Randomization + Derandomization?
Some Complexity Classes:

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

- e.g. \( \{ x \mid x \text{ is a graph with a hamilton path} \} \)
- \( \{ x \mid x \text{ is a collection of sets that have a proper } 2\text{-coloring} \} \)

**def** \( \mathbf{P} \) is class of languages \( L \) with \( \text{ptime} \) deterministic algorithms of

\[
\begin{align*}
\text{st.} & \quad x \in L \Rightarrow A(x) \text{ accepts} \\
& \quad x \notin L \Rightarrow A(x) \text{ rejects}
\end{align*}
\]

**def** \( \mathbf{RP} \) is class of languages \( L \) with \( \text{ptime} \) probabilistic algorithm of

\[
\begin{align*}
\text{st.} & \quad x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq \frac{1}{2} \\
& \quad x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] = 0
\end{align*}
\]

**def** \( \mathbf{BPP} \) is class of languages \( L \) with \( \text{ptime} \) probabilistic algorithm of

\[
\begin{align*}
\text{st.} & \quad x \in L \Rightarrow \Pr[A(x) \text{ accepts}] \geq \frac{2}{3} \\
& \quad x \notin L \Rightarrow \Pr[A(x) \text{ accepts}] \leq \frac{1}{3}
\end{align*}
\]

\( \frac{1}{3} \)-sided error

\( \frac{1}{3} \)-sided error
Comments

- Constants arbitrary
- with mult. 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

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 $\text{Accept}$

if $x \notin L$ $\Rightarrow$ $\text{Reject}$

Runtime

$O(2^{r(n)} \cdot t(n)) \leq O(2^{t(n)})$

time bound of $A$

Corollary

$\text{BPP} = \text{EXP}$

$\uparrow$

$\text{EXP} = \text{DTIME} (\cup 2^{n^c})$

Comments:

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

$$\sum_{u \in S, v \in T} w(u, v)$$

Size of $S, T$ cut

A randomized algorithm:

Flip $n$ coins $r_1, \ldots, r_n$

Put vertex $i$ on side $S_i$ to get $S, T \leftarrow i.e., add \ i \ to \ S$ if $r_i = 0$

$+$ 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 cut)

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

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

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

$$= \sum_{(u, v) \in E} \left( \frac{\Pr[r_u = 1 + r_v = 0]}{4} + \frac{\Pr[r_u = 0 + r_v = 1]}{4} \right) = \frac{E}{4}$$
Pairwise independent random variables: definition

Pick $n$ values $X_1 \ldots X_n$
each $X_i \in \mathcal{E}$ (domain) s.t. $|\mathcal{E}| = t$ (size of domain) in some way.

**Definition:** $X_1 \ldots X_n$ independent if $\forall b_1 \ldots b_n \in \mathcal{E}^n$

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

pairwise independent if $\forall i \neq j$ $b_i, b_j \in \mathcal{E}^2$

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

$k$-wise independent if $\forall i_1 \ldots i_k$ $b_1 \ldots b_k \in \mathcal{E}^k$

$$\Pr[X_{i_1}, \ldots, X_{i_k} = b_1 \ldots b_k] = \frac{1}{t^k}$$

Main point:

Only use pairwise independence in max-cut algorithm.
Derandomization of max-cut

Full enumeration:

1. Try all $2^n$ possible coin tosses
2. Pick best cut
3. Gets very best cut, not just $\frac{|E|}{2}$

"Partial enumeration":

1. Don't try all possible coin tosses
2. Just a subset that satisfies pairwise independence

E.g., $r_1, r_2, r_3$

Pick a row uniformly:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For $i \neq j$, $\forall b_i, b_j \in \{0, 1\}$

\[
\Pr[r_i = b_i, r_j = b_j] = \frac{1}{4}
\]

Good enough to give

\[
E[\text{cut}] = \frac{|E|}{2}
\]

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

Pairwise independent and good enough for our algorithm!

Can we make $n > m$?
derandomize Max-Cut is given "randomness generator" taking \((\log n) = n\) bits

First: construct new randomized MC alg MC'.

* choose \(\log n\) truly random bits \(b_1...b_{\log n+1}\)
* use generator to construct \(n\) p.i. random bits \(r_1...r_n\)
* use \(r_i's\) in MC alg & evaluate cutsize

Then: derandomize via enumeration

Deterministic MC alg:

For all choices of \(b_1...b_{\log n+1}\)
run MC' on \(b_1...b_{\log n+1}\) & evaluate cutsize
pick best cutsize

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

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
How to generate pairwise independent random variables?

1) Bits
   
   * choose $k$ truly random bits $b_1 \ldots b_k$
   
   $\forall s \subseteq [k]$ st. $s \neq \emptyset$ set $c_s = \theta b_s$
   
   * 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 $q = 2^p$ (note that this is not prime)
   
   * repeat "bits" construction independently for each position in 1..l

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

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

* pick \( a, b \in \mathbb{Z}_q \)
* \( r_i = a \cdot i + b \mod q \quad \forall i \in \mathbb{Z}_q \)
* output \( r_1, \ldots, r_q \)

Useful to think of as \( \text{fctns from } \mathbb{Z}_q \to \mathbb{Z}_q \)

Family of \( \text{fctns } H = \{ h_1, h_2, \ldots \} \) for \( h_i : [N] \to [M] \) is "pairwise independent" if:

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

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

\[
\forall y_1, y_2 \in [M] \quad \Pr \left[ \left( H(x_1) = y_1 \land H(x_2) = y_2 \right) \bigg| H \in_u \mathcal{H} \right] = \frac{1}{M^2}
\]
Comments

- no single fn is p.i. - have to pick a random fn from a family
- given $H : x \in [N]$ $H(x)$ should be computable in time $\text{poly}(\log N, \log M)$
- also called "strongly 2-universal hash fnms"

Why is our example p.i.?

$H = \{ h_{a,b} \mid \mathbb{Z}_q \rightarrow \mathbb{Z}_q \}$

$h_{a,b} = a \cdot x + b \mod q$

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

$\Pr_{a,b} [ ax + b = c \land aw + b = d ] = \frac{1}{q^2}$

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

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

\[\text{how many truly random bits?}\]

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

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

- Original motivation: hashing
  hash ffnas chosen from p.i.i. family instead of random ffnas.

  Why is this good?

  how would you store a random ffn on a domain of size $2^{10000000000}$?