CS151
Complexity Theory

Lecture 12

May 6, 2021

Extractors

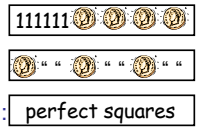
- PRGs: can remove randomness from algorithms
  - based on unproven assumption
  - polynomial slow-down
  - not applicable in other settings
- Question: can we use "real" randomness?
  - physical source
  - imperfect – biased, correlated

Extractors

- "Hardware" side
  - what physical source?
  - ask the physicists…
- "Software" side
  - what is the minimum we need from the physical source?

Extractors

- imperfect sources:
  - "stuck bits":
  - "correlation":
  - "more insidious correlation":



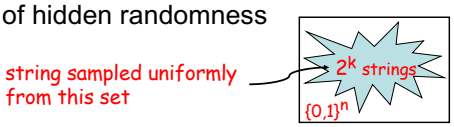
- there are specific ways to get independent unbiased random bits from specific imperfect physical sources

Extractors

- want to assume we don't know details of physical source
- **general model** capturing all of these?
  - yes: "min-entropy"
- **universal procedure** for all imperfect sources?
  - yes: "extractors"

Min-entropy

- General model of physical source w/ k < n bits of hidden randomness

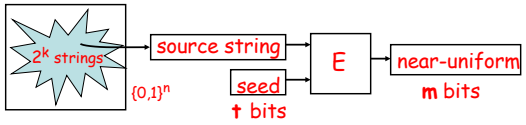


Definition: random variable X on $\{0,1\}^n$ has **min-entropy** $\min_x -\log(\Pr[X = x])$

- min-entropy k implies no string has weight more than 2^{-k}

Extractor

- Extractor: universal procedure for "purifying" imperfect source:



  - E is efficiently computable
  - truly random seed as "catalyst"

Extractor

"(k, ε)-extractor" ⇒ for all X with min-entropy k:

- output fools **all** circuits C:

$$|Pr_z[C(z) = 1] - Pr_{y,\ x \leftarrow X}[C(E(x, y)) = 1]| \leq \boldsymbol{\varepsilon}$$

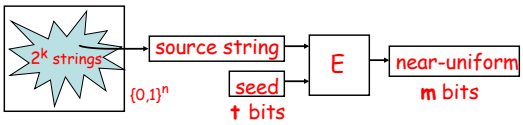
- distributions $E(X, U_t)$, U_m "**ε**-close" (L_1 dist ≤ 2**ε**)

- Notice similarity to PRGs
  - output of PRG fools all **efficient** tests
  - output of extractor fools **all** tests

Extractors

- Using extractors
  - use output in place of randomness in any application
  - alters probability of **any** outcome by at most ε
- Main motivating application:
  - use output in place of randomness in algorithm
  - how to get truly random seed?
  - enumerate all seeds, take majority

Extractors



- Goals:

	good:	best:
short seed	$O(\log n)$	$\log n + O(1)$
long output	$m = k^{\Omega(1)}$	$m = k + t - O(1)$
many k's	$k = n^{\Omega(1)}$	any $k = k(n)$

Extractors

- random function for E achieves best !
  - but we need **explicit** constructions
  - many known; often complex + technical
  - optimal extractors still open
- Trevisan Extractor:
  - insight: use NW generator with source string in place of hard function
  - this works (!!)
  - proof slightly different than NW, easier

Trevisan Extractor

- Ingredients: ($\delta > 0$, m are parameters)
  - error-correcting code

$$C:\{0,1\}^n \rightarrow \{0,1\}^{n'}$$

distance $(\frac{1}{2} - \frac{1}{4}m^{-4})n'$ blocklength $n' = \text{poly}(n)$

  - $(\log n', a = \delta \log n/3)$ design:
    $S_1, S_2, \ldots, S_m \in \{1 \ldots t = O(\log n')\}$

$$E(x, y) = C(x)[y_{|S_1}] \circ C(x)[y_{|S_2}] \circ \ldots \circ C(x)[y_{|S_m}]$$

Trevisan Extractor

$E(x, y)=C(x)[y_{|S_1}] \circ C(x)[y_{|S_2}] \circ \ldots \circ C(x)[y_{|S_m}]$

$C(x)$: 01010010111110101011001010

seed y

Theorem (T): E is an extractor for min-entropy $k = n^{\delta}$, with
- output length $m = k^{1/3}$
- seed length $t = O(\log n)$
- error $\varepsilon \le 1/m$

Trevisan Extractor

- Proof:
  - given $X \subseteq \{0,1\}^n$ of size 2^k
  - assume E fails to ε-pass statistical test C

    $|Pr_z[C(z) = 1] - Pr_{x \leftarrow X,\, y}[C(E(x, y)) = 1]| > \varepsilon$
  - **distinguisher** C ⇒ **predictor** P:

    $Pr_{x \leftarrow X,\, y}[P(E(x, y)_1 \ldots_{i-1})=E(x, y)_i] > ½ + \varepsilon/m$

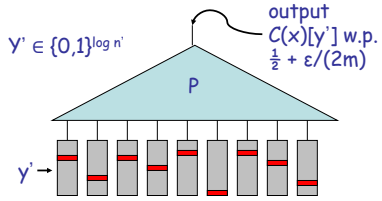
Trevisan Extractor

- Proof (continued):
  - for at least ε/2 of x ∈ X we have:

    $Pr_y[P(E(x, y)_1 \ldots_{i-1})=E(x, y)_i] > ½ + \varepsilon/(2m)$
  - fix bits α,β outside of S_i to preserve advantage

    $Pr_{y'}[P(E(x; \alpha y' \beta)_1 \ldots_{i-1})=C(x)[y']] >½ + \varepsilon/(2m)$
  - as vary y', for j ≠ i, j-th bit of $E(x; \alpha y' \beta)$ varies over only 2^a values
  - (m-1) tables of 2^a values supply $E(x;\alpha y' \beta)_1 \ldots_{i-1}$

Trevisan Extractor



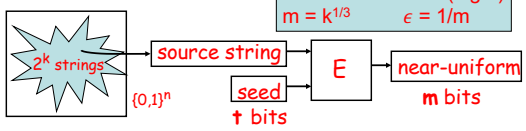
Trevisan Extractor

- Proof (continued):
  - (m-1) tables of size 2^a constitute a **description** of a string that has ½ + ε/(2m) agreement with C(x)
  - # of strings x with such a description?
    - $\exp((m-1)2^a) = \exp(n^{\delta 2/3}) = \exp(k^{2/3})$ strings
    - Johnson Bound: each string accounts for at most $O(m^4)$ x's
    - total #: $O(m^4)\exp(k^{2/3}) << 2^k(\varepsilon/2)$
    - contradiction

Extractors

- (k, ϵ)- extractor:

Trevisan:
$k = n^{\delta}$ $t = O(\log n)$
$m = k^{1/3}$ $\epsilon = 1/m$



  - E is efficiently computable
  - ∀ X with minentropy k, E fools **all** circuits C:

    $|Pr_z[C(z) = 1] - Pr_{y,\, x \leftarrow X}[C(E(x, y)) = 1]| \le \varepsilon$

Strong error reduction

- L ∈ **BPP** if there is a p.p.t. TM M:
  - $x \in L \Rightarrow Pr_y[M(x,y) \text{ accepts}] \geq 2/3$
  - $x \notin L \Rightarrow Pr_y[M(x,y) \text{ rejects}] \geq 2/3$
- Want:
  - $x \in L \Rightarrow Pr_y[M(x,y) \text{ accepts}] \geq 1 - 2^{-k}$
  - $x \notin L \Rightarrow Pr_y[M(x,y) \text{ rejects}] \geq 1 - 2^{-k}$
- We saw: repeat O(k) times
  - n = O(k)·|y| random bits; **2^{n-k} bad strings**

Want to spend n = poly(|y|) random bits; achieve << $2^{n/3}$ bad strings

Strong error reduction

- Better:
  - E extractor for minentropy $k=|y|^3=n^\delta$, $\varepsilon < 1/6$
  - pick random $w \in \{0,1\}^n$, run $M(x, E(w, z))$ for all $z \in \{0,1\}^t$, take majority
  - call w "bad" if $maj_z M(x, E(w, z))$ incorrect $|Pr_z[M(x,E(w,z))=b] - Pr_y[M(x,y)=b]| \geq 1/6$
  - extractor property: at most 2^k bad w
  - n random bits; **2^{n^δ} bad strings**

RL

- Recall: probabilistic Turing Machine
  - deterministic TM with extra tape for "coin flips"
- **RL** (Random Logspace)
  - L ∈ **RL** if there is a probabilistic logspace TM M:
    $x \in L \Rightarrow Pr_y[M(x,y) \text{ accepts}] \geq ½$
    $x \notin L \Rightarrow Pr_y[M(x,y) \text{ rejects}] = 1$
  - important detail #1: only allow one-way access to coin-flip tape
  - important detail #2: explicitly require to run in polynomial time

RL

- **L ⊆ RL ⊆ NL ⊆ SPACE**($\log^2 n$)
- Theorem (SZ) : **RL ⊆ SPACE**($\log^{3/2} n$)
- Belief: L = RL (major open problem)

RL

L ⊆ RL ⊆ NL

- Natural problem:

  Undirected STCONN: given an **undirected** graph G = (V, E), nodes s, t, is there a path from s → t?

Theorem: USTCONN ∈ **RL.**

(Recall: STCONN is **NL**-complete)

Undirected STCONN

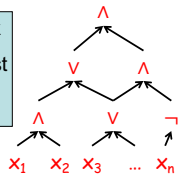
- Proof sketch: (in Papadimitriou)
  - add self-loop to each vertex (technical reasons)
  - start at s, random walk 2|V||E| steps, accept if see t
  - Lemma: expected **return time** for any node i is $2|E|/d_i$
  - suppose $s=v_1, v_2, \ldots, v_n=t$ is a path
  - expected time from v_i to v_{i+1} is $(d_i/2)(2|E|/d_i) = |E|$
  - expected time to reach $v_n \leq |V||E|$
  - Pr[fail reach t in 2|V||E| steps] ≤ ½
- Reingold 2005: USTCONN ∈ **L**

May 6, 2021 CS151 Lecture 12 25

A motivating question

- Central problem in logic synthesis:

> • given Boolean circuit C, integer k
>
> • is there a circuit C' of size at most k that computes the same function C does?



- Complexity of this problem?
  - **NP**-hard? in **NP**? in **coNP**? in **PSPACE**?
  - complete for any of these classes?

May 6, 2021 CS151 Lecture 12 26

Oracle Turing Machines

- Oracle Turing Machine (OTM):
  - multitape TM M with special "query" tape
  - special states $q_?$, q_{yes}, q_{no}
  - on input x, with oracle language A
  - M^A runs as usual, except…
  - when M^A enters state $q_?$:
    - y = contents of query tape
    - $y \in A \Rightarrow$ transition to q_{yes}
    - $y \notin A \Rightarrow$ transition to q_{no}

May 6, 2021 CS151 Lecture 12 27

Oracle Turing Machines

- Nondeterministic OTM
  - defined in the same way
  - (transition relation, rather than function)
- oracle is like a subroutine, or function in your favorite programming language
  - but each call counts as single step

  e.g.: given $\varphi_1, \varphi_2, \ldots, \varphi_n$ are even # satisfiable?
  - poly-time OTM solves with SAT oracle

May 6, 2021 CS151 Lecture 12 28

Oracle Turing Machines

Shorthand #1:

- applying oracles to entire complexity classes:
  - complexity class **C**
  - language A

  $\mathbf{C}^A$ = {L decided by OTM M with oracle A with M "in" **C**}
  - example: $\mathbf{P}^{SAT}$

May 6, 2021 CS151 Lecture 12 29

Oracle Turing Machines

Shorthand #2:

- using complexity classes as oracles:
  - OTM M
  - complexity class **C**
  - $M^{\mathbf{C}}$ decides language L if for some language $A \in \mathbf{C}$, M^A decides L

Both together: $\mathbf{C}^{\mathbf{D}}$ = languages decided by OTM "in" **C** with oracle language from **D**

exercise: show $\mathbf{P}^{SAT} = \mathbf{P}^{\mathbf{NP}}$

May 6, 2021 CS151 Lecture 12 30

The Polynomial-Time Hierarchy

- can define lots of complexity classes using oracles
- the classes on the next slide stand out
  - they have natural complete problems
  - they have a natural interpretation in terms of alternating quantifiers
  - they help us state certain consequences and containments (more later)

The Polynomial-Time Hierarchy

$\Sigma_0 = \Pi_0 = P$

$\Delta_1 = P^P$	$\Sigma_1 = NP$	$\Pi_1 = coNP$
$\Delta_2 = P^{NP}$	$\Sigma_2 = NP^{NP}$	$\Pi_2 = coNP^{NP}$
$\Delta_{i+1} = P^{\Sigma_i}$	$\Sigma_{i+i} = NP^{\Sigma_i}$	$\Pi_{i+1} = coNP^{\Sigma_i}$

Polynomial Hierarchy $PH = \cup_i \Sigma_i$