Pseudorandomness and Derandomization

Shuichi Hirahara

(National Institute of Informatics)

Inter-University Research Institute Corporation / Research Organization of Information and Systems
National Institute of Informatics

第三回量子情報スクール

２０２０年6月３０日（火）

Randomized algorithms

- Randomized algorithms are useful and fast, but…

- How can we implement randomized algorithms?
  - srand(time(NULL)); rand(); rand(); … ← No nice theoretical guarantee
  - Use noise, the motion of mouse pointers, radioactive rays (放射線)

    ← It costs a lot to obtain truly random bits.

- Two approaches: Randomness extractor and derandomization.

Two approaches

1. Randomness extractor
   - enables us to extract (almost) uniform bits from *"sufficiently random"* sources.
     ("sufficiently random": min-entropy is large)
   - Example: extracts uniform bits from the motion of a mouse pointer.

2. Derandomization
   - The set of techniques that reduces the amount of random bits used by an efficient randomized algorithm to (ideally) $O(\log n)$ bits.
   - $O(\log n)$ random bits can be simulated in polynomial time.

BPP: Bounded-error Probabilistic Polynomial-time

- $f: \{0,1\}^* \to \{0,1\}$, a decision problem.
- A: a two-sided-error polynomial-time randomized algorithm for solving f.

For some polynomial p, for all $n \in \mathbb{N}$, for any input $x \in \{0,1\}^n$, the following holds:

$$\Pr_{r \sim \{0,1\}^{p(n)}}[A(x; r) = f(x)] \geq \frac{3}{4}.$$

x: an input

r: random bits

- BPP is the class of decision problems f that can be solved by some two-sided-error polynomial-time randomized algorithm.

Hardness versus Randomness framework

- [Yao '82], [Blum & Micali '84], [Nisan & Wigderson '94], …
- If there is a circuit lower bound for explicit functions, then randomized algorithms can be derandomized.

Theorem [Impagliazzo & Wigderson 1997]

If $\mathrm{E} \not\subseteq \text{io-}\mathrm{SIZE}(2^{\epsilon n})$ for some constant $\epsilon > 0$, then $\mathrm{P} = \mathrm{BPP}$.

- An explicit function: computable by a Turing machine in time $2^{O(n)}$.

Complexity classes

- $\mathrm{E} \not\subseteq$ io-$\mathrm{SIZE}(2^{\epsilon n})$ means:

  There is a function $f:\{0,1\}^* \to \{0,1\}$ such that
  1. f is computable in time $2^{O(n)}$, and
  2. for all large $n \in \mathbb{N}$, f_n cannot be computed by a circuit of size $2^{\epsilon n}$.

  $f_n:\{0,1\}^n \to \{0,1\}$, the restriction of f to n-bit inputs.

- $\mathrm{E} = \mathrm{DTIME}\left(2^{O(n)}\right)$
- $\mathrm{SIZE}\big(s(n)\big)$ is the class of the functions $f:\{0,1\}^* \to \{0,1\}$ such that f_n is computable by a circuit of size $s(n)$ for all large n.
- io-$\mathcal{C} = \{f \mid \exists g \in \mathcal{C}, \exists n_0 \forall n \geq n_0,\ f_n = g_n\}$.

- It is an open question to prove $\mathrm{E} \not\subseteq \mathrm{SIZE}(6n)$. But believed to be $\mathrm{E} \not\subseteq$ io-$\mathrm{SIZE}(2^{\epsilon n})$.

Hardness versus Randomness

Theorem [Impagliazzo & Wigderson 1997]

$$\mathrm{E} \not\subseteq \text{io-SIZE}(2^{\epsilon n}) \implies \mathrm{P} = \mathrm{BPP}.$$

- The hypothesis: Cannot compute some explicit function. (Hardness)
- The conclusion: Can simulate BPP in deterministic polynomial time.

Impossibility $\Longrightarrow$ Possibility

Hardness versus Randomness

Theorem [Impagliazzo & Wigderson 1997]

$$\mathrm{E} \not\subseteq \text{io-SIZE}(2^{\epsilon n}) \implies \mathrm{P} = \mathrm{BPP}.$$

- The hypothesis: Cannot compute some explicit function. (Hardness)
- The conclusion: Can compute a *"pseudorandom generator"* that cannot be distinguished by any efficient algorithm.

Impossibility $\implies$ Possibility

Outline

1. The notion of pseudorandom generator

2. Constructions of pseudorandom generators

Is rand() a good pseudorandom sequence?

- Let's try to implement a randomized algorithm $A(x; r)$.
  - How should we deal with random bits r?

$r_0 \coloneqq \mathrm{srand}(\mathrm{time}(\mathrm{NULL}))$
$r_1 \coloneqq \mathrm{rand}()$
$r_2 \coloneqq \mathrm{rand}()$
$r_3 \coloneqq \mathrm{rand}()$
…

An implementation of rand()

```
int rand () {
   rand_next = rand_next * 1103515245 + 12345;
   return rand_next & 0x7fffffff;
}
```

- rand() is a *linear congruential generator*.

Is rand() a good pseudorandom sequence?

- Let's try to implement a randomized algorithm $A(x; r)$.
  - How should we deal with random bits r?

$$
\begin{aligned}
r_0 &\coloneqq \text{seed} \\
r_1 &\coloneqq (1103515245 \times r_0 + 12345) \bmod 2^{31} \\
r_2 &\coloneqq (1103515245 \times r_1 + 12345) \bmod 2^{31} \\
r_3 &\coloneqq (1103515245 \times r_2 + 12345) \bmod 2^{31} \\
&\dots
\end{aligned}
$$

An implementation of rand()

```
int rand () {
   rand_next = rand_next * 1103515245 + 12345;
   return rand_next & 0x7fffffff;
}
```

- rand() is a *linear congruential generator*.

Simulating a randomized algorithm $A(x; r)$

- $G: \{0,1\}^s \to \{0,1\}^m$ be the function that takes a seed z and outputs the sequence generated by $\mathrm{rand}()$.

$$G(z) = z r_1 r_2 r_3 \ldots, \text{ where } r_{i+1} = (a r_i + c) \bmod 2^{31}, r_0 = z,$$
$$a = 1103515245,$$
$$c = 12345.$$

- Is it possible to simulate $A(x; r)$ in the following sense?

$$\forall x, \quad \Pr_{r \sim \{0,1\}^m}[A(x; r) = f(x)] \approx \Pr_{z \sim \{0,1\}^s}\left[A\left(x; G(z)\right) = f(x)\right].$$

- $f: \{0,1\}^n \to \{0,1\}$, a function computed by A.

Simulating a randomized algorithm $A(x; r)$

- $G: \{0,1\}^s \to \{0,1\}^m$ be the function that takes a seed z and outputs the sequence generated by $\mathrm{rand}()$.

$$G(z) = z r_1 r_2 r_3 \ldots, \text{ where } r_{i+1} = (a r_i + c) \bmod 2^{31}, r_0 = z,$$
$$a = 1103515245,$$
$$c = 12345.$$

- Is it possible to simulate $A(x; r)$ in the following sense?

$$\forall x, \quad \Pr_{r \sim \{0,1\}^m}[A(x; r) = 1] \approx \Pr_{z \sim \{0,1\}^s}\left[A\left(x; G(z)\right) = 1\right].$$

- $f: \{0,1\}^n \to \{0,1\}$, a function computed by A.
- For simplicity, we assume $f \equiv 1$. (This does not lose the generality.)

rand() cannot “simulate” some A

- $G: \{0,1\}^s \to \{0,1\}^m$

  $G(z) = zr_1r_2r_3\ldots$, where $r_{i+1} = (ar_i + c) \bmod 2^{31}$, $r_0 = z$,

- Consider the following algorithm $A(;r)$:

$$A(;r_0r_1r_2\ldots) := \begin{cases} 1 & \text{if } r_1 = (ar_0 + c) \bmod 2^{31} \\ 0 & \text{otherwise} \end{cases}$$

- $\Pr_{z\sim\{0,1\}^s}[A(;G(z)) = 1] = 1.$
- $\Pr_{r\sim\{0,1\}^m}[A(;r) = 1] = \Pr[r_1 = (ar_0 + c) \bmod 2^{31}] = 2^{-31} \approx 0.$
- $A(;-)$ distinguishes a sequence $G(z)$ from the uniform distribution r.

Remark

Even if a and c are unknown,
there is an efficient algorithm A'
that distinguishes $G(z)$ from r.
A' solves the following linear equations:

$$r_1 = (ar_0 + c) \bmod 2^{31}$$
$$r_2 = (ar_1 + c) \bmod 2^{31}$$

Statistical Test

- Let $G: \{0,1\}^s \rightarrow \{0,1\}^m$ be a function such that $s < m$.

  Regarded as a generator that takes a seed z of length s
  and output a "pseudorandom sequence" $G(z)$.

- $T: \{0,1\}^m \rightarrow \{0,1\}$ is said to ϵ-distinguish $G(-)$ (from the uniform distribution) if

$$\left| \Pr_{z \sim \{0,1\}^s}\left[T\big(G(z)\big) = 1\right] - \Pr_{r \sim \{0,1\}^m}\left[T(r) = 1\right] \right| \geq \epsilon$$

- T is also called an ϵ-statistical test (or ϵ-distinguisher) for G.

- By default, we choose $\epsilon := 1/m$ and simply say T distinguishes $G(-)$.

Pseudorandom Generator (PRG)

- Let $G: \{0,1\}^s \to \{0,1\}^m$ be a function such that $s < n$.

- G is called a pseudorandom generator ϵ-secure against a class $\mathcal{C}$ if every $T \in \mathcal{C}$ cannot distinguish G. In other words, for every $T \in \mathcal{C}$,

$$\left| \Pr_{z\sim\{0,1\}^s}\left[T\big(G(z)\big) = 1\right] - \Pr_{r\sim\{0,1\}^m}\left[T(r) = 1\right] \right| < \epsilon.$$

  - rand() is a bad example of a candidate pseudorandom generator. Never use rand() for cryptographic purposes!

- We can simulate $A(x; r)$ if there is a PRG G secure against $\mathcal{C}$ such that $A(x; -) \in \mathcal{C}$ for every input $x \in \{0,1\}^*$. In other words:

$$\left| \Pr_{z\sim\{0,1\}^s}\left[A\big(x; G(z)\big) = 1\right] - \Pr_{r\sim\{0,1\}^m}\left[A(x; r) = 1\right] \right| < \frac{1}{m}.$$

$\exists$ PRG $\implies$ BPP can be derandomized

- Assume $\exists$ PRG $G = \{G_m: \{0,1\}^{O(\log m)} \to \{0,1\}^m\}$ secure against linear-size circuits (i.e., circuits of size m) and computable in time $m^{O(1)}$.

- Take any $f \in \mathrm{BPP}$ and a randomized algorithm $A(x;r)$ for f: for some polynomial p,

$$\Pr_{r\sim\{0,1\}^{p(n)}}[A(x;r) = f(x)] \geq \frac{3}{4}.$$

$\wr\wr$ (security) $\leftarrow$ Consider the circuit $C_x(r) := A(x;r) \oplus f(x) \oplus 1$.

$$\Pr_{z\sim\{0,1\}^{O(\log n)}}\left[A\left(x; G_{p(n)}(z)\right) = f(x)\right] \geq \frac{3}{4} - \frac{1}{p(n)} \geq \frac{2}{3}.$$

- The new algorithm $A\left(x; G_{p(n)}(z)\right)$ only uses $O(\log n)$ random bits!

  $\implies$ Can be simulated in polynomial time by exhaustively trying all the random bits.

Outline

1. The notion of pseudorandom generator

2. Constructions of pseudorandom generators

Three key ideas for constructing PRGs

1. Distinguishable $\Leftrightarrow$ Next-bit-predictable

   $G: \{0,1\}^n \to \{0,1\}^{n+1}$, 1-bit extension

2. Hybrid arguments

   $G: \{0,1\}^{nk} \to \{0,1\}^{nk+k}$, k-bit extension

3. Combinatorial design

   $G: \{0,1\}^{O(\log m)} \to \{0,1\}^m$, exponential stretch

The simplest construction of a PRG

- Let's construct a non-trivial pseudorandom generator $G: \{0,1\}^n \to \{0,1\}^{n+1}$.

Claim (essentially due to [Yao'82])

If $\mathrm{E} \not\subseteq \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; \delta)$, then there is a PRG $G: \{0,1\}^n \to \{0,1\}^{n+1}$ δ-secure against exponential-size circuits and computable in time $2^{O(n)}$.

- $\widetilde{\mathrm{SIZE}}(s(n); \delta)$: The class of functions $h: \{0,1\}^* \to \{0,1\}$ such that, for all n, there is a circuit C of size $s(n)$ that δ-approximates h_n, i.e.,

$$\Pr_{x \sim \{0,1\}^n}[C(x) = h_n(x)] \geq \frac{1}{2} + \delta\,.$$

- Take a hard function $h: \{0,1\}^* \to \{0,1\}$ such that $h \in \mathrm{E} \setminus \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; \delta)$.

The Construction of the Simple PRG

Construction: $G^h: \{0,1\}^n \to \{0,1\}^{n+1}$,
$G^h(z) \coloneqq \big(z, h_n(z)\big) \in \{0,1\}^{n+1}$.

h: a hard function in E.

Claim: If ∃ a distinguisher $D: \{0,1\}^{n+1} \to \{0,1\}$ for G^h, then h_n can be approximated.

D: a circuit of size $2^{\epsilon n}$.

(by a circuit of size $2^{\epsilon n}$.)

$\Longrightarrow$ Contradiction to $h \notin \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; \delta)$.

$\Longrightarrow$ G^h is secure against circuits of size $2^{\epsilon n}$.

Proof:
$$\left| \Pr_{z \in \{0,1\}^n}\big[D\big(G(z)\big) = 1\big] - \Pr_{w \in \{0,1\}^{n+1}}[D(w) = 1] \right| \geq \delta.$$

⇓

$$\Pr_{z \in \{0,1\}^n}\big[D\big(G(z)\big) = 1\big] - \Pr_{w \in \{0,1\}^{n+1}}[D(w) = 1] \geq \delta$$

or

$$\Pr_{z \in \{0,1\}^n}\big[D\big(G(z)\big) = 1\big] - \Pr_{w \in \{0,1\}^{n+1}}[D(w) = 1] \leq -\delta.$$

⇓ Define $D'(w) \coloneqq \neg D(w)$.

$$\Pr_{z \in \{0,1\}^n}\big[D'\big(G(z)\big) = 1\big] - \Pr_{w \in \{0,1\}^{n+1}}[D'(w) = 1] \geq \delta.$$

W.l.o.g, we consider the left case.

Distinguishable ⟹ Next-bit-predictable

Claim: If ∃ a distinguisher $D: \{0,1\}^{n+1} \to \{0,1\}$ for G^h, then h_n can be approximated.

D: a circuit of size $2^{\epsilon n}$. (by a circuit of size $2^{\epsilon n}$.)

$$\Pr_{z \in \{0,1\}^n}\left[D\left(z, h_n(z)\right) = 1\right] - \Pr_{w \in \{0,1\}^{n+1}}[D(w) = 1] \geq \delta.$$

- D can distinguish (1) $\left(z, h_n(z)\right)$, where $z \sim \{0,1\}^n$, from

  (2) (z, b), where $z \sim \{0,1\}^n$ and $b \sim \{0,1\}$.

[Yao'82]

⟹ Can construct a "next-bit predictor" P^D.

Given the first n-bits of $G(z)$, can you predict the next bit?

z $h_n(z)$

(Cf. Each next bit of rand() can be predicted easily.)

Distinguishable $\Longrightarrow$ Next-bit-predictable

Claim: If $\exists$ a distinguisher $D: \{0,1\}^{n+1} \to \{0,1\}$ for G^h, then h_n can be approximated.

$$\Pr_{z \in \{0,1\}^n}[D(z, h_n(z)) = 1] - \Pr_{w \in \{0,1\}^{n+1}}[D(w) = 1] \geq \delta.$$

- D can distinguish (1) $(z, h_n(z))$, where $z \sim \{0,1\}^n$, from
  (2) (z, b), where $z \sim \{0,1\}^n$ and $b \sim \{0,1\}$.

"next-bit predictor" P^D:

$$P^D(z; b) = \begin{cases} b & \text{if } D(z,b) = 1 \\ b \oplus 1 & \text{otherwise} \end{cases} \qquad b \sim \{0,1\}$$

Idea: If $D(z, b) = 1$, we can expect that $h_s(z) = b$.

Fact: $\Pr_z[P^D(z; b) = h_n(z)] \geq \frac{1}{2} + \delta$ $\Longrightarrow$ h_n can be δ-approximated.

Proof of the Fact

$$P^D(z;b) = \begin{cases} b & \text{if } D(z,b) = 1 \\ b \oplus 1 & \text{otherwise} \end{cases}$$

Fact: $\Pr_z[P^D(z;b) = h_n(z)] \geq \frac{1}{2} + \delta$

Assumption: $\Pr_{z\sim\{0,1\}^n}[D(z,h_s(z)) = 1] - \Pr_{\substack{z\sim\{0,1\}^n \\ b\sim\{0,1\}}}[D(z,b) = 1] \geq \delta.$

Observe $\Pr_{\substack{z\sim\{0,1\}^n \\ b\sim\{0,1\}}}[D(z,b) = 1] = \frac{1}{2}\Pr_z[D(z,h_n(z)) = 1] + \frac{1}{2}\Pr_z[D(z,\neg h_n(z)) = 1].$

($b = h_n(z)$ or $b = \neg h_n(z)$)

$$\Rightarrow \quad \frac{1}{2}\Pr_{z\sim\{0,1\}^n}[D(z,h_n(z)) = 1] - \frac{1}{2}\Pr_z[D(z,\neg h_n(z)) = 1] \geq \delta.$$

$$\Pr_{z,b}[P^D(z;b) = h_n(z)] = \frac{1}{2}\Pr[D(z,h_n(z)) = 1] + \frac{1}{2}\Pr[D(z,\neg h_n(z)) = 0]$$ ($b = h_n(z)$ or $b = \neg h_n(z)$)

$$= \frac{1}{2}\Pr[D(z,h_n(z)) = 1] + \frac{1}{2} - \Pr[D(z,\neg h_n(z)) = 1] \geq \frac{1}{2} + \delta.$$

The simplest construction of a PRG

Claim (essentially due to [Yao'82])

If $\mathrm{E} \not\subseteq \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; \delta)$, then there is a PRG $G: \{0,1\}^n \to \{0,1\}^{n+1}$ δ-secure against $2^{\epsilon n}$-sized circuits and computable in time $2^{O(n)}$.

- The Construction:
  - Take a hard function $h: \{0,1\}^* \to \{0,1\}$ such that $h \in \mathrm{E} \setminus \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; \delta)$.
  - Define $G^h(z) := \big(z, h(z)\big) \in \{0,1\}^{n+1}$, where $z \in \{0,1\}^n$.
- Key Idea: D: a distinguisher $\implies$ P^D: a next-bit predictor

Three key ideas for constructing PRGs

1. Distinguishable $\Leftrightarrow$ Next-bit-predictable

   $G: \{0,1\}^n \rightarrow \{0,1\}^{n+1}$, 1-bit extension

2. Hybrid arguments

   $G: \{0,1\}^{nk} \rightarrow \{0,1\}^{nk+k}$, k-bit extension

3. Combinatorial design

   $G: \{0,1\}^{O(\log m)} \rightarrow \{0,1\}^m$, exponential stretch

1-bit extension to k-bit extension

- Take a hard function $h: \{0,1\}^n \to \{0,1\}$.
- A PRG $G^h: \{0,1\}^n \to \{0,1\}^{n+1}$ that extends the seed by 1 bit:

$$G^h(z) := \big(z, h(z)\big).$$

Hardness of $h \Longrightarrow$ Security of G^h

- Want to extend the seed by k bits:

$$\mathrm{DP}_k^h: \{0,1\}^{nk} \to \{0,1\}^{nk+k}$$

$$\mathrm{DP}_k^h := \left(G^h\right)^{\oplus k}$$

$$\mathrm{DP}_k^h(z_1, \ldots, z_k) := \big(z_1, \ldots, z_k, h(z_1), \ldots, h(z_k)\big)$$

k-bit Extension: k-wise Direct Product Genertor

Claim

If $\mathrm{E} \not\subseteq \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; \delta/k)$, then there is a PRG $G: \{0,1\}^{kn} \to \{0,1\}^{kn+k}$ δ-secure against $2^{\epsilon n}$-sized circuits and computable in time $2^{O(n)}$.

- **The Construction:** k-wise Direct Product Generator
  - Take a hard function $h: \{0,1\}^* \to \{0,1\}$ such that $h \in \mathrm{E} \setminus \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; \delta/k)$.
  - Define $\mathrm{DP}_k^h: (\{0,1\}^n)^k \to \{0,1\}^{nk+k}$

$$\mathrm{DP}_k^h(z_1, \dots, z_k) := \big(z_1, \dots, z_k, h(z_1), \dots, h(z_k)\big).$$

Key Idea: Hybrid Argument

$$\mathrm{DP}_k^h : (\{0,1\}^n)^k \to \{0,1\}^{nk+k}$$

$$\mathrm{DP}_k^h(z_1, \dots, z_k) := \big(z_1, \dots, z_k, h(z_1), \dots, h(z_k)\big).$$

- Assume ∃ a distinguisher D for $\mathrm{DP}_k^h(-)$:

$$\Pr_{z_1,\dots,z_k}\left[D\left(\mathrm{DP}_k^h(z_1,\dots,z_k)\right) = 1\right] - \Pr_{\substack{z_1,\dots,z_k \\ b_1,\dots,b_k}}[D(z_1,\dots,z_k,b_1,\dots,b_k) = 1] \geq \delta$$

- It is difficult to directly compare $\big(z_1, \dots, z_k, h(z_1), \dots, h(z_k)\big)$ and $(z_1, \dots, z_k, b_1, \dots, b_k)$.

- Key Idea: Hybrid argument, which considers intermediate distributions $H_0, H_1, \dots, H_k$.

Hybrid Argument

$$\Pr_{z_1,\dots,z_k}\left[D\left(z_1,\dots,z_k,h(z_1),\dots,h(z_k)\right)=1\right]-\Pr_{\substack{z_1,\dots,z_k\\ b_1,\dots,b_k}}\left[D(z_1,\dots,z_k,b_1,\dots,b_k)=1\right]\geq\delta$$

$\equiv H_k$ $\qquad\qquad$ $\equiv H_0$

- Define the i-th hybrid $H_i \equiv (z_1,\dots,z_k,h(z_1),\dots,h(z_i),b_{i+1}\dots,b_k)$ where $i\in\{0,\dots,k\}$, $z_j\sim\{0,1\}^n$, $b_j\sim\{0,1\}$ for any $j\in\{1,..,k\}$.

$$\delta\leq\Pr[D(H_k)=1]-\Pr[D(H_0)=1]=\sum_{i=1}^{k}(\Pr[D(H_i)=1]-\Pr[D(H_{i-1})=1])$$

$\Longrightarrow\quad \Pr[D(H_i)=1]-\Pr[D(H_{i-1})=1]\geq\delta/k\qquad$ for some $i\in\{0,\dots,k\}$.

$$\Longrightarrow\quad \Pr[D(z_1,\dots,z_k,h(z_1),\dots,h(z_i),b_{i+1},\dots,b_k)=1]-\Pr[D(z_1,\dots,z_k,h(z_1),\dots,b_i,b_{i+1}\dots,b_k)=1]\geq\delta/k$$

$D'(z_i,b_i)\coloneqq D(z_1,\dots,z_k,h(z_1),\dots,b_k)$
Fix z's and b's except for (z_i,b_i).

$$\Longrightarrow\quad \Pr_{z_i}\left[D'\left(z_i,h(z_i)\right)=1\right]-\Pr_{z_i,b_i}[D'(z_i,b_i)=1]\geq\delta/k\quad\Longrightarrow\quad h\in\widetilde{\mathrm{SIZE}}(2^{\epsilon n};\delta/k).$$

Interlude: Recent applications of DP_k^h

- The k-wise direct product generator DP_k^h is not a good construction in the context of derandomization.

$$\mathrm{DP}_k^h: \{0,1\}^{nk} \to \{0,1\}^{nk+k}$$

$nk + k$ random bits can be reduced to nk.

- However, it recently turned out that DP_k^h is an important tool for analyzing the (meta-)complexity of Kolmogorov complexity.
[H. (FOCS'18)], [H. (STOC'20)], [H. (CCC'20)]

Three key ideas for constructing PRGs

1. Distinguishable $\Leftrightarrow$ Next-bit-predictable

   $G: \{0,1\}^n \to \{0,1\}^{n+1}$, 1-bit extension

2. Hybrid arguments

   $G: \{0,1\}^{nk} \to \{0,1\}^{nk+k}$, k-bit extension

3. Combinatorial design

   $G: \{0,1\}^{O(\log m)} \to \{0,1\}^m$, exponential stretch

k-bit extension to exponential extension

- The k-wise direct product generator:

$$\mathrm{DP}_k^h: \quad (z_1, \dots, z_k) \mapsto \big(z_1, \dots, z_k, h(z_1), \dots, h(z_k)\big)$$

Computing h is hard $\Longrightarrow$ DP_k^h is secure.

- Let's try to evaluate h on more (correlated) inputs!

$$\mathrm{NW}^h: \; z \; \mapsto \; \left(z_{S_1}, \dots, z_{S_m}\right) \mapsto \left(h\left(z_{S_1}\right), \dots, h\left(z_{S_m}\right)\right)$$

Exponential Stretch

Theorem [Nisan-Wigderson '94]

If $\mathrm{E} \not\subseteq \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; 2^{-\epsilon n})$, then there is a PRG $G: \{0,1\}^{O(\log m)} \to \{0,1\}^m$ secure against m-size circuits and computable in time $m^{O(1)}$, and in particular, $\mathrm{P} = \mathrm{BPP}$.

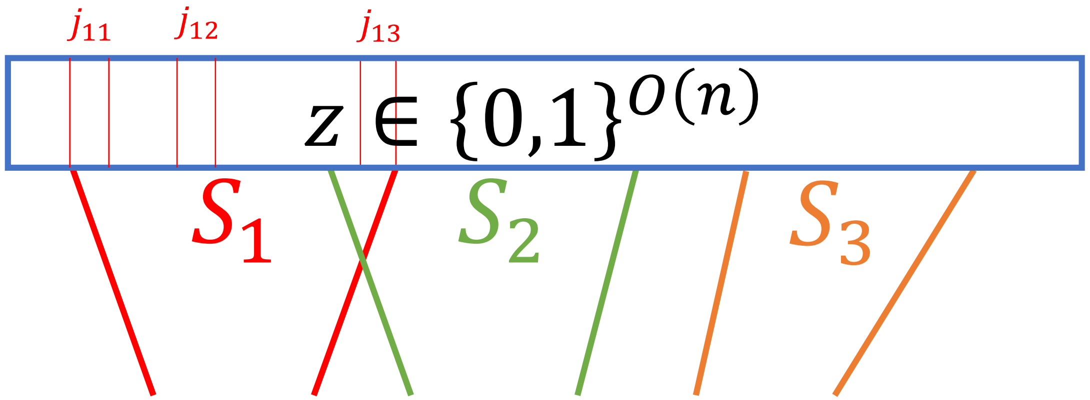
- **The Construction:** The Nisan-Wigderson Generator
  - Take a hard function $h: \{0,1\}^* \to \{0,1\}$ such that $h \in \mathrm{E} \setminus \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; 2^{-\epsilon n})$.
  - Define $\mathrm{NW}^h: \{0,1\}^{O(\log m)} \to \{0,1\}^m$ as

$$\mathrm{NW}^h(z) := \left(h_n(z_{S_1}), \ldots, h_n(z_{S_m})\right) \quad \text{where } n = O(\log m).$$

The Nisan-Wigderson Generator NW^h

- Take a hard function $h: \{0,1\}^n \to \{0,1\}$.

Seed z



$S_1 = \{j_{11}, j_{12}, j_{13}\}$

z_{S_1} z_{S_2} z_{S_3} z_{S_4} ⋯

$\in \{0,1\}^n$

$z_{S_1} = z_{j_{11}} z_{j_{12}} z_{j_{13}}$

$(n = 3)$

↧ ↧ ↧

Output $h(z_{S_1})$ $h(z_{S_2})$ $h(z_{S_3})$ ⋯ $=: \mathrm{NW}^h(z) \in \{0,1\}^{2^{\epsilon n}}$

$\in \{0,1\}$

$m := 2^{\epsilon n}$

Combinatorial Design

Fact (Construction of a combinatorial design)

For any $\epsilon > 0$, for some $d = O(n)$, for any $m \le 2^n$,
there exists a family of sets $S_1, \dots, S_m \subseteq \{1, \dots, d\}$ such that

1. $|S_i| = n$ for all $i \in \{1, \dots, m\}$ and
2. $\left|S_i \cap S_j\right| \le \epsilon n$ for any $i \ne j \in \{1, \dots, m\}$.

Moreover, $\{S_i\}_i$ can be computed by a greedy algorithm in time $m^{O(1)}$.

$z = (z_1, \dots, z_d) \in \{0,1\}^d = \{0,1\}^{O(n)}$.
$z_{S_i} := \left(z_{j_1}, \dots, z_{j_n}\right) \in \{0,1\}^n$, where $S_i = \{j_1 < \cdots < j_n\}$.

$\mathrm{NW}^h \colon \{0,1\}^{d=O(n)} \to \{0,1\}^{2^{\epsilon n}}$

$\mathrm{NW}^h(z) := \left(h_n\left(z_{S_1}\right), \dots, h_n\left(z_{S_m}\right)\right)$ where $n = O(\log m)$.

Security Proof of NW^h

$$\Pr_z\left[D\left(\mathrm{NW}^h(z)\right)=1\right]-\Pr_w[D(w)=1]\geq 1/m$$

$$\Pr_z\left[D\left(h(z_{S_1})\dots h(z_{S_m})\right)=1\right]-\Pr_w[D(w)=1]\geq 1/m$$

- The i-th hybrid distribution: $H_i := \left(h(z_{S_1}),\dots,h(z_{S_i}),w_{i+1},\dots,w_m\right)$, where $z\sim\{0,1\}^d, w\sim\{0,1\}^m$.

$$\Pr[D(H_i)=1]-\Pr[D(H_{i-1})=1]\geq 1/m^2 \text{ for some } i\in\{1,\dots,m\}.$$

$$H_i: \left(h(z_{S_1}),\dots,h(z_{S_i}),w_{i+1},\dots,w_m\right)$$

$$H_{i-1}: \left(h(z_{S_1}),\dots,w_i,w_{i+1},\dots,w_m\right)$$

- Fix $z_{\{1,\dots,d\}\setminus S_i}, w_{i+1},\dots,w_m \Longrightarrow \exists\, D'$ distinguishes $\left(h(z_{S_1}),\dots,h(z_{S_i})\right)$ from $\left(h(z_{S_1}),\dots,w_i\right)$.
- Yao's distinguisher to next-bit predictor transform $\Longrightarrow \exists\, P^{D'}$ predicts $h(z_{S_i})$:

$$\Pr_{z_{S_i}}\left[P^{D'}\left(h(z_{S_1}),\dots,h(z_{S_{i-1}})\right)=h(z_{S_i})\right]\geq\frac{1}{2}+\frac{1}{m^2}.$$

$z_{S_i}\mapsto\left(h(z_{S_1}),\dots,h(z_{S_{i-1}})\right)$ can be computed by a circuit of size $O(2^{\epsilon n}nm)=2^{O(\epsilon n)}$.

(because $|S_i\cap S_1|\leq\epsilon n$ and any function on ϵn bits can be computed by a circuit of size $O(2^{\epsilon n}n)$)

$$\Longrightarrow\quad h\in\widetilde{\mathrm{SIZE}}\left(2^{O(\epsilon n)};2^{-2\epsilon n}\right).$$

Three key ideas for constructing PRGs

1. Distinguishable $\Leftrightarrow$ Next-bit-predictable

   $G: \{0,1\}^n \to \{0,1\}^{n+1}$, 1-bit extension

2. Hybrid arguments

   $G: \{0,1\}^{nk} \to \{0,1\}^{nk+k}$, k-bit extension

3. Combinatorial design

   $G: \{0,1\}^{O(\log m)} \to \{0,1\}^m$, exponential stretch

Nisan-Wigderson to Impagliazzo-Wigderson

[Nisan-Wigderson '94] $\mathrm{E} \not\subseteq \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; 2^{-\epsilon n}) \implies \mathrm{P} = \mathrm{BPP}$

$\Updownarrow$ Locally list-decodable error-correcting code

[Impagliazzo-Wigderson '97] $\mathrm{E} \not\subseteq \text{io-SIZE}(2^{\epsilon n}) \implies \mathrm{P} = \mathrm{BPP}$

Properties of locally list-decodable error-correcting code $\mathrm{Enc}: f \mapsto \mathrm{Enc}(f)$

[Sudan-Trevisan-Vadhan '01]

1. $f \in \mathrm{E} \implies \mathrm{Enc}(f) \in \mathrm{E}^f$.
2. $\mathrm{Enc}(f) \in \text{io-}\widetilde{\mathrm{SIZE}}(2^{\epsilon n}; 2^{-\epsilon n}) \implies f \in \mathrm{SIZE}\left(2^{\epsilon' n}\right)$.

Hardness versus Randomness Trade-off

$$\mathrm{EXP} \not\subseteq \mathrm{ioSIZE}\left(n^{O(1)}\right) \implies \mathrm{BPP} \subseteq \mathrm{SUBEXP} := \bigcap_{\epsilon>0} \mathrm{DTIME}\left(2^{n^{\epsilon}}\right).$$

(∃ PRG $G: \{0,1\}^{m^{\epsilon}} \to \{0,1\}^{m}$, computable in time $2^{m^{\epsilon}}$)

$$\mathrm{EXP} \not\subseteq \bigcap_{\epsilon>0} \mathrm{ioSIZE}\left(2^{n^{\epsilon}}\right) \implies \mathrm{BPP} \subseteq \mathrm{QuasiP} := \mathrm{DTIME}\left(2^{(\log n)^{O(1)}}\right).$$

(∃ PRG $G: \{0,1\}^{(\log m)^{O(1)}} \to \{0,1\}^{m}$, computable in time $2^{(\log m)^{O(1)}}$)

$$\mathrm{E} \not\subseteq \bigcap_{\epsilon>0} \mathrm{ioSIZE}(2^{\epsilon n}) \implies \mathrm{BPP} \subseteq \mathrm{P}.$$

(∃ PRG $G: \{0,1\}^{O(\log m)} \to \{0,1\}^{m}$, computable in time $m^{O(1)}$)

More Applications Beyond Derandomization

- ➢ Black-box pseudorandom generator construction NW^h
  $\Longrightarrow$ a seeded extractor [Trevisan '01]

- ➢ Learning $\mathrm{AC}^0[\oplus]$ circuits.
  [Carmosino-Impagliazzo-Kabanets, Kolokolova CCC'16]

- ➢ Non-black-box worst-case to average-case reduction within NP.
  [H. FOCS'18]

Summary

➢ How can we derandomize a randomized algorithm $A(x;r)$?

1. Come up with a problem $h: \{0,1\}^{O(\log n)} \to \{0,1\}$ that cannot be computed by $A(x;-)$. $(\forall x)$ (More precisely, $h \notin \widetilde{\mathrm{SIZE}}^A(2^{\epsilon n}; 2^{-\epsilon n})$.)
   - Example: an E-complete problem
2. Generate a pseudorandom sequence $r := \mathrm{NW}^h(z)$ from a seed z.
3. Simulate $A(x;r)$.

➢ An excellent reference: Salil Vadhan, "Pseudorandomness", 2012