Determinant versus permanent

Markus Bläser, Saarland University

Draft — August 24, 20:13

Chapter 1

Valiant's framework

Throughout this lecture, $\mathbb{F}$ will denote some field, the underlying ground field. Let $X_1, X_2, \ldots$ denote indeterminates over $\mathbb{F}$. An algebraic circuit is a labeled acyclic graphs such that:

1. Every node has either indegree 0 or 2.
2. There is exactly one node of outdegree 0.
3. Every node of degree 0 is labeled with an indeterminant or an element from $\mathbb{F}$.
4. Every other node is either labeled with $+$ or $\times$.

The nodes of indegree 0 are called input nodes (or input gates). The node of outdegree 0 is called the output node. Occassionally, it will be covenient to consider circuits with multiple output nodes. With every node we can associate the polynomial computed at the node. This is done inductively. The polynomial at an input node is the label itself. And for every $+$-node ($\times$-node), the polynomial is the sum (product) of the polynomials computed at the two children. The output of the circuit is the polynomial computed at the output node.

The *size* of a circuit is the number of nodes in it. the *depth* of a circuit is the length of a longest path from an input node to the output node. For a polynomial f, the *complexity* $L(f)$ is the size of a smallest circuit computing f.

For further reading, the reader is referred to [Bür00, vzG87]

1.1 VP

Let $X = X_1, X_2, \ldots$ be an infinite family of indeterminates over some field $\mathbb{F}$. A function $p : \mathbb{N} \to \mathbb{N}$ is *p-bounded*, if there is some polynomial q such that $p(n) \leq q(n)$ for all n.

1.1.1 Definition. *A sequence of polynomials $(f_n) \in \mathbb{F}[X]$ is called a* p-family *if for all n,*

1. *$f_n \in \mathbb{F}[X_1, \ldots, X_{p(n)}]$ for some polynomially bounded function p and*
2. *$\deg f_n \leq q(n)$ for some polynomially bounded function q.*

1.1.2 Definition. *The class* VP *consists of all p-families (f_n) such that $L(f_n)$ is polynomially bounded.*

1.1.3 Example. *Let $\det_n = \sum_{\pi \in \mathfrak{S}_n} \operatorname{sgn}(\pi) X_{1,\pi(1)} \ldots X_{n,\pi(n)}$. We will see soon that $\det_n$ has polynomial-sized arithmetic circuits. Therefore, $(\det_n) \in$ VP.*

In the above example, the indeterminates have two indices instead of one. Of course we could write $\det_n$ as a polynomial in $X_1, X_2, \ldots$ by using a bijection between $\mathbb{N}^2$ and $\mathbb{N}$. However, we prefer the natural naming of the variables (and will do so with other polynomials).

Let $f \in \mathbb{F}[X]$ be a polynomial and $s : X \to \mathbb{F}[X]$ be a mapping that maps indeterminates to polynomials. s can be extended in a unique way to an algebra endomorphism $\mathbb{F}[X] \to \mathbb{F}[X]$. We call s a *substitution*. (Think of the variables replaced by polynomials.)

1.1.4 Definition. *1. Let $f, g \in \mathbb{F}[X]$. f is called a* projection *of g if there is a substitution $r : X \to X \cup \mathbb{F}$ such that $f = r(g)$. We write $f \leq_p g$ in this case. (Since g is a polynomial, it only depends on a finite number of indeterminates. Therefore, we only need to specify a finite part of r.)*

2. Let (f_n) and (g_n) be p-families. (f_n) is a p-projection *of (g_n) if there is a polynomially bounded function $p : \mathbb{N} \to \mathbb{N}$ such that $f_n \leq_p g_{p(n)}$. We write $(f_n) \leq_p (g_n)$.*

Projections are very simple reductions. Therefore, we can also use them to define hardness for "small" complexity classes like VP. Projections fulfill the usual requirements of a reductions:

1.1.5 Lemma. *1. If $(f_n) \leq_p (g_n)$ and $(g_n) \in$ VP, then $(f_n) \in$ VP.*

2. $\leq_p$ is a transitive relation.

Proof. 1. Let q be a polynomially bounded function and p_n be a projection such that $f_n = p_n(g_q(n))$ for all n. Let C_m be a circuit computing g_m. We get a circuit computing f_n by replacing every variable X_i in $C_{q(n)}$ by $p_n(X_i)$. This circuit has the same size as C_n.

2. The composition a two polynomially bounded function is polynomially bounded and the composition of two substitutions is a substitution again.

□

1.1.6 Definition. *1. A p-family (f_n) is called* VP-hard *(under p-projections) if $(g_n) \leq_p (f_n)$ for all $(g_n) \in$ VP.*

2. It is called VP-complete *if in addition $(f_n) \in$ VP.*

1.1.7 Lemma. *If (f_n) is* VP*-hard and $(f_n) \leq_p (g_n)$, then (g_n) is* VP*-hard, too.*

Proof. Let $(h_n) \in$ VP be arbitrary. Since (f_n) is VP-hard, $(h_n) \leq_p (f_n)$. By transitivity, $(h_n) \leq_p (g_n)$. Since (h_n) was arbitrary, the VP-hardness of (g_n) follows. □

Let $X_{i,j}^{(\ell)}$, $1 \leq i, j, \ell \leq n$, be indeterminates and let $M_\ell = (X_{i,j}^{(\ell)})_{1 \leq i,j \leq n}$ for $1 \leq \ell \leq n$. The polynomial imm_n is a polynomial in n^3 variables and is the $(1,1)$ entry of the matrix product $M_1 \cdots M_n$. The p-family $\mathrm{imm} = (\mathrm{imm}_n)$ is the called *iterated matrix multiplication*. By using the trivial algorithm for matrix multiplication, it is easy to see that imm $\in$ VP. We will see in the next chapter that imm and det are equivalent under p-projections. We do not know whether the determinant (or the iterated matrix multiplication polynomial) is VP-complete. However, there are generic problems that are VP-complete. But also more natural complete problems are known, see [DMM$^+$14].

1.1.8 Exercise. *(**, but a little bit tedious) Let $S_{n,i,j}$, $P_{n,i,j}$, and C_n, $n, i, j \in \mathbb{N}$, be indeterminates. Let*

$$\begin{aligned}\mathrm{gen}_1 &= C_1\\ \mathrm{gen}_n &= \sum_{i,j=1}^{n-1} S_{n,i,j}(\mathrm{gen}_i + \mathrm{gen}_j)\\ &\quad + \sum_{i,j=1}^{n-1} P_{n,i,j}\mathrm{gen}_i \cdot \mathrm{gen}_j\\ &\quad + C_n\end{aligned}$$

Prove that (gen_n) *is* VP*-hard. (This family is not in* VP*, since it can simulate arbitrary circuits of polynomial size. Such circuit can compute polynomials of exponential degree and therefore, the degree of* gen_n *is exponential. One can modify this construction to get a family in* VP*, however, we need to prove normal forms for circuits.)*

1.1.9 Exercise. *(*) Let* $\mathrm{sum} = (\mathrm{sum}_i)$ *be the p-family given by* $\mathrm{sum}_i = X_1 + \cdots + X_i$*. Likewise, let* prod *be given by* $\mathrm{prod}_i = X_1 \dots X_i$*. Prove that* prod *is not a p-projection of* sum *and vice versa.*

1.1.10 Question. *Is* det VP*-complete?*

When we replace polynomial upper bounds by quasipolynomial upper bounds (of the form $O(n^{\log^c(n)}$ for constant c) in the definition of VP and p-projections, then the determinant is complete for this class usually called VQP, see [Bür00] and [Blä01] for more complete families. Here, "QP" stands for "quasi-polynomial".

1.2 $\mathbf{VP}_e$

We call an arithmetic circuit a *formula* if the underlying graph structure is a tree. In this case, every computation gate has fanout one, that is, every intermediate result can only be used once.

1.2.1 Definition. *A p-family (f_n) is contained in the class* VP_e *if there is a family of formulas (F_n) such that F_n has polynomial size in n and computes f_n.*

The "e" in the subscript stands for *expression*, another word for formula. Since every formula is a circuit, we have $\mathrm{VP}_e \subseteq \mathrm{VP}$. It is not known that whether this inclusion is strict, but most researchers believe it is.

1.2.2 Question. *Is* VP_e *a strict subset of* VP*?*

1.2.3 Definition. *1. A p-family (f_n) is in the class* VNC_i *if there is a family of circuits (C_n) that the size of C_n is polynomial in n and the depth of C_n is bounded by $O(\log^i n)$.*

2. $\mathrm{VNC} = \bigcup_{i\in\mathbb{N}} \mathrm{VNC}_i$.

It turns out that $\mathrm{VP}_e = \mathrm{VNC}_1$, that is, every p-family that is computable by formulas of polynomial size has efficient parallel algorithms. (We also know that $\mathrm{VP} = \mathrm{VNC}_2$. This is in contrast to the Boolean setting, where it is widely believed that not all problems in P have efficient parallel algorithms. The reason is that when we compute a polynomial a degree d, we can always modify the circuit such that the degree at every node is also bounded by d at a cost which is polynomial in d. Since in Valiant's model, the degree is polynomially bounded, which is the key to get small depth circuits. On the other hand, if the degree in Valiant's model was not bounded, then VP could be easily separated from VNC by a degree argument. In the Boolean setting, we are only computing functions, not polynomials. Therefore, using high degree intermediate results can be helpful, but we do not know how to prove a separation of the classes.)

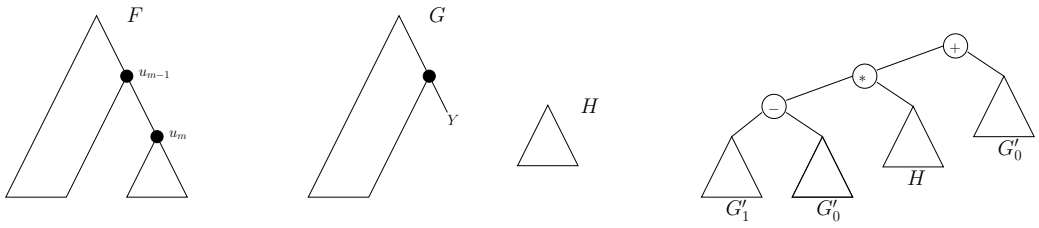

Figure 1.1: The formula F with the edge (u_{m-1}, u_m), the two formulas G and H, and the new formula computing f

1.2.4 Lemma. *Let T be a binary tree with n nodes. Then there is an edge e in T such that removing e separates T into two trees both having between $n/3$ and $2n/3$ nodes.*

Proof. We construct a path $u_1, \ldots, u_m$ starting from the root as follows: We set u_1 to be the root of the path. Let u_i be the current end node of the path and let w and w' be its children. If the subtree with root w is larger than then subtree with root w', then $u_{i+1} := w$, otherwise $u_{i+1} := w'$. We stop when the size of the subtree with root u_i is $< 2/3n$. The edge e is the edge (u_{m-1}, u_m). The subtree with root u_m has size $< 2/3n$ by construction. The subtree with root u_{m-1} has size $\geq 2/3n$. Since u_m is the root of the larger subtree, the subtree with root u_m has size at least $n/3$. The size of the remaining tree is between $n - 2/3n = n/3$ and $n - n/3 = 2n/3$. □

1.2.5 Theorem (Brent [Bre74])**.** *Let F be a formula of size s. Then there is a formula F' of size* $\mathrm{poly}(s)$ *and depth $O(\log s)$ computing the same polynomial as F.*

Proof. By Lemma 1.2.4, there is an edge in F such that when removing e, we get two parts, each of size between $s/3$ and $2s/3$. The part not containing the output gate of F is again a formula, which we call H. The part containing the output gate is not a formula, since one of the gates has fanin one after removal of e. We add a new child to this gate, which is labeled with a new input variable Y. Call the resulting formula G. G computes a linear form $aY + b$, since Y appears only once in G. (a and b are polynomials in the original input variables.) If we substitute the polynomial h computed by H for Y, then we get the polynomial f computed by F. If we substitute 1 for Y, then we get $a + b$, and if we substitute 0 for Y, then we get b. Therefore, there are formulas of size $\leq 2s/3$ computing $a + b$, b and h. With these formulas, we can proceed recursively. We get formulas G'_1, G'_0, and H' computing $a + b$, b, and h, respectively. We can combine them to a formula computing $ah + b = f$ as depicted in Figure 1.1: For the size $\sigma(s)$ of this new formula, we get the recursion

$$\sigma(s) = 4 \cdot \sigma(2s/3) + 3$$

and for the depth $d(s)$, we get the recursion

$$d(s) = d(2s/3) + 3.$$

It is a routine check that $\sigma(s) = \mathrm{poly}(s)$ and $d(s) = O(\log s)$. □

1.2.6 Corollary. $\mathrm{VP}_e = \mathrm{VNC}_1$.

1.3 Constant size iterated matrix multiplication

For some $c \in \mathbb{N}$, we define the family $(\mathrm{imm}_n^{(c)})$ like the family (imm_n), except that every polynomial is an iterated matrix product of $c \times c$-matrices (instead of $n \times n$-matrices), so $\mathrm{imm}_n^{(c)}$ is a polynomial in $c^2 n$ variables.

1.3.1 Theorem (Ben-Or & Cleve [BC92])**.** *Let F be a formula of depth d computing a polynomial f, then f is a projection of* $\mathrm{imm}^{(3)}_{4^d}$.

Proof. We will prove by induction on d, that we can find 4^d 3×3-matrices the entries of which are either indeterminates or constants such that the product of these matrices is

$$\begin{pmatrix} 1 & f & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This is obviously true for depth zero formulas, since these formulas compute constants or single variables.

If the depth d is larger than zero, we either have $f = g + h$ or $f = gh$ and g and h are both computed by formulas of depths $\leq d-1$. By the induction hypothesis, there are two sets of 4^{d-1} 3×3-matrices each such that their products are

$$\begin{pmatrix} 1 & g & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

respectively. In the case of an addition gate we have

$$\begin{pmatrix} 1 & g & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & g+h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore we can write f as a projection of a 3×3-iterated matrix multiplication of length $2 \cdot 4^{d-1} \leq 4^d$.

In the case of a multiplication gate, we have

$$\begin{pmatrix} 1 & g & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & h \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & g & gh \\ 0 & 1 & h \\ 0 & 0 & 1 \end{pmatrix}.$$

Note that h is standing in the "wrong" position. But we can easily fix this by applying permutation matrices from the left and the right. This just corresponds to exchanging the rows or columns of the first and last matrix of the corresponding matrix product, respectively. We proceed with

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -h \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & g & gh \\ 0 & 1 & h \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -g & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & gh \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note that we now have a $-g$ and $-h$ instead of a g and h. But this is easily by multiplying the second row and column by -1. This can again be achieved by doing this with the first and last matrix of the 4^{d-1} matrices. Altogether, we get that f is a projection of a product of $4 \cdot 4^{d-1} = 4^d$ matrices. □

1.3.2 Corollary. $\mathrm{imm}^{(3)}$ *is* VP_e*-complete.*

Proof. Let $f = (f_n) \in \mathrm{VP}_e$. Let F_n be a formula of polynomial size computing f_n. By Theorem 1.2.5, there is an equivalent formula of polynomial size and depth $O(\log n)$. By Theorem 1.3.1, f_n is a projection of $\mathrm{imm}^{(3)}_{\mathrm{poly}(n)}$. This proves the hardness.

To construct a formula of polynomial size for $\text{imm}_n^{(3)}$, we divide the product into two products of size $n/2$ each. We can assume that n is a power of 2, since $\text{imm}_{n'}^{(3)} \leq \text{imm}_n^{(3)}$ if $n' \leq n$. The entries of the result of the two products can we computed by 18 instances of $\text{imm}_{n/2}^{(3)}$, one for each entry of the two resulting matrices. From these two results, we can compute $\text{imm}_n^{(3)}$ by a constant size formula. Since each entry of the two resulting matrices is used three times, we need three distinct copies of the formulas for each entry. Therefore, we get the following recursion for the size $s(n)$ of the formula:

$$s(n) = 54s(n/2) + O(1).$$

Therefore, $s(n) = \text{poly}(n)$. □

Obviously, $\text{imm}^{(c)}$ is VP_e-complete for any $c \geq 3$. On the other hand, $\text{imm}^{(1)}$ is not, since it only computes a single monomial. Allender and Wang [AW16] prove that $\text{imm}^{(2)}$ is also not VP_e-complete by exhibiting a polynomial that is not the projection of $\text{imm}_n^{(2)}$ for any n!

1.4 Further exercises

In the following exercises, we develop a combinatorial algorithm for the determinant by Mahajan and Vinay [MV97]. A cycle cover of a directed graph is a collection of node disjoint cycles such that each node is part of exactly one cycle. Let A be an $n \times n$-matrix with entries from some ring. A defines an edge-weighted directed graph G on the node set $\{1, \ldots, n\}$ by given the edge (i, j) the weight $a_{i,j}$. Permutations π in $\mathfrak{S}_n$ stand in one-to-one correspondence with cycle covers C in G. The weight $w(C)$ of a cycle cover C is the product of the edge weights of its edges. The sign of a cycle cover C with k cycles is $\text{sgn}(C) = (-1)^{n+k}$. We can write

$$\det A = \sum_C \text{sgn}(C) w(C)$$

where the sum is over all cycle covers C.

A *clow* (closed ordered walk) is a sequence $C = (c_1, \ldots, c_i)$ of numbers in $\{1, \ldots, n\}$, that is, nodes in G, such that the first number c_1 is smaller than all other numbers in the sequence. The node c_1 is called the *head* of C. A clow is a closed walk that visits c_1 only once but may visits others nodes more than once. The weight of a clow is the product of the weight of its edges. A *clow sequence* S is a sequence of clows $C_1, \ldots, C_k$ such that the heads $h_1, \ldots, h_k$ of $C_1, \ldots, C_k$, are strictly increasing, that is, $h_1 < \cdots < h_k$.

1.4.1 Exercise. *Let $S = (C_1, \ldots, C_k)$ be a clow sequence and let i be the smallest index such that $C_{i+1}, \ldots, C_k$ are simple cycles. Assume that $i \geq 1$. Let v be the first node (starting from the head) in C_i that is either visited twice in C_i or is also in one of $C_{i+1}, \ldots, C_k$, say C_j.*

1. *(*) Prove that v can never satisfy both conditions simultaneously.*
2. *(**) In the first case, we define a new sequence by removing the simple cycle starting in v from C_i and add it as a new clow to the sequence (at the appropriate position with the appropriate node as head). Prove that the new sequence is indeed a clow.*
3. *(**) In the second case, we join C_j and C_i. Prove that we again get a clow sequence.*
4. *(***) Define a map I on the set of all clow sequences of length n as follows: If $i = 0$, then $I(S) = S$, that is, S is a fix point. Otherwise, $I(S)$ is either the clow sequence from part 2.) or 3.). Prove that I is an involution.*
5. *(*) Prove that when S is not a fixed point of I, then $w(S) = w(I(S))$ and $\text{sgn}(S) = -\text{sgn}(I(S))$.*

6. *(**) Prove that*

$$\det A = \sum_S \operatorname{sgn}(S) w(S)$$

where the sum is now over all clow sequences S of length n.

Next we will use dynamic programming to efficiently compute the determinant using clow sequences. A partial clow sequence consists of a number of clows plus one clow which is not yet completed. $[\ell, c, c_0, s]$ will denote the weight of all partial clow sequences of length ℓ, where the incomplete clow has head c_0 and ends in the current node c. $s = \pm 1$ denotes the parity of the finished clows so far. One can extend a partical clow sequence by either adding a new node to the last partial clow or by closing it and starting a new clow. We create a dynamic programming graph: The nodes in this graph are all of the form (ℓ, c, c_0, s) with $1 \leq \ell \leq n$, $1 \leq c_0 \leq c \leq n$, and $s = \pm 1$. From every node (ℓ, c, c_0, s) there are edges going to $(\ell + 1, c', c_0, s)$ for all $c' > c_0$ and edge weight $a_{c,c'}$. And there are edges going to $(\ell + 1, c_0', c_0', -s)$ for all $c_0' > c_0$ with edge weight a_{c,c_0}. The weight of a path in this dynamic programming graph is the product of the edge weights.

1.4.2 Exercise. 1. *(**) Prove that the $[\ell, c, c_0, s]$ is the weight of all paths starting in a node of the form $(0, h, h, 1)$, $1 \leq h \leq n$, and ending in (ℓ, c, c_0, s).*

2. *(**) Prove that there is an arithmetic circuit of size $O(n^4)$ for computing the determinant.*

3. *(***) Prove that we can even achieve that this circuit has depth $O(\log^2 n)$.*

Chapter 2

Universality of the determinant

2.1 Homogeneous circuits

Recall that a polynomial is homogeneous if all its monomials have the same total degree. A circuit is called *homogeneous* if at every gate, it computes a homogeneous circuit. Of course, nonhomogeneous polynomials cannot be computed by homogeneous circuits. However, we have the following result.

2.1.1 Lemma. *If f is a polynomial of degree d that is computed by a circuit of size s, then there is a homogeneous circuit of size $O(d^2 s)$ computing the homogeneous parts of f. Furthermore, at every gate, we only compute a polynomial of degree at most d.*

Proof. We replace every gate g by $d+1$ gates. If g computes a polynomial f, then the new gates will compute the homogeneous components of f. We do this in a bottom up fashion If g is an input gate, then there is nothing to do. We just have to add d dummy gates computing the zero polynomial. Let g be a gate with children h_1 and h_2 in the original circuit. Assume that h_1 and h_2 have been replaced by gates $h_{1,0}, \ldots, h_{1,d}$ and $h_{2,0}, \ldots, h_{2,d}$ computing polynomials $p_{1,0}, \ldots, p_{1,d}$ and $p_{2,0}, \ldots, p_{2,d}$, respectively. If g is an addition gate, then we will introduce new gates $g_0, \ldots, g_d$ and g_i computes $p_{1,i} + p_{2,i}$. If g is a multiplication gate, then g_i computes $\sum_{j=0}^{i} p_{1,j} p_{2,i-j}$. □

2.1.2 Corollary. *If f is a polynomial of degree d that is computed by an arithmetic circuit of size s, then there is a circuit C of size* $\mathrm{poly}(s,d)$ *computing f such that every node in C computes a polynomial of degree at most d. Furthermore, for every multiplication gate, at least one of the inputs is not a constant.*

Proof. We homogenize the given circuit as above. This immediately gives the upper bound on the degree. When two constants are multiplied, then either two degree zero components are multiplied or one of the higer degree homogeneous parts became zero. In the first case, we can replace the multiplication gate by an input gate labeled with the product of the two constants. (Remember that we can use every constant from $\mathbb{F}$.) In the second case, we simply can remove the gate that outputs 0. (Note that we do not have to construct the circuit, we just need to prove it existence.) □

2.2 Multiplicatively disjoint circuits

2.2.1 Definition. *An arithmetic circuit is* multiplicatively disjoint *if for all multiplication gates, the subcircuits induced by its two children are disjoint.*

Multiplicatively disjoint circuits are between circuits and formulas. In a formula, also the subcircuits of addition gates are disjoint.

2.2.2 Definition. *Let C be an arithmetic circuit. The formal degree of a gate g is defined inductively: A leaf has formal degree 1. If g is a multiplication gate, then its formal degree is the sum of the formal degrees of its two children. If g is an addition gate, then the formal degree of g is the maximum of the formal degrees of its children. The formal degree of C is the formal degree of its output gate.*

The formal degree of a circuit disregards that the degree at gate might drop when there are cancellations. Multiplications with constants might also increase the formal degree.

2.2.3 Lemma. *If a circuit has size s and formal degree d, then there is a multiplicatively disjoint circuit C' of size $\leq sd$ computing the same polynomial.*

Proof. Each gate g of formal degree $e \leq d$ will be replaced by $d+1-e$ copies $g_1, \ldots, g_e$. Let g_i be one of these copies. We call i the index of the copy. We will make sure that all gates of the subcircuit with output g_i are copies with an index lying between i and $i+e-1$. In this way we ensure that we will get multiplicatively disjoint circuits.

Inductively, we construct a circuit C_e with the following property: For each gate g for formal degree $f \leq e$ in C, there are copies of the gates $g_1, \ldots, g_{d+1-f}$ in C_e computing the same function as g and all the gates of the subcircuit with root g_i have indices lying between i and $i+f-1$.

The nodes of formal degree one are all input nodes and sums of degree one nodes. C_1 consists of d copies of the input nodes. Since C is acyclic, we can order the addition gates in such a way, that whenever we deal with a gate g, all its predecessors have been processed. For each addition gate g of formal degree one, we add copies $g_1, \ldots, g_d$. Let g' and g'' be the children of g in C with formal degrees one. We connect g_i with the copy g'_i and g''_i. The restriction on the ranges is fulfilled by construction.

Assume that we constructed C_{e-1} (induction hypothesis). To obtain C_e, we now add copies of all gates g of formal degree e in C. Let g' and g'' be the children of sucha gate g of formal degrees e' and e'', respectively.

We start with the multiplication gates. In this case $e = e' + e''$ with $e', e'' < e$. This means that the copies $g'_1, \ldots, g'_{d+1-e'}$ and $g''_1, \ldots, g''_{d+1-e''}$ were constructed in a previous step. We add the copies $g_1, \ldots, g_{d+1-e}$ and connect g_i with g'_i and $g''_{i+e'}$. These copies exist, since $i \leq d+1-e \leq d+1-e'$ and $i+e' \leq d+1-e+e' = d+1-e''$. The indices of the copies of the subcircuit with root g'_i lie between i and $i+e'-1$, the indices of the copies in the subcircuit with root $g''_{i+e'}$ lie between $i+e'$ and $i+e'+e''-1 = i+e-1$. Furthermore, $i \leq i+e' \leq i+e-1$. Therefore the condition on the indices of the subcircuits is fulfilled.

Next come the addition gates of formal degree e. Note that an addition gate of formal degree e might have a predecessor of formal degree e. As in the base case, we can order the addition gates in such a way, that whenever we deal with a gate g, all its predecessors have been processed. For each addition gate g of formal degree e, we add copies $g_1, \ldots, g_{d+1-e}$. Let g' and g'' be the children of g in C with formal degrees $e' \leq e$ and $e'' \leq e$, respectively. We connect g_i with the copy g'_i and g''_i. The indices of the copies in these subcircuits lie in the range from i to $i+e'-1 \leq i+e-1$ and $i+e''-1 \leq i+e-1$, respectively.

The circuit C_d is the circuit we are looking for. It contains a copy of the output gate of C. The circuit is multiplicatively disjoint by the way we chose the indices when connecting the copies of the children to the multiplication gate. □

2.2.4 Lemma. *Let f be a polynomial of degree d computed by a circuit C of size s. Then there is a circuit of size polynomial in d and s computing f such that its formal degree is bounded by $sd+1$.*

Proof. Let C be the given circuit and C' be the circuit constructed in Corollary 2.1.2. Let the depth of a gate be the length of a longest path from any leaf to this gate. We will now prove by induction on the depth that the formal degree of any gate g of depth δ computing a homogeneous component of degree i is bounded by $\delta \cdot i + 1$. Recall that the circuit C' is a simulation of the circuit C. Every node is replaced by $d+1$ nodes, one for each homogeneneous component. Then every operation in C is simulated by a bunch of operations in C'. We will measure the depth in the above inductive statement by the depth in C and we will only prove it for nodes in C' that correspond to nodes in C.

For the base case note that every leaf has formal degree 1. Now let g be a gate of depth δ computing a homogeneous component of degree i. If $i = 0$, then note that g has formal degree 1 by construction. So we assume that $i \geq 1$. We first treat the case when g is corresponds to an addition gate in C. In C', g is an addition gate, its two inputs are gates g' and g'' both computing homogeneous polynomials of degree i. The formal degree of these two gates are bounded by $\delta' \cdot i + 1$ and $\delta'' \cdot i + 1$ where δ' and δ'' are the depth of g' and g'', respectively. The formal degree of g is $\max\{\delta' \cdot i + 1, \delta'' \cdot i + 1\} \leq \delta \cdot i + 1$.

If g is a multiplication gate, then

$$g = \sum_{j=0}^{i} g'_j g''_{i-j}$$

where g'_j and g''_{i-j} are the homogeneous components of the predecessors of g. By the induction hypothesis, the formal degrees of g'_j and g''_{i-j} are bounded by $\delta' j + 1$ and $\delta''(i-j)+1$, respectively. The formal degree of $g'_j g''_{i-j}$ is bounded by $\delta' j + 1 + \delta''(i-j) + 1 \leq \delta i + 1$, when $0 < i < j$. Note for the upper bound that $\delta > \delta', \delta''$ and $i \geq 1$. The formal degree of $g'_0 g''_i$ is bounded by $1 + \delta'' i + 1 \leq \delta i + 1$. The same argument works for $g'_j g''_0$. This concludes the inductive step.

From the claim the bound on the formal degree of the new circuit follows immediately. □

2.2.5 Theorem (Malod & Portier [MP08])**.** *A p-family (g_n) is in* VP *if and only if there is a family of polynomial size multiplicative disjoint circuits (C_n) computing (g_n).*

Proof. If $(g_n) \in$ VP, then by Lemma 2.2.4, there is a sequence of circuits (C_n) of size poly(n) computing (g_n) such that the formal degree of C_n is polynomially bounded. Now we can apply Lemma 2.2.3.

For the other direction, note that it can be easily proven by induction that the degree of a multiplicatively disjoint circuit of size s is bounded by s. □

2.2.6 Exercise. *(*) Prove the last statement.*

2.3 Weakly skew circuits and algebraic branching programs

Let $M = (m_{i,j})$ be an $n \times n$ matrix. We can interpret M as the weighted adjacency matrix of some graph over the node set $\{1, \ldots, n\}$. For every (i, j), there is an edge (i, j) of weight $m_{i,j}$. A *cycle cover* in a directed graph is a collection of node-disjoint directed cycles such that every node is contained in exactly one cycle. Permutations in S_n stand in a one-to-one correspondence with cycle covers. Every permutation σ yields a cycle cover consisting of the edges $(i, \sigma(i))$. On the other hand, the edges of a cycle cover encode a permutation of the nodes with the intepretation that an edge (i, j) means that i is mapped to j. Note that this is nothing but the cycle decomposition of a permutation. The sign of the permutation is -1 if the number of cycles is even, and 1 if it is odd. The weight $w(C)$ of a cycle cover C is the product of the weights of the edges in it. Therefore,

$$\det M = \sum_{\text{cycle covers } C} (-1)^{n+\text{number of cycles in } C} w(C)$$

Conceptually, it is often easier to think of an edge of weight zero as not being present in the graph. Since the weight of a cycle cover is the product of its edge weights, this does not make any difference in the above equation for $\det M$.

2.3.1 Definition. *A circuit is called* weakly skew *if every multiplication gate g has at least one child g' such that after removing the edge (g', g), the graph consists of two weakly connected components.*

In a formula, this is true for every child of a gate, i.e., no intermediate result is reusable. In a weakly skew circuit, one child of every gate can be reused, but not both. Weakly skew is however stronger than multiplicatively disjoint, since in the later case, while the subcircuits need to be disjoint, they can be connected to the rest of the circuit.

2.3.2 Exercise. *(*) Construct a multiplicatively disjoint ciruit that is not weakly skew.*

2.3.3 Definition. *Let $\mathbb{F}$ be a field and $X_1, \ldots, X_n$ be indeterminates.*

1. *An* algebraic branching program *A is an acyclic graph with two distinguished nodes s and t and an edge labeling with labels from $\mathbb{F} \cup \{X_1, \ldots, X_n\}$.*
2. *The weight $w(P)$ of a path P from s to t is the product of the labels of the edges in the path.*
3. *The polynomial computed by A is*
$$\sum_{s\text{-}t \text{ path } P} w(P).$$
4. *The size of an arithmetic branching program is the number of edges in it.*
5. *A is called* layered *if for every node v in A, all s-v paths have the same length.*

If A is layered, then we can think of the nodes of A being grouped into layers: two nodes are in the same layer if the length of any path from s to them is the same. In a layered branching program, edges only go from one layer to the next.

2.3.4 Lemma. *Let A be a branching program of size s. Then there is a layered branching program of size $O(s^2)$ computing the same function.*

Proof. For a node v in the branching program, let $d(v)$ be the length of a longest path from s to v. Scan through the nodes by increasing value of $d(v)$ (breaking ties arbitrarily) until you find the first node v_0 such that there is a path from s to v_0 that has length $\ell < d(v_0)$. Take the last edge e of such a path and subdivide it $d(v_0) - \ell$ many times. If e has label w, then one of the new edges gets label w and all other ones will get label 1. We do this with all shorter paths from s to v_0 and then go on with the next node at which the layering property is violated. □

We formalize the notion of being *reusable*. Intuitively, a gate in a weakly skew circuit is reusable it is not in the subcircuit of a multiplication gate that is not connected to the rest of the circuit.

2.3.5 Definition. *Let C be a weakly skew arithmetic circuit. The set of* reuseable *gates in C is inductively defined as follows: Every gate of outdegree zero is reusable. (We consider circuits with multiple output gates to simplify some proofs in the following.) We remove every gate g of outdegree zero from C and for each such multiplication gate, we also remove the subcircuit of that child g' that is only connected to the rest of the circuit via the edge (g', g). Let C' be the resulting circuit. Every gate that is reusable in C' is reusable in C, too.*

2.3.6 Theorem. *Let $f \in k[X_1, \ldots, X_n]$ with $\deg f = \text{poly}(n)$. The following statements are equivalent:*

1. *f is computed by a weakly skew circuit of size* $\mathrm{poly}(n)$.
2. *f is computed by an algebraic branching program of size* $\mathrm{poly}(n)$.
3. *f is a projection of* $\mathrm{imm}_{p(n)}$ *for some polynomially bounded function p.*
4. *f is a projection of* $\det_{p(n)}$ *for some polynomially bounded function p.*

Proof. (1) $\Rightarrow$ (2): Assume that f is computed by a weakly skew circuit C of size m. We now prove by induction on m that there is a arithmetic branching program computing A of size $\leq 2m$ such that for every reusable gate g in C there is a node v_g such that the sum of the weights of all paths from s to v_g is the same polynomial as computed at g.

Let g be some output node. If g is also an input node, then A consists of a single edge. (This is the induction basis.)

For the induction set, assume that g is not an input gate. If g is an addition gate, then we remove g from C, let C' be the resulting circuit. By the induction hypothesis, there is an algebraic branching program such that for every gate g' that is reusable in C', there is a node $v_{g'}$ in C' such that the sum of the weights of all path from s to $v_{g'}$ equals the polynomial computed at g'. Let h and h' be the children of g. We add a new node v_g and connect the nodes v_h and $v_{h'}$ to it. Both edges get weight one. If $h = h'$, then we add only one edge with weight two. By construction, the sum of the weights of all paths from s to v_g is the sum of the polynomials computed at h and h'. The resulting arithmetic branching program has two more edges than A'.

If g is a multiplication gate, then after removal of g, we get two separate circuits C_1 and C_2. Let g_1 and g_2 be the children of g. Only the gates of one of them, say C_2, can be reusable in C. Let m_1 and m_2 be the sizes of C_1 and C_2. From the induction hypotheses, we get corresponding algebraic branching programs A_1 and A_2 with sources s_1 and s_2. In A_1, there are vertices s_1 and v_{g_1} such that the sum of the weights of all path from s_1 to v_{g_1} equals the polynomial computed at g_1. We identify the node s_1 of A_1 with the node v_{g_2} in A_2. Then the sum of the weights of all path from s_2 to v_{g_1} is the product computed at g. For all gates h in C_2, the sum of the weights of all paths from s_2 to v_h paths equals the polynomial computed at h. The size of the new branching program is $2m_1 + 2m_2 \leq 2m$.[1]

(2) $\Rightarrow$ (3): Let A be an algebraic branching program computing f. By Lemma 2.3.4 we can assume that A is layered. Let ℓ be the maximum size of a layer and let m be the number of layers. We will inductively construct $\ell \times \ell$-matrices $M_1, \ldots, M_m$ with entries from $k \cup \{X_1, \ldots, X_n\}$ such that the first row of $M_1 \cdots M_i$ are the polynomials computed at the nodes in the ith layer, that is, the sum of the weights of all path from s to each node in this layer. M_1 has a one in position $(1, 1)$ and zeroes elsewhere. This one corresponds the the source node s. Assume we constructed $M_1, \ldots, M_i$. Let $(a_1, \ldots, a_\ell)$ be the first row of $M_1 \cdots M_i$. A node v in the $(i+1)$th layer receives edges from the nodes of the ith layer. Let $(b_1, \ldots, b_\ell)$ be the labels of these edges (if an edge is not present, the corresponding $b_j = 0$.) The polynomial computed at v is given by

$$(a_1, \ldots, a_\ell) \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_\ell \end{pmatrix}.$$

The matrix M_{i+1} simply consists of the corresponding columns $(b_1, \ldots, b_\ell)^T$. Since we can embed a product of m $\ell \times \ell$-matrices into a product of d $d \times d$-matrices with $d = \max\{m, \ell\}$, we get that f is a projection of $\mathrm{imm}_{\mathrm{poly}(n)}$.

[1] This construction does not work if the circuit is only multiplicatively disjoint, since in this case, while the subcircuits of every multiplication gate are disjoint, they might both be connected to the rest of the circuit. However, the nodes of A_1 cannot be used any more, once s_1 is identified with v_{g_2}.

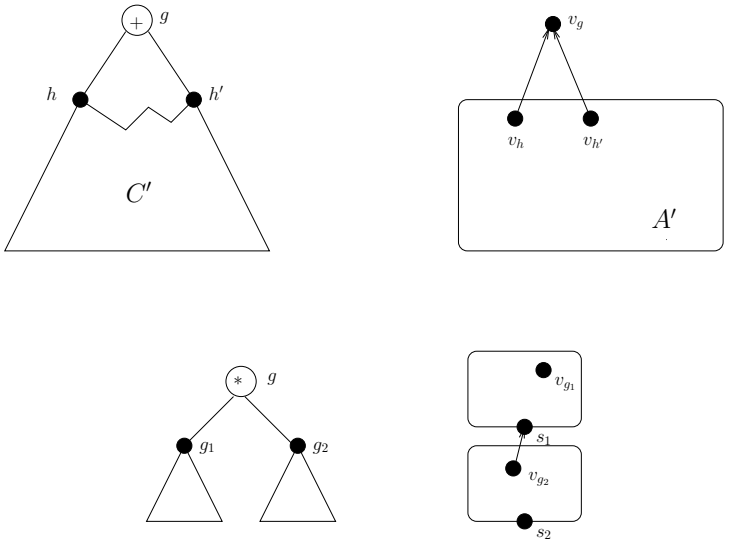

Figure 2.1: Transforming weakly skew circuits into algebraic branching programs. (Top: addition gate, bottom: multiplication gate)

(3) $\Rightarrow$ (4): Note that an iterated matrix product can be easily computed by a layered algebraic branching program, you just have to "reverse" the construction of the previous step. Therefore it suffices to prove that every polynomial that is computed by a layered algebraic branching program A is a projection of a determinant of polynomial size. We modify A as follows: add an edge of weight one from t to s and add a self loop of weight one to every node except s and t. Let M be the weighted adjacency matrix of this modified program A'. $\det M$ is the sum of the weights of all cycle covers in A'. All cycle covers in A' consist of one big cycle through s and t and the remaining nodes are covered by self-loops. Since the program is layered, all cycle covers have the same number of cycles and therefore the same sign. The weight of a cycle cover equals the weight of the corresponding path from s to t, potentially with an opposite sign. Therefore, f is a projection of a polynomially large matrix.

(4) $\Rightarrow$ (1): One way to evaluate the determinant by a weakly skew circuit was done in the exercise at the end of the previous chapter. □

2.3.7 Definition. *A p-family (f_n) is in VP_{ws} if it is computed by weakly skew circuits of polynomial size*

Theorem 2.3.6 gives us further, equivalent definitions of VP_{ws}. In particular, a p-family (f_n) is in VP_{ws} if it is a p-projection of the determinant family. Note that imm can be computed by very restricted weakly skew circuits, namely for every multiplication gate, one of the inputs consists of a variable or constant. This is achieved by sequentially multyplying the matrices using the trivial methods. We call such circuits *skew*. Since by Theorem 2.3.6, every polynomial that is computed by a weakly skew circuit of size s is a p-projection of imm, we get the following corollary.

2.3.8 Corollary. *If a polynomial is computed by a weakly skew circuit of size s, then it is computed by a skew circuit of size* $\mathrm{poly}(s)$.

2.3.9 Exercise. *(**) Write a 2×2-determinant as a projection of an iterated matrix multiplication (size of your choice). Write the $(1,1)$-entry of a the product of two 2×2-matrices as a projection of a determinant (size of your choice).*

2.4 Further exercises

The aim of this section is to prove that $\mathrm{VP} = \mathrm{VNC}_2$, which is originally due to Valiant et al. We follow ideas by Tavenas.

2.4.1 Definition. *Let C be a circuit with unbounded addition gates, multiplication gates of fanin bounded by 5 and scalar multiplication gates. We call a circuit $\times$-balanced, if for all multiplication gates g, the degree of the polynomial computed at each child g' of g is at most half the degree of the polynomial computed at g.*

2.4.2 Exercise. *(****) Let f be a homogeneous polynomial of degree d computed by a circuit of size s. Then f is computed by a homogeneous $\times$-balanced circuit of degree d and size* $\mathrm{poly}(s, d)$.

We break the above exercise into smaller parts. We may assume that the given circuit is homogeneous. Furthermore, we may assume that all internal nodes have positive degree by replacing an internal gate of degree zero by the corresponding constant. In particular, a constant cannot be the input to am addition gate. Furthermore, we replace every multiplication gate which has a constant as an input by a special scalar multiplication gate $\odot$. We order the multiplication gates in the circuit in such a way that the right child has higher degree than the left child (breaking ties arbitrarily). Let C be the resulting circuit.

2.4.3 Exercise. *(*) Let o be the output gate of C. We have*

$$f = \sum_{\text{parse tree } T} v(T) = \sum_{\text{leaf } \ell} (o, \ell).$$

Thus, if we can compute all values (o, ℓ), then we can compute f. Note that the second sum is polynomial while the first is exponential.

Let α and β be gates of C. We define (α, β) as follows:

- if β is a leaf, then (α, β) is the sum of the values of all parse trees rooted at α such that β appears on the rightmost path in the tree.
- if β is not a leaf, then (α, β) is the sum of the values of all parse trees rooted at α such that β appears on the rightmost path, but the subtree on the rightmost path with root β is deleted and replaced by 1.

Note that $(\alpha, \beta) = 0$, if β is not contained in the subcircuit with root α.

We now show how to compute the values (α, β) by a dynamic programming approach. If α is an addition gate, then this computation done by a single addition gate. If α is a multiplication gate, then this computation is done by an addition gate with a layer of multiplication gates below. The inputs of these multiplication gates will be polynomials of degree at most half the degree of (α, β).

From this it follows that we can turn this dymnamic programming scheme into a $\times$-balanced circuit:

- If β does not appear on any parse tree rooted in α, then $(\alpha, \beta) = 0$.
- If $\alpha = \beta$, then $(\alpha, \beta) = [\alpha]$ if α is a leaf and $(\alpha, \beta) = 1$ otherwise.

In the remaining cases, $\alpha \neq \beta$, α is not a leaf, and β appears on the rightmost path in some parse tree rooted at α.

2.4.4 Exercise. *(*) What is the expression when α is an addition gate?*

2.4.5 Exercise. *(*) What is the expression when α is a scalar multiplication gate?*

- The interesting case is when α is a multiplication gate. We first assume that β is a leaf. Then $\deg(\alpha) > \deg(\beta)$ and $\deg(\beta) \leq 1$. Fix a parse tree rooted at α and with the rightmost path ending in β. On this path, there is exactly one gate γ such that for the right child γ_r of γ,
$$\deg(\gamma) \geq \frac{1}{2}\deg(\alpha) \geq \deg(\gamma_r). \tag{2.4.6}$$
Then
$$(\alpha, \beta) = \sum_{\text{leaf s, } \gamma \text{ fulfilling (2.4.6)}} (\alpha, \gamma)(\gamma_\ell, s)(\gamma_r, \beta), \tag{2.4.7}$$
where γ_ℓ is the left child of γ.

2.4.8 Exercise. *(**) Prove that* $\deg(\alpha, \gamma)$, $\deg(\gamma_r, \beta)$, *and* $\deg(\gamma_\ell, s)$ *are bounded by* $\deg \alpha/2$

More problematic is the case when β is not a leaf. (2.4.7) still holds, but we have $\deg(\alpha, \beta) = \deg \alpha - \deg \beta$. So we choose γ in such a way that
$$\deg(\gamma) \geq \frac{1}{2}(\deg(\alpha) + \deg(\beta)) \geq \deg(\gamma_r). \tag{2.4.9}$$
The problem is that we can only bound $\deg(\gamma_\ell, s)$ by $\deg(\alpha)/2$ or $\deg(\alpha, \beta)$ but not by $\deg(\alpha, \beta)/2$. Therefore, we split the term (γ_ℓ, s) once more. Since s is a leaf, we are in the first subcase. The degree of γ_ℓ is at least one, since γ is not a $\odot$-gate. Therefore, the degree of (α, β) is at least two. If $\deg(\gamma_\ell) = 1$, then we are done. Otherwise, we choose a multiplication gate δ such that
$$\deg \delta \geq \frac{1}{2}\deg \gamma_\ell \geq \deg \delta_r \tag{2.4.10}$$
where δ_r is the right child of δ. As before, we can write:
$$(\gamma_\ell, s) = \sum_{\text{leaf } t;\ \delta \text{ fulfilling (2.4.10)}} (\gamma_\ell, \delta)(\delta_\ell, t)(\delta_r, s), \tag{2.4.11}$$
Altogether, we get
$$(\alpha, \beta) = \sum_{s,\gamma,t,\delta} (\alpha, \gamma)(\gamma_\ell, \delta)(\delta_\ell, t)(\delta_r, s)(\gamma_r, \beta)$$
where the sum is over all leaves s and γ fulfilling (2.4.6) and over all leaves t and δ fulfilling (2.4.10). The sum has polynomial size.

2.4.12 Exercise. *(**) Again, verify the degree constraints.*

2.4.13 Exercise. *(**) Let f be a homogenous polynomial of degree d that is computed by a circuit of size s. Then there is a semi-unbounded circuit of size* $\mathrm{poly}(s, d)$ *and depth* $O(\log d)$.

Chapter 3

The permanent

3.1 VNP and formulas

A language L is in NP if there is a deterministic polynomial time relation R such that for all x, $x \in L$ iff there is a polynomially long bit string y such that $R(x, y) = 1$. Think of x being a formula in 3-CNF and y being an assignment. $R(x, y) = 1$ means that y satisfies x. The class #P is a class of functions, to each x we assign the number of y such that $R(x, y) = 1$, that is, we compute

$$\sum_y [R(x, y) = 1].$$

Here, the bracket is Iverson bracket, which is one if the Boolean expression is true. So in our example, we want to count the number of satisfying assignments.

3.1.1 Definition. *1. A p-family (f_n) is in* VNP*, if there are polynomials p and q and a sequence $(g_n) \in$ VP of polynomials $g_n \in k[X_1, \ldots, X_{p(n)}, Y_1, \ldots, Y_{q(n)}]$ such that*

$$f_n = \sum_{e \in \{0,1\}^{q(n)}} g_n(X_1, \ldots, X_{p(n)}, e_1, \ldots, e_{q(n)}).$$

2. A family of polynomials f_n is in VNP_e if in the definition of VNP*, the family (g_n) is in VP_e.*

You can think of the X-variables representing the input and the Y-variables the witness. With this interpretation, VNP is more like #P. In particular, we will see that the permanent polynomial

$$\mathrm{per}_n = \sum_{\sigma \in S_n} X_{1,\sigma(1)} \cdots X_{n,\sigma(n)}$$

is complete for VNP.

3.1.2 Exercise. *(**) Let $G = (G_n)$ be a sequence of graphs. Assume that G_n has nodes $1, \ldots, n$. Let ind_G be the family of polynomials that is defined by*

$$\mathrm{ind}_{G,n} = \sum_I \prod_{i \in I} X_i$$

where the sum is taken over all independent sets I of G_n. Prove that ind_G is in VNP*.*

3.1.3 Definition. *Let C be an arithmetic circuit.*

1. *A* parse tree *of C is defined recursively as follows: Every circuit consisting of one node is a parse tree. If the size of C is larger than one, let g be the output gate and g_1 and g_2 be its children. Let C_1 and C_2 be the subcircuits with output gates g_1 and g_2. If g is an addition gate, then we get the set of all parse trees by either taking a parse tree of C_1 or a path tree of C_2 and connecting it to g. If g is a multiplication gate, then we get the set of all parse trees by taking a parse of C_1 and a parse tree of C_2 and connecting both to g.*

2. *The set of all parse trees of C is denoted by* $\mathrm{pt}(C)$.

3. *The weight $w(T)$ of a parse tree T is the product of the labels of its leaves.*

For every multiplication gate, we have to include both children in the parse tree, for every addition gate we have to choose one of them. Note that a gate may occur several times in a parse tree, since it is reused in the circuit several times. For each occurrence in the parse tree, we introduce a new copy. (Otherwise, it would not be a tree.)

3.1.4 Exercise. *Let C be a circuit and p be the polynomial computed by C. Prove (for instance by structural induction) that*

$$p = \sum_{T \in \mathrm{pt}(C)} w(T).$$

3.1.5 Lemma. *A circuit C is multiplicatively disjoint if every parse tree of C is a subcircuit of C.*

Proof. Assume that C is not multiplicatively disjoint. Then there is a node v in C such that there a two node disjoint paths to some multiplication gate g. Since g is a multiplication gate, these two paths can be extended to a parse tree.

Conversely, if there is a parse tree T that is not a subcircuit of C, then there are gates g and h in T such that there a two nodes disjoint path from g to h. Since T is a parse tree, h is a multiplication gate. Thus, C is not multiplicatively disjoint. □

3.1.6 Lemma. *Let C be a multiplicatively disjoint circuit with edge set E. For each edge $e \in E$, let X_e be an indeterminate. There is a formula F in the X_e's of size polynomial in the size of C such that for every $a \in \{0,1\}^{|E|}$, $F(a)$ is the weight of the parse tree, if the edges "selected" by the vector a form a parse tree in C, and zero otherwise.*

Proof. For every node v in C, we introduce an additional variable Y_v. Note that for $\{0,1\}$ valued variables X and Y, we can simulate Boolean AND by XY and Boolean NOT by $1 - X$. We can write the fact that a given vector encodes a parse tree by the following Boolean expressions:

$$\bigwedge_{(i,j) \in E} X_{(i,j)} \Rightarrow Y_i \wedge Y_j$$

ensures that whenever an edge is selected, its end points are selected, too. Let g be the output gate of C. Then

$$Y_g$$

ensures that the output gate is selected. For a gate g, let $\ell(g)$ and $r(g)$ be its children. The following expression ensures that for every multiplication gate g that is selected, both incoming edges are selected, too.

$$\bigwedge_{\text{multiplication gate } g} Y_g \Rightarrow X_{(\ell(g),g)} \wedge X_{(r(g),g)}.$$

If we replace the Boolean AND on the righthand side by a Boolean XOR, we get an expression that checks for every selected addition gate whether exactly one of the incoming edges is chosen. Finally, we have to check that every selected gate has at least one outgoing edge. This is done by the following expression:

$$\bigwedge_{v \in V} \left(Y_v \Rightarrow \bigvee_{(v,u) \in E} X_{(v,u)} \right).$$

We can eliminate all occurences of the newly introduced variables by replacing Y_v by the expression

$$\bigvee_{(v,u) \in E} X_{(v,u)}$$

and Y_g by 1. The Boolean AND of these expressions is a Boolean formula that is true iff the vector a encodes a parse tree. By the considerations above, it can be replaced by an arithmetic formula.

If a encodes a parsetree, we can get the corresponding weight by the following expression:

$$\prod_{v \in V} (Y_v \cdot w_v + 1 - Y_v).$$

Here w_v is the label of v if it is an input gate and 1 otherwise. Again, we can eliminate the Y_v's as above. The product of the two expressions, one for checking whether a is a parse tree and one for computing its weight, is the formula F. □

3.1.7 Corollary. *Let f be a polynomial computed by an arithmetic circuit of size s. Then there is an arithmetic formula F of size polynomial in s and a polynomial p such that*

$$f(X) = \sum_{a \in \{0,1\}^{p(s)}} F(X, a).$$

3.1.8 Theorem. $\mathrm{VNP} = \mathrm{VNP}_e$.

Proof. Let (f_n) be in VNP and $(g_n) \in \mathrm{VP}$ such that

$$f(X) = \sum_{e \in \{0,1\}^{q(n)}} g_n(X, e).$$

As seen above, there is a formula F_n of polynomial size such that

$$g_n(X, Y) = \sum_{a \in \{0,1\}^{p(n)}} F_n(X, Y, a).$$

Therefore,

$$f(X) = \sum_{e \in \{0,1\}^{p(n)},\, a \in \{0,1\}^{q(n)}} F_n(X, e, a). \quad \square$$

□

While the statement of the theorem sounds astonishing at a first glance, it just uses the fact that we can write the result of a polynomially large circuit by an exponential sum over a polynomially large formula and then combines the two exponential sums into one.

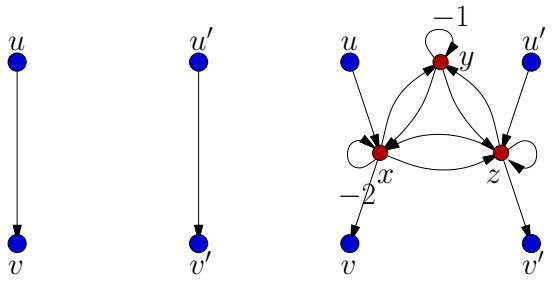

Figure 3.1: The equality gadget. The pair of edges (u, v) and (u', v') of the left-hand side is connected as shown on the right-hand side.

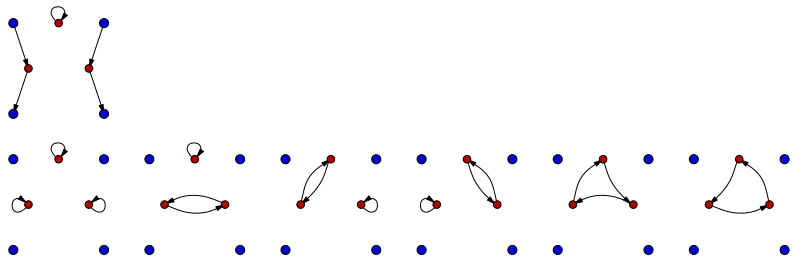

Figure 3.2: First row: The one possible configuration if both edges are taken. Second row: The six possible configurations if none of the edges is taken.

3.2 Hardness of the permanent

Let $G = (V, E)$ be an edge weighted graph. Recall that a cycle cover C of G is a selection of node disjoint directed cycles such that every node is contained in exactly one cycle. The weight $w(C)$ of C is the product of the weight of the edges in C. Cycle covers can be viewed as the graph of a permutation. The cycles in the cycle cover correspond to the cycles in the cycle decomposition of a permutation. If we also write G for the weighted adjacency matrix of G (by abuse of notation), then

$$\operatorname{per}(G) = \sum_{\text{cycle cover } C \text{ of } G} w(C).$$

Let G be a graph and $e = (u, v)$ and $e' = (u', v')$ be two edges in G. As a first step, we want to replace G by a graph $\hat{G}$ such that $\operatorname{per}(\hat{G})$ is the sum over all $w(C)$ such that C is a cycle cover of G that either contains both e and e' or none of them. This is achieved by subdividing the edges an connecting them by an equality gadget as depicted in Figure 3.1.

Let C be a cycle cover of G that takes both edges. Then there is one way to extend this to a cycle cover of $\hat{G}$. The weight of this new cycle cover is $2 \cdot w(C)$, see Figure 3.2. When C does not take any of the two edges, then there are six ways to extend C. These six ways sum up to weight $2 \cdot w(C)$.

If C is a cycle cover of G that takes only one edge of e and e', say e, then there are two ways to extend C to $\hat{G}$, see Figure 3.3. The weight of these covers is the same, but they differ in sign, therefore the contributions of these two covers cancel each other.

Finally, there are inconsistent ways to cover the equality gadget in $\hat{G}$, that is, covers of $\hat{G}$ that do not correspond to any cover in G, see Figure 3.3. Again, we can form pairs of these covers such that the contribution of these covers cancel each other.

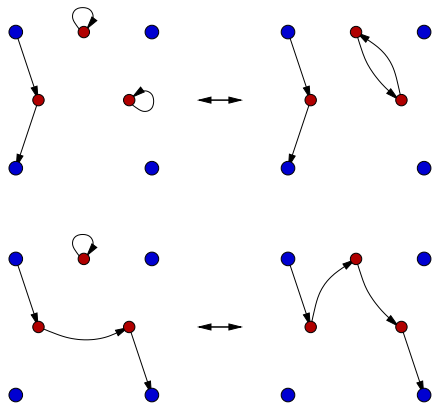

Figure 3.3: First row: The two covers of the equality gadget when only one edge is taken. Second row: Inconsistent covers of the equality gadget. (In both rows, there is a corresponding symmetric case).

3.2.1 Lemma. *Let $\mathbb{F}$ be a field of characteristic distinct from 2. Let G be a graph and e and e' be edges in G. Then there is a graph $\hat{G}$ such that*

$$\frac{1}{2}\mathrm{per}(\hat{G}) = \sum_C w(C),$$

where the sum is taken over all cycle covers C of G that either use both of e and e' or none of them.

Let $(f_n) \in \mathrm{VNP}$ and let $(g_n) \in \mathrm{VP}$ such that

$$f_n(X_1, \ldots, X_{p(n)}) = \sum_{e \in \{0,1\}^{q(n)}} g_n(X_1, \ldots, X_{p(n)}, e_1, \ldots, e_{q(n)}).$$

We may assume that $(g_n) \in \mathrm{VP}_e$. We proved that every polynomial that is computed by a formula of size s is a projection of a determinant of polynomial size. The same proof yields that it is also a projection of a polynomially large permanent, since the cycle covers of the arithmetic branchning program occuring in the proof all had the same sign. It follows that we can write f_n as an exponential sums of permanents. The permanent itself is an exponential sum. So we are done if we can "squeeze" the outer exponential sum into the inner one.

The *rosette graph* of size t consists of a directed cycle of size t. The edges $c_1, \ldots, c_t$ of this cycle are called connector edges. The head and the tail of each connector edge are connected by a path of length two. Every node has a self-loop. All edges have weight one in the rosette graph. The following fact is easily verified:

3.2.2 Lemma. *Let S be a subset of the connector edges.*

1. *If S is nonempty, then there is exactly one cycle cover of the rosette graph containing the edges in S and no other connector edges.*
2. *There are two cycle covers containing no connector edges.*

g_n is a projection of a polynomially large permanent. This means that there is an edge weighted graph G (with the weights being field elements and variables) such that

$$g_n(X_1, \ldots, X_{p(n)}, Y_1, \ldots, Y_{q(n)}) = \sum_{\text{cycle cover } C} w(C).$$

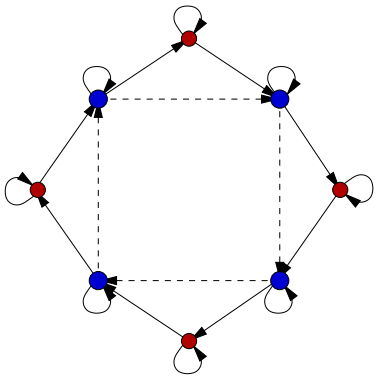

Figure 3.4: The rosette graph of size four. Connector edges are drawn dashed.

Assume that the variable Y_i occurrs ℓ_i times in G. We add a rosette graph of size ℓ_i and connect every edge labeled with Y_i with one of the connector edges of the rosette. All edges inherit their weights from the corresponding subgraphs except that the edges carrying a weight Y_i get the weight 1 instead. We do this for each i. Assume, we introduced t equality gadgets altogether. We will add one isolated self loop with weight $1/2^t$ to compensate for the 2 that is introduced by every equality gadget. (The characteristic of k should be distinct from 2 for this!) Let H be the resulting graph.

Let C be a cycle cover of G. $w(C)$ is a monomial $m(X_1, \ldots, X_n, Y_1, \ldots, Y_{q(n)})$. Let I be the set of indices such that Y_i appears in $w(C)$. What is the contribution of C in

$$\sum_e g_n(X_1, \ldots, X_{p(n)}, e_1, \ldots, e_{q(n)})?$$

If Y_i appears in $w(C)$, then we have to set $e_i = 1$, otherwise, the constribution to the exponential sum will be zero. If Y_i does not appear in $w(C)$, then we can set e_i to 0 or 1. Therefore, the contribution of C is

$$2^{q(n)-|I|} m(X_1, \ldots, X_{p(n)}, 1, \ldots, 1).$$

We call a cycle cover D of H consistent if for every equality gadget, either both edges it connects are chosen or none of them is chosen. A cycle cover C of G can be extended to a consistent cycle cover of H. If an edge with label Y_i appears in C, then we can extend it in one possible way in the corresponding rosette. If no such edge appears in C then there are two ways. In total, there are $2^{q(n)-|I|}$ extensions. By Lemma 3.2.1, we know that

$$\mathrm{per}H = \sum_{\text{consistent D}} w(D).$$

Therefore,

$$\mathrm{per}H = \sum_e g_n(X_1, \ldots, X_{p(n)}, e_1, \ldots, e_{q(n)}).$$

3.2.3 Theorem. *Over fields of characteristic distinct from* 2, per *is* VNP*-complete.*

Proof. It remains to show that per $\in$ VNP. It is quite easy to write a Boolean expression $E(Y)$ of polynomial size which checks whether a given matrix $Y \in \{0,1\}^{n\times n}$ is a permutations matrix. As done before, we can write this as an equivalent arithmetic formula $\hat{E}(Y)$. Now it is easy to check that

$$\mathrm{per}X = \sum_{Y\in\{0,1\}^{n\times n}} \hat{E}(Y) \prod_{i,j} (X_{i,j}Y_{i,j} + 1 - Y_{i,j}).$$

□

Over fields of characteristic 2, the permanent can only be VNP-hard, if VNP = VP, since it conincides with the determinant in this case. But there are other VNP-complete polynomials that are also hard over fields of characteristic two.

3.3 Valiant's conjecture

Valiant's conjecture is the algebraic counterpart of the P versus NP conjecture.

3.3.1 Conjecture (Valiant)**.** VP $\subsetneq$ VNP*.*

Since the permanent is VNP-complete, we can rephrase this conjecture as

$$\text{per} \notin \text{VP}.$$

Since $\text{VP}_{ws} \subseteq \text{VP}$, we can formulate a weaker (or stronger, depending on your point of view) version of Valiant's conjecture, namely, VNP $\subsetneq \text{VP}_{ws}$. Since VP_{ws} has a nice complete family, this version can be reformulated as

$$\text{per} \not\leq_p \det.$$

3.4 Further exercises

3.4.1 Exercise. *(Valiant's criterion, ***) Let ϕ be a function in* #P*. Define a family of polynomials (f_n) by*

$$f_n = \sum_{e \in \{0,1\}^n} \phi(e) X_1^{e_1} \dots X_n^{e_n}.$$

Prove that $(f_n) \in$ VNP*.*

Chapter 4

Determinantal complexity

The question whether $\mathrm{VP}_{ws} = \mathrm{VNP}$ can be rephrased as the question whether det is p-projection of per. Related questions have been studied. One of them is the so-called determinantal complexity.

4.0.1 Definition. *The determinantal complexity $D(f)$ of a polynomial $f \in \mathbb{F}[X_1, \ldots, X_n]$ is the smallest s such that there are affine linear forms $\alpha_{i,j} \in \mathbb{F}[X_1, \ldots, X_n]$, $1 \leq i, j \leq s$, such that we can write $f = \det_s(\alpha_{i,j})$.*

4.0.2 Lemma. *$(f_n) \in \mathrm{VP}_{ws}$ iff $D(f_n)$ is p-bounded.*

Proof. If $(f_n) \in \mathrm{VP}_{ws}$, then it is a p-projection of det. Therefore, its determinantal complexity is p-bounded. For the other direction, note that the determinant has weakly skew circuits of polynomial size. We can compute the affine linear forms by weakly skew circuits of polynomial size. Therefore, (f_n) has weakly polynomial circuits of polynomial size. □

4.1 Mignon-Resssayre bound

In this section, we prove the best lower bound for the determinantal complexity, due to Mignon and Ressayre [MR04].

4.1.1 Observation. *$\frac{\partial}{\partial X_{i,j}} \mathrm{per}_n(X)$ is the permanent of the $(n-1) \times (n-1)$ matrix obtained from X by deleting the ith row and the jth column. The same is true for the determinant.*

For a matrix $A \in \mathbb{F}^{n \times n}$, let $H_{\mathrm{per}}(A)$ denote the $n^2 \times n^2$-matrix with the entry in row (i, j) and column (k, ℓ) being equal to

$$\frac{\partial}{\partial X_{i,j} \partial X_{k,\ell}} \mathrm{per}_n(A).$$

That is, we take the permanent polynomial, differentiate it twice and then we plug in the values from the matrix C

For the proof, we need to construct a matrix A such that $\mathrm{per}(A) = 0$ but $H_{\mathrm{per}}(A)$ has full rank n^2. Let A be the matrix

$$A = \begin{pmatrix} 1-n & 1 & \ldots & 1 \\ 1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 \end{pmatrix},$$

that is, $a_{1,1} = 1 - n$ and all other entries are 1.

4.1.2 Lemma. $\mathrm{per}(A) = 0$.

Proof. Like for the determinant, we can do a Laplace expansion of the permanent. It is even easier, since there are no signs to keep track off. (If you prefer to think in terms of cycle covers, a Laplace expansion along the ith row just groups the cycle covers depending on which node is visisted right after node i.)

So we do a Laplace expansion along the first row. Since all other rows are filled only with 1's, all the submatrices that we get are the same. We get it once multiplied by $1-n$ and $n-1$ times multiplied by 1. So the sum is 0. □

4.1.3 Lemma. *$H_{\mathrm{per}}(A)$ has rank n^2.*

Proof. When $i=j$ or $k=\ell$, then

$$\frac{\partial}{\partial X_{i,j}\partial X_{k,\ell}}\mathrm{per}_n(X)=0.$$

This is due to the fact that every monomial contains only one variable from each row or column. (This property is called set-multilinear).

If $i\neq j$ and $j\neq\ell$, then

$$\frac{\partial}{\partial X_{i,j}\partial X_{k,\ell}}\mathrm{per}_n(X)$$

is the permanent of the submatrix obtained by deleting rows i and k and columns j and ℓ from X. If $1\in\{i,j,k,\ell\}$, then

$$\frac{\partial}{\partial X_{i,j}\partial X_{k,\ell}}\mathrm{per}_n(A)=(n-2)!,$$

since the matrix that we obtain from A after deleting the rows and columns is the all-ones-matrix of size $(n-2)\times(n-2)$, the permanent of which is $(n-2)!$. If $1\notin\{i,j,k,\ell\}$, then

$$\frac{\partial}{\partial X_{i,j}\partial X_{k,\ell}}\mathrm{per}_n(A)=-2(n-3)!,$$

since the matrix that we obtain from A in this case has $n-1$ in position $(1,1)$ and 1's elsewhere. Using Laplace expansion, one can easily see that the permanent of this matrix is $-2(n-3)!$. Therefore, we have that

$$H_{\mathrm{per}}(A)=(n-3)!\begin{pmatrix}0&B&B&\dots&B\\B&0&C&\dots&C\\B&C&0&\ddots&\vdots\\\vdots&\vdots&\ddots&\ddots&C\\B&C&\dots&C&0\end{pmatrix}$$

where

$$B=\begin{pmatrix}0&n-2&\dots&n-2\\n-2&0&\ddots&\vdots\\\vdots&\ddots&\ddots&\vdots\\n-2&\dots&n-2&0\end{pmatrix}$$

and

$$C=\begin{pmatrix}0&n-2&n-2&\dots&n-2\\n-2&0&-2&\dots&-2\\n-2&-2&0&\ddots&\vdots\\\vdots&\vdots&\ddots&\ddots&-2\\n-2&-2&\dots&-2&0\end{pmatrix}.$$

The matrix B has full rank: If we subtract the $(n-1)th$ row from the nth, then the $(n-2)th$ row from the $(n-1)$th and so on until we subtract the first row from the second, we get the matrix

$$(n-2)\begin{pmatrix} 0 & 1 & 1 & \dots & 1 \\ 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & -1 \end{pmatrix}.$$

From the structure of the rows 2 to n it follows that every nontrivial vector in the kernel of the matrix has to be a nonzero multiple of the all-ones-vector. The scalar product of this vector with the first row is however nonzero. Therefore, B has full rank. Doing the same transformation with C, but stopping one rwo earlier, we get the matrix

$$\begin{pmatrix} 0 & n-2 & n-2 & \dots & n-2 \\ n-2 & 0 & -2 & \dots & -2 \\ 0 & -2 & 2 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & -2 & 2 \end{pmatrix}.$$

From the structure of the third to nth row, we ge that the entries at positions 2 to n of every nontrivial vector in the kernel have to be the same. From the first row, it follows that these entries have to be 0. And finally, the second row tells us that also the first entry has to be zero then. Therefore, C is also invertible.

We have

$$\begin{pmatrix} CB^{-1} & 0 & \dots & 0 \\ 0 & I & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & I \end{pmatrix} H_{\mathrm{per}}(A) \begin{pmatrix} CB^{-1} & 0 & \dots & 0 \\ 0 & I & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & I \end{pmatrix} = (n-3)! \begin{pmatrix} 0 & C & C & C\dots & C \\ C & 0 & C & \dots & C \\ C & C & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & C \\ C & C & \dots & C & 0 \end{pmatrix}$$

$$= (n-3)! \begin{pmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \dots & 1 & 0 \end{pmatrix} \otimes C.$$

Since the Kronecker product of two full rank matrices has itself full rank, $H_{\mathrm{per}}(A)$ has full rank. □

4.1.4 Theorem. $D(\mathrm{per}_n) \geq n^2/2$

Proof. Let s be the determinantal complexity of per_n. We know that there are affine linear forms $\alpha_{i,j}(X)$, $1 \leq i,j \leq s$ such that.

$$\mathrm{per}_n(X) = \det_s(\alpha_{i,j}(X)).$$

Let $A = (a_{i,j})$ be the matrix from the previous lemma. Write $\alpha_{i,j}(X) = \lambda_{i,j}(X-A) + y_{i,j}$, where $\lambda_{i,j}$ is a homogeneous linear form and $y_{i,j}$ is a constant, i.e., we perform a translation on the coordinates. Thus,

$$\mathrm{per}_n(X) = \det_s(\lambda_{i,j}(X-A) + y_{i,j}). \tag{4.1.5}$$

Since $\mathrm{per}_n(A) = 0$, we have $\det_s(y_{i,j}) = 0$, so $Y := (y_{i,j})$ is not of full rank. Let S and T be invertible matrices such that

$$SYT = \begin{pmatrix} 0 & 0 \\ 0 & I_t \end{pmatrix}$$

for some $t < s$. Multiplying the matrix on the righthand side of (4.1.5) by

$$\begin{pmatrix} 1/\det S & 0 \\ 0 & I_{s-1} \end{pmatrix} S \quad \text{and} \quad T \begin{pmatrix} 1/\det T & 0 \\ 0 & I_{s-1} \end{pmatrix}$$

from the left and the right, respectively, does not change its determinant. As

$$\begin{pmatrix} 1/\det S & 0 \\ 0 & I_{s-1} \end{pmatrix} SYT \begin{pmatrix} 1/\det T & 0 \\ 0 & I_{s-1} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & I_t \end{pmatrix}$$

(recall that $t < s$), we can assume w.l.o.g. that $(y_{i,j})$ is of this form. Define $H_{\det}$ in the same way as H_{per}. Now we differentiate both sides of (4.1.5). We get

$$H_{\mathrm{per}}(X) = LH_{\det}(\alpha_{i,j}(X - A) + y_{i,j})L^T$$

for some matrix L with entries from $\mathbb{F}$ by the chain rule (see below). Therefore,

$$H_{\mathrm{per}}(A) = LH_{\det}(Y)L^T$$

and

$$\operatorname{rk} H_{\mathrm{per}}(A) = \operatorname{rk} H_{\det}(Y).$$

By the previous Lemma, we know that rank $H_{\mathrm{per}}(A) = n^2$. Therefore, we are done when we show that $\operatorname{rk}(H_{\det}(Y)) \leq 2s$.

Let us first consider the case when $t = s - 1$. An entry of $H_{\det}(y_{i,j})$ is of the form

$$\frac{\partial^2}{\partial X_{e,f} \partial X_{k,\ell}} \det_s(y_{i,j}).$$

This entry can only be nonzero, if differentiating removes the first row and the first column and the hth row and hth column for any other h. This means that

1. $(e, f) = (1, 1)$ and $(k, \ell) = (h, h)$,
2. $(e, f) = (1, h)$ and $(k, \ell) = (h, 1)$,
3. $(e, f) = (h, 1)$ and $(k, \ell) = (1, h)$,
4. $(e, f) = (h, h)$ and $(k, \ell) = (1, 1)$.

In the first case, we get one row with $s - 1$ ones in it. In the fourth case, we get one column with $s - 1$ ones in it. In the second case, we get $s - 1$ different rows, each having a single one and zeros elsewhere. The third case is similar. Altogether, we get that the rank is at most $2 + 2(s - 1) = 2s$. (In fact, equality holds.)

When $t = s - 2$, then we have to delete the first and second row and column, respectively, to get an nonzero entry. Therefore, the rank of the matrix can be at most four, which is less than $2s$.

When $t < s - 2$, then every entry of $H_{\det}(Y)$ will be zero. □

4.1.6 Observation. *Let f be a polynomial in $Y_1, \ldots, Y_m$ variables and $\ell_1, dots, \ell_m$ be affine linear forms in $X_1, \ldots, X_n$. Then by the chain rule*

$$\frac{\partial^2}{\partial X_i \partial X_j} f(\ell_1, \ldots, \ell_n) = \sum_{s=1}^{m} \sum_{t=1}^{m} \frac{\partial f^2}{\partial Y_s \partial Y_t} f(\ell_1, \ldots, \ell_n) \frac{\partial}{\partial X_i} \ell_s \frac{\partial}{\partial X_j} \ell_t$$

Note that $\frac{\partial}{\partial X_i} \ell_s$ and $\frac{\partial}{\partial X_j} \ell_t$ are just constants. Let $L = (\frac{\partial}{\partial X_i} \ell_s)_{1 \leq s \leq m, 1 \leq i \leq n}$. Then

$$\left(\frac{\partial^2}{\partial X_i \partial X_j} f(\ell_1, \ldots, \ell_n) \right) = L \left(\frac{\partial f^2}{\partial Y_s \partial Y_t} f(\ell_1, \ldots, \ell_n) \right) L^T$$

4.2 Grenet's construction

The best upper bound of the determinantal complexity is due to Grenet [Gre12]. It is (of course) exponential. Grenet's construction even writes the permanent as a projection of the determinant. It can be easily described in combinatorial terms (see [BES] for an alterative explanation). We construct a digraph G as follows: The nodes are all subsets of $\{1,\dots,n\}$. We identify $\emptyset$ and $\{1,\dots,n\}$ with each other, so there are 2^n-1 nodes in total. Let S and T be two nodes of G. There will be an edge from S to T with weight $X_{i,j}$ if $|S| = i-1$, $j \notin S$, and $T = S \cup \{j\}$. The node $\emptyset$ will have outgoing edges with weight $X_{1,j}$ to the node $\{j\}$, $1 \le j \le n$ and incoming edges with weights $X_{n,j}$ from the node $\{1,\dots,n\} \setminus \{j\}$. Furthermore, every node except $\emptyset$ gets a self loop of weight 1.

How does a cycle cover of G look like? Edges go only from nodes S to nodes T of larger cardinality. Therefore, the graph is "almost" acyclic, we only get cycles since we identified $\emptyset$ with $\{1,\dots,n\}$. Every cycle has to go through $\emptyset$. So there can be only one cycle which is not a self loop and since $\emptyset$ has no self loop, we have to use one such cycle and cover all other nodes with self loops. Therefore every cycle cover of G has the same number of cycles and therefore the same sign.

4.2.1 Observation. *Cycle covers of G stand in one-to-one correspondance with permutations in S_n.*

This is due to the fact that every cycle simulates the process of adding the numbers $1,\dots,n$ in some particular order to the empty set until we get $\{1,\dots,n\}$. Let π be this order. Then the weight of this cycle is $X_{1,\pi(i)} \cdots X_{n,\pi(n)}$. It follows that

$$\operatorname{per}(G) = \operatorname{per}(X)$$

Since all cycle covers of G have the same sign, $\operatorname{per}(G) = \pm\det(G)$. Thus we haven proven the following theorem.

4.2.2 Theorem. $D(\operatorname{per}_n) \le 2^n - 1$.

4.3 Why are lower bounds hard?

Kayal [Kay12] proved the following theorem.

4.3.1 Theorem. *It is* NP*-hard to decide whether for two given polynomials f and g (even in the sparse representation, that is, as a list of monomials), f is an affine projection of g.*

One can even formulate the symmetric tensor rank problem over $\mathbb{F}$ as a projection problem. This one has recently shown to be equivalent to the existential theory over the corresponding field, see [BRS17] for more details.

Bibliography

[AW16] Eric Allender and Fengming Wang. On the power of algebraic branching programs of width two. *Computational Complexity*, 25(1):217–253, 2016.

[BC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

[BES] Markus Bläser, David Eisenbud, and Frank-Olaf Schreyer. Ulrich complexity. *Differential Geometry and Applications*. to appear.

[Blä01] Markus Bläser. Complete problems for Valiant's class of qp-computable families of polynomials. In Jie Wang, editor, *Computing and Combinatorics, 7th Annual International Conference, COCOON 2001, Guilin, China, August 20-23, 2001, Proceedings*, volume 2108 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2001.

[Bre74] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974.

[BRS17] Markus Bläser, Raghavendra Rao, and Jayalal Sarma. Testing polynomial equivalence by scaling matrices. In *Proc. 21st Int. Symp. on Fundamentals of Computation Theory*, 2017. to appear.

[Bür00] Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Springer, 2000.

[DMM+14] Arnaud Durand, Meena Mahajan, Guillaume Malod, Nicolas de Rugy-Altherre, and Nitin Saurabh. Homomorphism polynomials complete for VP. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 493–504. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[Gre12] Bruno Grenet. An upper bound for the permanent versus determinant problem, 2012.

[Kay12] Neeraj Kayal. Affine projections of polynomials: extended abstract. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 643–662. ACM, 2012.

[MP08] Guillaume Malod and Natacha Portier. Characterizing valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008.

[MR04] Thierry Mignon and Nicolas Ressayre. A quadratic bound for the determinant and permanent problem. *International Mathematics Research Notices*, 2004(79):4241–4253, 2004.

[MV97] Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.

[vzG87] Joachim von zur Gathen. Feasible arithmetic computations: Valiant's hypothesis. *J. Symb. Comput.*, 4(2):137–172, 1987.