteq \{0,1\}^m$, such that $2^{r-1} \leq |S| \leq 2^r$. Let $h(S)$ denotes the image of $S$ under the hash function $h$. Let $p = \frac{|S|}{2^r}$. Then,

$$\Pr_{y \in \{0,1\}^r, h \in \mathcal{H}_{m,r}}[y \in h(S)] \in \left[\frac{3p}{4}, p\right]$$
Proof Upper bound is straight forward. Consider any \( y \in \{0, 1\}^r \). Let \( E_x \) be an event such that \( h(x) = y \). We have \( \Pr_{y \in \{0, 1\}^r} [E_x] = \frac{1}{2^r} \). Because of pairwise independence we also have \( \Pr_{y \in \{0, 1\}^r} [E_x \cap E_{x'}] = \frac{1}{2^{2r}} \) for \( x \neq x' \).

\[
\Pr_{y \in \{0, 1\}^r, h \in RH_{m,r}} [y \in h(S)] = \Pr_{y \in \{0, 1\}^r, h \in RH_{m,r}} [\bigcup E_x] \geq \sum_{x \in S} \Pr_{y \in \{0, 1\}^r, h \in RH_{m,r}} [E_x] - \frac{1}{2} \sum_{x, x' \neq h \in RH_{m,r}} \Pr_{y \in \{0, 1\}^r} [E_x \cap E_{x'}]
\]

\[
= \frac{|S|}{2^r} - \frac{1}{2} \left( \binom{|S|}{2} \frac{1}{2^{2r}} \right)
\]

\[
= \frac{|S|}{2^r} \left( 1 - \frac{1}{2} \cdot \frac{|S| - 1}{2} \cdot \frac{1}{2^r} \right)
\]

\[
\geq \frac{|S|}{2^r} \left( 1 - \frac{1}{4} \right)
\]

\[
= \frac{3}{4} \cdot p
\]

where the first step uses inclusion-exclusion principle.

The final protocol is, the verifier and prover run several rounds of the set lower bound protocol. At each step the verifier can check if the prover is lying or not by checking \( h(x) = y \). Finally the verifier accepts iff for at least \( 2p/3 \) fraction of the queries prover is able to send the certificate. Given the above claim, by Chernoff bound, constant number of rounds suffices to have completeness more than \( 2/3 \) and the soundness less than \( 1/3 \).

Remark The above idea can be generalized to show that any private coins protocols can be transformed into public coins protocol with a similar number of rounds.

3 \( \text{IP} = \text{PSPACE} \)

In the last section we showed that \( \text{GNI} \) has an interactive proof protocol. Given this, it is natural to ask the following question: How powerful is the class \( \text{IP} \)? A beautiful theorem of [LFKN, Shamir] states that the class \( \text{IP} \) is in fact \emph{equal} to \( \text{PSPACE} \).

Theorem 7 (LFKN, Shamir) \( \text{IP} = \text{PSPACE} \)

In this class, we will not prove the above theorem, instead we will prove somewhat weaker statement.

Theorem 8 \( \text{SAT} \in \text{IP} \) i.e. \( \text{coNP} \subseteq \text{IP} \)

Consider the following language:

Definition 9 \( \#\text{SAT} = \{ (\phi, k) | \phi \in 3\text{CNF} \text{ and } \phi \text{ has exactly } k \text{ satisfying assignments} \} \)

If \( \#\text{SAT} \) has an interactive proof the it is easy to see that \( \text{SAT} \) also has interactive proof. So, we will prove the following stronger statement.

Theorem 10 \( \#\text{SAT} \in \text{IP} \)
Consider a Boolean formula \( \phi \) in \( n \) variables. Let \( k \) be the number of satisfying assignments for \( \phi \).

One approach is to ask \textbf{prover} all the satisfying assignments. But this information could be of exponential size. We want the whole transcript to be bounded by polynomial in the input size.

Another approach is an inductive approach as follows: Compute the formula \( \phi_{x_1=0} \) and \( \phi_{x_1=1} \) by setting the variable \( x_1 = \text{False} \) and \( x_1 = \text{True} \) respectively. Note that above operation can be done efficiently. Now, ask the \textbf{prover} the number of satisfying assignment for both these formulas. Suppose the \textbf{prover} replies with the answers \( k_0 \) and \( k_1 \). The important observation to make is that each satisfying assignment to \( \phi_{x_1=0} \) or \( \phi_{x_1=1} \) gives unique satisfying assignment to \( \phi \) and also depending on the value of \( x_1 \) in the satisfying assignment to \( \phi \), the satisfying assignment without \( x_1 \) must be a satisfying assignment to either \( \phi_{x_1=0} \) or \( \phi_{x_1=1} \). And hence, we must have \( k = k_0 + k_1 \). If indeed, the formula has exactly \( k \) satisfying assignments then the \textbf{prover} would reply with the correct values of \( k_0 \) and \( k_1 \). But what if the formula does not have \( k \) satisfying assignments? Then the \textbf{prover} must lie in at least one of the two cases. The strategy the \textbf{verifier} could think of to catch lie is to randomly choose one of the two formulae \( \phi_{x_1=0} \) and \( \phi_{x_1=1} \) and ask \textbf{prover} to prove the number of satisfying assignment to the chosen formula is what he claimed before. And then the \textbf{verifier} can repeat the same procedure until the formula size is reduced to \( O(\log n) \) in which case, the \textbf{verifier} can compute the number of satisfying assignments in polynomial time. If the final answer matches with the \textbf{prover}'s answer then the \textbf{verifier} accepts otherwise he rejects. Let’s analyze how good or bad this strategy is:

- **Completeness**: If the formula has \( k \) satisfying assignment the the \textbf{prover} always replies with correct answer to every queries that the \textbf{verifier} asks and hence \textbf{verifier} accepts with probability 1 in this case.

- **Soundness**: Suppose that the formula does not have exactly \( k \) satisfying assignments. It means the \textbf{prover} has to lie in one of the two cases when the \textbf{verifier} sets a variable to True and False. In this case, the probability of catching lie at each step is \( 1/2 \). And hence the probability of catching lie till the end of the procedure is about \( 1/2^k \) which is exponentially small in \( n \).

The above approach gives rise to think of the following strategy: If the \textbf{prover} lies in step 1 then the protocol should be such that the lie is present in second step with high probability (in the previous protocol this probability is only \( 1/2 \)).

The clever idea is as follows: In Boolean formula, variables take values in \{0, 1\}. But somehow, allow variables to take value in some larger field. This gives rise to finding a way to convert a Boolean formula to some expression that involves addition and multiplication form the larger field. The conversion should be such that the for any satisfying assignment to formula \( \phi \) if we plug in the values to variables \( x_i \)’s from the assignment (where True maps to 1 and False maps to 0), then the formula should evaluate to 1. If the assignment is not a satisfying assignment then the evaluation should be zero. Define a map between a Boolean formula \( \phi(x_1, x_2, \ldots, x_n) \) to a polynomial \( \tilde{\phi}(x_1, x_2, \ldots, x_n) \) over a field as follows: Since the Boolean formula contains four different operations, it is enough to give a map for these operations:

- \( \tilde{x}_i = x_i \)
- \( \tilde{x}_i = (1 - x_i) \)
- \( x_i \land x_j = x_i x_j \)
- \( x_i \lor x_j = 1 - (1 - x_i)(1 - x_j) \)

where the arithmetic operations on right hand side are the field addition and multiplication. This process is called \textit{arithmetization}. In this way, we can convert any Boolean formula to a polynomial in the same number of variable. If \( \phi \) is a 3CNF formula having \( m \) clauses then the degree of the mapped polynomial
is at most $3m$. This low degree as compared to the field size is crucial which we will see later. Another main property of the map is the following: For all $x_i \in \{0, 1\}$,

$$\tilde{\phi}(x_1, x_2, \ldots, x_n) = \phi(x_1, x_2, \ldots, x_n)$$

Hence we can write down the expression for the number of satisfying assignments in terms of the polynomial $\phi$ as follows:

$$S = \sum_{x_1 \in \{0, 1\}} \sum_{x_2 \in \{0, 1\}} \ldots \sum_{x_n \in \{0, 1\}} \tilde{\phi}(x_1, x_2, \ldots, x_n)$$

$$= \sum_{x_1 \in \{\text{True, False}\}} \sum_{x_2 \in \{\text{True, False}\}} \ldots \sum_{x_n \in \{\text{True, False}\}} \phi(x_1, x_2, \ldots, x_n)$$

where $\phi(x_1, x_2, \ldots, x_n) \in \{0, 1\}$, depending on whether $x_i$’s is a satisfying assignment or not. In order to get the number of satisfying assignment exactly form the expression, we want the characteristic of field to be of size more than $2^n$, just because there could be $2^n$ satisfying assignments to a Boolean formula. Suppose we work on filed $F_p$ where $p > 2^n$.

The verifier’s strategy now is as follows: Keep the variable $x_1$ alive in the expression for $S$. By doing this, we get a uni-variate polynomial $\tilde{\phi}_{x_1}$ of small degree. Ask the prover for this polynomial $P_1 = \phi_{x_1}$.

$$P_1(x) = \sum_{x_2 \in \{0, 1\}} \sum_{x_3 \in \{0, 1\}} \ldots \sum_{x_n \in \{0, 1\}} \tilde{\phi}(x, x_2, \ldots, x_n)$$

First thing to note that the size of polynomial in polynomial in $n$ since it is a uni-variate polynomial of degree at most $3m$. Also, we can get the number of satisfying assignments to the formula $\phi_{x_1=0}$ and $\phi_{x_1=1}$ by just plugging $x_1 = 0$ and $x_1 = 1$ respectively in $P_1$, this evaluation can also be done in polynomial time. If the formula $\phi$ does not have exactly $k$ satisfying assignments, then the polynomial that the prover returns must be a wrong polynomial $P_1^*$. Since the degree of polynomial is at most $3m$, and $|F| >> 3m$, the polynomial differs form the correct one $P_1$ in many evaluations. If we sample a random $\alpha \in F_p$ then with very high probability $P_1^*(\alpha) \neq P_1(\alpha)$. The verifier now chooses a random point $\alpha_1$ in $F$ and asks prover to prove that $P_1^*(\alpha)$ is indeed the correct evaluation of $P_1(x)$ at point $\alpha_1$. This is the same goal as before - proof of evaluation of a polynomial. If the prover lies in the first step then he is left with proving incorrect claim. Hence, verifier can continue with the similar steps. In this case, the verifier asks prover for a polynomial $P_2(x)$ :

$$P_2(x) = \sum_{x_3 \in \{0, 1\}} \sum_{x_4 \in \{0, 1\}} \ldots \sum_{x_n \in \{0, 1\}} \tilde{\phi}(\alpha_1, x, x_2, \ldots, x_n)$$

By plugging in 0 and 1 in the polynomial ,

$$P_2(0) + P_2(1) = \sum_{x_2 \in \{0, 1\}} \sum_{x_3 \in \{0, 1\}} \ldots \sum_{x_n \in \{0, 1\}} \tilde{\phi}(\alpha_1, x_2, x_3, \ldots, x_n) = P_1(\alpha_1)$$

At each step the probability of catching lie is high. We will analyze the soundness and completeness of the protocol formally in the next lecture. To sum up, the complete protocol is as follows:
IP protocol for \#SAT:

1. Both verifier and prover have \((\phi, k)\), \(\phi\) is a 3CNF formula in \(n\) variables having \(m\) clauses.

2. Verifier asks prover a polynomial

\[
P_1(x) = \sum_{x_2 \in \{0,1\}} \sum_{x_3 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \tilde{\phi}(x_2, \ldots, x_n)
\]

3. Prover replies with some polynomial \(P^*_1(x)\).

4. If the degree of the polynomial \(P^*_1(x)\) is greater than \(3m\) or \(P^*_1(0) + P^*_1(1) \neq k\), then reject.

5. For \(i = 1\) to \(n - 1\)
   - Verifier chooses a random element \(\alpha_i\) in \(F_p\) and asks prover for the polynomial:
   \[
P_{i+1}(x) = \sum_{x_{i+2} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \tilde{\phi}(\alpha_1, \alpha_2, \ldots, \alpha_i, x, x_{i+2}, \ldots, x_n)
   \]
   - Prover replies with some polynomial \(P^*_{i+1}(x)\).
   - Verifier checks if the degree of the polynomial \(P^*_{i+1}(x)\) is at most \(3m\) and \(P^*_{i+1}(0) + P^*_{i+1}(1) = P^*_1(\alpha_i)\), if not reject.