Pseudo-randomness in streaming

Piotr Indyk
MIT
Randomness

- Most of the algorithms we have seen maintain $A x$, where $A$ is “random”
- Where do the random bits come from?
- Pseudo-random generator (PRG) $F$: maps a short truly random bit string $r$ into longer string $F(r)$ that is “random enough”
- We will see:
  - $k$-wise independence
  - PRG for bounded space computation
k-wise independence
k-wise independence

• A family of (hash) functions $H$ containing functions $h: U \rightarrow T$ is $k$-wise independent if for any sequence $s=(s_1,..s_k)$ in $U^k$, and $t$ in $T^k$, we have
  \[
  \Pr_{h \in H}[ (h(s_1),..,h(s_k)) = t] = 1/|T|^k
  \]

• Simple but powerful notion
  – Can construct $H$ of size $<<|T|^{|U|}$, i.e., can generate random $h$ from $H$ using few random bits
  – Can show $k$-wise independence suffices, often for small $k$
k-wise independence: construction

- Consider families of functions $h: U \rightarrow T$ where $U=T=\{0\ldots p-1\}$, $p$ prime
- $H$ consists of all degree $k-1$ polynomials mod $p$, i.e.,
  $$h(x)=a_0+a_1x+a_2x^2+\ldots+a_{k-1}x^{k-1} \mod p$$
- Size: $|H|=p^k$, i.e., need $k \log p$ random bits
- Correctness:
  - Take any $s=(s_1,\ldots,s_k)$, $t=(t_1,\ldots,t_k)$
  - There is exactly one $g$ such that $g(s_i)=t_i$, $i=1\ldots k$
  - The probability of selecting $g$ is $1/|H| = 1/p^k$
Application I: AMS

• We needed to bound the expectation of

\[(\sum_i r_i x_i)(\sum_i r_i x_i)(\sum_i r_i x_i)(\sum_i r_i x_i)\]

where \(r_0…r_{n-1}\) are i.i.d. random variables uniform over \{-1,1\}

• Reduces to analyzing the expectation of individual terms

\[r_i x_i r_j x_j r_k x_k r_l x_l\]

• 4-wise independence suffices

• Need only \(4 \log n\) random bits, assuming \(n\) is prime
Caveat

• Issue:
  – $h(i)$ is a number in $\{0 \ldots n-1\}$
  – We need $r_i$ in $\{-1,1\}$ (or $\{0,1\}$)
  – What should we do?

• Solution I (hack):
  – Take $p >> n$
  – Define $r_i = h(i) \mod 2$
  – We have $Pr[r_i=1] = 1/2 + O(1/p)$
  – Re-calculate expectation and variance, show it is “good enough”

• Solution II: construct a txn binary matrix with every $k$ columns linearly independent (mod 2), $t$ small
  – The polynomial construction in the previous slide provides such a matrix over $\{0 \ldots p-1\}$
Application II: Count-Min

• Need, for \( i \neq j \)
  \[ \Pr[h(i)=h(j)]=1/w \]
• 2-wise independence suffices
• Bounded space computation:
  – A set of $2^S$ states $Q$, a subset of them “accepting”
  – A $2^S \times 2^S$ transition matrix $M$:
    • $M[i,j]$=probability of moving from state $i$ to state $j$
    • Assume that using $K=O(S)$ truly random bits we can select $j$ given $i$
  – Run the computation for $R$ steps starting from start using random bit sequence $u$, resulting in $Q^R(start,u)$
  – The distribution of $Q^R(start,u)$ is given by $M^R[start, \ast]$

• Interpretation:
  – $S$ bits of storage
  – Program+input define the transition matrix $M$
Example

• Compute

\[ Y = r_1 x_1 + \ldots + r_n x_n \]

where \( x_i \) in \{-M \ldots M\}, \( r_1, \ldots, r_n \) i.i.d. from \{-1,1\}

• Accept if \( Y^2 > T \)

• In our setup:
  – States: \((Y,i)\)
  – Transitions from \((Y,i)\):
    • To \((Y+x_i,i+1)\) with probability \( \frac{1}{2} \)
    • To \((Y-x_i,i+1)\) with probability \( \frac{1}{2} \)
    • To other states with probability 0
Theorem [Nisan’92]

There is a mapping $F: \{0,1\}^L \rightarrow [\{0,1\}^K]^R$, $L = O(K \log R)$ s. t.

$$\Pr_u[Q^R(\text{start},u) \text{ accepts}] = \Pr_v[Q^R(\text{start},F(v)) \text{ accepts}] \pm \varepsilon$$

where:

• $u$: uniform over $\{0,1\}^{KR}$
• $v$: uniform over $\{0,1\}^L$
• $\varepsilon = 1/c^S$ for some $c < 2$

Proof sketch:

• Select $T = \log R$ 2-wise independent functions $h_i: \{0,\ldots,R-1\} \rightarrow \{0,1\}^K$
• Build a tree
Nisan’s generator for linear sketching

Claim: consider

• A vector \( x \) in \( \{-M\ldots M\}^n \),
• A linear sketch
  \[ Y = r_1 x_1 + \ldots + r_n x_n \]
  where \( r_i = G_i(u_i) \) in \( \{-M\ldots M\} \) and \( u_i \)'s are i.i.d. random variables chosen uniformly from \( \{0,1\}^K \), \( K=O(\log M) \)
• A pseudorandom linear sketch
  \[ Y' = r'_1 x_1 + \ldots + r'_n x_n \]
  where \( r'_i = G_i(F(v)_i) \), \( v \) uniform over \( \{0,1\}^L \), \( L=O(\log n \ast \log(nM)) \)

Then
\[ \Pr[Y \text{ accepts}] = \Pr[Y' \text{ accepts}] \pm \varepsilon \]

Need \( O(\log n \ast \log(nM)) \) random bits for space \( O(\log(nM)) \)
Implications

• Any streaming algorithm that uses linear sketches by matrices with i.i.d. entries using space S can be simulated using $O(S \log (nM))$ randomness

• Applies to all algorithms that we have seen in this class so far, notably $L_p$ norm estimation for $p$ in $(0,2]$  

• For the latter problem one can do better though:
  – Show that one can generate the matrix entries using $k$-wise independent families for $k=O(1)$  
  – This shaves of the logarithmic factor  
A bonus “war story”

• Once upon a time (in 1999), we (A+T+P) used min-hashing to cluster a large set of web pages
  – Documents = sets of words
  – Cluster together pairs of similar documents

• Problem: the home page of T’s advisor got clustered with “certain websites”

• Problem II: our algorithm was provably correct – the probability of failure was $10^{-6}$ (we calculated it exactly, under some pseudo-randomness assumption)
What happened?

• Implementation:
  – We implemented $g(A) = \min_{x \in A} h(x)$ using
    $h(x) = (ax \mod P) \mod 2^8$
    • $P = 2^{64} - 59$ (more or less)
    • $a$ randomly chosen
  – To speed up the process, we kept only words $x$ which were divisible by 8

• What happened?
  – Implementation bug: $ax$ was computed modulo $2^{64}$
  – $\mod P$ had essentially no effect
  – $x$ divisible by 8 $\Rightarrow$ $(ax)$ divisible by 8 $\Rightarrow$ $(ax) \mod 2^8$ divisible by 8
  – 3 lowest bits of $h(x)$ were zero, so the actual range was $2^5$ not $2^8$
  – Enough for word collisions to occur…
Moral

- Do your hashing right, or you might never graduate …