Enumeration: logical and algebraic approach

Yann Strozecki

Université Paris Sud - Paris 11

Novembre 2011, séminaire ALGO/LIX
Introduction to Enumeration

Enumeration and logic

Enumeration and polynomials
Enumeration problems

Polynomially balanced predicate $A(x, y)$, decidable in polynomial time.

- $\exists? y A(x, y)$: decision problem (class NP)
- $\#\{ y \mid A(x, y) \}$: counting problem (class $\#P$)
- $\{ y \mid A(x, y) \}$: enumeration problem (class EnumP)

Example

Perfect matching:
- The decision problem is to decide if there is a perfect matching.
- The counting problem is to count the number of perfect matchings.
- The enumeration problem is to list every perfect matching.
Enumeration problems

Polynomially balanced predicate $A(x, y)$, decidable in polynomial time.

- $\exists?y A(x, y)$: decision problem (class NP)
- $\#\{y \mid A(x, y)\}$: counting problem (class $\#P$)
- $\{y \mid A(x, y)\}$: enumeration problem (class EnumP)

Example

Perfect matching:
- The decision problem is to decide if there is a perfect matching.
- The counting problem is to count the number of perfect matchings.
- The enumeration problem is to list every perfect matching.
Time complexity measures for enumeration

1. the total time related to the number of solutions
   - polynomial total time: \( \text{TotalP} \)

2. the delay
   - incremental polynomial time: \( \text{IncP} \) (Circuits of a matroid)
   - polynomial delay: \( \text{DelayP} \) (Perfect Matching [Uno])
   - Constant or linear delay
     - A two steps algorithm: preprocessing + generation
     - An ad-hoc RAM model.
Time complexity measures for enumeration

1. the total time related to the number of solutions
   ▶ polynomial total time: TotalP

2. the delay
   ▶ incremental polynomial time: IncP (Circuits of a matroid)
   ▶ polynomial delay: DelayP (Perfect Matching [Uno])
   ▶ Constant or linear delay
     ▶ A two steps algorithm: preprocessing + generation
     ▶ An ad-hoc RAM model.
Enumeration problems

\( R \): polynomially balanced binary predicate

**Definition**

The problem **Enum**\cdot\( R \) belongs to the class **Delay**\((g, f)\) if there exists an enumeration algorithm that computes **Enum**\cdot\( R \) such that, for all input \( x \):

- Preprocessing in time \( O(g(|x|)) \),
- Solutions \( y \in R(x) \) are computed successively without repetition with a delay \( O(f(|x|)) \)

\( \text{Constant-Delay} = \bigcup_k \text{Delay}(n^k, 1) \).
Enumeration complexity classes

Separation:

\[ \text{QueryP} \subsetneq \text{SDelayP} \subseteq \text{DelayP} \subseteq \text{IncP} \subsetneq \text{TotalP} \subsetneq \text{EnumP}. \]
Separation:

\[ \text{QueryP} \subsetneq \text{SDelayP} \subseteq \text{DelayP} \subseteq \text{IncP} \subsetneq \text{TotalP} \subsetneq \text{EnumP}. \]
Enumeration complexity classes

Separation:

\[ \text{QueryP} \subsetneq \text{SDelayP} \subseteq \text{DelayP} \subseteq \text{IncP} \subsetneq \text{TotalP} \subsetneq \text{EnumP}. \]

Complete problem:

No good notion of reduction out of parsimonious reduction.
Enumeration complexity classes

Separation:

QueryP $\subsetneq$ SDelayP $\subseteq$ DelayP $\subseteq$ IncP $\subsetneq$ TotalP $\subsetneq$ EnumP.

Complete problem:

No good notion of reduction out of parsimonious reduction.
Proposition

If P \neq NP then the classes DelayP, IncP and TotalP are not stable by subtraction.

Proposition

If P \neq NP then the classes DelayP, IncP and TotalP are not stable by intersection.
Boolean combination of solutions

**Proposition**

*If* $P \neq NP$ *then the classes* Delay$P$, Inc$P$ *and Total$P$ *are not stable by subtraction.*

**Proposition**

*If* $P \neq NP$ *then the classes* Delay$P$, Inc$P$ *and Total$P$ *are not stable by intersection.*

The classes Delay$P$, Inc$P$ and Total$P$ are stable for:

- disjoint union
Boolean combination of solutions

Proposition

If $P \neq NP$ then the classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are not stable by subtraction.

Proposition

If $P \neq NP$ then the classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are not stable by intersection.

The classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are stable for:

- disjoint union
- union with an order
Boolean combination of solutions

Proposition

If $P \neq NP$ then the classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are not stable by subtraction.

Proposition

If $P \neq NP$ then the classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are not stable by intersection.

The classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are stable for:

- disjoint union
- union with an order
- union without order
Boolean combination of solutions

Proposition

If $P \neq NP$ then the classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are not stable by subtraction.

Proposition

If $P \neq NP$ then the classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are not stable by intersection.

The classes $\text{DelayP}$, $\text{IncP}$ and $\text{TotalP}$ are stable for:

- disjoint union
- union with an order
- union without order
Meta-algorithms for enumeration and CSP

Proposition (Creignou, Hebrard’97)

The problem $\text{Enum-SAT}(C)$ is in $\text{DelayP}$ when $C$ is one of the following classes: Horn formulas, anti-Horn formulas, affine formulas, bijunctive ($2\text{CNF}$) formulas.

Other meta-algorithms:

1. Schnoor: enumeration complexity dichotomy for conservative CSP over three element domain
2. Bulatov, Dalmau, Grohe, Marx: algebraic characterization of easy to enumerate CSP, bounded tree-width domain.
Meta-algorithms for enumeration and CSP

Proposition (Creignou, Hebrard’97)

The problem $\text{Enum-SAT}(\mathcal{C})$ is in $\text{DelayP}$ when $\mathcal{C}$ is one of the following classes: Horn formulas, anti-Horn formulas, affine formulas, bijunctive (2CNF) formulas.

Other meta-algorithms:

1. Schnoor: enumeration complexity dichotomy for conservative CSP over three element domain
2. Bulatov, Dalmau, Grohe, Marx: algebraic characterization of easy to enumerate CSP, bounded tree-width domain.
Introduction to Enumeration

Enumeration and logic

Enumeration and polynomials
First order logic (FO):

- Variables: $x, y, z \ldots$
- The language $\sigma$, relations and functions: $R(x, y), f(z)$
- Unary and binary connectors: $\wedge, \lor, \neg$
- Quantifiers: $\forall, \exists$

\[
\varphi \equiv \forall x \exists y E(x, y) \lor E(y, x)
\]
First order logic (FO):

- Variables: $x, y, z \ldots$
- The language $\sigma$, relations and functions: $R(x, y), f(z)$
- Unary and binary connectors: $\land, \lor, \lnot$
- Quantifiers: $\forall, \exists$
- $\varphi \equiv \forall x \exists y E(x, y) \lor E(y, x)$

Theorem (Goldberg)

For almost all first order graph property $\varphi$, the graphs of size $n$ which satisfies $\varphi$ can be enumerated with polynomial delay in $n$. 
First order logic (FO):

- Variables: \(x, y, z\ldots\)
- The language \(\sigma\), relations and functions: \(R(x, y), f(z)\)
- Unary and binary connectors: \(\land, \lor, \neg\)
- Quantifiers: \(\forall, \exists\)
- \(\varphi \equiv \forall x \exists y E(x, y) \lor E(y, x)\)

**Theorem (Goldberg)**

For almost all first order graph property \(\varphi\), the graphs of size \(n\) which satisfies \(\varphi\) can be enumerated with polynomial delay in \(n\).
Enumeration problem defined by a formula

**Second order logic (SO):**
Second order variable: $T$, denotes unknown relation over the domain.

Let $\Phi(z, T)$ be a first order formula with free first and second order variables.
Enumeration problem defined by a formula

Second order logic (SO):
Second order variable: $T$, denotes unknown relation over the domain.

Let $\Phi(z, T)$ be a first order formula with free first and second order variables.

\[ \text{Enum} \cdot \Phi \]

Input: A $\sigma$-structure $S$

Output: $\Phi(S) = \{(z^*, T^*) : (S, z^*, T^*) \models \Phi(z, T)\}$

Let $\mathcal{F}$ be a subclass of first order formulas. We denote by $\text{Enum} \cdot \mathcal{F}$ the collection of problems $\text{Enum} \cdot \Phi$ for $\Phi \in \mathcal{F}$. 
Enumeration problem defined by a formula

**Second order logic (SO):**
Second order variable: $T$, denotes unknown relation over the domain.

Let $\Phi(z, T)$ be a first order formula with free first and second order variables.

**$\text{Enum} \cdot \Phi$**

*Input:* A $\sigma$-structure $S$

*Output:* $\Phi(S) = \{(z^*, T^*) : (S, z^*, T^*) \models \Phi(z, T)\}$

Let $\mathcal{F}$ be a subclass of first order formulas. We denote by $\text{Enum} \cdot \mathcal{F}$ the collection of problems $\text{Enum} \cdot \Phi$ for $\Phi \in \mathcal{F}$. 
Independent sets:

\[ IS(T) \equiv \forall x \forall y \ T(x) \land T(y) \Rightarrow \neg E(x, y). \]

Hitting sets (vertex covers) of a hypergraph represented by the incidence structure \( \langle D, \{ V, E, R \} \rangle \).

\[ HS(T) \equiv \forall x \ (T(x) \Rightarrow V(x)) \land \forall y \exists x \ E(y) \Rightarrow (T(x) \land R(x, y)) \]
First-order queries with free second order variables

This presentation

- FO queries with free second-order variables
- Data complexity: the query is fixed
- The complexity in term of the size of the input structure’s domain
- Quantifier depth as a parameter: $\text{Enum} \cdot \Sigma_1$
- $\text{Enum} \cdot \text{IS} \in \text{Enum} \cdot \Pi_1$ and $\text{Enum} \cdot \text{HS} \in \text{Enum} \cdot \Pi_2$
First-order queries with free second order variables

This presentation

- FO queries with free second-order variables
- Data complexity: the query is fixed
- The complexity in term of the size of the input structure’s domain
- Quantifier depth as a parameter: $\text{Enum} \cdot \Sigma_1$
- $\text{Enum} \cdot \text{IS} \in \text{Enum} \cdot \Pi_1$ and $\text{Enum} \cdot \text{HS} \in \text{Enum} \cdot \Pi_2$
Previous results

1. Only first-order free variables and bounded degree structures. Durand-Grandjean’07, Lindell’08, Kazana-Segoufin’10: **linear preprocessing + constant delay**.

2. Only first-order free variables and acyclic conjunctive formula. Bagan-Durand-Grandjean’07: **linear preprocessing + linear delay**.

Example

Enumeration of the $k$-cliques of a graph of bounded degree.
Previous results


2. Only first-order free variables and acyclic conjunctive formula. Bagan-Durand-Grandjean’07: linear preprocessing + linear delay


Example

Typical database query. Simple paths of length $k$. 
Previous results


2. Only first-order free variables and acyclic conjunctive formula. Bagan-Durand-Grandjean’07: linear preprocessing + linear delay


Example

Enumeration of the cliques of a bounded tree-width graph.
A hierarchy result for counting functions

From a formula $\Phi(z, T)$, one defines the counting function:

$$\#\Phi : S \mapsto |\Phi(S)|.$$ 

**Theorem (Saluja, Subrahmanyam, Thakur 1995)**

*On linearly ordered structures:*

$$\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2 = \#\mathbb{P}.$$ 

Some $\#\mathbb{P}$-hard problems in $\#\Sigma_1$ (but existence of FPRAS at this level).

**Corollary**

*On linearly ordered structures:*

$$\text{Enum} \cdot \Sigma_0 \subsetneq \text{Enum} \cdot \Sigma_1 \subsetneq \text{Enum} \cdot \Pi_1 \subsetneq \text{Enum} \cdot \Sigma_2 \subsetneq \text{Enum} \cdot \Pi_2.$$
A hierarchy result for counting functions

From a formula \( \Phi(z, T) \), one defines the counting function:

\[
\# \Phi : S \mapsto |\Phi(S)|.
\]

**Theorem (Saluja, Subrahmanyam, Thakur 1995)**

*On linearly ordered structures:*

\[
\# \Sigma_0 \subsetneq \# \Sigma_1 \subsetneq \# \Pi_1 \subsetneq \# \Sigma_2 \subsetneq \# \Pi_2 = \# P.
\]

Some \# P-hard problems in \# \Sigma_1 (but existence of FPRAS at this level).

**Corollary**

*On linearly ordered structures:*

\[
\text{Enum} \cdot \Sigma_0 \subsetneq \text{Enum} \cdot \Sigma_1 \subsetneq \text{Enum} \cdot \Pi_1 \subsetneq \text{Enum} \cdot \Sigma_2 \subsetneq \text{Enum} \cdot \Pi_2.
\]
The first level: Enum\textcdot\Sigma_0

Theorem

For \( \varphi \in \Sigma_0 \), \( \text{Enum}\cdot\varphi \) can be enumerated with preprocessing \( O(|D|^k) \) and delay \( O(1) \) where \( k \) is the number of free first order variables of \( \varphi \) and \( D \) is the domain of the input structure.

Simple ingredients:

1. Transformation of a f.o. formula \( \Phi(z, T) \) into a propositional formula:
   - Try all values for first order variables:
     \( \Phi(z^*, T) \).
   - Replace the atomic formulas by their truth value.
   - Obtain a propositional formula with variables \( T(w) \).

Remark: The $k$-clique query is definable. No hope to improve the $O(|D|^k)$ preprocessing.

Theorem

Let $d \in \mathbb{N}$, on $d$-degree bounded input structures, $\text{Enum} \cdot \Sigma_0 \in \text{Delay}(|D|, 1)$ where $D$ is the domain of the input structure.
**Remark:** The \( k \)-clique query is definable. No hope to improve the \( O(|D|^k) \) preprocessing.

**Theorem**

Let \( d \in \mathbb{N} \), on \( d \)-degree bounded input structures, 
\( \text{Enum} \cdot \Sigma_0 \in \text{Delay}(|D|, 1) \) where \( D \) is the domain of the input structure.

**Idea of proof:**

- Another transformation: \( \Phi(z, T) \) seen as a propositional formula whose variables are the atoms of \( \Phi \).
- From each solution, create a quantifier free formula without free second order variables.
- Enumerate the solutions of this formula thanks to [DG 2007].
Remark: The $k$-clique query is definable. No hope to improve the $O(|D|^k)$ preprocessing.

Theorem

Let $d \in \mathbb{N}$, on $d$-degree bounded input structures, 
$\text{Enum} \cdot \Sigma_0 \in \text{Delay}(|D|, 1)$ where $D$ is the domain of the input structure.

Idea of proof:

- Another transformation: $\Phi(z, T)$ seen as a propositional formula whose variables are the atoms of $\Phi$.
- From each solution, create a quantifier free formula without free second order variables.
- Enumerate the solutions of this formula thanks to [DG 2007].
Theorem

$\text{Enum} \cdot \Sigma_1 \subseteq \text{DelayP}$. More precisely, $\text{Enum} \cdot \Sigma_1$ can be computed with precomputation $O(|D|^{h+k})$ and delay $O(|D|^k)$ where $h$ is the number of free first order variables of the formula, $k$ the number of existentially quantified variables and $D$ is the domain of the input structure.

Idea of Proof: $\Phi(y, T) = \exists x \varphi(x, y, T)$

- Substitute values for $x$. Collection of formulas of the form:

  $\varphi(x^*, y, T)$

- Need to enumerate the (non necessarily disjoint) union.
The case $\text{Enum} \cdot \Pi_1$

**Proposition**

Unless $P = NP$, there is no polynomial delay algorithm for $\text{Enum} \cdot \Pi_1$.

**Proof** Direct encoding of SAT.

Hardness even:

- on the class of bounded degree structure
- if all clauses but one have at most two occurrences of a second-order free variable
Problem $\text{Enum} \cdot \Phi$ with $\Phi \in \Sigma_i$: transformation of $\Phi$ into a propositional formula $\tilde{\Phi}$.

**Corollary**

Let $\Phi(z, T)$ be a formula, such that, for all $\sigma$ structures, all propositional formulas $\tilde{\Phi}$ are either Horn, anti-Horn, affine or bijunctive. Then $\text{Enum} \cdot \Phi \subseteq \text{DelayP}$. 
Problem $\text{Enum} \cdot \Phi$ with $\Phi \in \Sigma_i$: transformation of $\Phi$ into a propositional formula $\tilde{\Phi}$.

**Corollary**

Let $\Phi(z, T)$ be a formula, such that, for all $\sigma$ structures, all propositional formulas $\tilde{\Phi}$ are either Horn, anti-Horn, affine or bijunctive. Then $\text{Enum} \cdot \Phi \subseteq \text{DelayP}$.

**Example:** independent sets and hitting sets.
Problem $\text{Enum} \cdot \Phi$ with $\Phi \in \Sigma_i$: transformation of $\Phi$ into a propositional formula $\tilde{\Phi}$.

**Corollary**

Let $\Phi(z, T)$ be a formula, such that, for all $\sigma$ structures, all propositional formulas $\tilde{\Phi}$ are either Horn, anti-Horn, affine or bijunctive. Then $\text{Enum} \cdot \Phi \subseteq \text{DelayP}$.

**Example:** independent sets and hitting sets.
\[ \text{Enum} \cdot \Sigma_0 \subsetneq \text{Enum} \cdot \Sigma_1 \subsetneq \text{Enum} \cdot \Pi_1 \subsetneq \text{Enum} \cdot \Sigma_2 \subsetneq \text{Enum} \cdot \Pi_2 = \text{EnumP}. \]

- Nice but small hierarchy.
- Other tractable classes above \( \Sigma_1 \) (optimization operator)?
- Efficient probabilistic enumeration procedure?
Introduction to Enumeration

Enumeration and logic

Enumeration and polynomials
Polynomial given by a black-box

\[ P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3 \]
Polynomial given by a black-box

\[ P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3 \]

\[ X_1 = 1, \ X_2 = 2, \ X_3 = 1 \]

\[ 1 \times 2 + 1 \times 1 + 2 + 1 \]

\[ Output = 6 \]
Polynomial given by a black-box

\[ P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3 \]

\[ X_1 = -1, \ X_2 = 1, \ X_3 = 2 \]

\[ -1 \times 1 + -1 \times 2 + 1 + 2 \]

Output = 0
Polynomial given by a black-box

\[ P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3 \]

- Problem: interpolation, compute \( P \) from its values.
- Complexity: time and number of calls to the oracle.
Polynomial given by a black-box

\[ P(X_1, X_2, X_3) = X_1 X_2 + X_1 X_3 + X_2 + X_3 \]

- Problem: interpolation, compute \( P \) from its values.
- Complexity: time and number of calls to the oracle.
- Parameters: number of variables and total degree.
Polynomial given by a black-box

\[ P(X_1, X_2, X_3) = X_1 X_2 + X_1 X_3 + X_2 + X_3 \]

- **Problem**: interpolation, compute \( P \) from its values.
- **Complexity**: time and number of calls to the oracle.
- **Parameters**: number of variables and total degree.

**Enumeration problem**: output the monomials one after the other.
Polynomial given by a black-box

\[ P(X_1, X_2, X_3) = X_1 X_2 + X_1 X_3 + X_2 + X_3 \]

- Problem: interpolation, compute \( P \) from its values.
- Complexity: time and number of calls to the oracle.
- Parameters: number of variables and total degree.

**Enumeration problem:** output the monomials one after the other.
Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix: cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub 2002]: spanning hypertrees of a 3-uniform hypergraph.
- The polynomial representing the language accepted by a probabilistic automaton.
  Only multilinear polynomials.
Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix: cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.

Pfaffian Hypertree theorem [Masbaum and Vaintraub 2002]: spanning hypertrees of a 3-uniform hypergraph.

The polynomial representing the language accepted by a probabilistic automaton.

Only multilinear polynomials.
Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix: cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub 2002]: spanning hypertrees of a 3-uniform hypergraph.
Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix: cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub 2002]: spanning hypertrees of a 3-uniform hypergraph.
- The polynomial representing the language accepted by a probabilistic automaton.
Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix: cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub 2002]: spanning hypertrees of a 3-uniform hypergraph.
- The polynomial representing the language accepted by a probabilistic automaton.

Only multilinear polynomials.
Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

- Determinant of the adjacency matrix: cycle covers of a graph.
- Determinant of the Kirchoff matrix: spanning trees.
- Pfaffian Hypertree theorem [Masbaum and Vaintraub 2002]: spanning hypertrees of a 3-uniform hypergraph.
- The polynomial representing the language accepted by a probabilistic automaton.

Only multilinear polynomials.
**The decision problem**

**Polynomial Identity Testing**

*Input:* a polynomial given as a black box.

*Output:* decides if the polynomial is zero.

---

**Lemma (Schwarz-Zippel)**

Let $P$ be a non zero polynomial with $n$ variables of total degree $D$, if $x_1, \ldots, x_n$ are randomly chosen in a set of integers $S$ of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by $\epsilon$. 
The decision problem

**Polynomial Identity Testing**

*Input*: a polynomial given as a black box.  
*Output*: decides if the polynomial is zero.

**Lemma (Schwarz-Zippel)**

Let $P$ be a non zero polynomial with $n$ variables of total degree $D$, if $x_1, \ldots, x_n$ are randomly chosen in a set of integers $S$ of size $\frac{D}{\epsilon}$, then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by $\epsilon$.

No way to make PIT deterministic for black box.
Polynomial Identity Testing

Input: a polynomial given as a black box.
Output: decides if the polynomial is zero.

Lemma (Schwarz-Zippel)

Let $P$ be a non-zero polynomial with $n$ variables of total degree $D$, if $x_1, \ldots, x_n$ are randomly chosen in a set of integers $S$ of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by $\epsilon$.

No way to make PIT deterministic for black box.

Error exponentially small in the size of the integers!
**The decision problem**

**Polynomial Identity Testing**

*Input*: a polynomial given as a black box.  
*Output*: decides if the polynomial is zero.

**Lemma (Schwarz-Zippel)**

Let $P$ be a non zero polynomial with $n$ variables of total degree $D$, if $x_1, \ldots, x_n$ are randomly chosen in a set of integers $S$ of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \ldots, x_n) = 0$ is bounded by $\epsilon$.

No way to make PIT deterministic for black box.

Error *exponentially small* in the size of the integers!
Existing interpolation methods

- Zippel (1990): use a dense interpolation on a polynomial with a restricted number of variables
- Ben Or and Tiwari (1988): evaluation on big power of prime numbers
Existing interpolation methods

- Zippel (1990): use a dense interpolation on a polynomial with a restricted number of variables
- Ben Or and Tiwari (1988): evaluation on big power of prime numbers
- Klivans and Spielman (2001): transformation of a multivariate into an univariate one.
Existing interpolation methods

- Zippel (1990): use a dense interpolation on a polynomial with a restricted number of variables
- Ben Or and Tiwari (1988): evaluation on big power of prime numbers
- Klivans and Spielman (2001): transformation of a multivariate into an univariate one.
- Garg and Schost (2009): non black-box but complexity independent from the degree of the polynomial
Existing interpolation methods

- Zippel (1990): use a dense interpolation on a polynomial with a restricted number of variables
- Ben Or and Tiwari (1988): evaluation on big power of prime numbers
- Klivans and Spielman (2001): transformation of a multivariate into an univariate one.
- Garg and Schost (2009): non black-box but complexity independent from the degree of the polynomial

Enumeration complexity: produce the monomials one at a time with a good delay.
Existing interpolation methods

- Zippel (1990): use a dense interpolation on a polynomial with a restricted number of variables
- Ben Or and Tiwari (1988): evaluation on big power of prime numbers
- Klivans and Spielman (2001): transformation of a multivariate into an univariate one.
- Garg and Schost (2009): non black-box but complexity independent from the degree of the polynomial

Enumeration complexity: produce the monomials one at a time with a good delay.
Assume there is a procedure which returns a monomial of a polynomial $P$, then it can be used to interpolate $P$.

**Idea:** Subtract the monomial found by the procedure to the polynomial and recurse to recover the whole polynomial.
Assume there is a procedure which returns a monomial of a polynomial $P$, then it can be used to interpolate $P$.

**Idea:** Substruct the monomial found by the procedure to the polynomial and recurse to recover the whole polynomial.

**Drawback:** one has to store the polynomial $Q = \text{the sum of the generated monomials.}$
When there is a call, compute $P - Q$. 
Assume there is a procedure which returns a monomial of a polynomial $P$, then it can be used to interpolate $P$.

**Idea:** Substract the monomial found by the procedure to the polynomial and recurse to recover the whole polynomial.

**Drawback:** one has to store the polynomial $Q = \text{the sum of the generated monomials}$.  
When there is a call, compute $P - Q$.

Incremental delay.
Assume there is a procedure which returns a monomial of a polynomial \( P \), then it can be used to interpolate \( P \).

**Idea:** Subtract the monomial found by the procedure to the polynomial and recurse to recover the whole polynomial.

**Drawback:** one has to store the polynomial \( Q = \) the sum of the generated monomials. When there is a call, compute \( P - Q \).

Incremental delay.
Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7D^4)$ calls, $n$ number of variables and $D$ the total degree

Open question: how to efficiently represent and compute the partial polynomial at each step? Easier with circuits, formulas, polynomials of low degree, over fixed finite fields?
Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7 D^4)$ calls, $n$ number of variables and $D$ the total degree
- Question: is it possible to decrease the number of calls to a more manageable polynomial.
Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7D^4)$ calls, $n$ number of variables and $D$ the total degree
- Question: is it possible to decrease the number of calls to a more manageable polynomial.
- Yes for polynomial of fixed degree $d$. One can find the "highest" degree polynomial with $O(n^2D^{d-1})$ calls.
**Finding one monomial**

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7D^4)$ calls, $n$ number of variables and $D$ the total degree
- Question: is it possible to decrease the number of calls to a more manageable polynomial.
- Yes for polynomial of fixed degree $d$. One can find the "highest" degree polynomial with $O(n^2D^{d-1})$ calls.
- Yes for polynomial whose each two monomials have distinct supports: $O(n^2)$ calls.
Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- KS algorithm: $O(n^7D^4)$ calls, $n$ number of variables and $D$ the total degree
- Question: is it possible to decrease the number of calls to a more manageable polynomial.
- Yes for polynomial of fixed degree $d$. One can find the ”highest“ degree polynomial with $O(n^2D^{d-1})$ calls.
- Yes for polynomial whose each two monomials have distinct supports: $O(n^2)$ calls.

**Open question:** how to efficiently represent and compute the partial polynomial at each step? Easier with circuits, formulas, polynomials of low degree, over fixed finite fields?
Finding one monomial

**Aim:** reducing the number of calls to the black-box at each step.

- **KS algorithm:** \(O(n^7D^4)\) calls, \(n\) number of variables and \(D\) the total degree
- **Question:** is it possible to decrease the number of calls to a more manageable polynomial.
- **Yes for polynomial of fixed degree** \(d\). One can find the ”highest“ degree polynomial with \(O(n^2D^{d-1})\) calls.
- **Yes for polynomial whose each two monomials have distinct supports:** \(O(n^2)\) calls.

**Open question:** how to efficiently represent and compute the partial polynomial at each step? Easier with circuits, formulas, polynomials of low degree, over fixed finite fields?
Improving the delay

How to achieve a polynomial delay?

We want to determine the degree of a subset $S$ of variables of the polynomial.
Improving the delay

How to achieve a polynomial delay?

We want to determine the degree of a subset $S$ of variables of the polynomial.

1. pick random values for variables outside of $S$ and look at the remaining polynomial as an univariate one, interpolate it to get its degree
Improving the delay

How to achieve a polynomial delay?

We want to determine the degree of a subset $S$ of variables of the polynomial.

1. pick random values for variables outside of $S$ and look at the remaining polynomial as an univariate one, interpolate it to get its degree

2. evaluate the polynomial on a large value for the variables of $S$ and small random values for the others
Improving the delay

How to achieve a polynomial delay?

We want to determine the degree of a subset $S$ of variables of the polynomial.

1. pick random values for variables outside of $S$ and look at the remaining polynomial as an univariate one, interpolate it to get its degree
2. evaluate the polynomial on a large value for the variables of $S$ and small random values for the others
3. if the polynomial is given by a circuit, transform it into its homogeneous components with regard to $S$
Improving the delay

How to achieve a polynomial delay?

We want to determine the degree of a subset $S$ of variables of the polynomial.

1. pick random values for variables outside of $S$ and look at the remaining polynomial as an univariate one, interpolate it to get its degree
2. evaluate the polynomial on a large value for the variables of $S$ and small random values for the others
3. if the polynomial is given by a circuit, transform it into its homogeneous components with regard to $S$

These algorithms are randomized (again the error is exponentially small) and in polynomial time in the number of variables.
How to achieve a polynomial delay?

We want to determine the degree of a subset $S$ of variables of the polynomial.

1. pick random values for variables outside of $S$ and look at the remaining polynomial as an univariate one, interpolate it to get its degree

2. evaluate the polynomial on a large value for the variables of $S$ and small random values for the others

3. if the polynomial is given by a circuit, transform it into its homogeneous components with regard to $S$

These algorithms are randomized (again the error is exponentially small) and in polynomial time in the number of variables.
**Partial-Monomial**

*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$

*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

When the polynomial is *multilinear*, this problem can be solved by finding the degree of $P_{\bar{L}_1}$ with regard to $L_2$: test if the degree is equal to $|L_2|$. 
Partial-Monomial

*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$

*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

When the polynomial is **multilinear**, this problem can be solved by finding the degree of $P_{L_1}$ with regard to $L_2$: test if the degree is equal to $|L_2|$.

Use this procedure for a depth first traversal of a tree whose leaves are the monomials.
**Partial-Monomial**

*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$

*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

When the polynomial is **multilinear**, this problem can be solved by finding the degree of $P_{\overline{L_1}}$ with regard to $L_2$: test if the degree is equal to $|L_2|$.

Use this procedure for a depth first traversal of a tree whose leaves are the monomials.
\[ P(X_1, X_2, X_3) = 2X_1 - X_2 X_3 + X_1 X_3 + 5X_2 \]
Theorem

Let $P$ be a multilinear polynomial with $n$ variables and a total degree $D$. There is an algorithm which computes the set of monomials of $P$ with probability $1 - \epsilon$ and a delay polynomial in $n$, $D$ and $\log(\epsilon)^{-1}$.

- The algorithm can be parallelized.
Polynomial delay algorithm

**Theorem**

Let $P$ be a multilinear polynomial with $n$ variables and a total degree $D$. There is an algorithm which computes the set of monomials of $P$ with probability $1 - \epsilon$ and a delay polynomial in $n$, $D$ and $\log(\epsilon)^{-1}$.

- The algorithm can be parallelized.
- It works on finite fields of small characteristic (can be used to speed up computation).
Theorem

Let $P$ be a multilinear polynomial with $n$ variables and a total degree $D$. There is an algorithm which computes the set of monomials of $P$ with probability $1 - \epsilon$ and a delay polynomial in $n$, $D$ and $\log(\epsilon)^{-1}$.

- The algorithm can be parallelized.
- It works on finite fields of small characteristic (can be used to speed up computation).
- On classes of polynomials given by circuits on which PIT can be derandomized, this algorithm also can be derandomized.

STOC 2011, Saraf, Volkovich: deterministic identity testing of depth-4 multilinear circuits with bounded top fan-in
Theorem

Let $P$ be a multilinear polynomial with $n$ variables and a total degree $D$. There is an algorithm which computes the set of monomials of $P$ with probability $1 - \epsilon$ and a delay polynomial in $n, D$ and $\log(\epsilon)^{-1}$.

- The algorithm can be parallelized.
- It works on finite fields of small characteristic (can be used to speed up computation).
- On classes of polynomials given by circuits on which PIT can be derandomized, this algorithm also can be derandomized.

STOC 2011, Saraf, Volkovich: deterministic identity testing of depth-4 multilinear circuits with bounded top fan-in
Comparison to other algorithms

<table>
<thead>
<tr>
<th>Algorithm type</th>
<th>Ben-Or Tiwari</th>
<th>Zippel</th>
<th>KS</th>
<th>My Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm type</td>
<td>Deterministic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>Number of calls</td>
<td>$2T$</td>
<td>$tnD$</td>
<td>$tn^7D^4$</td>
<td>$tnD(n + \log(e^{-1}))$</td>
</tr>
<tr>
<td>Total time</td>
<td>Quadratic in $T$</td>
<td>Quadratic in $t$</td>
<td>Quadratic in $t$</td>
<td>Linear in $t$</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Exponential</td>
<td>TotalPP</td>
<td>IncPP</td>
<td>DelayPP</td>
</tr>
<tr>
<td>Size of points</td>
<td>$T \log(n)$</td>
<td>$\log(nT^2e^{-1})$</td>
<td>$\log(nDe^{-1})$</td>
<td>$\log(D)$</td>
</tr>
</tbody>
</table>

Figure: Comparison of interpolation algorithms on multilinear polynomials

Good total time and best delay, but only on multilinear polynomials.
Comparison to other algorithms

<table>
<thead>
<tr>
<th>Algorithm type</th>
<th>Ben-Or Tiwari</th>
<th>Zippel</th>
<th>KS</th>
<th>My Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm type</td>
<td>Deterministic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>Number of calls</td>
<td>$2T$</td>
<td>$tnD$</td>
<td>$tn^7D^4$</td>
<td>$tnD(n + \log(e^{-1}))$</td>
</tr>
<tr>
<td>Total time</td>
<td>Quadratic in $T$</td>
<td>Quadratic in $t$</td>
<td>Quadratic in $t$</td>
<td>Linear in $t$</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Exponential</td>
<td>TotalPP</td>
<td>IncPP</td>
<td>DelayPP</td>
</tr>
<tr>
<td>Size of points</td>
<td>$T \log(n)$</td>
<td>$\log(nT^2e^{-1})$</td>
<td>$\log(nDe^{-1})$</td>
<td>$\log(D)$</td>
</tr>
</tbody>
</table>

Figure: Comparison of interpolation algorithms on multilinear polynomials

Good total time and best delay, but only on multilinear polynomials.
Limits to efficient interpolation

**Strategy**: relate the enumeration problem to some decision problem.

**Partial-Monomial**
*Input*: a polynomial given as a black box and two sets of variables $L_1$ and $L_2$
*Output*: accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.
Limits to efficient interpolation

**Strategy:** relate the enumeration problem to some decision problem.

**Partial-Monomial**

*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$

*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

The polynomial delay algorithm works by repeatedly solving this problem.
Limits to efficient interpolation

**Strategy:** relate the enumeration problem to some decision problem.

**Partial-Monomial**

*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$

*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

The polynomial delay algorithm works by repeatedly solving this problem.

**Proposition**

The problem Partial-Monomial restricted to degree 2 polynomials is NP-hard.
Limits to efficient interpolation

**Strategy:** relate the enumeration problem to some decision problem.

**Partial-Monomial**

*Input:* a polynomial given as a black box and two sets of variables $L_1$ and $L_2$

*Output:* accept if there is a monomial in the polynomial in which no variables of $L_1$ appear, but all of those of $L_2$ do.

The polynomial delay algorithm works by repeatedly solving this problem.

**Proposition**

*The problem Partial-Monomial restricted to degree 2 polynomials is NP-hard.*
Thanks!
Thanks!
Thanks,
Thanks!
Thanks, thanks,
Thanks!
Thanks, thanks, thanks,
Thanks!
Thanks, thanks, thanks, thanks,
Thanks!
Thanks, thanks, thanks, thanks, thanks, thanks,
Thanks!
Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks,
Thanks!

Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks,
Thanks!
Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks,
Thanks!
Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks,
Thanks!
Thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks, thanks
Thanks!
Thanks, thanks, thanks, thanks, thanks,
thanks, thanks, thanks, thanks, thanks,
thanks, thanks
Let’s all do enumeration