0368.4163 Randomness and Computation March 13, 2009

Lecture 3

Lecturer: Ronitt Rubinfeld *Scribe: Roza Pogalnikova and Yaron Orenstein*

Announcements

Homework 1 is released, due 25/03.

Lecture Plan

1. Do we need randomness?
   - A simple randomized algorithm
   - Pairwise independent sample spaces
   - Derandomization
2. How much randomness is needed?
   - 2-point sampling

1 A simple randomized algorithm.

1.1

Problem: Finding a Maximum Cut in a Graph.

Input: Graph $G = (V, E)$, $|V| = n$, $|E| = m$.

Output: Partition of V that maximizes the size of the cut, i.e. partition to distinct sets S and T, that maximizes $(S, T) = \{(u, v) \,|\, (u, v) \in E, u \in S, v \in T\}$

Remark: Max-Cut Problem is NP-complete, i.e no polynomial time solution is expected.

1.2 A greedy algorithm.

Pick vertices one by one. Put a vertex in S or T to maximize number of edges to vertices already in the cut between S and T.
The algorithm gives cut that is at least half the size of the max-cut since at least half the edges cross the cut. (This is not an expectation – it is a deterministic algorithm).

1.3 A randomized algorithm.

The algorithm to get a random cut:

- Flip n coins r_1, r_2, ..., r_n.
- Put vertex i on side r_i to get (S, T).

Analysis:

$$\mathbb{E}[Cut\ size] =^1 \mathop{\mathbb{E}}_{(u,v)\in E}[\Sigma\ I_{(u,v)\ in\ Cut}] \quad =^2 \sum_{(u,v)\in E} \mathbb{E}[I_{(u,v)\ in\ Cut}] =^3 \sum_{(u,v)\in E} Pr[r_v \neq r_u] \quad =^4 \frac{|E|}{2}$$

[1] Examine expectation of every edge to be in the cut
[2] Linearity of expectations ($\mathbb{E}[\Sigma] = \Sigma(\mathbb{E})$)
[3] Expectation of indicator variable is just its probability ($E[I] = Pr.$)
[4] $Pr[r_v = S] \cdot Pr[r_u = T] + Pr[r_v = T] \cdot Pr[r_u = S] = 1/2$

Remark 1: It is not sufficient to find an expectation. One also needs to show that we come close to expectation with reasonable probability (for example, via Chebyshev bounds), but we will not show it here.

Remark 2: In our probability calculations we used only pairwise independence, and it is a much weaker requirement, as we are going to show today.

2 Constructing Pairwise Independent Sample Spaces.

2.1 Pairwise independent Random Variables.

Pick n values x_1, x_2, ..., x_n from a domain T of size t. There are t^n possibilities for $x_1 \dots x_n$.

Definition 1: $x_1, \dots, x_n$ is *(uniform) "independent"*, if $\forall\, b_1 \dots b_n \in T^n$, $Pr[x_1 \dots x_n = b_1 \dots b_n] = \frac{1}{t^n}$

Definition 2: $x_1, \dots, x_n$ is *(uniform) "pairwise independent"*, if $\forall\, i, j$, $b_i b_j \in T^2$, $Pr[x_i x_j = b_i b_j] = \frac{1}{t^2}$

Definition 3: $x_1, \dots, x_n$ is *(uniform) "k-wise independent"*, if $\forall\, i_1 \dots i_k$, $b_1 \dots b_k \in T^k$, $Pr[x_{i_1} \dots x_{i_k} = b_1 \dots b_k] = \frac{1}{t^k}$

Example: $T = \{0,\ 1\}$, P and Q different distributions.

x_0	x_1	x_2	$Pr[P]$	$Pr[Q]$
0	0	0	1/8	1/4
0	0	1	1/8	0
0	1	0	1/8	0
0	1	1	1/8	1/4
1	0	0	1/8	0
1	0	1	1/8	1/4
1	1	0	1/8	1/4
1	1	1	1/8	0

It is easy to see that P is uniform, totally independent.

Observe: Q is pairwise independent. It is uniform on the following subset of size four:

x_0	x_1	x_2	$Pr[Q]$
0	0	0	1/4
0	1	1	1/4
1	0	1	1/4
1	1	0	1/4

Therefore only two bits are needed to pick a triple (x_1, x_2, x_3) according to distribution Q (whereas three bits are needed to pick a triple (x_1, x_2, x_3) according to P).

Question: How many random bits are needed to create N pairwise independent bits?

2.2 Generating pairwise independent bits:

The algorithm to generate pairwise independent *bits*:

1. Choose k truly random bits $b_1 \dots b_k$.
2. $\forall S \subseteq [k]$ (where $[k]$ is set $1 \dots k$), $S \neq \emptyset$, set $C_S = \bigoplus_{i \in S} b_i$.

   $C_{\{1\}} = b_1$, $C_{\{1,2\}} = b_1 \oplus b_2$, ...
3. Output all C_S.

The algorithm generates $2^k - 1$ random bits $b_1 \dots b_k$.

Answer: To get N pairwise independent random bits only $K = O(\log N)$ truly random bits are required.

Claim 1 *For $S \neq T$, C_S and C_T are pairwise independent.*

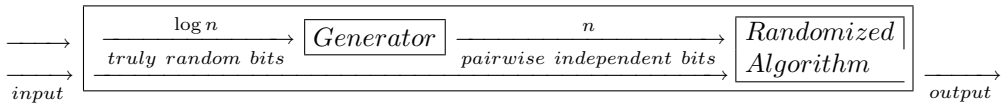
2.2.1 Derandomize Max-Cut

Derandomization here means using as little randomness as possible.

The algorithm to derandomize Max-Cut:

1. choose $\log N + 1$ truly random bits $b_1 \dots b_{\log N+1}$
2. generate $r_1 \dots r_n$ pairwise independent bits from b_i's
3. use $r_1 \dots r_n$ directly in the Max-Cut algorithm and evaluate the cut size

Repeat steps 1-3 on all possible choices of $b_1 \dots b_{\log N+1}$, and output the biggest Cut that was found.



Comments:

- This technique uses derandomization via enumeration.
- The running time of the randomized algorithm is N^2 (same as deterministically running Max-Cut N times), and the Cut size expectancy is no different.
- There are other deterministic algorithms which are more efficient than the derandomization of the randomized algorithm, but this is a demonstration of an interesting technique.

Remark: Pairwise and k-wise independence has been used in parallel algorithms.

2.3 Generating pairwise independent numbers

Now we will show how to generate random numbers (as opposed to bits), which are pairwise independent. But first, a definition:

Definition: $H = \{h : [N] \to [M]\}$ is a *"pairwise independent family of functions"*, if $\forall\ x \neq y \in [N]$ and $\forall\ c, d \in [M]$, $Pr[h(x) = c \wedge h(y) = d] = \frac{1}{M^2}$

This definition is equivalent to the definition of pairwise independent we mentioned before. x and y are the indices, and the probability for every two values for x and y is $\frac{1}{M^2}$. There is no pairwise independent function, but a class of functions which is pairwise independent, and from which a function is randomly chosen.

Example: We "pick" *hash-function*:
pairwise independent bits: $\left\{f_b |\ f_b(S) = \bigoplus_{i \in S} b_i \text{ for } S \neq \emptyset\right\}$
pairwise independent numbers (see below): $\left\{f_{a,b} |\ f_{a,b}(i) = (a \cdot i + b) \bmod q\right\}$

2.3.1 Constructing pairwise independent numbers over prime fields

The algorithm to generate pairwise independent *numbers*: on $(Z_q \equiv [0...(q-1)])$

1. Pick $a,\ b\ \in_R\ Z_q$ (uses $2 \cdot log(q)$ truly random bits).
2. $r_i = (a \cdot i\ +\ b) \bmod q$, $\forall\ i \in [q]$.
3. Output $r_1\ \ldots\ r_q$ (q pairwise independent numbers).

Claim 2 *The algorithm outputs q different numbers, which are pairwise independent.*

Proof Fix $i \neq j, c, d$. Examine the probability: $Pr_{a,b}[ai + b = c \wedge aj + b = d]$. This probability is over the hash function (a and b define the function).

This probability is equal to: $Pr_{a,b}\left[\begin{pmatrix} i & 1 \\ j & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix}\right] = \frac{1}{q^2}$. The determinant of the matrix is $i - j$ and since $i \neq j$, its determinant does not equal 0. So, there exists one unique solution to the set of equations for every (c, d). So, every pair c, d are mapped to by i, j with the same probability. In other words, if you just look at two indices i, j of the output, they will be independent and uniformly distributed. ■

Remark: It is possible to prove k-wise independent in the same way, using polynomials instead of linear functions. This is based on the fact that a Vandermonde matrix has a determinant which is different from 0. For 3-wise independence, the construction of the numbers uses $a \cdot i^2 + b \cdot i + c$, for truly random a, b, c.

2.3.2 Constructing pairwise independent n-bit numbers

We also note that there exists an algorithm for picking up pairwise independent numbers from a finite field of size 2^n. We can view the elements of $GF(2^n)$ as polynomials with $\{0, 1\}$ coefficients of degree $(n-1)$ (mod 2): $\sum_{i=0}^{n-1} a_i \cdot x^i (mod 2)$ s.t. $a_i, x \in \{0, 1\}$.

In this case $a \cdot i + b$ belongs to $GF(2^n)$ and the operations + and $\cdot$ (addition and multiplication) are over $GF(2^n)$. The hash function is $h_{a,b}(x) = a \cdot x + b$. The difference is that the number is picked from a field of size 2^n instead of some prime q. This is a family of pairwise independent functions.

Claim 3 $\exists$ *family of pairwise independent hash functions from* $\{0,1\}^n \to \{0,1\}^n$.

3 Randomness-efficient error reduction in sampling

If the algorithm needs only pairwise independent numbers, then it is enough to have $logN$ truly random bits. This way we can save in truly random bits used.

We introduce another algorithm, which uses pairwise independent random numbers, and we show how to lower its dependency on truly random numbers. In this example we show a one-sided error algorithm.

Notation: $A(x, R)$ = output of algorithm A on input x and random string R.

Given A with one-sided error:

- if $x \in L$ $Pr_R[A(x, R) = 0] < \frac{1}{2}$
- if $x \notin L$ $Pr_R[A(x, R) = 0] = 1$

Definition: A *witness* is R, such that $A(x, R) = 1$.

3.1 Error probability reduction

We can reduce the error probability by repetition: Repeat k times and output $'x \in L'$ if see any 1. Then we have the following behavior:

- if $x \in L$ $Pr[A^k(x, R^k) = 0] < \frac{1}{2^k}$
- if $x \notin L$ $Pr[A^k(x, R^k) = 0] = 1$

How many random bits are used? $|R| \cdot k$

Can we save some random bits? We can choose R more efficiently by choosing R's which are pairwise independent. The error probability is almost the same. The result and analysis are taken from a paper by Chor and Goldreich called: 'On the Power of Two-Point Based Sampling'. The number of random bits used by the Chor Goldreich method is $O(|R| + k)$. Note that the runtime is not better, just the number of random bits used.

Use pairwise independence to save random bits.

- Pick $h : \{0,1\}^m \to \{0,1\}^m$ randomly from pairwise independent family H (we showed we can construct such a family)
- Construct $R_1 \dots R_q$ where $R_i = h(i)$, where $i \in \{0,1\}^m$
- Compute $A(x, R_1), \dots, A(x, R_q)$. If see i such that $A(x, R_i) = 1$ output '$x \in L$', else '$x \notin L$'

When does the algorithm misclassify?

- if $x \notin L$ never, since the algorithm always outputs 0.
- if $x \in L$, the algorithm misclassifies if it never sees an R such that $A(x, R) = 1$.

If we look at all the possible R's, meaning all the binary strings of length m, then at least half of them make A output 1. In this case we missed each one of them.

Definition: $Y \equiv \sum_{i=1}^{q} A(x, R_i)$ (q here is 2^m).

Error reduction.

- If the sum is 0, we output 0.
- If positive, we output 1.

Error reduction analysis: $E[Y] = \sum_{i=1}^{q} E[A(x, R_i)] = \sum_{i=1}^{q} Pr[A(x, R_i) = 1] \geq \sum_{i=1}^{q} \frac{1}{2} = \frac{q}{2}$

Observe: $A(x, R)$ is an indicator variable. It equals 1 when the algorithm outputs 1 for the specific input of (x, R). Thus its expectancy is equal to the probability it is equal to 1. Since each R is uniformly distributed (1-wise independent) we get that the probability is as before, bigger then $\frac{1}{2}$.

Now we need to calculate what is the probability that $Y = 0$. In order to give a bound on the probability we use Chebyshev's inequality.

An important note: When the random variables are pairwise independent we cannot use Chernoff bound. Chernoff is good only in the case where the variables are totally independent. Chebyshev is suitable for pairwise independent variables.

Chebyshev's Inequality: $Pr(|X - \mu| \geq \epsilon) \leq \frac{Var[X]}{\epsilon^w}$
X is a random variable with expectancy $E[X] = \mu$, and the inequality gives the probability that X is far from its expectancy.

But Y is a sum of random variables, so we can use the Pairwise Independent Tail Inequality:

Pairwise Independent Tail Inequality

$$\left.\begin{array}{ll} \text{a)} & \text{Pairwise independent random variables } x_1 \dots x_t \text{ in [0,1]} \\ \text{b)} & \text{The average } X = \frac{1}{t} \cdot \sum_{i=1}^{t} x_i \\ \text{c)} & \text{The average expectancy } \mu = E[X] = \frac{1}{t} \cdot \sum E[x_i] \end{array}\right\} \Rightarrow Pr[|X - \mu| \geq \epsilon] \leq \frac{1}{t\epsilon^2}$$

We see that the dependency is linear in t, while in Chernoff we get an exponential dependency (which is tighter). But again Chernoff can be used only for totally independent variables.

Error reduction analysis continued: We look at the average value $\frac{Y}{q}$.
$\mu = E[\frac{Y}{q}] = \frac{E[Y]}{q} \geq \frac{q/2}{q} = \frac{1}{2} \Rightarrow Pr[Y = 0] \leq^{1} Pr[|\frac{Y}{q} - \mu| \geq \mu] \leq \frac{1}{q \cdot \mu^2} \leq^{2} \frac{1}{q} \cdot 4 = \frac{4}{q} = O(q^{-1})$

[1] Derives from the fact the $Y = 0$ is the only time that the output is the wrong answer.

[1⋆] This event is included in the event that $\frac{Y}{q}$ is far from the expectancy by at least μ.

[2] Regarding μ we proved that $\mu \geq \frac{1}{2}$ and thus $\frac{1}{\mu^2} \leq 4$.