**6.895 Randomness and Computation** February 22, 2012

# Lecture 4

*Lecturer: Ronitt Rubinfeld* *Scribe: Cesar A. Cuenca*

## 1 Overview:

In this lecture, we look at some of the complexity theory aspects of the course and try to find at how extent we need randomness in computation. The topics for today are:

- Randomized Complexity Classes
- Derandomization via Enumeration
- Pairwise Independence and Derandomization (we will use MAXCUT as an example)

## 2 Randomized Complexity Classes:

We start by introducing some of the most basic definitions.

**Definition 1** *A **language** $L$ is a subset of $\{0,1\}^*$.*

A language $L$ may define all kinds of objects, like graphs that satisfy certain properties, a set of prime numbers, etc.

**Definition 2** ***P (polynomial time)** is the class of languages $L$ for which there is a deterministic polynomial-time algorithm $\mathcal{A}$ such that:*

- $x \in L \Longrightarrow \mathcal{A}(x)$ *accepts*
- $x \notin L \Longrightarrow \mathcal{A}(x)$ *rejects*

Now we introduce two of the most important Randomized Complexity Classes.

**Definition 3** ***RP (randomized polynomial time)** is the class of languages $L$ for which there is a polynomial-time (probabilistic) algorithm $\mathcal{A}$ such that:*

- $x \in L \Longrightarrow Pr[\mathcal{A}(x)\ accepts] \geq 1/2$
- $x \notin L \Longrightarrow Pr[\mathcal{A}(x)\ accepts] = 0$

We can think of RP as the class of languages $L$ with *one-sided* error. This is, if $x \in \{0,1\}^*$ and $\mathcal{A}(x)$ accepts, then we know that $x \in L$; however if $\mathcal{A}(x)$ rejects, it may occur that $x \in L$ or $x \notin L$.

**Definition 4** ***BPP (bounded-error probabilistic polynomial time)** is the class of languages $L$ for which there is a polynomial-time (probabilistic) algorithm $\mathcal{A}$ such that:*

- $x \in L \Longrightarrow Pr[\mathcal{A}(x)\ accepts] \geq 2/3$
- $x \notin L \Longrightarrow Pr[\mathcal{A}(x)\ accepts] \leq 1/3$

The class BPP may fail to guess whether or not $x \in L$ for any $x \in \{0,1\}^*$ (it has *two-sided error*). Note also that the constants $2/3$ and $1/3$ (in the definition of BPP) can be replaced for any other constants $a$ and $b$ such that $1 > a > 1/2$ and $1/2 > b > 0$ respectively. Similarly the constant $1/2$ in the definition of RP can be replaced for any constant $1 > c > 0$.

# 3 Derandomization via Enumeration

From definitions 2, 3 and 4, we clearly see that $P \subseteq RP \subseteq BPP$. However, it remains an open question to determine whether $P$ equals $BPP$. There have been lots of results about this recently and some researchers believe that indeed $P = BPP$. This leads us to believe that it is possible to remove randomness (or use as little of it as possible) from any efficient randomized algorithm. A most basic method is what's called *derandomization via enumeration*. The idea is outlined below.

**Idea:** Assume that a randomized algorithm $\mathcal{A}$ randomly makes choices from a domain $D$ of values (which are strings of bits) and uses them to give an approximate solution to the problem at hand. To derandomize $\mathcal{A}$, a deterministic algorithm $\mathcal{B}$ would instead enumerate all values of $D$ and loop through all of them to find an optimal solution.

**Observation 5** *The algorithm $\mathcal{B}$ described above is deterministic and gives the optimal solution (for an optimization problem), but its running time may be very slow. Define $T_{\mathcal{A}}(n)$ and $T_{\mathcal{B}}(n)$ to be upper bounds on the runtimes of $\mathcal{A}$ and $\mathcal{B}$ on inputs of size $n$ respectively. Then we clearly have $T_{\mathcal{B}}(n) \leq T_{\mathcal{A}}(n)|D|$.*

**Observation 6** *Let $r(n)$ be the number of random bits used by $\mathcal{A}$ for inputs of size $n$, then $D \subseteq \{0,1\}^{r(n)}$ and so $|D| \leq 2^{r(n)}$. In addition, since $\mathcal{A}$ uses $r(n)$ random bits (and assuming a bit is chosen in one unit of time), then $r(n) \leq T_{\mathcal{A}}(n)$. We conclude that $|D| \leq 2^{T_{\mathcal{A}}(n)}$.*

From the inequalities derived above, we obtain $T_{\mathcal{B}}(n) \leq T_{\mathcal{A}}(n)|D| \leq T_{\mathcal{A}}(n)2^{T_{\mathcal{A}}(n)}$. Therefore

$$T_{\mathcal{B}}(n) = O(T_{\mathcal{A}}(n)2^{T_{\mathcal{A}}(n)}) \tag{1}$$

**Corollary 7** $BPP \subseteq EXP = \bigcup_{k \in \mathbb{N}} DTIME(2^{n^k})$

**Proof** Indeed, since $\mathcal{A}$ is a polynomial-time randomized algorithm, then we can consider $T_{\mathcal{A}}(n)$ as a polynomial in $n$. Therefore the expression $T_{\mathcal{A}}(n)2^{T_{\mathcal{A}}(n)}$ is exponential in $n$. From this observation and equation (1), the result follows. ■

**Observation 8** *If we reduce the number of bits $r(n)$ used by algorithm $\mathcal{A}$, we can find a polynomial-time algorithm $\mathcal{B}$ under this method of derandomization. More specifically, if $r(n) = O(\lg n)$ (say $r(n) \leq c \lg n$ for some constant $c$), the runtime of algorithm $\mathcal{B}$ becomes $O(T_{\mathcal{A}}(n)2^{r(n)}) = O(T_{\mathcal{A}}(n)2^{c \lg n}) = O(T_{\mathcal{A}}(n)n^c)$, which is polynomial in $n$.*

# 4 Pairwise Independence and Derandomization

## 4.1 Randomized MAXCUT:

Oftentimes, the only condition we ever use from randomly selected bits is pairwise independence, and not independence at its full strength. We use the MAXCUT algorithm as an example to work on a concrete problem. We will show that the only condition we need from the random bits MAXCUT uses is pairwise independence (to be abbreviated as p.i.). We begin by defining the *Maximum Cut* problem.

**Input:** Graph $G = (V, E)$ (w.l.o.g. assume that $V = \{1, 2, \ldots, |V|\}$).

**Output:** A partition $S \cup T$ of $V$ that maximizes the *cut value* $f(S,T) = |\{\{u,v\} \in E | u \in S, v \in T\}|$.

The first attempt to try to solve it could be to toss a coin for each vertex $v \in V$. If the coin gives a head, then place $v$ in $S$ and otherwise, place it in $T$, where $S \cup T$ becomes the partition of $V$. This

simple algorithm (to be called MAXCUT) actually works well; more specifically it returns a partition $V = S \cup T$ whose value $f(S,T)$ is at most half the maximum value for a cut of $G$ (in average).

**Randomized MAXCUT**

**for** $i = 1$ to $|V|$ **do**
  Let *randBit* be a random bit uniformly chosen (from $\{0,1\}$)
  **if** randBit $= 0$ **then**
    $S \leftarrow S \cup \{i\}$
  **else**
    $T \leftarrow T \cup \{i\}$
  **end if**
**end for**

**Analysis of the algorithm:** First, define a function $side : V \to \{S,T\}$ so that side$(v)$ (to be denoted as $r_v$) is the set of the partition where $v$ belongs. Also define the next indicator random variables for $u \neq v$ in $V$:

$$I_{side(u)\neq side(v)} = I_{u,v} = \begin{cases} 1 & \text{if } r_u \neq r_v \\ 0 & \text{if } r_u = r_v \end{cases}$$

Clearly the value of $f(S,T)$ of the cut is the number of edges $(u,v)$ for which side$(u) \neq$ side$(v)$, or more formally $f(S,T) = \sum_{(u,v)\in E} I_{u,v}$. Therefore:

$$\begin{aligned}
\mathbb{E}[f(S,T)] = \mathbb{E}[\sum_{(u,v)\in E} I_{u,v}] &= \sum_{(u,v)\in E} \mathbb{E}[I_{u,v}] && (2)\\
&= \sum_{(u,v)\in E} \mathbb{P}[r_u \neq r_v] && (3)\\
&= \sum_{(u,v)\in E} \mathbb{P}[\{r_u = 1 \wedge r_v = 0\} \vee \{r_u = 0 \wedge r_v = 1\}] && (4)\\
&= \sum_{(u,v)\in E} \mathbb{P}[r_u = 1]\mathbb{P}[r_v = 0] + \mathbb{P}[r_u = 0]\mathbb{P}[r_v = 1] && (5)\\
&= \sum_{(u,v)\in E} \frac{1}{2} = \frac{|E|}{2} && (6)
\end{aligned}$$

,where (2) comes from linearity of expectation; (3) comes from the definition of $I_{u,v}$; (4) comes from the interpretation of the event $\{r_u \neq r_v\}$; (5) comes from the fact that events $\{r_u = 1 \wedge r_v = 0\}$ and $\{r_u = 0 \wedge r_v = 1\}$ are disjoint and events $\{r_u = a\}$ and $\{r_v = b\}$ (for any $a, b \in \{0,1\}$) are independent; finally (6) comes from noticing all probability values involved are $1/2$.

**Observation 9** *Note that the maximum value of any cut cannot exceed $|E|$, and we just proved that the expected value of the cut returned by MAXCUT is $|E|/2$. Then we proved that MAXCUT gives a multiplicative approximation to within the factor of 2 for the maximum cut problem.*

## 4.2 Pairwise Independent Random Variables:

We now introduce some definitions on independence. Denote the event $\{X = a \wedge Y = b\}$ by the shorter form $\{XY = ab\}$, if $X, Y$ are random variables which may take on values $a, b$, respectively. A similar notation will be used for an arbitrary number of random variables.

In the next definitions, the setting is as follows: We have $n$ random variables $X_1, X_2, \ldots X_n$ which have values from a domain $T$ of size $|T| = t$, i.e. $X_i \in T$ for all $1 \leq i \leq n$.

**Definition 10** *We say that $X_1, X_2, \ldots, X_n$ are **independent** if for all values $b_1, b_2, \ldots, b_n \in T$, we have that*

$$\mathbb{P}[X_1 X_2 \ldots X_n = b_1 b_2 \ldots b_n] = \frac{1}{t^n}.$$

**Definition 11** *We say that $X_1, X_2, \ldots, X_n$ are **pairwise independent(p.i.)** if for any $1 \leq i < j \leq n$ and any values $b_i, b_j \in T$, we have that*

$$\mathbb{P}[X_i X_j = b_i b_j] = \frac{1}{t^2}.$$

**Definition 12** *We say that $X_1, X_2, \ldots, X_n$ are **k-wise independent** if for any $1 \leq i_1 < \ldots < i_k \leq n$ and any values $b_1, \ldots, b_k \in T$, we have that*

$$\mathbb{P}[X_{i_1} \ldots X_{i_k} = b_1 \ldots b_k] = \frac{1}{t^k}.$$

**Observation 13** *From definitions 10, 11 and 12, we see that p.i. is weaker than k-wise independence, which in turn is weaker than independence. This is a key observation, because we will see that if an algorithm only uses p.i., then we may reduce (or totally eliminate) the amount of randomness used by the algorithm. We do this in the next subsection for the MAXCUT algorithm.*

## 4.3 Derandomization of Randomized MAXCUT

**First attempt:** By using enumeration, we now derandomize MAXCUT. In fact, enumerate all possible strings of bits of length $|V|$, which are used to find the partition $S \cup T$ of $V$. There are a total of $2^{|V|} = 2^n$ strings. The deterministic algorithm we obtain under this method finds the optimal solution, but its running time (as analyzed before, when describing the enumeration method) is exponential, more specifically $O(2^n T_{\text{MC}}(n))$, where $T_{\text{MC}}(n)$ is the time MAXCUT takes on inputs of size $n$.

**Second attempt:** Instead of using enumeration, we will use partial enumeration. The key observation is that MAXCUT only requires the random bits that it uses to be pairwise independent. First we will show how one can use $O(\lg n)$ random bits to generate $O(n)$ p.i. bits. Then we see how to make another randomized algorithm $\overline{\text{MAXCUT}}$ that uses exponentially fewer random bits and has a similar behavior to MAXCUT. Finally we show how to derandomize $\overline{\text{MAXCUT}}$ using enumeration, obtaining a deterministic algorithm that has the same behavior as MAXCUT and runs in polynomial time.

We start by giving an example of how we can obtain 3 p.i. bits $b_1, b_2$ and $b_3$ using only 2 *truly random bits*. Instead of tossing a coin to find the bits for all three of $b_1, b_2$ and $b_3$, we only toss a coin for $b_1$ and $b_2$ and then determine $b_3$ deterministically as $b_3 = b_1 \oplus b_2$. The truth table looks like this:

| $b_1$ | $b_2$ | $b_3$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Notice that if we delete any of the columns, we obtain all 4 possible combinations of bits for the 2 remaining variables. After some thought, it is clear that $b_1, b_2$ and $b_3$ are p.i..

More generally, we can generate $2^k - 1$ p.i. bits using only $k$ *truly random bits* with the following algorithm, which is called GENERATOR:

- Choose randomly $k$ bits $b_1, b_2, \ldots, b_k$.
- For any non-empty set $S \subseteq [k] = \{1, 2, \ldots, k\}$, define $C_S = \bigoplus_{i \in S} b_i$.
- Then $\{C_S\}_{S \subseteq [k], S \neq \emptyset}$ is a set of cardinality $2^k - 1$, in which the bits are p.i..

The proof that the bits returned by GENERATOR are p.i. is left as an exercise.
Now we show an algorithm that behaves like MAXCUT, but uses significantly less amount of randomness.

**Randomized Algorithm $\overline{\text{MAXCUT}}$**

- Produce $\lfloor \lg n \rfloor + 1$ random bits $b_1, b_2, \ldots, b_{\lfloor \lg n \rfloor + 1}$.
- Use the GENERATOR procedure to construct $n$ p.i. bits $r_1, r_2, \ldots, r_n$ from $b_1, b_2, \ldots, b_{\lfloor \lg n \rfloor + 1}$.
- Use $r_1, r_2, \ldots, r_n$ as the random bits used by MAXCUT.

The main point is that this algorithm has the same expected behavior as that of MAXCUT, but uses only $O(\lg n)$ random bits, as opposed to $O(n)$ random bits of our first attempt. This decrease allows us to use enumeration to find a much more efficient deterministic algorithm. In fact, $\overline{\text{MAXCUT}}$ makes $\lfloor \lg n \rfloor + 1$ random choices. We can then enumerate all possible strings of $\lfloor \lg n \rfloor + 1$ random bits, and loop through all of them keeping the best solution so far. This deterministic algorithm, by the analysis made when we described derandomization by enumeration, has runtime $O(2^{\lfloor \lg n \rfloor + 1}.T_{MC}(n)) = O(2 \times 2^{\lg n} \times T_{MC}(n)) = O(nT_{MC}(n))$, where $T_{MC}(n)$ is a polynomial bound on the time used by MAXCUT. Therefore this deterministic algorithm takes only polynomial time. Note that this new algorithm no longer tries all partitions, and thus does not guarantee achieving the optimal solution (as we had with the exponential time derandomization).