gct2022 : School and Conference on Geometric Complexity Theory

17-28 Jan 2022 Chennai (India)

Algebraic Complexity: Upper bounds

Tavenas

October 5th, 2021

Why to look at upper bounds?

- Main point of algorithmic!

Why to look at upper bounds?

- Main point of algorithmic!
- The theme of this school is to analyze why some polynomials are "hard".

Why to look at upper bounds?

- Main point of algorithmic!
- The theme of this school is to analyze why some polynomials are “hard”.
- Interesting to undestand why some polynomials are “easy”.

Overview

1. Upper bounds
   - First algorithms
   - Multiplications
2. Reductions
   - Classes VBP, VP, VF
   - Counting problems
3. An efficient algorithm calls for many others
   - Homogeneous Components / Derivatives
   - Closures

First "counterintuitive" algorithms

- Recalls Elementary symmetric polynomials (Srikanth's talk)
  - E_n^d projection of $\mathsf{IMM}_{d,n}$ (Dynamic programming)
  - E_n^d has small formulas (Interpolation).

First "counterintuitive" algorithms

- Recalls Elementary symmetric polynomials (Srikanth's talk)
  - E_n^d projection of $\mathrm{IMM}_{d,n}$ (Dynamic programming)
  - E_n^d has small formulas (Interpolation).
- Computing any univariate $P \in \mathbb{C}[X]$ of degree d:

First “counterintuitive” algorithms

- Recalls Elementary symmetric polynomials (Srikanth's talk)
  - E_n^d projection of $\mathsf{IMM}_{d,n}$ (Dynamic programming)
  - E_n^d has small formulas (Interpolation).
- Computing any univariate $P \in \mathbb{C}[X]$ of degree d:
  $P = \sum^d a_i X^i = (\cdots((a_d X + a_{d-1})X + a_{d-2})\cdots)X + a_0$

First “counterintuitive” algorithms

- Recalls Elementary symmetric polynomials (Srikanth's talk)
  - E_n^d projection of $\mathsf{IMM}_{d,n}$ (Dynamic programming)
  - E_n^d has small formulas (Interpolation).
- Computing any univariate $P \in \mathbb{C}[X]$ of degree d:
  $P = \sum^d a_i X^i = (\cdots((a_d X + a_{d-1})X + a_{d-2})\cdots)X + a_0$
  Horner's rule: d multiplications and d additions. Best?

First "counterintuitive" algorithms

- Recalls Elementary symmetric polynomials (Srikanth's talk)
  - E_n^d projection of $\mathrm{IMM}_{d,n}$ (Dynamic programming)
  - E_n^d has small formulas (Interpolation).
- Computing any univariate $P \in \mathbb{C}[X]$ of degree d:
  $P = \sum^d a_i X^i = (\cdots((a_d X + a_{d-1})X + a_{d-2})\cdots)X + a_0$
  Horner's rule: d multiplications and d additions. Best?
  Possible with $d+1$ additions and $1 + d/2$ multiplications!
  [Knuth,Eve]

$$P(X) = (X^2 - u^2)\underbrace{Q_u(X)}_{\deg = d-2} + \underbrace{R_u(X)}_{\deg \leq 1}.$$

We want $\deg(R_u) = 0$

$\iff P(u) = P(-u) \iff u$ root of $\sum_{i \mid i \equiv 1[2]} a_i X^i$

$\Rightarrow P_d(X) = (X^2 - v) P_{d-2}(X) + w$

Multiplication of polynomials

Let $P_a = \sum^d a_i X^i, P_b = \sum^d b_i X^i \in \mathbb{C}[X]$
Goal: compute $P_c(X) = \sum^{2d} c_i X^i = P_a(X) \cdot P_b(X)$.

Multiplication of polynomials

Let $P_a = \sum^d a_i X^i, P_b = \sum^d b_i X^i \in \mathbb{C}[X]$
Goal: compute $P_c(X) = \sum^{2d} c_i X^i = P_a(X) \cdot P_b(X)$. So compute $(c_i)_{i \leq 2d}$.

- Developing everything:

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} \quad \Rightarrow \text{Sums: } (d+1)^2 \text{ mult.}$$
$$d^2 \text{ additions.}$$

Multiplication of polynomials

Let $P_a = \sum^d a_i X^i, P_b = \sum^d b_i X^i \in \mathbb{C}[X]$
Goal: compute $P_c(X) = \sum^{2d} c_i X^i = P_a(X) \cdot P_b(X)$. So compute $(c_i)_{i \leq 2d}$.

- Developing everything: $(d+1)^2$ mult. & d^2 add.

Multiplication of polynomials

Let $P_a = \sum^d a_i X^i, P_b = \sum^d b_i X^i \in \mathbb{C}[X]$
Goal: compute $P_c(X) = \sum^{2d} c_i X^i = P_a(X) \cdot P_b(X)$. So compute $(c_i)_{i \leq 2d}$.

- Developing everything: $(d+1)^2$ mult. & d^2 add.
- Karatsuba's algorithm:

Take $H = X^{\lfloor d/2 \rfloor}$

$$P = P_1 H + P_2$$

$$Q = Q_1 H + Q_2$$

$$PQ = \underbrace{(P_1 Q_1)}_{1} H^2 + \underbrace{(P_1 Q_2 + P_2 Q_1)}_{} H + \underbrace{P_2 Q_2}_{1}$$

$$= \underbrace{(P_1 + P_2)(Q_1 + Q_2)}_{1} - P_1 Q_1 - P_2 Q_2$$

$$N_d \leq 3 N_{d/2}$$

(N_k : # multiplications for computing the prod of 2 poly of deg k)

$$\Rightarrow N_d = O(n^{\log_2 3})$$

Multiplication of polynomials

Let $P_a = \sum^d a_i X^i, P_b = \sum^d b_i X^i \in \mathbb{C}[X]$
Goal: compute $P_c(X) = \sum^{2d} c_i X^i = P_a(X) \cdot P_b(X)$. So compute $(c_i)_{i \leq 2d}$.

- Developing everything: $(d+1)^2$ mult. & d^2 add.
- Karatsuba's algorithm: $3d^{\log_2 3}$ mult.
- Fast Fourier Transform:

Idea: $N > 2d$ sufficient to solve the product in $\mathbb{C}[X]/(X^N - 1)$

→ Choose $N = 2^p$, ω a N^{th} primitive root.

$$D_N \quad \mathbb{C}[X]/(X^N-1) \longrightarrow \mathbb{C}^N \qquad \text{isom of algebras}$$
$$P \longmapsto (P(\omega^0), \dots, P(\omega^{N-1}))$$

$$P_c = D_N^{-1}(D_N(P_a) \cdot D_N(P_b))$$

Multiplication of polynomials

Let $P_a = \sum^d a_i X^i, P_b = \sum^d b_i X^i \in \mathbb{C}[X]$
Goal: compute $P_c(X) = \sum^{2d} c_i X^i = P_a(X) \cdot P_b(X)$. So compute $(c_i)_{i \leq 2d}$.

- Developing everything: $(d+1)^2$ mult. & d^2 add.
- Karatsuba's algorithm: $3d^{\log_2 3}$ mult.
- Fast Fourier Transform:

$$D_N\left(\sum a_i X^i\right) = \underbrace{\begin{pmatrix} \omega^0 & \omega^0 & \\ \omega^0 & \omega^1 & \\ & & \ddots \; \omega^{ij} \end{pmatrix}}_{DFT_N} \begin{pmatrix} a_0 \\ \vdots \\ a_d \end{pmatrix}$$

$$DFT^{-1} = \frac{1}{N}\left((\omega^{-1})^{ij} \right)$$

$$\forall i : \sum_j a_j \omega^{ij} = \left(\sum_{j<N/2} a_{2j} (\omega^2)^{ij}\right) + \omega^i \left(\sum_{j<N/2} a_{2j+1} (\omega^2)^{ij}\right)$$

$$DFT_N \cdot a = \begin{pmatrix} DFT_{N/2} & \Delta DFT_{N/2} \\ DFT_{N/2} & -\Delta DFT_{N/2} \end{pmatrix} \cdot \begin{pmatrix} ev(a) \\ odd(a) \end{pmatrix}$$

$$T_N = 2T_{N/2} + O(N) \Rightarrow T_N = O(N \log N)$$

Multiplication of polynomials

Let $P_a = \sum^d a_i X^i, P_b = \sum^d b_i X^i \in \mathbb{C}[X]$
Goal: compute $P_c(X) = \sum^{2d} c_i X^i = P_a(X) \cdot P_b(X)$. So compute $(c_i)_{i \leq 2d}$.

- Developing everything: $(d+1)^2$ mult. & d^2 add.
- Karatsuba's algorithm: $3d^{\log_2 3}$ mult.
- Fast Fourier Transform: $O(d \log d)$
- [Schönage, Strassen] Over $\mathbb{Z}, \mathbb{Q}$: $O(d(\log d)(\log \log d))$.

Multiplication of polynomials

Let $P_a = \sum^d a_i X^i, P_b = \sum^d b_i X^i \in \mathbb{C}[X]$
Goal: compute $P_c(X) = \sum^{2d} c_i X^i = P_a(X) \cdot P_b(X)$. So compute $(c_i)_{i \leq 2d}$.

- Developing everything: $(d+1)^2$ mult. & d^2 add.
- Karatsuba's algorithm: $3d^{\log_2 3}$ mult.
- Fast Fourier Transform: $O(d \log d)$
- [Schönage, Strassen] Over $\mathbb{Z}, \mathbb{Q}$: $O(d(\log d)(\log\log d))$.

Conclusion: multiplication of u, v integers (with $u < v$) in $O((\log v)(\log\log v)(\log\log\log v))$.

Multiplication of matrices

$$\begin{pmatrix} X_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \cdots & X_{n,n} \end{pmatrix} = \begin{pmatrix} Y_{1,1} & \cdots & Y_{1,n} \\ \vdots & \ddots & \vdots \\ Y_{n,1} & \cdots & Y_{n,n} \end{pmatrix} \cdot \begin{pmatrix} Z_{1,1} & \cdots & Z_{1,n} \\ \vdots & \ddots & \vdots \\ Z_{n,1} & \cdots & Z_{n,n} \end{pmatrix}$$

$$X_{i,j} = \sum_{k=1}^{n} Y_{i,k} Z_{k,j} \qquad (1 \leq i, j \leq n)$$

Multiplication of matrices

$$\begin{pmatrix} X_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \cdots & X_{n,n} \end{pmatrix} = \begin{pmatrix} Y_{1,1} & \cdots & Y_{1,n} \\ \vdots & \ddots & \vdots \\ Y_{n,1} & \cdots & Y_{n,n} \end{pmatrix} \cdot \begin{pmatrix} Z_{1,1} & \cdots & Z_{1,n} \\ \vdots & \ddots & \vdots \\ Z_{n,1} & \cdots & Z_{n,n} \end{pmatrix}$$

$$X_{i,j} = \sum_{k=1}^{n} Y_{i,k} Z_{k,j} \qquad (1 \leq i, j \leq n)$$

- Full decomposition

Multiplication of matrices

$$\begin{pmatrix} X_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \cdots & X_{n,n} \end{pmatrix} = \begin{pmatrix} Y_{1,1} & \cdots & Y_{1,n} \\ \vdots & \ddots & \vdots \\ Y_{n,1} & \cdots & Y_{n,n} \end{pmatrix} \cdot \begin{pmatrix} Z_{1,1} & \cdots & Z_{1,n} \\ \vdots & \ddots & \vdots \\ Z_{n,1} & \cdots & Z_{n,n} \end{pmatrix}$$

$$X_{i,j} = \sum_{k=1}^{n} Y_{i,k} Z_{k,j} \qquad (1 \leq i, j \leq n)$$

- Full decomposition $O(n^3)$

Multiplication of matrices

$$\begin{pmatrix} X_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \cdots & X_{n,n} \end{pmatrix} = \begin{pmatrix} Y_{1,1} & \cdots & Y_{1,n} \\ \vdots & \ddots & \vdots \\ Y_{n,1} & \cdots & Y_{n,n} \end{pmatrix} \cdot \begin{pmatrix} Z_{1,1} & \cdots & Z_{1,n} \\ \vdots & \ddots & \vdots \\ Z_{n,1} & \cdots & Z_{n,n} \end{pmatrix}$$

$$X_{i,j} = \sum_{k=1}^{n} Y_{i,k} Z_{k,j} \qquad (1 \leq i, j \leq n)$$

- Full decomposition $\qquad O(n^3)$
- Strassen's algorithm

Strassen's algorithm

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$$

Strassen's algorithm

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$$\begin{aligned}
M_1 &= A \times (F - H) \\
M_2 &= (A + B) \times H \\
M_3 &= (C + D) \times E \\
M_4 &= D \times (G - E) \\
M_5 &= (A + D) \times (E + H) \\
M_6 &= (B - D) \times (G - H) \\
M_7 &= (A - C) \times (E + F)
\end{aligned}$$

Strassen's algorithm

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} -M_2 + M_4 + M_5 + M_6 & M_1 + M_2 \\ M_3 + M_4 & M_1 - M_3 + M_5 - M_7 \end{pmatrix}$$

$$\begin{aligned}
M_1 &= A \times (F - H) \\
M_2 &= (A + B) \times H \\
M_3 &= (C + D) \times E \\
M_4 &= D \times (G - E) \\
M_5 &= (A + D) \times (E + H) \\
M_6 &= (B - D) \times (G - H) \\
M_7 &= (A - C) \times (E + F)
\end{aligned}$$

Multiplication of matrices

$$\begin{pmatrix} X_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \cdots & X_{n,n} \end{pmatrix} = \begin{pmatrix} Y_{1,1} & \cdots & Y_{1,n} \\ \vdots & \ddots & \vdots \\ Y_{n,1} & \cdots & Y_{n,n} \end{pmatrix} \cdot \begin{pmatrix} Z_{1,1} & \cdots & Z_{1,n} \\ \vdots & \ddots & \vdots \\ Z_{n,1} & \cdots & Z_{n,n} \end{pmatrix}$$

$$X_{i,j} = \sum_{k=1}^{n} Y_{i,k} Z_{k,j} \qquad (1 \leq i, j \leq n)$$

- Full decomposition $O(n^3)$
- Strassen's algorithm $O(n^{\log_2 7}) = O(n^{2.81})$

Multiplication of matrices

$$\begin{pmatrix} X_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \cdots & X_{n,n} \end{pmatrix} = \begin{pmatrix} Y_{1,1} & \cdots & Y_{1,n} \\ \vdots & \ddots & \vdots \\ Y_{n,1} & \cdots & Y_{n,n} \end{pmatrix} \cdot \begin{pmatrix} Z_{1,1} & \cdots & Z_{1,n} \\ \vdots & \ddots & \vdots \\ Z_{n,1} & \cdots & Z_{n,n} \end{pmatrix}$$

$$X_{i,j} = \sum_{k=1}^{n} Y_{i,k} Z_{k,j} \qquad (1 \leq i, j \leq n)$$

- Full decomposition $O(n^3)$
- Strassen's algorithm $O(n^{\log_2 7}) = O(n^{2.81})$
- [Schönhage,Coppersmith-Winograd,Williams,Le Gall] $O(n^{2.3728596})$

IMM to Determinant

Recall: IMM can be seen as an Algebraic Branching Program (ABP).
[Damm, Vinay, Toda]
Idea seen in Avi's talk.

$$p = \overbrace{\begin{bmatrix} X & Y & Z \end{bmatrix}}^{n_1} \times \overbrace{\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}}^{n_2} \times \overbrace{\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}}^{n_3} \times \begin{bmatrix} u \\ v \end{bmatrix}$$

$(-1)^{j+1}$

$$\det \begin{pmatrix} x & Y & z & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & a & b & 0 & 0 & 0 \\ 0 & 1 & 0 & c & d & 0 & 0 & 0 \\ 0 & 0 & 1 & e & f & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \alpha & \beta & 0 \\ 0 & 0 & 0 & 0 & 1 & \gamma & \delta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & u \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & v \end{pmatrix} \overset{?}{=} (\pm 1)\, p$$

$(-1)^{n_1+j}$

$(-1)^{n_1-j}$

$(-1)^{n_1-1}$

$\rightarrow (-1)^{(\sum n_i)+d-1}$

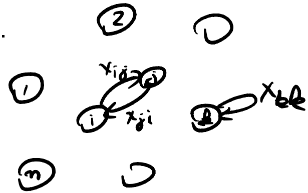
Determinant to IMM

$$\mathrm{Det}(A) = \sum_{\sigma \in S_n} (-1)^{\varepsilon(\sigma)} \prod_{i=1}^{n} x_{i\sigma_i}$$

[Berkowitz]
Here, we follow [MV97].

$$= (-1)^n \sum_{\substack{\text{Ucycles} \\ \text{covering } n}} (-1)^{\#\text{cycles}} \prod_{i=1}^{n} x_{i,c_i}$$

$\mathrm{Det}(A) \longrightarrow$



$\Rightarrow O(n^3)$

Claws: closed walk in the graph $(i_1, \ldots, i_p) \cdot i_1 = i_p \ \& \ \forall j \ i_j > i_1$

claws seq: $(C_1, \ldots, C_q)$ st $\mathrm{head}(C_i) < \mathrm{head}(C_j)$ as a $i<j$.

$$\sum_{\substack{\text{cycles} \\ \text{cov}}} (-1)^{\#\text{cycles}} \prod_{i=1}^{n} x_{i,c_i} = \sum_{\substack{\text{claws seq} \\ \sum |C_i| = n}} (-1)^{\#\text{claws}} \prod_{i=1}^{n} x_{i,c_i}$$

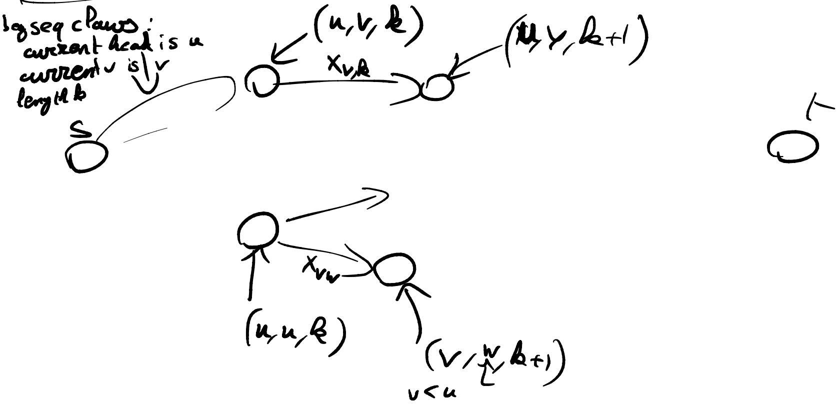
$(C_1, \ldots, C_i)$ $\downarrow$

$*$ $C_i \cap C_j$ $(j<i)$

$*$

Determinant to IMM

[Berkowitz]
Here, we follow [MV97].



From Circuits to Formulas

$$\mathrm{VF} \subseteq \mathrm{VBP} \subseteq \mathrm{VP}$$

If $P \in \mathbb{F}_d[X_1, \ldots, X_n]$ is computed by a sized-s circuit, then it is computed by a formula of size $s^{O(\log d)}$.

From Circuits to Formulas

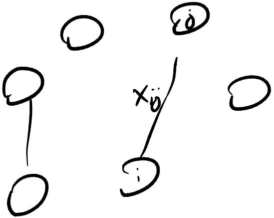
If $P \in \mathbb{F}_d[X_1, \ldots, X_n]$ is computed by a sized-s circuit, then it is computed by a formula of size $s^{O(\log d)}$.
Consequence of [VSBR]. We will see it in the next talk.

Counting problems

$x_{ij} = x_{ji}$

Counting the number of:

- cliques,



$$Clique_n^* = \sum_{A \subseteq [n]} \prod_{\substack{(i,j) \in A^2 \\ i<j}} x_{ij}$$

$$Clique_n^*(x_{12}t, x_{13}t, \dots) \in \mathbb{C}[\bar{x}][t]$$

$$[t^{\binom{p}{2}}]\, Clique_n^* \Leftrightarrow \text{cliques of size } p$$

Counting problems

Counting the number of:

- cliques,
- Hamiltonian circuits,

Counting problems

Counting the number of:

- cliques,
- Hamiltonian circuits,
- perfect matchings in a bipartite graph, $\longrightarrow$ We obtain Perm

Counting problems

Counting the number of:

- cliques,
- Hamiltonian circuits,
- matchings in a bipartite graph,

[Valiant] Perm is VNP-complete ($\mathrm{char}(\mathbb{F}) \neq 2$).

Counting problems

Counting the number of:

- cliques,
- Hamiltonian circuits,
- matchings in a bipartite graph,

[Valiant] Perm is VNP-complete ($\mathrm{char}(\mathbb{F}) \neq 2$).
It means that if Perm is easy, all are easy.

Counting problems

Counting the number of:

- cliques,
- Hamiltonian circuits,
- matchings in a bipartite graph,

$$X = \begin{pmatrix} 0 & & -x_{ij} \\ & \ddots & \\ x_{ij} & & 0 \end{pmatrix}$$

$$\mathrm{Pf}(X) = \sum_{\sigma \in S_{2n}} (-1)^{\varepsilon(\sigma)} \prod_{i=1}^{n} x_{\sigma(2i),\sigma(2i+1)}$$

[Valiant] Perm is VNP-complete (char($\mathbb{F}$) $\neq 2$).
It means that if Perm is easy, all are easy.

$$\rightarrow \mathrm{Pf}(X) = \sqrt{\mathrm{Det}(X)}$$

Some counting problems are easy:

- [Kasteleyn] number of matchings in a planar graph,
- [Kirchhoff] number of trees, ...

Homogeneous components

- Circuits and ABPs are closed by Homogeneization.

> If $P \in \mathbb{F}_d[X_1, \ldots, X_n]$, is computed by a circuit (resp. an ABP) of size s,
> we can compute $(P_0, P_1, \ldots, P_d)$, the homogeneous components, by a circuit (resp. ABP) of size
>
> $$s(\frac{d^2}{2} + 4d) \text{ (resp. } s(d+1))$$

Homogeneous components

- Circuits and ABPs are closed by Homogeneization.

> If $P \in \mathbb{F}_d[X_1, \ldots, X_n]$, is computed by a circuit (resp. an ABP) of size s,
> we can compute $(P_0, P_1, \ldots, P_d)$, the homogeneous components, by a circuit (resp. ABP) of size
>
> $$s(\frac{d^2}{2} + 4d) \text{ (resp. } s(d+1)\text{)}$$

(see in the next talk).

Homogeneous components

- Circuits and ABPs are closed by Homogeneization.

> If $P \in \mathbb{F}_d[X_1, \ldots, X_n]$, is computed by a circuit (resp. an ABP) of size s,
> we can compute $(P_0, P_1, \ldots, P_d)$, the homogeneous components, by a circuit (resp. ABP) of size
>
> $$s(\frac{d^2}{2} + 4d) \text{ (resp. } s(d+1))$$

(see in the next talk).

- Not known for formulas.

Homogeneous components

- Circuits and ABPs are closed by Homogeneization.

  If $P \in \mathbb{F}_d[X_1, \ldots, X_n]$, is computed by a circuit (resp. an ABP) of size s,
  we can compute $(P_0, P_1, \ldots, P_d)$, the homogeneous components, by a circuit (resp. ABP) of size

  $$s(\tfrac{d^2}{2} + 4d) \text{ (resp. } s(d+1))$$

  (see in the next talk).
- Not known for formulas.
  (Ex: Does E_n^d have small homogeneous formulas?)

Derivatives in Linear time

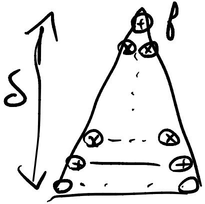
$f \in \mathbb{F}[X_1, \ldots, X_n]$ computed by a sized-s circuit C
(where multiplications have fan-in 2)

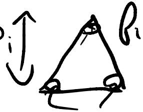
Derivatives in Linear time

$f \in \mathbb{F}[X_1, \ldots, X_n]$ computed by a sized-s circuit C
(where multiplications have fan-in 2)
[Baur-Strassen] $(f, \frac{\partial f}{\partial X_1}, \ldots, \frac{\partial f}{\partial X_n})$ can be computed in $O(s)$!

Derivatives in Linear time

$f \in \mathbb{F}[X_1, \ldots, X_n]$ computed by a sized-s circuit C
(where multiplications have fan-in 2)
[Baur-Strassen] $\left(f, \frac{\partial f}{\partial X_1}, \ldots, \frac{\partial f}{\partial X_n}\right)$ can be computed in $O(s)$!
Ex: A circuit for $\sum_{i=1}^{n} X_i^{k+1}$ gives one of same size for $(X_1^k, \ldots, X_n^k)$.





$f_0 \leftarrow x_0^{(0)}$

$f_0 = x_0^{(0)}$

$\frac{\partial f_i}{\partial x^{(i)}} \to \frac{\partial f_{\delta+1}}{\partial x^{(\delta+1)}} = \frac{\partial f}{\partial x}$

$f_{i+1} = f_i \circ \left(x^{(i)}\left(x^{(i+1)}\right)\right)$

$\frac{\partial f_{i+1}}{\partial x_k^{(i+1)}} = \sum_j \frac{\partial f_i}{\partial x_j^{(i)}} \frac{\partial x_j^{(i)}}{\partial x_k^{(i+1)}}$

$\text{size}_{new} < 3\,\text{size}_{old}$

$\boxed{1^{st}\text{ case}}$ $x_j^{(i)} = \sum_k x_k^{(i+1)}$

$\boxed{2^{nd}\text{ case}}$ $x_j^{(i)} = x_k^{(i+1)} x_\ell^{(i+1)}$

$\frac{\partial f_0}{\partial x_0^{(0)}}$

$\frac{\partial f_i}{\partial x^{(i)}}, \quad \frac{\partial f_i}{\partial x_j^{(i)}} x_k^{(i+1)}, \quad \frac{\partial f_i}{\partial x_j^{(i)}} x_\ell^{(i+1)}$

First closures

The classes VNP, VP, VBP, VF are closed by:

First closures

The classes VNP, VP, VBP, VF are closed by:

- ‘small’ additions, ‘small’ multiplications

First closures

The classes VNP, VP, VBP, VF are closed by:

- 'small' additions, 'small' multiplications
- composition,

First closures

The classes VNP, VP, VBP, VF are closed by:

- ‘small’ additions, ‘small’ multiplications
- ‘composition’
- p-projections.

Closure by taking coefficients?

- Univariate case:

Closure by taking coefficients?

- Univariate case: $O(sd)$ for all classes!
  This is just by interpolation.
  Notice: it contains the case $f \in \mathbb{F}[X_1, \ldots, X_n][Y]$.

Closure by taking coefficients?

- Univariate case: $O(sd)$ for all classes!
  This is just by interpolation.
  Notice: it contains the case $f \in \mathbb{F}[X_1, \ldots, X_n][Y]$.
- Multivariate case:

Closure by taking coefficients?

- Univariate case: $O(sd)$ for all classes!
  This is just by interpolation.
  Notice: it contains the case $f \in \mathbb{F}[X_1, \ldots, X_n][Y]$.
- Multivariate case: Does not seem true for VP, VBP, VF:

$$[t_1 t_2 \cdots t_n] \left(\prod_{i=1}^{n} \sum_{j=1}^{n} X_{i,j} t_i \right) = \mathrm{Perm}(X_{i,j}).$$

Closure by taking coefficients?

- Univariate case: $O(sd)$ for all classes!
  This is just by interpolation.
  Notice: it contains the case $f \in \mathbb{F}[X_1, \ldots, X_n][Y]$.
- Multivariate case: Does not seem true for VP, VBP, VF:

$$[t_1 t_2 \cdots t_n] \left(\prod_{i=1}^{n} \sum_{j=1}^{n} X_{i,j} t_i \right) = \mathrm{Perm}(X_{i,j}).$$

  True for VNP.

Closure by taking coefficients?

- Univariate case: $O(sd)$ for all classes!
  This is just by interpolation.
  Notice: it contains the case $f \in \mathbb{F}[X_1, \ldots, X_n][Y]$.
- Multivariate case: Does not seem true for VP, VBP, VF:

$$[t_1 t_2 \cdots t_n] \left(\prod_{i=1}^{n} \sum_{j=1}^{n} X_{i,j} t_i \right) = \mathrm{Perm}(X_{i,j}).$$

  True for VNP.
  Via the univariate case, possibility to extract some extremals coefficients.

Closure by taking factors

- Let $f = gh$ where $f \in \mathbb{C}_d[X_1, \ldots, X_n]$ computed by a sized-s circuit.

Closure by taking factors

- Let $f = gh$ where $f \in \mathbb{C}_d[X_1, \ldots, X_n]$ computed by a sized-s circuit. Then g is computed by a sized-poly(s) circuit. [Kaltofen]

Closure by taking factors

- Let $f = gh$ where $f \in \mathbb{C}_d[X_1, \ldots, X_n]$ computed by a sized-s circuit. Then g is computed by a sized-poly(s) circuit. [Kaltofen]
- [Sinhababu-Thierauf] VBP also closed by taking factors. But not known for formulas.
- Idea of the proof from [Chou-Kumar-Solomon]

Thank you.