Lecture 7

Max Cut:
randomized algorithm
derandomizing via pairwise independence
Generating pairwise independent bits
Pairwise Independence & Derandomization

Let's start with a simple algorithm for MaxCut:

Max Cut:

- given: $G = (V, E)$
- output: partition $V$ into $S, T$ to maximize $\left\{ \frac{1}{2} \sum_{(u,v) \in E, u \in S, v \in T} l(u,v) \right\}$
  
  Size of $S$-$T$ cut

Randomized Algorithm:

- Flip coins $r_1 \ldots r_n$
- Put node $i$ on side $r_i$ to get $S, T$
  - if $r_i = 0$ add $i$ to $S$
  - else $r_i = 1$ add $i$ to $T$
Analysis:

Let \( 1_{u,v} = \begin{cases} 1 & \text{if } r_u \neq r_v \\ 0 & \text{o.w.} \end{cases} \)

Cut size = \( \sum_{(u,v) \in E} 1_{u,v} \)

\[ E[\text{cut size}] = \mathbb{E} \left[ \sum_{(u,v) \in E} 1_{u,v} \right] \]

\[ = \sum_{(u,v) \in E} \mathbb{E} [1_{u,v}] = \sum_{(u,v) \in E} \mathbb{P} [1_{u,v} = 1] \]

\[ = \sum_{(u,v) \in E} \mathbb{P} [(r_u = 1 \lor r_v = 0) \lor (r_u = 0 \land r_v = 1)] \]

\[ = |E| - \left[ \frac{1}{q} + \frac{1}{q} \right] = \frac{|E|}{2} \]

So expect \( \frac{1}{2} \) the edges to cross cut!

Note: \( E[\text{cut size}] = \frac{|E|}{2} \Rightarrow \exists \text{ cut of size } \frac{|E|}{2} \)

average cut size produced by algorithm must be one that is at least as big as the average
Why is $\frac{|E|}{2}$ considered a success?

Oh, right...

The best you can do is $|E|$

Gives multiplicative approximation to within a factor of 2
Derandomization via Enumeration

Given: probabilistic algorithm $A$ on input $x$

Algorithm:
Run $A$ on every possible random string of length $r(n)$

Output majority answer
Randomized Max Cut Algorithm:

Flip coins \( r_1, \ldots, r_n \)
Put node \( i \) on side \( r_i \) to get \( S, T \)

Derandomization: first attempt

use “derandomization via enumeration”

Run \( A \) on every possible random string of length \( r(n) \)
Output majority answer

here \( r(n) = n \), so need \( 2^n \) runs of \( A \)

Hope: reduce \( r(n) \)?

still use “derandomization via enumeration”

find subset \( S \subseteq E \cdot S^{r(n)} \) of random strings that “works”
only enumerate over \( S \)
Pairwise Independent Random Variables

Given domain $T$ s.t. $|T| = t$
Pick $n$ values $x_1 \ldots x_n$ s.t. each $x_i \in T$

def $x_1 \ldots x_n$ independent if $\forall \ b_1, \ldots, b_n \in T^n$
$\Pr \left[ x_1 \ldots x_n = b_1 \ldots b_n \right] = \frac{1}{t^n}$
pairwise independent if $\forall \ 1 \neq j \ b_i, b_j \in T^2$
$\Pr \left[ x_i x_j = b_i b_j \right] = \frac{1}{t^2}$
$k$-wise independent if $\forall$ distinct $i_1, \ldots, i_k$
$\ b_1 \ldots b_k \in T^k$
$\Pr \left[ x_{i_1} \ldots x_{i_k} = b_1 \ldots b_k \right] = \frac{1}{t^k}$
Example:

**Total independence** $S_0 \cong 3$

<table>
<thead>
<tr>
<th>Bits</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

If pick "row" setting of $r_i$'s, the $r_i$'s are totally uniform.

$S$ is not independent:

$Pr_{row} \left[ r_1 r_2 r_3 = 110 \right] = \frac{1}{8}$

$S$ is pairwise independent:

$Pr_{row} \left[ r_1 r_3 = 11 \right] = Pr_{row} \left[ r_i r_j = b_i b_j \right] = \frac{1}{4}$

works for any $r_i r_j = b_1 b_2$ for $i, j \in \{1, 2, 3\}$ and $b_1, b_2 \in \{0, 1\}$.

**Pairwise independence** $S \leq 3$

<table>
<thead>
<tr>
<th>Bits</th>
<th>$r_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

If pick uniform setting "row" of $r_i$'s, the $r_i$'s are pairwise independent.

Pick from these 4 rows using only 2 random bits.

$Pr_{row} \left[ r_1 r_2 r_3 = 110 \right] = \frac{1}{4}$

$Pr_{row} \left[ r_1 r_3 = 11 \right] = Pr_{row} \left[ r_i r_j = b_i b_j \right] = \frac{1}{4}$

works for any $r_i r_j = b_1 b_2$ for $i, j \in \{1, 2, 3\}$ and $b_1, b_2 \in \{0, 1\}$.
Main points:

• Picking from fewer rows requires fewer random bits, so need to do fewer calls to derandomize. Doing 
  derandomization via enumeration is faster.

• Picking from smaller "subset" of rows has some of the same properties as picking from all rows.
  Only need pairwise independence in max cut algorithm.

  \[ \Rightarrow \text{Analysis of expectation still same if only use pairwise independent random bits.} \]

  Let's check this:

  \[
  E[\text{cut size}] = \sum_{(u,v) \in E} E[1_{u,v}] = \sum_{(u,v) \in E} \Pr[1_{u,v} = 1]
  \]

  Pairwise indep. enough for

  \[
  \Pr[r_u = 1 + r_v = 0] = \Pr[r_u = 0 + r_v = 1] \Rightarrow \Pr[r_u = 1 + r_v = 0] = \frac{1}{N^2} \]

  \[ \Rightarrow \text{Can do "derandomization via enumeration" on pairwise independent bits.} \]

  \[
  \text{How much faster?}
  \]
Example using our 3 pairwise indep bits:

Max cut:

\[ |E| = 2 \]

so looking for

Max cut of size \[ \frac{|E|}{2} = 1 \]

All cuts:

Value = 0:
\[
\begin{align*}
\gamma_1 = \gamma_2 = \gamma_3 &= 0 \\
\gamma_1 &= 0, \gamma_2 = 1, \gamma_3 &= 0 \\
\gamma_1 &= 1, \gamma_2 = 0, \gamma_3 &= 0
\end{align*}
\]

Value = 1:
\[
\begin{align*}
\gamma_1 = \gamma_2 = 0, & \gamma_3 = 1 \\
\gamma_1 &= 1, \gamma_2 = 0, \gamma_3 &= 1 \\
\gamma_1 &= 0, \gamma_2 = 1, \gamma_3 &= 0
\end{align*}
\]

Value = 2:
\[
\begin{align*}
\gamma_1 &= 0, \gamma_2 = 1, \gamma_3 &= 0 \\
\gamma_1 &= 1, \gamma_2 = 0, \gamma_3 &= 1
\end{align*}
\]

Pairwise indep cuts:
\[
\begin{align*}
\gamma_1 = \gamma_2 = \gamma_3 &= 0 \\
\gamma_1 &= 1, \gamma_2 = 0, \gamma_3 &= 1 \\
\gamma_1 &= 0, \gamma_2 = 1, \gamma_3 &= 0
\end{align*}
\]

Average value = 1 \implies \exists \text{ cut of value } \geq 1
Another picture:

- \( b_1 \ldots b_m \) are totally independent and enumerated all \( 2^m \) choices of \( b_i \)s.

- A "randomness generator" picks random row in \( S \) using \( b_i \)s.

- Output \( n \) pairwise independent random bits.

<table>
<thead>
<tr>
<th>Pick ( b_i b_2 )</th>
<th>Output ( r_1 r_2 r_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0 0 0</td>
</tr>
<tr>
<td>01</td>
<td>0 1 1</td>
</tr>
<tr>
<td>10</td>
<td>1 0 1</td>
</tr>
<tr>
<td>11</td>
<td>1 1 0</td>
</tr>
</tbody>
</table>

Table lookup; e.g., if pick \( b_i b_2 = 01 \) output \( r_1 r_2 r_3 = 011 \).

**Question**: What do you do when you need \( >3 \) p.i. random bits? (to be answered soon)
Derandomize Max Cut

**New Max Cut algorithm MC':**

Using \( m \ll n \) random bits, do derandomization via enumeration on this in \( O(2^m) \) calls to \( MC' \) (one for each setting of \( b_1...b_m \))

**In words:**

1. Construct \( MC' \): given \( m \) random bits \( b_1...b_m \) for graph \( G \)
   
   **procedure:**
   
   generates \( r_1...r_n \) (pairwise indep) from \( b_1...b_m \)
   
   use \( r_i 's \) to run Max Cut & evaluate cut-size

2. Derandomize \( MC' \):

   for all choices of \( b_1...b_m \)
   
   run \( MC' \) on \( b_1...b_m \) & \( G \) & evaluate cut-size
   
   pick best cut-size
try all \( b_1 \ldots b_m \)

\[
\text{randomness generator} \rightarrow \Gamma_1 \ldots \Gamma_n \rightarrow \text{Original Max Cut Algorithm} \rightarrow \text{Answer}
\]

\text{derandomized MC'}

**Runtime:** \( 2^m \times (\text{time for generator} + \text{time for MC}) \)

will show \( m = O(\log n) \) + \text{time for generator} \( \text{poly}(n) \)

so total time is \( \text{poly}(n) \)

**Comments:**

if derandomize MC by enumeration, you end up trying all cuts \( \Rightarrow \) get \( \text{OPT} \)

here, we are trying very few cuts

so no guarantee of getting \( \text{OPT}. \ \text{just} \frac{\text{OPT}}{2} \)
Generating Pairwise Independent Random Variables:

1. Bits

   - choose $k$ truly random bits $b_1 \ldots b_k$

     $\forall S \subseteq [k] \text { s.t. } S \neq \emptyset$

     set $C_S = \bigoplus_{i \in S} b_i$

   - output all $C_S$

   K truly random bits $\rightarrow 2^k-1$ p.i. bits

   $\log n \text { to } n-1$

   proof of correctness:

   upcoming homework
2. Integers in $[0, \ldots, q-1]$ $q$ prime

1st idea: if $q < 2^l$ can be represented via $l$ bits
repeat "bits" construction independently for each position $q_i$ in $1..l$

uses $O(\log n \cdot \log q)$ bits of true randomness

Slightly better idea: $O(\log q)$ bits of randomness

- Pick $a, b \in \mathbb{Z}_q$
- $r_i \leftarrow a \cdot i + b \mod q$ $\forall i \in \{0, \ldots, q-1\}$
- output $r_1 \ldots r_q$
Useful to think of construction as input/output description of a function:

$$h_{a,b} : \{0, \ldots, q-1\} \rightarrow \mathbb{Z}_q$$

& family of fetsns $\mathcal{H} = \{ h_{a,b} \mid a, b \in \mathbb{Z}_q \}$

Family of fetsns $\mathcal{H} = \{ h_1, h_2, \ldots \}$

for $h_i : [N] \rightarrow [M]$ is "pairwise independent" if

when $h \in \mathcal{H}$

(1) $\forall x \in [N], \ h(x) \in \mathbb{Z}_m \mathbb{[M]}$

(2) $\forall x_1 \neq x_2 \in [N], \ (h_1(x_1), h_2(x_2)) \in \mathbb{Z}_m \mathbb{[M]}^2$

any locn x is mapped uniformly

any pair of locns $x_1 \neq x_2$ mapped uniformly independently
equivalently:

\[
\forall x_1 \neq x_2 \in [N] \\
\forall y_1, y_2 \in [M] \\
\Pr[h(x_1) = y_1, \ \& \ h(x_2) = y_2] = \frac{1}{M^2}
\]

Comments:

- no single fctn is p.i. on its own - need to pick a random fctn from a collection of fctns.

- given \( h \neq x \in [N] \), \( h(x) \) should be computable in time \( \text{poly}(N, \log M) \)

- also called "strongly 2-universal hash fctns"
Why is our example p.i.?

Our family:
\[ \mathcal{H} = \{ h_{a,b} \mid \mathbb{Z}_q \to \mathbb{Z}_q \} \]
Recall \( q \) is prime

\[ h_{a,b}(x) = ax + b \mod q \]

Proof of p.i.:

\[ \forall x \neq w, \ c, d \]

\[ \Pr_{a,b \leftarrow \mathcal{H}}\left[ ax + b = c \quad \land \quad aw + b = d \right] = \frac{1}{q^2} \]

\[ \begin{pmatrix} x \\ 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix} \]

\[ w \neq x \quad \text{so det}(A) \neq 0 \quad \Rightarrow \quad \text{unique soln} \]

How many truly random bits?

Pick \( a, b \) uniformly, needs \( 2\log q \)

random bits
More comments:

Can construct for all finite fields, even when domain & range have different sizes.

Original motivation: hashing
choose hash ftns from p.i. family instead of totally random ftns.

Why?
how to store random ftn?
need |domain| \cdot \log |range| bits to write down input/output table

how to store p.i., hash ftn?
write down $\alpha \cdot b$