Simple and Fast Derandomization from Very Hard Functions

Lijie Chen and Roei Tell

Warwick-Oxford Seminar, November 2020
In this talk

1. Background
2. Our results
3. A taste of techniques
1 Background
simple and fast derandomization
Randomness in computation

- context

- We need randomness for crypto, learning, sublinear-time...

- Conjecture: We don't need randomness to efficiently
  1. solve decision problems
  2. solve “verifiable” search problems

- Question stands at the heart of complexity theory
BPP \not= P

- Historic recap

- BPP formally defined in [Gill'77]

- Immediately conjectured to “sort-of” equal P

We believe that for the unrelativized classes of Turing machines, only speedups for infinitely many inputs can be achieved by probabilistic machines.
BPP $\not\equiv$ P

› historic recap

› ... in fact, paper even raises stronger conjecture:

Conjecture: If $f$ is a recursive function computed in time $T^*$ by some probabilistic Turing machine with error probability bounded away from 1/2, then there is a deterministic Turing machine which computes $f$ in time $O(T^*(x))$ for infinitely many $x$.

1 the original paper probably means “infinitely-many input lengths”
Hardness-to-randomness

• more recent history

• Hard functions ⇒ efficient pseudorandomness
  [Yao,’82, BM’84]

• Hard functions ⇒ derandomization of BPP
  [NW’94, IW’99, STV’01, SU’01, Uma’03, and others]

• Conditioned on lower bounds, we have an answer
Smooth trade-off from [Uma’03]:

\[ \text{DTIME}[2^n] \nsubseteq \text{ioSIZE}[s] \quad \Rightarrow \quad \text{BPP} \subseteq \text{DTIME}[ \approx 2^{s^{\alpha-1}(n)}] \]

Extreme point [IW’99]:

\[ \text{DTIME}[2^n] \nsubseteq \text{ioSIZE}[2^{0.1n}] \quad \Rightarrow \quad \text{BPP} = \text{P} \]
\[ \Rightarrow \quad \text{BPTIME}[T] \subseteq \text{DTIME}[T^{O(1)}] \]

---

1  we’ll revisit the smoothness of this trade-off later (as well as the “≈” sign...)
Doron, Moshkovitz, Oh, and Zuckerman (FOCS 2020) recently asked: Can we do it faster? \( \text{BPTIME}[T] \subseteq \text{DTIME}[T^c] \) for a small \( c \)?

Classical results can yield “reasonable” \( c \) when scaled-up.

Diff between (say) \( c = 10 \) and \( c = 3 \) is substantial!
Superfast derandomization

◊ ... snap back to now

◊ What is the actual cost of simulating randomness?
  ◊ new area to explore
  ◊ theoretical basis not formed yet

◊ The obvious “end-goal” question:

  Can we simulate randomness with no cost?
Superfast derandomization

- ... snap back to now

- Main result of [DMOZ’20]:

  \[ \text{BPTIME}[T] \subseteq \text{DTIME}[T^{2.01}] \]

  conditioned on

  \[ \text{DTIME}[2^n] \not\subset \text{ioMASIZE}[2^{0.99n}] \]
Superfast derandomization

› ... snap back to now

› Takeaways:

  1. Super-fast derand possible, under assumptions!
  2. Hypothesis is “too strong”
Superfast derandomization

› snap back to now

› Takeaways:

1. Super-fast derand possible, under assumptions!
2. Hypothesis is “too strong”

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[IW’99]</td>
<td>SIZE[2^{0.01n}]</td>
<td>T^{O(1)}</td>
</tr>
<tr>
<td>[DMOZ’20]</td>
<td>MASIZE[2^{-0.99n}]</td>
<td>T^{2.01}</td>
</tr>
</tbody>
</table>
2 Our results

simple and fast derandomization
Derandomization with no overhead

our first main result

Thm 1:

\[ \text{BPTIME}[T] \subseteq \text{DTIME}[n \cdot T^{1.01}] \]

conditioned on

one-way functions &
non-uniformly-strong time hierarchy
Derandomization with no overhead

our first main result

Thm 1:

\[ \text{BPTIME}[T] \subseteq \text{DTIME}[n \cdot T^{1.01}] \]

conditioned on

one-way functions (non-uniformly secure) &

\[ \text{DTIME}[2^{kn}] \not\subseteq \text{iDTIME}[2^{0.99k \cdot n}]/2^{0.99n} \] (k = suff. large const)

\(^1\) the precise technical form of the second hypothesis (a relaxed form) is necessary for obtaining the conclusion using the standard approach of PRGs
Derandomization with no overhead

- meaning of the main result

- Takeaways:
  1. Derandomization with near-linear overhead possible! (under reasonable assumptions)
  2. Hypothesis more standard than in [DMOZ’20]
  3. Theoretical basis for superfast derandomization
Derandomization with no overhead

meaning of the main result

› Randomness might be nearly useless for large T
  › our derandomization is simple & also solves search problems
  › time overhead is minor (to be improved in 2 slides)

› Derandomize “better-than-brute-force” algorithms

› Lower bds for DTIME ⇒ lower bds for BPTIME
  › SETH ⇒ rSETH (assuming Thm 1 for arbitrarily small savings)
Simple and intuitive proofs

- understanding superfast derandomization

- Proof of Thm 1 is intuitive and technically non-involved
  - combining new insights with known technical tools

- In addition: Simple proof for the [DMOZ’20] result
  - extends it: relaxed hardness $\implies$ cubic/quartic derand
Complementing the first result

› zooming-in on the precise overhead

› **Thm 1’**: Under assumptions mildly stronger / more involved:

\[
\text{BPTIME}[T] \subseteq \text{DTIME}[n^{1.01} \cdot T] \quad (\forall T \leq \text{subexp})
\]
Complementing the first result

- zooming-in on the precise overhead

- **Thm 1':** Under assumptions mildly stronger / more involved:
  \[
  \text{BPTIME}[T] \subseteq \text{DTIME}[n^{1.01} \cdot T] \quad (\forall \; T \leq \text{subexp})
  \]

- **Thm 2:** Conditioned on \#NSETH,
  \[
  \text{BPTIME}[T] \not\subseteq \text{DTIME}[n^{0.99} \cdot T] \quad (\forall \; T = \text{poly})
  \]

- **\#NSETH:** We cannot count solutions for a given k-SAT formula in \text{NTIME}[2^{(1-\varepsilon) \cdot n}] \; (\text{assuming suff. large } k=k_{\varepsilon})
Average-case derandomization

› bypassing this barrier

› Thm 3: Under assumptions very similar to Thm 1:

$$\text{BPTIME}[T] \subseteq \text{DTIME}[n^{0.01} \cdot T] \text{ in average-case}$$

› ... with respect to all T-time samplable distributions

› ... with success probability $1 - n^{-\omega(1)}$

› $L \in \text{BPTIME}[T] \Rightarrow$ one alg $A_L$ “looks correct” to all T-time dist.
Extra goodies in the paper

› technical insights intertwined in our proofs

1. Easy way to bypass an informidable-looking barrier
2. Batch-computable PRGs vs amortized time-complexity
3. General simplification of a well-known PRG paradigm
   › ”extract-from-pseudoentropic string” as a special case of an easy-to-analyze strategy
4. New light on “hybrid argument” barrier (it’s not the one above)
3 A taste of techniques
observations & proof sketches
Technical roadmap

› what we’ll talk about

› Bypassing the seed-length barrier
› Proof sketch for Thm 1
› Simplifying a well-known PRG paradigm
Bypassing the seed-length barrier
one technical observation to remember
Hardness-to-randomness

- the basic idea

- hard functions $\Rightarrow$ random-looking bits [Yao,’82, BM’84]

\[
\begin{array}{cccccccc}
X_1 & X_2 & X_3 & \ldots & X_n & f(x)
\end{array}
\]

- \( f \) hard on average $\Rightarrow (x \circ f(x)) \) looks random
Hardness-to-randomness

› reconstructive PRGs

PRG

$G^f_{\ell(n)}$ → $T(n)$
Hardness-to-randomness

- reconstructive PRGs

\[ G_f^e : \ell(n) \to T(n) \]

PRG

distinguisher

T(n)
Hardness-to-randomness

› reconstructive PRGs

PRG based on hard function $f$

$G^f_{e(n)} \rightarrow T(n)$
Hardness-to-randomness

› reconstructive PRGs

distinguisher yields efficient procedure for $f$

$G_f^\ell(n) \rightarrow T(n)$
Hardness-to-randomness

- derandomization from PRGs

- replace $T(n)$ coins with $\ell(n)$ coins, enumerate in time $2^{\ell(n)}$
- textbook results [NW’94, IW’99, STV’01, SU’01, Uma’03]:

$$\text{DTIME}[2^n] \not\subset \text{SIZE}[s]$$

$$\text{PRG with stretch } \approx s$$

$$\text{BPP in DTIME } \approx 2^{s^{-1}(n)}$$
An informidable-looking barrier

- why experts might think that $c<2$ requires “new techniques”

- Textbook approach: To derandomize time-$T$ algs, construct a PRG that fools non-uniform size-$T$ circuits

- Such a PRG requires a seed of length $\log(T)$

- The derandomization time is $2^{\log(T)} \cdot T(n) \geq T(n)^2$
Tracking the non-uniformity

- modeling distinguishers, carefully

who is this distinguisher?

T(n)
Tracking the non-uniformity

- modeling distinguishers, carefully

- For any $L \in \text{BPTIME}[T]$, our focus is:
  
  Does the probabilistic machine $M_L$ behave the same on $G^f(\mu_{\ell(n)}) \& \mu_{T(n)}$ for all inputs $x$?

- Distinguisher is $M_L$ with an arbitrary fixed input $x$
Tracking the non-uniformity

⇒ modeling distinguishers, carefully

⇒ Textbook distinguisher:
   
   Non-uniform circuit of size $T$

⇒ Our pivotal observation:

   Distinguisher is a time-$T$
   machine with $|n| \ll T$ bits of non-uniformity
Why is this helpful?

- fooling small non-uniformity with small seed length

- non-uniformity is $n \ll T(n)$

- exists non-explicit PRG with seed length $\log(n) \ll \log(T)$!

- opens the door to derandomization in time $n \cdot T(n)$

  - we just to make this PRG explicit, under assumptions
Proof sketch for Thm 1
main ideas & some parameters
Hardness-to-randomness

- reconstructive PRGs

distinguisher yields efficient procedure for $f$

$G^f$: $\ell(n)$ $\rightarrow$ $T(n)$
Reconstruction overhead

› and its discontents

› “Reconstructive” procedure for f as oracle machine

\[ \text{Reconst}^{\text{DISTINGUISHER}}(x) = f(x) \]

› Reconstruction overhead is the main bottleneck

› Inefficient reconstruction

⇒ inefficient procedure for f

⇒ stronger hardness hypothesis
Reconstruction overhead

› and its discontents

› Best known overhead [Uma’03]:
  
  distinguisher in time $T \Rightarrow$ procedure for $f$ in time $T^{O(1)}$

› ... so we need to assume $f$ is hard for time $T^{O(1)}$

› ... since the PRG computes $f \Rightarrow$ PRG takes time $\geq T^{O(1)}$

› Derandomization with large polynomial overhead
Our goal is to avoid this overhead

Two ideas in the proof:

1. Compose two PRGs computable in time $\approx T^{1.01}$
2. Use a tiny & super-exponentially-hard truth-table
Composing two “low-cost” PRGs

- each computable in time $\approx T^{1.01}$

- Focus on $T(n) = n^c$ for simplicity
Composing two “low-cost” PRGs

› each computable in time $\approx T^{1.01}$

› Focus on $T(n) = n^c$ for simplicity
Composing two “low-cost” PRGs

- the “inner” PRG

- Small seed, but small output length

- **Obs:** Small output length \( \Rightarrow \) small reconst. overhead

Nisan-Wigderson PRG

\[ 1.01 \cdot \log(n) \rightarrow n^\epsilon \]
Composing two “low-cost” PRGs

› the “outer” PRG

› Large output length, but large seed

› **Obs:** $\text{OWF} \Rightarrow \text{crypto PRG} \Rightarrow \text{near-linear time PRG}^1$

  › we’ll use it as a non-crypto PRG, i.e. distinguisher is weaker

---

$n^\varepsilon \mapsto 2n^\varepsilon$ computable in time $n^{O(\varepsilon)}$, and “compose” it $\approx n^c$ times to extend output to $n^c$
Tiny, superexp-hard truth-table

\[ 1.01 \cdot \log(n) \]

\( n^\epsilon \)

\( n^c \)

\( \text{NW} \)

\( \text{crypto} \)
Tiny, superexp-hard truth-table

› a parametric overview

\[ 1.01 \cdot \log(n) \]

\[ n^\epsilon \]

\[ n^c \]
Tiny, superexp-hard truth-table

› a parametric overview

› Our distinguisher uses
  1. time \( n^c \)
  2. advice \( n \)
Tiny, superexp-hard truth-table

▷ a parametric overview

▷ We use $|f| = n^{1+O(\varepsilon)}$ that is hard for

1. time $n^{1.01 \cdot c}$
2. advice $n + |f|^{0.99}$
We use $|f| = n^{1+O(\varepsilon)}$ that is hard for

1. time $n^{1.01 \cdot c} \approx 2^{c \cdot \ell}$
2. advice $n+|f|^{0.99} \approx 2^{0.99 \cdot \ell}$

$n^{1+O(\varepsilon)} = 2^\ell$
Hardness hypothesis

- generalizing classical hardness hypotheses

- Our derandomization uses a tiny truth-table with super-exponential time complexity

- Our hardness hypothesis (for \( k \approx c \))

  \[ f \in \text{DTIME}[2^k \cdot n] \text{ and hard for } \text{DTIME}[2^{0.99k} \cdot n]/2^{0.99n} \]
Hardness hypothesis

- generalizing classical hardness hypotheses

- Natural “scale-up” of classical hypotheses:
  \[ f \in \text{DTIME}[2^{k \cdot n}] \setminus \text{DTIME}[2^{0.99k \cdot n}/2^{0.99n}] \quad [\text{this work}] \]
  \[ f \in \text{DTIME}[2^n] \setminus \text{SIZE}[2^{-0.1n}] \quad [\text{IW’99}] \]

- Both interpreted as non-uniformly-strong time-hierarchy
A last small gap

› final running-time of derandomization?

› we’ll have $n^{1.01}$ seeds (for the inner PRG NW)

› if on each seed PRG is computable in time $\approx T$,
  then we get derandomization in time $O(n^{1.01} \cdot T)$
A last small gap

- we didn’t really see that the PRG is linear-time computable yet

- PRG isn’t computable in time \( \approx T \) on each seed!

- … but it’s computable on all seeds in amortized time \( \approx T \)
  - suffices for derandomization

- … and we can relax the hypothesis, and only require \( f \) to be computable on all inputs in amortized time \( \approx T \)
A last small gap

- we didn’t really see that the PRG is linear-time computable yet

- Assuming OWFs, tight equivalence of
  1. batch-computable PRGs
  2. hard functions with small amortized time-complexity

- The “right” objects to study in hardness-to-randomness
  - the tightness is significant for superfast derandomization
Zooming-in on the overhead

- **Thm 1’**: Reduce overhead to $n^{1.01} \cdot T$, extend to $T(n) = n^{\omega(1)}$
- **Thm 2**: Assuming #NSETH, overhead of $n^{.99} \cdot T$ is optimal
- **Thm 3**: Average-case derandomization with effectively no overhead at all (only $n^\epsilon$, below lower bound)

a reminder of additional results in the paper
Simplifying a well-known PRG paradigm
via quantified derandomization
Well-known PRG paradigm

- underlies [HILL’99, BSW’03, ..., DMOZ’20]

- Paradigm for constructing PRGs
- Based on composition of two algorithms
- We will show: Any such composition can be viewed & analyzed in a very simple way
Well-known PRG paradigm

› underlies [HILL'99, BSW'03, ..., DMOZ'20]

1. a pseudoentropy generator (PEG)
Well-known PRG paradigm

› underlies [HILL’99, BSW’03, ..., DMOZ’20]

1. a pseudoentropy generator (PEG)
Well-known PRG paradigm

- underlies [HILL'99, BSW'03, ..., DMOZ'20]

1. a pseudoentropy generator (PEG)

\[ \ell(n) \xrightarrow{G_f} T(n) \]

wow, this has high entropy!
Well-known PRG paradigm

- underlies [HILL'99, BSW'03, ..., DMOZ'20]

1. a pseudoentropy generator (PEG)
2. a randomness extractor
Well-known PRG paradigm

› underlies [HILL’99, BSW’03, ..., DMOZ’20]

1. a pseudoentropy generator (PEG)

2. a randomness extractor

all the entropy “extracted” to almost-uniform string

\[
\text{Ext: } T(n) + \ell(n) \rightarrow T'(n)
\]
Well-known PRG paradigm

› underlies [HILL'99, BSW'03, ..., DMOZ'20]

› PRG: \[ G(s_1, s_2) = \text{Ext}( \text{PEG}(s_1), s_2 ) \]

› Intuition: If \( \text{PEG}(s_1) \) looks entropic, then \( \text{Ext}( \text{PEG}(s_1), s_2 ) \) should look random

› Good extractors are known, so we “just” need a PEG, and to make the idea above for composition work
The main challenge

and the approach of [DMOZ'20]

Constructing PEGs challenging, only few known ideas

The approach of [DMOZ'20]:

1. Construct “metric” (weak) PEG from hard funcs; but
2. Highly non-trivial composition with extractor
   › requires PEG to fool strong & non-standard class
Easier & more general paradigm

- error-reduction then quantified derandomization

- PRG: $G(s_1, s_2) = \text{Ext}(\text{PEG}(s_1), s_2)$
Easier & more general paradigm

PrG: \( G(s_1, s_2) = \text{Ext}(\text{PEG}(s_1), s_2) \)

- error-reduction then quantified derandomization
- PRG: \( G(s_1, s_2) = \text{Ext}(\text{PEG}(s_1), s_2) \)
- We show a simpler analysis of the composition
- ... which paves way to using a weaker “inner” generator
Easier & more general paradigm

- error-reduction then quantified derandomization

- PRG: \( G(s_1, s_2) = \text{Ext}(\text{PEG}(s_1), s_2) \)

- Our analysis involves two steps:
  1. (non-standard) error reduction, using Ext
  2. quantified derandomization, using the “inner” generator
Easier & more general paradigm

- error-reduction then quantified derandomization

- PRG: \( G(s_1, s_2) = \text{Ext}(\text{QD}(s_1), s_2) \)

- Generators for QD are equivalent to metric ("weak") PEGs

- ... but no need now for generator to fool a stronger class

- And analysis of composition is very simple
Easier & more general paradigm

- error-reduction then quantified derandomization

- Prop: Any construction that can be analyzed as “extract from a pseudo-entropic string” can also be analyzed as “non-standard error-reduction and QD”
  - (converse not known)

- To materialize this approach we need a generator for QD
  - not an easy task, in general
Derandomization with overhead $c \in \{2,3,4\}$

- easy & versatile proof for superfast derandomization

- Prop: Assuming $\text{DTIME}[2^n] \not\subseteq \text{iO-MASIZE}[2^{0.99n}]$, there exists a very simple & fast generator for QD

- Cor: New and easy proof for main result of [DMOZ'20]
  - pf: combine QD construction with easy high-level analysis

- Proof versatile, extends to cubic/quartic derandomization from hardness only for NSIZE (details in paper)
4 Key takeaways
results to remember
Take-home message

1. Derandomization with overhead $\approx n \cdot T(n)$ possible under reasonable assumptions

2. Simple & intuitive proofs yield conditional derandomization with overhead $c \in \{1,2,3,4\} + \varepsilon$

3. Broadening the theoretical basis for superfast derandomization
Take-home open questions!

› a sample of some directions

1. Is the overhead of $n \cdot T$ optimal?
   › evidence without #NSETH

2. **Search-to-decision** with minimal overhead?
   › true given OWFs, show unconditional reduction

3. Superfast derandomization from *classical hypotheses*?
   › no crypto, no hardness for MASIZE/NSIZE
Thank you!

⇒ derandomization with essentially no overhead
⇒ simple & intuitive proofs relying on new high-level insights