Proving Lower Bounds via Pseudo-Random Generators

Manindra Agrawal

IIT Kanpur

FSTTCS 2005

Overview

1. Lower Bounds History
2. Pseudo-Random Generators
3. Applications of Time-Bounded Pseudo-Random Generators
   - Derandomizing Randomized Algorithms
   - Formalizing Cryptographic Security
   - Lower Bounds
4. Lower Bounds on Boolean Circuits
5. Lower Bounds on Arithmetic Circuits

Outline

1. Lower Bounds History

2. Pseudo-Random Generators

3. Applications of Time-Bounded Pseudo-Random Generators
   - Derandomizing Randomized Algorithms
   - Formalizing Cryptographic Security
   - Lower Bounds

4. Lower Bounds on Boolean Circuits

5. Lower Bounds on Arithmetic Circuits

Approaches to Lower Bounds

- Proving lower bounds on the complexity of problems is the central aim of complexity theory.
- Most important amongst these is to prove $P \neq NP$.
- So far, we have not been very successful.
- Two approaches have been used over last thirty years but both have hit roadblocks.

Approaches to Lower Bounds

- Proving lower bounds on the complexity of problems is the central aim of complexity theory.
- Most important amongst these is to prove P $\neq$ NP.
- So far, we have not been very successful.
- Two approaches have been used over last thirty years but both have hit roadblocks.

Approaches to Lower Bounds

- Proving lower bounds on the complexity of problems is the central aim of complexity theory.
- Most important amongst these is to prove P $\neq$ NP.
- So far, we have not been very successful.
- Two approaches have been used over last thirty years but both have hit roadblocks.

First Approach: Diagonalization

Basic Idea

To prove that the set A does not belong to complexity class $\mathcal{C}$.

- Consider the (infinite) sequence of Turing machines accepting precisely the class of sets in $\mathcal{C}$.
- Let this sequence be $M_1, M_2, \ldots$.
- Show that for every i, there is a string x_i that belongs to set A iff M_i rejects x_i.

First Approach: Diagonalization

Basic Idea

To prove that the set A does not belong to complexity class $\mathcal{C}$.

- Consider the (infinite) sequence of Turing machines accepting precisely the class of sets in $\mathcal{C}$.
- Let this sequence be $M_1, M_2, \ldots$.
- Show that for every i, there is a string x_i that belongs to set A iff M_i rejects x_i.

First Approach: Diagonalization

Basic Idea

To prove that the set A does not belong to complexity class $\mathcal{C}$.

- Consider the (infinite) sequence of Turing machines accepting precisely the class of sets in $\mathcal{C}$.
- Let this sequence be $M_1, M_2, \ldots$.
- Show that for every i, there is a string x_i that belongs to set A iff M_i rejects x_i.

First Approach: Diagonalization

- Earliest approach, popular in 1970s.
- Useful for seperating complexity classes that are very "far apart," e.g., P and EXP.
- Did not work for closer classes, e.g., P and NP.
- Baker-Gill-Solovay (1975) showed that standard approaches to diagonalization cannot seperate P and NP.
- They proved that standard techniques diagonalize and no diagonalizable technique can prove $P \neq NP$ or $P = NP$.

First Approach: Diagonalization

- Earliest approach, popular in 1970s.
- Useful for seperating complexity classes that are very "far apart," e.g., P and EXP.
- Did not work for closer classes, e.g., P and NP.
- Baker-Gill-Solovay (1975) showed that standard approaches to diagonalization cannot seperate P and NP.
- They proved that standard techniques diagonalize and no diagonalizable technique can prove $\mathsf{P} \neq \mathsf{NP}$ or $\mathsf{P} = \mathsf{NP}$.

First Approach: Diagonalization

- Earliest approach, popular in 1970s.
- Useful for seperating complexity classes that are very "far apart," e.g., P and EXP.
- Did not work for closer classes, e.g., P and NP.
- Baker-Gill-Solovay (1975) showed that standard approaches to diagonalization cannot seperate P and NP.
- They proved that standard techniques diagonalize and no diagonalizable technique can prove $\mathsf{P} \neq \mathsf{NP}$ or $\mathsf{P} = \mathsf{NP}$.

First Approach: Diagonalization

- Earliest approach, popular in 1970s.
- Useful for seperating complexity classes that are very "far apart," e.g., P and EXP.
- Did not work for closer classes, e.g., P and NP.
- Baker-Gill-Solovay (1975) showed that standard approaches to diagonalization cannot seperate P and NP.
- They proved that standard techniques diagonalize and no diagonalizable technique can prove $P \neq NP$ or $P = NP$.

Example: Seperating P From EXP

- Let M_1, M_2, $\ldots$ be an enumeration of deterministic TMs with M_i running for at most $n^{|i|}$ steps on an input of size n.
- Define a set A as:

$$A = \{i \mid M_i \text{ rejects } i\}.$$

- Set A is in EXP.
- If TM M_j from the above sequence accepts A then M_j accepts j iff M_j rejects j.

Example: Seperating P From EXP

- Let M_1, M_2, $\ldots$ be an enumeration of deterministic TMs with M_i running for at most $n^{|i|}$ steps on an input of size n.
- Define a set A as:

$$A = \{i \mid M_i \text{ rejects } i\}.$$

- Set A is in EXP.
- If TM M_j from the above sequence accepts A then M_j accepts j iff M_j rejects j.

Example: Seperating P From EXP

- Let M_1, M_2, $\ldots$ be an enumeration of deterministic TMs with M_i running for at most $n^{|i|}$ steps on an input of size n.
- Define a set A as:

$$A = \{i \mid M_i \text{ rejects } i\}.$$

- Set A is in EXP.
- If TM M_j from the above sequence accepts A then M_j accepts j iff M_j rejects j.

Second Approach: Combinatorial Arguments on Circuits

- Most of the complexity classes have a circuit characterization.
- A family of circuits, one for each input length, corresponds to a set in the class.
- We consider circuits that are layered and have unbounded fanin gates.

Second Approach: Combinatorial Arguments on Circuits

- Most of the complexity classes have a circuit characterization.
- A family of circuits, one for each input length, corresponds to a set in the class.
- We consider circuits that are layered and have unbounded fanin gates.

Second Approach: Combinatorial Arguments on Circuits

- Most of the complexity classes have a circuit characterization.
- A family of circuits, one for each input length, corresponds to a set in the class.
- We consider circuits that are layered and have unbounded fanin gates.

Second Approach: Combinatorial Arguments on Circuits

Basic Idea

To prove that the set A does not belong to complexity class $\mathcal{C}$.

- Consider the circuit characterization of $\mathcal{C}$.
- This is given by a family of circuits, one circuit for every input length, for each set in $\mathcal{C}$.
- Prove that any circuit on input length n from the families can be transformed to a "simple" circuit that "approximates" the original circuit well.
- Prove that no "simple" circuit can approximate the set A well.

Second Approach: Combinatorial Arguments on Circuits

Basic Idea

To prove that the set A does not belong to complexity class $\mathcal{C}$.

- Consider the circuit characterization of $\mathcal{C}$.
- This is given by a family of circuits, one circuit for every input length, for each set in $\mathcal{C}$.
- Prove that any circuit on input length n from the families can be transformed to a "simple" circuit that "approximates" the original circuit well.
- Prove that no "simple" circuit can approximate the set A well.

Second Approach: Combinatorial Arguments on Circuits

Basic Idea

To prove that the set A does not belong to complexity class $\mathcal{C}$.

- Consider the circuit characterization of $\mathcal{C}$.
- This is given by a family of circuits, one circuit for every input length, for each set in $\mathcal{C}$.
- Prove that any circuit on input length n from the families can be transformed to a "simple" circuit that "approximates" the original circuit well.
- Prove that no "simple" circuit can approximate the set A well.

Second Approach: Combinatorial Arguments on Circuits

Basic Idea

To prove that the set A does not belong to complexity class $\mathcal{C}$.

- Consider the circuit characterization of $\mathcal{C}$.
- This is given by a family of circuits, one circuit for every input length, for each set in $\mathcal{C}$.
- Prove that any circuit on input length n from the families can be transformed to a "simple" circuit that "approximates" the original circuit well.
- Prove that no "simple" circuit can approximate the set A well.

Second Approach: Combinatorial Arguments on Circuits

- Proposed in 1980s.
- Biggest successes were lower bounds on monotone and constant depth circuit classes.
- Razborov (1985) seperated the class of sets characterized by polynomial sized monotone circuits from the class of sets in NP accepted by monotone circuits.
- Furst-Saxe-Sipser (1984), Håstad (1986) showed that the set PARITY does not belong to the class of sets characterized by constant depth, polynomial sized circuits.

PARITY is the set of all strings that have an odd number of 1's.

Second Approach: Combinatorial Arguments on Circuits

- Proposed in 1980s.
- Biggest successes were lower bounds on monotone and constant depth circuit classes.
- Razborov (1985) seperated the class of sets characterized by polynomial sized monotone circuits from the class of sets in NP accepted by monotone circuits.
- Furst-Saxe-Sipser (1984), Håstad (1986) showed that the set PARITY does not belong to the class of sets characterized by constant depth, polynomial sized circuits.

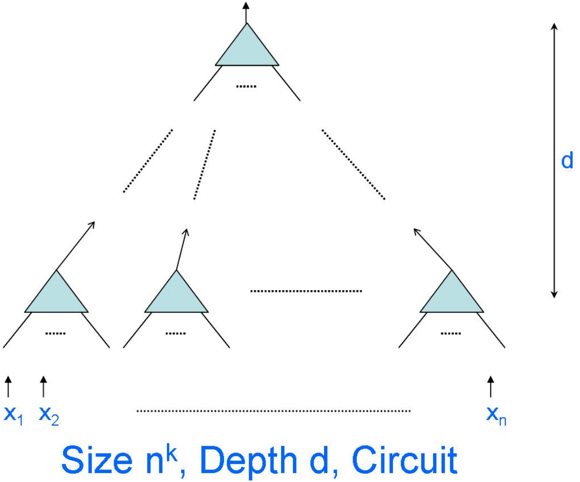
PARITY is the set of all strings that have an odd number of 1's.

Second Approach: Combinatorial Arguments on Circuits

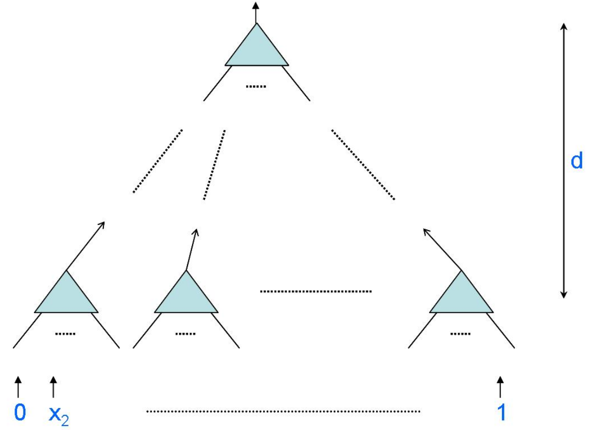
- Proposed in 1980s.
- Biggest successes were lower bounds on monotone and constant depth circuit classes.
- Razborov (1985) seperated the class of sets characterized by polynomial sized monotone circuits from the class of sets in NP accepted by monotone circuits.
- Furst-Saxe-Sipser (1984), Håstad (1986) showed that the set PARITY does not belong to the class of sets characterized by constant depth, polynomial sized circuits.

PARITY is the set of all strings that have an odd number of 1's.

Example: Lower Bounds on PARITY

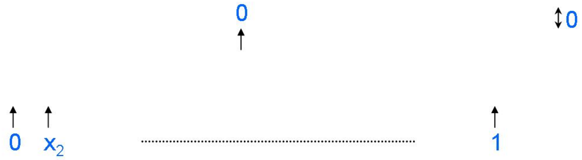


Example: Lower Bounds on PARITY



Random Assignment to n-n$^{\delta}$ Input Bits

Example: Lower Bounds on PARITY



Reduces to Fixed Circuit with Prob > 0

Second Approach: Combinatorial Arguments on Circuits

- Appeared very promising in the beginning.
- However, Razborov-Rudich (1994) proved otherwise.
- They classified the combinatorial arguments used as natural proofs.
- And showed, under very reasonable assumptions, that no natural proof can prove lower bounds on circuit classes significantly larger than constant depth, polynomial sized.

Second Approach: Combinatorial Arguments on Circuits

- Appeared very promising in the beginning.
- However, Razborov-Rudich (1994) proved otherwise.
- They classified the combinatorial arguments used as natural proofs.
- And showed, under very reasonable assumptions, that no natural proof can prove lower bounds on circuit classes significantly larger than constant depth, polynomial sized.

Second Approach: Combinatorial Arguments on Circuits

- Appeared very promising in the beginning.
- However, Razborov-Rudich (1994) proved otherwise.
- They classified the combinatorial arguments used as natural proofs.
- And showed, under very reasonable assumptions, that no natural proof can prove lower bounds on circuit classes significantly larger than constant depth, polynomial sized.

A New Approach: Pseudo-Random Generators

- Pseudo-random generators were defined in 1980s for two reasons:
  - To formalize the notion of cryptographic security.
  - To derandomize probabilistic algorithms.
- In 1990s, they were shown to be equivalent to certain types of lower bounds.
- Recently, there are indications that they might be useful in proving lower bounds.

A New Approach: Pseudo-Random Generators

- Pseudo-random generators were defined in 1980s for two reasons:
  - To formalize the notion of cryptographic security.
  - To derandomize probabilistic algorithms.
- In 1990s, they were shown to be equivalent to certain types of lower bounds.
- Recently, there are indications that they might be useful in proving lower bounds.

Outline

1. Lower Bounds History
2. Pseudo-Random Generators
3. Applications of Time-Bounded Pseudo-Random Generators
   - Derandomizing Randomized Algorithms
   - Formalizing Cryptographic Security
   - Lower Bounds
4. Lower Bounds on Boolean Circuits
5. Lower Bounds on Arithmetic Circuits

Definition

Let $\mathcal{C}(n, d)$ be the class of depth d, size n boolean circuits on n inputs.

Let $f : \{0,1\}^* \mapsto \{0,1\}^*$ be a function such that $|f(y)| = n$ for all strings y of length $\ell(n) < n$.

Definition

Function f is a $(\ell(n), n)$-pseudo-random generator against $\mathcal{C}(n, d)$ if for every circuit $C \in \mathcal{C}(n, d)$,

$$\frac{1}{2^n} \mid \{x \mid C(x) = 1\} \mid - \frac{1}{2^{\ell(n)}} \mid \{y \mid C(f(y)) = 1\} \mid \leq \frac{1}{n}.$$

String y is called the seed, and the difference $n - \ell(n)$ is called the stretch of the generator.

Definition

Function f is a $(\ell(n), n)$-pseudo-random generator against $\mathcal{C}(n, d)$ if for every circuit $C \in \mathcal{C}(n, d)$,

$$\frac{1}{2^n} \mid \{x \mid C(x) = 1\} \mid - \frac{1}{2^{\ell(n)}} \mid \{y \mid C(f(y)) = 1\} \mid \leq \frac{1}{n}.$$

String y is called the seed, and the difference $n - \ell(n)$ is called the stretch of the generator.

Existance of Pseudo-Random Generators

Let C be any circuit in $\mathcal{C}(n, n)$. Define F as: On input y, $|y| = 5 \log n$, output a random string of length n.

- For any y, define random variable Z_y as: $Z_y = C(f(y))$.
- Then,

$$\sum_y Z_y = \mid \{y \mid C(f(y)) = 1\} \mid .$$

- And,

$$\Pr[Z_y = 1] = \frac{1}{2^n} \mid \{x \mid C(x) = 1\} \mid = \mu_C \text{ (say)}.$$

Existance of Pseudo-Random Generators

Let C be any circuit in $\mathcal{C}(n,n)$. Define F as: On input y, $|y| = 5\log n$, output a random string of length n.

- For any y, define random variable Z_y as: $Z_y = C(f(y))$.
- Then,

$$\sum_y Z_y = \mid \{y \mid C(f(y)) = 1\} \mid .$$

- And,

$$\Pr[Z_y = 1] = \frac{1}{2^n} \mid \{x \mid C(x) = 1\} \mid = \mu_C \text{ (say)}.$$

Existance of Pseudo-Random Generators

Let C be any circuit in $\mathcal{C}(n, n)$. Define F as: On input y, $|y| = 5 \log n$, output a random string of length n.

- For any y, define random variable Z_y as: $Z_y = C(f(y))$.
- Then,
$$\sum_y Z_y = \mid \{y \mid C(f(y)) = 1\} \mid .$$
- And,
$$\Pr[Z_y = 1] = \frac{1}{2^n} \mid \{x \mid C(x) = 1\} \mid = \mu_C \text{ (say)}.$$

Existance of Pseudo-Random Generators

Let C be any circuit in $\mathcal{C}(n, n)$. Define F as: On input y, $|y| = 5\log n$, output a random string of length n.

- For any y, define random variable Z_y as: $Z_y = C(f(y))$.
- Then,
$$\sum_y Z_y = | \{y \mid C(f(y)) = 1\} | .$$
- And,
$$\Pr[Z_y = 1] = \frac{1}{2^n} | \{x \mid C(x) = 1\} | = \mu_C \text{ (say)}.$$

Existance of Pseudo-Random Generators

- By Chernoff's bound:

$$\Pr[|\frac{1}{n^5}\sum_y Z_y - \mu_C| > \delta\mu_C] < e^{-n^5\mu_C\delta^2/4} < e^{-n^5\delta^2/4}.$$

- Choosing $\delta = \frac{1}{n}$, we get:

$$\Pr[|\frac{1}{n^5}\sum_y Z_y - \mu| > \frac{1}{n}] < e^{-n^3/4}.$$

- Since there are less than 2^{n^2} circuits in $\mathcal{C}(n,n)$, probability that F fails to approximate μ_C for some $C \in \mathcal{C}(n,n)$ is at most $\frac{1}{2^{n/4}}$.
- Hence, most of the functions from $\{0,1\}^{5\log n}$ to $\{0,1\}^n$ are pseudo-random against $\mathcal{C}(n,n)$.

Existance of Pseudo-Random Generators

- By Chernoff's bound:

$$\Pr[\mid \frac{1}{n^5}\sum_y Z_y - \mu_C \mid > \delta\mu_C] < e^{-n^5\mu_C\delta^2/4} < e^{-n^5\delta^2/4}.$$

- Choosing $\delta = \frac{1}{n}$, we get:

$$\Pr[\mid \frac{1}{n^5}\sum_y Z_y - \mu \mid > \frac{1}{n}] < e^{-n^3/4}.$$

- Since there are less than 2^{n^2} circuits in $\mathcal{C}(n,n)$, probability that F fails to approximate μ_C for some $C \in \mathcal{C}(n,n)$ is at most $\frac{1}{2^{n/4}}$.
- Hence, most of the functions from $\{0,1\}^{5\log n}$ to $\{0,1\}^n$ are pseudo-random against $\mathcal{C}(n,n)$.

Existance of Pseudo-Random Generators

- By Chernoff's bound:

$$\Pr[| \frac{1}{n^5} \sum_y Z_y - \mu_C | > \delta\mu_C] < e^{-n^5 \mu_C \delta^2/4} < e^{-n^5\delta^2/4}.$$

- Choosing $\delta = \frac{1}{n}$, we get:

$$\Pr[| \frac{1}{n^5} \sum_y Z_y - \mu | > \frac{1}{n}] < e^{-n^3/4}.$$

- Since there are less than 2^{n^2} circuits in $\mathcal{C}(n,n)$, probability that F fails to approximate μ_C for some $C \in \mathcal{C}(n,n)$ is at most $\frac{1}{2^{n/4}}$.
- Hence, most of the functions from $\{0,1\}^{5\log n}$ to $\{0,1\}^n$ are pseudo-random against $\mathcal{C}(n,n)$.

Existance of Pseudo-Random Generators

- By Chernoff's bound:

$$\Pr[\mid \frac{1}{n^5}\sum_y Z_y - \mu_C \mid > \delta\mu_C] < e^{-n^5\mu_C\delta^2/4} < e^{-n^5\delta^2/4}.$$

- Choosing $\delta = \frac{1}{n}$, we get:

$$\Pr[\mid \frac{1}{n^5}\sum_y Z_y - \mu \mid > \frac{1}{n}] < e^{-n^3/4}.$$

- Since there are less than 2^{n^2} circuits in $\mathcal{C}(n,n)$, probability that F fails to approximate μ_C for some $C \in \mathcal{C}(n,n)$ is at most $\frac{1}{2^{n/4}}$.
- Hence, most of the functions from $\{0,1\}^{5\log n}$ to $\{0,1\}^n$ are pseudo-random against $\mathcal{C}(n,n)$.

Optimal Pseudo-Random Generators

Function f is an optimal pseudo-random generator against $\mathcal{C}(n, d)$ if it is a $(O(\log n), n)$-pseudo-random generator against $\mathcal{C}(n, d)$.

A simple argument shows that most of the functions are optimal pseudo-random generators against $\mathcal{C}(n, n)$.

Optimal Pseudo-Random Generators

Function f is an optimal pseudo-random generator against $\mathcal{C}(n, d)$ if it is a $(O(\log n), n)$-pseudo-random generator against $\mathcal{C}(n, d)$.

A simple argument shows that most of the functions are optimal pseudo-random generators against $\mathcal{C}(n, n)$.

Time-Bounded Pseudo-Random Generators

An $(\ell(n), n)$-pseudo-random generator f is $t(m)$-computable if there is a $t(m)$-time bounded DTM that, on input (y, j), $|y| = m = \ell(n)$ and $1 \leq j \leq n$, outputs the jth bit of $f(y)$.

Time-bounded pseudo-random generators are very interesting!

Time-Bounded Pseudo-Random Generators

An $(\ell(n), n)$-pseudo-random generator f is $t(m)$-computable if there is a $t(m)$-time bounded DTM that, on input (y, j), $|y| = m = \ell(n)$ and $1 \leq j \leq n$, outputs the jth bit of $f(y)$.

Time-bounded pseudo-random generators are very interesting!

Outline

Outline

1. Lower Bounds History
2. Pseudo-Random Generators
3. Applications of Time-Bounded Pseudo-Random Generators
   - Derandomizing Randomized Algorithms
   - Formalizing Cryptographic Security
   - Lower Bounds
4. Lower Bounds on Boolean Circuits
5. Lower Bounds on Arithmetic Circuits

Derandomizing BPP

Suppose there exists a $2^{O(m)}$-computable optimal pseudo-random generator f against $\mathcal{C}(n, n)$.

- Let $\mathcal{B}$ be a randomized polynomial-time algorithm accepting a set B in BPP.
- View $\mathcal{B}$ as taking two inputs x and r, with x being the "real" input and r being a sequence of random bits.
- Assume that $|r|$ equals the square of time taken by $\mathcal{B}$ on input x.

Derandomizing BPP

Suppose there exists a $2^{O(m)}$-computable optimal pseudo-random generator f against $\mathcal{C}(n, n)$.

- Let $\mathcal{B}$ be a randomized polynomial-time algorithm accepting a set B in BPP.
- View $\mathcal{B}$ as taking two inputs x and r, with x being the "real" input and r being a sequence of random bits.
- Assume that $|r|$ equals the square of time taken by $\mathcal{B}$ on input x.

Derandomizing BPP

Suppose there exists a $2^{O(m)}$-computable optimal pseudo-random generator f against $\mathcal{C}(n, n)$.

- Let $\mathcal{B}$ be a randomized polynomial-time algorithm accepting a set B in BPP.
- View $\mathcal{B}$ as taking two inputs x and r, with x being the "real" input and r being a sequence of random bits.
- Assume that $|r|$ equals the square of time taken by $\mathcal{B}$ on input x.

Derandomizing BPP

- Fix any x. Then $\mathcal{B}(x, r)$ can be thought of as a circuit C of size $n = |r|$ operating on input r.
- Circuit C outputs a 1 on either at least $\frac{2}{3}$-fraction or at most $\frac{1}{3}$-fraction of these inputs depending on whether x is in the set B or not.
- Therefore, C will output a 1 on either at least $(\frac{2}{3} - \frac{1}{n})$-fraction or at most $(\frac{1}{3} + \frac{1}{n})$-fraction of inputs of the form $f(y)$.

Derandomizing BPP

- Fix any x. Then $\mathcal{B}(x, r)$ can be thought of as a circuit C of size $n = |r|$ operating on input r.
- Circuit C outputs a 1 on either at least $\frac{2}{3}$-fraction or at most $\frac{1}{3}$-fraction of these inputs depending on whether x is in the set B or not.
- Therefore, C will output a 1 on either at least $(\frac{2}{3} - \frac{1}{n})$-fraction or at most $(\frac{1}{3} + \frac{1}{n})$-fraction of inputs of the form $f(y)$.

Derandomizing BPP

- Fix any x. Then $\mathcal{B}(x, r)$ can be thought of as a circuit C of size $n = |r|$ operating on input r.
- Circuit C outputs a 1 on either at least $\frac{2}{3}$-fraction or at most $\frac{1}{3}$-fraction of these inputs depending on whether x is in the set B or not.
- Therefore, C will output a 1 on either at least $(\frac{2}{3} - \frac{1}{n})$-fraction or at most $(\frac{1}{3} + \frac{1}{n})$-fraction of inputs of the form $f(y)$.

Derandomizing BPP

- Since f is optimal, $|y| = O(\log n)$.
- Since f is $2^{O(m)}$-computable and $m = |y| = O(\log n)$, $f(y)$ can be computed in time $n^{O(1)}$.
- Therefore, in time polynomial in n, one can deterministically decide if x is in the set B or not.
- Since $n = |r|$, n is a polynomial in $|x|$.
- This shows that $B \in \mathrm{P}$.
- Thus, $\mathrm{BPP} = \mathrm{P}$.

Derandomizing BPP

- Since f is optimal, $|y| = O(\log n)$.
- Since f is $2^{O(m)}$-computable and $m = |y| = O(\log n)$, $f(y)$ can be computed in time $n^{O(1)}$.
- Therefore, in time polynomial in n, one can deterministically decide if x is in the set B or not.
- Since $n = |r|$, n is a polynomial in $|x|$.
- This shows that $B \in \mathrm{P}$.
- Thus, $\mathrm{BPP} = \mathrm{P}$.

Derandomizing BPP

- Since f is optimal, $|y| = O(\log n)$.
- Since f is $2^{O(m)}$-computable and $m = |y| = O(\log n)$, $f(y)$ can be computed in time $n^{O(1)}$.
- Therefore, in time polynomial in n, one can deterministically decide if x is in the set B or not.
- Since $n = |r|$, n is a polynomial in $|x|$.
- This shows that $B \in \mathrm{P}$.
- Thus, $\mathrm{BPP} = \mathrm{P}$.

Derandomizing BPP

- Since f is optimal, $|y| = O(\log n)$.
- Since f is $2^{O(m)}$-computable and $m = |y| = O(\log n)$, $f(y)$ can be computed in time $n^{O(1)}$.
- Therefore, in time polynomial in n, one can deterministically decide if x is in the set B or not.
- Since $n = |r|$, n is a polynomial in $|x|$.
- This shows that $B \in \mathrm{P}$.
- Thus, $\mathrm{BPP} = \mathrm{P}$.

Derandomizing BPP

- Since f is optimal, $|y| = O(\log n)$.
- Since f is $2^{O(m)}$-computable and $m = |y| = O(\log n)$, $f(y)$ can be computed in time $n^{O(1)}$.
- Therefore, in time polynomial in n, one can deterministically decide if x is in the set B or not.
- Since $n = |r|$, n is a polynomial in $|x|$.
- This shows that $B \in \mathrm{P}$.
- Thus, $\mathrm{BPP} = \mathrm{P}$.

Derandomizing BPP

- Since f is optimal, $|y| = O(\log n)$.
- Since f is $2^{O(m)}$-computable and $m = |y| = O(\log n)$, $f(y)$ can be computed in time $n^{O(1)}$.
- Therefore, in time polynomial in n, one can deterministically decide if x is in the set B or not.
- Since $n = |r|$, n is a polynomial in $|x|$.
- This shows that $B \in \mathrm{P}$.
- Thus, $\mathrm{BPP} = \mathrm{P}$.

Outline

1. Lower Bounds History
2. Pseudo-Random Generators
3. Applications of Time-Bounded Pseudo-Random Generators
   - Derandomizing Randomized Algorithms
   - Formalizing Cryptographic Security
   - Lower Bounds
4. Lower Bounds on Boolean Circuits
5. Lower Bounds on Arithmetic Circuits

Formalizing Cryptographic Security

Suppose there exists a $m^{O(1)}$-computable $(n^{o(1)}, n)$-pseudo-random generator f against $\mathcal{C}(n, n)$.

- Define function g as: on input y, $|y| = m$, output the first m^4 bits of $f(y)$.
- Function g is efficiently computed since first m^4 bits of f can be computed in time $m^{O(1)}$.

Formalizing Cryptographic Security

Suppose there exists a $m^{O(1)}$-computable $(n^{o(1)}, n)$-pseudo-random generator f against $\mathcal{C}(n, n)$.

- Define function g as: on input y, $|y| = m$, output the first m^4 bits of $f(y)$.
- Function g is efficiently computed since first m^4 bits of f can be computed in time $m^{O(1)}$.

Formalizing Cryptographic Security

Suppose there exists a $m^{O(1)}$-computable $(n^{o(1)}, n)$-pseudo-random generator f against $\mathcal{C}(n, n)$.

- Define function g as: on input y, $|y| = m$, output the first m^4 bits of $f(y)$.
- Function g is efficiently computed since first m^4 bits of f can be computed in time $m^{O(1)}$.

Formalizing Cryptographic Security

No randomized polynomial-time bounded adversary can distinguish the output of function g from a random sequence.

- Let $\mathcal{A}$ be a randomized polynomial-time algorithm.
- Suppose that $\mathcal{A}$ can distinguish the output of g from a random sequence.
- View $\mathcal{A}$ on input of size m^4 as a size $n = m^{O(1)}$ circuit C.
- Modify function g to $\hat{g}$ which outputs first n bits of f instead of first m^4.
- $\mathcal{A}$ can distinguish the output of $\hat{g}$ from a random sequence by simply ignoring all except first m^4 input bits.
- This, however, is not possible since f is pseudo-random against $\mathcal{C}(n, n)$.

Formalizing Cryptographic Security

No randomized polynomial-time bounded adversary can distinguish the output of function g from a random sequence.

- Let $\mathcal{A}$ be a randomized polynomial-time algorithm.
- Suppose that $\mathcal{A}$ can distinguish the output of g from a random sequence.
- View $\mathcal{A}$ on input of size m^4 as a size $n = m^{O(1)}$ circuit C.
- Modify function g to $\hat{g}$ which outputs first n bits of f instead of first m^4.
- $\mathcal{A}$ can distinguish the output of $\hat{g}$ from a random sequence by simply ignoring all except first m^4 input bits.
- This, however, is not possible since f is pseudo-random against $\mathcal{C}(n, n)$.

Formalizing Cryptographic Security

No randomized polynomial-time bounded adversary can distinguish the output of function g from a random sequence.

- Let $\mathcal{A}$ be a randomized polynomial-time algorithm.
- Suppose that $\mathcal{A}$ can distinguish the output of g from a random sequence.
- View $\mathcal{A}$ on input of size m^4 as a size $n = m^{O(1)}$ circuit C.
- Modify function g to $\hat{g}$ which outputs first n bits of f instead of first m^4.
- $\mathcal{A}$ can distinguish the output of $\hat{g}$ from a random sequence by simply ignoring all except first m^4 input bits.
- This, however, is not possible since f is pseudo-random against $\mathcal{C}(n, n)$.

Formalizing Cryptographic Security

No randomized polynomial-time bounded adversary can distinguish the output of function g from a random sequence.

- Let $\mathcal{A}$ be a randomized polynomial-time algorithm.
- Suppose that $\mathcal{A}$ can distinguish the output of g from a random sequence.
- View $\mathcal{A}$ on input of size m^4 as a size $n = m^{O(1)}$ circuit C.
- Modify function g to $\hat{g}$ which outputs first n bits of f instead of first m^4.
- $\mathcal{A}$ can distinguish the output of $\hat{g}$ from a random sequence by simply ignoring all except first m^4 input bits.
- This, however, is not possible since f is pseudo-random against $\mathcal{C}(n, n)$.

Formalizing Cryptographic Security

No randomized polynomial-time bounded adversary can distinguish the output of function g from a random sequence.

- Let $\mathcal{A}$ be a randomized polynomial-time algorithm.
- Suppose that $\mathcal{A}$ can distinguish the output of g from a random sequence.
- View $\mathcal{A}$ on input of size m^4 as a size $n = m^{O(1)}$ circuit C.
- Modify function g to $\hat{g}$ which outputs first n bits of f instead of first m^4.
- $\mathcal{A}$ can distinguish the output of $\hat{g}$ from a random sequence by simply ignoring all except first m^4 input bits.
- This, however, is not possible since f is pseudo-random against $\mathcal{C}(n, n)$.

Formalizing Cryptographic Security

Function g is a provably secure stream cipher.

- View input y to g as key.
- View $g(y)$ as pseudo-random stream.
- For example, for key size 128 bits, g provides 256 Mbits of random stream.

This notion can be used to formalize block ciphers and public-key encryption algorithms too.

Formalizing Cryptographic Security

Function g is a provably secure stream cipher.

- View input y to g as key.
- View $g(y)$ as pseudo-random stream.
- For example, for key size 128 bits, g provides 256 Mbits of random stream.

This notion can be used to formalize block ciphers and public-key encryption algorithms too.

Formalizing Cryptographic Security

Function g is a provably secure stream cipher.

- View input y to g as key.
- View $g(y)$ as pseudo-random stream.
- For example, for key size 128 bits, g provides 256 Mbits of random stream.

This notion can be used to formalize block ciphers and public-key encryption algorithms too.

Formalizing Cryptographic Security

Function g is a provably secure stream cipher.

- View input y to g as key.
- View $g(y)$ as pseudo-random stream.
- For example, for key size 128 bits, g provides 256 Mbits of random stream.

This notion can be used to formalize block ciphers and public-key encryption algorithms too.

Formalizing Cryptographic Security

Function g is a provably secure stream cipher.

- View input y to g as key.
- View $g(y)$ as pseudo-random stream.
- For example, for key size 128 bits, g provides 256 Mbits of random stream.

This notion can be used to formalize block ciphers and public-key encryption algorithms too.

Outline

Lower Bounds via Pseudo-Random Generators

Suppose there exists a $2^{O(m)}$-computable optimal pseudo-random generator f against $\mathcal{C}(n, n)$.

- Define a set B as: on input z, $|z| = 2m$, accept if there exists a y, $|y| = m$, such that z is a prefix of $f(y)$.
- Set B is in E.

Lower Bounds via Pseudo-Random Generators

Suppose there exists a $2^{O(m)}$-computable optimal pseudo-random generator f against $\mathcal{C}(n, n)$.

- Define a set B as: on input z, $|z| = 2m$, accept if there exists a y, $|y| = m$, such that z is a prefix of $f(y)$.
- Set B is in E.

Lower Bounds via Pseudo-Random Generators

Suppose there exists a $2^{O(m)}$-computable optimal pseudo-random generator f against $\mathcal{C}(n, n)$.

- Define a set B as: on input z, $|z| = 2m$, accept if there exists a y, $|y| = m$, such that z is a prefix of $f(y)$.
- Set B is in E.

Lower Bounds via Pseudo-Random Generators

- Let f be a $(c \log n, n)$-pseudo-random generator.
- Suppose B can be accepted by a circuit family of size $n = 2^{\frac{m}{2c}}$.
- Let C be a circuit from this family on $2m$ inputs.
- By definition of B, C accepts at most 2^m inputs.
- On the other hand, C accepts all prefixes of $f(y)$ of length $2m$ for $|y| = m$.
- Contradicts pseudo-randomness of f.

Lower Bounds via Pseudo-Random Generators

- Let f be a $(c \log n, n)$-pseudo-random generator.
- Suppose B can be accepted by a circuit family of size $n = 2^{\frac{m}{2c}}$.
- Let C be a circuit from this family on $2m$ inputs.
- By definition of B, C accepts at most 2^m inputs.
- On the other hand, C accepts all prefixes of $f(y)$ of length $2m$ for $|y| = m$.
- Contradicts pseudo-randomness of f.

Lower Bounds via Pseudo-Random Generators

- Let f be a $(c \log n, n)$-pseudo-random generator.
- Suppose B can be accepted by a circuit family of size $n = 2^{\frac{m}{2c}}$.
- Let C be a circuit from this family on $2m$ inputs.
- By definition of B, C accepts at most 2^m inputs.
- On the other hand, C accepts all prefixes of $f(y)$ of length $2m$ for $|y| = m$.
- Contradicts pseudo-randomness of f.

Lower Bounds via Pseudo-Random Generators

- Let f be a $(c \log n, n)$-pseudo-random generator.
- Suppose B can be accepted by a circuit family of size $n = 2^{\frac{m}{2c}}$.
- Let C be a circuit from this family on $2m$ inputs.
- By definition of B, C accepts at most 2^m inputs.
- On the other hand, C accepts all prefixes of $f(y)$ of length $2m$ for $|y| = m$.
- Contradicts pseudo-randomness of f.

Lower Bounds via Pseudo-Random Generators

- Let f be a $(c \log n, n)$-pseudo-random generator.
- Suppose B can be accepted by a circuit family of size $n = 2^{\frac{m}{2c}}$.
- Let C be a circuit from this family on $2m$ inputs.
- By definition of B, C accepts at most 2^m inputs.
- On the other hand, C accepts all prefixes of $f(y)$ of length $2m$ for $|y| = m$.
- Contradicts pseudo-randomness of f.

Lower Bounds via Pseudo-Random Generators

- Let f be a $(c \log n, n)$-pseudo-random generator.
- Suppose B can be accepted by a circuit family of size $n = 2^{\frac{m}{2c}}$.
- Let C be a circuit from this family on $2m$ inputs.
- By definition of B, C accepts at most 2^m inputs.
- On the other hand, C accepts all prefixes of $f(y)$ of length $2m$ for $|y| = m$.
- Contradicts pseudo-randomness of f.

- Thus we get that sets in the class E require exponential sized circuits.
- One can vary the depth and time-complexity of the generator to obtain different lower bounds.

Lower Bounds via Pseudo-Random Generators

- Thus we get that sets in the class E require exponential sized circuits.
- One can vary the depth and time-complexity of the generator to obtain different lower bounds.

Equivalence of Lower Bounds and Pseudo-Random Generators

Theorem (Håstad-Impagliazzo-Levin-Luby (1990))

There exist $m^{O(1)}$-computable $(n^{o(1)}, n)$-pseudo-random generators against $\mathcal{C}(n, n)$ iff there exist one-way functions.

One-way functions are functions computable in polynomial-time whose inverse is hard-to-compute.

Equivalence of Lower Bounds and Pseudo-Random Generators

Theorem (Impagliazzo-Wigderson,1997)

There exist $2^{O(m)}$-computable optimal pseudo-random generators against $\mathcal{C}(n,n)$ iff there exist sets in E that cannot be computed by subexponential-sized circuit family.

In both the results, proving the 'if' direction required a lot of work.

Equivalence of Lower Bounds and Pseudo-Random Generators

Theorem (Impagliazzo-Wigderson,1997)

There exist $2^{O(m)}$-computable optimal pseudo-random generators against $\mathcal{C}(n, n)$ iff there exist sets in E that cannot be computed by subexponential-sized circuit family.

In both the results, proving the 'if' direction required a lot of work.

Outline

1. Lower Bounds History
2. Pseudo-Random Generators
3. Applications of Time-Bounded Pseudo-Random Generators
   - Derandomizing Randomized Algorithms
   - Formalizing Cryptographic Security
   - Lower Bounds
4. **Lower Bounds on Boolean Circuits**
5. Lower Bounds on Arithmetic Circuits

Why Should Pseudo-Random Generators be Any Easier to Construct?

Pseudo-random generators avoid natural proof block.

- Since they imply lower bounds, they cannot satisfy natural proof axioms.
- Checking if a truth-table codes an optimal pseudo-random function is in PH.

Why Should Pseudo-Random Generators be Any Easier to Construct?

Pseudo-random generators avoid natural proof block.

- Since they imply lower bounds, they cannot satisfy natural proof axioms.
- Checking if a truth-table codes an optimal pseudo-random function is in PH.

Why Should Pseudo-Random Generators be Any Easier to Construct?

Pseudo-random generators avoid natural proof block.

- Since they imply lower bounds, they cannot satisfy natural proof axioms.
- Checking if a truth-table codes an optimal pseudo-random function is in PH.

Why Should Pseudo-Random Generators be Any Easier to Construct?

Some techniques in circuit model are known to be non-relativizable, e.g., Håstad's Switching Lemma.

Why Should Pseudo-Random Generators be Any Easier to Construct?

The problem is of designing an algorithm.

- We know that optimal pseudo-random generators can be computed in $2^{O(m)}$ space.
- We need to improve it to $2^{O(m)}$ time.

Why Should Pseudo-Random Generators be Any Easier to Construct?

The problem is of designing an algorithm.

- We know that optimal pseudo-random generators can be computed in $2^{O(m)}$ space.
- We need to improve it to $2^{O(m)}$ time.

Why Should Pseudo-Random Generators be Any Easier to Construct?

The problem is of designing an algorithm.

- We know that optimal pseudo-random generators can be computed in $2^{O(m)}$ space.
- We need to improve it to $2^{O(m)}$ time.

Why Should Pseudo-Random Generators be Any Easier to Construct?

There are a number of derandomization primitives available, e.g., extractors, expanders, pairwise independence.

- Expander graphs were recently used by Reingold (2005) to derandomize searching in undirected graphs proving SL = L.

A Possible Way of Proving P $\neq$ NP

- We now give a stepwise approach to prove P $\neq$ NP.
- It is based on construction of successively stronger optimal pseudo-random generators.

A Possible Way of Proving P $\neq$ NP

- We now give a stepwise approach to prove $\mathsf{P} \neq \mathsf{NP}$.
- It is based on construction of successively stronger optimal pseudo-random generators.

First Step: Against Constant Depth Circuits

- Håstad (1986) proved that PARITY cannot be accepted by depth d circuits of size $2^{n^{1/14d}}$.
- By Nisan-Wigderson (1987), this yields a $m^{O(1)}$-computable, $(\log^{O(d)} n, n)$-pseudo-random generator against $\mathcal{C}(n, d)$.
- This is almost an optimal pseudo-random generator – the seed length is $\log^{O(d)} n$ instead of $O(\log n)$.

First Step: Against Constant Depth Circuits

- Håstad (1986) proved that PARITY cannot be accepted by depth d circuits of size $2^{n^{1/14d}}$.
- By Nisan-Wigderson (1987), this yields a $m^{O(1)}$-computable, $(\log^{O(d)} n, n)$-pseudo-random generator against $\mathcal{C}(n, d)$.
- This is almost an optimal pseudo-random generator – the seed length is $\log^{O(d)} n$ instead of $O(\log n)$.

First Step: Against Constant Depth Circuits

- Håstad (1986) proved that PARITY cannot be accepted by depth d circuits of size $2^{n^{1/14d}}$.
- By Nisan-Wigderson (1987), this yields a $m^{O(1)}$-computable, $(\log^{O(d)} n, n)$-pseudo-random generator against $\mathcal{C}(n, d)$.
- This is almost an optimal pseudo-random generator – the seed length is $\log^{O(d)} n$ instead of $O(\log n)$.

First Step: Against Constant Depth Circuits

Step 1.

For each $d > 0$, construct a $2^{O(m)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, d)$.

First Step: Against Constant Depth Circuits

There exists a $2^{O(m)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, d)$

$$\Downarrow$$

There is a set B in E that cannot be accepted by any subexponential sized depth d circuit family

$$\Downarrow$$

B cannot be accepted by any $n^{d-\epsilon}$ size, $(d - \epsilon)\log n$ depth circuit family with bounded fanin AND gates for any $\epsilon > 0$

First Step: Against Constant Depth Circuits

There exists a $2^{O(m)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, d)$

$$\Downarrow$$

There is a set B in E that cannot be accepted by any subexponential sized depth d circuit family

$$\Downarrow$$

B cannot be accepted by any $n^{d-\epsilon}$ size, $(d - \epsilon) \log n$ depth circuit family with bounded fanin AND gates for any $\epsilon > 0$

First Step: Against Constant Depth Circuits

There exists a $2^{O(m)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, d)$

$$\Downarrow$$

There is a set B in E that cannot be accepted by any subexponential sized depth d circuit family

$$\Downarrow$$

B cannot be accepted by any $n^{d-\epsilon}$ size, $(d-\epsilon)\log n$ depth circuit family with bounded fanin AND gates for any $\epsilon > 0$

Second Step: Improve the Time Complexity

Step 2.

For each $d > 0$, construct a $m^{O(1)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, d)$.

Second Step: Improve the Time Complexity

- These generators yield hard sets in the class NP instead of E.
- For example, the generator aginst depth d circuits yields a set in NP that cannot be accepted by any $n^{d-\epsilon}$ size, $(d-\epsilon)\log n$ depth circuit family with bounded fanin AND gates.

Second Step: Improve the Time Complexity

- These generators yield hard sets in the class NP instead of E.
- For example, the generator aginst depth d circuits yields a set in NP that cannot be accepted by any $n^{d-\epsilon}$ size, $(d-\epsilon)\log n$ depth circuit family with bounded fanin AND gates.

Third Step: Enlarge the Class of Circuits

Step 3.

Construct a $m^{O(1)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, \log n)$.

Third Step: Enlarge the Class of Circuits

- Although the increase in depth is small, it improves the lower bound enormously because of inherent exponentiation.
- The generator implies that NP cannot be accepted by any family of sublinear depth and subexponential sized circuits.
- In particular, NC $\neq$ NP.

Third Step: Enlarge the Class of Circuits

- Although the increase in depth is small, it improves the lower bound enormously because of inherent exponentiation.
- The generator implies that NP cannot be accepted by any family of sublinear depth and subexponential sized circuits.
- In particular, NC $\neq$ NP.

Third Step: Enlarge the Class of Circuits

- Although the increase in depth is small, it improves the lower bound enormously because of inherent exponentiation.
- The generator implies that NP cannot be accepted by any family of sublinear depth and subexponential sized circuits.
- In particular, $\mathsf{NC} \neq \mathsf{NP}$.

Fourth Step: Further Enlarge the Class of Circuits

Step 4.

Construct a $m^{O(1)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, \log^{O(1)} n)$.

Fourth Step: Further Enlarge the Class of Circuits

- Again, because of exponentiation, this implies that NP cannot be accepted by any family of polynomial depth and subexponential sized circuits.
- In particular, P $\neq$ NP.

Fourth Step: Further Enlarge the Class of Circuits

- Again, because of exponentiation, this implies that NP cannot be accepted by any family of polynomial depth and subexponential sized circuits.
- In particular, P $\neq$ NP.

Current Status

- We known $m^{O(1)}$-computable optimal pseudo-random generator against $\mathcal{C}(n,2)$, the class of depth two circuits.
- The construction does not appear to generalize to even to depth three circuits.
- So there is a long way to go!

Current Status

- We known $m^{O(1)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, 2)$, the class of depth two circuits.
- The construction does not appear to generalize to even to depth three circuits.
- So there is a long way to go!

Current Status

- We known $m^{O(1)}$-computable optimal pseudo-random generator against $\mathcal{C}(n, 2)$, the class of depth two circuits.
- The construction does not appear to generalize to even to depth three circuits.
- So there is a long way to go!

Outline

1. Lower Bounds History
2. Pseudo-Random Generators
3. Applications of Time-Bounded Pseudo-Random Generators
   - Derandomizing Randomized Algorithms
   - Formalizing Cryptographic Security
   - Lower Bounds
4. Lower Bounds on Boolean Circuits
5. **Lower Bounds on Arithmetic Circuits**

Arithmetic Circuits

- Arithmetic circuits over field F are circuits with addition, subtraction, and multiplication gates.
- These compute a polynomial over the field F.
- A number of algrbraic problems admit arithmetic circuits.
- For example, computing determinant, finding roots of a polynomial, finding short vectors in a lattice etc.

Arithmetic Circuits

- Arithmetic circuits over field F are circuits with addition, subtraction, and multiplication gates.
- These compute a polynomial over the field F.
- A number of algrbraic problems admit arithmetic circuits.
- For example, computing determinant, finding roots of a polynomial, finding short vectors in a lattice etc.

Arithmetic Circuits

- Arithmetic circuits over field F are circuits with addition, subtraction, and multiplication gates.
- These compute a polynomial over the field F.
- A number of algrbraic problems admit arithmetic circuits.
- For example, computing determinant, finding roots of a polynomial, finding short vectors in a lattice etc.

Power of Arithmetic Circuits

- Polynomial sized arithmetic circuits can solve all the above problems.
- They can also be easily simulated by boolean circuits of similar size.
- The converse is unlikely as shown by Valiant et. al. (1983):
  - A polynomial sized arithmetic circuit of polynomial degree can be transformed to polynomial sized arithmetic circuit of logarithmic depth and fanin two multiplication gates.

Power of Arithmetic Circuits

- Polynomial sized arithmetic circuits can solve all the above problems.
- They can also be easily simulated by boolean circuits of similar size.
- The converse is unlikely as shown by Valiant et. al. (1983):
  - A polynomial sized arithmetic circuit of polynomial degree can be transformed to polynomial sized arithmetic circuit of logarithmic depth and fanin two multiplication gates.

Power of Arithmetic Circuits

- Polynomial sized arithmetic circuits can solve all the above problems.
- They can also be easily simulated by boolean circuits of similar size.
- The converse is unlikely as shown by Valiant et. al. (1983):
  - A polynomial sized arithmetic circuit of polynomial degree can be transformed to polynomial sized arithmetic circuit of logarithmic depth and fanin two multiplication gates.

Lower Bounds on Arithmetic Circuits

- Due to their algebraic structure, it appears that obtaining lower bounds on the arithmetic circuits should be easier.
- It has not happened so far!
- We do not even know lower bounds on constant depth arithmetic circuits!

Lower Bounds on Arithmetic Circuits

- Due to their algebraic structure, it appears that obtaining lower bounds on the arithmetic circuits should be easier.
- It has not happened so far!
- We do not even know lower bounds on constant depth arithmetic circuits!

Lower Bounds on Arithmetic Circuits

- Due to their algebraic structure, it appears that obtaining lower bounds on the arithmetic circuits should be easier.
- It has not happened so far!
- We do not even know lower bounds on constant depth arithmetic circuits!

Identity Testing and Lower Bounds

- Identity Testing problem is that given a polynomial computed by an arithmetic circuit, test if the polynomial is identically zero.
- It is a classical problem and there exist a number of randomized polynomial time algorithms for solving it.
- Kabanets-Impagliazzo (2003) showed that a derandomization of identity testing problem implies a lower bound on arithmetic circuits!
- We strengthen this relationship by defining pseudo-random generators against arithmetic circuits.

Identity Testing and Lower Bounds

- Identity Testing problem is that given a polynomial computed by an arithmetic circuit, test if the polynomial is identically zero.
- It is a classical problem and there exist a number of randomized polynomial time algorithms for solving it.
- Kabanets-Impagliazzo (2003) showed that a derandomization of identity testing problem implies a lower bound on arithmetic circuits!
- We strengthen this relationship by defining pseudo-random generators against arithmetic circuits.

Identity Testing and Lower Bounds

- Identity Testing problem is that given a polynomial computed by an arithmetic circuit, test if the polynomial is identically zero.
- It is a classical problem and there exist a number of randomized polynomial time algorithms for solving it.
- Kabanets-Impagliazzo (2003) showed that a derandomization of identity testing problem implies a lower bound on arithmetic circuits!
- We strengthen this relationship by defining pseudo-random generators against arithmetic circuits.

Identity Testing and Lower Bounds

- Identity Testing problem is that given a polynomial computed by an arithmetic circuit, test if the polynomial is identically zero.
- It is a classical problem and there exist a number of randomized polynomial time algorithms for solving it.
- Kabanets-Impagliazzo (2003) showed that a derandomization of identity testing problem implies a lower bound on arithmetic circuits!
- We strengthen this relationship by defining pseudo-random generators against arithmetic circuits.

Pseudo-Random Generators Against Arithmetic Circuits

Let $\mathcal{A}(n, F)$ be a subclass of size n arithmetic circuits over field F.

Let $f : \mathbb{N} \mapsto (F[y])^*$ be a function such that $f(n) = (f_1(y), \ldots, f_n(y), g(y))$ for all n.

Pseudo-Random Generators Against Arithmetic Circuits

Function f is an efficiently computable optimal pseudo-random generator against $\mathcal{A}(n, F)$ if

- Each $f_i(y)$ and $g(y)$ is of degree $n^{O(1)}$.
- Each $f_i(y)$ and $g(y)$ is computable in time $n^{O(1)}$.
- For any circuit $C \in \mathcal{A}(n, F)$ with $m \leq n$ inputs:

$$C(x_1, x_2, \ldots, x_m) = 0 \text{ iff } C(f_1(y), f_2(y), \ldots, f_m(y)) = 0 \ (mod\ g(y)).$$

Pseudo-Random Generators Against Arithmetic Circuits

Function f is an efficiently computable optimal pseudo-random generator against $\mathcal{A}(n, F)$ if

- Each $f_i(y)$ and $g(y)$ is of degree $n^{O(1)}$.
- Each $f_i(y)$ and $g(y)$ is computable in time $n^{O(1)}$.
- For any circuit $C \in \mathcal{A}(n, F)$ with $m \leq n$ inputs:

$$C(x_1, x_2, \ldots, x_m) = 0 \text{ iff } C(f_1(y), f_2(y), \ldots, f_m(y)) = 0 \ (mod\ g(y)).$$

Pseudo-Random Generators Against Arithmetic Circuits

Function f is an efficiently computable optimal pseudo-random generator against $\mathcal{A}(n, F)$ if

- Each $f_i(y)$ and $g(y)$ is of degree $n^{O(1)}$.
- Each $f_i(y)$ and $g(y)$ is computable in time $n^{O(1)}$.
- For any circuit $C \in \mathcal{A}(n, F)$ with $m \leq n$ inputs:

$$C(x_1, x_2, \ldots, x_m) = 0 \text{ iff } C(f_1(y), f_2(y), \ldots, f_m(y)) = 0 \ (mod\ g(y)).$$

Pseudo-Random Generators Against Arithmetic Circuits

Function f is an efficiently computable optimal pseudo-random generator against $\mathcal{A}(n, F)$ if

- Each $f_i(y)$ and $g(y)$ is of degree $n^{O(1)}$.
- Each $f_i(y)$ and $g(y)$ is computable in time $n^{O(1)}$.
- For any circuit $C \in \mathcal{A}(n, F)$ with $m \leq n$ inputs:

$$C(x_1, x_2, \ldots, x_m) = 0 \text{ iff } C(f_1(y), f_2(y), \ldots, f_m(y)) = 0 \ (mod\ g(y)).$$

Pseudo-Random Generators Against Arithmetic Circuits

- Schwartz-Zippel lemma shows that optimal pseudo-random generators exist against the entire class of size n circuits.
  - Of course, these are not efficiently computable.
- If there exist efficiently computable optimal pseudo-random generators against the entire class of size n circuits then:
  - The identity testing problem can be solved in deterministic polynomial-time.
  - There exists a multilinear polynomial in PSPACE that cannot be computed by subexponential sized arithmetic circuits.

Pseudo-Random Generators Against Arithmetic Circuits

- Schwartz-Zippel lemma shows that optimal pseudo-random generators exist against the entire class of size n circuits.
  - Of course, these are not efficiently computable.
- If there exist efficiently computable optimal pseudo-random generators against the entire class of size n circuits then:
  - The identity testing problem can be solved in determinstic polynomial-time.
  - There exists a multilinear polynomial in PSPACE that cannot be computed by subexponential sized arithmetic circuits.

Pseudo-Random Generators Against Arithmetic Circuits

- Schwartz-Zippel lemma shows that optimal pseudo-random generators exist against the entire class of size n circuits.
  - Of course, these are not efficiently computable.
- If there exist efficiently computable optimal pseudo-random generators against the entire class of size n circuits then:
  - The identity testing problem can be solved in determinstic polynomial-time.
  - There exists a multilinear polynomial in PSPACE that cannot be computed by subexponential sized arithmetic circuits.

Pseudo-Random Generators Against Arithmetic Circuits

- Schwartz-Zippel lemma shows that optimal pseudo-random generators exist against the entire class of size n circuits.
  - Of course, these are not efficiently computable.
- If there exist efficiently computable optimal pseudo-random generators against the entire class of size n circuits then:
  - The identity testing problem can be solved in determinstic polynomial-time.
  - There exists a multilinear polynomial in PSPACE that cannot be computed by subexponential sized arithmetic circuits.

Pseudo-Random Generators Imply Lower Bounds

- Suppose f is an efficiently computable optimal pseudo-random generator against $\mathcal{A}(n, F)$.
- Let the degree of all polynomials in $f_1(y)$, ..., $f_n(y)$ be bounded by $d = n^{O(1)}$ and $m = \log d$.
- Define polynomial q as:

$$q(x_1, x_2, \ldots, x_{2m}) = \sum_{S \subseteq [1,m]} c_S \prod_{i \in S} x_i.$$

- Here $c_S \in F$ satisfying:

$$\sum_{S \subseteq [1,m]} c_S \prod_{i \in S} f_i(y) = 0.$$

Pseudo-Random Generators Imply Lower Bounds

- Suppose f is an efficiently computable optimal pseudo-random generator against $\mathcal{A}(n, F)$.
- Let the degree of all polynomials in $f_1(y)$, ..., $f_n(y)$ be bounded by $d = n^{O(1)}$ and $m = \log d$.
- Define polynomial q as:

$$q(x_1, x_2, \ldots, x_{2m}) = \sum_{S \subseteq [1,m]} c_S \prod_{i \in S} x_i.$$

- Here $c_S \in F$ satisfying:

$$\sum_{S \subseteq [1,m]} c_S \prod_{i \in S} f_i(y) = 0.$$

Pseudo-Random Generators Imply Lower Bounds

- Suppose f is an efficiently computable optimal pseudo-random generator against $\mathcal{A}(n, F)$.
- Let the degree of all polynomials in $f_1(y)$, ..., $f_n(y)$ be bounded by $d = n^{O(1)}$ and $m = \log d$.
- Define polynomial q as:

$$q(x_1, x_2, \ldots, x_{2m}) = \sum_{S \subseteq [1,m]} c_S \prod_{i \in S} x_i.$$

- Here $c_S \in F$ satisfying:

$$\sum_{S \subseteq [1,m]} c_S \prod_{i \in S} f_i(y) = 0.$$

Pseudo-Random Generators Imply Lower Bounds

- Suppose f is an efficiently computable optimal pseudo-random generator against $\mathcal{A}(n, F)$.
- Let the degree of all polynomials in $f_1(y)$, ..., $f_n(y)$ be bounded by $d = n^{O(1)}$ and $m = \log d$.
- Define polynomial q as:

$$q(x_1, x_2, \ldots, x_{2m}) = \sum_{S \subseteq [1,m]} c_S \prod_{i \in S} x_i.$$

- Here $c_S \in F$ satisfying:

$$\sum_{S \subseteq [1,m]} c_S \prod_{i \in S} f_i(y) = 0.$$

Pseudo-Random Generators Imply Lower Bounds

- A non-zero q always exists:
  - Number of coefficients c_S are exactly $2^{2m} = d^2$.
  - These need to satisfy a polynomial equation of degree at most $2m2^m = 2d \log d$.
  - This requires satisfying $2d \log d + 1$ homogeneous constraints.
  - Since $d^2 > 2d \log d + 1$ for $d \geq 8$, this is always possible.
- Polynomial q can be computed by solving a system of $2^{O(m)}$ linear equations, thus is computable in PSPACE.

Pseudo-Random Generators Imply Lower Bounds

- A non-zero q always exists:
  - Number of coefficients c_S are exactly $2^{2m} = d^2$.
  - These need to satisfy a polynomial equation of degree at most $2m2^m = 2d \log d$.
  - This requires satisfying $2d \log d + 1$ homogeneous constraints.
  - Since $d^2 > 2d \log d + 1$ for $d \geq 8$, this is always possible.
- Polynomial q can be computed by solving a system of $2^{O(m)}$ linear equations, thus is computable in PSPACE.

Pseudo-Random Generators Imply Lower Bounds

- A non-zero q always exists:
  - Number of coefficients c_S are exactly $2^{2m} = d^2$.
  - These need to satisfy a polynomial equation of degree at most $2m2^m = 2d\log d$.
  - This requires satisfying $2d\log d + 1$ homogeneous constraints.
  - Since $d^2 > 2d\log d + 1$ for $d \geq 8$, this is always possible.
- Polynomial q can be computed by solving a system of $2^{O(m)}$ linear equations, thus is computable in PSPACE.

Pseudo-Random Generators Imply Lower Bounds

- A non-zero q always exists:
  - Number of coefficients c_S are exactly $2^{2m} = d^2$.
  - These need to satisfy a polynomial equation of degree at most $2m2^m = 2d \log d$.
  - This requires satisfying $2d \log d + 1$ homogeneous constraints.
  - Since $d^2 > 2d \log d + 1$ for $d \geq 8$, this is always possible.
- Polynomial q can be computed by solving a system of $2^{O(m)}$ linear equations, thus is computable in PSPACE.

Pseudo-Random Generators Imply Lower Bounds

- A non-zero q always exists:
  - Number of coefficients c_S are exactly $2^{2m} = d^2$.
  - These need to satisfy a polynomial equation of degree at most $2m2^m = 2d \log d$.
  - This requires satisfying $2d \log d + 1$ homogeneous constraints.
  - Since $d^2 > 2d \log d + 1$ for $d \geq 8$, this is always possible.
- Polynomial q can be computed by solving a system of $2^{O(m)}$ linear equations, thus is computable in PSPACE.

Pseudo-Random Generators Imply Lower Bounds

- A non-zero q always exists:
  - Number of coefficients c_S are exactly $2^{2m} = d^2$.
  - These need to satisfy a polynomial equation of degree at most $2m2^m = 2d \log d$.
  - This requires satisfying $2d \log d + 1$ homogeneous constraints.
  - Since $d^2 > 2d \log d + 1$ for $d \geq 8$, this is always possible.
- Polynomial q can be computed by solving a system of $2^{O(m)}$ linear equations, thus is computable in PSPACE.

Pseudo-Random Generators Imply Lower Bounds

- Suppose that q can be computed by a circuit C in $\mathcal{A}(n, F)$.
- By definition of q, $C(f_1(y), f_2(y), \ldots, f_{2m(y)}) = 0$.
- However, $C(x_1, x_2, \ldots, x_{2m})$ is non-zero.
- This contradicts pseudo-randomness of f.

Pseudo-Random Generators Imply Lower Bounds

- Suppose that q can be computed by a circuit C in $\mathcal{A}(n, F)$.
- By definition of q, $C(f_1(y), f_2(y), \ldots, f_{2m(y)}) = 0$.
- However, $C(x_1, x_2, \ldots, x_{2m})$ is non-zero.
- This contradicts pseudo-randomness of f.

Pseudo-Random Generators Imply Lower Bounds

- Suppose that q can be computed by a circuit C in $\mathcal{A}(n, F)$.
- By definition of q, $C(f_1(y), f_2(y), \ldots, f_{2m(y)}) = 0$.
- However, $C(x_1, x_2, \ldots, x_{2m})$ is non-zero.
- This contradicts pseudo-randomness of f.

Pseudo-Random Generators Imply Lower Bounds

- Suppose that q can be computed by a circuit C in $\mathcal{A}(n, F)$.
- By definition of q, $C(f_1(y), f_2(y), \ldots, f_{2m(y)}) = 0$.
- However, $C(x_1, x_2, \ldots, x_{2m})$ is non-zero.
- This contradicts pseudo-randomness of f.

Why Should Pseudo-Random generators be Easier to Construct?

- A-Kayal-Saxena (2002) constructed an efficiently computable optimal pseudo-random generator against a very special class of circuits.
- This contained circuits computing the polynomial $(1+x)^m - x^m - 1$ over ring Z_m.
- The pseudo-random generator was:

$$f(n) = (x, x, \ldots, x, g(x)), g(x) = x^{16n^5} \prod_{r=1}^{16n^5} \prod_{a=1}^{4n^4} ((x-a)^r - 1).$$

- This derandomized a primality testing algorithm.

Why Should Pseudo-Random generators be Easier to Construct?

- A-Kayal-Saxena (2002) constructed an efficiently computable optimal pseudo-random generator against a very special class of circuits.
- This contained circuits computing the polynomial $(1+x)^m - x^m - 1$ over ring Z_m.
- The pseudo-random generator was:

$$f(n) = (x, x, \ldots, x, g(x)), g(x) = x^{16n^5} \prod_{r=1}^{16n^5} \prod_{a=1}^{4n^4} ((x-a)^r - 1).$$

- This derandomized a primality testing algorithm.

Why Should Pseudo-Random generators be Easier to Construct?

- A-Kayal-Saxena (2002) constructed an efficiently computable optimal pseudo-random generator against a very special class of circuits.
- This contained circuits computing the polynomial $(1+x)^m - x^m - 1$ over ring Z_m.
- The pseudo-random generator was:

$$f(n) = (x, x, \ldots, x, g(x)), g(x) = x^{16n^5} \prod_{r=1}^{16n^5} \prod_{a=1}^{4n^4} ((x-a)^r - 1).$$

- This derandomized a primality testing algorithm.

Why Should Pseudo-Random generators be Easier to Construct?

- A-Kayal-Saxena (2002) constructed an efficiently computable optimal pseudo-random generator against a very special class of circuits.
- This contained circuits computing the polynomial $(1+x)^m - x^m - 1$ over ring Z_m.
- The pseudo-random generator was:

$$f(n) = (x, x, \ldots, x, g(x)), g(x) = x^{16n^5} \prod_{r=1}^{16n^5} \prod_{a=1}^{4n^4} ((x-a)^r - 1).$$

- This derandomized a primality testing algorithm.

Why Should Pseudo-Random generators be Easier to Construct?

- A-Kayal-Saxena (2002) constructed an efficiently computable optimal pseudo-random generator against a very special class of circuits.
- This contained circuits computing the polynomial $(1+x)^m - x^m - 1$ over ring Z_m.
- The pseudo-random generator was:

$$f(n) = (x, x, \ldots, x, g(x)), g(x) = x^{16n^5} \prod_{r=1}^{16n^5} \prod_{a=1}^{4n^4} ((x-a)^r - 1).$$

- This derandomized a primality testing algorithm.

A Possible Way of Proving Hardness of Permanent

- The complexity of computing permanent of a matrix characterizes the class #P.
- #P is the arithmetic analog of the class NP.
- We give a stepwise approach to prove hardness of permanent.
- As before, it is based on constructing successively stronger optimal pseudo-random generators.

A Possible Way of Proving Hardness of Permanent

- The complexity of computing permanent of a matrix characterizes the class #P.
- #P is the arithmetic analog of the class NP.
- We give a stepwise approach to prove hardness of permanent.
- As before, it is based on constructing successively stronger optimal pseudo-random generators.

A Possible Way of Proving Hardness of Permanent

- The complexity of computing permanent of a matrix characterizes the class #P.
- #P is the arithmetic analog of the class NP.
- We give a stepwise approach to prove hardness of permanent.
- As before, it is based on constructing successively stronger optimal pseudo-random generators.

First Step: Against Constant Depth Circuits

Step 1.

For each $d > 0$, construct an efficiently computable optimal pseudo-random generator against the class of size n, depth d arithmetic circuits.

First Step: Against Constant Depth Circuits

There exists an efficiently computable optimal pseudo-random generator against the class of size n, depth d arithmetic circuits

$$\Downarrow$$

There is a multilinear polynomial q computable in PSPACE that cannot be computed by subexponential sized, depth d circuits

$$\Downarrow$$

Polynomial q cannot be computed by any size $n^{d-\epsilon}$, depth $(d-\epsilon)\log n$ circuit family with bounded fanin multiplication gates

First Step: Against Constant Depth Circuits

There exists an efficiently computable optimal pseudo-random generator against the class of size n, depth d arithmetic circuits

$$\Downarrow$$

There is a multilinear polynomial q computable in PSPACE that cannot be computed by subexponential sized, depth d circuits

$$\Downarrow$$

Polynomial q cannot be computed by any size $n^{d-\epsilon}$, depth $(d-\epsilon)\log n$ circuit family with bounded fanin multiplication gates

First Step: Against Constant Depth Circuits

There exists an efficiently computable optimal pseudo-random generator against the class of size n, depth d arithmetic circuits

$$\Downarrow$$

There is a multilinear polynomial q computable in PSPACE that cannot be computed by subexponential sized, depth d circuits

$$\Downarrow$$

Polynomial q cannot be computed by any size $n^{d-\epsilon}$, depth $(d-\epsilon)\log n$ circuit family with bounded fanin multiplication gates

Second Step: Against Superconstant Depth Circuits

- The union over all d's spans all polynomial sized circuits!
- This motivates the second step.

Step 2.

Construct an efficiently computable optimal pseudo-random generator against the class of size n, depth $\omega(1)$ arithmetic circuits.

This yields a multilinear polynomial in PSPACE that requires superpolynomial sized arithmetic circuits.

Second Step: Against Superconstant Depth Circuits

- The union over all d's spans all polynomial sized circuits!
- This motivates the second step.

Step 2.

Construct an efficiently computable optimal pseudo-random generator against the class of size n, depth $\omega(1)$ arithmetic circuits.

This yields a multilinear polynomial in PSPACE that requires superpolynomial sized arithmetic circuits.

Second Step: Against Superconstant Depth Circuits

- The union over all d's spans all polynomial sized circuits!
- This motivates the second step.

Step 2.

Construct an efficiently computable optimal pseudo-random generator against the class of size n, depth $\omega(1)$ arithmetic circuits.

This yields a multilinear polynomial in PSPACE that requires superpolynomial sized arithmetic circuits.

Third Step: Improve Efficiency of the Generator

- Suppose each coefficient of the hard-to-compute multilinear polynomial given by a generator can be computed by a #P-function.
- Then the polynomial can be expressed as the permanent of a $O(m) \times O(m)$ matrix.
- Call such generators #P-computable.

Step 3.

Construct a #P-computable optimal pseudo-random generator against the class of size n, depth $\omega(1)$ arithmetic circuits.

Third Step: Improve Efficiency of the Generator

- Suppose each coefficient of the hard-to-compute multilinear polynomial given by a generator can be computed by a #P-function.
- Then the polynomial can be expressed as the permanent of a $O(m) \times O(m)$ matrix.
- Call such generators #P-computable.

Step 3.

Construct a #P-computable optimal pseudo-random generator against the class of size n, depth $\omega(1)$ arithmetic circuits.

Third Step: Improve Efficiency of the Generator

- Suppose each coefficient of the hard-to-compute multilinear polynomial given by a generator can be computed by a #P-function.
- Then the polynomial can be expressed as the permanent of a $O(m) \times O(m)$ matrix.
- Call such generators #P-computable.

Step 3.

Construct a #P-computable optimal pseudo-random generator against the class of size n, depth $\omega(1)$ arithmetic circuits.

Third Step: Improve Efficiency of the Generator

- Suppose each coefficient of the hard-to-compute multilinear polynomial given by a generator can be computed by a #P-function.
- Then the polynomial can be expressed as the permanent of a $O(m) \times O(m)$ matrix.
- Call such generators #P-computable.

Step 3.

Construct a #P-computable optimal pseudo-random generator against the class of size n, depth $\omega(1)$ arithmetic circuits.

Third Step: Improve Efficiency of the Generator

Such a generator implies that Permanent requires superpolynomial sized arithmetic circuits.

Current Status

- We know efficiently computable optimal pseudo-random generators against size n, depth two arithmetic circuits.
- Still some way to go!

Current Status

- We know efficiently computable optimal pseudo-random generators against size n, depth two arithmetic circuits.
- Still some way to go!

A Conjecture

- Define
$$F(n,k) = (y, y^k, y^{k^2}, \ldots, y^{k^{n-1}}, y^r - 1),$$
where $r \geq n^4$ is a prime and $1 \leq k < r$.

Conjecture

F is a #P-computable optimal pseudo-random generator against arithmetic circuits of size n and depth $\omega(1)$.

A Conjecture

- Define
$$F(n,k) = (y, y^k, y^{k^2}, \ldots, y^{k^{n-1}}, y^r - 1),$$
where $r \geq n^4$ is a prime and $1 \leq k < r$.

Conjecture

F is a #P-computable optimal pseudo-random generator against arithmetic circuits of size n and depth $\omega(1)$.

Predictions for the Future

By 2010.

All the steps for arithmetic circuits. [Proves hardness of Permanent]

By 2020.

First two steps for boolean circuits. [Proves NP requires exponential sized, constant depth circuits; should also prove $\mathsf{NC}^1 \neq \mathsf{NP}$]

By 2022.

Third step for boolean circuits. [Proves $\mathsf{NC} \neq \mathsf{NP}$]

By 2030.

Fourth step for boolean circuits. [Proves $\mathsf{P} \neq \mathsf{NP}$]

Predictions for the Future

By 2010.

All the steps for arithmetic circuits. [Proves hardness of Permanent]

By 2020.

First two steps for boolean circuits. [Proves NP requires exponential sized, constant depth circuits; should also prove $NC^1 \neq NP$]

By 2022.

Third step for boolean circuits. [Proves $NC \neq NP$]

By 2030.

Fourth step for boolean circuits. [Proves $P \neq NP$]

Predictions for the Future

By 2010.

All the steps for arithmetic circuits. [Proves hardness of Permanent]

By 2020.

First two steps for boolean circuits. [Proves NP requires exponential sized, constant depth circuits; should also prove $NC^1 \neq NP$]

By 2022.

Third step for boolean circuits. [Proves $NC \neq NP$]

By 2030.

Fourth step for boolean circuits. [Proves $P \neq NP$]

Predictions for the Future

By 2010.

All the steps for arithmetic circuits. [Proves hardness of Permanent]

By 2020.

First two steps for boolean circuits. [Proves NP requires exponential sized, constant depth circuits; should also prove $\mathsf{NC}^1 \neq \mathsf{NP}$]

By 2022.

Third step for boolean circuits. [Proves $\mathsf{NC} \neq \mathsf{NP}$]

By 2030.

Fourth step for boolean circuits. [Proves $\mathsf{P} \neq \mathsf{NP}$]

Predictions for the Future

By 2010.

All the steps for arithmetic circuits. [Proves hardness of Permanent]

By 2020.

First two steps for boolean circuits. [Proves NP requires exponential sized, constant depth circuits; should also prove $\mathsf{NC}^1 \neq \mathsf{NP}$]

By 2022.

Third step for boolean circuits. [Proves $\mathsf{NC} \neq \mathsf{NP}$]

By 2030.

Fourth step for boolean circuits. [Proves $\mathsf{P} \neq \mathsf{NP}$]

THANK YOU!