Reductions between Families of Polynomials in Theory and in Practice

David Urbanik

University of Waterloo

Abstract. In 1979, Valiant introduced a complexity theory for classes of polynomial families. In particular, he showed that it was possible to define a notion of "completeness", analogous to the classical theory of **NP**-completeness, for families of polynomials, and showed that the determinant and permanent polynomials from linear algebra were complete in this sense. His theory accomplishes this by establishing a notion of reduction between families of polynomials, and then shows that it is possible to reduce computations of a wide range of polynomial families to computations of determinant and permanent polynomials. In this paper, we give an overview of how these reductions work, what they mean theoretically, and present some original analysis of the efficiency of computing polynomials via reductions to the determinant polynomial.

1 Introduction

The field of *Algebraic Complexity Theory* has seen a flurry of activity in recent years. Its origins date back to a seminal 1979 paper [11] of Valiant, in which he proved several "completeness theorems" for families of polynomials over a fixed field. These theorems formalize a notion of reduction, and can be used to demonstrate the relative easiness or hardness of certain computations involving polynomials in analogy with the classical theory of **NP**-completeness and the **P** vs. **NP** problem. In the modern formulation, Valiant's work gives rise to the so-called **VP** vs. **VNP** problem, and a rich theory which contains many results which are "parallel" to the classical theory.

In this work, we give a brief introduction to the key ideas in Valiant's theory. We also take a closer look at Valiant's determinant reduction, and investigate the question of whether there exist natural classes of polynomials which may be represented efficiently by determinant computations on matrices.

2 Valiant's Determinant Reduction

In his 1979 paper, Valiant investigated the question of when and how computations of certain "polynomial families" could be reduced to the computations of others. The definitions that follow will make these notions precise. We will denote by $\mathbb{F}$ an arbitrary field, which we will fix throughout, and by $\mathbb{F}[x_1, \ldots, x_n]$ the ring of polynomials over $\mathbb{F}$ in the n indeterminates $x_1, \ldots, x_n$.

Definition 1. A *polynomial family* is a function

$$P : \mathbb{Z}_{>0} \to \mathbb{F}[x_1, x_2, \ldots] \tag{1}$$

where the right-hand side is the ring of polynomials over $\mathbb{F}$ in countably many variables.

We write P evaluated at n as P_n, which we interpret as the nth polynomial in the family P. Typically, the polynomials P_n will have total degree strictly increasing with n. The next two examples are by far the two most important polynomial families in Valiant's theory.

Example 1. Let DET denote the family of determinant polynomials, where

$$\mathrm{DET}_n = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} x_{i,\sigma(i)} \tag{2}$$

and by S_n we mean the set of all permutations of the set $\{1, \ldots, n\}$.

Example 2. Let PERM denote the family of permanent polynomials, where

$$\mathrm{PERM}_n = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)} \tag{3}$$

To be able to speak of when the computation of one polynomial family may be reduced to another, and just how efficient such a reduction may be, we need to formalize these notions. The next several definitions will be sufficient for our purposes in this section, and we will give a more nuanced treatment of these ideas in a later section.

Definition 2. A *formula* is one of the following:

(i) An element $a \in \mathbb{F}$ or a variate x_i.
(ii) $(f_1 + f_2)$ where f_1 and f_2 are formulas.
(iii) $(f_1 \cdot f_2)$ where f_1 and f_2 are formulas.

We will think of formulas as specifying a method of computing a polynomial.

Definition 3. The size $|f|$ of a formula f is the total number of $+$ and $\cdot$ operations used to compute it. The *formula size* $|P_n|$ of a polynomial P_n is the size of the minimal formula computing P_n.

Definition 4. We say that a polynomial $f \in \mathbb{F}[x_1, \ldots x_n]$ is a *projection* of $g \in \mathbb{F}[y_1, \ldots, y_m]$ if $f(x_1, \ldots, x_n) = g(a_1, \ldots, a_m)$ where each $a_i \in \mathbb{F} \cup \{x_1, \ldots x_n\}$. Informally, f is obtained from g by substituting indeterminates and field elements.

Initially, we may be concerned that our notion of reduction is too restrictive. For instance, in Definition 4, we could allow arbitrary linear combinations of the variates as arguments to g, or perhaps even polynomials of the variates. However, we will see in a later section that the relevant theory is only concerned that the size of these reductions remains "polynomially bounded" (in a sense that we will make precise), and hence we could easily redefine such a g so that f is obtained from g as described by Definition 4 while keeping any desired "polynomial boundedness" properties. Thus, the definition is sufficient.

Our goal for the remainder of this section will be to give a constructive proof of the following result of Valiant.

Theorem 1. *If $f \in \mathbb{F}[x_1, \ldots x_m]$, then f is a projection of DET_n where $n = |f| + 2$.*

Proof. Our argument will proceed by three observations[1], which we will consider in turn:

(i) If we view the entries of an $n \times n$ matrix as the edgeset of a directed graph on n vertices, graphs where all cycle covers contain only odd cycles exhibit a correspondence between those cycle covers and the terms in the sum-over-permutations expansion of the determinant.
(ii) It is possible to associate to each formula a graph where the computation of the formula may be viewed as a "sum over paths" in a natural way.
(iii) We can modify the graphs from (ii) to the form required by (i) so that the summations coincide.

To show (i), we first recall the standard result that every permutation $\sigma \in S_n$ has a unique *cycle decomposition*, that is, we can write σ as product of disjoint cyclic permutations in a way which is unique up to reordering. Suppose we consider the complete directed graph G on $\{1, \ldots, n\}$, where the edge (i, j) is weighted by $x_{i,j}$. Then the summation

$$\mathrm{DET}_n = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} x_{i,\sigma(i)} \tag{4}$$

can be thought of as a sum over all *cycle covers* of G (partitions of the vertex set so that each element of the partition corresponds to the vertex set of a directed cycle), as the cycle covers of G are in 1-to-1 correspondence with the permutations via the cycle decomposition of σ. To facilitate the reduction in part (iii), we will wish to ensure that all the terms in this summation have the

[1] We have chosen to avoid making these "observations" into precise statements, although it is not difficult to do so, because they correspond more to general techniques which can typically be done in more than one way. For instance, there are several possible ways of achieving (ii) and (iii), and although we will eventually fix a particular one to achieve the precise bound in the statement of the theorem, one should not come away with the idea that this is the only way to do things.

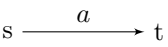
same sign. We may accomplish this by setting the weight of an edge which is a member of an even length cycle to zero, effectively "removing" it from the graph, and continuing in this way so that the only cycles which remain in the graph (i.e., those for which all edges have non-zero weight) are ones with odd length[2]. Since the sign of an odd-length cyclic permutation is 1, and the sign of a permutation is a multiplicative function on the permutation group, it follows that all the terms in DET_n will have positive sign. Hence, we get for these graphs that

$$\mathrm{DET}_n = \sum_{\substack{\text{odd cycle} \\ \text{covers}}} \prod_{\substack{\text{edges in} \\ \text{cover}}} \text{weight on edge} \tag{5}$$

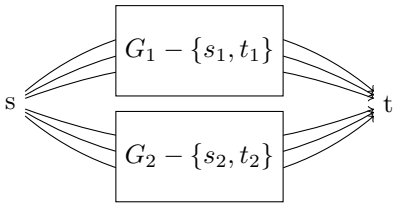
Our goal is thus reduced to constructing appropriate graphs for which such a summation computes our polynomial.

Part (ii) of the theorem proceeds by associating to each formula a directed graph from a source node s to a sink node t where the evaluation of the formula may be obtained by "summing over" the paths from s to t. It is easiest to define this alongside the recursive definition of a formula, which we do as follows.

1. If our formula is simply an element $a \in \mathbb{F} \cup \{\text{indeterminants}\}$, we have the graph



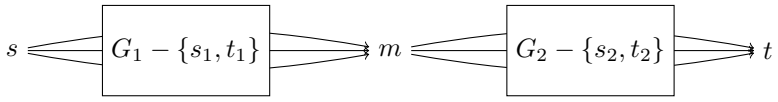
2. If our formula f is $f = f_1 + f_2$, where f_1 is represented by (G_1, s_1, t_1) and f_2 is represented by (G_2, s_2, t_2), we have



   That is, we construct a new graph by identifying s_1 with s_2 (and call it s) and t_1 with t_2 (and call it t) to get (G, s, t). It is clear that the directed paths from s to t in G is exactly the "sum" of the directed paths from s $(= s_1 = s_2)$ to t $(= t_1 = t_2)$ in the graphs G_1 and G_2.

3. If our formula is $f = f_1 \cdot f_2$, with f_1 and f_2 represented by (G_1, s_1, t_1) and (G_2, s_2, t_2) respectively, then we may represent f via

[2] Note that although every edge is in a cycle with even length initially, many of those cycles will be destroyed in the process of removing edges, and so we will eventually reach graphs with no even length cycles. Of course, the order of removal matters here.



where we have identified t_1 and s_2 to obtain m, and set $s = s_1$ and $t = t_2$.

We now claim that the graph (G, s, t) associated to a formula f via this procedure can be used to express the polynomial represented by f as

$$\sum_{\substack{\text{paths} \\ s \to t}} \prod_{\substack{\text{edges on} \\ \text{path}}} \text{weight on edge} \tag{6}$$

It is easy to see this holds inductively for case 1 and case 2. For case 3, it suffices to observe that any path from s to t can be uniquely expressed as a combination of a path from s to m and a path from m to t, and hence the identity

$$\left(\sum_{\substack{\text{paths} \\ s \to m}} \prod_{\substack{\text{edges on} \\ \text{path}}} \text{weight} \right) \cdot \left(\sum_{\substack{\text{paths} \\ m \to t}} \prod_{\substack{\text{edges on} \\ \text{path}}} \text{weight} \right) = \sum_{\substack{\text{paths} \\ s \to t}} \prod_{\substack{\text{edges on} \\ \text{path}}} \text{weight} \tag{7}$$

holds. To complete the proof, it suffices to show that we can modify our graph (G, s, t) associated to f to make equation (6) have the form of equation (5). As it stands, this amounts to justifying the replacement of the words "paths from $s \to t$" with "odd cycle covers". This is step (iii) in our proof.

We now imagine adding to our graph (G, s, t) a self-loop weighted 1 at each vertex which is not s or t, and a single edge from t to s weighted 1 (alternatively, one can identify t and s). The cycle covers of the resulting graph would then be in direct correspondence with the paths from s to t of the original, with a single non-trivial cycle corresponding to each path. However, we are not guaranteed that all the non-trivial cycles will have odd length. In order to ensure this is the case, we would need to ensure that the lengths of all the paths from s to t have the same parity, and then either add an edge from t to s weighted 1 or identify t and s as required. To ensure this parity condition holds, we can modify cases 1, 2, and 3 as follows.

1. Case 1 is as before.
2. In case 2, let r_1 be the parity of the path lengths of (G_1, s_1, t_1) and r_2 be the parity of the path lengths of (G_2, s_2, t_2) (this is well-defined by induction). If $r_1 = r_2$, we proceed as before. If (say) $r_1 = 0$ and $r_2 = 1$, then we add an edge weighted 1 from t_1 to a new vertex t_1', and then the resulting graph will have parity $r_1' = r_1 + 1 = r_2$ and we may proceed as before with (G_1, s_1, t_1') and (G_2, s_2, t_2). We work analogously if $r_1 = 1$ and $r_2 = 0$.
3. Case 3 is as before.

It now suffices to establish the resulting graph has at most $|f| + 2$ vertices. Let $|V(G)|$ be the number of vertices in G. We get the following relationships for cases 1, 2, and 3:

1. $|V(G)| = 2$.
2. We add at most one extra vertex to one of the graphs, and then identify two pairs of vertices. Thus $|V(G)| \leq |V(G_1)| + |V(G_2)| - 1$.
3. We identify one pair of vertices, so $|V(G)| \leq |V(G_1)| + |V(G_2)| - 1$.

We note that formula size obeys a similar relationship, i.e., if $f = f_1 + f_2$ or $f = f_1 \cdot f_2$, then $|f| = |f_1| + |f_2| + 1$. Since $|V(G)|$ is the largest when equality holds in cases 2 and 3, it will be the largest when it agrees with $|f| + 2$ (since in case 1 $|f| = 0$ and $|V(G)| = 2$), and so we need at most $|f| + 2$ vertices. This completes the proof. □

3 Efficient Representations of Polynomials via Matrices

The reduction given in the previous section is clearly not optimal. For instance, the polynomial $f(x) = 3x^2+2x+5$ has a minimal formula of $(5+x\cdot(2+(3\cdot x)))$, so $|f| = 4$. But it is the determinant of the following 4×4 matrix, and $4 < |f|+2 = 6$.

$$f(x) = \det \begin{pmatrix} 5 & x & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 0 \\ x & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

An intriguing possibility is that, in some cases, determinants of matrices might be able to represent polynomials more efficiently than a naïve evaluation. This is certainly the case for the determinant polynomial: evaluating the determinant polynomial can be done in $O(n^3)$ time in practice, simply by filling the corresponding matrix and using a number of available determinant computation algorithms. Evaluating the determinant polynomial (equation 2) directly, however, takes time $\Omega(n!\ n)$, which is dramatically slower.

In fact, it is easy to generate many such examples – simply randomly fill the entries of a matrix with (say) monomials in multiple variables, and the resulting polynomial will have an exponential number of terms with high probability. The question we are interested in, then, is whether or not there are any natural examples. That is, can determinants be used to represent and evaluate polynomials which are practically interesting?

A possible approach to this problem proceeds via the following observations:

(i) If $f = \det A$, and $g = \det B$, where A and B are both $n \times n$ matrices, then $f \cdot g = \det AB$.

(ii) If $A' = PA$, where P is any permutation matrix, then $\det A' = \pm \det A$. The same holds if $A' = AP$. Hence inserting permutation matrices anywhere into the product AB will result in a matrix with the required determinant up to a sign.

(iii) More generally, we may consider some set S of $n \times n$ matrices, whose determinants all lie in some subgroup of the multiplicative group of the underlying field $\mathbb{F}$. Then given some sequence of polynomials $f_i = \det A_i$,

we may obtain a representation of $\prod_i f_i$ by considering a product of the A_i's, in any order, with any number of elements of S inserted anywhere in the product. The result will be a matrix representing $(\alpha \prod_i f_i)$ for $\alpha \in \mathbb{F}$, and we divide out α from a row of the resulting matrix to obtain what we want.

Polynomials which are built as products of smaller polynomials appear naturally in many situations, and it is conceivable that by iterating through the many different possible products and combinations offered by (ii) and (iii) that we may find matrices for which the resulting matrix representation of the polynomial is better than the naive one[3]. In the coming sections, we follow the technique outlined by (i), (ii) and (iii) for the explicit case of products of polynomials given by Horner formulas and test this hypothesis.

3.1 Products of Horner Polynomials

To see how efficient matrix representations of polynomials can be in practice, it is important to find efficient formulas for polynomials to maximize the efficiency of the reduction in Theorem 1. A simple case is the known optimality of *Horner's Method*. Given a polynomial $f(x) = \sum_{k=0}^{n} a_k x^k$, Horner's method of computing f is to recursively compute

$$b_n := a_n \ , \quad b_{n-k} := a_{n-k} + x b_{n-k+1} \qquad 1 \leq k \leq n$$

The result will be $b_0 = f(x)$. It is clear from the definition that a Horner formula has size $2n$. In fact, the following result is true.

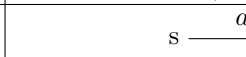
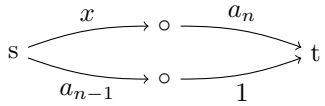
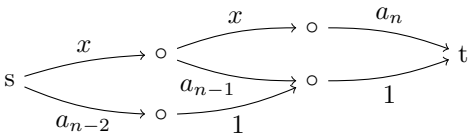
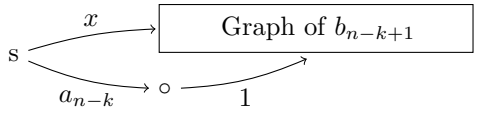
Theorem 2. *For single variate polynomials over a field $\mathbb{F}$, the Horner formula is optimal.*

The proof is due to the combination of results due to Ostrowski [8] and Pan [9], who separately proved that the number of additions and the number of multiplications, respectively, is optimal for Horner formulas. Together, their results imply that the overall size of Horner formulas is also optimal, as any formula of smaller size would have to have either fewer additions or fewer multiplications.

We now apply the technique of Theorem 1 to the explicit case of Horner formulas. Following the technique in Part (ii) of Theorem 1, and ensuring that all the paths from the source to the sink node have the same parity, we obtain the following graphs and matrices. The column labeled "Matrix (before)" shows the matrix obtained by labeling the vertices from right to left, top to bottom, and using the matrix to describe the resulting edgeset. The column labeled "Matrix (after)" shows the final matrices obtained after adding self-loops where necessary and either identifying s and t, or adding an edge weighted 1 from t to s (depending on the parity of the paths).

[3] By "naïve" representation, we mean the representation where the polynomial is fully expanded, rather than its factored form.

Table 1: Applying the reduction of Theorem 1 to Horner Formulas.

Formula	Graph (in path form)	Matrix (before)	Matrix (after)
a_n		$\begin{bmatrix} 0 & 0 \\ a_n & 0 \end{bmatrix}$	$\begin{bmatrix} a_n \end{bmatrix}$
$(a_{n-1} + (x \cdot a_n))$		$\begin{bmatrix} 0 & 0 & 0 & 0 \\ a_n & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & x & a_{n-1} & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 1 \\ a_n & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & x & a_{n-1} & 0 \end{bmatrix}$
$(a_{n-2} + (x \cdot b_{n-1}))$		$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ a_n & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & a_{n-1} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & a_{n-2} & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & x & a_{n-2} \\ a_n & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & x & a_{n-1} & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(a_{n-k} + (x \cdot b_{n-k+1}))$		$\begin{bmatrix} \begin{bmatrix} \text{Matrix} \\ \text{of } b_{n-k+1} \end{bmatrix} & \vdots & \vdots \\ \cdots \quad 1 \; 0 & 0 & 0 \\ \cdots\cdots \; x & a_{n-k} & 0 \end{bmatrix}$	Varies with parity

From the table, we may derive the number of non-zero entries in the final matrix as follows. Let s_k be the number of entries in the final matrix in the kth stage, but before the last step of either identifying s and t or drawing an edge from t to s. Hence $s_0 = 1$. We notice that at each stage, we add three edges and two vertices to the "path form" of the graph. The three edges correspond to three additional non-zero entries in the "before" matrix, and the two additional vertices correspond to two additional diagonal ones in the "after" matrix. Ignoring the final step where we either identify t and s or add an edge from t to s, this gives the recurrence $s_{k+1} = s_k + 5$, and thus $s_n = 5n + 1$. Taking into account an extra edge if the parity of n is 1, we find that the total number is $5n+1+\text{parity}(n)$ non-zero entries in a matrix representing a polynomial of degree n. Since Valiant's determinant reduction from a formula of size s produces a matrix with $\Theta(s^2)$ entries, we see that for Horner formulas, a $\Theta(s)/\Theta(s^2) = \Theta(1/s)$ fraction of those entries will be non-zero. Asymptotically, this means that for large degree polynomials the corresponding matrices will be very sparse, so we can hope to compute the product of a large number of such matrices, possibly $O(s)$ of them, and have the average formula size of the matrix entries remain small.

3.2 Computational Results

To test our hypothesis, we wrote computer software to translate arbitrary formulas to their matrix form, apply random permutations to those matrices, compute their products, and search for matrix representations of those products where the total formula size (defined below) is minimal. As a toy example, the software can be given the Horner formulas $(a_0 + x \cdot (a_1 + x))$ and $(b_0 + y \cdot (b_1 + y))$ and derive the following identity in a matter of milliseconds.

$$(x^2 + a_1x + a_0)(y^2 + b_1x + b_0) = \det \begin{pmatrix} y & 2 & 1 & b_1 \\ x & a_0 & a_0 & x + b_1a_0 \\ b_0 & x & x + y & b_1x \\ a_1y + b_0 + 1 & a_1 & y & 1 \end{pmatrix} \quad (9)$$

Definition 5. The *total formula size* of an $n \times n$ matrix $A = [a_{ij}]$, where a_{ij} are multivariate polynomials, is $\sum_{1 \leq i,j \leq n} |a_{ij}|$, that is, the sum of the minimal formula size of all of its entries.

For instance, the total formula size of the matrix in equation 9 is 7.

It is worth remarking on the relevance of total formula size. Computing the determinant of an $s \times s$ matrix can be done in practice in $O(s^3)$ time. By taking the product of arbitrarily many matrices of dimension $s \times s$ which represent polynomials, we may form a product matrix which represents a product polynomial of arbitrarily large size, and potentially one for which a naïve computation is less efficient than computing the corresponding determinant. However, forming product matrices in this way will result in matrices for which the entries are themselves polynomials in the variates, and so there is extra computational overhead which must be taken into account when evaluating the efficiency of the resulting representation. Total formula size is a measure of this computational overhead.

We will also be interested in the *minimal* total formula size possible for classes of polynomials and matrices of a fixed size (i.e., the size resulting from the procedure in Table 1). In this case, it is difficult to (provably) find matrix representations which are optimal with respect to total formula size as the degree of the polynomials increases. For instance, if we apply the method of (ii) and (iii) to search for these matrix representations, we are forced to apply random permutations to the constituent matrices, and the number of such permutations grows as $\Omega((2n)!)$ where n is the degree of the polynomials in question. In general it does not seem to be possible to find the minimum total formula size without actually trying all these possibilities, but we can nevertheless obtain a reasonable upper bound on the minimum total formula size by repeatedly randomly sampling the possible permutations and taking the matrix product. The upper bound on minimum total formula size will then be "reasonable" in the sense that it indicates what is the minimum formula size one can expect to find in practice using a bounded amount of computational resources.

To make this more concrete, we performed the following computational experiment.

(i) Define $T_{n,k}$ to be the minimal total formula size possible for a matrix representation of the product of k single-variate polynomials of degree n with explicit non-zero[4] integer coefficients, whose individual matrix representations are obtained using the procedure described in Table 1 and modified using permutation matrices. Note that the product polynomials here have k distinct variates, since each of the variates in the factors are assumed to be distinct.

(ii) We may obtain an upper bound on $T_{n,k}$ by selecting k degree-n polynomials at random, applying the transformation in Table 1 to all k of them, applying a random permutation matrix to each of the resulting matrices, taking the product, and then computing the total formula size.

(iii) If we repeat (ii) a number of times (say 10^6), the minimal value obtained should be a "reasonable" upper bound on $T_{n,k}$, where "reasonable" is in the sense described earlier.

Example 3. The following identity obtained in this fashion gives an upper bound of $T_{3,2} \leq 2$.

$$(4r^3 - 3r^2 + 2r + 1)(3s^3 - s^2 + 4s + 2) = \det \begin{pmatrix} 2 & 0 & 1 & 0 & 0 & 0 & s \\ 0 & -1 & r & 1 & 3 & 0 & 0 \\ 2 & 0 & 0 & 0 & r & sr & s \\ 0 & 0 & 0 & s & 0 & 4 & 1 \\ -3 & -3 & 0 & 4 & 12 & -2 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & r & 2 & 0 & 1 & r+s & 0 \end{pmatrix} \tag{10}$$

Steps (i) through (iii) were performed for the (n, k) values in the following table. Concretely, for each value of (n, k), 300 random sets of k polynomials were selected, and for each of those sets $3 \cdot 10^5$ different choices of k permutations were applied to the resulting matrices before taking a product. This gave a total of $9 \cdot 10^7$ upper bounds on $T_{n,k}$, the smallest of which appears in the table. For simplicity, coefficients were sampled from -7 to 7, with zero excluded (since we require the coefficients to be non-zero).

[4] This condition is required to avoid trivial cases, like representing polynomials of the form $f(x) = x^n$, which can skew the results.

Table 2: Upper bounds on $T_{n,k}$.

	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$
$n=1$	0	2	6	17	42	96
$n=2$	4	20	69	186	442	962
$n=3$	1	14	70	191	459	1062
$n=4$	4	45	180	524	1314	3081
$n=5$	3	32	150	462	1116	2723
$n=6$	8	70	288	876	2187	5491
$n=7$	6	52	232	740	1936	4565
$n=8$	11	94	386	1183	3215	8302
$n=9$	7	71	317	1031	2715	6642
$n=10$	15	118	481	1503	4144	10039
$n=11$	12	93	389	1294	3592	8587
$n=12$	18	146	572	1888	5108	12849
$n=13$	17	115	460	1514	4292	11292
$n=14$	21	162	661	2181	6237	15542
$n=15$	17	133	552	1706	5536	12937
$n=16$	24	188	781	2497	7114	18496
$n=17$	22	153	636	2072	5784	15878
$n=18$	28	211	873	2867	8110	20897

The results listed in Table 2 are not as bad as they may seem. For instance, consider the case for $n = 11$ and $k = 7$. In this case, we have 7 univariate polynomials of degree 11, whose product is a polynomial in 7 variates. Since each factor has 12 terms, the naïve expansion of the product polynomial has 12^7 terms, and hence a naïve evaluation requires at least 12^7 operations (far more in practice) and storing 12^7 field elements. Table 2 however gives the bound $T_{11,7} \leq 8587$, which shows that it is possible to achieve a determinant representation for some such polynomials where the overhead is only on the order of $10^4 \ll 12^7$. Since evaluating the determinant requires $O(n^3)$ operations (here $n = 11$), both evaluating the polynomial and storing the polynomial is actually significantly more efficient for the matrix representation than the naïve version. The same holds for the other entries in the table.

The author is aware that there is no requirement that users seeking to evaluate polynomials do so naïvely. Indeed, they could simply use the factored version and do so efficiently. However, we suggest three reasons why this work is justified.

The first is standard: the work is of theoretical importance. The second is that we have demonstrated that there exist reasonably efficient matrix representations of "naturally occurring" polynomials – polynomials that someone might conceivably want to compute – and so searching for matrix representations of polynomials may be feasible for some problems, if not for this specific case. The third is on the basis of *numerical stability*. Namely, when working over infinite fields such as $\mathbb{R}$ or $\mathbb{C}$, often one wants to avoid multiplications so as not to run into issues relating to the inaccuracy of floating-point approximations to those fields. Determinants can be computed in a way that ensures that the formula

used is not "heavily-factored", in the sense that they do not use many consecutive multiplications, so it is possible that in some cases matrix representations of polynomials may turn out to be more numerically stable than heavily-factored versions.

4 Algebraic Complexity Classes

Although it did not appear in Valiant's original paper, Valiant's work laid the foundations for what we now call the **VP** vs. **VNP** problem. To understand just what this problem says, we must first generalize our model for computing polynomials to that of *arithmetic circuits*. To motivate this, consider the computation of the polynomial $f(x) = x^4$. If we are interested in the minimal size formula for f, then we can do no better than 3 multiplications. However, in practice, computing f requires only two: $x^2 = x \cdot x$, and $x^4 = x^2 \cdot x^2$. The extra power is the ability to reuse previously computed values, in this case the value of x^2, which makes the model both more practical and leads to more efficient programs. Moreover, the approach of considering arithmetic circuits rather than formulas results in a much tighter relationship between algebraic and classical complexity theory. For instance, we may simulate a NAND gate with inputs x and y in $\{0, 1\}$ as the polynomial $1 - xy$, and hence efficiently recover boolean circuits.

Definition 6. An *arithmetic circuit* over $\mathbb{F}$ is a directed acyclic graph satisfying:

(i) Each node of indegree 0 (called an "input gate") is labeled by either an indeterminate or an element of $\mathbb{F}$.
(ii) All other nodes have indegree 2 and are labeled by either $+$ or $\times$, and represent the sum or product of their inputs respectively.
(iii) There is a single node of outdegree 0 called the "output gate".

As a special case, we can see that formulas are arithmetic circuits where every non-output node has outdegree 1. The number of non-input nodes is called the *size* of the circuit, which we may see is a generalization of the notion of size for formulas.

Definition 7. The complexity $C(f)$ of a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ is the size of the minimal arithmetic circuit computing f.

When dealing with complexity classes like **VP** and **VNP**, and considering the completeness results for polynomial families like DET and PERM, we will see that we will need to have a measure of size for polynomial families rather than just for individual polynomials. This is analogous to how classical complexity theory has a measure of size, namely *asymptotic complexity*, for entire problems rather than just for individual instances. Similar ideas motivate the definitions that follow.

Definition 8. We say that a function $t : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ is *p-bounded* if $t \in O(n^k)$ for some $k \in \mathbb{Z}_{>0}$. We will avoid using the term "polynomial" to describe such functions so as not to cause confusion with "polynomial families".

Definition 9. We say that a polynomial family P is a *p-projection* of a polynomial family Q if there exists a p-bounded function $t : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that P_i is a projection of $Q_{t(i)}$ for all i, and if $\deg Q_i$ is a p-bounded function of i. We similarly define qp-projection (quasi-polynomial projection) and exponential projection by the appropriate conditions on t and $\deg Q_i$.

Definition 10. We say that a polynomial family P is *p-computable* (resp. qp-computable), if $C(P_i)$ is a p-bounded (resp. qp-bounded) function of i.

Definition 11. We define **VP** to be the set of all p-computable polynomial families. Strictly speaking we are defining $\mathbf{VP}_{\mathbb{F}}$, but since $\mathbb{F}$ is fixed throughout, we will simply refer to **VP**.

Definition 12. Similarly, we define **VQP** to be the set of all qp-computable polynomial families.

We now consider what our result Theorem 1 tells us about the polynomial families in **VP** and **VQP**. Firstly, we know from standard algorithms for computing DET_n that DET has polynomial-sized arithmetic circuits, and so $\mathrm{DET} \in \mathbf{VP} \subseteq \mathbf{VQP}$. We would now like to ask whether or not DET is *complete* for any of these classes. That is, we define:

Definition 13. We say a polynomial family P in a complexity class $\mathcal{C}$ is $\mathcal{C}$-*complete* if for any $Q \in \mathcal{C}$, there exists a p-projection from Q to P. Sometimes we will wish to consider projections which are qp-bounded, in which case we will say "complete with respect to qp-projections".

To answer this question, we need to understand the relationship between the smallest formula for a given polynomial and the smallest arithmetic circuit. A result of Hyafil [4] shows that for any arithmetic circuit of size s, there exists a formula of size $s^{O(\log d)}$ computing the same polynomial, where d is the degree. Hence, Theorem 1, which tells us that there is a p-bounded projection between any family of formulas and DET, tells us that there is a qp-bounded projection between any family of arithmetic circuits and DET. Hence, the conclusion is not that DET is **VP**-complete, but it is **VQP**-complete with respect to quasi-polynomial projections.

A yet more important result turns out to be the relationship between PERM and **VNP**. To this end, we turn our attention to the notion of *p-definability*.

Definition 14. Suppose that P is a polynomial family, where $P_n \in \mathbb{F}[x_1, \ldots, x_{u(n)}]$. Then we say that P is *p-definable*, if there exists a p-computable family Q, $Q_n \in \mathbb{F}[x_1, \ldots, x_{v(n)}]$ where $u(n) \leq v(n)$ for all n, and

$$P_n(x_1, \ldots, x_{u(n)}) = \sum_{e \in \{0,1\}^{v(n)-u(n)}} Q_n(x_1, \ldots, x_{u(n)}, e_{u(n)+1}, \ldots, e_{v(n)}) \quad (11)$$

This definition initially appears quite mysterious, but it is motivated by an observation by Valiant. In his original paper [11], Valiant pointed out that there appear to be a number of naturally occurring polynomials for which no efficient formulae are known, and observed that in almost all cases such polynomials arise because we have an efficient description for *their coefficients* but not for the polynomial itself. Hence, the p-computable polynomials Q_n in the definition can be thought of as simply the polynomials which efficiently compute the monomials in the desired polynomial; the input e is simply an index which specifies which term is being computed, of which there may be exponentially many. Note that it is tempting to replace the summation over vectors e by a summation over the integers 0 through $2^{v(n)-u(n)} - 1$, but such a definition is not universal since we are only ever guaranteed that 0 and 1 exist in our field $\mathbb{F}$.

We may now define **VNP**.

Definition 15. The class **VNP** is the set of all p-definable polynomial families.

It is clear that $\mathbf{VP} \subseteq \mathbf{VNP}$, since we may construct Q_n in Definition 14 so that Q_n has a factor which is 1 if e is the all 1s string and zero otherwise, and such that the other factor of Q_n is any p-computable polynomial corresponding to a family $Q' \in \mathbf{VP}$. A more interesting example is the following.

Lemma 1. PERM $\in \mathbf{VNP}$.

Proof. We use an adaptation of Valiant's original argument, which is to consider two matrices of indeterminants x_{ij} and y_{lm}, with $1 \leq i, j, k, l \leq n$, and to define

$$Q_n := \underbrace{\left(\prod_{\substack{i,j,l,m \\ i=l \text{ if } j \neq m}} (1 - y_{ij} y_{lm}) \right) \left(\prod_{i=1}^{n} \sum_{j=1}^{n} y_{ij} \right)}_{R_n} \left(\prod_{i=1}^{n} \sum_{j=1}^{n} x_{ij} y_{ij} \right) \tag{12}$$

The key idea is that if $[y_{ij}]$ is a matrix consisting of only 0s and 1s, then R_n is non-zero if and only $[y_{ij}]$ is a permutation matrix. We can see this by observing that the first factor of R_n is non-zero if and only if every row and column contains at most one 1, and given that the first factor is non-zero, the second is non-zero if and only if every row contains at least one 1. Hence R_n ensures that for $e \in \{0,1\}^{n \times n}$, we have

$$Q_n(x, e) = \begin{cases} 0 & \text{if } e \text{ is not a permutation matrix} \\ 1 \cdot \prod_{i=1}^{n} x_{i\sigma(i)} & \text{if } e \text{ describes the permutation } \sigma \end{cases} \tag{13}$$

Since Q is a p-computable family (the number of additions and multiplications is polynomial in n), we see from Definition 14 that PERM is p-definable, as

$$\text{PERM}_n = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i\sigma(i)} = \sum_{e \in \{0,1\}^{n \times n}} Q_n(x, e) \tag{14}$$

□

We already see from Lemma 1 that working with p-definability directly is rather tedious, even for rather simple examples like PERM. However, the following theorem of Valiant makes things easier.

Theorem 3. *If $\phi : \{0,1\}^* \to \mathbb{N}$ is a function in $\#\boldsymbol{P}$, then the family P defined by*

$$P_n := \sum_{e \in \{0,1\}^n} \phi(e) \prod_{i=1}^{n} x_i^{e_i} \tag{15}$$

is p-definable.

Proof. We give a similar argument to Prop 2.20 in [1]. By the standard **NP**-completeness argument for SAT, we may assume that there exists a polynomially-sized Boolean formula ψ_n so that

$$\phi(e) = \#\{y \in \{0,1\}^{\alpha(n)} : \psi_n(e,y) \text{ is true}\}, \tag{16}$$

where $\alpha(n)$ is a p-bounded function. By using polynomials to emulate logical connectives (e.g., $1 - xy$ is NAND, as discussed before), we may build a polynomial R_n which on binary vectors e and y of the prescribed size, $R_n(e,y) = 1$ if $\psi_n(e,y) = 1$, and $R_n(e,y) = 0$ if $\psi_n(e,y) = 0$. Then the summation

$$\phi(e) = \sum_{y \in \{0,1\}^{\alpha(n)}} R_n(e,y) \tag{17}$$

has a polynomial sized arithmetic circuit, and so there is a polynomial sized arithmetic circuit which agrees with $\phi(e) \prod_{i=1}^{n} x_i^{e_i}$ on binary vectors e, which completes the proof. □

The next result is arguably the most important in Valiant's theory.

Theorem 4. *The family PERM is **VNP**-complete.*

We first observe that we may perform the reduction in Theorem 1 just as well for PERM as we can for DET – in fact, the case for PERM is simpler since we need not worry about the sign of the permutation and hence we don't need to keep track of the parity of the path lengths in the construction. Therefore, if the p-definable family P is defined through the p-computable family Q via

$$P_n = \sum_{e \in \{0,1\}^{\alpha(n)}} Q_n(\cdot, e) \tag{18}$$

then we may easily construct graphs G_n so that the associated permanent of the graph (i.e., the permanent of the matrix corresponding to the edgeset of the graph) equals Q_n. It turns out it is possible to modify the graphs G_n to obtain new graphs G'_n so that summing over the cycle covers of G'_n corresponds to $2^{\alpha(n)}$ summations over the cycle covers of G_n, except with weights corresponding to the argument e attached to the variates in G_n that effectively "substitute" the appropriate value of e into the summation. The argument which shows this is rather technical, and there is no particular value in reproducing it here. The reader may refer to Section 2.2 of [1] for details.

5 Conclusion

Since Valiant's work, there have been numerous papers inspired by his ideas. For instance, more recent work in papers by Bürgisser [2], and Mahajan and Saurabh [6] have shown the existence of so-called **VNP**-intermediate polynomials – polynomials which are neither in **VP** nor are **VNP**-complete – in analogy with the classical result of Ladner [5] on the existence of **NP**-intermediate problems. What's more, [2] gives numerous examples of **VNP**-intermediate polynomials which correspond to natural problems, as opposed to simply showing their existence.

Perhaps the most inspired attempt to build on Valiant's (and others') work is in the development of *Geometric Complexity Theory* [7]. Proposed by researchers Ketan Mulmuley and Milind Sohoni, the Geometric Complexity Theory programme aims to leverage the powerful mathematical machinery of algebraic geometry and representation theory to address both **P** vs. **NP** and **VP** vs. **VNP** in a unified framework. The proposal has the distinction of being one of the few known approaches to **P** vs. **NP** which appears immune to the well-known relativization, natural proofs, and algebrization barriers. If correct, the authors expect the programme's major conjectures to take at least 100 years to resolve [3].

Although we were only able to give a brief introduction to the field of Algebraic Complexity Theory in this work, we've presented several key results and conjectures which demonstrate its importance. These results and conjectures are analogous to ones in classical complexity theory, and hence warrant attention for the same reasons researchers are rightly interested in problems like **P** vs. **NP**. Likewise, the efficiency of reductions to complete polynomial families like DET and PERM are also worth studying, as is done for complete problems in the classical case. For this reason, we expect that the preliminary results and the brief theoretical introduction we have presented here is only the beginning of much future work on this rich subject.

References

1. Bürgisser P., (2000) Completeness and Reduction in Algebraic Complexity Theory. Springer, Berlin Heidelberg
2. Bürgisser P., (1999) On the structure of Valiant's complexity classes. Discrete Mathematics & Theoretical Computer Science, pp. 73-94.
3. Fortnow, L., (2009), The Status of the P Versus NP Problem, Communications of the ACM, 52 (9): 7886, doi:10.1145/1562164.1562186
4. Hyafil L., On the parallel evaluation of multivariate polynomials. Proceedings of the Tenth ACM Symposium on the Theory of Computing (1978) pp. 193-195.
5. Ladner R., On the structure of polynomial time reducibility. J. ACM, 22(1):155171, 1975
6. Mahajan M., Saurabh N. (2016) Some Complete and Intermediate Polynomials in Algebraic Complexity Theory. In: Kulikov A., Woeginger G. (eds) Computer Science Theory and Applications. CSR 2016. Lecture Notes in Computer Science, vol 9691. Springer, Cham

7. K. Mulmuley, M. Sohoni. (2007) Geometric Complexity Theory: Introduction. arXiv. Retrieved from https://arxiv.org/pdf/0709.0746.pdf
8. Ostrowski A. (1954) On two problems in abstract algebra connected with Horner's rule. Studies in Mathematics and Mechanics. Academic Press. pp. 40-48
9. Pan Y. (1966) On means of calculating values of polynomials. Russian Math. Surveys. 21, pp. 105-136.
10. Shpilka, A., Yehudayoff, A., (2010) Arithmetic Circuits: A Survey of Recent Results and Open Questions, Foundations and Trends in Theoretical Computer Science, pp. 207 – 388, Now Publishers Inc., Hanover, MA, USA
11. Valiant L. G., (1979) Completeness Classes in Algebra, Proceedings of the Eleventh Annual CM Symposium on Theory of Computing, pp. 249 – 261, ACM, New York, NY, USA
12. Valiant L. G., (1992) Why is Boolean Complexity Theory Difficult?, Proceedings of the London Mathematical Society Symposium on Boolean Function Complexity, pp. 84 – 94, Cambridge University Press, New York, NY, USA