6.842 Randomness and Computation September 18, 2017

Lecture 4

Lecturer: Ronitt Rubinfeld *Scribe: Tossaporn Saengja*

1 Randomized complexity classes RP, BPP

Definition 1 *A "language" L is a subset of 0,1**

Definition 2 *"P" is a class of languages with polynomial time* ***deterministic*** *algorithms A such that*

$$X \in L \Rightarrow A(x) \text{ accepts}$$

$$X \notin L \Rightarrow A(x) \text{ rejects}$$

Definition 3 *"RP" is a class of languages with polynomial time* ***probabilistic*** *algorithms A such that*

$$X \in L \Rightarrow Pr[A(x) \text{ accepts}] \geq \frac{1}{2}$$

$$X \notin L \Rightarrow Pr[A(x) \text{ accepts}] = 0$$

This is called "1-sided error"

We can get more reliable answer by run $A^k(x)$ which is running k times of $A(x)$ with fresh random coins each time:

Algorithm If all k runs reject then reject, else accept

Behavior of algorithms

$$x \notin L \Rightarrow Pr[accept] = 0$$

$$x \in L \Rightarrow Pr[accept] \geq 1 - 2^{-k}$$

$$\beta = 2^{-k} \Rightarrow k \geq \log \frac{1}{\beta}$$

Definition 4 *"BPP" is a class of languages with polynomial time* ***probabilistic*** *algorithms A such that*

$$X \in L \Rightarrow Pr[A(x) \text{ accepts}] \geq \frac{2}{3}$$

$$X \notin L \Rightarrow Pr[A(x) \text{ accepts}] \leq \frac{1}{3}$$

This is called "2-sided error"

We can still get a more reliable answer by running A for k times and taking the majority answer, yielding the following behavior:

$$Pr[each\ run\ is\ correct] \geq \frac{2}{3}$$

$$Pr[majority\ of\ runs\ correct] \geq 1 - Pr[majority\ incorrect]$$

$$\sum_{i=1}^{k} \sigma_{[i^{th}\ run\ correct]} > \frac{k}{2}$$

$$E[\sum_{i=1}^{k} \sigma_{[i^{th}\ run\ correct]}] = \sum_{i=1}^{k} E[\sigma_{[i^{th}\ run\ correct]}] \geq \frac{2}{3}k$$

By Chernoff bound with $\beta = \frac{1}{4}$,

$$Pr[\#runs\ correct < (1-\frac{1}{4})\frac{2}{3}k] \leq e^{\frac{-(\frac{1}{4})^2(\frac{2}{3})k}{2}}$$

$$Pr[\#runs\ correct < \frac{k}{2}] \leq e^{-\frac{k}{48}}$$

Let $k = 48 \log \frac{1}{\delta}$,

$$Pr[\#runs\ correct < \frac{k}{2}] \leq \delta$$

$$Pr[majority\ of\ runs\ correct] \geq 1 - \delta$$

Observation 5 $P \subseteq RP \subseteq BPP$
An open question is whether $P \stackrel{?}{=} BPP$

2 Derandomization

2.1 via enumeration

Given probabilistic algorithm A and input x
$r(n)$ is the number of random bits used by A on inputs of size n.

1. Run A on **every** random string of length $r(|x|)$

   $r(n) \leq$ runtime of A on inputs of size n

2. Output majority answer

Runtime $O(2^{r(n)}t(n))$ where $t(n)$ is the time bound of A

2.2 via pairwise independence

2.2.1 Max Cut problem

Given $G(V, E)$, output partition V into S, T to maximize $|\{(u, v)|u \in S, V \in T\}|$ (i.e. number of cuts)

Randomized algorithm

- Flip n coins $r_1 \cdots r_n$
- Put vertex i on side r_i

Analysis let

$$\mathbb{1}_{u,v} = \quad 1 \quad \text{if } r_u \neq r_v, \quad 0 \quad otherwise$$

$$E[cut] = E[\sum_{(u,v)\in E} \mathbb{1}_{u,v}] = \sum_{(u,v)\in E} E[\mathbb{1}_{(u,v)}] = \sum_{(u,v)\in E} Pr[r_u \neq r_v] = \frac{|E|}{2}$$

This is "2-approximation" as the best answer could be $|E|$

2.2.2 Pairwise Independence

Definition 6 *n values $x_1 \cdots x_n, x_i \in T$ such that $|T| = t$*

"independent" if $\forall b_1 \cdots b_n \in T^n, Pr[x_1 \cdots x_n = b_1 \cdots b_n] = \frac{1}{t^n}$

"pairwise independent" if $\forall i \neq j, b_i b_j \in T^2, Pr[x_i x_j = b_i b_j] = \frac{1}{t^2}$

"k-wise independent" if $\forall$ distinct $i_1 \cdots i_k, b_{i_1} \cdots b_{i_k} \in T^k, Pr[x_i \cdots x_k = b_{i_1} \cdots b_{i_k}] = \frac{1}{t^k}$

Example

r_1	r_2	r_3
0	0	0
0	1	1
1	0	1
1	1	0

say for indices 1,2 $b_1 b_2 = 00 \Rightarrow Pr[x_1 = 0, x_2 = 0] = \frac{1}{4} = \frac{1}{t^2}$

2.2.3 Using Pairwise Independence in Max Cut

	$b_1 \cdots b_m$	$\Rightarrow$	$\boxed{\text{"randomness generator"}}$	$\Rightarrow r_1 \cdots r_n$	$\Rightarrow$	$\boxed{\text{Max Cut algorithm}}$
From the above example:	$m = 2$		$n \geq m$	$n = 3$		

Observation 7 *If the random bits of the generator are good enough for the algorithm, then one can derandomize the algorithm by doing enumeration on the m bits going into the randomness generator. This would require time $O(2^m)$, rather than the usual $O(2^n)$*

Idea Use $m = \log n$ independent random bits, and turn them into n pairwise independent random bits

How to generate?

1. Choose m truly random bits $b_1 \cdots b_m$
2. $\forall s \subset [m]$ s.t. $s \neq \emptyset$, set $c_s = \bigoplus_{i \in S} b_i$
3. Output all $c_s \Rightarrow 2^{m-1}$ bits

<u>exercise</u> why are they pair-wise independent?

Algorithm

For all choices of $b_1 \cdots b_{\log n+1}$

- Run Max Cut using random bits of randomness generator on input $b_1 \cdots b_{\log n+1}$
- Evaluate cut size
- Output best cut size

Runtime $2^{\log n+1} \times$ (runtime of Max Cut + runtime of generator)

Randomness generator as a function $b_1 \cdots b_m$ $a, b \in Z_q$ where q prime

$$r_i \leftarrow a_i + b \text{ } mod \text{ } q, \forall i \in 0...q-1$$

$$b, a + b \text{ } mod \text{ } q, 2a + b \text{ } mod \text{ } q$$

can take a, b, c and use $ci^2 + ai + b \text{ } mod \text{ } q$ to do 3-wise $\Rightarrow$ can generalize to k-wise