Algebraic Complexity Theory

Manindra Agrawal

IIT Kanpur

Symposium on Learning, Algorithms and Complexity, IISc Bangalore 2015
Overview

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
Outline

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
**Computation without Bits**

- An algorithm, in general, can use individual bits of the input in very complex ways. In particular, making execution decisions based on the values of a bit.

- Certain algorithms, however, use the individual bits in a much simpler way.

- Example: matrix multiplication. For \( c_{ij} = a_{ij} \cdot b_{ij} \), we have:

\[
c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}.
\]

- If we assume operations + and \( \ast \) as primitives, and the input being a sequence of numbers denoting entries of matrices, then the algorithm does not need to access bit values.
Computation without Bits

- An algorithm, in general, can use individual bits of the input in very complex ways. In particular, making execution decisions based on the values of a bit.
- Certain algorithms, however, use the individual bits in a much simpler way.
- Example: matrix multiplication. For \( c_{ij} = [a_{ij}] \cdot [b_{ij}] \), we have:

\[
c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}.
\]

- If we assume operations + and \( \cdot \) as primitives, and the input being a sequence of numbers denoting entries of matrices, then the algorithm does not need to access bit values.
Computation without Bits

- An algorithm, in general, can use individual bits of the input in very complex ways. In particular, making execution decisions based on the values of a bit.

- Certain algorithms, however, use the individual bits in a much simpler way.

- Example: matrix multiplication. For $c_{ij} = [a_{ij}] \cdot [b_{ij}]$, we have:

  $$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}.$$ 

- If we assume operations $+$ and $\cdot$ as primitives, and the input being a sequence of numbers denoting entries of matrices, then the algorithm does not need to access bit values.
Outline

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
We can formalize such computations as follows:

- Let $R$ be a ring with operations $+$ and $\ast$.
- Let the input be variables $x_1, x_2, \ldots, x_n$.
- An algorithm applies a sequence of ring operations on the input variables and constants from $R$.
- The output is a polynomial in $R[x_1, x_2, \ldots, x_n]$.

This is called arithmetic circuit model.
Computation without Bits

- We can formalize such computations as follows:
  - Let $R$ be a ring with operations $+$ and $\ast$.
  - Let the input be variables $x_1, x_2, \ldots, x_n$.
  - An algorithm applies a sequence of ring operations on the input variables and constants from $R$.
  - The output is a polynomial in $R[x_1, x_2, \ldots, x_n]$.
- This is called arithmetic circuit model.
We can formalize such computations as follows:

- Let $R$ be a ring with operations $+$ and $\ast$.
- Let the input be variables $x_1, x_2, \ldots, x_n$.
- An algorithm applies a sequence of ring operations on the input variables and constants from $R$.
- The output is a polynomial in $R[x_1, x_2, \ldots, x_n]$.

This is called arithmetic circuit model.
Computation without Bits

- We can formalize such computations as follows:
  - Let $R$ be a ring with operations $+$ and $\ast$.
  - Let the input be variables $x_1, x_2, \ldots, x_n$.
  - An algorithm applies a sequence of ring operations on the input variables and constants from $R$.
  - The output is a polynomial in $R[x_1, x_2, \ldots, x_n]$.

- This is called **arithmetic circuit model**.
An Example

Output:

$$\text{output} = (ux + vy)^2 + (vx - uy)^2 - (u^2 + v^2) \cdot (x^2 + y^2)$$
As in the boolean settings, arithmetic circuit model is a non-uniform model of computation.

For each problem, one has, therefore, an infinite family of circuits computing its solution.
Arithmetic Circuit Families

- As in the boolean settings, arithmetic circuit model is a non-uniform model of computation.
- For each problem, one has, therefore, an infinite family of circuits computing its solution.
Power of the Model

- The model proposed by [Valiant 1979].
- It can compute all of the following operations:
  - Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial, permanent
  - Polynomial operations: addition, multiplication
  - Multivariate polynomial factorization when the polynomial is fixed
Power of the Model

The model proposed by [Valiant 1979].

It can compute all of the following operations:

- Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial, permanent
- Polynomial operations: addition, multiplication
- Multivariate polynomial factorization when the polynomial is fixed
Power of the Model

- The model proposed by [Valiant 1979].
- It can compute all of the following operations:
  - Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial, permanent
  - Polynomial operations: addition, multiplication
  - Multivariate polynomial factorization when the polynomial is fixed
Power of the Model

- The model proposed by [Valiant 1979].
- It can compute all of the following operations:
  - Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial, permanent
  - Polynomial operations: addition, multiplication
  - Multivariate polynomial factorization when the polynomial is fixed
Power of the Model

- The model proposed by [Valiant 1979].
- It can compute all of the following operations:
  - Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial, permanent
  - Polynomial operations: addition, multiplication
  - Multivariate polynomial factorization when the polynomial is fixed
Crucial parameters associated with an arithmetic circuit are:

- **Input length**: number of input variables. Notice that the size of individual variables is not counted!
- **Size**: equals the number of operations in the circuit (measured as a function of input length).
- **Depth**: equals the length of the longest path from a variable to output of the circuit.
- **Degree**: equals the formal degree of circuit defined inductively as: 1 for input variables, max for addition gates, and sum for multiplication gates.
- **Fanin**: equals the largest number of inputs to a gate in the circuit. We allow arbitrary fanin.
Arithmetic Complexity

Crucial parameters associated with an arithmetic circuit are:

- **Input length**: number of input variables. Notice that the size of individual variables is not counted!
- **Size**: equals the number of operations in the circuit (measured as a function of input length).
- **Depth**: equals the length of the longest path from a variable to output of the circuit.
- **Degree**: equals the formal degree of circuit defined inductively as: 1 for input variables, max for addition gates, and sum for multiplication gates.
- **Fanin**: equals the largest number of inputs to a gate in the circuit. We allow arbitrary fanin.
Arithmetic Complexity

Crucial parameters associated with an arithmetic circuit are:

- **Input length**: number of input variables. Notice that the size of individual variables is not counted!
- **Size**: equals the number of operations in the circuit (measured as a function of input length).
- **Depth**: equals the length of the longest path from a variable to output of the circuit.
- **Degree**: equals the formal degree of circuit defined inductively as: 1 for input variables, max for addition gates, and sum for multiplication gates.
- **Fanin**: equals the largest number of inputs to a gate in the circuit. We allow arbitrary fanin.
Arithmetic Complexity

Crucial parameters associated with an arithmetic circuit are:

- **Input length**: number of input variables. Notice that the size of individual variables is not counted!
- **Size**: equals the number of operations in the circuit (measured as a function of input length).
- **Depth**: equals the length of the longest path from a variable to output of the circuit.
- **Degree**: equals the formal degree of circuit defined inductively as: 1 for input variables, max for addition gates, and sum for multiplication gates.
- **Fanin**: equals the largest number of inputs to a gate in the circuit. We allow arbitrary fanin.
Arithmetic Complexity

Crucial parameters associated with an arithmetic circuit are:

- **Input length**: number of input variables. Notice that the size of individual variables is not counted!
- **Size**: equals the number of operations in the circuit (measured as a function of input length).
- **Depth**: equals the length of the longest path from a variable to output of the circuit.
- **Degree**: equals the formal degree of circuit defined inductively as: 1 for input variables, max for addition gates, and sum for multiplication gates.
- **Fanin**: equals the largest number of inputs to a gate in the circuit. We allow arbitrary fanin.
Circuit Parameters

SIZE = 16  DEPTH = 4  DEGREE = 4  FANIN = 3
Outline

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
Extension with Zero-test

- Many other algebraic operations cannot be computed in arithmetic circuit model: solving system of linear equations, rank of a matrix, gcd of polynomials, primality testing . . .
- Generalize the model by including another operation: zero-test.
  - This is a branching operation: check if the input is zero; if yes do $A$ else do $B$.
- All the above operations can be computed in the new model.
Extension with Zero-test

• Many other algebraic operations cannot be computed in arithmetic circuit model: solving system of linear equations, rank of a matrix, gcd of polynomials, primality testing . . .

• Generalize the model by including another operation: zero-test.
  ▶ This is a branching operation: check if the input is zero; if yes do $A$ else do $B$.

• All the above operations can be computed in the new model.
Extension with Zero-test

- Many other algebraic operations cannot be computed in arithmetic circuit model: solving system of linear equations, rank of a matrix, gcd of polynomials, primality testing ...
- Generalize the model by including another operation: zero-test.
  - This is a branching operation: check if the input is zero; if yes do \( A \) else do \( B \).
- All the above operations can be computed in the new model.
BSS Model

- The generalized model can still not compute simple functions, e.g., "Is $x < y$?"
- [Blum-Shub-Smale 1989] replaced zero-test with $\leq$ operator.
  - The operator makes sense only in rings with a total ordering, e.g., $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$.
- They showed that the model, for $R = \mathbb{Z}$ or $\mathbb{Q}$ restores access to bits, and is therefore equivalent to the standard boolean model.
- For $R = \mathbb{R}$, they developed a new theory of complexity.
- We will not consider this model any further.
The generalized model can still not compute simple functions, e.g., "Is $x < y$?"

[Blum-Shub-Smale 1989] replaced zero-test with $\leq$ operator.

- The operator makes sense only in rings with a total ordering, e.g., $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$.

They showed that the model, for $R = \mathbb{Z}$ or $\mathbb{Q}$ restores access to bits, and is therefore equivalent to the standard boolean model.

For $R = \mathbb{R}$, they developed a new theory of complexity.

We will not consider this model any further.
The generalized model can still not compute simple functions, e.g., ”Is $x < y$?”

[Blum-Shub-Smale 1989] replaced zero-test with $\leq$ operator.

- The operator makes sense only in rings with a total ordering, e.g., $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$.

They showed that the model, for $R = \mathbb{Z}$ or $\mathbb{Q}$ restores access to bits, and is therefore equivalent to the standard boolean model.

- For $R = \mathbb{R}$, they developed a new theory of complexity.
- We will not consider this model any further.
BSS Model

- The generalized model can still not compute simple functions, e.g., "Is $x < y$?"
- [Blum-Shub-Smale 1989] replaced zero-test with $\leq$ operator.
  - The operator makes sense only in rings with a total ordering, e.g., $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$.
- They showed that the model, for $R = \mathbb{Z}$ or $\mathbb{Q}$ restores access to bits, and is therefore equivalent to the standard boolean model.
- For $R = \mathbb{R}$, they developed a new theory of complexity.
- We will not consider this model any further.
The generalized model can still not compute simple functions, e.g., "Is \( x < y \)?"

[Blum-Shub-Smale 1989] replaced zero-test with \( \leq \) operator.
- The operator makes sense only in rings with a total ordering, e.g., \( \mathbb{Z} \), \( \mathbb{Q} \), \( \mathbb{R} \).

They showed that the model, for \( R = \mathbb{Z} \) or \( \mathbb{Q} \) restores access to bits, and is therefore equivalent to the standard boolean model.

For \( R = \mathbb{R} \), they developed a new theory of complexity.

We will not consider this model any further.
Outline

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
The Class P

- For both the models, the class $P$ can be defined in an analogous way to boolean settings: all problems that can be solved by a circuit family of polynomial size.

- In the arithmetic circuit model, a problem is simply a family of polynomials, typically parameterized by the number of variables, or degree, or both:
  - Chebyshev polynomials
    \[
    T_d(x) = \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{d}{2k} (x^2 - 1)^k x^{d-2k}
    \]
    by degree,
  - Determinant polynomial by number of variables, and
  - Elementary symmetric polynomials
    \[
    S_d(x_1, x_2, \ldots, x_n) = \sum_{I \subseteq [1, n], |I|=d} \prod_{j \in I} x_j
    \]
    by both degree and number of variables.
The Class P

- For both the models, the class $P$ can be defined in an analogous way to boolean settings: all problems that can be solved by a circuit family of polynomial size.
- In the arithmetic circuit model, a problem is simply a family of polynomials, typically parameterized by the number of variables, or degree, or both:
  - Chebyshev polynomials
    \[
    T_d(x) = \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{d}{2k} (x^2 - 1)^k x^{d-2k}
    \]
    by degree,
  - Determinant polynomial by number of variables, and
  - Elementary symmetric polynomials
    \[
    S_d(x_1, x_2, \ldots, x_n) = \sum_{I \subseteq [1,n], |I| = d} \prod_{j \in I} x_j,
    \]
    by both degree and number of variables.
**The Class P**

- For both the models, the class $\mathbb{P}$ can be defined in an analogous way to boolean settings: *all problems that can be solved by a circuit family of polynomial size.*

- In the arithmetic circuit model, a problem is simply a family of polynomials, typically parameterized by the number of variables, or degree, or both:
  - Chebyshev polynomials
    \[
    \sum_{k=0}^{[d/2]} \binom{d}{2k} (x^2 - 1)^k x^{d-2k}
    \]
    by degree,
  - Determinant polynomial by number of variables, and
  - Elementary symmetric polynomials
    \[
    \sum_{I \subseteq [1,n], |I|=d} \prod_{j \in I} x_j,
    \]
    by both degree and number of variables.
The Class P

- For both the models, the class $\mathbf{P}$ can be defined in an analogous way to boolean settings: all problems that can be solved by a circuit family of polynomial size.

- In the arithmetic circuit model, a problem is simply a family of polynomials, typically parameterized by the number of variables, or degree, or both:
  
  ▶ Chebyshev polynomials

  $$T_d(x) = \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{d}{2k} (x^2 - 1)^k x^{d-2k}$$

  by degree,

  ▶ Determinant polynomial by number of variables, and

  ▶ Elementary symmetric polynomials

  $$S_d(x_1, x_2, \ldots, x_n) = \sum_{I \subseteq \{1, \ldots, n\}, |I| = d} \prod_{j \in I} x_j,$$

  by both degree and number of variables.
The Class \( P \)

- For both the models, the class \( P \) can be defined in an analogous way to boolean settings: all problems that can be solved by a circuit family of polynomial size.

- In the arithmetic circuit model, a problem is simply a family of polynomials, typically parameterized by the number of variables, or degree, or both:
  - Chebyshev polynomials

\[
T_d(x) = \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{d}{2k} (x^2 - 1)^k x^{d-2k}
\]

- by degree,
  - Determinant polynomial by number of variables, and
  - Elementary symmetric polynomials

\[
S_d(x_1, x_2, \ldots, x_n) = \sum_{l \subseteq [1, n], |l| = d} \prod_{j \in l} x_j,
\]

- by both degree and number of variables.
Examples

In the arithmetic circuit model, the following problems are in $P$:

- Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial
- Polynomial operations: addition, multiplication, elementary symmetric polynomials
- Multivariate polynomial factorization when the polynomial is fixed
- In the arithmetic circuits with zero-test model, the following problems are also in $P$: solving a system of linear equations, rank of a matrix, gcd of polynomials, primality testing.
Examples

In the arithmetic circuit model, the following problems are in $P$:

- Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial
- Polynomial operations: addition, multiplication, elementary symmetric polynomials
- Multivariate polynomial factorization when the polynomial is fixed
- In the arithmetic circuits with zero-test model, the following problems are also in $P$: solving a system of linear equations, rank of a matrix, gcd of polynomials, primality testing.
In the arithmetic circuit model, the following problems are in \( P \):

- Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial
- Polynomial operations: addition, multiplication, elementary symmetric polynomials
- Multivariate polynomial factorization when the polynomial is fixed
- In the arithmetic circuits with zero-test model, the following problems are also in \( P \): solving a system of linear equations, rank of a matrix, gcd of polynomials, primality testing.
Examples

In the arithmetic circuit model, the following problems are in $P$:

- Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial
- Polynomial operations: addition, multiplication, elementary symmetric polynomials
- Multivariate polynomial factorization when the polynomial is fixed
- In the arithmetic circuits with zero-test model, the following problems are also in $P$: solving a system of linear equations, rank of a matrix, gcd of polynomials, primality testing.
Examples

In the arithmetic circuit model, the following problems are in $\mathbf{P}$:

- Matrix operations: addition, multiplication, determinant, inverse, characteristic polynomial
- Polynomial operations: addition, multiplication, elementary symmetric polynomials
- Multivariate polynomial factorization when the polynomial is fixed
- In the arithmetic circuits with zero-test model, the following problems are also in $\mathbf{P}$: solving a system of linear equations, rank of a matrix, gcd of polynomials, primality testing.
A Poor Definition of NP

- Analogous definition of NP to the boolean settings fails.
- Consider arithmetic circuit model, where each computation results in a polynomial, over $R = \mathbb{C}$.
- Say polynomial family $P_n(x_1, \ldots, x_n)$ is in NP if there exists another polynomial family $Q_{n+m+1}(x_1, \ldots, x_n, y_1, \ldots, y_m, z)$ in $P$ such that:
  1. $m = n^{O(1)}$, and
  2. $P_n(\alpha_1, \ldots, \alpha_n) = \gamma$ iff there exists $\beta_1, \ldots, \beta_m$ with $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m, \gamma) = 0$. 
A Poor Definition of NP

- Analogous definition of NP to the boolean settings fails.
- Consider arithmetic circuit model, where each computation results in a polynomial, over $R = \mathbb{C}$.
- Say polynomial family $P_n(x_1, \ldots, x_n)$ is in NP if there exists another polynomial family $Q_{n+m+1}(x_1, \ldots, x_n, y_1, \ldots, y_m, z)$ in $P$ such that:
  - $m = n^{O(1)}$, and
  - $P_n(\alpha_1, \ldots, \alpha_n) = \gamma$ iff there exists $\beta_1, \ldots, \beta_m$ with $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m, \gamma) = 0$. 
A Poor Definition of NP

- Analogous definition of NP to the boolean settings fails.
- Consider arithmetic circuit model, where each computation results in a polynomial, over $R = \mathbb{C}$.
- Say polynomial family $P_n(x_1, \ldots, x_n)$ is in NP if there exists another polynomial family $Q_{n+m+1}(x_1, \ldots, x_n, y_1, \ldots, y_m, z)$ in P such that:
  - $m = n^{O(1)}$, and
  - $P_n(\alpha_1, \ldots, \alpha_n) = \gamma$ iff there exists $\beta_1, \ldots, \beta_m$ with $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m, \gamma) = 0$. 
A Poor Definition of NP

- By definition, $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, y_1, \ldots, y_m, z) = 0$ iff $z = \gamma$.
- Therefore,
  $$Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, y_1, \ldots, y_m, z) = \delta \cdot (z - \gamma)^t,$$
  $t > 0$.
- Since this is true for all $\alpha_1, \ldots, \alpha_n$, we can reset $Q_{n+m+1}$ to $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, 0, \ldots, 0, z)$. 
By definition, $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, y_1, \ldots, y_m, z) = 0$ iff $z = \gamma$.

Therefore,

$$Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, y_1, \ldots, y_m, z) = \delta \cdot (z - \gamma)^t,$$

$t > 0$.

Since this is true for all $\alpha_1, \ldots, \alpha_n$, we can reset $Q_{n+m+1}$ to $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, 0, \ldots, 0, z)$. 


A Poor Definition of NP

- By definition, $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, y_1, \ldots, y_m, z) = 0$ iff $z = \gamma$.
- Therefore,
  
  $$Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, y_1, \ldots, y_m, z) = \delta \cdot (z - \gamma)^t,$$
  
  $t > 0$.

- Since this is true for all $\alpha_1, \ldots, \alpha_n$, we can reset $Q_{n+m+1}$ to $Q_{n+m+1}(\alpha_1, \ldots, \alpha_n, 0, \ldots, 0, z)$. 
A Better Definition of NP

The Class NP [Valiant 1979]

Polynomial family \( \{P_n\} \) is in NP if there exists a family \( \{P_{n+m}\} \in P \) such that \( m = n^{O(1)} \), and for every \( n \):

\[
P_n(x_1, \ldots, x_n) = \sum_{y_1 \in \{0,1\}} \cdots \sum_{y_m \in \{0,1\}} Q_{n+m}(x_1, \ldots, x_n, y_1, \ldots, y_m).
\]

1. Here 0 and 1 are identities of \( R \).
2. The definition can be easily generalized to arithmetic circuit with zero-test model.
Examples

- All problems in $P$,
- Permanent family,
- Jones polynomials: representing invariants of knots,
- Tutte polynomials:

$$T_G(x, y) = \sum_{A \subseteq E} (x - 1)^{k(A) - k(E)} (y - 1)^{k(A) + |A| - |V|}$$

where $G = (V, E)$ is an undirected graph and $k(A)$ is the number of connected components in the subgraph $(V, A)$. 
Examples

- All problems in \( P \),
- Permanent family,
- Jones polynomials: representing invariants of knots,
- Tutte polynomials:

\[ T_G(x, y) = \sum_{A \subseteq E} (x - 1)^{k(A) - k(E)}(y - 1)^{k(A) + |A| - |V|} \]

where \( G = (V, E) \) is an undirected graph and \( k(A) \) is the number of connected components in the subgraph \((V, A)\).
Examples

- All problems in \( P \),
- Permanent family,
- Jones polynomials: representing invariants of knots,
- Tutte polynomials:

\[
T_G(x, y) = \sum_{A \subseteq E} (x - 1)^{k(A) - k(E)} (y - 1)^{k(A) + |A| - |V|}
\]

where \( G = (V, E) \) is an undirected graph and \( k(A) \) is the number of connected components in the subgraph \( (V, A) \).
**Examples**

- All problems in \( P \),
- Permanent family,
- Jones polynomials: representing invariants of knots,
- Tutte polynomials:

\[
T_G(x, y) = \sum_{A \subseteq E} (x - 1)^{k(A) - k(E)}(y - 1)^{k(A) + |A| - |V|}
\]

where \( G = (V, E) \) is an undirected graph and \( k(A) \) is the number of connected components in the subgraph \( (V, A) \).
NP-complete Problems

**Theorem [Valient 1979]**

Computing permanent family is complete for NP in arithmetic circuit model: for every polynomial family \( \{Q_n\} \) in NP, for every \( n \), \( Q_n \) can be expressed as permanent of a \( n^{O(1)} \)-size matrix with variable and constant entries.

- Several other polynomial families are also NP-complete: Jones polynomials, Tutte polynomials, matching polynomial etc.
NP-complete Problems

**Theorem [Valient 1979]**

Computing permanent family is complete for NP in arithmetic circuit model: for every polynomial family \( \{Q_n\} \) in NP, for every \( n \), \( Q_n \) can be expressed as permanent of a \( n^{O(1)} \)-size matrix with variable and constant entries.

- Several other polynomial families are also NP-complete: Jones polynomials, Tutte polynomials, matching polynomial etc.
Is $P \neq NP$?

- The classes $P$ and $NP$ of arithmetic circuit model roughly correspond to computing the boolean classes $\#L$ and $\#P$ respectively:
  - Permanent is complete for $\#P$ in boolean model and for $NP$ in arithmetic circuit model.
  - Determinant is complete for $\#L$ in boolean model and for $P$ under quasi-polynomial size reductions in arithmetic circuit model.
- Therefore, it is a weaker question that $P \neq NP$ in boolean model: If $P \neq NP$ in boolean model then $P \neq NP$ in arithmetic circuit model.
Is $P \neq NP$?

- The classes $P$ and $NP$ of arithmetic circuit model roughly correspond to computing the boolean classes $#L$ and $#P$ respectively:
  - Permanent is complete for $#P$ in boolean model and for $NP$ in arithmetic circuit model.
  - Determinant is complete for $#L$ in boolean model and for $P$ under quasi-polynomial size reductions in arithmetic circuit model.

Therefore, it is a weaker question that $P \neq NP$ in boolean model: If $P \neq NP$ in boolean model then $P \neq NP$ in arithmetic circuit model.
Is \( P \neq NP \)?

- The classes \( P \) and \( NP \) of arithmetic circuit model roughly correspond to computing the boolean classes \( \#L \) and \( \#P \) respectively:
  - Permanent is complete for \( \#P \) in boolean model and for \( NP \) in arithmetic circuit model.
  - Determinant is complete for \( \#L \) in boolean model and for \( P \) under quasi-polynomial size reductions in arithmetic circuit model.

Therefore, it is a weaker question that \( P \neq NP \) in boolean model: If \( P \neq NP \) in boolean model then \( P \neq NP \) in arithmetic circuit model.
Is $P \neq NP$?

- The classes $P$ and $NP$ of arithmetic circuit model roughly correspond to computing the boolean classes $\#L$ and $\#P$ respectively:
  - Permanent is complete for $\#P$ in boolean model and for $NP$ in arithmetic circuit model.
  - Determinant is complete for $\#L$ in boolean model and for $P$ under quasi-polynomial size reductions in arithmetic circuit model.

- Therefore, it is a weaker question that $P \neq NP$ in boolean model: If $P \neq NP$ in boolean model then $P \neq NP$ in arithmetic circuit model.
Is $P \neq NP$?

- Even for arithmetic circuit model, proving $P \neq NP$ has been very challenging, and has remained a hypothesis.
- Henceforth, we restrict ourselves to the arithmetic model of computation.
- For arithmetic circuit model, the classes $P$ and $NP$ are called $VP$ and $VNP$: named after Valiant.
- Over the years, this problem has become one of the most active areas of research in complexity theory.
Is $P \neq NP$?

- Even for arithmetic circuit model, proving $P \neq NP$ has been very challenging, and has remained a hypothesis.
- Henceforth, we restrict ourselves to the arithmetic model of computation.
- For arithmetic circuit model, the classes $P$ and $NP$ are called $VP$ and $VNP$: named after Valiant.
- Over the years, this problem has become one of the most active areas of research in complexity theory.
Is $P \neq NP$?

- Even for arithmetic circuit model, proving $P \neq NP$ has been very challenging, and has remained a hypothesis.
- Henceforth, we restrict ourselves to the arithmetic model of computation.
- For arithmetic circuit model, the classes $P$ and $NP$ are called $VP$ and $VNP$: named after Valiant.
- Over the years, this problem has become one of the most active areas of research in complexity theory.
Is P $\neq$ NP?

- Even for arithmetic circuit model, proving $P \neq NP$ has been very challenging, and has remained a hypothesis.
- Henceforth, we restrict ourselves to the arithmetic model of computation.
- For arithmetic circuit model, the classes $P$ and $NP$ are called $VP$ and $VNP$: named after Valiant.
- Over the years, this problem has become one of the most active areas of research in complexity theory.
Outline

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
Reducing Depth to $O(\log d)$

**Theorem (Valiant-Skyum-Berkowitz-Rackoff, 1983)**

If polynomial $P(x_1, \ldots, x_n)$ of degree $d$ is computable by an arithmetic circuit of size $s \geq n$, then it can also be computed by an arithmetic circuit of size $s^{O(1)}$ whose depth is $O(\log d)$ and fanin of multiplication gates is two.

Another construction was given by [Allender-Jiao-Mahajan-Vinay 1994].
Reducing Depth to 4

**Theorem (A-Vinay 2008)**

If polynomial $P(x_1, \ldots, x_n)$ of degree $d$ is computable by an arithmetic circuit of size $s = 2^{o(d + d \log \frac{n}{d})}$, then it can also be computed by an arithmetic circuit of size $s^{O(1)}$ of depth 4.

Extended by [Koiran 2012, Tavenas 2013].
Proof

Let the polynomial $P(x_1, \ldots, x_n)$ be computed by an arithmetic circuit $C$ of size $t = 2^{o(d+d \log \frac{n}{d})}$.

- [Allender-Jiao-Mahajan-Vinay 1994] shows that $C$ can be transformed to a circuit $D$ of degree $d$, size $t^{O(1)}$ and depth $O(\log d)$ with multiplication gates of fanin two.
- We modify this transformation slightly to obtain a circuit $D$ of degree $d$, size $t^{O(1)}$ and depth $\leq 2 \log d$ with multiplication gates of fanin $\leq 6$.
- Further, the circuit $D$ consists of alternating layers of addition and multiplication gates.
- We now describe the construction of the circuit $D$. 


Proof

Let the polynomial $P(x_1, \ldots, x_n)$ be computed by an arithmetic circuit $C$ of size $t = 2^{o(d + d \log \frac{n}{d})}$.

- [Allender-Jiao-Mahajan-Vinay 1994] shows that $C$ can be transformed to a circuit $D$ of degree $d$, size $t^{O(1)}$ and depth $O(\log d)$ with multiplication gates of fanin two.

- We modify this transformation slightly to obtain a circuit $D$ of degree $d$, size $t^{O(1)}$ and depth $\leq 2 \log d$ with multiplication gates of fanin $\leq 6$.

- Further, the circuit $D$ consists of alternating layers of addition and multiplication gates.

- We now describe the construction of the circuit $D$. 
Proof

Let the polynomial $P(x_1, \ldots, x_n)$ be computed by an arithmetic circuit $C$ of size $t = 2^{o(d+d \log \frac{n}{d})}$.

- [Allender-Jiao-Mahajan-Vinay 1994] shows that $C$ can be transformed to a circuit $D$ of degree $d$, size $t^{O(1)}$ and depth $O(\log d)$ with multiplication gates of fanin two.

- We modify this transformation slightly to obtain a circuit $D$ of degree $d$, size $t^{O(1)}$ and depth $\leq 2 \log d$ with multiplication gates of fanin $\leq 6$.

- Further, the circuit $D$ consists of alternating layers of addition and multiplication gates.

- We now describe the construction of the circuit $D$. 
Proof

Let the polynomial \( P(x_1, \ldots, x_n) \) be computed by an arithmetic circuit \( C \) of size \( t = 2^{o(d + d \log \frac{n}{d})} \).

- [Allender-Jiao-Mahajan-Vinay 1994] shows that \( C \) can be transformed to a circuit \( D \) of degree \( d \), size \( t^{O(1)} \) and depth \( O(\log d) \) with multiplication gates of fanin two.
- We modify this transformation slightly to obtain a circuit \( D \) of degree \( d \), size \( t^{O(1)} \) and depth \( \leq 2 \log d \) with multiplication gates of fanin \( \leq 6 \).
- Further, the circuit \( D \) consists of alternating layers of addition and multiplication gates.
- We now describe the construction of the circuit \( D \).
Construction of $D$: Setup

- Make the circuit $C$ layered with alternating layers of addition and multiplication gates.
- Make fanin of every multiplication gate two.
- Rearrange children of multiplication gates so that degree of the right child is greater than or equal to the degree of the left child.
Construction of $D$: Setup

- Make the circuit $C$ layered with alternating layers of addition and multiplication gates.
- Make fanin of every multiplication gate two.
- Rearrange children of multiplication gates so that degree of the right child is greater than or equal to the degree of the left child.
Construction of $D$: Setup

- Make the circuit $C$ layered with alternating layers of addition and multiplication gates.
- Make fanin of every multiplication gate two.
- Rearrange children of multiplication gates so that degree of the right child is greater than or equal to the degree of the left child.
Construction of $D$: Proof Trees

A proof tree rooted at gate $g$ of circuit $C$ is a subcircuit of $C$ obtained as follows:

- Start with the subcircuit of $C$ that has gate $g$ at the top and computes the polynomial at gate $g$.
- For every $+$-gate in the subcircuit, retain only one input to the gate deleting the remaining input lines.
- For every $*$-gate in the subcircuit, retain both the inputs to the gate.

A proof tree rooted at gate $g$ computes a monomial and the polynomial at $g$ is the sum over monomials computed by all proof trees rooted at $g$. 

Manindra Agrawal (IIT Kanpur)  
Algebraic Complexity  
SLAC 2015 30 / 73
Construction of $D$: Proof Trees

A proof tree rooted at gate $g$ of circuit $C$ is a subcircuit of $C$ obtained as follows:

- Start with the subcircuit of $C$ that has gate $g$ at the top and computes the polynomial at gate $g$.
- For every $+$-gate in the subcircuit, retain only one input to the gate deleting the remaining input lines.
- For every $\ast$-gate in the subcircuit, retain both the inputs to the gate.

A proof tree rooted at gate $g$ computes a monomial and the polynomial at $g$ is the sum over monomials computed by all proof trees rooted at $g$. 
**Construction of \( D \): Proof Trees**

A proof tree rooted at gate \( g \) of circuit \( C \) is a subcircuit of \( C \) obtained as follows:

- Start with the subcircuit of \( C \) that has gate \( g \) at the top and computes the polynomial at gate \( g \).
- For every \( + \)-gate in the subcircuit, retain only one input to the gate deleting the remaining input lines.
- For every \( \ast \)-gate in the subcircuit, retain both the inputs to the gate.

A proof tree rooted at gate \( g \) computes a monomial and the polynomial at \( g \) is the sum over monomials computed by all proof trees rooted at \( g \).
**Construction of $D$: Defining Intermediate Polynomials**

- For every input variable $x_i$, let $[x_i]$ stand for the polynomial $x_i$.
- For every gate $g$ of $C$, let $[g]$ stand for polynomial computed at gate $g$.
- For every pair of gates $g$ and $h$ of $C$, let $[g, h]$ be the polynomial:

$$[g, h] = \sum_T m(T, h)$$

where $T$ runs over all proof trees rooted at $g$ and $m(T, h)$ is the monomial computed by proof tree $T$ when gate $h$ is replaced by 1 if gate $h$ occurs in the rightmost path of $T$, $m(T, h)$ is 0 otherwise.

- It follows that

$$[g] = \sum_{i=1}^{n} [g, x_i][x_i].$$
Construction of $D$: Defining Intermediate Polynomials

- For every input variable $x_i$, let $[x_i]$ stand for the polynomial $x_i$.
- For every gate $g$ of $C$, let $[g]$ stand for polynomial computed at gate $g$.
- For every pair of gates $g$ and $h$ of $C$, let $[g, h]$ be the polynomial:

$$[g, h] = \sum_T m(T, h)$$

where $T$ runs over all proof trees rooted at $g$ and $m(T, h)$ is the monomial computed by proof tree $T$ when gate $h$ is replaced by 1 if gate $h$ occurs in the rightmost path of $T$, $m(T, h)$ is 0 otherwise.

- It follows that

$$[g] = \sum_{i=1}^{n} [g, x_i][x_i].$$
CONSTRUCTION OF $D$: DEFINING INTERMEDIATE POLYNOMIALS

- For every input variable $x_i$, let $[x_i]$ stand for the polynomial $x_i$.
- For every gate $g$ of $C$, let $[g]$ stand for polynomial computed at gate $g$.
- For every pair of gates $g$ and $h$ of $C$, let $[g, h]$ be the polynomial:

$$[g, h] = \sum_{T} m(T, h)$$

where $T$ runs over all proof trees rooted at $g$ and $m(T, h)$ is the monomial computed by proof tree $T$ when gate $h$ is replaced by 1 if gate $h$ occurs in the rightmost path of $T$, $m(T, h)$ is 0 otherwise.

It follows that

$$[g] = \sum_{i=1}^{n} [g, x_i][x_i].$$
**Construction of $D$: Defining Intermediate Polynomials**

- For every input variable $x_i$, let $[x_i]$ stand for the polynomial $x_i$.
- For every gate $g$ of $C$, let $[g]$ stand for polynomial computed at gate $g$.
- For every pair of gates $g$ and $h$ of $C$, let $[g, h]$ be the polynomial:

$$[g, h] = \sum_T m(T, h)$$

where $T$ runs over all proof trees rooted at $g$ and $m(T, h)$ is the monomial computed by proof tree $T$ when gate $h$ is replaced by 1 if gate $h$ occurs in the rightmost path of $T$, $m(T, h)$ is 0 otherwise.

- It follows that

$$[g] = \sum_{i=1}^{n} [g, x_i][x_i].$$
Construction of $D$: Defining gates $[g, h]$

- If $g$ is a $+$-gate with children $g_1, \ldots, g_t$, then
  
  $$[g, h] = \sum_{i=1}^{t} [g_i, h].$$

- Let $g$ be a $\ast$-gate with children $g_L$ (left child) and $g_R$ (right child).

- A rightmost path from $g$ to $h$ is a path from $g$ to $h$ in the circuit obtained from $C$ by deleting input line from left child of every $\ast$-gate.

- If there are only $+$-gates on every rightmost path from $g$ to $h$ then
  
  $$[g, h] = [g_L].$$
Construction of $D$: Defining gates $[g, h]$

- If $g$ is a $+$-gate with children $g_1, \ldots, g_t$, then
  
  $$ [g, h] = \sum_{i=1}^{t} [g_i, h]. $$

- Let $g$ be a $*$-gate with children $g_L$ (left child) and $g_R$ (right child).

- A rightmost path from $g$ to $h$ is a path from $g$ to $h$ in the circuit obtained from $C$ by deleting input line from left child of every $*$-gate.

- If there are only $+$-gates on every rightmost path from $g$ to $h$ then
  
  $$ [g, h] = [g_L]. $$
Construction of \( D \): Defining gates \([g, h]\)

- If \( g \) is a \(+\)-gate with children \( g_1, \ldots, g_t \), then
  \[
  [g, h] = \sum_{i=1}^{t} [g_i, h].
  \]

- Let \( g \) be a \(*\)-gate with children \( g_L \) (left child) and \( g_R \) (right child).
- A rightmost path from \( g \) to \( h \) is a path from \( g \) to \( h \) in the circuit obtained from \( C \) by deleting input line from left child of every \(*\)-gate.
- If there are only \(+\)-gates on every rightmost path from \( g \) to \( h \) then
  \[
  [g, h] = [g_L].
  \]
Construction of $D$: Defining $[g, h]$

- Otherwise, there exists a $*$-gate $p$ with children $p_L$ and $p_R$ in a rightmost path from $g$ to $h$ such that
  \[ \deg(p) \geq \frac{1}{2}(\deg(g) + \deg(h)) > \deg(p_R). \]
- Then, we have:
  \[
  [g, h] = \sum_p [g, p] \cdot [p_L] \cdot [p_R, h]
  \]
  where the sum ranges over all gates $p$ satisfying the above condition.

$\deg(g)$ stands for degree of gate $g$
Construction of $D$: Defining $[g, h]$

- Otherwise, there exists a $*$-gate $p$ with children $p_L$ and $p_R$ in a rightmost path from $g$ to $h$ such that $\deg(p) \geq \frac{1}{2}(\deg(g) + \deg(h)) > \deg(p_R)$.

- Then, we have:

$$[g, h] = \sum_{p} [g, p] \cdot [p_L] \cdot [p_R, h]$$

where the sum ranges over all gates $p$ satisfying the above condition.

$\deg(g)$ stands for degree of gate $g$
**Construction of $D$: Defining $[g, h]$**

\[ [g, h] = \sum_p [g, p][p_L][p_R, h]. \]

\[ \deg([g, h]) = \deg(g) - \deg(h) \]

\[ \deg([p_L]) \leq \deg(g) - \deg(h) \]

\[ [p_L] = \sum_i [p_L, x_i][x_i], \]
\[ p_L = \sum_j p_L^j. \]
\[ [p_L^j, x_i] = \sum_q [p_L^j, q][q_L][q_R, x_i]. \]

\[ \deg([p_L^j, q] \leq \frac{1}{2} \deg(p_L) \]

\[ \deg([q_L] \leq \frac{1}{2} \deg(p_L) \]

\[ \deg([q_R, x_i] \leq \frac{1}{2} \deg(p_L) \]
**Construction of $D$: Defining $[g, h]$**

$[g, h] = \sum_p [g, p][p_L][p_R, h].$

$\text{deg}([g, h]) = \text{deg}(g) - \text{deg}(h)$

$\text{deg}([p_L]) \leq \text{deg}(g) - \text{deg}(h)$

$[p_L] = \sum_i [p_L, x_i][x_i],$

$p_L = \sum_j p_L^j.$

$[p_L^j, x_i] = \sum_q [p_L^j, q][q_L][q_R, x_i].$

$\text{deg}([p_L^j, q] \leq \frac{1}{2} \text{deg}(p_L)$

$\text{deg}([q_L] \leq \frac{1}{2} \text{deg}(p_L)$

$\text{deg}([q_R, x_i] \leq \frac{1}{2} \text{deg}(p_L)$
**Construction of \( D \): Defining \( [g, h] \)**

\[
[g, h] = \sum_p [g, p][p_L][p_R, h].
\]

\[
\text{deg}([g, h]) = \text{deg}(g) - \text{deg}(h)
\]

\[
\text{deg}([g, p]) \leq \frac{1}{2}(\text{deg}(g) - \text{deg}(h))
\]

\[
\text{deg}([p_L]) \leq \text{deg}(g) - \text{deg}(h)
\]

\[
[p_L] = \sum_i [p_L, x_i][x_i],
\]

\[
p_L = \sum_j p'_L.
\]

\[
[p'_L, x_i] = \sum_q [p'_L, q][q_L][q_R, x_i].
\]

\[
\text{deg}([p'_L, q] \leq \frac{1}{2} \text{deg}(p_L)
\]
**Construction of \( D \): Defining \([g, h]\)**

\[
[g, h] = \sum_p [g, p][p_L][p_R, h].
\]

\[
\text{deg}([g, h]) = \text{deg}(g) - \text{deg}(h)
\]

\[
\text{deg}([p_R, h]) \leq \frac{1}{2}(\text{deg}(g) - \text{deg}(h))
\]

\[
\text{deg}([p_L]) \leq \text{deg}(g) - \text{deg}(h)
\]

\[
[p_L] = \sum_i [p_L, x_i][x_i],
\]

\[
p_L = \sum_j p_L^j.
\]

\[
[p_L^j, x_i] = \sum_q [p_L^j, q][q_L][q_R, x_i].
\]

\[
\text{deg}([p_L^j, q] \leq \frac{1}{2} \text{deg}(p_L)
\]
Construction of $D$: Defining $[g, h]$

$[g, h] = \sum p [g, p][p_L][p_R, h]$.  
$\deg([g, h]) = \deg(g) - \deg(h)$

$\deg([p_L]) \leq \deg(g) - \deg(h)$

$[p_L] = \sum_i [p_L, x_i][x_i]$,  
$p_L = \sum_j p'_L[j]$.

$[p'_L, x_i] = \sum_q [p'_L, q][q_L][q_R, x_i]$.  
$\deg([p'_L, q] \leq \frac{1}{2} \deg(p_L)$

$\deg([q_L] \leq \frac{1}{2} \deg(p_L)$

$\deg([q_R, x_i] \leq \frac{1}{2} \deg(p_L)$
**Construction of** $D$: **Defining** $[g, h]$

$$[g, h] = \sum_p [g, p][p_L][p_R, h].$$

$$\deg([g, h]) = \deg(g) - \deg(h)$$

$$\deg([p_L]) \leq \deg(g) - \deg(h)$$

$$[p_L] = \sum_i [p_L, x_i][x_i],$$

$$p_L = \sum_j p_L^j.$$  

$$[p_L^j, x_i] = \sum_q [p_L^j, q][q_L][q_R, x_i].$$

$$\deg([p_L^j, q] \leq \frac{1}{2} \deg(p_L)$$

$$\deg([q_L] \leq \frac{1}{2} \deg(p_L)$$

$$\deg([q_R, x_i] \leq \frac{1}{2} \deg(p_L)$$
**Construction of $D$: Defining $[g, h]$**

$$[g, h] = \sum_p [g, p] [p_L] [p_R, h].$$

\[\text{deg}([g, h]) = \text{deg}(g) - \text{deg}(h)\]

\[\text{deg}([p_L]) \leq \text{deg}(g) - \text{deg}(h)\]

\[[p_L] = \sum_i [p_L, x_i] [x_i],\]

\[p_L = \sum_j p'_L.\]

\[[p'_L, x_i] = \sum_q [p'_L, q] [q_L] [q_R, x_i].\]

\[\text{deg}([p'_L, q] \leq \frac{1}{2} \text{deg}(p_L)\]

\[\text{deg}([q_L] \leq \frac{1}{2} \text{deg}(p_L)\]

\[\text{deg}([q_R, x_i] \leq \frac{1}{2} \text{deg}(p_L)\]
CONSTRUCTION OF $D$: DEFINING $[g, h]$

\[ [g, h] = \sum_p [g, p][p_L][p_R, h]. \]

\[ \deg([g, h]) = \deg(g) - \deg(h) \]

\[ \deg([p_L]) \leq \deg(g) - \deg(h) \]

\[ [p_L] = \sum_i [p_L, x_i][x_i], \quad p_L = \sum_j p_L^j. \]

\[ [p_L^j, x_i] = \sum_q [p_L^j, q][q_L][q_R, x_i]. \]

\[ \deg([p_L^j, q] \leq \frac{1}{2} \deg(p_L) \]

\[ \deg([q_L] \leq \frac{1}{2} \deg(p_L) \]

\[ \deg([q_R, x_i] \leq \frac{1}{2} \deg(p_L) \]
**Construction of $D$: Defining $[g, h]$**

\[ [g, h] = \sum_p [g, p][p_L][p_R, h]. \]

\[ \text{deg}([g, h]) = \text{deg}(g) - \text{deg}(h) \]

\[ \text{deg}([p_L]) \leq \text{deg}(g) - \text{deg}(h) \]

\[ [p_L] = \sum_i [p_L, x_i][x_i], \]

\[ p_L = \sum_j p_L^j. \]

\[ [p_L^j, x_i] = \sum_q [p_L^j, q][q_L][q_R, x_i]. \]

\[ \text{deg}([p_L^j, q] \leq \frac{1}{2} \text{deg}(p_L) \]

\[ \text{deg}([q_L] \leq \frac{1}{2} \text{deg}(p_L) \]

\[ \text{deg}([q_R, x_i] \leq \frac{1}{2} \text{deg}(p_L) \]
**Construction of \( D \): Defining \( [g, h] \)**

\[
[g, h] = \sum_p [g, p][p_L][p_R, h].
\]

\[
\deg([g, h]) = \deg(g) - \deg(h)
\]

\[
\deg([p_L]) \leq \deg(g) - \deg(h)
\]

\[
[p_L] = \sum_i [p_L, x_i][x_i],
\]

\[
p_L = \sum_j p'_L.
\]

\[
[p'_L, x_i] = \sum_q [p'_L, q][q_L][q_R, x_i].
\]

\[
\deg([p'_L, q] \leq \frac{1}{2} \deg(p_L)
\]

\[
\deg([q_L] \leq \frac{1}{2} \deg(p_L)
\]

\[
\deg([q_R, x_i] \leq \frac{1}{2} \deg(p_L)
\]
Construction of $D$: Defining $[g, h]$

$$[g, h] = \sum_p [g, p][p_L][p_R, h].$$

$$\deg([g, h]) = \deg(g) - \deg(h)$$

$$\deg([p_L]) \leq \deg(g) - \deg(h)$$

$$[p_L] = \sum_i [p_L, x_i][x_i],$$

$$p_L = \sum_j p_L^j.$$  

$$[p_L^j, x_i] = \sum_q [p_L^j, q][q_L][q_R, x_i].$$  

$$\deg([p_L^j, q] \leq \frac{1}{2} \deg(p_L)$$

$$\deg([q_L] \leq \frac{1}{2} \deg(p_L)$$

$$\deg([q_R, x_i] \leq \frac{1}{2} \deg(p_L)$$
**Construction of \( D \): Defining \([g, h]\)**

Flatten the subcircuit to write \([g, h]\) as:

\[
[g, h] = \sum_p \sum_i \sum_j \sum_q \left(g, p\right)\left[p_L, j, q\right]\left[q_L\right]\left[q_R, x_i\right]\left[x_i\right]\left[p_R, h\right]
\]

with degree of each of the six polynomials in the product bounded by \(\frac{1}{2} \deg([g, h])\).
Construction of $D$

- By adding dummy $+$-gates and merging adjacent $+$-gates, it can be ensured that the circuit has alternating layers of $+$- and $\ast$-gates.
- The size of resulting circuit is $t^{O(1)}$.
- Since the degree of children of a $\ast$-gate is at most half of the degree of the gate, the depth of the circuit $D$ is $\leq 2 \log d$. 

Manindra Agrawal (IIT Kanpur)
Algebraic Complexity
SLAC 2015  36 / 73
Construction of $D$

- By adding dummy $+$-gates and merging adjacent $+$-gates, it can be ensured that the circuit has alternating layers of $+$- and $\ast$-gates.
- The size of resulting circuit is $t^{O(1)}$.
- Since the degree of children of a $\ast$-gate is at most half of the degree of the gate, the depth of the circuit $D$ is $\leq 2 \log d$. 
Replacing $D$

- We now replace $D$ by a depth four circuit.
- The circuit is defined by cutting $D$ in two halves and replacing each half by a depth two circuit.
Replacing $D$

- We now replace $D$ by a depth four circuit.
- The circuit is defined by cutting $D$ in two halves and replacing each half by a depth two circuit.
Cutting $D$

- Let $\ell$ be any function such that $\ell \leq \frac{d+d \log d}{\log t}$ and $\ell = \omega(1)$.

- Let $u = \frac{1}{2} \log_6 \ell$.

- Cut $D$ into two halves with top half consisting of $u$ layers of $\ast$-gates with the bottom layer being of $\ast$-gates.

- Let $g_1, g_2, \ldots, g_k$ be the output gates of the bottom layer.

- Let the polynomial computed by gate $g_i$ be $P_i(x_1, x_2, \ldots, x_n)$.

- The top layer can be viewed as computing a polynomial in $k$ new variables; let this be $P_0(y_1, y_2, \ldots, y_k)$.

- Then:

$$P(x_1, \ldots, x_n) = P_0(P_1(x_1, \ldots, x_n), P_2(x_1, \ldots, x_n), \ldots, P_k(x_1, \ldots, x_n)).$$
Cutting $D$

- Let $\ell$ be any function such that $\ell \leq \frac{d+d \log d}{\log n}$ and $\ell = \omega(1)$.
- Let $u = \frac{1}{2} \log_6 \ell$.
- Cut $D$ into two halves with top half consisting of $u$ layers of $\ast$-gates with the bottom layer being of $\ast$-gates.
  - Let $g_1, g_2, \ldots, g_k$ be the output gates of the bottom layer.
  - Let the polynomial computed by gate $g_i$ be $P_i(x_1, x_2, \ldots, x_n)$.
  - The top layer can be viewed as computing a polynomial in $k$ new variables; let this be $P_0(y_1, y_2, \ldots, y_k)$.
  - Then:

$$P(x_1, \ldots, x_n) = P_0(P_1(x_1, \ldots, x_n), P_2(x_1, \ldots, x_n), \ldots, P_k(x_1, \ldots, x_n)).$$
Cutting $D$

- Let $\ell$ be any function such that $\ell \leq \frac{d + d \log d}{\log t}$ and $\ell = \omega(1)$.
- Let $u = \frac{1}{2} \log_6 \ell$.
- Cut $D$ into two halves with top half consisting of $u$ layers of $\ast$-gates with the bottom layer being of $\ast$-gates.
- Let $g_1, g_2, \ldots, g_k$ be the output gates of the bottom layer.
- Let the polynomial computed by gate $g_i$ be $P_i(x_1, x_2, \ldots, x_n)$.
- The top layer can be viewed as computing a polynomial in $k$ new variables; let this be $P_0(y_1, y_2, \ldots, y_k)$.
- Then:

$$P(x_1, \ldots, x_n) = P_0(P_1(x_1, \ldots, x_n), P_2(x_1, \ldots, x_n), \ldots, P_k(x_1, \ldots, x_n)).$$
Cutting $D$

- Let $\ell$ be any function such that $\ell \leq \frac{d + d \log d}{\log t}$ and $\ell = \omega(1)$.
- Let $u = \frac{1}{2} \log_6 \ell$.
- Cut $D$ into two halves with top half consisting of $u$ layers of $\ast$-gates with the bottom layer being of $\ast$-gates.
- Let $g_1, g_2, \ldots, g_k$ be the output gates of the bottom layer.
- Let the polynomial computed by gate $g_i$ be $P_i(x_1, x_2, \ldots, x_n)$.
- The top layer can be viewed as computing a polynomial in $k$ new variables; let this be $P_0(y_1, y_2, \ldots, y_k)$.
- Then:

$$P(x_1, \ldots, x_n) = P_0(P_1(x_1, \ldots, x_n), P_2(x_1, \ldots, x_n), \ldots, P_k(x_1, \ldots, x_n)).$$
A direct counting shows that each $P_j$, $0 \leq j \leq k$, can be replaced by a depth two circuit of size $2^{o(d+d \log \frac{n}{d})}$.

Since $k = 2^{o(d+d \log \frac{n}{d})}$, the resulting depth four circuit, $E$, is of size $2^{o(d+d \log \frac{n}{d})}$.

The fanin of second layer of $\ast$-gates in $E$ is at most $6^{u} = \sqrt{\ell}$ which is any small function in $\omega(1)$.

The fanin of bottom layer of $\ast$-gates in $E$ is at most $\frac{d}{2^{u}} = o(d)$. 

The Circuit $E$
The Circuit $E$

- A direct counting shows that each $P_j$, $0 \leq j \leq k$, can be replaced by a depth two circuit of size $2^{o(d+d \log \frac{n}{d})}$.

- Since $k = 2^{o(d+d \log \frac{n}{d})}$, the resulting depth four circuit, $E$, is of size $2^{o(d+d \log \frac{n}{d})}$.

- The fanin of second layer of $*$-gates in $E$ is at most $6^u = \sqrt{\ell}$ which is any small function in $\omega(1)$.

- The fanin of bottom layer of $*$-gates in $E$ is at most $\frac{d}{2^u} = o(d)$. 
The Circuit $E$

- A direct counting shows that each $P_j$, $0 \leq j \leq k$, can be replaced by a depth two circuit of size $2^{o(d+d\log \frac{n}{d})}$.
- Since $k = 2^{o(d+d\log \frac{n}{d})}$, the resulting depth four circuit, $E$, is of size $2^{o(d+d\log \frac{n}{d})}$.
- The fanin of second layer of $\ast$-gates in $E$ is at most $6^{u} = \sqrt{\ell}$ which is any small function in $\omega(1)$.
- The fanin of bottom layer of $\ast$-gates in $E$ is at most $\frac{d}{2^{u}} = o(d)$. 
Reducing Depth to 3

**Theorem (Gupta-Kamath-Kayal-Saptharishi 2013)**

If polynomial $P(x_1, \ldots, x_n)$ of degree $d$ is computable by an arithmetic circuit of size $s = 2^{o(d + d \log \frac{n}{d})}$, then it can also be computed by an arithmetic circuit of size $s^{O(1)}$ of depth 3 if the underlying field has characteristic zero or large ($\Omega(\log s)$).
Proof Outline

- Replace each $\prod$ layer of a depth four circuit by $\sum \land \sum$ layers resulting in a $\sum \land \sum \land \sum$ circuit using [Fischer 1994]:

$$\prod_{j=1}^{n} x_j = \frac{1}{2^{n-1}n!} \sum_{r_2,\ldots,r_n \in \{-1,1\}} (-1)^{wt(r)} (x_1 + \sum_{j=2}^{n} r_j x_j)^n,$$

where $wt(r) = |\{j \mid r_j = -1\}|$. This works for $\text{char} = 0$ or $> n$.

- Replace $\land \sum \land$ by $\sum \prod \sum$ resulting in $\sum \prod \sum$ circuit using [Saxena 2008]:

$$(\alpha_1 x_1^{\beta_1} + \alpha_2 x_2^{\beta_2} + \cdots + \alpha_n x_n^{\beta_n})^d = \text{degree } d \text{ coefficient of } d! \cdot \prod_{j=1}^{n} e^{\alpha_j x_j^{\beta_j} z}.$$

This works for $\text{char} = 0$ or $> d$. 
Proof Outline

- Replace each $\prod$ layer of a depth four circuit by $\sum \land \sum$ layers resulting in a $\sum \land \sum \land \sum$ circuit using [Fischer 1994]:

$$\prod_{j=1}^{n} x_j = \frac{1}{2^{n-1}n!} \sum_{r_2,...,r_n \in \{-1,1\}} (-1)^{wt(r)} (x_1 + \sum_{j=2}^{n} r_j x_j)^n,$$

where $wt(r) = |\{j \mid r_j = -1\}|$. This works for $\text{char} = 0$ or $> n$.

- Replace $\land \sum \land$ by $\sum \prod \sum$ resulting in $\sum \prod \sum$ circuit using [Saxena 2008]:

$$(\alpha_1 x_1^{\beta_1} + \alpha_2 x_2^{\beta_2} + \cdots + \alpha_n x_n^{\beta_n})^d = \text{degree } d \text{ coefficient of } d! \cdot \prod_{j=1}^{n} e^{\alpha_j x_j^{\beta_j} z}.$$

This works for $\text{char} = 0$ or $> d$. 
Outline

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
Lower Bounds on Permanent and Determinant

[Jerrum-Snir 1982] Any monotone circuit family computing permanent is of exponential size.

- Monotone circuits are circuits with no negative constant.

[Shpilka-Wigderson 1999] Any depth three circuit family computing permanent (or even determinant) over $\mathbb{Q}$ is of size $\Omega(n^2)$.

[Grigoriev-Razborov 2000] Any depth three circuit family computing permanent or determinant over a finite field is of exponential size.
Lower Bounds on Permanent and Determinant

[Jerrum-Snir 1982] Any monotone circuit family computing permanent is of exponential size.

- Monotone circuits are circuits with no negative constant.

[Shpilka-Wigderson 1999] Any depth three circuit family computing permanent (or even determinant) over $\mathbb{Q}$ is of size $\Omega(n^2)$.

[Grigoriev-Razborov 2000] Any depth three circuit family computing permanent or determinant over a finite field is of exponential size.
Lower Bounds on Permanent and Determinant

[Jerrum-Snir 1982] Any monotone circuit family computing permanent is of exponential size.

- Monotone circuits are circuits with no negative constant.

[Shpilka-Wigderson 1999] Any depth three circuit family computing permanent (or even determinant) over $\mathbb{Q}$ is of size $\Omega(n^2)$.

[Grigoriev-Razborov 2000] Any depth three circuit family computing permanent or determinant over a finite field is of exponential size.
**Lower Bounds on Permanent and Determinant**

**[Raz 2004]** Any multilinear formula family computing permanent or determinant is of size $n^{\Omega(\log n)}$.

- Formulas are circuits with outdegree one.
- Multilinear formulas are formulas in which every gate computes a multilinear polynomial.

**[Kayal-Saha 2014]** Any depth three circuit family of bottom fanin $\leq r$ computing a polynomial family in VP of degree $d$ in $n$ variables over fields of characteristic zero, is of size $n^{\Omega(d/r)}$.

**[Kayal-Limaye-Saha-Srinivasan 2014]** $2^{\Omega(\sqrt{n} \log n)}$ lower bound on homogeneous depth four circuits computing permanent over characteristic zero.

A circuit is **homogeneous** if every intermediate polynomial is homogeneous.
Lower Bounds on Permanent and Determinant

[Raz 2004] Any multilinear formula family computing permanent or determinant is of size $n^{\Omega(\log n)}$.

- Formulas are circuits with outdegree one.
- Multilinear formulas are formulas in which every gate computes a multilinear polynomial.

[Kayal-Saha 2014] Any depth three circuit family of bottom fanin $\leq r$ computing a polynomial family in VP of degree $d$ in $n$ variables over fields of characteristic zero, is of size $n^{\Omega(\frac{d}{r})}$.

[Kayal-Limaye-Saha-Srinivasan 2014] $2^{\Omega(\sqrt{n} \log n)}$ lower bound on homogeneous depth four circuits computing permanent over characteristic zero.

A circuit is homogeneous if every intermediate polynomial is homogeneous.
Lower Bounds on Permanent and Determinant

[Raz 2004] Any multilinear formula family computing permanent or determinant is of size $n^{\Omega(\log n)}$.

- Formulas are circuits with outdegree one.
- Multilinear formulas are formulas in which every gate computes a multilinear polynomial.

[Kayal-Saha 2014] Any depth three circuit family of bottom fanin $\leq r$ computing a polynomial family in VP of degree $d$ in $n$ variables over fields of characteristic zero, is of size $n^{\Omega(\frac{d}{r})}$.

[Kayal-Limaye-Saha-Srinivasan 2014] $2^{\Omega(\sqrt{n}\log n)}$ lower bound on homogeneous depth four circuits computing permanent over characteristic zero.

A circuit is homogeneous if every intermediate polynomial is homogeneous.
Outline

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
Definitions

**PIT**

Given an arithmetic circuit of size $s$ over ring $R$, test if the polynomial computed by the circuit is non-zero.

**Low Degree PIT (LPIT)**

Given an arithmetic circuit of size $s$ over ring $R$ computing a polynomial of degree $\leq s$, test if the polynomial computed by the circuit is non-zero.
**Definitions**

**PIT**

Given an arithmetic circuit of size $s$ over ring $R$, test if the polynomial computed by the circuit is non-zero.

**Low Degree PIT (LPIT)**

Given an arithmetic circuit of size $s$ over ring $R$ computing a polynomial of degree $\leq s$, test if the polynomial computed by the circuit is non-zero.
An Example

Is \((ux + vy)^2 + (vx - uy)^2 - (u^2 + v^2) \cdot (x^2 + y^2) \neq 0?\) [NO!]
Bipartite Matching: for graph \( G = (U, V, E) \), check if

\[
\det \begin{bmatrix}
e_{1,1}x_{1,1} & \ldots & e_{1,n}x_{1,n} \\
\vdots & \ddots & \vdots \\
e_{n,1}x_{n,1} & \ldots & e_{n,n}x_{n,n}
\end{bmatrix} \neq 0
\]

over any field, where \( E = [e_{i,j}] \). An example of LPIT.

Primality Testing: for number \( n \), check if

\[
(x + y)^n = x^n + y^n
\]

over ring \( \mathbb{Z}_n[x, y]/(x^r - 1, y^s - 1) \) for suitable \( r \) and \( s \), both \( \log^{O(1)} n \).
Applications

Bipartite Matching: for graph $G = (U, V, E)$, check if

$$\det \begin{bmatrix} e_{1,1}x_{1,1} & \cdots & e_{1,n}x_{1,n} \\ \vdots & \ddots & \vdots \\ e_{n,1}x_{n,1} & \cdots & e_{n,n}x_{n,n} \end{bmatrix} \neq 0$$

over any field, where $E = [e_{i,j}]$. An example of LPIT.

Primality Testing: for number $n$, check if

$$(x + y)^n = x^n + y^n$$

over ring $\mathbb{Z}_n[x, y]/(x^r - 1, y^s - 1)$ for suitable $r$ and $s$, both $\log^{O(1)} n$. 
Complexity of PIT

A number of randomized polynomial time algorithms are known for the problem.

- The simplest one is by [Schwartz, Zippel 1979]: Substitute random values from a small subset of $\mathbb{R}$ (using a small extension of $\mathbb{R}$ if required) for each variable, evaluate the circuit, and output NON-ZERO iff the result is a non-zero number.
- Others are [Chen-Kao 1997], [Lewis-Vadhan 1998], [A-Biswas 1999], ...
A number of randomized polynomial time algorithms are known for the problem.

- The simplest one is by [Schwartz, Zippel 1979]: Substitute random values from a small subset of \( R \) (using a small extension of \( R \) if required) for each variable, evaluate the circuit, and output NON-ZERO iff the result is a non-zero number.
- Others are [Chen-Kao 1997], [Lewis-Vadhan 1998], [A-Biswas 1999], ....
A number of randomized polynomial time algorithms are known for the problem.

- The simplest one is by [Schwartz, Zippel 1979]: Substitute random values from a small subset of $R$ (using a small extension of $R$ if required) for each variable, evaluate the circuit, and output NON-ZERO iff the result is a non-zero number.

- Others are [Chen-Kao 1997], [Lewis-Vadhan 1998], [A-Biswas 1999], ...
Determination Algorithm for PIT

Open Question

Is there a deterministic polynomial time algorithm for PIT?

- Long-standing open problem.
- A positive answer also yields a lower bound.
**Deterministic Algorithm for PIT**

**Open Question**

Is there a deterministic polynomial time algorithm for PIT?

- Long-standing open problem.
- A positive answer also yields a lower bound.
Deterministic Algorithm for PIT

Open Question
Is there a deterministic polynomial time algorithm for PIT?

- Long-standing open problem.
- A positive answer also yields a lower bound.
Two Types of Deterministic Algorithms for PIT

**White Box**
A white-box time $t(n)$ algorithm for PIT is a deterministic algorithm solving the problem in time at most $t(n)$.

**Black Box**
A black-box time $t(n)$ algorithm for PIT is a deterministic algorithm running in time $t(n)$ that, given an arithmetic circuit, determines if it computes non-zero polynomial with access only to input-output lines and size of the circuit.
Two Types of Deterministic Algorithms for PIT

White Box

A white-box time $t(n)$ algorithm for PIT is a deterministic algorithm solving the problem in time at most $t(n)$.

Black Box

A black-box time $t(n)$ algorithm for PIT is a deterministic algorithm running in time $t(n)$ that, given an arithmetic circuit, determines if it computes non-zero polynomial with access only to input-output lines and size of the circuit.
Outline

1. Computation Over Rings
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. Classes P and NP

3. Depth Reduction

4. Status of Lower Bounds

5. Polynomial Identity Testing

6. LPIT and Lower Bounds

7. Algorithms for 2-PIT and 3-PIT
Theorem (Kabanets-Impagliazzo 2003)

If there exists a white-box polynomial-time algorithm for LPIT then $\text{NEXP}$ requires superpolynomial size arithmetic circuits.
LPIT and Lower Bounds I

Proof.

- Assume \textbf{NEXP} has polynomial-size arithmetic circuits and PIT has a polynomial-time algorithm.
- Construct an \textbf{NP} machine to compute permanent that guesses the circuit for the permanent and verifies it recursively using PIT:
  - If $C(x_1,1, \ldots, x_1,n, \ldots, x_n,1, \ldots, x_n,n)$ is circuit for permanent of $n \times n$ matrices, then we can extract from it circuit $C_j$ for permanent of $j \times j$ matrices for $j < n$.
  - Using LPIT, verify the correctness of $C$:
    \[
    C_j(\bar{x}) = x_1,1 C_{j-1}(\bar{x}_1) + \cdots + x_1,j C_{j-1}(\bar{x}_j)
    \]
    where $\bar{x}_i$ drops first row and $i$th column.
- This implies $\#P$ is in \textbf{NP}. Since \textbf{NEXP} $=$ $\#P$ by assumption, we get $\textbf{NEXP} = \textbf{NP}$ contradicting time hierarchy theorem.
Proof.

- Assume $\text{NEXP}$ has polynomial-size arithmetic circuits and $\text{PIT}$ has a polynomial-time algorithm.
- Construct an $\text{NP}$ machine to compute permanent that guesses the circuit for the permanent and verifies it recursively using $\text{PIT}$:
  - If $C(x_1,1,\ldots,x_1,n,\ldots,x_n,1,\ldots,x_n,n)$ is circuit for permanent of $n \times n$ matrices, then we can extract from it circuit $C_j$ for permanent of $j \times j$ matrices for $j < n$.
  - Using LPIT, verify the correctness of $C$:
    \[
    C_j(\bar{x}) = x_{1,1}C_{j-1}(\bar{x}_1) + \cdots + x_{1,j}C_{j-1}(\bar{x}_j)
    \]
    where $\bar{x}_i$ drops first row and $i$th column.
- This implies $\#P$ is in $\text{NP}$. Since $\text{NEXP} = \#P$ by assumption, we get $\text{NEXP} = \text{NP}$ contradicting time hierarchy theorem.
LPIT and Lower Bounds I

Proof.

- Assume \textsc{NEXP} has polynomial-size arithmetic circuits and \textsc{PIT} has a polynomial-time algorithm.
- Construct an \textsc{NP} machine to compute permanent that guesses the circuit for the permanent and verifies it recursively using \textsc{PIT}:
  - If \( C(x_1,1, \ldots, x_1,n, \ldots, x_n,1, \ldots, x_n,n) \) is circuit for permanent of \( n \times n \) matrices, then we can extract from it circuit \( C_j \) for permanent of \( j \times j \) matrices for \( j < n \).
  - Using LPIT, verify the correctness of \( C \):
    \[
    C_j(\bar{x}) = x_{1,1} C_{j-1}(\bar{x}_1) + \cdots + x_{1,j} C_{j-1}(\bar{x}_j)
    \]
    where \( \bar{x}_i \) drops first row and \( i \)th column.
- This implies \( \#P \) is in \textsc{NP}. Since \( \textsc{NEXP} = \#P \) by assumption, we get \( \textsc{NEXP} = \textsc{NP} \) contradicting time hierarchy theorem.
**LPIT and Lower Bounds I**

**Proof.**

- Assume **NEXP** has polynomial-size arithmetic circuits and **PIT** has a polynomial-time algorithm.

- Construct an **NP** machine to compute permanent that guesses the circuit for the permanent and verifies it recursively using **PIT**:
  
  ▶ If $C(x_1,1, \ldots, x_1,n, \ldots, x_n,1, \ldots, x_n,n)$ is circuit for permanent of $n \times n$ matrices, then we can extract from it circuit $C_j$ for permanent of $j \times j$ matrices for $j < n$.
  
  ▶ Using LPIT, verify the correctness of $C$:

  $$C_j(\bar{x}) = x_{1,1}C_{j-1}(\bar{x}_1) + \cdots + x_{1,j}C_{j-1}(\bar{x}_j)$$

  where $\bar{x}_i$ drops first row and $i$th column.

- This implies **#P** is in **NP**. Since **NEXP** = **#P** by assumption, we get **NEXP** = **NP** contradicting time hierarchy theorem.
LPIT and Lower Bounds I

Proof.

- Assume \textbf{NEXP} has polynomial-size arithmetic circuits and \textbf{PIT} has a polynomial-time algorithm.

- Construct an \textbf{NP} machine to compute permanent that guesses the circuit for the permanent and verifies it recursively using \textbf{PIT}:
  
  ▶ If \( C(x_1, 1, \ldots, x_1, n, \ldots, x_n, 1, \ldots, x_n, n) \) is circuit for permanent of \( n \times n \) matrices, then we can extract from it circuit \( C_j \) for permanent of \( j \times j \) matrices for \( j < n \).

  ▶ Using \textbf{LPIT}, verify the correctness of \( C \):

  \[
  C_j(\overline{x}) = x_{1,1} C_{j-1}(\overline{x}_1) + \cdots + x_{1,j} C_{j-1}(\overline{x}_j)
  \]

  where \( \overline{x}_i \) drops first row and \( i \)th column.

- This implies \( \#P \) is in \textbf{NP}. Since \( \text{NEXP} = \#P \) by assumption, we get \( \text{NEXP} = \text{NP} \) contradicting time hierarchy theorem.
**Theorem (Heintz-Schnorr 1980, A 2005)**

*If there exist a black-box polynomial-time algorithm for LPIT then $E$ requires exponential size arithmetic circuits.*
LPIT and Lower Bounds II

Proof.

- Let \( \mathcal{A} \) be a black-box polynomial-time algorithm for LPIT.
- For a circuit of size \( s \) on \( n \) variables, \( \mathcal{A} \) will evaluate it on a sequence of inputs and accept iff any of the outputs in non-zero.
- Let these inputs be \((\alpha_{1,1}, \ldots, \alpha_{1,n}), \ldots, (\alpha_{t,1}, \ldots, \alpha_{t,n})\) with \( t = s^{O(1)} \).
- Let \( m = \lceil \log(t + 1) \rceil = O(\log s) \).
- Define polynomial \( r_m \) as:

\[
r_m(x_1, x_2, \ldots, x_m) = \sum_{S \subseteq [1,m]} c_S \prod_{i \in S} x_i.
\]

Coefficients \( c_S \in F \) satisfy:

\[
\sum_{S \subseteq [1,m]} c_S \prod_{i \in S} \alpha_{j,i} = 0
\]

for every \( 1 \leq j \leq t \).
LPIT and Lower Bounds II

Proof.

- Let $A$ be a black-box polynomial-time algorithm for LPIT.
- For a circuit of size $s$ on $n$ variables, $A$ will evaluate it on a sequence of inputs and accept iff any of the outputs in non-zero.
- Let these inputs be $(\alpha_{1,1}, \ldots, \alpha_{1,n}), \ldots, (\alpha_{t,1}, \ldots, \alpha_{t,n})$ with $t = s^{O(1)}$.
- Let $m = \lceil \log(t + 1) \rceil = O(\log s)$.
- Define polynomial $r_m$ as:
  \[ r_m(x_1, x_2, \ldots, x_m) = \sum_{S \subseteq [1, m]} c_S \prod_{i \in S} x_i. \]
- Coefficients $c_S \in F$ satisfy:
  \[ \sum_{S \subseteq [1, m]} c_S \prod_{i \in S} \alpha_{j,i} = 0 \]
  for every $1 \leq j \leq t$. 
LPIT and Lower Bounds II

Proof.

- Let $A$ be a black-box polynomial-time algorithm for LPIT.
- For a circuit of size $s$ on $n$ variables, $A$ will evaluate it on a sequence of inputs and accept iff any of the outputs in non-zero.
- Let these inputs be $(\alpha_{1,1}, \ldots, \alpha_{1,n}), \ldots, (\alpha_{t,1}, \ldots, \alpha_{t,n})$ with $t = s^{O(1)}$.
- Let $m = \lceil \log(t + 1) \rceil = O(\log s)$.
- Define polynomial $r_m$ as:

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

- Coefficients $c_S \in F$ satisfy:

$$\sum_{S \subseteq [1,m]} c_S \prod_{i \in S} \alpha_{j,i} = 0$$

for every $1 \leq j \leq t$. 
A non-zero $r_m$ always exists since it has $\geq t + 1$ coefficients that satisfy $t$ homogeneous linear equations.

- Polynomial $r_m$ can be computed by exponential size arithmetic circuits.

- Circuit complexity of $r_m$ is more than $s = 2^{\Omega(m)}$. 
A non-zero $r_m$ always exists since it has $\geq t + 1$ coefficients that satisfy $t$ homogeneous linear equations.

Polynomial $r_m$ can be computed by exponential size arithmetic circuits.

Circuit complexity of $r_m$ is more than $s = 2^{\Omega(m)}$. 
• A non-zero $r_m$ always exists since it has $\geq t + 1$ coefficients that satisfy $t$ homogeneous linear equations.
• Polynomial $r_m$ can be computed by exponential size arithmetic circuits.
• Circuit complexity of $r_m$ is more than $s = 2^{\Omega(m)}$. 
**Fixed Depth PIT**

**Depth $d$ PIT**

$d$-PIT is the problem to decide if a given arithmetic circuit of depth $d$ (alternating sums and products with top gate being sum) computes a non-zero polynomial.

$d$-PIT is a restriction of LPIT.
Fixed Depth PIT

**Depth \( d \) PIT**

\( d \)-PIT is the problem to decide if a given arithmetic circuit of depth \( d \) (alternating sums and products with top gate being sum) computes a non-zero polynomial.

\( d \)-PIT is a restriction of LPIT.
3-PIT and Lower Bounds

**Theorem (Gupta-Kamath-Kayal-Saptharishi 2013)**

*If there exist a polynomial-time black-box algorithm for 3-PIT then $E$ requires exponential size arithmetic circuits if the underlying field has characteristic zero or large ($\Omega(\log s)$).*

**Theorem**

*If there exists a white-box polynomial-time algorithm for 3-PIT then $\text{NEXP}$ requires superpolynomial size arithmetic circuits.*
**Theorem (Gupta-Kamath-Kayal-Saptharishi 2013)**

*If there exist a polynomial-time black-box algorithm for 3-PIT then $E$ requires exponential size arithmetic circuits if the underlying field has characteristic zero or large ($= \Omega(\log s)$).*

**Theorem**

*If there exists a white-box polynomial-time algorithm for 3-PIT then $\text{NEXP}$ requires superpolynomial size arithmetic circuits.*
# Outline

1. **Computation Over Rings**
   - Arithmetic Circuit Model
   - Generalizing Arithmetic Circuits

2. **Classes P and NP**

3. **Depth Reduction**

4. **Status of Lower Bounds**

5. **Polynomial Identity Testing**

6. **LPIT and Lower Bounds**

7. **Algorithms for 2-PIT and 3-PIT**
Theorem (Folklore)

There exists a polynomial-time black-box algorithm for $2$-PIT.
Proof.

- A $\sum \prod$ circuit computes a sparse polynomial.
- Let $C$ be the given $\sum \prod$ circuit of size $s$ computing a polynomial of degree $\leq d$.
- One of the substitutions
  \[(x_1, \ldots, x_i, \ldots, x_n) = (y, \ldots, y^{(d+1)^i-1} \pmod r, \ldots, y^{(d+1)^{n-1} \pmod r}),
  1 < r < s^2,
\] will ensure that all terms of the polynomial remain distinct.
2-PIT

Proof.

- A $\Sigma \Pi$ circuit computes a sparse polynomial.
- Let $C$ be the given $\Sigma \Pi$ circuit of size $s$ computing a polynomial of degree $\leq d$.
- One of the substitutions
  
  $(x_1, \ldots, x_i, \ldots, x_n) = (y, \ldots, y^{(d+1)^{i-1}} \pmod{r}, \ldots, y^{(d+1)^{n-1}} \pmod{r})$,
  
  $1 < r < s^2$, will ensure that all terms of the polynomial remain distinct.
3-PIT with Bounded Top Fanin

Sequence of solutions for 3-PIT with top sum gate of fanin $k$:

[Dvir-Shpilka 2005] White-box $2^{(\log s)^k}$ time algorithm.


[Karnin-Shpilka 2008] Black-box $s^{O(\log^k s)}$ time algorithm.

[Saxena-Seshadri 2009] Black-box $s^{k^3 \log s}$ time algorithm.

[Kayal-Saraf 2009] Black-box $s^k$ time algorithm over characteristic zero fields.

[Saxena-Seshadri 2011] Black-box $s^{O(k)}$ time algorithm.

[A-Saha-Saptharishi-Saxena 2012] Black-box $s^{O(k)}$ time algorithm for zero or large characteristic fields.
3-PIT with Bounded Top Fanin

Sequence of solutions for 3-PIT with top sum gate of fanin $k$:

[Dvir-Shpilka 2005] White-box $2^{(\log s)^k}$ time algorithm.


[Karnin-Shpilka 2008] Black-box $s^{O(\log^k s)}$ time algorithm.

[Saxena-Seshadri 2009] Black-box $s^{k^3 \log s}$ time algorithm.

[Kayal-Saraf 2009] Black-box $s^{k^k}$ time algorithm over characteristic zero fields.

[Saxena-Seshadri 2011] Black-box $s^{O(k)}$ time algorithm.

[A-Saha-Saptharishi-Saxena 2012] Black-box $s^{O(k)}$ time algorithm for zero or large characteristic fields.
Sequence of solutions for 3-PIT with top sum gate of fanin $k$:

[Dvir-Shpilka 2005] White-box $2^{(\log s)^k}$ time algorithm.


[Karnin-Shpilka 2008] Black-box $s^{O(\log^k s)}$ time algorithm.

[Saxena-Seshadri 2009] Black-box $s^{k^3 \log s}$ time algorithm.

[Kayal-Saraf 2009] Black-box $s^{k^k}$ time algorithm over characteristic zero fields.

[Saxena-Seshadri 2011] Black-box $s^{O(k)}$ time algorithm.

[A-Saha-Saptharishi-Saxena 2012] Black-box $s^{O(k)}$ time algorithm for zero or large characteristic fields.
3-PIT with Bounded Top Fanin

Sequence of solutions for 3-PIT with top sum gate of fanin $k$:

[**Dvir-Shpilka 2005**] White-box $2^{(\log s)^k}$ time algorithm.

[**Kayal-Saxena 2006**] White-box $s^{O(k)}$ time algorithm.

[**Karnin-Shpilka 2008**] Black-box $s^{O(\log^k s)}$ time algorithm.

[**Saxena-Seshadri 2009**] Black-box $s^{k^3 \log s}$ time algorithm.

[**Kayal-Saraf 2009**] Black-box $s^{k^k}$ time algorithm over characteristic zero fields.

[**Saxena-Seshadri 2011**] Black-box $s^{O(k)}$ time algorithm.

[**A-Saha-Saptharishi-Saxena 2012**] Black-box $s^{O(k)}$ time algorithm for zero or large characteristic fields.
3-PIT with Bounded Top Fanin

Sequence of solutions for 3-PIT with top sum gate of fanin $k$:

[Dvir-Shpilka 2005] White-box $2^{(\log s)^k^2}$ time algorithm.


[Karnin-Shpilka 2008] Black-box $s^{O(\log^k s)}$ time algorithm.

[Saxena-Seshadri 2009] Black-box $s^{k^3 \log s}$ time algorithm.

[Kayal-Saraf 2009] Black-box $s^{k^k}$ time algorithm over characteristic zero fields.

[Saxena-Seshadri 2011] Black-box $s^{O(k)}$ time algorithm.

[A-Saha-Saptharishi-Saxena 2012] Black-box $s^{O(k)}$ time algorithm for zero or large characteristic fields.
Sequence of solutions for 3-PIT with top sum gate of fanin $k$:

[Dvir-Shpilka 2005] White-box $2^{(\log s)^k}$ time algorithm.


[Karnin-Shpilka 2008] Black-box $s^{O(\log^k s)}$ time algorithm.

[Saxena-Seshadri 2009] Black-box $s^{k^3 \log s}$ time algorithm.

[Kayal-Saraf 2009] Black-box $s^{k^k}$ time algorithm over characteristic zero fields.

[Saxena-Seshadri 2011] Black-box $s^{O(k)}$ time algorithm.

[A-Saha-Saptharishi-Saxena 2012] Black-box $s^{O(k)}$ time algorithm for zero or large characteristic fields.
3-PIT with Bounded Top Fanin

Sequence of solutions for 3-PIT with top sum gate of fanin $k$:

[Dvir-Shpilka 2005] White-box $2^{(\log s)^k^2}$ time algorithm.


[Karnin-Shpilka 2008] Black-box $s^{O(\log^k s)}$ time algorithm.

[Saxena-Seshadri 2009] Black-box $s^{k^3 \log s}$ time algorithm.

[Kayal-Saraf 2009] Black-box $s^{k^k}$ time algorithm over characteristic zero fields.

[Saxena-Seshadri 2011] Black-box $s^{O(k)}$ time algorithm.

[A-Saha-Saptharishi-Saxena 2012] Black-box $s^{O(k)}$ time algorithm for zero or large characteristic fields.
Jacobian Based Algorithm

- Let $P = \sum_{i=1}^{k} T_i$, $T_i = \prod_{j=1}^{s} L_{i,j}$ be the given circuit with $L_{i,j} = \alpha_{i,j,0} + \sum_{\ell=1}^{n} \alpha_{i,j,\ell}x_{\ell}$.
- Assume that $P \neq 0$ and $T_i$’s are algebraically independent:
  - There is no polynomial $Q(y_1, y_2, \ldots, y_k)$ such that $Q(T_1, T_2, \ldots, T_k) = 0$.
- For characteristic zero or $> s^k$: $T_1, \ldots, T_k$ are algebraically independent iff $J(T_1, T_2, \ldots, T_k)$ has full rank, where

$$J(y_1, y_2, \ldots, y_k) = \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial y_k}{\partial x_1} & \frac{\partial y_k}{\partial x_2} & \cdots & \frac{\partial y_k}{\partial x_n}
\end{bmatrix}.$$
Jacobian Based Algorithm

- Let \( P = \sum_{i=1}^{k} T_i, \quad T_i = \prod_{j=1}^{s} L_{i,j} \) be the given circuit with \( L_{i,j} = \alpha_{i,j,0} + \sum_{\ell=1}^{n} \alpha_{i,j,\ell} x_{\ell} \).
- Assume that \( P \neq 0 \) and \( T_i \)'s are algebraically independent:
  - There is no polynomial \( Q(y_1, y_2, \ldots, y_k) \) such that \( Q(T_1, T_2, \ldots, T_k) = 0 \).
- For characteristic zero or \( > s^k \): \( T_1, \ldots, T_k \) are algebraically independent iff \( J(T_1, T_2, \ldots, T_k) \) has full rank, where

\[
J(y_1, y_2, \ldots, y_k) = \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial y_k}{\partial x_1} & \frac{\partial y_k}{\partial x_2} & \cdots & \frac{\partial y_k}{\partial x_n}
\end{bmatrix}.
\]
Jacobian Based Algorithm

- Let \( P = \sum_{i=1}^{k} T_i \), \( T_i = \prod_{j=1}^{s} L_{i,j} \) be the given circuit with \( L_{i,j} = \alpha_{i,j,0} + \sum_{\ell=1}^{n} \alpha_{i,j,\ell} x_{\ell} \).
- Assume that \( P \neq 0 \) and \( T_i \)'s are algebraically independent:
  - There is no polynomial \( Q(y_1, y_2, \ldots, y_k) \) such that \( Q(T_1, T_2, \ldots, T_k) = 0 \).
- For characteristic zero or \( > s^k \): \( T_1, \ldots, T_k \) are algebraically independent iff \( J(T_1, T_2, \ldots, T_k) \) has full rank, where

\[
J(y_1, y_2, \ldots, y_k) = \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial y_k}{\partial x_1} & \frac{\partial y_k}{\partial x_2} & \cdots & \frac{\partial y_k}{\partial x_n}
\end{bmatrix}.
\]
Therefore, $J(T_1, \ldots, T_k)$ has rank $k$.

We have:

$$J(T_1, T_2, \ldots, T_k) = \begin{bmatrix}
\frac{\partial T_1}{\partial x_1} & \frac{\partial T_1}{\partial x_2} & \cdots & \frac{\partial T_1}{\partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial T_k}{\partial x_1} & \frac{\partial T_k}{\partial x_2} & \cdots & \frac{\partial T_k}{\partial x_n}
\end{bmatrix} = \begin{bmatrix}
T_1 \sum_{j=1}^{d} \frac{\alpha_{1,j,1}}{L_{1,j}} & \cdots & T_1 \sum_{j=1}^{d} \frac{\alpha_{1,j,n}}{L_{1,j}} \\
\vdots & \ddots & \vdots \\
T_k \sum_{j=1}^{d} \frac{\alpha_{k,j,1}}{L_{k,j}} & \cdots & T_k \sum_{j=1}^{d} \frac{\alpha_{k,j,n}}{L_{k,j}}
\end{bmatrix}$$
Therefore, \( J(T_1, \ldots, T_k) \) has rank \( k \).

We have:

\[
J(T_1, T_2, \ldots, T_k) = \begin{bmatrix}
\frac{\partial T_1}{\partial x_1} & \frac{\partial T_1}{\partial x_2} & \cdots & \frac{\partial T_1}{\partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial T_k}{\partial x_1} & \frac{\partial T_k}{\partial x_2} & \cdots & \frac{\partial T_k}{\partial x_n}
\end{bmatrix}
\]

\[
= \begin{bmatrix}
T_1 \sum_{j=1}^d \frac{\alpha_{1,j,1}}{L_{1,j}} & \cdots & T_1 \sum_{j=1}^d \frac{\alpha_{1,j,n}}{L_{1,j}} \\
\vdots & \ddots & \vdots \\
T_k \sum_{j=1}^d \frac{\alpha_{k,j,1}}{L_{k,j}} & \cdots & T_k \sum_{j=1}^d \frac{\alpha_{k,j,n}}{L_{k,j}}
\end{bmatrix}
\]
**Jacobian Based Algorithm**

- Assume, wlog, that columns corresponding to variables $x_1, x_2, \ldots, x_k$ have rank $k$.

- Let

$$
\hat{P} = \begin{vmatrix}
T_1 \sum_{j=1}^{d} \frac{\alpha_{1,j,1}}{L_{1,j}} & \cdots & T_1 \sum_{j=1}^{d} \frac{\alpha_{1,j,k}}{L_{1,j}} \\
\vdots & \ddots & \vdots \\
T_k \sum_{j=1}^{d} \frac{\alpha_{k,j,1}}{L_{k,j}} & \cdots & T_k \sum_{j=1}^{d} \frac{\alpha_{k,j,k}}{L_{k,j}}
\end{vmatrix}
$$

$$
= \prod_{i=1}^{k} T_i \cdot \begin{vmatrix}
\sum_{j=1}^{d} \frac{\alpha_{1,j,1}}{L_{1,j}} & \cdots & \sum_{j=1}^{d} \frac{\alpha_{1,j,k}}{L_{1,j}} \\
\vdots & \ddots & \vdots \\
\sum_{j=1}^{d} \frac{\alpha_{k,j,1}}{L_{k,j}} & \cdots & \sum_{j=1}^{d} \frac{\alpha_{k,j,k}}{L_{k,j}}
\end{vmatrix}
$$

$$
= \prod_{i=1}^{k} T_i \cdot R,
$$

where $R$ is a sparse rational function.
**Jacobian Based Algorithm**

- Assume, wlog, that columns corresponding to variables $x_1, x_2, \ldots, x_k$ have rank $k$.
- Let

$$\hat{P} = \begin{vmatrix}
T_1 \sum_{j=1}^{d} \frac{\alpha_{1,j,1}}{L_{1,j}} & \cdots & T_1 \sum_{j=1}^{d} \frac{\alpha_{1,j,k}}{L_{1,j}} \\
\vdots & \ddots & \vdots \\
T_k \sum_{j=1}^{d} \frac{\alpha_{k,j,1}}{L_{k,j}} & \cdots & T_k \sum_{j=1}^{d} \frac{\alpha_{k,j,k}}{L_{k,j}}
\end{vmatrix}$$

$$= \prod_{i=1}^{k} T_i \cdot R,$$

where $R$ is a sparse rational function.
Since $\hat{P}$ is a product of sparse polynomials and rational functions, the set of substitutions as used for 2-PIT will ensure that $\hat{P}$ remains non-zero under one of them.

For this substitution, the Jacobian has full rank and therefore the circuit output remains non-zero.
Since $\hat{P}$ is a product of sparse polynomials and rational functions, the set of substitutions as used for 2-PIT will ensure that $\hat{P}$ remains non-zero under one of them.

- For this substitution, the Jacobian has full rank and therefore the circuit output remains non-zero.
3-PIT for Diagonal Circuits

Diagonal Circuits
Circuits where each multiplication gate is a powering gate.

Theorem (Forbes-Shpilka 2012, A-Saha-Saxena 2013)
There exists a $s^{O(\log s)}$-time black-box algorithm for diagonal 3-PIT.
3-PIT for Diagonal Circuits

Diagonal Circuits
Circuits where each multiplication gate is a powering gate.

Theorem (Forbes-Shpilka 2012, A-Saha-Saxena 2013)

There exists a $s^{O(\log s)}$-time black-box algorithm for diagonal 3-PIT.
Let $P = \sum_{i=1}^{k} T_i$, $T_i = L_i^d$ be the given circuit with $L_i = \alpha_{i,0} + \sum_{\ell=1}^{n} \alpha_{i,\ell} x_\ell$.

The polynomial can be rewritten as:

$$P = \bar{1} \cdot (\bar{u}_0 + \bar{u}_1 x_1 + \cdots + \bar{u}_n x_n)^d,$$

where $\bar{u}_\ell = [\alpha_{1,\ell} \cdots \alpha_{k,\ell}]$. 
Let $P = \sum_{i=1}^{k} T_i$, $T_i = L_i^d$ be the given circuit with $L_i = \alpha_{i,0} + \sum_{\ell=1}^{n} \alpha_{i,\ell} x_\ell$.

The polynomial can be rewritten as:

$$P = \bar{1} \cdot (\bar{u}_0 + \bar{u}_1 x_1 + \cdots + \bar{u}_n x_n)^d,$$

where $\bar{u}_\ell = [\alpha_{1,\ell} \cdots \alpha_{k,\ell}]$. 
Now consider the following polynomial with vectors over $F^k$ as coefficients:

$$Q = (\bar{u}_0 + \bar{u}_1 x_1 + \cdots + \bar{u}_n x_n)^d = \sum_{S \in [0,d]^n} \bar{v}_S \bar{x}^S$$

where $S = (d_1, d_2, \ldots, d_n)$, $\bar{v}_S$ is Hadamard product of $d$ $\bar{u}$'s, and $\bar{x}^S = \prod_{i=1}^n x_i^{d_i}$.

Consider the vectors $\bar{v}_S \in F^k$.

Let the dimension of the space spanned by these vectors be $m \leq k$. 

Rank Concentration Based Algorithm
Now consider the following polynomial with vectors over $F^k$ as coefficients:

$$Q = (\bar{u}_0 + \bar{u}_1 x_1 + \cdots + \bar{u}_n x_n)^d$$

$$= \sum_{S \in [0,d]^n} \bar{v}_S \bar{x}^S$$

where $S = (d_1, d_2, \ldots, d_n)$, $\bar{v}_S$ is Hadamard product of $d$ $\bar{u}$'s, and $\bar{x}^S = \prod_{i=1}^{n} x_i^{d_i}$.

Consider the vectors $\bar{v}_S \in F^k$.

Let the dimension of the space spanned by these vectors be $m \leq k$. 
**Rank Concentration Based Algorithm**

- $\ell$-rank concentration is the property that $\bar{v}_S$ of support $\ell$ (i.e., $S$ with only $\ell$ non-zero $d_i$’s) span this space.

- If there is $\ell$-rank concentration, the PIT can be solved by setting all but $\ell$ $x$’s to zero and evaluating the resulting polynomial.
**Rank Concentration Based Algorithm**

- \(\ell\)-rank concentration is the property that \(\bar{\nu}_S\) of support \(\ell\) (i.e., \(S\) with only \(\ell\) non-zero \(d_i\)'s) span this space.

- If there is \(\ell\)-rank concentration, the PIT can be solved by setting all but \(\ell\) \(x\)'s to zero and evaluating the resulting polynomial.
The space spanned by $\bar{v}_S$ has $\log m$-rank concentration:

- Consider a monomial $\bar{x}^S$ with support $> \log m$. It has $> m$ monomials strictly below it in lex-ordering.
- There must be linear dependence between coefficients associated with these lower monomials.
- Define a total ordering on monomials by fixing an arbitrary order between variables.
- Take a linear dependence equation for lower monomial coefficients, identify the largest monomial in total order, and multiply the equation with coefficient of a monomial such that the largest monomial becomes $\bar{x}^S$.
- This makes coefficient of $\bar{x}^S$ linearly dependent on smaller monomial coefficients in total order.
The space spanned by $\bar{v}_S$ has $\log m$-rank concentration:

- Consider a monomial $\bar{x}^S$ with support $> \log m$. It has $> m$ monomials strictly below it in lex-ordering.
- There must be linear dependence between coefficients associated with these lower monomials.
- Define a total ordering on monomials by fixing an arbitrary order between variables.
- Take a linear dependence equation for lower monomial coefficients, identify the largest monomial in total order, and multiply the equation with coefficient of a monomial such that the largest monomial becomes $\bar{x}^S$.
- This makes coefficient of $\bar{x}^S$ linearly dependent on smaller monomial coefficients in total order.
The space spanned by $\bar{v}_S$ has $\log m$-rank concentration:

- Consider a monomial $\bar{x}^S$ with support $> \log m$. It has $> m$ monomials strictly below it in lex-ordering.
- There must be linear dependence between coefficients associated with these lower monomials.
- Define a total ordering on monomials by fixing an arbitrary order between variables.
- Take a linear dependence equation for lower monomial coefficients, identify the largest monomial in total order, and multiply the equation with coefficient of a monomial such that the largest monomial becomes $\bar{x}^S$.
- This makes coefficient of $\bar{x}^S$ linearly dependent on smaller monomial coefficients in total order.
The space spanned by $\bar{v}_S$ has $\log m$-rank concentration:

- Consider a monomial $\bar{x}^S$ with support $> \log m$. It has $> m$ monomials strictly below it in lex-ordering.
- There must be linear dependence between coefficients associated with these lower monomials.
- Define a total ordering on monomials by fixing an arbitrary order between variables.
- Take a linear dependence equation for lower monomial coefficients, identify the largest monomial in total order, and multiply the equation with coefficient of a monomial such that the largest monomial becomes $\bar{x}^S$.
- This makes coefficient of $\bar{x}^S$ linearly dependent on smaller monomial coefficients in total order.
The space spanned by $\bar{v}_S$ has log $m$-rank concentration:

- Consider a monomial $\bar{x}^S$ with support $> \log m$. It has $> m$ monomials strictly below it in lex-ordering.
- There must be linear dependence between coefficients associated with these lower monomials.
- Define a total ordering on monomials by fixing an arbitrary order between variables.
- Take a linear dependence equation for lower monomial coefficients, identify the largest monomial in total order, and multiply the equation with coefficient of a monomial such that the largest monomial becomes $\bar{x}^S$.
- This makes coefficient of $\bar{x}^S$ linearly dependent on smaller monomial coefficients in total order.
**Rank Concentration Based Algorithm**

- The algorithm is now simple: for all subsets of $\log m$ variables, set the remaining variables to zero, and test if the resulting polynomial is zero on $d^{\log m}$ distinct values.
- This gives a $d^{O(\log d)}$-time black-box algorithm.
- In certain situations, there may not be rank concentration to begin with.
- So first apply a transformation on variables that yields rank concentration.
- For certain other restrictions of 3-PIT, the following transformation works:
  \[ x_i \mapsto x_i + t^{d_i} \]
  for small $d_i$'s.
The algorithm is now simple: for all subsets of $\log m$ variables, set the remaining variables to zero, and test if the resulting polynomial is zero on $d^{\log m}$ distinct values.

This gives a $d^{O(\log d)}$-time black-box algorithm.

In certain situations, there may not be rank concentration to begin with.

So first apply a transformation on variables that yields rank concentration.

For certain other restrictions of 3-PIT, the following transformation works:

$$x_i \mapsto x_i + t^{d_i}$$

for small $d_i$'s.
The algorithm is now simple: for all subsets of $\log m$ variables, set the remaining variables to zero, and test if the resulting polynomial is zero on $d^{\log m}$ distinct values.

This gives a $d^{O(\log d)}$-time black-box algorithm.

In certain situations, there may not be rank concentration to begin with.

So first apply a transformation on variables that yields rank concentration.

For certain other restrictions of 3-PIT, the following transformation works:

$$x_i \mapsto x_i + t^{d_i}$$

for small $d_i$'s.
**Rank Concentration Based Algorithm**

- The algorithm is now simple: for all subsets of $\log m$ variables, set the remaining variables to zero, and test if the resulting polynomial is zero on $d^{\log m}$ distinct values.
- This gives a $d^{O(\log d)}$-time black-box algorithm.
- In certain situations, there may not be rank concentration to begin with.
- So first apply a transformation on variables that yields rank concentration.
- For certain other restrictions of 3-PIT, the following transformation works:

  $$x_i \mapsto x_i + t^{d_i}$$

  for small $d_i$'s.