6.842 Randomness and Computation February 19, 2014

Lecture 5

Lecturer: Ronitt Rubinfeld *Scribe: Maryam Aliakbarpour*

1 Review

In the previous lecture, we saw a sample of interactive proofs via private coins. Here is a short review about what we had.

Definition 1 *An* Interactive proof *for language L is a protocol such that*

- *if $x \in L$ and both verifier V and prover P follow the protocol, then* $\Pr[V \text{ accepts } x] \geq \frac{2}{3}$.
- *if $x \notin L$ and V follows the protocol (no matter what P does), then* $\Pr[V \text{ rejects } x] \geq \frac{2}{3}$.

IP *denotes the set of problems that have an interactive proof.*

It is easy to see that the graph isomorphism problem is in **NP** since we can check it in polynomial time by giving the corresponding permutation of vertices. Also, using this permutation as a proof, we get that that graph isomorphism is in **IP** as well.

Here, we consider the non-isomorphism problem (denoted by $\ncong$). Given two input graphs A and B, we want to answer: Is A a non-isomorphism of B? We do not know if this problem is in **NP**, even though there are some proofs for the non-isomorphism problem in special cases. For example, we can show degree inconsistency in two graphs or find a local subgraph in one graph that does not exist in the other. However, there is no way to verify the answer in general (in polynomial time).

In the previous lecture, we provided a protocol to show that the graph non-isomorphism problem is in **IP** with private coins. Although, private coin tossing was a key component of that protocol, we are going to see another protocol via public coins which shows that even if the prover has access to the verifier's random tape, the problem still has an interactive proof system [Goldwassar, Sipser]. Moreover, it is true that **IP** with public coins = **IP** with private coins. The proof of this theorem is beyond the scope of this class.

2 A Protocol for Non-Isomorphism with Public Coins

Notations: Let $[A]$ denote the set of all graphs isomorphic to A. An example of $[G]$ is shown in Figure 1. In this lecture, we only consider graphs with no "non-trivial isomorphism". i.e. any permutation of labels results in a new graph. Thus, $|[A]| = n!$ where n is the number of nodes in A.

Clearly, if $A \cong B$, then $|[A] \cup [B]| = n!$. On the other hand, if $A \ncong B$, $|[A] \cup [B]| = 2(n!)$. Let $\mathcal{U}$ denote $[A] \cup [B]$. Intuitively, if the verifier is able to somehow estimate the size of $\mathcal{U}$, then he can distinguish between the two.

Note that in our context, the prover tries to convince the verifier that $A \ncong B$. So, he wants to show that as many graphs as possible are in $\mathcal{U}$. However, he cannot prove anything about graphs that are not in $\mathcal{U}$. So whenever, the verifier asks about a graph in $\mathcal{U}$, the prover reasonably gives the desired proof. Next we present two protocols for the problem.

2.1 Naïve Approach for Sampling

As we mentioned before, the verifier wants to estimate the size of $\mathcal{U}$. In this regard, he samples a random n-node graph G and asks the prover if it is in $\mathcal{U}$. Since, if $A \ncong B$, the size of $\mathcal{U}$ is twice bigger and the verifier can pass after seeing a big enough number of successes. We describe the protocol in Figure 2.

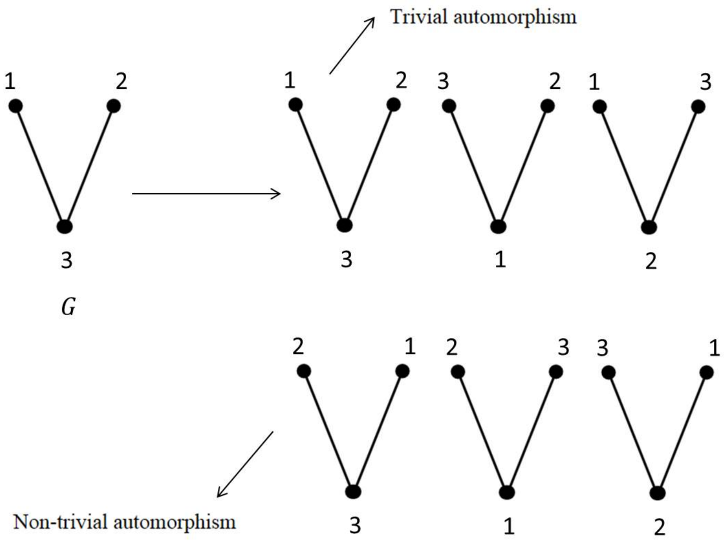

Figure 1: Graph G and all its isomorphisms

```
Naïve Protocol:
  repeat ? times:
    $V \to P$: verifier gives a random $n$-node graph $G$ to the prover.
    $P \to V$: If $G \in \mathcal{U}$, prover send a proof to show that $G \cong A$ or $G \cong B$.
               Else, nothing.
  If successes big enough number of iteration, $V$ passes.
  Else, $V$ fails.
```

Figure 2: A Naïve Protocol for graph non-isomorphism problem.

Unfortuntely the running time of this protocol is not polynomial. The size of the set of all n-node graphs is almost $2^{\binom{n}{2}}$ (excluding a small portion of them with non-tirivial automorphism). This space is very big compared to $\mathcal{U}$ (See Figure 3). This implies that with high probability, the random graph G is not in $\mathcal{U}$.

So, the verifier has to repeat the protocol many times which contradicts its polynomial running time limit.

2.2 Sampling via Universal Hashing

As we had before, we need a sampling that hits $\mathcal{U}$ more frequently. The idea is to use a hash function to make the space much smaller than before (see Figure 4). Assume m is the number of bits that we need to describe an n-node graph. So, m is roughly $\binom{n}{2}$. We need a hash function $h : \{0,1\}^m \to \{0,1\}^l$with following properties:

- $|h(\mathcal{U})| \approx |\mathcal{U}|$. Since there are some collisions, $|h(\mathcal{U})| \leq |\mathcal{U}|$. However, it should not be much smaller since we need to estimate $|\mathcal{U}|$ based on $|h(\mathcal{U})|$. So, $|h(\mathcal{U})|$ should be big if and only if $|\mathcal{U}|$ is big.
- $\frac{|h(\mathcal{U})|}{2^l} \approx \frac{1}{Poly(m)}$. This property guarantees that we don't need a huge number of samples to hit $\mathcal{U}$.
- Obviously, h has to be a computable function in polynomial time.

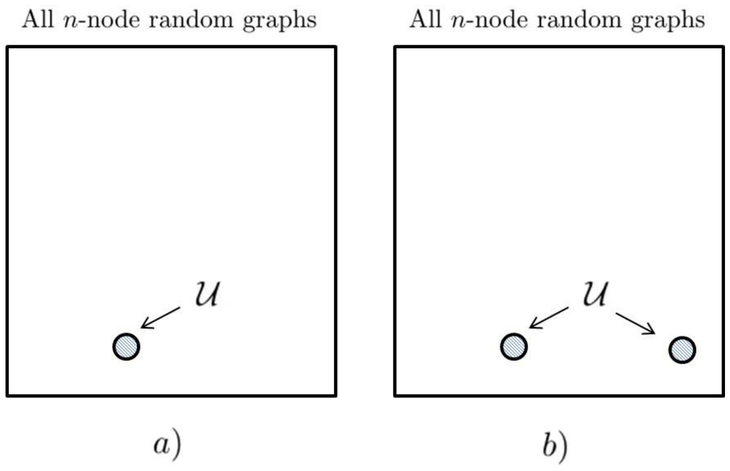

Figure 3: $a)$ Set $\mathcal{U}$ in all n-node random graphs space, while $A \cong B$, $b)$ Set $\mathcal{U}$ in all n-node random graphs space, while $A \ncong B$,

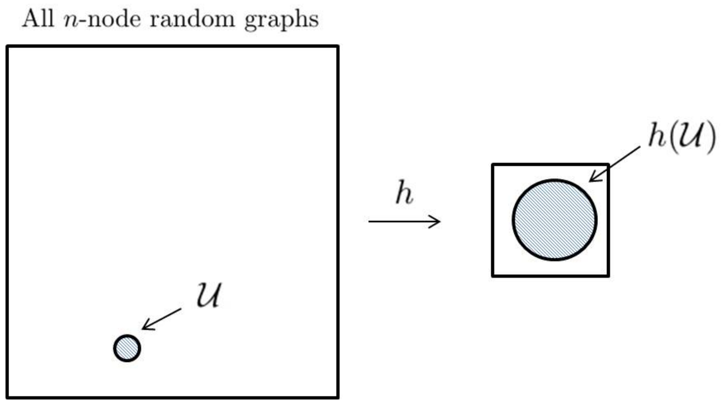

Figure 4: The domain and range of hash function h compare to $\mathcal{U}$ and $h(\mathcal{U})$

Figure 5 is the protocol that verifier V and prover P have to use:

Protocol:

Given $\mathcal{H}$, a collection of all pairwise independent functions mapping $\{0,1\}^m \rightarrow \{0,1\}^l$
V picks a function h from $\mathcal{H}$ uniformly at random (i.e. $h \in_R \mathcal{H}$).
$V \rightarrow P$: Verifier gives h to the prover.
$P \rightarrow V$: The prover gives a graph $X \in \mathcal{U}$ such that $h(X) = 0^l$ with proof that $X \in \mathcal{U}$
If P provides the proof, then V accepts.
Else, V rejects.

Figure 5: A Protocol using universal hash for the graph non-isomorphism problem.

The idea in this protocol is that if $|\mathcal{U}|$ is big (i.e. $2(n!)$), then $h(\mathcal{U})$ "fills" the range of h and with high probability hits 0^l.

On the other hand, if $|\mathcal{U}|$ is small (i.e. $n!$), then $h(\mathcal{U})$ hits part of the range and has less chance to hit 0^l. Lemma 3 describes this more precisely.

Definition 2 *h is a* pairwise independent function *if for all $x, y \in \{0,1\}^m$ and for all $a, b \in \{0,1\}^l$,* $\Pr[h(x) = a \text{ and } h(y) = b] = 2^{-2l}$

Lemma 3 *Assume $h : \{0,1\}^m \to \{0,1\}^l$ is a pairwise independent function which is picked uniformly at random from $\mathcal{H}$. Let $\mathcal{U} \subseteq \{0,1\}^m$ and $a = \frac{|\mathcal{U}|}{2^l}$. Then, we have:*

$$a - \frac{a^2}{2} \le \Pr_h[0^l \in h(\mathcal{U})] \le a$$

Proof

- Proof of the right hand side: Since, h is pairwise independent, $\Pr[h(x) = 0^l] = 2^{-l}$. By union bound, we have:

$$\Pr_h[0^l \in h(\mathcal{U})] \le \sum_{x \in \mathcal{U}} \Pr[h(x) = 0^l] = \frac{|\mathcal{U}|}{2^l} = a$$

- Proof of the left hand side:

  Recall the inclusion-exclusion principle:

$$\Pr[\cup_{i=1}^n A_i] = \sum_{i=1}^n \Pr[A_i] - \sum_{1 \le i < j \le n} \Pr[A_i \cap A_j] + \sum_{1 \le i < j < k \le n} \Pr[A_i \cap A_j \cap A_k] - \ldots + (-1)^{n-1} \Pr[A_1 \cap \ldots \cap A_n]$$

  Consider that if we pick an even number of terms from the right hand side, we can get a lower-bound for $\Pr[\cup_{i=1}^n A_i]$. Intuitively,
  when $n = 3$, we have:

$$\begin{aligned} \Pr[\cup_{i=1}^3 A_i] &= \textstyle\sum_{i=1}^3 \Pr[A_i] - \sum_{1 \le i < j \le 3} \Pr[A_i \cap A_j] + \Pr[A_1 \cap A_2 \cap A_3] \\ &\ge \textstyle\sum_{i=1}^3 \Pr[A_i] - \sum_{1 \le i < j \le 3} \Pr[A_i \cap A_j] \end{aligned}$$

  We use the first two terms as a lower-bound:

$$\Pr[\cup_{i=1}^n A_i] \ge \sum_{i=1}^n \Pr[A_i] - \sum_{1 \le i < j \le n} \Pr[A_i \cap A_j]$$

  Let A_x denote the event that $h(x) = 0^l$. So, we can say:

$$\begin{aligned} \Pr[0^l \in h(\mathcal{U})] &\ge \textstyle\sum_{x \in \mathcal{U}} \Pr[0^l = h(x)] - \sum_{x,y \in \mathcal{U}, x \ne y} \Pr[h(x) = h(y) = 0^l] \\ &\ge \textstyle\sum_{x \in \mathcal{U}} 2^{-l} - \sum_{x,y \in \mathcal{U}, x \ne y} 2^{-2l} \\ &\ge \tfrac{|\mathcal{U}|}{2^l} - \tbinom{\mathcal{U}}{2} \tfrac{1}{2^{2l}} \\ &\ge \tfrac{|\mathcal{U}|}{2^l} - \tfrac{|\mathcal{U}|^2}{2} \tfrac{1}{2^{2l}} \\ &\ge a - \tfrac{a^2}{2} \end{aligned}$$

■

Finishing up?

Pick l such that $2^{l-2} \leq 2(n!) \leq 2^l$.

- if $A \ncong B$, then $|\mathcal{U}| = 2(n!)$. So, $\frac{1}{2} \leq a \leq 1$. Thus, we have $\Pr[V \text{ accepts }] \geq a - \frac{a^2}{2} = \frac{3}{8} = \alpha$.
- if $A \cong B$, then $|\mathcal{U}| = n!$. So, $\frac{1}{4} \leq a \leq \frac{1}{2}$. Thus, we have $\Pr[V \text{ accepts }] \leq a = \frac{1}{2} = \beta$.

Whoops! However, we need $\alpha > \beta$. So, we should use a small trick. The solution is in homework.

3 More Derandomization: Conditional Expectations

The main idea of this method is viewing the coin tosses of the algorithm as a path down a tree of depth m where m is the number of coins. See Figure 6. Clearly, for a good randomized algorithm, most leaves are good. i.e. most of the time the algorithm gives a good answer. Here, the definition of a "good answer" depends on the problem and our desired accuracy. We want to find the path of the algorithm "bit-by-bit", which means we will pick the branch that we expect to result in a better answer.

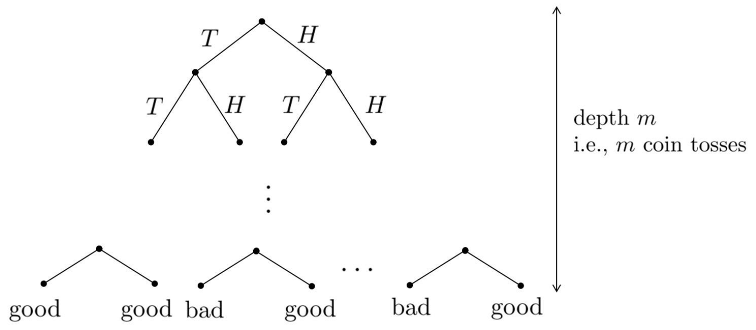

Figure 6: A run tree of a randomized algorithm uses m random bit.

More formally, fix a randomized algorithm $\mathcal{A}$ and input X. Suppose $\mathcal{A}$ uses m random bits on input X. For all $1 \leq i \leq m$ and $r_1, r_2, \ldots, r_i \in \{0,1\}$, let $P(r_1, \ldots, r_i)$ be the fraction of continuations that end in a "good" leaf. So, we have:

$$P(r_1, \ldots, r_i) = \frac{1}{2}P(r_1, \ldots, r_i, 0) + \frac{1}{2}P(r_1, \ldots, r_i, 1)$$

Equivalently, we have:

$$\begin{aligned} \Pr_{R_{i+1},\ldots,R_m}[A(X, r_1, \ldots, r_i, R_{i+1}, \ldots, R_m) \text{ correct}] &= \tfrac{1}{2}\Pr_{R_{i+2},\ldots,R_m}[A(X, r_1, \ldots, r_i, 0, R_{i+2}, \ldots, R_m) \text{ correct}] \\ &+ \tfrac{1}{2}\Pr_{R_{i+2},\ldots,R_m}[A(X, r_1, \ldots, r_i, 1, R_{i+2}, \ldots, R_m) \text{ correct}] \end{aligned}$$

Thus, by averaging, there exists a setting of r_{i+1} to 0 or 1 such that

$$P(r_1, \ldots, r_{i+1}) \geq P(r_1, \ldots, r_i)$$

If we have an algorithm to figure out which setting is appropriate for r_{i+1}, we can always pick the corresponding branch. So for all $1 \leq i \leq m$, we have:

$$P(r_1, \ldots, r_m) \geq P(r_1, \ldots, r_{m-1}) \geq \ldots \geq Pr(r_1) \geq p(\Lambda) \geq \frac{2}{3}$$

where Λ is the fraction of good leave in the tree. However, after having m random bits we are in a leaf. So, $P(r_1, \ldots, r_m)$ is one or zero. Since it is greater than $\frac{2}{3}$, it must be one.

The hard part in this approach is figuring out the best setting of r_{i+1} at each step. Here is an example of how we may do that:

3.1 Derandomization of Max-Cut

In Figure 7, we describe a randomized algorithm for the max-cut problem.

```
Max-Cut
  Flip $n$ coins $r_1, \ldots, r_n$.
  Put node $i$ in $S$ if $r_i = 0$.
  Otherwise, put node $i$ in $T$.
  output $S$, $T$.
```

Figure 7: A randomized algorithm for the max-cut problem

Derandomization: Let $e(r_1, \ldots, r_i)$ denote $E_{R_{i+1},\ldots,R_n}[|cut(S,T)|$ given $r_1, \ldots, r_i$ choices made.]. In the previous lecture, we showed that $e(\Lambda) \geq \frac{|E|}{2}$. Now, we want to show how to calculate $e(r_1, \ldots, r_i)$. Let

$S_{i+1} = \{j \in V | j \leq i+1 \text{ and } r_j = 0\}$

$T_{i+1} = \{j \in V | j \leq i+1 \text{ and } r_j = 1\}$

$U_{i+1} = \{j \in V | i+1 \leq j \leq n\}$.

Actually, S_{i+1} and T_{i+1} denote S and T till $(i+1)$-th iteration of the algorithm. Also, U_{i+1} denotes undecided nodes.

So, we have:

$e(r_1, \ldots, r_{i+1}) = (\text{number of edges between } S_{i+1} \text{ and } T_{i+1}) + \frac{1}{2}(\text{number of edges touching } U_{i+1})$.

Note that we do not have to calculate the exact value of $e(r_1, \ldots, r_{i+1})$, we just need to compare them. In the above equation, the second term is equal for both $e(r_1, \ldots, r_i, 0)$ and $e(r_1, \ldots, r_i, 1)$. Moreover, the second term only differs on edges adjacent to the node $i+1$. So, to maximize the $e(r_1, \ldots, r_{i+1})$, place node $i+1$ in the set that maximize number of edges between S_{i+1} and T_{i+1}. Thus, we reach the following algorithm:

```
Derandomized Max-Cut
  $S \leftarrow \emptyset$, $T \leftarrow \emptyset$
  For $i = 0, \ldots, N-1$
  If (number of edges between $i$ and $T$) $\leq$ (number of edges between $i$ and $S$)
    Place node $i$ in $S$.
  Else
    Place node $i$ in $T$.
  output $S$, $T$.
```

Figure 8: The derandomized version of max-cut algorithm