Pseudorandom Objects and Generators

Journées ALEA 2012

Lecture 2: Pseudorandomness in Algorithms and Complexity

David Xiao

LIAFA

CNRS, Université Paris 7

Example: Polynomial Identity Testing

- Given multi-variate polynomial $p \in Z[x_1 \dots x_m]$, decide if $p \neq 0$
  - Ex. $p = (3x_1 - 4x_2)^7 (45x_1^3x_2 - 4x_1x_3^2 - x_1x_3)^2 + (4x_1^2x_2 - x_2^3x_3)^5$
  - Brute force takes exponential time in degree
- Randomized algorithm:
  - Let $d = \text{degree}(p)$
  - Pick $z_1 \dots z_m$ each randomly from $[q] = \{1, \dots, 100d\}$
  - Output 1 if $p(z_1, \dots, z_m) \neq 0$
  - Output 0 if $p(z_1, \dots, z_m) = 0$
- Clearly algorithm outputs 0 if $p \equiv 0$
- Theorem [Schwartz-Zippel'79]:
  if $p \neq 0$ then $\Pr_z[\, p(z_1 \dots z_m) = 0 \,] \leq d/100d = 1/100$
- We don't know how to derandomize!

Eliminating or Reducing Randomness

- Using randomness in algorithms raises questions:
  - ~~How to obtain randomness?~~

ask the physicists
(or the philosophers)

  - How to save on randomness?
  - How to purify non-uniform randomness?
  - Does randomness fundamentally accelerate computation?
- Pseudorandomness: use little or no randomness but behaves indistinguishable from random

Pseudorandomness in Algorithms

Randomness

- U_n = uniform distribution over $\{0, 1\}^n$
- Each string has same probability mass = $1/2^n$
- Can approximate other distributions: e.g. uniform over F_q, Gauss(0, 1), etc.

Using Randomness: Algorithms

- Problem: deciding language $L : \{0,1\}^* \to \{0, 1\}$
- Randomized algorithm A deciding L:
  - Take input x, random bits r drawn from U_m
  - Perform some precise deterministic operations (depending on x, r)
  - $\Pr_r[\ A(x; r) = L(x)\ ] \geq 2/3$ for all x
- Efficiency: perform at most n^c operations where $n = |x|$ ("polynomial time")
- Also measure number of bits used, i.e. $|r| = m$

$x \rightarrow$

r random $\rightarrow$

A

$\rightarrow$ 0 or 1

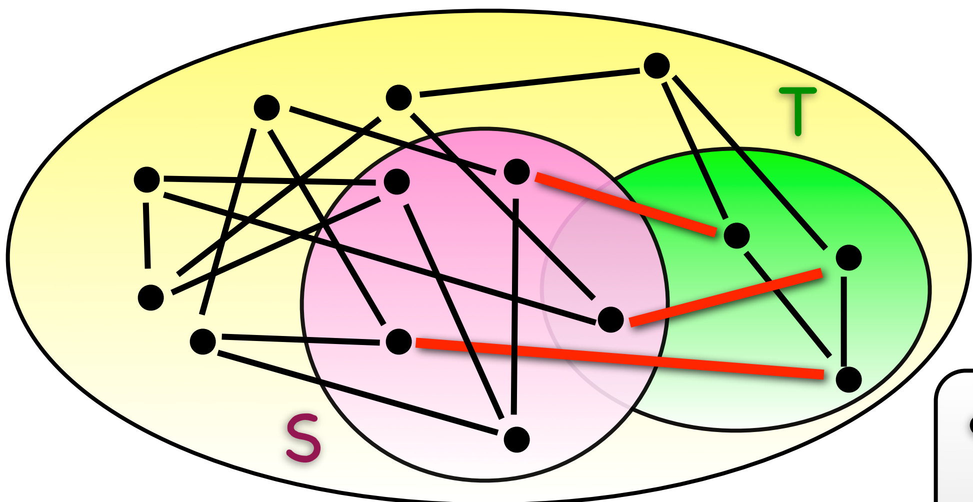
- Can reduce error by taking majority of running algorithm with independent randomness
- Analyze using uniform randomness

Randomness in Algorithms

- Treat random bits as expensive resource
- Example: error reduction
  - For all inputs x, $\Pr[\ A(x;\ U_m)\ \text{errs}\ ] \leq 1/3$
  - Chernoff-Hoeffding: majority of k independent repetitions of A has error $2^{-\Omega(k)}$
  - If each execution costs m random bits, k executions cost km random bits
  - Can we do better?

Expander graphs

- Recall from yesterday



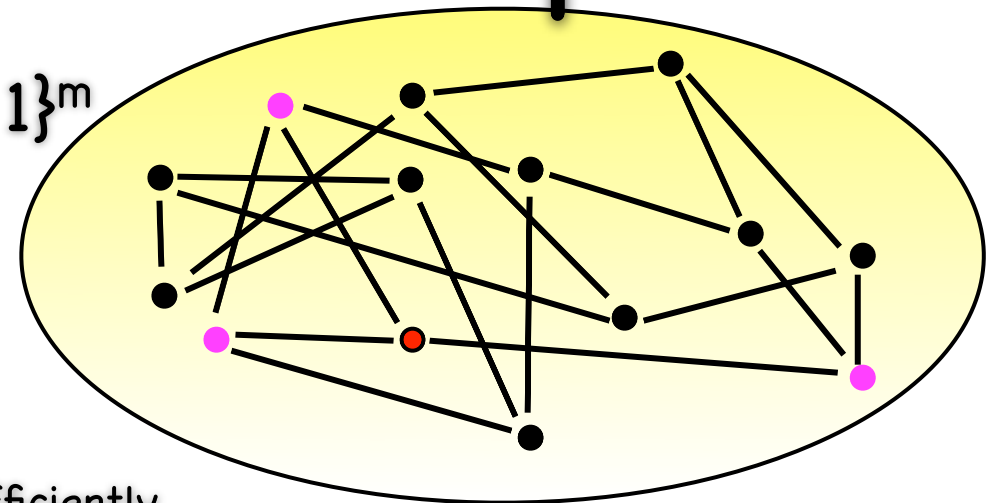
Spectral expander: G is (N, D, λ)-expander if:

- G is D-regular, $|V| = N$
- Let M = adjacency matrix of G
  - $M_{ij} = 1/D$ if $(i, j) \in G$, 0 else
  - Eigenvalues of M in $[-1, 1]$
  - Max eigenvalue = 1
- $\lambda \geq$ all other eigenvalues of M in absolute value

- Expander mixing lemma: For all $S, T \subseteq G$: $\big| |E(S, T)| - |S|\,|T|\,D/N \big| \leq \lambda D \sqrt{(|S|\,|T|)}$
- $E(S, T)$ = edges between S and T in G
- $|S|\,|T|\,D/N$ = expected # edges in random D-regular graph

Using Expander Graphs

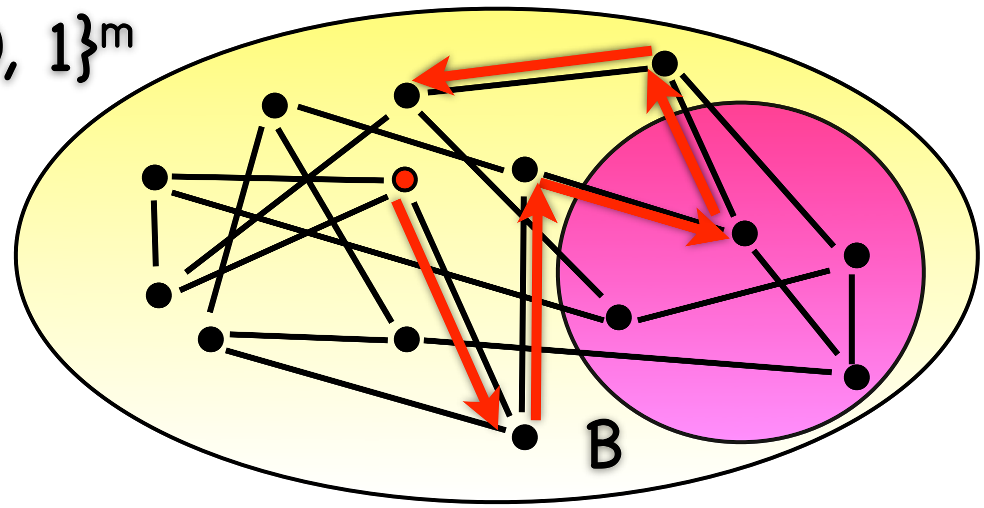
$\{0, 1\}^m$



- Theorem [Cohen-Wigderson'89]: can efficiently reduce error of A to $1/n^c$ without any additional randomness
  - Suppose we have $(2^m, D = \text{poly}(n), \lambda = 1/(12n^c))$ expander graph
  - Each vertex corresponds to string in $\{0, 1\}^m$
  - New algorithm:
    - Use m random bits to pick vertex r
    - In expander, calculate neighbors $\{r_1 \dots r_D\} = N(r)$
    - Output majority of $A(x; r_1) \dots A(x; r_D)$
  - Claim: new algorithm has error $1/n^c$

Proof...

Exponentially small error



- Use $O(1)$ constant expander graph
- Take random walk, let $r_1 \dots r_k$ be visited vertices
- Output majority of $A(x; r_1) \dots A(x; r_k)$
- Costs $m + O(k)$
- From Expander Chernoff Bound:
  $\Pr[\,\mathrm{Maj}(A(x; r_1) \dots A(x; r_k))\ \text{errs}\,] \le 2^{-(1-\lambda)\,k}$
  (Good expander => w.h.p. fraction of bad steps in walk $\le |B|/n = 1/3$)

Imperfect Randomness

- Analyze algorithms assuming uniform random bits
- Natural sources unlikely to be uniform:
  - Current time?
  - Mouse gestures?
  - Quantum phenomena?
- All have dependencies, noise, etc.
- How to purify?
  - Ad hoc: linear feedback shift registers
  - Better: randomness extractors

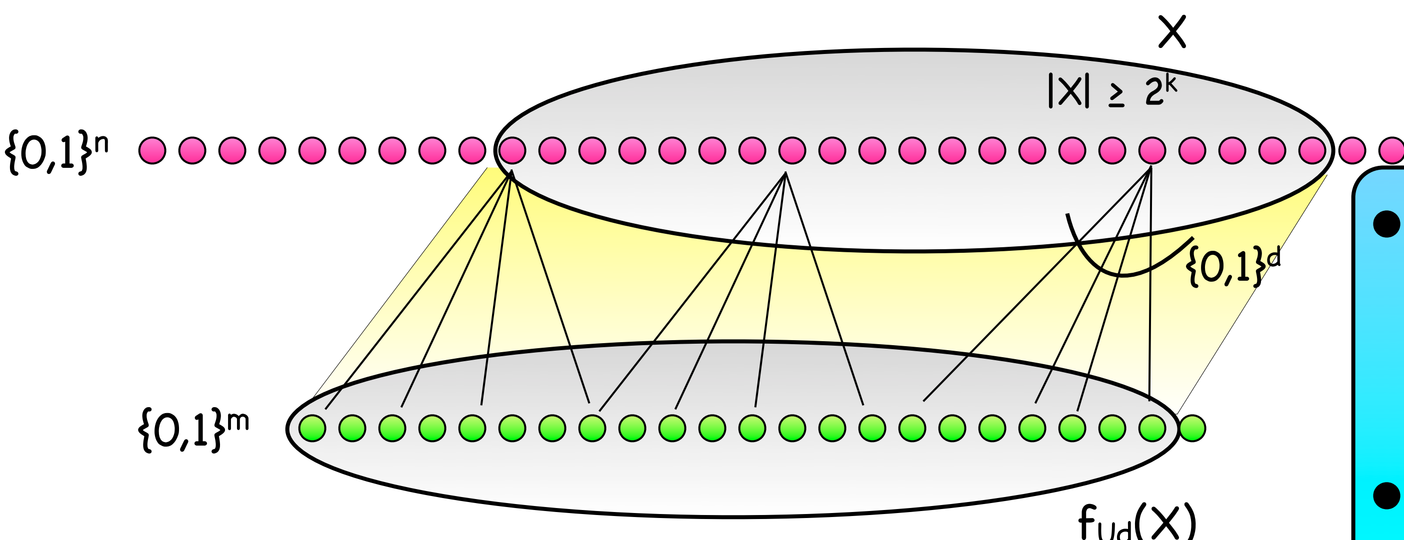
Useful random sources

- What kinds of random sources are useful?
  - Must have sufficient entropy
  - Use min-entropy
    $H_\infty(X) = \min_x \log (1/\Pr[X = x])$
  - $H_\infty(X) \geq k \quad \Leftrightarrow \quad \forall x, \Pr[X = x] \leq 2^{-k}$
- Build deterministic extractor?
  $f : \{0,1\}^n \to \{0,1\}$, s.t. for all X over $\{0,1\}^n$ with $H_\infty(X) \geq n-1$,
  $f(X)$ = uniform bit
  - f cannot exist: $|f^{-1}(0)|$ or $|f^{-1}(1)|$ must be larger than 2^{n-1}.
    For X uniform over larger preimage, $f(X)$ constant

Randomness Extractors

- Allow for (small) **collection** of functions
- k-extractor: family $f_y : \{0,1\}^n \to \{0,1\}^m$, $y \in \{0,1\}^d$
  - For all X with $H_\infty(X) \geq k$, $f_{U_d}(X) \approx U_m$
- For fixed k and n, want minimal d and maximal m

- Where does seed come from?
- When $d = O(\log n)$, can eliminate by enumeration



- Random function w.h.p. is optimal extractor (up to additive factors) [Radhakrishnan-TaShma'97]
- $d = \log(n - k) + O(1)$
- $m = k + d - O(1)$

Building Extractors

- Example of explicit k-extractor (for $k = 0.99n$, $d = O(\log n)$) [Zuc'06]:
  - Fix $(2^m, D, \lambda)$ expander
  - $n = m + m \log D$
  - Each $w \in \{0,1\}^n$ determines random walk of length $m+1$ in expander
  - $f_i : \{0,1\}^n \to \{0,1\}^m$, $i \in \{1 \ldots m+1\}$ given by $f_i(w)$ = i'th vertex visited in walk w
  - (Still useful despite large k)
- Other constructions based on error-correcting codes, etc.
- Can build explicit optimal extractors (up to multiplicative factors) [Lu-Reingold-Vadhan-Wigderson'03, Guruswami-Umans-Vadhan'06]

Is Randomness Powerful?

- So far: possible to save on randomness
- Question: possible to eliminate randomness?
- Natural strategy: take majority of $A(x; r)$ for all r
  - Exponential time
- Enumerate over poly-size set of random bits that are **indistinguishable** for efficient algorithms

Pseudorandom Generators

- Pseudorandom generator:
  $G : \{0, 1\}^{O(\log m)} \rightarrow \{0, 1\}^m$ computable in time poly(m)
  For all efficient algorithms D,
  $\Pr[D(G(U_{O(\log m)})) = 1] \approx \Pr[D(U_m) = 1]$
- Derandomization: run algorithm with $G(s)$ for all $s \in \{0,1\}^{O(\log m)}$, output majority

Simple(?) Case: Fooling Linear Functions

- ε-biased generator:
  $G : \{0, 1\}^{O(\log m)} \to \{0, 1\}^m$ computable in time poly(m)
  For all non-zero linear functions $f : \{0,1\}^m \to \{0,1\}$,
  $| \Pr[f(G(U_{O(\log m)})) = 1] - 1/2 | \leq \varepsilon$
- More or less equivalent to linear codes
  - From yesterday we know explicit constructions
- For more general classes of functions, only know conditional constructions
  - Assume existence of hard functions

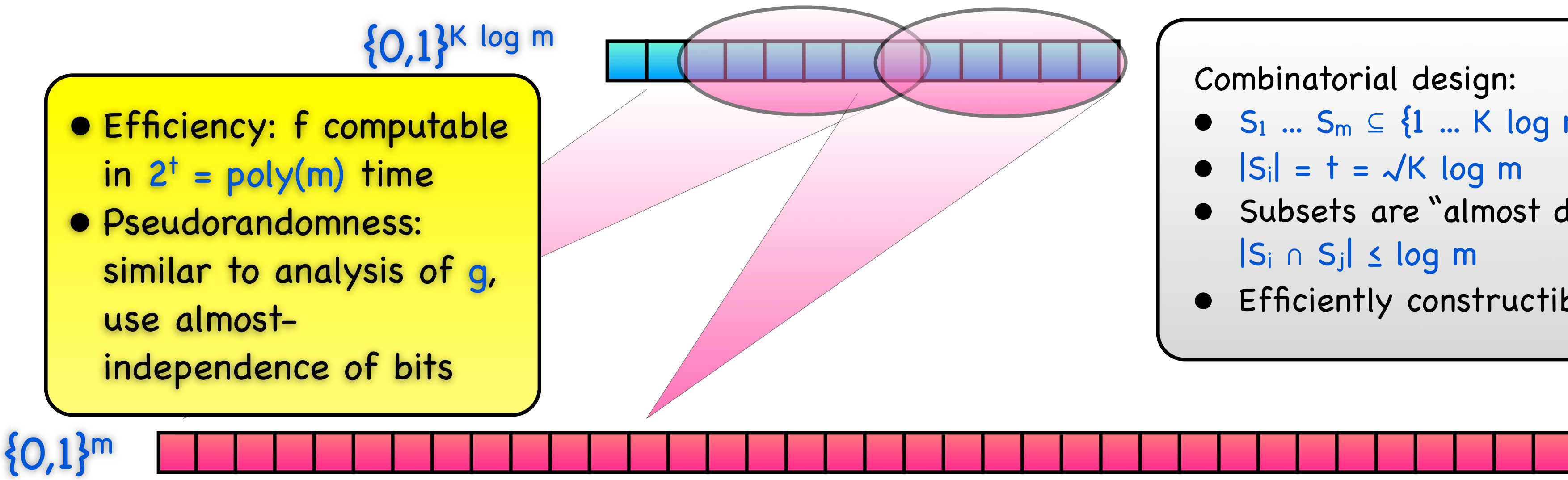
Hardness vs. Randomness

- Suppose $f : \{0,1\}^t \to \{0, 1\}$ hard to compute on average: For all efficient algorithms C, $Pr_{s \leftarrow U_t}[f(s) = C(s)] \approx 1/2$
- g stretching 1 bit: $g(s) = (s, f(s))$
- Proposition: $g(U_t)$ indistinguishable by any efficient algorithm from U_{t+1}
- Problems: stretches only 1 bit, g hard-to-compute

Proof...

Nisan-Wigderson Generator

- Theorem [NW'88]: given $f : \{0,1\}^t \rightarrow \{0,1\}$ sufficiently hard but computable in exponential time, can build PRG $G : \{0,1\}^{K \log m} \rightarrow \{0,1\}^m$



- Efficiency: f computable in $2^t = \text{poly}(m)$ time
- Pseudorandomness: similar to analysis of g, use almost-independence of bits

Combinatorial design:

- $S_1 \ldots S_m \subseteq \{1 \ldots K \log m\}$
- $|S_i| = t = \sqrt{K} \log m$
- Subsets are "almost disjoint": $|S_i \cap S_j| \leq \log m$
- Efficiently constructible

$G(x)_i = f(x|_{S_i})$

More about PRG's

- PRG's useful in cryptography [Blum-Micali'82]
- Unconditional PRG's against weaker classes of algorithms:
  - Space-bounded algorithms [Nisan'90]
  - Constant-depth circuits [Ajtai-Wigderson'85, Braverman'09]
  - Linear functions [Naor-Naor'90]
  - etc...

Fin