Pseudo-deterministic Algorithms

Crypto-Reading Group

Tamalika Mukherjee

Purdue University
Recall

**BPP.** The class of decision problems solvable by a probabilistic Turing machine in polynomial time with an error probability bounded away from 1/2 for all instances.
Is $P = BPP$?
Major Ongoing Derandomization Efforts

• A one-way permutation can be used to construct a pseudorandom generator. [Yao 1982]
• A one-way function is sufficient to construct a pseudorandom generator. [Håstad et al. 1999]
• Nisan-Wigderson generator. [Nisan and Wigderson 1994]
• If $E$ contains a language of circuit complexity $2^{\Omega(n)}$ almost everywhere then $P = BPP$. [Impagliazzo and Wigderson 1997]
Major Ongoing Derandomization Efforts

- A one-way permutation can be used to construct a pseudorandom generator. [Yao 1982]
- A one-way function is sufficient to construct a pseudorandom generator. [Håstad et al. 1999]
- Nisan-Wigderson generator. [Nisan and Wigderson 1994]
- If $E$ contains a language of circuit complexity $2^{\Omega(n)}$ almost everywhere then $P = BPP$. [Impagliazzo and Wigderson 1997]

------------
Major Ongoing Derandomization Efforts

- A one-way permutation can be used to construct a pseudorandom generator. [Yao 1982]
- A one-way function is sufficient to construct a pseudorandom generator. [Håstad et al. 1999]
- Nisan-Wigderson generator. [Nisan and Wigderson 1994]
- If $E$ contains a language of circuit complexity $2^{\Omega(n)}$ almost everywhere then $P = \textit{BPP}$. [Impagliazzo and Wigderson 1997]

---------------------

Primes $\in P$ [Agrawal, Kayal, and Saxena 2004]
Major Ongoing Derandomization Efforts

- A one-way permutation can be used to construct a pseudorandom generator. [Yao 1982]
- A one-way function is sufficient to construct a pseudorandom generator. [Håstad et al. 1999]
- Nisan-Wigderson generator. [Nisan and Wigderson 1994]
- If $E$ contains a language of circuit complexity $2^\Omega(n)$ almost everywhere then $P = BPP$. [Impagliazzo and Wigderson 1997]

Primes $\in P$ [Agrawal, Kayal, and Saxena 2004]

Polynomial identity testing?
Search vs Decision Problems

Given a relation $R \subseteq \{0, 1\}^* \times \{0, 1\}^*$,

- $\text{Search}_R(x) = y$ such that $(x, y) \in R$, if $y$ exists, otherwise output $\perp$.
- $\text{Decision}_R(x) = 1$ if $x \in L_R$, otherwise output 0.
**Motivating Problem**

**Primes.** Given an integer, determine if it is prime or not.

\(^1\)Problem posed by Terence Tao in Polymath4 Project
Motivating Problem

Primes. Given an integer, determine if it is prime or not.

Primes $\in P$

---

1 Problem posed by Terence Tao in Polymath4 Project
Motivating Problem

**Primes.** Given an integer, determine if it is prime or not.

**Primes \( \in P \)**

**Primes (search-version).** Given an integer \( N \), find a prime in the range \([N, 2N]\). \(^1\)

- Easy *randomized* polynomial time algorithm.
- No known *deterministic* polynomial time algorithm.

\(^1\)Problem posed by Terence Tao in Polymath4 Project
Search problems that have an *elementary* probabilistic algorithms but *no elementary* deterministic algorithm:

- Find a prime $> N$.
- Find a generator for $\mathbb{Z}_p^*$.
- Find quadratic non-residue of $\mathbb{Z}_p^*$.
- Finding irreducible polynomials over $\mathbb{Z}_p$. 

........
Deterministic vs Probabilistic for Search Problems

Given an input $x$,

**Deterministic:**
- No randomness used
- Same/unique output

**Probabilistic:**
- Randomness used
- Different output over different randomness

\[2\text{Summarizing the words of Von Neumann: Where are the coins coming from?}\]
Given an input $x$, 

**Deterministic:**
- No randomness used
- Same/unique output

**Probabilistic:**
- Randomness used
- Different output over different randomness

**Traditional Goal:** *Derandomize* Probabilistic algorithms to obtain Deterministic algorithms. \(^2\)

---

\(^2\)Summarizing the words of Von Neumann: Where are the coins coming from?
Given an input $x$,

**Deterministic:**
- No randomness used
- Same/unique output

**Probabilistic:**
- Randomness used
- Different output over different randomness

Traditional Goal: *Derandomize* Probabilistic algorithms to obtain Deterministic algorithms. \(^2\)

Current Goal (for this talk): Assume we have randomness. *Partially derandomize* Probabilistic algorithms to achieve something close to deterministic. \(^3\)

---

\(^2\)Summarizing the words of Von Neumann: Where are the coins coming from?

\(^3\)improve probabilistic algorithms to give same guarantees as deterministic ones
Design randomized algorithms which output a unique solution with high probability.
Design randomized algorithms which output a unique solution with high probability.

Randomized algorithms that cannot be distinguished from deterministic algorithms except with negligible probability.
Let $R \subseteq \{0, 1\}^* \times \{0, 1\}^*$.

**Definition**

Probabilistic algorithm $A$ is *pseudo-deterministic* if, for every $x$:

- (correctness) $\Pr_r[(x, A(x, r)) \in R] > 2/3$.
- (uniqueness) there exists $y$ such that $\Pr_r[A(x, r) = y] > 2/3$. 
Usefulness

Pseudo-deterministic algorithms are useful in search problem contexts where an efficient randomized algorithm exists but no *elementary* deterministic solution exists.

- **Non-elementary solution**:
  - Non-efficient algorithm. Example: Primes.
  - Efficient algorithm may exist but complex solution. Example: $q$-ary linear codes achieving GV bound (where $q$ is prime power).
  - No explicit constructions exist. Example: binary linear codes achieving GV bound, general Ramsey graphs.
Usefulness

Pseudo-deterministic algorithms are useful in search problem contexts where an efficient randomized algorithm exists but no *elementary* deterministic solution exists.

Non-elementary solution:

- Non-efficient algorithm.
  Example: Primes.
Usefulness

Pseudo-deterministic algorithms are useful in search problem contexts where an efficient randomized algorithm exists but no elementary deterministic solution exists.

Non-elementary solution:

- Non-efficient algorithm.
  Example: Primes.
- Efficient algorithm may exist but complex solution.
  Example: $q$-ary linear codes achieving GV bound (where $q$ is prime power).
Pseudo-deterministic algorithms are useful in search problem contexts where an efficient randomized algorithm exists but no \textit{elementary} deterministic solution exists.

Non-elementary solution:

- Non-efficient algorithm. Example: Primes.
- Efficient algorithm may exist but complex solution. Example: $q$-ary linear codes achieving GV bound (where $q$ is prime power).
- No explicit constructions exist. Example: binary linear codes achieving GV bound, general Ramsey graphs.
Complexity Importance

P vs BPP

search-P vs search-BPP
Given an integer $N$, find a prime in the range $[N, 2N]$.

- Easy *randomized* polynomial time algorithm.
- No known *deterministic* polynomial time algorithm.

Even assuming $P = BPP$ there is still no known deterministic polynomial time algorithm.
Connecting Search and Decision problems

Theorem

If $P = BPP$ then every pseudo-deterministic polynomial time search algorithm can be converted into deterministic polynomial time search algorithm.

[Gat and Goldwasser 2011]
Pseudo-deterministic Algorithm for search – Primes?

- Randomized algorithm runs in polynomial time.
- Best known deterministic algorithm runs in $2^{n/2+o(n)}$.

\[^4\text{non-constructive: sequence of applications of hardness vs randomness paradigm}\]
• Randomized algorithm runs in polynomial time.
• Best known deterministic algorithm runs in $2^{n/2 + o(n)}$.

There is a **sub-exponential** time pseudo-deterministic algorithm for generating infinitely many primes.\(^4\) [Oliveira and Santhanam 2017]

\(^4\)non-constructive: sequence of applications of hardness vs randomness paradigm
Specifically look at case where \( p = kq + 1 \) where \( q \) is prime and \( k \) is of size \( \text{polylog}(p) \).\(^5\)

- Efficient probabilistic algorithm known to run in expected polynomial time in \( k \) and \( \log p \).
- No known efficient deterministic algorithm.

\(^5\)factorization of \( p - 1 \) is known.
Specifically look at case where $p = kq + 1$ where $q$ is prime and $k$ is of size $\text{polylog}(p)$.\(^5\)

- Efficient probabilistic algorithm known to run in expected polynomial time in $k$ and $\log p$.
- No known efficient deterministic algorithm.

There exists a pseudo-deterministic algorithm that runs in expected polynomial time. [Gat and Goldwasser 2011]

\(^5\)factorization of $p - 1$ is known.
Other Results, Part 1

Number Theoretic Problems.

• Pseudo-deterministic algorithm that finds $q$-th quadratic non-residue in $\mathbb{Z}_p^*$ that runs in expected polynomial time.$^6$ [Gat and Goldwasser 2011]

• Pseudo-deterministic algorithm that, given a prime $p$, finds a primitive root modulo $p$ in time $\exp(O(\sqrt{\log p \log \log p}))$. $^6$ [Grossman 2015]

$^6$Same time as best known existing randomized algorithms.
Other Results, Part 1

Number Theoretic Problems.

- Pseudo-deterministic algorithm that finds $q$-th quadratic non-residue in $\mathbb{Z}_p^*$ that runs in expected polynomial time.\(^6\) [Gat and Goldwasser 2011]

- Pseudo-deterministic algorithm that, given a prime $p$, finds a primitive root modulo $p$ in time $\exp(O(\sqrt{\log p \log \log p}))$.\(^6\) [Grossman 2015]

Learning and Approximation problems.

- Pseudo-derandomization techniques in this context. [Oliveira and Santhanam 2018]

\(^6\)Same time as best known existing randomized algorithms.
Graph problems.

- Pseudo-deterministic RNC algorithm for finding perfect matching in bipartite graphs which uses poly(n) processors, poly(log n) depth, poly(log n) random bits.\(^7\) [Goldwasser and Grossman 2015]

- Pseudo-deterministic NC algorithm for finding perfect matching in bipartite graphs which uses poly(n) processors, poly(log n) depth, poly(log n) random bits. [Goldwasser and Grossman 2017].

\(^7\)first algorithm to return unique perfect matchings with only polynomially many processors.
Strategies So Far

• **Canonization Strategy**: Use classical probabilistic search algorithm to find random solution then reduce to canonical solution.

• **Reduce to decision**: Reduce search to a decision problem solvable in probabilistic polynomial time.

Theorem: Any pseudo-deterministic polynomial time algorithm is just a deterministic polynomial time algorithm that makes oracle queries to some BPP-decision problem. [Gat and Goldwasser 2011]
Strategies So Far

- **Canonization Strategy**: Use classical probabilistic search algorithm to find random solution then *reduce* to canonical solution.
Strategies So Far

- **Canonization Strategy**: Use classical probabilistic search algorithm to find random solution then *reduce* to canonical solution.

- **Reduce to decision**: Reduce search to a decision problem solvable in probabilistic polynomial time.
Strategies So Far

- **Canonization Strategy**: Use classical probabilistic search algorithm to find random solution then *reduce* to canonical solution.

- **Reduce to decision**: Reduce search to a decision problem solvable in probabilistic polynomial time.

**Theorem**

*Any pseudo-deterministic polynomial time algorithm is just a deterministic polynomial time algorithm that makes oracle queries to some BPP-decision problem.* [Gat and Goldwasser 2011]
Representative Open Problems

Finding explicit pseudo-deterministic algorithm for

- Generating a prime. ¹
- Generating a Ramsey graph. ⁸
- Generating a linear code that achieves the GV bound.

¹Derandomization problem posed by Oded Goldreich
²Derandomization problem posed by Gil Kalai
Representative Open Problems

Finding explicit pseudo-deterministic algorithm for

- Generating a prime. ¹
- Generating a Ramsey graph. ⁸
- Generating a linear code that achieves the GV bound.

Any search problem that has easy randomized algorithms but non-elementary deterministic ones!

⁸Derandomization problem posed by Gil Kalai
So far ... 

*Find* a unique solution
Extension to Interactive Proofs

So far ... 

*Find* a unique solution

Now ... 

*Verify* that a solution is unique.
Example: Graph Isomorphism

Given graphs $G_1, G_2$ :

Verifier  
\[\text{Isomorphism } \Phi\]  
Prover

OR

Prover

21
Pseudo-deterministic Proofs

$R$ is a relation and $L_R : x \text{ s.t. } \exists y \text{ s.t. } R(x,y) = 1$

Input : $x$

Verifier $V$

Prover $P$

Output solution $y$ or rejects

- Completeness : $\forall x \in L_R \exists P \text{ s.t. } Pr_r[(P,V)(x,r) = y \text{ s.t. } (x,y) \in R] > 2/3$. 
Pseudo-deterministic Proofs

$R$ is a relation and $L_R : x \text{ s.t. } \exists y \text{ s.t. } R(x, y) = 1$

Input : $x$

Verifier $V$ \hspace{2cm} Prover $P$

Output solution $y$ or rejects

$\bullet$ Completeness : $\forall x \in L_R \exists P$ s.t. $\Pr_r[(P, V)(x, r) = y \text{ s.t. } (x, y) \in R] > 2/3$.

$\bullet$ Canonical : $\forall x \in L_R \exists y$ s.t. $\forall P' : \Pr_r[(P', V)(x, r) = y' \text{ s.t. } y' \neq y] < 1/3$. 
Pseudo-deterministic Proofs

$R$ is a relation and $L_R : x \text{ s.t. } \exists y \text{ s.t. } R(x, y) = 1$

Input : $x$

Verifier V

Prover P

OR

Prover P

Output solution $y$ or rejects

• Completeness : $\forall x \in L_R \exists P \text{ s.t. } \Pr_r[(P, V)(x, r) = y \text{ s.t. } (x, y) \in R] > 2/3$.

• Canonical : $\forall x \in L_R \exists y \text{ s.t. } \forall P' : \Pr_r[(P', V)(x, r) = y' \text{ s.t. } y' \neq y] < 1/3$.

• Soundness : $\forall x \notin L_R \forall P' \Pr_r[(P', V)(x, r) \neq \bot] < 1/3$
Theorem

There is an interactive protocol that outputs the lexicographically smallest graph isomorphism in constant rounds.\(^9\) [Goldwasser, Grossman, and Holden 2017]

\(^9\)More formally, $\text{GRAPH-ISOMORPHISM} \in \text{psdAM}$
• **Number Theory:** So far don’t have pseudo-deterministic algorithm for generating a canonical prime. Are there psdAM proofs to verify that given prime is canonical?
• **Number Theory:** So far don’t have pseudo-deterministic algorithm for generating a canonical prime. Are there $psdAM$ proofs to verify that given prime is canonical?

• **Pseudo-determinism and TFNP:** So far $psdNP$ as been defined and studied. Interesting question: $psdNP = TFNP$?
Open Problems

• **Number Theory:** So far don’t have pseudo-deterministic algorithm for generating a canonical prime. Are there $psdAM$ proofs to verify that given prime is canonical?

• **Pseudo-determinism and TFNP:** So far $psdNP$ as been defined and studied. Interesting question: $psdNP = TFNP$?

• **Pseudo-determinism and Quantum:** Quantum equivalent question of $P = BPP : psdBQP = BQP$?
References


Shafi Goldwasser and Ofer Grossman. “Perfect Bipartite Matching in Pseudo-Deterministic RNC.” In: *Electronic*


