Randomness Through the Lens of Polynomials and Compression

Habilitation thesis  4dec2023  Sylvain Perifel
Connecting people®

(Coauthors of the articles in the manuscript)

Philippe Moser
Mahsa Shirmohammadi
James Worrell
Rémi de Verclos
Hervé Fournier
Guillaume Lagarde
Guillaume Malod
Elvira Mayordomo
Amir Yehudayoff
Nikhil Balaji
Zeev Dvir

+ all other collaborations!
1. 1000 times a dice: a random sequence

666...666 really random?
Randomness?

1. 1000 times a dice: a random sequence

666…666 really random?

How to define randomness?
Randomness?

1. 1000 times a dice: a random sequence

\[666...666\text{ really random?}\]

How to define randomness?

2. Does randomness help algorithms?
Outline

Definitions of randomness

- **Overview**: compressors
- **Focus**: one-bit catastrophe
- **Perspective**: normality

Randomness in algorithms

- **Overview**: lower bounds
- **Focus**: cyclotomic IT
- **Perspective**: the power of PIT
Overview:
Efficient compression

Three kinds of compressors:
• Pushdown transducers
• Polylogspace compressors
• LZ’78
Compressors & randomness

Random

= “impredictible”
= no short description
= incompressible

for some class of machines
Compressors & randomness

Random

= “impredictible”

= no short description

= incompressible for some class of machines

\[ f(x) = y_1 \ y_2 \ y_3 \ y_4 \ \ldots \]

\[ x_1 \ x_2 \ x_3 \ \ldots \ x_{n-1} \ x_n \]
Compressors & randomness

Random

= “impredictible”
= no short description
= incompressible for some class of machines

Compressor:

\[ f \] must be one-to-one

\[
\begin{align*}
\text{Input} & : & x_1 & x_2 & x_3 & \ldots & x_{n-1} & x_n \\
\text{Output} & : & f(x) & = & y_1 & y_2 & y_3 & y_4 & \ldots
\end{align*}
\]
Compressors & randomness

Random

= “impredictible”

= no short description

= incompressible for some class of machines

Input

\[ x_1 \ x_2 \ x_3 \ \cdots \ x_{n-1} \ x_n \]

Output

\[ f(x) = y_1 \ y_2 \ y_3 \ y_4 \ \cdots \]

Compressor:

\[ f \text{ must be one-to-one} \]

Compression ratio

\[ \rho(x) = \frac{|f(x)|}{|x|} \]
Randomness for resource-limited compressors

Pushdown compressors: finite state automata with a stack
Randomness for resource-limited compressors

**Pushdown compressors**: finite state automata with a stack

**Structured documents** (XML)
Randomness for resource-limited compressors

**Pushdown compressors**: finite state automata with a stack

**Polylogspace compressors**: machines with very limited memory

Structured documents (XML)
Randomness for resource-limited compressors

**Pushdown compressors:** finite state automata with a stack

**Polylogspace compressors:** machines with very limited memory

Structured documents (XML)

Massive data

$(\log n)^{O(1)}$
Randomness for resource-limited compressors

**Pushdown compressors**: finite state automata with a stack

**Polylogspace compressors**: machines with very limited memory

How do they compare with a general-purpose compressor like **LZ’78**?

- Understand the strengths and weaknesses of the models
- Understand randomness for them

Structured documents (XML)

Massive data
Randomness for resource-limited compressors

**Pushdown compressors**: finite state automata with a stack

**Polylogspace compressors**: machines with very limited memory

How do they compare with a general-purpose compressor like **LZ’78**?

- Understand the strengths and weaknesses of the models
- Understand randomness for them

**Theorem [MMP’11]** These three models are pairwise incomparable.
Randomness for resource-limited compressors

**Pushdown compressors**: finite state automata with a stack

**Polylogspace compressors**: machines with very limited memory

How do they compare with a general-purpose compressor like LZ’78?

- Understand the strengths and weaknesses of the models
- Understand randomness for them

**Theorem [MMP’11]** These three models are pairwise incomparable.

- LZ’78 can compress repetitions even far apart
- Pushdown transducers can compress palindromes
- Polylogspace compressors can compress simple enumerations of words
Focus: Lempel-Ziv

- Variants used in gzip&gif
- Widely studied
LZ'78  Compression of a word $w$

Parsing = cutting $w$ into blocks

$0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1$ 

Next block: shortest prefix not in the dictionary yet

Compression of a block: 
a pointer to its predecessor + one bit

LZ'78  Compression of a word \( w \)

**Parsing** = cutting \( w \) into blocks

\[
\begin{array}{l}
\varepsilon \\
0 \\
\varepsilon \\
\varepsilon \\
\end{array}
\]

**Dictionary:**  \( \varepsilon \)

**Compression:**

**Parsing tree**

Next block: shortest prefix not in the dictionary yet

Compression of a block: a pointer to its predecessor + one bit
**LZ'78** Compression of a word $w$

**Parsing** = cutting $w$ into blocks

\[
\begin{array}{c}
\varepsilon & 0 & 01 & 0111 & 1 & 010 & 0110 & 11 & 111
\
0 & 1
\end{array}
\]

Dictionary: \( \varepsilon \) 0

Compression: (0,0)

**Next block:** shortest prefix not in the dictionary yet

**Compression** of a block: a pointer to its predecessor + one bit
 Parsing = cutting \( w \) into blocks

\[ \varepsilon \quad 0 \quad 01 \quad 011 \quad 1 \quad 010 \quad 0110 \quad 11 \quad 111 \]

\[ 0 \quad 1 \quad 2 \]

Dictionary: \( \varepsilon \quad 0 \quad 01 \)

Compression: \( (0,0) \quad (1,1) \)

Next block: shortest prefix not in the dictionary yet

Compression of a block:

a pointer to its predecessor + one bit

LZ'78 Compression of a word \( w \)
LZ'78  Compression of a word $w$

Parsing = cutting $w$ into blocks

$$\varepsilon \quad 0 \quad 01 \quad 011 \quad 1 \quad 0 \quad 1 \quad 0 \quad 011 \quad 0 \quad 11 \quad 0 \quad 11 \quad 1$$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>01</td>
<td>011</td>
</tr>
</tbody>
</table>

Dictionary:

Compression:

$(0,0)$  $(1,1)$  $(2,1)$

Next block: shortest prefix not in the dictionary yet

Compression of a block:

a pointer to its predecessor + one bit
**LZ’78**  Compression of a word \( w \)

**Parsing** = cutting \( w \) into blocks

```
ε  0  01  011  1  01  0110  11  111
```

- **Dictionary:** \( ε \) 0 01 011 1
- **Compression:** (0,0) (1,1) (2,1) (0,1)

**Next block:** shortest prefix not in the dictionary yet

**Compression of a block:** a pointer to its predecessor + one bit
**LZ'78** Compression of a word $w$

**Parsing** = cutting $w$ into blocks

**Dictionary:**

$\epsilon$ 0 01 011 1 010

**Compression:**

$(0,0) (1,1) (2,1) (0,1) (2,0)$

**Next block:** shortest prefix not in the dictionary yet

**Compression** of a block:

a pointer to its predecessor + one bit
LZ'78 Compression of a word $w$

Parsing = cutting $w$ into blocks

Dictionary: $\varepsilon$ 0 01 011 1 010 0110

Compression: (0,0) (1,1) (2,1) (0,1) (2,0) (3,0)

Next block: shortest prefix not in the dictionary yet

Compression of a block:
a pointer to its predecessor + one bit

Parsing tree

$\varepsilon$

$0$
Parsing tree

$1$

$01$

$010$

$011$

$0110$
LZ'78 Compression of a word $w$

Parsing = cutting $w$ into blocks

Dictionary:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>01</th>
<th>011</th>
<th>1</th>
<th>010</th>
<th>0110</th>
<th>11</th>
</tr>
</thead>
</table>

Compression:

<table>
<thead>
<tr>
<th></th>
<th>(0,0)</th>
<th>(1,1)</th>
<th>(2,1)</th>
<th>(0,1)</th>
<th>(2,0)</th>
<th>(3,0)</th>
<th>(4,1)</th>
</tr>
</thead>
</table>

Next block: shortest prefix not in the dictionary yet

Compression of a block: a pointer to its predecessor + one bit
**LZ’78** Compression of a word $w$

**Parsing** = cutting $w$ into blocks

```
Parsing tree
```

```
Dictionary:  
\[ \varepsilon, 0, 01, 011, 1, 010, 0110, 11, 111 \]

Compression:  
\[ (0,0), (1,1), (2,1), (0,1), (2,0), (3,0), (4,1), (7,1) \]
```

**Next block**: shortest prefix not in the dictionary yet

**Compression** of a block: a pointer to its predecessor + one bit
Extremal cases

Optimal compression, $\Theta(\sqrt{n})$ bocks:

$$\text{Pref}(x) = x_0 \ x_0x_1 \ x_0x_1x_2 \ x_0x_1x_2x_3 \ x_0x_1x_2x_3x_4$$

Parsing tree
Extremal cases

Optimal compression, $\Theta(\sqrt{n})$ bocks:

$$\text{Pref}(x) = x_0 \ x_0x_1 \ x_0x_1x_2 \ x_0x_1x_2x_3 \ x_0x_1x_2x_3x_4$$

Worst compression (incompressible, $\Theta(n/\log n)$ blocks):

0 1 00 01 10 11 000 001 010 011 ...
A one-bit catastrophe...

Is LZ’78 “robust” w.r.t. small changes?
A one-bit catastrophe...

Look at that! I compressed a file from 1 TB to 100 MB

There is a typo in your original file. Ah, let me correct it...

What? The new compression has size 900 GB! Is it really possible?

**One-bit catastrophe question**

Is it possible to find a word \( \omega \) such that

- \( \omega \) compressible
- \( \omega \) not compressible?

© Guillaume Lagarde
A one-bit catastrophe...

Theorem [LP’18]

\[ \exists w \in \{0,1\}^\mathbb{N} : \begin{cases} \rho(w) = 0 \\ \rho(0w) \geq \frac{1}{6075} \end{cases} \]
A one-bit catastrophe...

**Theorem [LP'18]**

\[ \exists w \in \{0,1\}^\mathbb{N} : \]

\[
\begin{cases}
\rho(w) = 0 \\
\rho(0w) \geq \frac{1}{6075}
\end{cases}
\]

**Step 1 – a “weak” catastrophe**

\[ \#\text{blocks}(w) = \Theta(\sqrt{n}) \quad \text{(near optimal compression)} \]

\[ \#\text{blocks}(0w) = \Theta(n^{3/4}) \]

\[ w \simeq \text{Pref}(x) \]

Idea: “desynchronize” the parsing of 0w (different history)

Bad compression for 0w if \( x \) has a lot of different factors

\[ x \] is a de Bruijn word
A one-bit catastrophe...

**Theorem [LP’18]**

\[
\exists w \in \{0,1\}^\mathbb{N} : \begin{cases} 
\rho(w) = 0 \\
\rho(0w) \geq \frac{1}{6075}
\end{cases}
\]

**Step 1 – a “weak” catastrophe**

\[
\#\text{blocks}(w) = \Theta(\sqrt{n}) \quad \text{(near optimal compression)}
\]

\[
\#\text{blocks}(0w) = \Theta(n^{3/4})
\]

\[w \simeq \text{Pref}(x)\]

Idea: “desynchronize” the parsing of \(0w\) (different history)

Bad compression for \(0w\) if \(x\) has a lot of different factors

\[x \text{ is a de Bruijn word}\]

... but \(0w\) still compressible
Step 2 – start from \( w \) less compressible:
concatenate independent weak catastrophes

- construct random independent
  “de Bruijn style” words \( x_1, \ldots, x_N \)
- for well-chosen parameters,
  \[ \text{Pref}'(x_1) \cdots \text{Pref}'(x_N) \]
is a “true” catastrophe on finite words
A one-bit catastrophe...

**Theorem [LP’18]**

\[ \exists w \in \{0,1\}^\mathbb{N} : \begin{cases} 
\rho(w) = 0 \\
\rho(0w) \geq \frac{1}{6075} 
\end{cases} \]

**Step 3** – concatenate independent true catastrophes
- construct infinitely many independent finite catastrophes
- for well-chosen parameters, their concatenation is the (infinite) one-bit catastrophe
... but not a tragedy

Theorem [LP’18]

\[ \rho(0w) \leq 3 \sqrt{2} \sqrt{\rho(w) \cdot \log |w|} \]

\(0w\) can become incompressible only if \(w\) is “poorly” compressible

\[ \rho(w) = o\left(\frac{1}{\log |w|}\right) \implies \rho(0w) = o(1) \]
... but not a tragedy

**Theorem [LP’18]**

\[
\rho(0w) \leq 3\sqrt{2} \sqrt{\rho(w) \cdot \log |w|}
\]

0w can become incompressible only if \( w \) is “poorly” compressible

\[
\rho(w) = o\left(\frac{1}{\log |w|}\right) \implies \rho(0w) = o(1)
\]

- \#junction blocks \( \leq \# \text{green blocks} \)
- \#offset blocks \( \leq \# \text{different factors in green blocks} \)
  \[\leq 2\sqrt{|w| \cdot \#\text{blocks}(w)}\]
- \[\leq \#\text{blocks}(w) \text{ factors of each length } i\]
  \[\implies \text{“large” offset blocks}\]
In a “random” sequence you expect to find:

- almost “as many” 0’s and 1’s
- same frequency (1/4) of 00, 01, 10 and 11
- ...

**Perspective:** normality
Normality

Definition

[Borel'09]

An infinite word $w \in \{0,1\}^\mathbb{N}$ is simply normal if the frequency of 0's and 1's is 1/2.

It is normal if all finite words $u \in \{0,1\}^k$ of same size $k$ appear in $w$ with the same frequency $2^{-k}$

Examples

Champernowne (1933): 0 1 10 11 100 101 110 111 1000 1001... (integers in base 2)

Copeland&Erdös (1946): 10 11 101 111 1011 1101 10001... (primes)
Normality

Definition

[Borel'09]

An infinite word $w \in \{0,1\}^\mathbb{N}$ is simply normal if the frequency of 0's and 1's is 1/2.

It is normal if all finite words $u \in \{0,1\}^k$ of same size $k$ appear in $w$ with the same frequency $2^{-k}$

Examples

Champernowne (1933): 0 1 10 11 100 101 110 111 1000 1001… (integers in base 2)

Copeland&Erdös (1946): 10 11 101 111 1011 1101 10001… (primes)

Proposition

[SS'72+DLLM'04] $w$ is normal iff it is incompressible by finite state transducers (normality = randomness for finite automata)
Open questions

Theorem (Borel'09)  Almost all real numbers are normal.

Conjecture (Borel'50)  Irrational algebraic numbers are normal.
Open questions

Theorem (Borel’09) Almost all real numbers are normal.
Conjecture (Borel’50) Irrational algebraic numbers are normal.

However...

Is $\sqrt{2}$ simply normal in base 2?

Open since at least Borel 1950...

Best result so far: $\geq \sqrt{n}$ zeroes and ones among the first $n$ bits [BBCP’04]

However...

$\sqrt{2} \times \sqrt{2} = 2$

$$= 1.011010100000100111100110011$$

$$\times 1.011010100000100111100110011$$

$$= 1.11111111111111111111111111110100110...$$

$\geq n$ ones
Open questions

Theorem (Borel'09) Almost all real numbers are normal.

Conjecture (Borel'50) Irrational algebraic numbers are normal.

However...

Is $\sqrt{2}$ simply normal in base 2?

Open since at least Borel 1950...

Best result so far: $\geq \sqrt{n}$ zeroes and ones among the first $n$ bits [BBCP'04]

Open for only 30 years: Ehrenfeucht-Mycielski'92

“Pseudo-random” sequence EM

Is EM simply normal?

[KS07] frequency of 0s and 1s $\geq 1/4$
Some algorithms use randomness (e.g. quicksort).

Are there always deterministic algorithms that are “as efficient”?

Randomness in algorithms

- Some algorithms use randomness (e.g. quicksort)
- Are there always deterministic algorithms that are “as efficient”? 

Derandomization
Complexity classes

- **P**: "Efficient" deterministic algorithms
- **BPP**: "Efficient" randomized algorithms
- **EXP**: Naive derandomization of BPP
Arithmetic circuits & PIT aka Circuix the Gaul

(still resists derandomization)

Arithmetic circuits

Compute "formal" polynomials like

\[ p(x, y) = 1 + x^2 - xy + y^3 \]

\[ q(x) = (2 + x)^2^n \]
Arithmetic circuits & PIT aka Circuiix the Gaul

Arithmetic circuits

Compute “formal” polynomials like

\[ p(x, y) = 1 + x^2 - xy + y^3 \]

PIT

- **Input:** a circuit \( C \) computing a polynomial \( p \)
- **Question:** \( p = 0 \)?

\[ p(x, y) = 1 + x^2 - xy + y^3 \quad \text{and} \quad q(x) = (2 + x)^2^n \]
Algorithms

Naive deterministic algorithm:
Iteratively compute (expand)
the polynomials at all gates
Algorithms

Naive deterministic algorithm:
Iteratively compute (expand) the polynomials at all gates

But possibly:
- exponential degree
- coefficients of exponential bitsize
- exponential number of monomials

$\rightarrow$ exponential time
Algorithms

Naive deterministic algorithm:
Iteratively compute (expand) the polynomials at all gates

**But** possibly:
- exponential degree
- coefficients of exponential bitsize
- exponential number of monomials

→ exponential time

Schwartz-Zippel lemma:
\[
p \neq 0 \implies \Pr_{(s_1, \ldots, s_k) \in S^k} (p(s_1, \ldots, s_k) = 0) \leq \frac{\deg(p)}{|S|}
\]
Algorithms

Naive deterministic algorithm:
Iteratively compute (expand) the polynomials at all gates

But possibly:
• exponential degree
• coefficients of exponential bitsize
• exponential number of monomials

→ exponential time

Schwartz-Zippel lemma:
\[ p \neq 0 \implies \Pr_{(s_1, \ldots, s_k) \in S^k} (p(s_1, \ldots, s_k) = 0) \leq \frac{\deg(p)}{|S|} \]

Randomized algorithm
• \( S = \{0, \ldots, 2^{n^2}\} \)
• choose \( m \) and \( s_1, \ldots, s_k \in S \) at random
• evaluate \( p(s_1, \ldots, s_k) \mod m \) gate by gate
• accept iff \( p(s_1, \ldots, s_k) \equiv 0 \mod m \)

→ polynomial time
Algorithms

Naive deterministic algorithm:
Iteratively compute (expand) the polynomials at all gates

But possibly:
- exponential degree
- coefficients of exponential bitsize
- exponential number of monomials

\[ \rightarrow \text{exponential time} \]

Schwartz-Zippel lemma:
\[ p \neq 0 \implies \Pr_{(s_1, \ldots, s_k) \in S^k} (p(s_1, \ldots, s_k) = 0) \leq \frac{\deg(p)}{|S|} \]

Randomized algorithm
- \[ S = \{0, \ldots, 2^{n^2}\} \]
- choose \( m \) and \( s_1, \ldots, s_k \in S \) at random
- evaluate \( p(s_1, \ldots, s_k) \mod m \) gate by gate
- accept iff \( p(s_1, \ldots, s_k) \equiv 0 \mod m \)

\[ \rightarrow \text{polynomial time} \]

Derandomize?
PIT vs lower bounds

Toy reasoning

One (natural) way to derandomize PIT:

- hitting set $H_s \subset \mathbb{N}^k$ for circuits of size $s$

$p \neq 0$ computed by $C$ of size $s \implies \exists h \in H_s : p(h) \neq 0$
PIT vs lower bounds

Toy reasoning

One (natural) way to derandomize PIT:

hitting set $H_s \subset \mathbb{N}^k$ for circuits of size $s$

\[ p \neq 0 \text{ computed by } C \text{ of size } s \implies \exists h \in H_s : p(h) \neq 0 \]
Toy reasoning

One (natural) way to derandomize PIT:

hitting set $H_s \subset \mathbb{N}^k$ for circuits of size $s$

$p \neq 0$ computed by $C$ of size $s \implies \exists h \in H_s : p(h) \neq 0$

Then $p(x_1, \ldots, x_k) = \prod_{h \in H_s} (x_1 - h_1)$

has no circuits of size $s$

lower bound
PIT vs lower bounds

Toy reasoning

One (natural) way to derandomize PIT:

hitting set $H_s \subset \mathbb{N}^k$ for circuits of size $s$

$p \neq 0$ computed by $C$ of size $s$ $\implies$ $\exists h \in H_s : p(h) \neq 0$

Then $p(x_1, \ldots, x_k) = \prod_{h \in H_s} (x_1 - h_1)$

has no circuits of size $s$

Known links

[KI’03] PIT in P implies **circuit lower bound**

for NEXP or permanent

[Yao, Nisan, Wigderson, Sudan…]

EXP $\not\subset$ SIZE($2^{en}$) $\implies$ BPP = P
Complexity classes

- \( P \)
- \( BPP \)
- \( EXP \)
- \( NEXP \)
Overview:
Lower bounds

• Lower bounds and derandomization are connected
• Understanding weaknesses of the computation model / hardness of the polynomial
Theorems

[FPV’15] “Explicit” polynomials with no circuits of size $n^k$

[DMPY’12] Multilinear ABPs compute polynomials that require superpolynomial formulas

[LMP’19] Determinant/permanent require non-commutative unambiguous circuits of exponential size (+PIT)
Theorems

[FPV’15] “Explicit” polynomials with no circuits of size $n^k$

[MPY’12] Multilinear ABPs compute polynomials that require superpolynomial formulas

[LMP’19] Determinant/permanent require non-commutative unambiguous circuits of exponential size (+PIT)

Diagonalisation in MA if $\text{per} \in \text{VP}$
Theorems

[FPV'15] "Explicit" polynomials with no circuits of size $n^k$

[DMPY'12] Multilinear ABPs compute polynomials that require superpolynomial formulas

[LMP'19] Determinant/permanent require non-commutative unambiguous circuits of exponential size (+PIT)

Diagonalisation in MA if $\text{per} \in \text{VP}$
Theorems

[FPV'15] “Explicit” polynomials with no circuits of size $n^k$

[MPY'12] Multilinear ABPs compute polynomials that require superpolynomial formulas

[LMP'19] Determinant/permanent require non-commutative unambiguous circuits of exponential size (+PIT)

Diagonalisation in MA if $\text{per} \in \mathsf{VP}$
Theorems

[FPV'15] “Explicit” polynomials with no circuits of size $n^k$.

[DMPY'12] Multilinear ABPs compute polynomials that require superpolynomial formulas.

[LMP'19] Determinant/permanent require non-commutative unambiguous circuits of exponential size (+PIT).

Diagonalisation in MA if $\text{per} \in \text{VP}$.

The rank method using arc-partitions.
Theorems

[FPV’15] “Explicit” polynomials with no circuits of size $n^k$

[MPY’12] Multilinear ABPs compute polynomials that require superpolynomial formulas

[LMP’19] Determinant/permanent require non-commutative unambiguous circuits of exponential size (+PIT)

Diagonalisation in MA if per ∈ VP

The rank method using arc-partitions

Generalisation of Nisan’s result on ABPs
Focus:
Cyclotomic IT

- PIT: constant-free circuits
- What if we allow particular complex constants?
Identity testing in cyclotomic fields:

**Input:** $n$ in binary, C computing $g(x)$

**Question:** $g(\zeta_n) = 0$?

\[ \zeta_n = e^{\frac{2\pi i}{n}} \text{ complex } n\text{-th root of unity} \]
Identity testing in cyclotomic fields:

Input: $n$ in binary, C computing $g(x)$

Question: $g(\zeta_n) = 0$?

Harder than PIT...

[CTV'10] places CIT in $\text{CH}$
CIT

\[ \zeta_n = e^{i \frac{2\pi}{n}} \] complex \( n \)-th root of unity

Identity testing in cyclotomic fields:

**CIT**

**Input:** \( n \) in binary, \( C \) computing \( g(x) \)

**Question:** \( g(\zeta_n) = 0? \)

Harder than PIT...

[CTV'10] places CIT in \( \text{CH} \)

[BPSW'21] CIT in BPP (assuming GRH) or in \( \text{coNP} \) (unconditionally)
Complexity classes

- $P$
- $BPP$
- $coNP$
- $CH$
- $EXP$
- $NEXP$
Algorithm

Theorem \( \begin{cases} \text{CIT is in coNP} \\ \text{CIT is in BPP under GRH} \end{cases} \)

**Idea:** compute in the field \( \mathbb{F}_p \) having a primitive \( n \)-th root of unity (true if \( p \equiv 1 \mod n \))

**Algorithm:**

1. Randomly pick \( p \) of polynomial bitsize such that \( p \equiv 1 \mod n \)
2. Find a primitive root:
   - randomly pick \( h \in \mathbb{F}_p \) such that \( h^{p^{-1}} \neq 1 \) for small \( q \)'s
   - then \( \omega_n = h^{p^{-1}} n \) is a primitive \( n \)-th root w.h.p.
3. Evaluate \( g(\omega_n) \) in \( \mathbb{F}_p \): for most \( p \) and \( h \),
   \[ \bar{g}(\omega_n) = 0 \iff g(\zeta_n) = 0 \]
Algorithm

Theorem \begin{align*}
&\text{CIT is in coNP} \\
&\text{CIT is in BPP under GRH}
\end{align*}

**Idea:** compute in the field $\mathbb{F}_p$ having a primitive $n$-th root of unity (true if $p \equiv 1 \mod n$)

**Algorithm:**

1. Randomly pick $p$ of polynomial bitsize such that $p \equiv 1 \mod n$
2. Find a primitive root:
   - randomly pick $h \in \mathbb{F}_p$ such that $h^{\frac{p-1}{q}} \neq 1$ for small $q$'s
   - then $\omega_n = h^{\frac{p-1}{n}}$ is a primitive $n$-th root w.h.p.
3. Evaluate $g(\omega_n)$ in $\mathbb{F}_p$:
   - for most $p$ and $h$,
   - $\bar{g}(\omega_n) = 0 \iff g(\zeta_n) = 0$

**Primes in arithmetic progressions under GRH**
Algorithm

Theorem
\[ \begin{align*}
\text{CIT is in coNP} \\
\text{CIT is in BPP under GRH}
\end{align*} \]

Idea: compute in the field \( \mathbb{F}_p \) having a primitive \( n \)-th root of unity (true if \( p \equiv 1 \mod n \))

Algorithm:

1. Randomly pick \( p \) of polynomial bitsize such that \( p \equiv 1 \mod n \)
2. Find a primitive root:
   - randomly pick \( h \in \mathbb{F}_p \) such that \( h^{\frac{p-1}{q}} \neq 1 \) for small \( q \)'s
   - then \( \omega_n = h^{\frac{p-1}{n}} \) is a primitive \( n \)-th root w.h.p.
3. Evaluate \( g(\omega_n) \) in \( \mathbb{F}_p \): for most \( p \) and \( h \),
   \[ \tilde{g}(\omega_n) = 0 \iff g(\zeta_n) = 0 \]

Primes in arithmetic progressions under GRH
\[ \forall 0 < i < n, \omega^i_n \neq 1 \]
• Derandomization of PIT out of reach…

• Could PIT be too “powerful”?
• Believed: PIT in P  (Implied by strong circuit lower bounds)
PIT

- Believed: PIT in P  (Implied by strong circuit lower bounds)
- Still consistent: PIT might be EXP-complete!
Complexity classes

- **P**
- **BPP**
- **coNP**
- **EXP**
- **NEXP**
- **CH**
PIT

- Believed: PIT in P (Implied by strong circuit lower bounds)
- Still consistent: PIT might be EXP-complete!

Still out of reach...
PIT

• Believed: PIT in P  (Implied by strong circuit lower bounds)

• Still consistent: PIT might be EXP-complete!

Conjecture:

PIT is not EXP-complete for "local reductions"

Still out of reach...
• Believed: PIT in P (Implied by strong circuit lower bounds)
• Still consistent: PIT might be EXP-complete!

Conjecture:
PIT is not EXP-complete for “local reductions”

Still out of reach...
PIT

- Believed: PIT in P  (Implied by strong circuit lower bounds)
- Still consistent: PIT might be EXP-complete!

Conjecture:

PIT is not EXP-complete for "local reductions"

(or SuccinctPIT is not 2EXP-complete)

polylog-time reductions

Still out of reach...
PIT

- **Believed**: PIT in P (Implied by strong circuit lower bounds)
- **Still consistent**: PIT might be EXP-complete!

Still out of reach...

**Conjecture:**

PIT is not EXP-complete for "local reductions"

(or **SuccinctPIT** is not 2EXP-complete)

- Keeps the essence of the initial question ("power of PIT"?)
- No relativization barrier
- No dramatic collapse
- Plenty of room for different methods

polylog-time reductions
SuccinctPIT

Possible tools
SuccinctPIT

Possible tools

• Resource-bounded Kolmogorov complexity / expanders: build a sequence of high complexity to go beyond P/poly
SuccinctPIT

Possible tools

• Resource-bounded Kolmogorov complexity / expanders: build a sequence of high complexity to go beyond P/poly

• Algebraic tools (polynomials)
SuccinctPIT

Possible tools

- Resource-bounded Kolmogorov complexity / expanders: build a sequence of high complexity to go beyond P/poly
- Algebraic tools (polynomials)
- Indirect diagonalisation along this way:
  - if succinctPIT is 2EXP-complete:
    - succinct circuits can compute large products
    - they can be used as advice to decide efficiently PIT
  ⟷ a contradiction
Possible tools

- Resource-bounded Kolmogorov complexity / expanders: build a sequence of high complexity to go beyond P/poly
- Algebraic tools (polynomials)
- Indirect diagonalisation along this way:

  if succinctPIT is 2EXP-complete:
  . succinct circuits can compute large products
  . they can be used as advice to decide efficiently PIT

  ⟹ a contradiction

- ...

Possible tools
DANKE!
THANK YOU!
MERCI!
GRAZIE!
GRACIAS!
DANK JE WEL!

Question time

(answers might be randomized)