Randomization + Derandomization?
Today

- Randomized complexity classes
- Derandomization via enumeration
  \[ \text{BPP} \subseteq \text{EXP} \]
- Pairwise independence & derandomization
  Max Cut Algorithm
  Defn. of p.i.
  Derandomizing max cut
Some Complexity Classes:

**def. a language** \( L \) is a subset of \( \Sigma^* \)

\[\exists X \mid X \text{ is a graph with a hamilton path}\]

\[\exists X \mid X \text{ is a collection of sets that have a proper 2-coloring}\]

**def** \( P \) is class of languages \( L \)

with ptime deterministic algorithms \( A \)

\[\text{st. } x \in L \Rightarrow A(x) \text{ accepts}\]

\[x \notin L \Rightarrow A(x) \text{ rejects}\]

**def** \( RP \) is class of languages \( L \)

with ptime probabilistic algorithm \( A \)

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

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

**def** \( BPP \) is class of languages \( L \)

with ptime probabilistic algorithm \( A \)

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

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

- Constants arbitrary -
  with mult. cost of \( O(\log \frac{1}{\beta}) \) can get error \( \leq \beta \)

- Clearly \( P \leq \text{RP} \leq \text{BPP} \)

Big Open Question:

is \( P = \text{BPP} \)?

do we need random coins for efficient algorithms?

Derandomization via enumeration

- Given probabilistic algorithm \( A \) on input \( x \)
- Run \( A \) on every possible random string of length \( r(n) \)
- output majority answer

at most time bound of \( A \). Is there a better bound?
Behavior
if $x \in L$, $\geq \frac{2}{3}$ of random strings cause $A$ to accept $\Rightarrow$ majority answer is ACCEPT
if $x \notin L$ reject $\Rightarrow$ REJECT

runtime
$$O(2^{r(n)} \cdot t(n)) \leq O(2^{t(n)})$$
time bound of $A$

Corollary
$BPP \subseteq \text{EXP}$
$\uparrow$
$\text{EXP} = \text{DTIME} (\cup 2^{n^c})$

Comments:
$r(n) \leq t(n)$ since can use at most 1 bit per step
if can get better bound on $r(n)$, can improve runtime

E.g., if $r(n) = O(\log n)$, runtime is $\text{poly}(n)$ for ptime $A$
Given a problem with a randomized ptime algorithm, 1-sided error

Homework problem 3

⇒ ∃ one random string that works for all inputs of size n

i.e. ∃ ckt (with no random bits) that work for all inputs of size n.

• What about 2-sided error?

    also true!
Pairwise independence & derandomization

- A simple randomized algorithm for MaxCut
- Pairwise independent sample spaces
- Derandomization

**Max Cut:**

given: \( G = (V, E) \)
output: partition \( V \) into \( S, T \) to maximize
\[
\sum_{(u,v) \in E} \delta(u,v) |u \in S, v \in T|
\]

Size of \( S, T \) cut

**A randomized algorithm:**

Flip \( n \) coins \( r_1, \ldots, r_n \)
Put vertex \( i \) on side \( S \) to get \( S, T \) \( \iff \) add \( i \) to \( S \) if \( r_i = 0 \) and to \( T \) o.w.

**Analysis:**

Let \( 1_{u,v} = 1 \) if \( r_u \neq r_v \) (i.e., placed on different sides so \((u,v)\) crosses \( S \), \( T \))

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

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

\[
= \sum_{(u,v) \in E} \Pr[(r_u = 1 + r_v = 0) \text{ or } (r_u = 0 + r_v = 1)]
\]

\[
= \sum_{(u,v) \in E} \left( \Pr[r_u = 1 r_v = 0] + \Pr[r_u = 0 r_v = 1] \right) = \frac{|E|}{2}\]
Pairwise independent random variables: definition

Pick \( n \) values \( X_1 \ldots X_n \)

each \( X_i \in \mathcal{T} \) (domain) s.t. \( |\mathcal{T}| = t \) (size of domain)

in some way.

def. \( X_1 \ldots X_n \) independent if \( \forall b_1 \ldots b_n \in \mathcal{T}^n \)

\[ \Pr[X_1 \ldots X_n = b_1 \ldots b_n] = \frac{1}{t^n} \]

pairwise independent if \( \forall i \neq j \) \( b_i, b_j \in \mathcal{T}^2 \)

\[ \Pr[X_i X_j = b_i b_j] = \frac{1}{t^2} \]

\( k \)-wise independent if \( \forall \) \( b_1 \ldots b_k \in \mathcal{T}^k \)

\[ \Pr[X_1 \ldots X_k = b_1 \ldots b_k] = \frac{1}{t^k} \]

Main point:

Only use pairwise independence in max-cut algorithm

(i.e., algorithm analysis still works if random bits are only pairwise indep).
Derandomization of max-cut

Full enumeration:
- $n$ fully random bits $\rightarrow$ Algorithm $\rightarrow$ cut
- try all $2^n$ possible coin tosses
- pick best cut

"Partial enumeration":
- $m$ pairwise independent random bits $\rightarrow$ Algorithm $\rightarrow$ cut
- don't try all possible coin tosses
- just a subset that satisfies pairwise independence

E.g., $r_1, r_2, r_3$
- pick a row uniformly
- $\begin{cases} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{cases}$

For $i \neq j$, $\forall b_1, b_2 \in \{0, 1\}$:
- $Pr[r_i = b_1 \wedge r_j = b_2] = \frac{1}{4}$
- Good enough to give
- $E[\text{cut}] = \frac{|E|}{2}$

For 3 node graphs, only need to enumerate over 4 rows instead of 8 rows.

Another picture

- $b_1 \ldots b_m$ totally independent
- Enumerate all $2^m$ choices

"Randomness generator"
- Pick a random row

$\begin{cases} r_1 \ldots r_m \end{cases}$
- Pairwise independent + good enough for our algorithm!
- Can we make $n > m$?
derandomize Max-Cut is given "randomness generator" taking \( (\log n + 1) \) bits

- First: construct new randomized MC alg MC'.
  - Given \( \log n \) truly random bits \( b_0 \ldots b_{\log n + 1} \)
  - Use generator to construct \( n \) p.i. random bits \( r_0 \ldots r_n \)
  - Use \( r_i \)'s in MC alg & evaluate cutsize

Then i derandomize via enumeration

Deterministic MC alg:

For all choices of \( b_0 \ldots b_{\log n + 1} \)

run MC' on \( b_0 \ldots b_{\log n + 1} \) & evaluate cutsize

pick best cutsize

Runtime: \( (2^{\log n}) \times (\text{time for generator + time to run MC}) = \text{poly}(n) \)

\( \# \) choices of \( b_i \)'s

Comments
- no guarantee of getting OPT cut as in basic enumeration method
- generator determines a very small set of random strings, at least one of which gives a good cut
New Max Cut Algorithm $MC'$ (using $m$ more bits)

do "full enumeration" derandomization on this in $O(2^m) \times [\text{time to generate} + \text{time to run MaxCut}]$
How to generate pairwise independent random variables?

1) Bits

- choose \( k \) truly random bits \( b_1, \ldots, b_k \)

\[ \forall s \subseteq [k] \text{ s.t. } s \neq \emptyset \text{ set } c_s = \bigoplus_{i \in s} b_i \]

- output all \( c_s \)

Generates \( 2^k - 1 \) bits from \( k \) truly random bits

i.e. \( m = \log n \)

Generated bits are pairwise independent

proof: exercise

2) Integers in \([0, \ldots, q-1]\) (\( q \) prime)

trivial method that works for \( q = 2^l \) (note that \( q \) is not prime)

- repeat "bits" construction independently for each position in \( 1 \ldots l \)

uses \( O(\log n \cdot \log q) = O(\log n) \) bits of true randomness
Somewhat better construction:

(when \( n \neq q \) needs \( O(\log q) \) bits of randomness)

- pick \( a, b \in \mathbb{Z}_q \)
  
- \( r_i \leftarrow a \cdot i + b \mod q \quad \forall i \in [0..q] \)

- output \( r_0 \ldots r_q \)

Useful to think of as an input/output description of a

\[ h_{a, b} : [0..q] \to \mathbb{Z}_q \]

Family of functions \( H = \{ h_{a_1, b_1}, \ldots \} \) for \( h_i : [N] \to [M] \) is

"pairwise independent" if:

\[ H \in \mathcal{D} \]

1. \( \forall x \in [N], \ H(x) \in [M] \)

2. \( \forall x_1 \neq x_2 \in [N], \ H(x_1) \perp H(x_2) \) independent

Equivalently:

\( \forall x_1 \neq x_2 \in [N] \)

\[ \forall y_1, y_2 \in [M] \]

\[ \Pr_{H \in \mathcal{D}} \left[ \sum_{H(x_i) = y_1} H(x_2) = y_2 \right] = \frac{1}{M^2} \]
Comments

- no single hash is p.i. - have to pick a random hash from a family

- given $H \times x \in \mathcal{N}$, $H(x)$ should be computable in time poly($\log N, \log M$)

- also called "strongly $2$-universal hash fncts"

Why is our example p.i.?

$H = \{ h_{a, b} \mid Z_q \rightarrow Z_q \}$

(recall $q$ is prime)

$h_{a, b} = a \times b \mod q$

fix any $x \neq w$, $c, d$

$Pr_{a, b}[a x + b = c \land h_{a, b}(w) = d] = \frac{1}{q^2}$

$(x, 1) \cdot (b) = (c)$

$w \neq x$ so nonsingular $\exists$ unique soln

how many truly random bits?

$2 \log q$ yields $9$ p.i. random field elts.
More Comments

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

- Original motivation: hashing

  hash fets chosen from p.i. family instead of random fets.

  Why is this good?

  how would you store a random ftn on a domain of size $2^{100000000000000000000}$?