Factoring, Root finding, and several other things

Amit Sinhababu

Chennai Mathematical Institute

WACT 2023, Warwick

Outline of the Talk

- Multivariate Polynomial Factoring
  - Background and Motivation.
  - Factoring Algebraic Branching Programs.
- Multivariate Factoring and PIT.

Outline of the Talk

- Multivariate Polynomial Factoring
  - Background and Motivation.
  - Factoring Algebraic Branching Programs.
- Multivariate Factoring and PIT.

Multivariate Polynomial Factoring: Background

- We encounter integer and polynomial factoring in school.
- Polynomials can be factored in polynomial time.
- Factor $f(x) \in \mathbb{Q}[x]$ using LLL algorithm in deterministic polynomial time.
- Factor $f(x) \in \mathbb{F}_q[x]$ using Berlekamp's algorithm.

- We encounter integer and polynomial factoring in school.
- Polynomials can be factored in polynomial time.
- Factor $f(x) \in \mathbb{Q}[x]$ using LLL algorithm in deterministic polynomial time.
- Factor $f(x) \in \mathbb{F}_q[x]$ using Berlekamp's algorithm.

- Polynomials are often easier cases than integers.
- Squarefree: Test if a given integer or polynomial has a factor that repeats.
- For integers, no polynomial time algorithm for this is known.
- Derivatives rescue us in case of polynomials. Test if $f(x)$ and its derivative are relatively prime.

Polynomials vs Integers

- Polynomials are often easier cases than integers.
- Squarefree: Test if a given integer or polynomial has a factor that repeats.
- For integers, no polynomial time algorithm for this is known.
- Derivatives rescue us in case of polynomials. Test if $f(x)$ and its derivative are relatively prime.

Polynomials vs Integers

- Polynomials are often easier cases than integers.
- Squarefree: Test if a given integer or polynomial has a factor that repeats.
- For integers, no polynomial time algorithm for this is known.
- Derivatives rescue us in case of polynomials. Test if $f(x)$ and its derivative are relatively prime.

Polynomials vs Integers

- Polynomials are often easier cases than integers.
- Squarefree: Test if a given integer or polynomial has a factor that repeats.
- For integers, no polynomial time algorithm for this is known.
- Derivatives rescue us in case of polynomials. Test if $f(x)$ and its derivative are relatively prime.

- The focus of today's talk is multivariate polynomial factorization.
- Multivariate factoring can be reduced to univariate factoring.

- The focus of today's talk is multivariate polynomial factorization.
- Multivariate factoring can be reduced to univariate factoring.

- Suppose $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)h(x_1, \ldots, x_n)$.
- Degree of each variable in $f(x_1, \ldots, x_n)$ is $\leq d$.
- Apply Kronecker substitution $\phi : x_i \mapsto z^{D^{i-1}}$ where $D = d + 1$.
- Each monomial in f uniquely maps to a monomial in $\phi(f)$. Thus, we can invert the map ϕ.

- Suppose $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)h(x_1, \ldots, x_n)$.
- Degree of each variable in $f(x_1, \ldots, x_n)$ is $\leq d$.
- Apply Kronecker substitution $\phi : x_i \mapsto z^{D^{i-1}}$ where $D = d + 1$.
- Each monomial in f uniquely maps to a monomial in $\phi(f)$. Thus, we can invert the map ϕ.

- Suppose $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)h(x_1, \ldots, x_n)$.
- Degree of each variable in $f(x_1, \ldots, x_n)$ is $\leq d$.
- Apply Kronecker substitution $\phi : x_i \mapsto z^{D^{i-1}}$ where $D = d + 1$.
- Each monomial in f uniquely maps to a monomial in $\phi(f)$. Thus, we can invert the map ϕ.

- Suppose $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)h(x_1, \ldots, x_n)$.
- Degree of each variable in $f(x_1, \ldots, x_n)$ is $\leq d$.
- Apply Kronecker substitution $\phi : x_i \mapsto z^{D^{i-1}}$ where $D = d + 1$.
- Each monomial in f uniquely maps to a monomial in $\phi(f)$. Thus, we can invert the map ϕ.

- If $f = gh$, then $\phi(f) = \phi(g)\phi(h)$.
- Factorize $\phi(f)$ into univariate irreducible factors.
- Though g is irreducible, $\phi(g)$ may not be irreducible.
- Product of a subset of the factors of $\phi(f)$ would correspond to $\phi(g)$.
- Try all subsets. Apply inverse Kronecker and test divisibility.
- Time complexity: Exponential in degree in worst-case (even for bivariates).

- If $f = gh$, then $\phi(f) = \phi(g)\phi(h)$.
- Factorize $\phi(f)$ into univariate irreducible factors.
- Though g is irreducible, $\phi(g)$ may not be irreducible.
- Product of a subset of the factors of $\phi(f)$ would correspond to $\phi(g)$.
- Try all subsets. Apply inverse Kronecker and test divisibility.
- Time complexity: Exponential in degree in worst-case (even for bivariates).

- If $f = gh$, then $\phi(f) = \phi(g)\phi(h)$.
- Factorize $\phi(f)$ into univariate irreducible factors.
- Though g is irreducible, $\phi(g)$ may not be irreducible.
- Product of a subset of the factors of $\phi(f)$ would correspond to $\phi(g)$.
- Try all subsets. Apply inverse Kronecker and test divisibility.
- Time complexity: Exponential in degree in worst-case (even for bivariates).

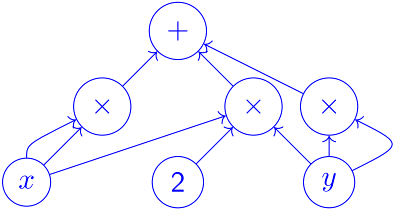
- If $f = gh$, then $\phi(f) = \phi(g)\phi(h)$.
- Factorize $\phi(f)$ into univariate irreducible factors.
- Though g is irreducible, $\phi(g)$ may not be irreducible.
- Product of a subset of the factors of $\phi(f)$ would correspond to $\phi(g)$.
- Try all subsets. Apply inverse Kronecker and test divisibility.
- Time complexity: Exponential in degree in worst-case (even for bivariates).

- If $f = gh$, then $\phi(f) = \phi(g)\phi(h)$.
- Factorize $\phi(f)$ into univariate irreducible factors.
- Though g is irreducible, $\phi(g)$ may not be irreducible.
- Product of a subset of the factors of $\phi(f)$ would correspond to $\phi(g)$.
- Try all subsets. Apply inverse Kronecker and test divisibility.
- Time complexity: Exponential in degree in worst-case (even for bivariates).

- Kaltofen (1982): Efficient reduction of bivariate to univariate factoring.
- Tools: Newton iteration/ Hensel lifting, Linear System Solving.
- We have to define the size of input and output polynomials in the multivariate setting to talk about time complexity.

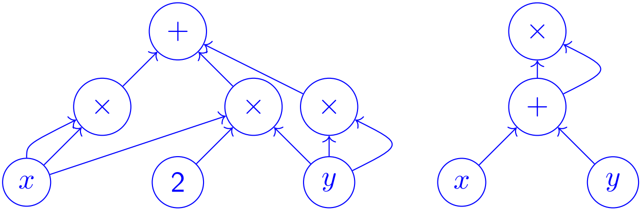
- Dense: List all the coefficients of $\binom{n+d}{d}$ many monomials up to degree d.
- Sparse: List only the monomials with nonzero coefficients. Eg. $x_1^2 + x_2 x_3 + 5x^4$.
- Formula: $(1 + x_1)(1 + x_2)x_3 - (1 + x_1)^2$. Reuse of computation not allowed. Structurally, looks like a tree.
- Straight-Line Programs or Arithmetic Circuits.

Two circuits for computing $x^2 + 2xy + y^2$



Size: Total number of nodes or edges.

Two circuits for computing $x^2 + 2xy + y^2$



Size: Total number of nodes or edges.

Complexity of Factors

Factorization of a polynomial

Let f be a polynomial of degree d that has size s in some model.

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{m} f_i^{e_i}$$

Let f_i's be its irreducible factors over $\mathbb{F}$.

Factor size bound question: Do all its factors have $\text{poly}(s, d)$ size in the same model?

FACTORIZATION OF A POLYNOMIAL

Let f be a polynomial of degree d that has size s in some model.

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{m} f_i^{e_i}$$

Let f_i's be its irreducible factors over $\mathbb{F}$.

FACTOR SIZE BOUND QUESTION: Do all its factors have $\text{POLY}(s, d)$ size in the same model?

- Let $\mathcal{C}$ be a class of polynomials.
- Closure under multiplication: $f, g \in \mathcal{C} \implies f \times g \in \mathcal{C}$.
- Closure under factoring: If $f \times g$ is in $\mathcal{C}$, are f, g also in $\mathcal{C}$?
- Apriori, it is not obvious. The smallest representation of fg may not be via computing f and g.

- Let $\mathcal{C}$ be a class of polynomials.
- Closure under multiplication: $f, g \in \mathcal{C} \implies f \times g \in \mathcal{C}$.
- Closure under factoring: If $f \times g$ is in $\mathcal{C}$, are f, g also in $\mathcal{C}$?
- Apriori, it is not obvious. The smallest representation of fg may not be via computing f and g.

- Factors can be larger in size. For example, $x^d - 1 = (x-1)(1 + x + \cdots + x^{d-1})$.

- Sparsity of factors can be superpolynomial wrt input polynomial's sparsity.

- Kaltofen 1986: If size denotes arithmetic circuit size, $g \mid f$ $\implies$ $\text{size}(g) \leq \textsc{poly}(\text{size}(f), \deg(f))$.

- Goal: Extend Kaltofen's result for factors of formulas, constant depth circuits, algebraic branching programs (ABPs), etc.

- Factors can be larger in size. For example, $x^d - 1 = (x-1)(1 + x + \cdots + x^{d-1})$.
- Sparsity of factors can be superpolynomial wrt input polynomial's sparsity.
- Kaltofen 1986: If size denotes arithmetic circuit size, $g \mid f \implies \mathsf{size}(g) \leq \textsc{poly}(\mathsf{size}(f), \deg(f))$.
- Goal: Extend Kaltofen's result for factors of formulas, constant depth circuits, algebraic branching programs (ABPs), etc.

- Factors can be larger in size. For example, $x^d - 1 = (x-1)(1+x+\cdots+x^{d-1})$.
- Sparsity of factors can be superpolynomial wrt input polynomial's sparsity.
- Kaltofen 1986: If size denotes arithmetic circuit size, $g \mid f \implies \mathsf{size}(g) \leq \textsc{poly}(\mathsf{size}(f), \deg(f))$.
- Goal: Extend Kaltofen's result for factors of formulas, constant depth circuits, algebraic branching programs (ABPs), etc.

- Multivariate Polynomial Factoring has applications in coding theory and various other problems.

- Helps to bridge two central questions in algebraic complexity: VP vs VNP and polynomial identity testing (PIT).

- Kabanets and Impagliazzo (2003): Exponential lower bound for arithmetic circuits $\implies$ Quasi-poly blackbox deterministic PIT for circuits.

- Hardness of multiples: If f is hard for $\mathcal{C}$, all its nonzero multiples are hard for $\mathcal{C}$.

Motivation

- Multivariate Polynomial Factoring has applications in coding theory and various other problems.

- Helps to bridge two central questions in algebraic complexity: VP vs VNP and polynomial identity testing (PIT).

- Kabanets and Impagliazzo (2003): Exponential lower bound for arithmetic circuits $\implies$ Quasi-poly blackbox deterministic PIT for circuits.

- Hardness of multiples: If f is hard for $\mathcal{C}$, all its nonzero multiples are hard for $\mathcal{C}$.

- Multivariate Polynomial Factoring has applications in coding theory and various other problems.
- Helps to bridge two central questions in algebraic complexity: VP vs VNP and polynomial identity testing (PIT).
- Kabanets and Impagliazzo (2003): Exponential lower bound for arithmetic circuits $\implies$ Quasi-poly blackbox deterministic PIT for circuits.
- Hardness of multiples: If f is hard for $\mathcal{C}$, all its nonzero multiples are hard for $\mathcal{C}$.

- Multivariate Polynomial Factoring has applications in coding theory and various other problems.
- Helps to bridge two central questions in algebraic complexity: VP vs VNP and polynomial identity testing (PIT).
- Kabanets and Impagliazzo (2003): Exponential lower bound for arithmetic circuits $\implies$ Quasi-poly blackbox deterministic PIT for circuits.
- Hardness of multiples: If f is hard for $\mathcal{C}$, all its nonzero multiples are hard for $\mathcal{C}$.

- Oliveira (2015) proved poly(s) factor size bounds for constant depth circuits assuming the individual degree is constant.
- Dutta, Saxena, S (2018): If we take formula/ABP, $g \mid f \implies$ size$(g) \leq$ poly(size$(f), d^{O(\log d)})$.
- Chou, Kumar, Solomon (2018) showed that VNP is closed under factors.

- Oliveira (2015) proved $\text{poly}(s)$ factor size bounds for constant depth circuits assuming the individual degree is constant.

- Dutta, Saxena, S (2018): If we take formula/ABP, $g \mid f \implies \text{size}(g) \leq \text{poly}(\text{size}(f), d^{O(\log d)})$.

- Chou, Kumar, Solomon (2018) showed that VNP is closed under factors.

Factorization of Arithmetic Branching Programs

ABPs are closed under factors

Theorem (S, Thierauf, 2020)

Let polynomial $p(\bar{x})$ over $\mathbb{F}$ have ABP-size s.

All factors of p have ABP-size $\textsc{poly}(s)$

Algorithm: Factors can be efficiently constructed in randomized polynomial time.

ABPs are closed under factors

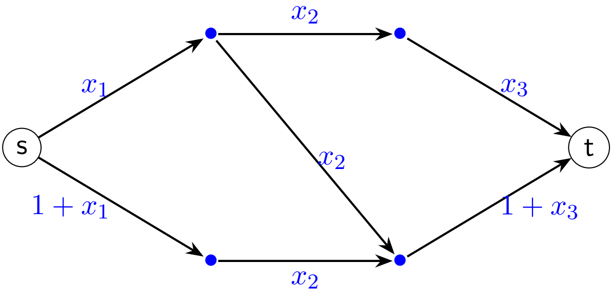
Theorem (S, Thierauf, 2020)

Let polynomial $p(\bar{x})$ over $\mathbb{F}$ have ABP-size s.

All factors of p have ABP-size $\textsc{poly}(s)$

Algorithm: Factors can be efficiently constructed in randomized polynomial time.

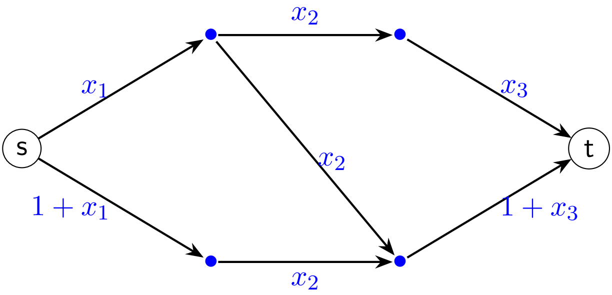
Example:



The polynomial computed by the above ABP is

$$x_1 x_2 x_3 + x_1 x_2 (1 + x_3) + (1 + x_1) x_2 (1 + x_3).$$

EXAMPLE:



The polynomial computed by the above ABP is

$$x_1x_2x_3 + x_1x_2(1 + x_3) + (1 + x_1)x_2(1 + x_3).$$

Computational power of ABPs

Arithmetic Formula $\leq$ ABP $\leq$ Arithmetic Circuit

DET: compute determinant of $n \times n$ matrices

- $\mathrm{DET} \in \mathrm{NC}^2$ [Csanky-Faddeev-LeVerrier 1976] [Berkowitz-Samuelson 1985, Chistov 1985]
- DET by poly(n)-size ABPs [Mahajan-Vinay 1997]

Consequence

Poly-size ABPs can compute solutions of linear systems over $\mathbb{F}(x_1, \ldots, x_n)$.

- Not known for formulas

Computational power of ABPs

Arithmetic Formula $\leq$ ABP $\leq$ Arithmetic Circuit

DET: compute determinant of $n \times n$ matrices

- $\mathrm{DET} \in \mathrm{NC}^2$ [Csanky-Faddeev-LeVerrier 1976] [Berkowitz-Samuelson 1985, Chistov 1985]
- DET by poly(n)-size ABPs [Mahajan-Vinay 1997]

Consequence

Poly-size ABPs can compute solutions of linear systems over $\mathbb{F}(x_1, \ldots, x_n)$.

- Not known for formulas

Computational power of ABPs

Arithmetic Formula $\leq$ ABP $\leq$ Arithmetic Circuit

DET: compute determinant of $n \times n$ matrices

- $\mathrm{DET} \in \mathrm{NC}^2$ [Csanky-Faddeev-LeVerrier 1976] [Berkowitz-Samuelson 1985, Chistov 1985]
- DET by poly(n)-size ABPs [Mahajan-Vinay 1997]

Consequence

Poly-size ABPs can compute solutions of linear systems over $\mathbb{F}(x_1, \ldots, x_n)$.

- Not known for formulas

Computational power of ABPs

Arithmetic Formula $\leq$ ABP $\leq$ Arithmetic Circuit

DET: compute determinant of $n \times n$ matrices

- $\mathrm{DET} \in \mathrm{NC}^2$ [Csanky-Faddeev-LeVerrier 1976] [Berkowitz-Samuelson 1985, Chistov 1985]
- DET by poly(n)-size ABPs [Mahajan-Vinay 1997]

Consequence

Poly-size ABPs can compute solutions of linear systems over $\mathbb{F}(x_1, \ldots, x_n)$.

- Not known for formulas

Computational power of ABPs

Arithmetic Formula $\leq$ ABP $\leq$ Arithmetic Circuit

DET: compute determinant of $n \times n$ matrices

- $\mathrm{DET} \in \mathrm{NC}^2$ [Csanky-Faddeev-LeVerrier 1976] [Berkowitz-Samuelson 1985, Chistov 1985]
- DET by poly(n)-size ABPs [Mahajan-Vinay 1997]

Consequence

Poly-size ABPs can compute solutions of linear systems over $\mathbb{F}(x_1, \ldots, x_n)$.

- Not known for formulas

Computational power of ABPs

Arithmetic Formula $\leq$ ABP $\leq$ Arithmetic Circuit

DET: compute determinant of $n \times n$ matrices

- $\mathrm{DET} \in \mathrm{NC}^2$ [Csanky-Faddeev-LeVerrier 1976] [Berkowitz-Samuelson 1985, Chistov 1985]
- DET by poly(n)-size ABPs [Mahajan-Vinay 1997]

Consequence

Poly-size ABPs can compute solutions of linear systems over $\mathbb{F}(x_1, \ldots, x_n)$.

- Not known for formulas

Newton Iteration [Oliveira 2016, Dutta, Saxena, S 2018]

Factoring $\leq$ root approximation in power series

- $$p(\boldsymbol{x}, y) \text{ has factor } y - q(\boldsymbol{x}) \iff p(\boldsymbol{x}, q(\boldsymbol{x})) = 0$$

- approximate root via Newton iteration

$$y_{t+1} = y_t - \frac{p(\boldsymbol{x}, y_t)}{p'(\boldsymbol{x}, y_t)}$$

- $\log d$ iterations leads to $d^{\log d}$-size ABPs

Hensel Lifting (Kurt Hensel, 1861 - 1941)

Newton Iteration [Oliveira 2016, Dutta, Saxena, S 2018]

Factoring $\leq$ root approximation in power series

- $$p(\boldsymbol{x}, y) \text{ has factor } y - q(\boldsymbol{x}) \iff p(\boldsymbol{x}, q(\boldsymbol{x})) = 0$$

- approximate root via Newton iteration

$$y_{t+1} = y_t - \frac{p(\boldsymbol{x}, y_t)}{p'(\boldsymbol{x}, y_t)}$$

- $\log d$ iterations leads to $d^{\log d}$-size ABPs

Hensel Lifting (Kurt Hensel, 1861 - 1941)

Newton Iteration [Oliveira 2016, Dutta, Saxena, S 2018]

Factoring $\leq$ root approximation in power series

- $$p(\boldsymbol{x}, y) \text{ has factor } y - q(\boldsymbol{x}) \iff p(\boldsymbol{x}, q(\boldsymbol{x})) = 0$$

- approximate root via Newton iteration

$$y_{t+1} = y_t - \frac{p(\boldsymbol{x}, y_t)}{p'(\boldsymbol{x}, y_t)}$$

- $\log d$ iterations leads to $d^{\log d}$-size ABPs

Hensel Lifting (Kurt Hensel, 1861 - 1941)

Techniques for factorization

Newton Iteration [Oliveira 2016, Dutta, Saxena, S 2018]

Factoring $\leq$ root approximation in power series

- $$p(\boldsymbol{x}, y) \text{ has factor } y - q(\boldsymbol{x}) \iff p(\boldsymbol{x}, q(\boldsymbol{x})) = 0$$

- approximate root via Newton iteration

$$y_{t+1} = y_t - \frac{p(\boldsymbol{x}, y_t)}{p'(\boldsymbol{x}, y_t)}$$

- $\log d$ iterations leads to $d^{\log d}$-size ABPs

Hensel Lifting (Kurt Hensel, 1861 - 1941)

Techniques for factorization

Newton Iteration [Oliveira 2016, Dutta, Saxena, S 2018]

Factoring $\leq$ root approximation in power series

- $$p(\boldsymbol{x}, y) \text{ has factor } y - q(\boldsymbol{x}) \iff p(\boldsymbol{x}, q(\boldsymbol{x})) = 0$$
- approximate root via Newton iteration
$$y_{t+1} = y_t - \frac{p(\boldsymbol{x}, y_t)}{p'(\boldsymbol{x}, y_t)}$$
- $\log d$ iterations leads to $d^{\log d}$-size ABPs

Hensel Lifting (Kurt Hensel, 1861 - 1941)

Factoring via Hensel Lifting

After preprocessing: $f(x, y)$, 2-variate, degree d, *monic* in x (i.e. highest x-power has constant coefficient)

Goal: Compute g s.t. $f = g\,h$

1. Initial step: Factorize univariate polynomial $f(x, 0)$
   - $f(x, 0) = g_0(x)\,h_0(x)$
   - Equivalently: $f(x, y) \equiv g_0(x)\,h_0(x) \pmod{y}$
2. Lifting: compute $g_1(x, y), h_1(x, y)$
   - $f \equiv g_1\,h_1 \pmod{y^2}$

Iterate lifting $t = O(\log d)$ times: $f \equiv g_t\,h_t \pmod{y^{2^t}}$

After preprocessing: $f(x,y)$, 2-variate, degree d, *monic* in x (i.e. highest x-power has constant coefficient)

Goal: Compute g s.t. $f = g\,h$

1. Initial step: Factorize univariate polynomial $f(x,0)$
   - $f(x,0) = g_0(x)\,h_0(x)$
   - Equivalently: $f(x,y) \equiv g_0(x)\,h_0(x) \pmod{y}$
2. Lifting: compute $g_1(x,y), h_1(x,y)$
   - $f \equiv g_1\,h_1 \pmod{y^2}$

Iterate lifting $t = O(\log d)$ times: $f \equiv g_t\,h_t \pmod{y^{2^t}}$

Factoring via Hensel Lifting

After preprocessing: $f(x,y)$, 2-variate, degree d, *monic* in x (i.e. highest x-power has constant coefficient)

Goal: Compute g s.t. $f = g\,h$

1. Initial step: Factorize univariate polynomial $f(x,0)$
   - $f(x,0) = g_0(x)\,h_0(x)$
   - Equivalently: $f(x,y) \equiv g_0(x)\,h_0(x) \pmod{y}$
2. Lifting: compute $g_1(x,y), h_1(x,y)$
   - $f \equiv g_1\,h_1 \pmod{y^2}$

Iterate lifting $t = O(\log d)$ times: $f \equiv g_t\,h_t \pmod{y^{2^t}}$

Factoring via Hensel Lifting

After preprocessing: $f(x, y)$, 2-variate, degree d, *monic* in x (i.e. highest x-power has constant coefficient)

Goal: Compute g s.t. $f = g\,h$

1. Initial step: Factorize univariate polynomial $f(x, 0)$
   - $f(x, 0) = g_0(x)\,h_0(x)$
   - Equivalently: $f(x, y) \equiv g_0(x)\,h_0(x) \pmod{y}$

2. Lifting: compute $g_1(x, y), h_1(x, y)$
   - $f \equiv g_1\,h_1 \pmod{y^2}$

Iterate lifting $t = O(\log d)$ times: $f \equiv g_t\,h_t \pmod{y^{2^t}}$

Factoring via Hensel Lifting

After preprocessing: $f(x, y)$, 2-variate, degree d, *monic* in x (i.e. highest x-power has constant coefficient)

Goal: Compute g s.t. $f = g\,h$

1. Initial step: Factorize univariate polynomial $f(x, 0)$
   - $f(x, 0) = g_0(x)\,h_0(x)$
   - Equivalently: $f(x, y) \equiv g_0(x)\,h_0(x) \pmod{y}$
2. Lifting: compute $g_1(x, y), h_1(x, y)$
   - $f \equiv g_1\,h_1 \pmod{y^2}$

Iterate lifting $t = O(\log d)$ times: $f \equiv g_t\,h_t \pmod{y^{2^t}}$

Factoring via Hensel Lifting

After preprocessing: $f(x, y)$, 2-variate, degree d, *monic* in x
(i.e. highest x-power has constant coefficient)

Goal: Compute g s.t. $f = g\,h$

1. Initial step: Factorize univariate polynomial $f(x, 0)$
   - $f(x, 0) = g_0(x)\,h_0(x)$
   - Equivalently: $f(x, y) \equiv g_0(x)\,h_0(x) \pmod{y}$
2. Lifting: compute $g_1(x, y), h_1(x, y)$
   - $f \equiv g_1\,h_1 \pmod{y^2}$

Iterate lifting $t = O(\log d)$ times: $f \equiv g_t\,h_t \pmod{y^{2^t}}$

Hensel Lifting

$$\begin{aligned} f &= g\,h \\ f &\equiv g_t\,h_t \pmod{y^{2^t}} \end{aligned}$$

One can show: for some polynomial h'_t

$$g \equiv g_t\,h'_t \pmod{y^{2^t}}$$

- g_t known, but g and h'_t unknown
- Set up linear system in unknown coefficients of g and h'
- Jump Step: Right choice of $t \implies$ we get g, without any mod! Can be proved using resultants.

Hensel Lifting

$$\begin{aligned} f &= g\,h \\ f &\equiv g_t\,h_t \pmod{y^{2^t}} \end{aligned}$$

One can show: for some polynomial h'_t

$$g \equiv g_t\,h'_t \pmod{y^{2^t}}$$

- g_t known, but g and h'_t unknown
- Set up linear system in unknown coefficients of g and h'
- Jump Step: Right choice of $t \implies$ we get g, without any mod! Can be proved using resultants.

Hensel Lifting

$$\begin{aligned} f &= g\,h \\ f &\equiv g_t\,h_t \pmod{y^{2^t}} \end{aligned}$$

One can show: for some polynomial h'_t

$$g \equiv g_t\,h'_t \pmod{y^{2^t}}$$

- g_t known, but g and h'_t unknown
- Set up linear system in unknown coefficients of g and h'
- Jump Step: Right choice of $t \implies$ we get g, without any mod! Can be proved using resultants.

Hensel Lifting

$$
\begin{aligned}
f &= g\,h \\
f &\equiv g_t\,h_t \pmod{y^{2^t}}
\end{aligned}
$$

One can show: for some polynomial h_t'

$$g \equiv g_t\,h_t' \pmod{y^{2^t}}$$

- g_t known, but g and h_t' unknown
- Set up linear system in unknown coefficients of g and h'
- Jump Step: Right choice of $t \implies$ we get g, without any mod! Can be proved using resultants.

$$
\begin{aligned}
f &= g\,h \\
f &\equiv g_t\,h_t \pmod{y^{2^t}}
\end{aligned}
$$

One can show: for some polynomial h'_t

$$g \equiv g_t\,h'_t \pmod{y^{2^t}}$$

- g_t known, but g and h'_t unknown
- Set up linear system in unknown coefficients of g and h'
- Jump Step: Right choice of $t \implies$ we get g, without any mod! Can be proved using resultants.

Main difference to earlier liftings

Start with g_0, h_0 monic

- standard Hensel Lifting maintains g_k, h_k monic, for all k
- simplified version gives up monicness: saves a division
- ABP-size grows by a constant factor in each iteration

  $\implies$ overall size $\mathrm{poly}(c^{\log d}, s) = \mathrm{poly}(s)$
- Crucial technical part: Jump Step still works!

Main difference to earlier liftings

Start with g_0, h_0 monic

- standard Hensel Lifting maintains g_k, h_k monic, for all k
- simplified version gives up monicness: saves a division
- ABP-size grows by a constant factor in each iteration
  $\implies$ overall size $\text{poly}(c^{\log d}, s) = \text{poly}(s)$
- Crucial technical part: Jump Step still works!

Main difference to earlier liftings

Start with g_0, h_0 monic

- standard Hensel Lifting maintains g_k, h_k monic, for all k
- simplified version gives up monicness: saves a division
- ABP-size grows by a constant factor in each iteration

  $\Longrightarrow$ overall size $\mathrm{poly}(c^{\log d}, s) = \mathrm{poly}(s)$
- Crucial technical part: Jump Step still works!

Main difference to earlier liftings

Start with g_0, h_0 monic

- standard Hensel Lifting maintains g_k, h_k monic, for all k
- simplified version gives up monicness: saves a division
- ABP-size grows by a constant factor in each iteration

  $\Longrightarrow$ overall size $\mathrm{poly}(c^{\log d}, s) = \mathrm{poly}(s)$
- Crucial technical part: Jump Step still works!

Main difference to earlier liftings

Start with g_0, h_0 monic

- standard Hensel Lifting maintains g_k, h_k monic, for all k
- simplified version gives up monicness: saves a division
- ABP-size grows by a constant factor in each iteration

  $\Longrightarrow$ overall size $\text{poly}(c^{\log d}, s) = \text{poly}(s)$
- Crucial technical part: Jump Step still works!

Main difference to earlier liftings

Start with g_0, h_0 monic

- standard Hensel Lifting maintains g_k, h_k monic, for all k
- simplified version gives up monicness: saves a division
- ABP-size grows by a constant factor in each iteration

  $\implies$ overall size $\text{poly}(c^{\log d}, s) = \text{poly}(s)$
- Crucial technical part: Jump Step still works!

- Let $\mathcal{R}$ be a commutative ring with 1 and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal.
- Condition for lift: Let $f, g, h, a, b \in \mathcal{R}$ such that
  - (factorization) $f \equiv gh \pmod{\mathcal{I}}$
  - (pseudo-coprimality) $ag + bh \equiv 1 \pmod{\mathcal{I}}$.
- Then g', h' is lift of g, h w.r.t. f if it satisfy the following.
  - (Better factorization) $f \equiv g'h' \pmod{\mathcal{I}^2}$,
  - (Lifts) $g' \equiv g \pmod{\mathcal{I}}$ and $h' \equiv h \pmod{\mathcal{I}}$, and
  - (pseudo-coprimality) $\exists a', b' \in \mathcal{R} \quad a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$.

- Let $\mathcal{R}$ be a commutative ring with 1 and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal.
- Condition for lift: Let $f, g, h, a, b \in \mathcal{R}$ such that
  - (factorization) $f \equiv gh \pmod{\mathcal{I}}$
  - (pseudo-coprimality) $ag + bh \equiv 1 \pmod{\mathcal{I}}$.
- Then g', h' is lift of g, h w.r.t. f if it satisfy the following.
  - (Better factorization) $f \equiv g'h' \pmod{\mathcal{I}^2}$,
  - (Lifts) $g' \equiv g \pmod{\mathcal{I}}$ and $h' \equiv h \pmod{\mathcal{I}}$, and
  - (pseudo-coprimality) $\exists a', b' \in \mathcal{R} \quad a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$.

- Let $\mathcal{R}$ be a commutative ring with 1 and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal.
- Condition for lift: Let $f, g, h, a, b \in \mathcal{R}$ such that
  - (factorization) $f \equiv gh \pmod{\mathcal{I}}$
  - (pseudo-coprimality) $ag + bh \equiv 1 \pmod{\mathcal{I}}$.
- Then g', h' is lift of g, h w.r.t. f if it satisfy the following.
  - (Better factorization) $f \equiv g'h' \pmod{\mathcal{I}^2}$,
  - (Lifts) $g' \equiv g \pmod{\mathcal{I}}$ and $h' \equiv h \pmod{\mathcal{I}}$, and
  - (pseudo-coprimality) $\exists a', b' \in \mathcal{R} \quad a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$.

Hensel Lifting: Definition of Lift

- Let $\mathcal{R}$ be a commutative ring with 1 and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal.
- Condition for lift: Let $f, g, h, a, b \in \mathcal{R}$ such that
  - (factorization) $f \equiv gh \pmod{\mathcal{I}}$
  - (pseudo-coprimality) $ag + bh \equiv 1 \pmod{\mathcal{I}}$.
- Then g', h' is lift of g, h w.r.t. f if it satisfy the following.
  - (Better factorization) $f \equiv g'h' \pmod{\mathcal{I}^2}$,
  - (Lifts) $g' \equiv g \pmod{\mathcal{I}}$ and $h' \equiv h \pmod{\mathcal{I}}$, and
  - (pseudo-coprimality) $\exists a', b' \in \mathcal{R} \quad a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$.

- Let $\mathcal{R}$ be a commutative ring with 1 and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal.
- Condition for lift: Let $f, g, h, a, b \in \mathcal{R}$ such that
  - (factorization) $f \equiv gh \pmod{\mathcal{I}}$
  - (pseudo-coprimality) $ag + bh \equiv 1 \pmod{\mathcal{I}}$.
- Then g', h' is lift of g, h w.r.t. f if it satisfy the following.
  - (Better factorization) $f \equiv g'h' \pmod{\mathcal{I}^2}$,
  - (Lifts) $g' \equiv g \pmod{\mathcal{I}}$ and $h' \equiv h \pmod{\mathcal{I}}$, and
  - (pseudo-coprimality) $\exists a', b' \in \mathcal{R} \quad a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$.

- Let $\mathcal{R}$ be a commutative ring with 1 and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal.
- Condition for lift: Let $f, g, h, a, b \in \mathcal{R}$ such that
  - (factorization) $f \equiv gh \pmod{\mathcal{I}}$
  - (pseudo-coprimality) $ag + bh \equiv 1 \pmod{\mathcal{I}}$.
- Then g', h' is lift of g, h w.r.t. f if it satisfy the following.
  - (Better factorization) $f \equiv g'h' \pmod{\mathcal{I}^2}$,
  - (Lifts) $g' \equiv g \pmod{\mathcal{I}}$ and $h' \equiv h \pmod{\mathcal{I}}$, and
  - (pseudo-coprimality) $\exists a', b' \in \mathcal{R} \quad a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$.

- Let $\mathcal{R}$ be a commutative ring with 1 and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal.
- Condition for lift: Let $f, g, h, a, b \in \mathcal{R}$ such that
  - (factorization) $f \equiv gh \pmod{\mathcal{I}}$
  - (pseudo-coprimality) $ag + bh \equiv 1 \pmod{\mathcal{I}}$.
- Then g', h' is lift of g, h w.r.t. f if it satisfy the following.
  - (Better factorization) $f \equiv g'h' \pmod{\mathcal{I}^2}$,
  - (Lifts) $g' \equiv g \pmod{\mathcal{I}}$ and $h' \equiv h \pmod{\mathcal{I}}$, and
  - (pseudo-coprimality) $\exists a', b' \in \mathcal{R} \quad a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$.

Nonmonic Lifting is cheaper

- Compute the following.
- $e = f - gh$
- $g' = g + be$ and $h' = h + ae$
- $c = ag' + bh' - 1$
- $a' = a(1 - c)$ and $b' = b(1 - c)$.
- If g, h, a, b are small formulas, we have to make 5 copies of them to compute the next round g', h', a', b'.

- Compute the following.
- $e = f - gh$
- $g' = g + be$ and $h' = h + ae$
- $c = ag' + bh' - 1$
- $a' = a(1-c)$ and $b' = b(1-c)$.
- If g, h, a, b are small formulas, we have to make 5 copies of them to compute the next round g', h', a', b'.

Nonmonic Lifting is cheaper

- Compute the following.
- $e = f - gh$
- $g' = g + be$ and $h' = h + ae$
- $c = ag' + bh' - 1$
- $a' = a(1 - c)$ and $b' = b(1 - c)$.
- If g, h, a, b are small formulas, we have to make 5 copies of them to compute the next round g', h', a', b'.

- Compute the following.
- $e = f - gh$
- $g' = g + be$ and $h' = h + ae$
- $c = ag' + bh' - 1$
- $a' = a(1-c)$ and $b' = b(1-c)$.
- If g, h, a, b are small formulas, we have to make 5 copies of them to compute the next round g', h', a', b'.

- Compute the following.
- $e = f - gh$
- $g' = g + be$ and $h' = h + ae$
- $c = ag' + bh' - 1$
- $a' = a(1 - c)$ and $b' = b(1 - c)$.
- If g, h, a, b are small formulas, we have to make 5 copies of them to compute the next round g', h', a', b'.

- Compute the following.
- $e = f - gh$
- $g' = g + be$ and $h' = h + ae$
- $c = ag' + bh' - 1$
- $a' = a(1 - c)$ and $b' = b(1 - c)$.
- If g, h, a, b are small formulas, we have to make 5 copies of them to compute the next round g', h', a', b'.

- Assume f, g, h are polynomials monic in x.
- Additionally compute polynomials q and r such that $g' - g = qg + r$, where $\deg_x(r) < \deg_x(g)$.
- $\hat{g} = g + r$ and $\hat{h} = h'(1 + q)$ are a monic lift of g, h w.r.t. f.
- Advantage: We can avoid the linear system-solving step if we start monic lifting from $g(x, 0)$ and $h(x, 0)$!
- Disadvantage: Implementing it for formulas/ABPs requires making d^2 many copies of previous lifts in each round.

- Assume f, g, h are polynomials monic in x.
- Additionally compute polynomials q and r such that $g' - g = qg + r$, where $\deg_x(r) < \deg_x(g)$.
- $\hat{g} = g + r$ and $\hat{h} = h'(1+q)$ are a monic lift of g, h w.r.t. f.
- Advantage: We can avoid the linear system-solving step if we start monic lifting from $g(x,0)$ and $h(x,0)$!
- Disadvantage: Implementing it for formulas/ABPs requires making d^2 many copies of previous lifts in each round.

Monic Version of Hensel Lifting

- Assume f, g, h are polynomials monic in x.
- Additionally compute polynomials q and r such that $g' - g = qg + r$, where $\deg_x(r) < \deg_x(g)$.
- $\hat{g} = g + r$ and $\hat{h} = h'(1 + q)$ are a monic lift of g, h w.r.t. f.
- Advantage: We can avoid the linear system-solving step if we start monic lifting from $g(x, 0)$ and $h(x, 0)$!
- Disadvantage: Implementing it for formulas/ABPs requires making d^2 many copies of previous lifts in each round.

- Assume f, g, h are polynomials monic in x.
- Additionally compute polynomials q and r such that $g' - g = qg + r$, where $\deg_x(r) < \deg_x(g)$.
- $\hat{g} = g + r$ and $\hat{h} = h'(1+q)$ are a monic lift of g, h w.r.t. f.
- Advantage: We can avoid the linear system-solving step if we start monic lifting from $g(x,0)$ and $h(x,0)$!
- Disadvantage: Implementing it for formulas/ABPs requires making d^2 many copies of previous lifts in each round.

Lifted Factor and Actual Factor

Lemma (Actual factor vs lifted factor)

$g \equiv g_t h'_t \pmod{y^{2^t}}$ for some polynomial h'_t.

Proof Idea

Inductively apply Hensel lifting to both f and factor g starting from $f = g_0 h_0 \pmod{y}$ and $g = g_0 h'_0 \pmod{y}$ respectively.

From the proof, we do not get h'_t explicitly if we do not know g.

Lemma (Actual factor vs lifted factor)

$g \equiv g_t h'_t \pmod{y^{2^t}}$ for some polynomial h'_t.

Proof Idea

Inductively apply Hensel lifting to both f and factor g starting from $f = g_0 h_0 \pmod{y}$ and $g = g_0 h'_0 \pmod{y}$ respectively.

From the proof, we do not get h'_t explicitly if we do not know g.

- We want to compute g from the Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- Here g_t is known but both g and h'_t are unknown. We know the degree upper bounds of g, g_t, h'_t.
- Compare the coefficients of each monomial $x^i y^j$ in LHS and RHS of Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- We get a system of linear equations in the unknowns (coefficients of g and h'_t).

- We want to compute g from the Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- Here g_t is known but both g and h'_t are unknown. We know the degree upper bounds of g, g_t, h'_t.
- Compare the coefficients of each monomial $x^i y^j$ in LHS and RHS of Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- We get a system of linear equations in the unknowns (coefficients of g and h'_t).

Factor Reconstruction via Linear System

- We want to compute g from the Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- Here g_t is known but both g and h'_t are unknown. We know the degree upper bounds of g, g_t, h'_t.
- Compare the coefficients of each monomial $x^i y^j$ in LHS and RHS of Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- We get a system of linear equations in the unknowns (coefficients of g and h'_t).

- We want to compute g from the Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- Here g_t is known but both g and h'_t are unknown. We know the degree upper bounds of g, g_t, h'_t.
- Compare the coefficients of each monomial $x^i y^j$ in LHS and RHS of Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- We get a system of linear equations in the unknowns (coefficients of g and h'_t).

Factor Reconstruction via Linear System

- We want to compute g from the Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- Here g_t is known but both g and h'_t are unknown. We know the degree upper bounds of g, g_t, h'_t.
- Compare the coefficients of each monomial $x^i y^j$ in LHS and RHS of Eqn. $g \equiv g_t h'_t \pmod{y^{2^t}}$.
- We get a system of linear equations in the unknowns (coefficients of g and h'_t).

- Let $\tilde{g}$ be a least degree monic polynomial that satisfies $\tilde{g} = g_t\tilde{h} \pmod{y^{2^t}}$ for some $\tilde{h}$. We prove that $\tilde{g} = g$.
- First prove that $\tilde{g}$ and the factor g have nontrivial gcd by showing that their Resultant is zero.
- As factor g is irreducible, we get g divides $\tilde{g}$.
- As both $\tilde{g}$ and g are monic polynomials of same degree, they must be equal.

- Let $\tilde{g}$ be a least degree monic polynomial that satisfies $\tilde{g} = g_t\tilde{h} \pmod{y^{2^t}}$ for some $\tilde{h}$. We prove that $\tilde{g} = g$.
- First prove that $\tilde{g}$ and the factor g have nontrivial gcd by showing that their Resultant is zero.
- As factor g is irreducible, we get g divides $\tilde{g}$.
- As both $\tilde{g}$ and g are monic polynomials of same degree, they must be equal.

- Let $\tilde{g}$ be a least degree monic polynomial that satisfies $\tilde{g} = g_t\tilde{h} \pmod{y^{2^t}}$ for some $\tilde{h}$. We prove that $\tilde{g} = g$.
- First prove that $\tilde{g}$ and the factor g have nontrivial gcd by showing that their Resultant is zero.
- As factor g is irreducible, we get g divides $\tilde{g}$.
- As both $\tilde{g}$ and g are monic polynomials of same degree, they must be equal.

- Let $\tilde{g}$ be a least degree monic polynomial that satisfies $\tilde{g} = g_t\tilde{h} \pmod{y^{2^t}}$ for some $\tilde{h}$. We prove that $\tilde{g} = g$.
- First prove that $\tilde{g}$ and the factor g have nontrivial gcd by showing that their Resultant is zero.
- As factor g is irreducible, we get g divides $\tilde{g}$.
- As both $\tilde{g}$ and g are monic polynomials of same degree, they must be equal.

GCD and Resultants

- The resultant $r(y) = \mathrm{Res}_x(g, \tilde{g})$ is a polynomial (of degree $\leq 2d^2$) in y defined via determinant of Sylvester matrix.

- $\mathrm{Res}_x(g, \tilde{g}) = 0 \iff g, \tilde{g}$ share nontrivial gcd.

- The resultant $r(y) = \mathsf{Res}_x(g, \tilde{g})$ is a polynomial (of degree $\leq 2d^2$) in y defined via determinant of Sylvester matrix.

- $\mathsf{Res}_x(g, \tilde{g}) = 0 \iff g, \tilde{g}$ share nontrivial gcd.

- The resultant $r(y) = \mathrm{Res}_x(g, \tilde{g})$ is a polynomial (of degree $\leq 2d^2$) in y defined via determinant of Sylvester matrix.
- $\mathrm{Res}_x(g, \tilde{g}) = 0 \iff g, \tilde{g}$ share nontrivial gcd.

- Resultant as linear combination: $\exists u, v$ s.t $r(y) = ug + v\tilde{g}$.
- Plug-in $g = g_t h_t' \pmod{y^{2^t}}$ and $\tilde{g} = g_t \tilde{h} \pmod{y^{2^t}}$.
- So we get $r(y) = ug + v\tilde{g} \equiv g_t(uh_t' + v\tilde{h}) \pmod{y^{2^t}}$.
- Let w denote $(uh_t' + v\tilde{h})$. So we have $r(y) = g_t w \pmod{y^{2^t}}$.
- Assume for sake of contradiction $r(y)$ is nonzero.

- Resultant as linear combination: $\exists u, v$ s.t $r(y) = ug + v\tilde{g}$.
- Plug-in $g = g_t h'_t \pmod{y^{2^t}}$ and $\tilde{g} = g_t \tilde{h} \pmod{y^{2^t}}$.
- So we get $r(y) = ug + v\tilde{g} \equiv g_t(uh'_t + v\tilde{h}) \pmod{y^{2^t}}$.
- Let w denote $(uh'_t + v\tilde{h})$. So we have $r(y) = g_t w \pmod{y^{2^t}}$.
- Assume for sake of contradiction $r(y)$ is nonzero.

- Resultant as linear combination: $\exists u, v$ s.t $r(y) = ug + v\tilde{g}$.
- Plug-in $g = g_t h'_t \pmod{y^{2^t}}$ and $\tilde{g} = g_t \tilde{h} \pmod{y^{2^t}}$.
- So we get $r(y) = ug + v\tilde{g} \equiv g_t(uh'_t + v\tilde{h}) \pmod{y^{2^t}}$.
- Let w denote $(uh'_t + v\tilde{h})$. So we have $r(y) = g_t w \pmod{y^{2^t}}$.
- Assume for sake of contradiction $r(y)$ is nonzero.

- Resultant as linear combination: $\exists u, v$ s.t $r(y) = ug + v\tilde{g}$.
- Plug-in $g = g_t h_t' \pmod{y^{2^t}}$ and $\tilde{g} = g_t \tilde{h} \pmod{y^{2^t}}$.
- So we get $r(y) = ug + v\tilde{g} \equiv g_t(uh_t' + v\tilde{h}) \pmod{y^{2^t}}$.
- Let w denote $(uh_t' + v\tilde{h})$. So we have $r(y) = g_t w \pmod{y^{2^t}}$.
- Assume for sake of contradiction $r(y)$ is nonzero.

- Resultant as linear combination: $\exists u, v$ s.t $r(y) = ug + v\tilde{g}$.
- Plug-in $g = g_t h_t' \pmod{y^{2^t}}$ and $\tilde{g} = g_t \tilde{h} \pmod{y^{2^t}}$.
- So we get $r(y) = ug + v\tilde{g} \equiv g_t(uh_t' + v\tilde{h}) \pmod{y^{2^t}}$.
- Let w denote $(uh_t' + v\tilde{h})$. So we have $r(y) = g_t w \pmod{y^{2^t}}$.
- Assume for sake of contradiction $r(y)$ is nonzero.

- g_t is monic in x and $w \neq 0 \implies$ coefficient of highest power of x in $g_t w \pmod{y^{2^t}}$ is nonzero.
- On the other hand, $r(y)$ is free of x. That gives a contradiction. Thus $r(y) = 0 \pmod{y^{2^t}}$.
- But here g_t is nonmonic. So the coefficient of highest power of x in g_t is a multiple of y.
- Thus the highest power of x in $g_t w$ may vanish modulo y^{2^t}. Can we still save the argument?

- g_t is monic in x and $w \neq 0 \implies$ coefficient of highest power of x in $g_t w \pmod{y^{2^t}}$ is nonzero.
- On the other hand, $r(y)$ is free of x. That gives a contradiction. Thus $r(y) = 0 \pmod{y^{2^t}}$.
- But here g_t is nonmonic. So the coefficient of highest power of x in g_t is a multiple of y.
- Thus the highest power of x in $g_t w$ may vanish modulo y^{2^t}. Can we still save the argument?

- g_t is monic in x and $w \neq 0 \implies$ coefficient of highest power of x in $g_t w \pmod{y^{2^t}}$ is nonzero.
- On the other hand, $r(y)$ is free of x. That gives a contradiction. Thus $r(y) = 0 \pmod{y^{2^t}}$.
- But here g_t is nonmonic. So the coefficient of highest power of x in g_t is a multiple of y.
- Thus the highest power of x in $g_t w$ may vanish modulo y^{2^t}. Can we still save the argument?

- g_t is monic in x and $w \neq 0 \implies$ coefficient of highest power of x in $g_t w \pmod{y^{2^t}}$ is nonzero.

- On the other hand, $r(y)$ is free of x. That gives a contradiction. Thus $r(y) = 0 \pmod{y^{2^t}}$.

- But here g_t is nonmonic. So the coefficient of highest power of x in g_t is a multiple of y.

- Thus the highest power of x in $g_t w$ may vanish modulo y^{2^t}. Can we still save the argument?

- Idea: Look at the least power of y in both w and g_t.
- View g_t and w as polynomials in y with coefficients in x. Suppose $g_t = c_0(x) + c_1(x)y + \ldots + c_{d'}(x)y^{d'}$.
- Now, $g_t \equiv g_0 \pmod{y}$, so $c_0(x) = g_0(x)$, a nonzero poly in x.
- The least power of y in $g_t w$ has coefficient $g_0(x)w_j(x)$, a nonzero polynomial in x.
- Thus $g_t w \bmod y^{2^t}$ is not free of x. Contradiction. Thus, $r(y)$ must be 0 mod y^{2^t}. If $2^t \geq 2d^2 + 1$, then $r(y) = 0$. QED.

- Idea: Look at the least power of y in both w and g_t.
- View g_t and w as polynomials in y with coefficients in x. Suppose $g_t = c_0(x) + c_1(x)y + \ldots + c_{d'}(x)y^{d'}$.
- Now, $g_t \equiv g_0 \pmod{y}$, so $c_0(x) = g_0(x)$, a nonzero poly in x.
- The least power of y in $g_t w$ has coefficient $g_0(x)w_j(x)$, a nonzero polynomial in x.
- Thus $g_t w \bmod y^{2^t}$ is not free of x. Contradiction. Thus, $r(y)$ must be 0 mod y^{2^t}. If $2^t \geq 2d^2+1$, then $r(y) = 0$. QED.

Finishing the Proof

- Idea: Look at the least power of y in both w and g_t.
- View g_t and w as polynomials in y with coefficients in x. Suppose $g_t = c_0(x) + c_1(x)y + \ldots + c_{d'}(x)y^{d'}$.
- Now, $g_t \equiv g_0 \pmod{y}$, so $c_0(x) = g_0(x)$, a nonzero poly in x.
- The least power of y in $g_t w$ has coefficient $g_0(x)w_j(x)$, a nonzero polynomial in x.
- Thus $g_t w \bmod y^{2^t}$ is not free of x. Contradiction. Thus, $r(y)$ must be 0 mod y^{2^t}. If $2^t \geq 2d^2 + 1$, then $r(y) = 0$. QED.

- Idea: Look at the least power of y in both w and g_t.
- View g_t and w as polynomials in y with coefficients in x. Suppose $g_t = c_0(x) + c_1(x)y + \ldots + c_{d'}(x)y^{d'}$.
- Now, $g_t \equiv g_0 \pmod{y}$, so $c_0(x) = g_0(x)$, a nonzero poly in x.
- The least power of y in $g_t w$ has coefficient $g_0(x)w_j(x)$, a nonzero polynomial in x.
- Thus $g_t w \bmod y^{2^t}$ is not free of x. Contradiction. Thus, $r(y)$ must be 0 mod y^{2^t}. If $2^t \geq 2d^2 + 1$, then $r(y) = 0$. QED.

- Idea: Look at the least power of y in both w and g_t.
- View g_t and w as polynomials in y with coefficients in x. Suppose $g_t = c_0(x) + c_1(x)y + \ldots + c_{d'}(x)y^{d'}$.
- Now, $g_t \equiv g_0 \pmod{y}$, so $c_0(x) = g_0(x)$, a nonzero poly in x.
- The least power of y in $g_t w$ has coefficient $g_0(x)w_j(x)$, a nonzero polynomial in x.
- Thus $g_t w \bmod y^{2^t}$ is not free of x. Contradiction. Thus, $r(y)$ must be 0 mod y^{2^t}. If $2^t \geq 2d^2 + 1$, then $r(y) = 0$. QED.

- Let $f(x, z_1, \ldots, z_n)$ be the given polynomial to be factored
- Create a new polynomial $\widehat{f}(x, y, \boldsymbol{z}) = f(x, yz_1, \ldots, yz_n)$
- Consider $\widehat{f}$ as a bivariate in x and y with coefficients in $\mathbb{F}[\boldsymbol{z}]$.
- To get back f from $\widehat{f}$, simply put y to 1 in $\widehat{f}$.
- Putting y to 0 in $\widehat{f}$, we get univariate $\widehat{f}(x)$.

- Let $f(x, z_1, \ldots, z_n)$ be the given polynomial to be factored
- Create a new polynomial $\widehat{f}(x, y, \boldsymbol{z}) = f(x, yz_1, \ldots, yz_n)$
- Consider $\widehat{f}$ as a bivariate in x and y with coefficients in $\mathbb{F}[\boldsymbol{z}]$.
- To get back f from $\widehat{f}$, simply put y to 1 in $\widehat{f}$.
- Putting y to 0 in $\widehat{f}$, we get univariate $\widehat{f}(x)$.

- Let $f(x, z_1, \ldots, z_n)$ be the given polynomial to be factored
- Create a new polynomial $\widehat{f}(x, y, \boldsymbol{z}) = f(x, yz_1, \ldots, yz_n)$
- Consider $\widehat{f}$ as a bivariate in x and y with coefficients in $\mathbb{F}[\boldsymbol{z}]$.
- To get back f from $\widehat{f}$, simply put y to 1 in $\widehat{f}$.
- Putting y to 0 in $\widehat{f}$, we get univariate $\widehat{f}(x)$.

- Let $f(x, z_1, \ldots, z_n)$ be the given polynomial to be factored
- Create a new polynomial $\widehat{f}(x, y, \boldsymbol{z}) = f(x, yz_1, \ldots, yz_n)$
- Consider $\widehat{f}$ as a bivariate in x and y with coefficients in $\mathbb{F}[\boldsymbol{z}]$.
- To get back f from $\widehat{f}$, simply put y to 1 in $\widehat{f}$.
- Putting y to 0 in $\widehat{f}$, we get univariate $\widehat{f}(x)$.

Linear System over a big field

- The reduction to bivariate seems cheap.
- But we have to pay the price in the jump step. Our linear system now has coefficients from $\mathbb{F}(z_1, \ldots, z_n)$.
- Using Cramer's rule, the solutions can be expressed via Determinants/ABPs.
- Derandomization of the jump step requires PIT for ABPs.

Linear System over a big field

- The reduction to bivariate seems cheap.
- But we have to pay the price in the jump step. Our linear system now has coefficients from $\mathbb{F}(z_1, \ldots, z_n)$.
- Using Cramer's rule, the solutions can be expressed via Determinants/ABPs.
- Derandomization of the jump step requires PIT for ABPs.

- The reduction to bivariate seems cheap.
- But we have to pay the price in the jump step. Our linear system now has coefficients from $\mathbb{F}(z_1, \ldots, z_n)$.
- Using Cramer's rule, the solutions can be expressed via Determinants/ABPs.
- Derandomization of the jump step requires PIT for ABPs.

- The reduction to bivariate seems cheap.
- But we have to pay the price in the jump step. Our linear system now has coefficients from $\mathbb{F}(z_1, \ldots, z_n)$.
- Using Cramer's rule, the solutions can be expressed via Determinants/ABPs.
- Derandomization of the jump step requires PIT for ABPs.

Factorization and PIT

- Test if $g(x_1, \ldots, x_n)$ divides $f(x_1, \ldots, x_n)$.
- Reduces to Polynomial Identity Testing.
- We don't know a deterministic poly-time algorithm even when f, g are sparse.

- Test if $g(x_1, \ldots, x_n)$ divides $f(x_1, \ldots, x_n)$.
- Reduces to Polynomial Identity Testing.
- We don't know a deterministic poly-time algorithm even when f, g are sparse.

- Test if $g(x_1, \ldots, x_n)$ divides $f(x_1, \ldots, x_n)$.
- Reduces to Polynomial Identity Testing.
- We don't know a deterministic poly-time algorithm even when f, g are sparse.

- Whether a linear polynomial divides a sparse polynomial can be tested in polynomial time.
- $f(x_1, \ldots, x_n)$ is divisible by $x_1 - \ell(x_2, \ldots, x_n)$ iff $f(\ell, x_2, \ldots, x_n) = 0$.
- Semidiagonal model: $\Sigma m_i \ell^{e_i}$ where m_i is a monomial and ℓ are linear polynomials [Saha-Saptharishi-Saxena 2010].
- Testing if a quadratic polynomial divides a s-sparse polynomial in $s^{\log s}$ time [Forbes 2015]. Reduces to PIT of sums of monomials times powers of quadratics.
- If f, g are sparse polynomials with bounded individual degrees, [Volkovich 2017] gave a poly-time test.

- Whether a linear polynomial divides a sparse polynomial can be tested in polynomial time.
- $f(x_1, \ldots, x_n)$ is divisible by $x_1 - \ell(x_2, \ldots, x_n)$ iff $f(\ell, x_2, \ldots, x_n) = 0$.
- Semidiagonal model: $\Sigma m_i \ell^{e_i}$ where m_i is a monomial and ℓ are linear polynomials [Saha-Saptharishi-Saxena 2010].
- Testing if a quadratic polynomial divides a s-sparse polynomial in $s^{\log s}$ time [Forbes 2015]. Reduces to PIT of sums of monomials times powers of quadratics.
- If f, g are sparse polynomials with bounded individual degrees, [Volkovich 2017] gave a poly-time test.

- Whether a linear polynomial divides a sparse polynomial can be tested in polynomial time.
- $f(x_1, \dots, x_n)$ is divisible by $x_1 - \ell(x_2, \dots, x_n)$ iff $f(\ell, x_2, \dots, x_n) = 0$.
- Semidiagonal model: $\Sigma m_i \ell^{e_i}$ where m_i is a monomial and ℓ are linear polynomials [Saha-Saptharishi-Saxena 2010].
- Testing if a quadratic polynomial divides a s-sparse polynomial in $s^{\log s}$ time [Forbes 2015]. Reduces to PIT of sums of monomials times powers of quadratics.
- If f, g are sparse polynomials with bounded individual degrees, [Volkovich 2017] gave a poly-time test.

Sparse Divisibility Testing

- Whether a linear polynomial divides a sparse polynomial can be tested in polynomial time.
- $f(x_1, \ldots, x_n)$ is divisible by $x_1 - \ell(x_2, \ldots, x_n)$ iff $f(\ell, x_2, \ldots, x_n) = 0$.
- Semidiagonal model: $\Sigma m_i \ell^{e_i}$ where m_i is a monomial and ℓ are linear polynomials [Saha-Saptharishi-Saxena 2010].
- Testing if a quadratic polynomial divides a s-sparse polynomial in $s^{\log s}$ time [Forbes 2015]. Reduces to PIT of sums of monomials times powers of quadratics.
- If f, g are sparse polynomials with bounded individual degrees, [Volkovich 2017] gave a poly-time test.

- Whether a linear polynomial divides a sparse polynomial can be tested in polynomial time.
- $f(x_1, \ldots, x_n)$ is divisible by $x_1 - \ell(x_2, \ldots, x_n)$ iff $f(\ell, x_2, \ldots, x_n) = 0$.
- Semidiagonal model: $\Sigma m_i \ell^{e_i}$ where m_i is a monomial and ℓ are linear polynomials [Saha-Saptharishi-Saxena 2010].
- Testing if a quadratic polynomial divides a s-sparse polynomial in $s^{\log s}$ time [Forbes 2015]. Reduces to PIT of sums of monomials times powers of quadratics.
- If f, g are sparse polynomials with bounded individual degrees, [Volkovich 2017] gave a poly-time test.

- Given sparse polynomials $f, g_1, \ldots, g_k$, test if $f = \prod_{i=1}^{k} g_i$. Or more generally, $f = \prod_i g_i^{e_i}$.
- Testing in deterministic polynomial time open, even when g_i have bounded degree.
- More general question: Given sparse polynomials $f_1, \ldots, f_k$ and $g_1, \ldots, g_r$, test if $\prod_{i=1}^{r} f_i = \prod_{i=1}^{k} g_i$.
- Bisht and Volkovich recently solved a related question. They assume f_i, g_i are sparse and have bounded individual degrees.

Factorization Testing

- Given sparse polynomials $f, g_1, \ldots, g_k$, test if $f = \prod_{i=1}^{k} g_i$. Or more generally, $f = \prod_i g_i^{e_i}$.
- Testing in deterministic polynomial time open, even when g_i have bounded degree.
- More general question: Given sparse polynomials $f_1, \ldots, f_k$ and $g_1, \ldots, g_r$, test if $\prod_{i=1}^{r} f_i = \prod_{i=1}^{k} g_i$.
- Bisht and Volkovich recently solved a related question. They assume f_i, g_i are sparse and have bounded individual degrees.

- Given sparse polynomials $f, g_1, \ldots, g_k$, test if $f = \prod_{i=1}^{k} g_i$. Or more generally, $f = \prod_i g_i^{e_i}$.
- Testing in deterministic polynomial time open, even when g_i have bounded degree.
- More general question: Given sparse polynomials $f_1, \ldots, f_k$ and $g_1, \ldots, g_r$, test if $\prod_{i=1}^{r} f_i = \prod_{i=1}^{k} g_i$.
- Bisht and Volkovich recently solved a related question. They assume f_i, g_i are sparse and have bounded individual degrees.

- Given sparse polynomials $f, g_1, \ldots, g_k$, test if $f = \prod_{i=1}^k g_i$. Or more generally, $f = \prod_i g_i^{e_i}$.
- Testing in deterministic polynomial time open, even when g_i have bounded degree.
- More general question: Given sparse polynomials $f_1, \ldots, f_k$ and $g_1, \ldots, g_r$, test if $\prod_{i=1}^r f_i = \prod_{i=1}^k g_i$.
- Bisht and Volkovich recently solved a related question. They assume f_i, g_i are sparse and have bounded individual degrees.

- Derandomization of Multivariate Factoring over $\mathbb{Q}$ reduces to derandomization of PIT [Kopparty-Saraf-Shpilka 2015].
- This is known in both black-box and white-box settings.
- Can we derandomize factoring in some special cases, such as sparse polynomials?

- Derandomization of Multivariate Factoring over $\mathbb{Q}$ reduces to derandomization of PIT [Kopparty-Saraf-Shpilka 2015].
- This is known in both black-box and white-box settings.
- Can we derandomize factoring in some special cases, such as sparse polynomials?

- Derandomization of Multivariate Factoring over $\mathbb{Q}$ reduces to derandomization of PIT [Kopparty-Saraf-Shpilka 2015].
- This is known in both black-box and white-box settings.
- Can we derandomize factoring in some special cases, such as sparse polynomials?

- Given an n-variate degree d polynomial of sparsity $\leq s$, test if it is irreducible in deterministic $\text{POLY}(n, s, d)$ time.
- Challenge: Currently, it requires PIT for symbolic Determinants.
- Given two n-variate degree d polynomial of sparsity $\leq s$, test if they are coprime in deterministic $\text{POLY}(n, s, d)$ time.
- Challenge: The resultant of two sparse polynomials may not be sparse.

- Given an n-variate degree d polynomial of sparsity $\leq s$, test if it is irreducible in deterministic $\text{POLY}(n, s, d)$ time.
- Challenge: Currently, it requires PIT for symbolic Determinants.
- Given two n-variate degree d polynomial of sparsity $\leq s$, test if they are coprime in deterministic $\text{POLY}(n, s, d)$ time.
- Challenge: The resultant of two sparse polynomials may not be sparse.

Frontier questions

- Given an n-variate degree d polynomial of sparsity $\leq s$, test if it is irreducible in deterministic $\textsc{poly}(n, s, d)$ time.
- Challenge: Currently, it requires PIT for symbolic Determinants.
- Given two n-variate degree d polynomial of sparsity $\leq s$, test if they are coprime in deterministic $\textsc{poly}(n, s, d)$ time.
- Challenge: The resultant of two sparse polynomials may not be sparse.

Frontier questions

- Given an n-variate degree d polynomial of sparsity $\leq s$, test if it is irreducible in deterministic $\text{POLY}(n, s, d)$ time.
- Challenge: Currently, it requires PIT for symbolic Determinants.
- Given two n-variate degree d polynomial of sparsity $\leq s$, test if they are coprime in deterministic $\text{POLY}(n, s, d)$ time.
- Challenge: The resultant of two sparse polynomials may not be sparse.

Black-box case: Kaltofen and Trager

- Given a black box computing a multivariate polynomial f, black boxes of the irreducible factors of f can be computed in randomized polynomial time [Kaltofen-Trager 1991].

- Dimension reduction: Randomly project to bivariates.

- This works due to an effective version of Hilbert's irreducibility theorem.

- If $f(x, z_1, \ldots, z_n)$ is irreducible, then $f(x, \beta_1 + \alpha_1 y, \ldots, \beta_n + \alpha_n y)$ is irreducible with high probability if β_i, α_i picked at random.

- Currently, derandomization of this theorem for sparse polynomials reduces to ABP PIT.

- Given a black box computing a multivariate polynomial f, black boxes of the irreducible factors of f can be computed in randomized polynomial time [Kaltofen-Trager 1991].
- Dimension reduction: Randomly project to bivariates.
- This works due to an effective version of Hilbert's irreducibility theorem.
- If $f(x, z_1, \ldots, z_n)$ is irreducible, then $f(x, \beta_1 + \alpha_1 y, \ldots, \beta_n + \alpha_n y)$ is irreducible with high probability if β_i, α_i picked at random.
- Currently, derandomization of this theorem for sparse polynomials reduces to ABP PIT.

- Given a black box computing a multivariate polynomial f, black boxes of the irreducible factors of f can be computed in randomized polynomial time [Kaltofen-Trager 1991].
- Dimension reduction: Randomly project to bivariates.
- This works due to an effective version of Hilbert's irreducibility theorem.
- If $f(x, z_1, \ldots, z_n)$ is irreducible, then $f(x, \beta_1 + \alpha_1 y, \ldots, \beta_n + \alpha_n y)$ is irreducible with high probability if β_i, α_i picked at random.
- Currently, derandomization of this theorem for sparse polynomials reduces to ABP PIT.

Black-box case: Kaltofen and Trager

- Given a black box computing a multivariate polynomial f, black boxes of the irreducible factors of f can be computed in randomized polynomial time [Kaltofen-Trager 1991].
- Dimension reduction: Randomly project to bivariates.
- This works due to an effective version of Hilbert's irreducibility theorem.
- If $f(x, z_1, \ldots, z_n)$ is irreducible, then $f(x, \beta_1 + \alpha_1 y, \ldots, \beta_n + \alpha_n y)$ is irreducible with high probability if β_i, α_i picked at random.
- Currently, derandomization of this theorem for sparse polynomials reduces to ABP PIT.

- Given a black box computing a multivariate polynomial f, black boxes of the irreducible factors of f can be computed in randomized polynomial time [Kaltofen-Trager 1991].
- Dimension reduction: Randomly project to bivariates.
- This works due to an effective version of Hilbert's irreducibility theorem.
- If $f(x, z_1, \ldots, z_n)$ is irreducible, then $f(x, \beta_1 + \alpha_1 y, \ldots, \beta_n + \alpha_n y)$ is irreducible with high probability if β_i, α_i picked at random.
- Currently, derandomization of this theorem for sparse polynomials reduces to ABP PIT.

- $f(x, z_1, \ldots, z_n)$ may not be monic in x.
- Apply the shift: $z_i \mapsto z_i + \alpha_i x$ where α_i picked at random.
- Say, factors $g(x, z_1, \ldots, z_n), h(x, z_1, \ldots, z_n)$ are coprime. But $g(x, 0, \ldots, 0)$ and $h(x, 0, \ldots, 0)$ are not coprime.
- If the resultant (determinant of Sylvester matrix) of them is nonzero at some point $\alpha_1, \ldots, \alpha_n$, translate by that point.
- We need PIT also in the linear system-solving step.

White-box case

- $f(x, z_1, \ldots, z_n)$ may not be monic in x.
- Apply the shift: $z_i \mapsto z_i + \alpha_i x$ where α_i picked at random.
- Say, factors $g(x, z_1, \ldots, z_n), h(x, z_1, \ldots, z_n)$ are coprime. But $g(x, 0, \ldots, 0)$ and $h(x, 0, \ldots, 0)$ are not coprime.
- If the resultant (determinant of Sylvester matrix) of them is nonzero at some point $\alpha_1, \ldots, \alpha_n$, translate by that point.
- We need PIT also in the linear system-solving step.

- $f(x, z_1, \ldots, z_n)$ may not be monic in x.
- Apply the shift: $z_i \mapsto z_i + \alpha_i x$ where α_i picked at random.
- Say, factors $g(x, z_1, \ldots, z_n), h(x, z_1, \ldots, z_n)$ are coprime. But $g(x, 0, \ldots, 0)$ and $h(x, 0, \ldots, 0)$ are not coprime.
- If the resultant (determinant of Sylvester matrix) of them is nonzero at some point $\alpha_1, \ldots, \alpha_n$, translate by that point.
- We need PIT also in the linear system-solving step.

White-box case

- $f(x, z_1, \ldots, z_n)$ may not be monic in x.
- Apply the shift: $z_i \mapsto z_i + \alpha_i x$ where α_i picked at random.
- Say, factors $g(x, z_1, \ldots, z_n), h(x, z_1, \ldots, z_n)$ are coprime. But $g(x, 0, \ldots, 0)$ and $h(x, 0, \ldots, 0)$ are not coprime.
- If the resultant (determinant of Sylvester matrix) of them is nonzero at some point $\alpha_1, \ldots, \alpha_n$, translate by that point.
- We need PIT also in the linear system-solving step.

White-box case

- $f(x, z_1, \ldots, z_n)$ may not be monic in x.
- Apply the shift: $z_i \mapsto z_i + \alpha_i x$ where α_i picked at random.
- Say, factors $g(x, z_1, \ldots, z_n), h(x, z_1, \ldots, z_n)$ are coprime. But $g(x, 0, \ldots, 0)$ and $h(x, 0, \ldots, 0)$ are not coprime.
- If the resultant (determinant of Sylvester matrix) of them is nonzero at some point $\alpha_1, \ldots, \alpha_n$, translate by that point.
- We need PIT also in the linear system-solving step.

- Koiran and Ressyare (2018): Test if $f(x_1, \ldots, x_n)$ is of the form $f(x) = \ell_1(x)^{\alpha_1} \cdots \ell_n(x)^{\alpha_n}$. If yes, output the linear factors.

- They give three randomized algorithms that are different from Kaltofen and Trager's algorithm.

- Koiran and Ressyare (2018): Test if $f(x_1, \ldots, x_n)$ is of the form $f(x) = \ell_1(x)^{\alpha_1} \cdots \ell_n(x)^{\alpha_n}$. If yes, output the linear factors.

- They give three randomized algorithms that are different from Kaltofen and Trager's algorithm.

- The first one uses the characterization of the Lie algebras of the polynomials in the orbit of a monomial.
- The second algorithm reconstructs a factorization from several bivariate projections.
- The third algorithm reconstructs it from the determination of the zero set of the input polynomial, which is a union of hyperplanes.

- The first one uses the characterization of the Lie algebras of the polynomials in the orbit of a monomial.
- The second algorithm reconstructs a factorization from several bivariate projections.
- The third algorithm reconstructs it from the determination of the zero set of the input polynomial, which is a union of hyperplanes.

- The first one uses the characterization of the Lie algebras of the polynomials in the orbit of a monomial.
- The second algorithm reconstructs a factorization from several bivariate projections.
- The third algorithm reconstructs it from the determination of the zero set of the input polynomial, which is a union of hyperplanes.

- Given a black box computing product of linear/bounded degree polynomials, output the factors in polynomial time.
- Over $\mathbb{Q}$, this can done.
- Work under progress: Given a black-box computing product of sparse polynomials with bounded individual degrees, output factors in polynomial time.
- Note that we cannot directly use Bhargava-Saraf-Volkovich: They assume the input is sparse and have bounded individual degree.

- Given a black box computing product of linear/bounded degree polynomials, output the factors in polynomial time.
- Over $\mathbb{Q}$, this can done.
- Work under progress: Given a black-box computing product of sparse polynomials with bounded individual degrees, output factors in polynomial time.
- Note that we cannot directly use Bhargava-Saraf-Volkovich: They assume the input is sparse and have bounded individual degree.

- Given a black box computing product of linear/bounded degree polynomials, output the factors in polynomial time.
- Over $\mathbb{Q}$, this can done.
- Work under progress: Given a black-box computing product of sparse polynomials with bounded individual degrees, output factors in polynomial time.
- Note that we cannot directly use Bhargava-Saraf-Volkovich: They assume the input is sparse and have bounded individual degree.

- Open: All the factors of size s formulas have size $\mathrm{POLY}(s)$ formulas?
- If not, what are the candidate counterexamples?
- Hensel lifting/Newton iteration may have further applications in algebraic complexity.
- Open: Given a black-box that computes product of two sparse polynomials, output the sparse factors in deterministic polynomial time.

Conclusion

- Open: All the factors of size s formulas have size $\text{POLY}(s)$ formulas?
- If not, what are the candidate counterexamples?
- Hensel lifting/Newton iteration may have further applications in algebraic complexity.
- Open: Given a black-box that computes product of two sparse polynomials, output the sparse factors in deterministic polynomial time.

Conclusion

- Open: All the factors of size s formulas have size $\text{POLY}(s)$ formulas?
- If not, what are the candidate counterexamples?
- Hensel lifting/Newton iteration may have further applications in algebraic complexity.
- Open: Given a black-box that computes product of two sparse polynomials, output the sparse factors in deterministic polynomial time.

Conclusion

- Open: All the factors of size s formulas have size $\textsc{poly}(s)$ formulas?
- If not, what are the candidate counterexamples?
- Hensel lifting/Newton iteration may have further applications in algebraic complexity.
- Open: Given a black-box that computes product of two sparse polynomials, output the sparse factors in deterministic polynomial time.

Thank You!