6.842 Randomness and Computation February 28, 2022

Lecture 9

Lecturer: Ronitt Rubinfeld *Scribe: Richard Qi*

1 Interactive Proofs

Definition 1 (Review from previous Lecture) *An **interactive proof system** for a language L with a verifier V and prover P is a protocol such that:*

1. *If $x \in L$, and V and P follow protocol, then $Pr_{V\text{'s coins}}[V \text{ accepts}] \geq 2/3$*
2. *If $x \notin L$, and V follows protocol, $Pr_{V\text{'s coins}}[V \text{ accepts}] \leq 1/3$, no matter what P does.*

It is possible to show that V does not need to use private coins. In other words, any interactive proof system that uses private coins can be modified so that is an interactive proof system that does not use private coins.

1.1 Introduction to the Set Size Interactive Proof Problem

Instead of proving that we can always use public coins instead of private coins, we will consider a simpler example. Suppose that P wants to convince V that a set if big. Let $S_\phi = \{X | X \text{ satisfies } \phi\}$ be some set of boolean assignments. (This could be replaced by any language $L \in IP$, but we use this example for simplicity.)

Theorem 2 (Interactive Proof for Set Size) *There exists some protocol **using only public coins** such that on input ϕ:*

1. *If $|S_\phi| > k$ and V, P follow protocol, then $Pr[V \text{ accepts}] \geq 2/3$.*
2. *If $|S_\phi < \frac{k}{\Delta}$ and V follows protocol, then $Pr[V \text{ accepts}] \leq 1/3$.*

Consider the space of all assignments. One idea we could use is the following process:

1. For some number of iterations, V picks a random assignment x, then evaluates $\phi(x)$.
2. Then, V outputs $\frac{\text{number of satisfying assignments}}{\text{total number of repetitions}}$ as an estimate for the fraction of all assignments that are satisfying.

The expected amount of time V needs in the above problem just to find a single satisfying assignment is $\frac{2^n}{|S_\phi|}$, which could be very large if $|S_\phi|$ is relatively small. Alternatively, we could ask the Prover in the protocol to return all satisfying assignments in S_ϕ and have the Verifier check each one. This could also take a large amount of time if $|S_\phi|$ is large.

1.2 Universal Hashing Introduction

Some family of functions $\mathcal{H} = \{h_1, h_2, \cdots\}$ for $h_i : [N] \to [M]$ is "pairwise independent" if, when h is chosen randomly from $\mathcal{H}$:

1. $\forall x \in [N], h(x) \in_U [M]$.
2. $\forall x_1 \neq x_2 \in [N]$, $(h(x_1), h(x_2)) \in_U [M]^2$.

Here, $\in_U$ denotes uniformity. Equivalently, the first condition implies that any location x is mapped uniformly, and the second condition implies that any pair of locations x_1, x_2 is mapped uniformly and independently.

Now, suppose we use some randomly drawn h from a family of pairwise independent functions $\mathcal{H}$ that maps the set of all assignments to a set of assignments of size 2^l, where we pick l such that $2^{l-1} \leq k \leq 2^l$. Then, h maps any element $x \in S_p$ to some element in $h(S_\phi)$. Clearly, $|h(S_\phi)| \leq |S_\phi|$ because every input is mapped to a single output. Our hope is that $|h(S_\phi)|$ is not too much smaller than $|S_\phi|$.

1.3 Note on Pairwise Independent Hash Functions

Pairwise independent hash functions can be used to map some space into a smaller space, for example mapping a list of names to some smaller sized strings. This is good because it reduces the amount of storage needed. However, there could be collisions, which is bad. Pairwise independence give some upper bound on the number of collisions.

1.4 Protocol

We now define the actual protocol used for solving the "set size" problem that was first introduced.

1. V picks some h uniformly and randomly from $\mathcal{H}$.
2. V sends h to P.
3. P sends some $x \in S_\phi$ such that $h(x) = 0^l$, if possible.
4. V accepts iff $x \in S_\phi$, which they can check quickly. Note that if S_ϕ was some other type of language in IP, this step would become slightly more complicated because another protocol would have to be used to check whether $x \in L$. For $L = S_\phi$, V can easily check this by themself.

First, suppose $|S_\phi| > k$. Our hope is that $|h(S_\phi)| \approx k$ so that 0^l is hit by the hash function with a reasonably large probability. Then, P will be able to find some x such that $h(x) = 0^l$, which they will send back to V.

Next, suppose $|S_\phi| < \frac{k}{\Delta}$. Then, $|h(S_\phi)| < \frac{k}{\Delta}$, so our hope is that it is much less likely that 0^l is hit compared to the first case. If 0^l is not hit by h, then P cannot find some x such that $h(x) = 0^l$. If they send an incorrect x such that $h(x) \neq 0^l$, V will be able to detect that an incorrect x was sent.

1.5 Formal Analysis of the Protocol

Lemma 3 *If $\mathcal{H}$ is some pairwise independent hash family, and $U \in \{0,1\}^n$, and $a = \frac{|U|}{2^l}$, then we must have $a - \frac{a^2}{2} \leq Pr_{h\in\mathcal{H}}[0^l \in h(U)] \leq a$.*

Proof $\forall x, Pr_{h\in\mathcal{H}}[0^l = h(x)] = \frac{1}{2^l}$ by pairwise independence. Next,

$$Pr[0^l \in h(U)] \leq \sum_{x\in U} Pr[0^l \in h(x)] = \frac{|U|}{2^l} = a, \tag{1}$$

where the inequality follows from Union Bound. This proves the right hand side of the lemma.
For the other side, consider

$$Pr_{h\in\mathcal{H}}[0^l \in h(U)] \geq \sum_{x\in U} Pr[0^l = h(x)] - \sum_{x\neq y\in U} Pr[0^l = h(x) = h(y)], \tag{2}$$

which follows from inclusion-exclusion. Again, from pairwise independence, we have that

$$\sum_{x\in U} Pr[0^l = h(x)] - \sum_{x\neq y\in U} Pr[0^l = h(x) = h(y)] = a - \binom{|U|}{2} \cdot \frac{1}{2^{2l}} \geq a - \frac{a^2}{2}, \quad (3)$$

as desired. ■

Recall that l was chosen such that $2^{l-1} \leq k \leq 2^l$. Let $a = \frac{|S_\phi|}{2^l}$. Also, choose $\Delta = 4$.

- If $|S_\phi| > k$, then $a \geq \frac{k}{2^l} \geq 1/2$, so $Pr[0^l \in h(S_\phi)] \geq a - \frac{a^2}{2} \geq 3/8$.
- If $|S_\phi| < \frac{k}{\Delta}$, then $a < \frac{\frac{k}{\Delta}}{2^l} \leq \frac{1}{\Delta}$, so $Pr[0^l \in h(S_\phi)] \leq a \leq \frac{1}{\Delta} = 1/4$.

Now, we can repeat the process of choosing hash functions h and following the described protocol. If $|S_\phi| > k$ and protocol is followed, then number of times that V accepts approaches some fraction at least 3/8. If $|S_\phi| < \frac{k}{\Delta}$, then no matter what the Prover does, the number of times that V accepts approaches some fraction at most 1/4. With high probability using Chernoff bounds, we can distinguish between the two cases.

2 Derandomization via Conditional Expectation

2.1 General Description

2.1.1 Intuitive Idea

Consider some tree of scenarios, where at each node of the tree, we flip a coin, and depending on whether the result is 0 or 1, we can possibly go to two possible child nodes. Then, if there are m total coin tosses, some of that 2^m possible results of the coin tosses are "good", and some of them are "bad". In other words, the "good" results of the coin tosses are results where our randomized algorithm works, and the opposite happens for "bad results". For example, good results could correspond to when an algortihm returns a cut with a high number of edges crossing the cut, while bad results could correspond to when an algorithm returns a cut with a low number of edges. The general idea of Derandomization via Conditional Expectation is to deterministically decide which node of the tree to travel down instead of flipping a coin.

2.1.2 Formal Definition

Definition 4 *Let m be the number of coin tosses. Denote the results of the coin tosses be $r_1, r_2 \cdots r_i \in \{0,1\}^d$. Let $p(r_1, \cdots r_i)$ be the fraction of continuations of the coin tosses that end up in a "good" leaf, given that the results of our first i coin tosses were $r_1, \cdots r_i$. Let $e(r_1, \cdots r_i)$ be the expected cut value of the maximum cut value given the first i coin toss results.*

By definition, we have $p(r_1 \cdots r_i) = \frac{1}{2}p(r_1 \cdots r_i, 0) + \frac{1}{2}p(r_1 \cdots r_i, 1)$ and $e(r_1 \cdots r_i) = \frac{1}{2}e(r_1 \cdots r_i, 0) + \frac{1}{2}e(r_1 \cdots r_i, 1)$.

Lemma 5 *Suppose that at each step, we chose the value of our coin tosses r_i such that for all i, $p(r_1 \cdots r_{i+1}) \geq p(r_1 \cdots r_i)$. Suppose $p(\Lambda) > 0$, where Λ denotes the situation where no coin tosses have been determined. Then, the sequence of deterministically chosen tosses $r_1, \cdots r_m$ corresponds to a good result.*

Proof From the statement, we must have $p(r_1 \cdots r_m) \geq p(r_1 \cdots r_{m-1}) \geq \cdots \geq p(r_1) \geq p(\Lambda) > 0$. However, $p(r_1 \cdots r_m)$ corresponds to the situation where all m tosses have been determined and no random coin flips remain, so it must be either 1 or 0. Since it is greater than 0, it must be 1, so $r_1, \cdots r_m$ corresponds to a good result. ■

Lemma 6 *Suppose that at each step, we chose the value of our coin tosses r_i such that for all i, $e(r_1, \cdots r_{i+1}) \geq e(r_1, \cdots 1 - r_{i+1})$. Then, the sequence of deterministically chosen tosses $r_1, \cdots r_m$ satisfies $e(r_1, \cdots r_m) \geq e(\Lambda)$.*

Proof Recall that $e(r_1 \cdots r_i) = \frac{1}{2}e(r_1 \cdots r_i, 0) + \frac{1}{2}e(r_1 \cdots r_i, 1)$. Thus, if we choose r_{i+1} such that $e(r_1, \cdots r_{i+1}) \geq e(r_1, \cdots 1 - r_{i+1})$, then we also have $e(r_1, \cdots r_i) = \frac{e(r_1, \cdots r_{i+1}) + e(r_1, \cdots 1 - r_{i+1})}{2} \leq e(r_1, \cdots r_{i+1})$. Then, using the same argument as Lemma 5, we have $e(r_1, \cdots r_m) \geq e(r_1, \cdots r_{m-1}) \cdots \geq e(\Lambda)$. ■

Essentially, what this last lemma is saying is that if at every node of the tree, we go down the branch that leads to a higher expected value, then our ending value is at least our starting expected value if we chose to flip a fair coin at each node to determine our path.

2.2 Derandomization in Max Cut

Recall in an earlier lecture the randomization algorithm for finding a large max cut. We flip n coins, and place node i in S if $r_i = 0$, and otherwise place node i in T. We previously showed that the expected size of the cut is at least $\frac{|E|}{2}$. This indicates that $e(\Lambda) \geq \frac{|E|}{2}$, where Λ denotes that no r_i have been decided yet.

2.2.1 Greedy Algorithm

Consider the following greedy algorithm:

1. Initialize S, T as empty sets.
2. For all i from 0 to $n - 1$, do the following: place v_{i+1} in S if the number of edges between v_{i+1} and T is at least the number of edges between v_{i+1} and S. Else, place v_{i+1} in T.

Note that our greedy algorithm is essentially choosing the flips of the random bits that the randomization algorithm uses.

2.2.2 Analysis of the Greedy Algorithm

First, we consider the expression $e(r_1, \cdots r_i)$, which the expected value of the max cut given that we have already fixed the sets that the first i nodes belong to.

Definition 7 *Consider some time i, where for every node $j \leq i$ we have decided which set it belongs to. Define $S_i = \{\text{nodes } j | j \leq i, r_j = 0\}$, which is the set of nodes we have decided to be in S. Similarly, define $T_i = \{\text{nodes } j | j \leq i, r_j = 1\}$ to be the set of nodes we have decided to be in T. Define $U_i = V \setminus S_i \setminus T_i$ as the set of nodes that have not yet been decided to be in either S or T.*

Claim 8 $e(r_1 \cdots r_{i+1}) = (\text{number of edges between } S_{i+1} \text{ and } T_{i+1}) + \frac{1}{2}(\text{number of edges touching } U_{i+1})$.

Proof This follows from the analysis of a previous lecture. Note that the edges between S_i and T_i will end up in the final cut, and the set of all possible remaining edges that could be in the cut is precisely the set of edges touching U_i. Each of these has a probability 1/2 of being in the final cut if we flip a coin for each node $j \geq i + 1$ to go on either side of the cut. ■

Lemma 9 *The greedy algorithm chooses r_{i+1} such that $e(r_1, \cdots r_{i+1}) \geq e(r_1, \cdots 1 - r_{i+1})$.*

Proof Suppose the greedy algorithm chooses $r_{i+1} = 0$ (the other case is exactly the same). This means that the number of edges between node $i+1$ and T_i was at least the number of edges between node $i+1$ and S_i.

Now, compare $e(r_1, \cdots r_{i+1})$ against $e(r_1, \cdots 1 - r_{i+1})$, which is the situation where an algorithm copies the greedy algorithm but chooses the opposite of the what the greedy would have chosen at time $i+1$. Clearly, the set U_{i+1} is the same in both cases, so the term $\frac{1}{2}$(number of edges touching U_{i+1}) in claim 8 is equal in both cases. The only difference in the term (number of edges between S_{i+1} and T_{i+1}) is when examining edges between node $i+1$ and either S_i or T_i. However, the greedy algorithm precisely chose r_{i+1} such that the number of nodes between $i+1$ and either S_i or T_i was maximized, so we must have $e(r_1, \cdots r_{i+1}) \geq e(r_1, \cdots 1 - r_{i+1})$. ■

Theorem 10 *The previously mentioned greedy algorithm deterministically achieves a max cut of size at least* $\frac{|E|}{2}$.

Proof This follows from our claims that $e(\Lambda) = \frac{|E|}{2}$, Lemma 9, and Lemma 6. ■