Enumeration: logic and algebric methods

Yann Strozecki

Université Paris Diderot - Paris 7

Dept. of Computer Science, University of Toronto

Avril 2011, Bordeaux
Séminaire graphe et logique

Enumeration problems

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

- $\exists ? y A(x, y)$: decision problem (class NP)

Enumeration problems

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

- $\exists ? y A(x, y)$: decision problem (class NP)

Example

The predicate $A(x, y)$ means y is a perfect matching in the graph x.

- The decision problem is to decide if there is a perfect matching.

Enumeration problems

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

- $\exists ? y A(x, y)$: decision problem (class NP)
- $\sharp\{y | A(x, y)\}$: counting problem (class $\sharp\mathrm{P}$)

Example

The predicate $A(x, y)$ means y is a perfect matching in the graph x.

- The decision problem is to decide if there is a perfect matching.

Enumeration problems

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

- $\exists ? y A(x, y)$: decision problem (class NP)
- $\sharp\{y | A(x, y)\}$: counting problem (class $\sharp\mathrm{P}$)

Example

The predicate $A(x, y)$ means y is a perfect matching in the graph x.

- The decision problem is to decide if there is a perfect matching.
- The counting problem is to count the number of perfect matchings.

Enumeration problems

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

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

Example

The predicate $A(x, y)$ means y is a perfect matching in the graph x.

- The decision problem is to decide if there is a perfect matching.
- The counting problem is to count the number of perfect matchings.

Enumeration problems

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

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

Example

The predicate $A(x, y)$ means y is a perfect matching in the graph x.

- 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 find every perfect matching.

Enumeration problems

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

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

Example

The predicate $A(x, y)$ means y is a perfect matching in the graph x.

- 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 find every perfect matching.

Complexity measures for enumeration

For enumeration problems we have two interesting complexity measures:

1. the total time related to the number of solutions

Complexity measures for enumeration

For enumeration problems we have two interesting complexity measures:

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

Complexity measures for enumeration

For enumeration problems we have two interesting complexity measures:

1. the total time related to the number of solutions
   - polynomial total time: $\mathbf{TotalP}$
2. the delay

Complexity measures for enumeration

For enumeration problems we have two interesting complexity measures:

1. the total time related to the number of solutions
   - polynomial total time: $\mathbf{TotalP}$
2. the delay
   - incremental polynomial time: $\mathbf{IncP}$

Complexity measures for enumeration

For enumeration problems we have two interesting complexity measures:

1. the total time related to the number of solutions
   - polynomial total time: $\mathbf{TotalP}$
2. the delay
   - incremental polynomial time: $\mathbf{IncP}$
   - polynomial delay: $\mathbf{DelayP}$ (Perfect Matching [Uno])

Complexity measures for enumeration

For enumeration problems we have two interesting complexity measures:

1. the total time related to the number of solutions
   - polynomial total time: **TotalP**
2. the delay
   - incremental polynomial time: **IncP**
   - polynomial delay: **DelayP** (Perfect Matching [Uno])
   - precomputation and constant or linear delay

Complexity measures for enumeration

For enumeration problems we have two interesting complexity measures:

1. the total time related to the number of solutions
   - polynomial total time: $\mathbf{TotalP}$
2. the delay
   - incremental polynomial time: $\mathbf{IncP}$
   - polynomial delay: $\mathbf{DelayP}$ (Perfect Matching [Uno])
   - precomputation and constant or linear delay

Open question: is $\mathbf{DelayP} \neq \mathbf{IncP}$ modulo some complexity hypothesis ?

Complexity measures for enumeration

For enumeration problems we have two interesting complexity measures:

1. the total time related to the number of solutions
   - polynomial total time: $\mathbf{TotalP}$
2. the delay
   - incremental polynomial time: $\mathbf{IncP}$
   - polynomial delay: $\mathbf{DelayP}$ (Perfect Matching [Uno])
   - precomputation and constant or linear delay

Open question: is $\mathbf{DelayP} \neq \mathbf{IncP}$ modulo some complexity hypothesis ?

Enumeration problem defined by a formula

Let $\mathscr{F}$ be a subclass of first order formulas and let $\Phi(\mathbf{z}, \mathbf{T}) \in \mathscr{F}$.

W.l.o.g. the tuple $\mathbf{T}$ contains only one relation T of arity $r = \max_{i \leq h} \mathrm{ar}(T_i) + 1$

Enumeration problem defined by a formula

Let $\mathscr{F}$ be a subclass of first order formulas and let $\Phi(\mathbf{z}, \mathbf{T}) \in \mathscr{F}$.

W.l.o.g. the tuple $\mathbf{T}$ contains only one relation T of arity $r = \max_{i \leq h} \mathrm{ar}(T_i) + 1$

$\textsc{Enum}{\cdot}\Phi$

Input: A σ-structure $\mathcal{S}$

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

Enumeration problem defined by a formula

Let $\mathscr{F}$ be a subclass of first order formulas and let $\Phi(\mathbf{z}, \mathbf{T}) \in \mathscr{F}$.

W.l.o.g. the tuple $\mathbf{T}$ contains only one relation T of arity $r = \max_{i \leq h} \mathrm{ar}(T_i) + 1$

$\textsc{Enum}{\cdot}\Phi$

Input: A σ-structure $\mathcal{S}$

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

Definition

We denote by $\textsc{Enum}{\cdot}\mathscr{F}$ the collection of problems $\textsc{Enum}{\cdot}\Phi$ for $\Phi \in \mathscr{F}$.

Enumeration problem defined by a formula

Let $\mathscr{F}$ be a subclass of first order formulas and let $\Phi(\mathbf{z}, \mathbf{T}) \in \mathscr{F}$.

W.l.o.g. the tuple $\mathbf{T}$ contains only one relation T of arity $r = \max_{i \leq h} \mathrm{ar}(T_i) + 1$

$\textsc{Enum}{\cdot}\Phi$

Input: A σ-structure $\mathcal{S}$

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

Definition

We denote by $\textsc{Enum}{\cdot}\mathscr{F}$ the collection of problems $\textsc{Enum}{\cdot}\Phi$ for $\Phi \in \mathscr{F}$.

Aim: Study of $\textsc{Enum}{\cdot}\mathscr{F}$, where $\mathscr{F}$ is defined by quantifier alternations, e.g. $\textsc{Enum}{\cdot}\Pi_2$.

Enumeration problem defined by a formula

Let $\mathscr{F}$ be a subclass of first order formulas and let $\Phi(\mathbf{z}, \mathbf{T}) \in \mathscr{F}$.

W.l.o.g. the tuple $\mathbf{T}$ contains only one relation T of arity $r = \max_{i \leq h} \mathrm{ar}(T_i) + 1$

$\textsc{Enum}{\cdot}\Phi$

Input: A σ-structure $\mathcal{S}$

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

Definition

We denote by $\textsc{Enum}{\cdot}\mathscr{F}$ the collection of problems $\textsc{Enum}{\cdot}\Phi$ for $\Phi \in \mathscr{F}$.

Aim: Study of $\textsc{Enum}{\cdot}\mathscr{F}$, where $\mathscr{F}$ is defined by quantifier alternations, e.g. $\textsc{Enum}{\cdot}\Pi_2$.

Example

Example

A formula for independent sets:

$$IS(T) \equiv \forall x \forall y \; T(x) \wedge T(y) \Rightarrow \neg E(x, y).$$

The formula is in Π_1, thus $\textsc{Enum}{\cdot}\mathrm{IS} \in \textsc{Enum}{\cdot}\Pi_1$.

Example

The hitting sets (vertex covers) of an hypergraph. An hypergraph H is represented by the incidence structure $\langle D, \{V, E, R\}\rangle$, D is partitioned into V (vertices) and E (edges), R is the incidence relation.

$$HS(T) \equiv \forall x\,(T(x) \Rightarrow V(x)) \wedge \forall y \exists x\, E(y) \Rightarrow (T(x) \wedge R(x, y))$$

The problem $\textsc{Enum}{\cdot}\mathrm{HS} \in \textsc{Enum}{\cdot}\Pi_2$.

Example

Example

A formula for independent sets:

$$IS(T) \equiv \forall x \forall y \; T(x) \wedge T(y) \Rightarrow \neg E(x,y).$$

The formula is in Π_1, thus $\textsc{Enum}{\cdot}\mathrm{IS} \in \textsc{Enum}{\cdot}\Pi_1$.

Example

The hitting sets (vertex covers) of an hypergraph. An hypergraph H is represented by the incidence structure $\langle D, \{V, E, R\}\rangle$, D is partitioned into V (vertices) and E (edges), R is the incidence relation.

$$HS(T) \equiv \forall x \, (T(x) \Rightarrow V(x)) \wedge \forall y \exists x \, E(y) \Rightarrow (T(x) \wedge R(x,y))$$

The problem $\textsc{Enum}{\cdot}\mathrm{HS} \in \textsc{Enum}{\cdot}\Pi_2$.

Previous results

Theorem (Durand, Grandjean 2007)

Let $\varphi(\mathbf{x})$ be a formula of the first order logic over bounded-degree structures. Then $\textsc{Enum}{\cdot}\varphi$ can be enumerated after a linear preprocessing with constant delay.

Theorem (Courcelle 2009)

Let $\varphi(\mathbf{x}, \mathbf{T})$ be a formula of the monadic second order logic over trees, or relational structure of tree-width at most k. Then $\textsc{Enum}{\cdot}\varphi$ can be enumerated after a preprocessing that takes time $O(n.log(n))$ and with delay $O(n)$, where n is the number of vertices or elements.

Previous results

Theorem (Durand, Grandjean 2007)

Let $\varphi(\mathbf{x})$ *be a formula of the first order logic over bounded-degree structures. Then* $\textsc{Enum}{\cdot}\varphi$ *can be enumerated after a linear preprocessing with constant delay.*

Theorem (Courcelle 2009)

Let $\varphi(\mathbf{x}, \mathbf{T})$ *be a formula of the monadic second order logic over trees, or relational structure of tree-width at most* k*. Then* $\textsc{Enum}{\cdot}\varphi$ *can be enumerated after a preprocessing that takes time* $O(n.log(n))$ *and with delay* $O(n)$*, where* n *is the number of vertices or elements.*

A hierarchy of quantifier alternations

From a formula $\Phi(\mathbf{z}, \mathbf{T})$, one defines the counting function $\#\Phi$ which to a model $\mathcal{S}$ associates $|\Phi(\mathcal{S})|$.

Theorem (Saluja, Thakur 1995)

On linearly ordered structures, we have the following inclusions:
$\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2$.

A hierarchy of quantifier alternations

From a formula $\Phi(\mathbf{z}, \mathbf{T})$, one defines the counting function $\#\Phi$ which to a model $\mathcal{S}$ associates $|\Phi(\mathcal{S})|$.

Theorem (Saluja, Thakur 1995)

On linearly ordered structures, we have the following inclusions: $\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2$.

Corollary

On linearly ordered structures, we have the following inclusions: $\textsc{Enum}{\cdot}\Sigma_0 \subsetneq \textsc{Enum}{\cdot}\Sigma_1 \subsetneq \textsc{Enum}{\cdot}\Pi_1 \subsetneq \textsc{Enum}{\cdot}\Sigma_2 \subsetneq \textsc{Enum}{\cdot}\Pi_2$.

A hierarchy of quantifier alternations

From a formula $\Phi(\mathbf{z}, \mathbf{T})$, one defines the counting function $\#\Phi$ which to a model $\mathcal{S}$ associates $|\Phi(\mathcal{S})|$.

Theorem (Saluja, Thakur 1995)

On linearly ordered structures, we have the following inclusions: $\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2$.

Corollary

On linearly ordered structures, we have the following inclusions: $\textsc{Enum}{\cdot}\Sigma_0 \subsetneq \textsc{Enum}{\cdot}\Sigma_1 \subsetneq \textsc{Enum}{\cdot}\Pi_1 \subsetneq \textsc{Enum}{\cdot}\Sigma_2 \subsetneq \textsc{Enum}{\cdot}\Pi_2$.

Other levels are not interesting: $\textsc{Enum}{\cdot}\Pi_2$ is complete under parsimonious reductions.

A hierarchy of quantifier alternations

From a formula $\Phi(\mathbf{z}, \mathbf{T})$, one defines the counting function $\#\Phi$ which to a model $\mathcal{S}$ associates $|\Phi(\mathcal{S})|$.

Theorem (Saluja, Thakur 1995)

On linearly ordered structures, we have the following inclusions: $\#\Sigma_0 \subsetneq \#\Sigma_1 \subsetneq \#\Pi_1 \subsetneq \#\Sigma_2 \subsetneq \#\Pi_2$.

Corollary

On linearly ordered structures, we have the following inclusions: $\textsc{Enum}{\cdot}\Sigma_0 \subsetneq \textsc{Enum}{\cdot}\Sigma_1 \subsetneq \textsc{Enum}{\cdot}\Pi_1 \subsetneq \textsc{Enum}{\cdot}\Sigma_2 \subsetneq \textsc{Enum}{\cdot}\Pi_2$.

Other levels are not interesting: $\textsc{Enum}{\cdot}\Pi_2$ is complete under parsimonious reductions.

From first order to propositional logic

- a first order formula $\Phi(\mathbf{z}, T)$, $|z| = k$ and $ar(T) = r$.
- a model $\mathcal{S}$ of domain D (size n), an enumeration $\mathbf{z}_i$ of the elements of D^k

1. Replace $\Phi(\mathbf{z}, T)$ by $\bigvee_{i=0}^{n^k-1} \Phi(\mathbf{z}_i, T)$

From first order to propositional logic

- a first order formula $\Phi(\mathbf{z}, T)$, $|z| = k$ and $ar(T) = r$.
- a model $\mathcal{S}$ of domain D (size n), an enumeration $\mathbf{z}_i$ of the elements of D^k

1. Replace $\Phi(\mathbf{z}, T)$ by $\bigvee_{i=0}^{n^k-1} \Phi(\mathbf{z}_i, T)$
2. In $\Phi(\mathbf{z}_i, T)$, inductively replace each sub-formula $\exists \mathbf{y} \varphi(\mathbf{z}_i, \mathbf{y}, T)$ by $\bigvee_{j=0}^{n^p-1} \varphi(\mathbf{z}_i, \mathbf{y}_j, T)$ with $|\mathbf{y}| = p$

From first order to propositional logic

- a first order formula $\Phi(\mathbf{z}, T)$, $|z| = k$ and $ar(T) = r$.
- a model $\mathcal{S}$ of domain D (size n), an enumeration $\mathbf{z}_i$ of the elements of D^k

1. Replace $\Phi(\mathbf{z}, T)$ by $\bigvee_{i=0}^{n^k-1} \Phi(\mathbf{z}_i, T)$
2. In $\Phi(\mathbf{z}_i, T)$, inductively replace each sub-formula $\exists \mathbf{y} \varphi(\mathbf{z}_i, \mathbf{y}, T)$ by $\bigvee_{j=0}^{n^p-1} \varphi(\mathbf{z}_i, \mathbf{y}_j, T)$ with $|\mathbf{y}| = p$
3. Replace universal quantification by conjunction

From first order to propositional logic

- a first order formula $\Phi(\mathbf{z}, T)$, $|z| = k$ and $ar(T) = r$.
- a model $\mathcal{S}$ of domain D (size n), an enumeration $\mathbf{z}_i$ of the elements of D^k

1. Replace $\Phi(\mathbf{z}, T)$ by $\bigvee_{i=0}^{n^k-1} \Phi(\mathbf{z}_i, T)$
2. In $\Phi(\mathbf{z}_i, T)$, inductively replace each sub-formula $\exists \mathbf{y} \varphi(\mathbf{z}_i, \mathbf{y}, T)$ by $\bigvee_{j=0}^{n^p-1} \varphi(\mathbf{z}_i, \mathbf{y}_j, T)$ with $|\mathbf{y}| = p$
3. Replace universal quantification by conjunction
4. Replace every atomic formula $R(\mathbf{w})$ with $R \in \sigma$ by its truth value in $\mathcal{S}$

From first order to propositional logic

- a first order formula $\Phi(\mathbf{z}, T)$, $|z| = k$ and $ar(T) = r$.
- a model $\mathcal{S}$ of domain D (size n), an enumeration $\mathbf{z}_i$ of the elements of D^k

1. Replace $\Phi(\mathbf{z}, T)$ by $\bigvee_{i=0}^{n^k-1} \Phi(\mathbf{z}_i, T)$
2. In $\Phi(\mathbf{z}_i, T)$, inductively replace each sub-formula $\exists \mathbf{y} \varphi(\mathbf{z}_i, \mathbf{y}, T)$ by $\bigvee_{j=0}^{n^p-1} \varphi(\mathbf{z}_i, \mathbf{y}_j, T)$ with $|\mathbf{y}| = p$
3. Replace universal quantification by conjunction
4. Replace every atomic formula $R(\mathbf{w})$ with $R \in \sigma$ by its truth value in $\mathcal{S}$

A disjunction of propositional formulas $\tilde{\Phi}_i$, with variables $T(\mathbf{w})$ where $\mathbf{w} \in D^r$.

From first order to propositional logic

- a first order formula $\Phi(\mathbf{z}, T)$, $|z| = k$ and $ar(T) = r$.
- a model $\mathcal{S}$ of domain D (size n), an enumeration $\mathbf{z}_i$ of the elements of D^k

1. Replace $\Phi(\mathbf{z}, T)$ by $\bigvee_{i=0}^{n^k-1} \Phi(\mathbf{z}_i, T)$
2. In $\Phi(\mathbf{z}_i, T)$, inductively replace each sub-formula $\exists \mathbf{y} \varphi(\mathbf{z}_i, \mathbf{y}, T)$ by $\bigvee_{j=0}^{n^p-1} \varphi(\mathbf{z}_i, \mathbf{y}_j, T)$ with $|\mathbf{y}| = p$
3. Replace universal quantification by conjunction
4. Replace every atomic formula $R(\mathbf{w})$ with $R \in \sigma$ by its truth value in $\mathcal{S}$

A disjunction of propositional formulas $\tilde{\Phi}_i$, with variables $T(\mathbf{w})$ where $\mathbf{w} \in D^r$.

The first level: Enum$\cdot\Sigma_0$

Theorem

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

Idea of proof:

Remark that each $\tilde{\Phi}_i$ is of constant size.

The first level: Enum$\cdot\Sigma_0$

Theorem

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

Idea of proof:
Remark that each $\tilde{\Phi}_i$ is of constant size.

- Compute the set of solutions of each $\tilde{\Phi}_i$ which are satisfiable (preprocessing).

The first level: Enum$\cdot\Sigma_0$

Theorem

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

Idea of proof:

Remark that each $\tilde{\Phi}_i$ is of constant size.

- Compute the set of solutions of each $\tilde{\Phi}_i$ which are satisfiable (preprocessing).
- A solution of $\tilde{\Phi}_i$ gives a T^* such that $\Phi(\mathbf{z}_i, T^*)$ holds.

The first level: Enum$\cdot\Sigma_0$

Theorem

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

Idea of proof:

Remark that each $\tilde{\Phi}_i$ is of constant size.

- Compute the set of solutions of each $\tilde{\Phi}_i$ which are satisfiable (preprocessing).
- A solution of $\tilde{\Phi}_i$ gives a T^* such that $\Phi(\mathbf{z}_i, T^*)$ holds.
- Enumerate by Gray code all extensions of T (they are all solutions).

The first level: Enum$\cdot\Sigma_0$

Theorem

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

Idea of proof:

Remark that each $\tilde{\Phi}_i$ is of constant size.

- Compute the set of solutions of each $\tilde{\Phi}_i$ which are satisfiable (preprocessing).
- A solution of $\tilde{\Phi}_i$ gives a T^* such that $\Phi(\mathbf{z}_i, T^*)$ holds.
- Enumerate by Gray code all extensions of T (they are all solutions).

Bounded degree structure

Not possible to improve the preprocessing: computing if there is a k-clique should depend on k.

Theorem

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

Bounded degree structure

Not possible to improve the preprocessing: computing if there is a k-clique should depend on k.

Theorem

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

Idea of proof:

- Do not replace each free variable by an element of the domain.

Bounded degree structure

Not possible to improve the preprocessing: computing if there is a k-clique should depend on k.

Theorem

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

Idea of proof:

- Do not replace each free variable by an element of the domain.
- An abstract formula whose variables are the atoms of Φ.

Bounded degree structure

Not possible to improve the preprocessing: computing if there is a k-clique should depend on k.

Theorem

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

Idea of proof:

- Do not replace each free variable by an element of the domain.
- An abstract formula whose variables are the atoms of Φ.
- From each solution, create a new Σ_0 formula without free second order variables.

Bounded degree structure

Not possible to improve the preprocessing: computing if there is a k-clique should depend on k.

Theorem

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

Idea of proof:

- Do not replace each free variable by an element of the domain.
- An abstract formula whose variables are the atoms of Φ.
- From each solution, create a new Σ_0 formula without free second order variables.
- Enumerate the solutions of this formula thanks to [DG 2007].

Bounded degree structure

Not possible to improve the preprocessing: computing if there is a k-clique should depend on k.

Theorem

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

Idea of proof:

- Do not replace each free variable by an element of the domain.
- An abstract formula whose variables are the atoms of Φ.
- From each solution, create a new Σ_0 formula without free second order variables.
- Enumerate the solutions of this formula thanks to [DG 2007].

Union: elimination of some existential quantifiers

Proposition

Let R and S be two polynomially balanced predicates such that S can be decided in time $O(h(n))$. Assume that one can solve $\textsc{Enum}\cdot R$ and $\textsc{Enum}\cdot S$ with preprocessing $f(n)$ and delay $g(n)$, then one can solve $\textsc{Enum}\cdot R \cup S$ with preprocessing $2f(n) + c$ and delay $2g(n) + h(n) + c$, where c is a constant.

Idea: run in parallel an algorithm for $\textsc{Enum}\cdot R$ and $\textsc{Enum}\cdot S$

Union: elimination of some existential quantifiers

Proposition

Let R and S be two polynomially balanced predicates such that S can be decided in time $O(h(n))$. Assume that one can solve $\textsc{Enum}\cdot R$ and $\textsc{Enum}\cdot S$ with preprocessing $f(n)$ and delay $g(n)$, then one can solve $\textsc{Enum}\cdot R \cup S$ with preprocessing $2f(n) + c$ and delay $2g(n) + h(n) + c$, where c is a constant.

Idea: run in parallel an algorithm for $\textsc{Enum}\cdot R$ and $\textsc{Enum}\cdot S$

Remark

This proposition allows to remove the first block of existential quantifier at the cost of a polynomial (the exponent being the size of the block) increase of the delay.

Union: elimination of some existential quantifiers

Proposition

Let R and S be two polynomially balanced predicates such that S can be decided in time $O(h(n))$. Assume that one can solve $\textsc{Enum}{\cdot}R$ *and* $\textsc{Enum}{\cdot}S$ *with preprocessing $f(n)$ and delay $g(n)$, then one can solve* $\textsc{Enum}{\cdot}R \cup S$ *with preprocessing $2f(n) + c$ and delay $2g(n) + h(n) + c$, where c is a constant.*

Idea: run in parallel an algorithm for $\textsc{Enum}{\cdot}R$ and $\textsc{Enum}{\cdot}S$

Remark

This proposition allows to remove the first block of existential quantifier at the cost of a polynomial (the exponent being the size of the block) increase of the delay.

Second level: Enum$\cdot\Sigma_1$

Theorem

$\textsc{Enum}\cdot\Sigma_1 \subseteq \textsc{DelayP}$. *More precisely,* $\textsc{Enum}\cdot\Sigma_1$ *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.*

- **Problem:** is it possible to improve the delay, such that is independent or less dependent on k?

Second level: Enum$\cdot\Sigma_1$

Theorem

$\textsc{Enum}\cdot\Sigma_1 \subseteq \textsc{Delay}\mathrm{P}$. *More precisely,* $\textsc{Enum}\cdot\Sigma_1$ *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.*

- **Problem:** is it possible to improve the delay, such that is independent or less dependent on k?
- Bounded degree improve the preprocessing but not the delay.

Second level: Enum$\cdot\Sigma_1$

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.*

- **Problem:** is it possible to improve the delay, such that is independent or less dependent on k?
- Bounded degree improve the preprocessing but not the delay.
- Somewhat equivalent to the enumeration of the models of a DNF formula with n variables and clauses of fixed size.

Second level: Enum$\cdot\Sigma_1$

Theorem

$\textsc{Enum}\cdot\Sigma_1 \subseteq \textsc{DelayP}$. *More precisely,* $\textsc{Enum}\cdot\Sigma_1$ *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.*

- **Problem:** is it possible to improve the delay, such that is independent or less dependent on k?
- Bounded degree improve the preprocessing but not the delay.
- Somewhat equivalent to the enumeration of the models of a DNF formula with n variables and clauses of fixed size.

Tractable fragments

Proposition

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

Proposition

The problem $\textsc{Enum}\cdot\mathrm{SAT}(\mathcal{C})$ *is in* $\textsc{Delay}\mathrm{P}$ *when* $\mathcal{C}$ *is one of the following classes: Horn formulas, anti-Horn formulas, affine formulas, bijunctive (2CNF) formulas*

Tractable fragments

Proposition

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

Proposition

The problem $\textsc{Enum}{\cdot}\mathrm{SAT}(\mathcal{C})$ *is in* $\textsc{Delay}\mathrm{P}$ *when* $\mathcal{C}$ *is one of the following classes: Horn formulas, anti-Horn formulas, affine formulas, bijunctive (2CNF) formulas*

Corollary

Let $\Phi(\mathbf{z}, T)$ *be a formula, such that, for all* σ *structures, all propositional formulas* $\tilde{\Phi}_i$ *are either Horn, anti-Horn, affine or bijunctive. Then* $\textsc{Enum}{\cdot}\Phi \subseteq \textsc{Delay}\mathrm{P}$.

Tractable fragments

Proposition

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

Proposition

The problem $\textsc{Enum}{\cdot}\mathrm{SAT}(\mathcal{C})$ *is in* $\textsc{Delay}\mathrm{P}$ *when* $\mathcal{C}$ *is one of the following classes: Horn formulas, anti-Horn formulas, affine formulas, bijunctive (2CNF) formulas*

Corollary

Let $\Phi(\mathbf{z}, T)$ *be a formula, such that, for all* σ *structures, all propositional formulas* $\tilde{\Phi}_i$ *are either Horn, anti-Horn, affine or bijunctive. Then* $\textsc{Enum}{\cdot}\Phi \subseteq \textsc{Delay}\mathrm{P}$.

Example: independent sets and hitting sets.

Tractable fragments

Proposition

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

Proposition

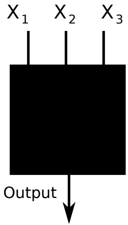
The problem $\textsc{Enum}{\cdot}\mathrm{SAT}(\mathcal{C})$ *is in* $\textsc{Delay}\mathrm{P}$ *when* $\mathcal{C}$ *is one of the following classes: Horn formulas, anti-Horn formulas, affine formulas, bijunctive (2CNF) formulas*

Corollary

Let $\Phi(\mathbf{z}, T)$ *be a formula, such that, for all* σ *structures, all propositional formulas* $\tilde{\Phi}_i$ *are either Horn, anti-Horn, affine or bijunctive. Then* $\textsc{Enum}{\cdot}\Phi \subseteq \textsc{Delay}\mathrm{P}$.

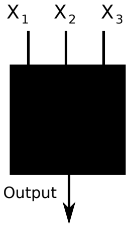
Example: independent sets and hitting sets.

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



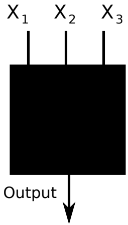
$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

$$X_1 = 1,\ X_2 = 2,\ X_3 = 1$$

$$1 * 2 + 1 * 1 + 2 + 1$$

$$Output = 6$$

Polynomial given by a black-box



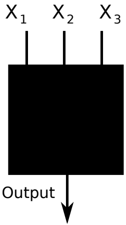
$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

$$X_1 = -1,\ X_2 = 1,\ X_3 = 2$$

$$-1 * 1 + -1 * 2 + 1 + 2$$

$$Output = 0$$

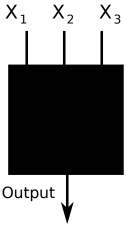
Polynomial given by a black-box



$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

- Problem: interpolation, compute P from its values.
- Parameters: number of variables and total degree.

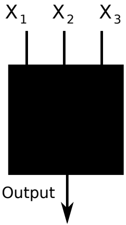
Polynomial given by a black-box



$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

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

Polynomial given by a black-box

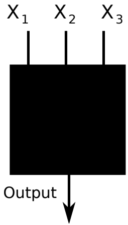


$$P(X_1, X_2, X_3) = X_1X_2 + X_1X_3 + X_2 + X_3$$

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

Enumeration problem: output the monomials one after the other.

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.
- Parameters: number of variables and total degree.
- Complexity: time and number of calls to the oracle.

Enumeration problem: output the monomials one after the other.

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**.

Motivation

Easy to evaluate polynomials whose monomials represent interesting combinatorial objects.

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

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 .

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.
- A polynomial given by the multiplication of several matrices which represents 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.
- A polynomial given by the multiplication of several matrices which represents 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.
- A polynomial given by the multiplication of several matrices which represents 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 $x_1, \dots, x_n$ are randomly chosen in a set of integers S of size $\frac{D}{\epsilon}$ then the probability that $P(x_1, \dots, x_n) = 0$ is bounded by ϵ.

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 ϵ.

No way to make PIT deterministic for black box.

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 ϵ.

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 ϵ.

No way to make PIT deterministic for black box.

Error **exponentially small** in the size of the integers!

From finding a monomial to interpolation

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.

From finding a monomial to interpolation

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 which is the sum of the generated monomials and to evaluate it at each step.

From finding a monomial to interpolation

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 which is the sum of the generated monomials and to evaluate it at each step.

Incremental time.

From finding a monomial to interpolation

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 which is the sum of the generated monomials and to evaluate it at each step.

Incremental time.

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

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
- 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: $O(n^7 D^4)$ calls, n number of variables and D
- 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 $O(n^2 D^{d-1})$ calls.

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
- 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 $O(n^2 D^{d-1})$ 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: $O(n^7 D^4)$ calls, n number of variables and D
- 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 $O(n^2 D^{d-1})$ 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 and the 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
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 and the degree.

Multilinear polynomials

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 $|L_2|$.

Multilinear polynomials

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 $|L_2|$.

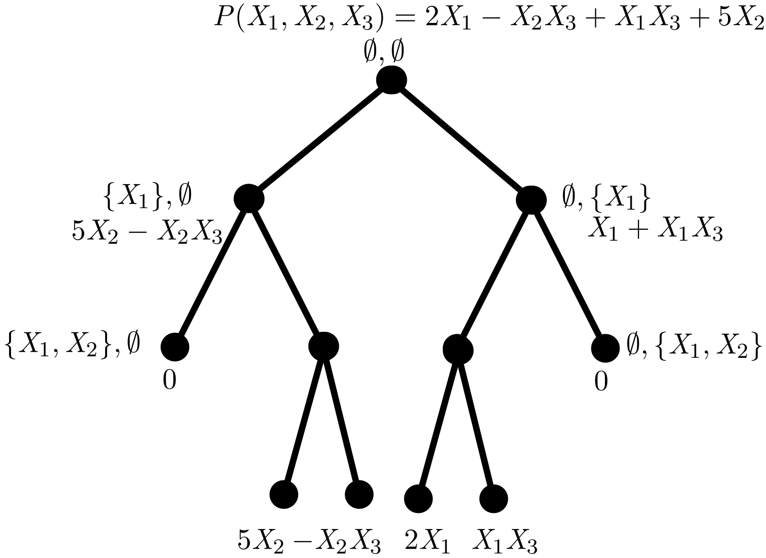
Use this procedure for a depth first traversal of a tree whose leaves are the monomials.

Multilinear polynomials

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 $|L_2|$.

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


$P(X_1, X_2, X_3) = 2X_1 - X_2X_3 + X_1X_3 + 5X_2$
$\emptyset, \emptyset$
$\{X_1\}, \emptyset$
$5X_2 - X_2X_3$
$\emptyset, \{X_1\}$
$X_1 + X_1X_3$
$\{X_1, X_2\}, \emptyset$
0
$\emptyset, \{X_1, X_2\}$
0
$5X_2$
$-X_2X_3$
$2X_1$
X_1X_3

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 $\log(\epsilon)^{-1}$.*

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.

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).

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).
- On classes of polynomials given by circuits on which PIT can be derandomized, this algorithm also can be derandomized. STOC 2010, Karnin, Mukhopadhyay, Shpilka, Volkovich: deterministic identity testing of depth-4 multilinear circuits with bounded top fan-in

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).
- On classes of polynomials given by circuits on which PIT can be derandomized, this algorithm also can be derandomized. STOC 2010, Karnin, Mukhopadhyay, Shpilka, Volkovich: deterministic identity testing of depth-4 multilinear circuits with bounded top fan-in

Comparison to other algorithms

	Ben-Or Tiwari	Zippel	KS	My Algorithm
Algorithm type	Deterministic	Probabilistic	Probabilistic	Probabilistic
Number of calls	$2T$	tnD	tn^7D^4	$tnD(n + \log(\epsilon^{-1}))$
Total time	Quadratic in T	Quadratic in t	Quadratic in t	Linear in t
Enumeration	Exponential	**TotalPP**	**IncPP**	**DelayPP**
Size of points	$T \log(n)$	$\log(nT^2\epsilon^{-1})$	$\log(nD\epsilon^{-1})$	$\log(D)$

Figure: Comparison of interpolation algorithms on multilinear polynomials

Good total time and best delay, but only on multilinear polynomials.

Comparison to other algorithms

	Ben-Or Tiwari	Zippel	KS	My Algorithm
Algorithm type	Deterministic	Probabilistic	Probabilistic	Probabilistic
Number of calls	$2T$	tnD	tn^7D^4	$tnD(n+\log(\epsilon^{-1}))$
Total time	Quadratic in T	Quadratic in t	Quadratic in t	Linear in t
Enumeration	Exponential	**TotalPP**	**IncPP**	**DelayPP**
Size of points	$T\log(n)$	$\log(nT^2\epsilon^{-1})$	$\log(nD\epsilon^{-1})$	$\log(D)$

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.

NON-ZERO-MONOMIAL
Input: a polynomial and a term $\vec{X}^{\vec{e}}$
Output: accept if $\vec{X}^{\vec{e}}$ has a coefficient different from zero in the polynomial

Limits to efficient interpolation

Strategy: relate the enumeration problem to some decision problem.

Non-Zero-Monomial
Input: a polynomial and a term $\vec{X}^{\vec{e}}$
Output: accept if $\vec{X}^{\vec{e}}$ has a coefficient different from zero in the polynomial

An algorithm similar to the polynomial delay one can solve this problem.

Limits to efficient interpolation

Strategy: relate the enumeration problem to some decision problem.

NON-ZERO-MONOMIAL
Input: a polynomial and a term $\vec{X}^{\vec{e}}$
Output: accept if $\vec{X}^{\vec{e}}$ has a coefficient different from zero in the polynomial

An algorithm similar to the polynomial delay one can solve this problem.

Idea: if they are hard for some family of easy to compute polynomials, the polynomial delay interpolation should also be hard

Limits to efficient interpolation

Strategy: relate the enumeration problem to some decision problem.

Non-Zero-Monomial
Input: a polynomial and a term $\vec{X}^{\vec{e}}$
Output: accept if $\vec{X}^{\vec{e}}$ has a coefficient different from zero in the polynomial

An algorithm similar to the polynomial delay one can solve this problem.

Idea: if they are hard for some family of easy to compute polynomials, the polynomial delay interpolation should also be hard

Degree 2 polynomial

Proposition

The problem NON-ZERO-MONOMIAL *restricted to degree* 2 *polynomials is* NP*-hard.*

Proof.

Reduction from HAMILTONIAN PATH over degree 2 directed graphs. Use a polynomial derived from the Matrix Tree theorem. Use NON-ZERO-MONOMIAL on a polynomial number of terms of this polynomial, if one is in there is a spanning tree which is also an Hamiltonian path. $\square$

Degree 2 polynomial

Proposition

The problem NON-ZERO-MONOMIAL *restricted to degree* 2 *polynomials is* NP*-hard.*

Proof.

Reduction from HAMILTONIAN PATH over degree 2 directed graphs. Use a polynomial derived from the Matrix Tree theorem. Use NON-ZERO-MONOMIAL on a polynomial number of terms of this polynomial, if one is in there is a spanning tree which is also an Hamiltonian path. $\square$

- Conclusion: some monomials are harder than others.

Degree 2 polynomial

Proposition

The problem Non-Zero-Monomial *restricted to degree* 2 *polynomials is* NP*-hard.*

Proof.

Reduction from Hamiltonian Path over degree 2 directed graphs. Use a polynomial derived from the Matrix Tree theorem. Use Non-Zero-Monomial on a polynomial number of terms of this polynomial, if one is in there is a spanning tree which is also an Hamiltonian path. □

- Conclusion: some monomials are harder than others.
- Question of Kayal: what is the complexity of computing the leading monomial of a depth three circuit?

Degree 2 polynomial

Proposition

The problem NON-ZERO-MONOMIAL *restricted to degree* 2 *polynomials is* NP*-hard.*

Proof.

Reduction from HAMILTONIAN PATH over degree 2 directed graphs. Use a polynomial derived from the Matrix Tree theorem. Use NON-ZERO-MONOMIAL on a polynomial number of terms of this polynomial, if one is in there is a spanning tree which is also an Hamiltonian path. $\square$

- Conclusion: some monomials are harder than others.
- Question of Kayal: what is the complexity of computing the leading monomial of a depth three circuit?

Thank for listening!