Algorithmica (2022) 84:436–463
https://doi.org/10.1007/s00453-021-00900-0

Fast Exact Algorithms Using Hadamard Product of Polynomials

V. Arvind[1] · Abhranil Chatterjee[1] · Rajit Datta[2] · Partha Mukhopadhyay[2]

Received: 19 October 2020 / Accepted: 18 November 2021 / Published online: 10 January 2022

Abstract

Let C be an arithmetic circuit of size s, given as input that computes a polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, where $\mathbb{F}$ is a finite field or the field of rationals. Using the Hadamard product of polynomials, we obtain new algorithms for the following two problems first studied by Koutis and Williams (Faster algebraic algorithms for path and packing problems, 2008, https://doi.org/10.1007/978-3-540-70575-8_47; ACM Trans Algorithms 12(3):31:1–31:18, 2016, https://doi.org/10.1145/2885499; Inf Process Lett 109(6):315–318, 2009, https://doi.org/10.1016/j.ipl.2008.11.004):

- $(k,n)-$MLC: is the problem of computing the sum of the coefficients of all degree-k multilinear monomials in the polynomial f. We obtain a deterministic algorithm of running time $\binom{n}{\downarrow k/2} \cdot n^{O(\log k)} \cdot s^{O(1)}$. This improvement over the $O(n^k)$ time brute-force search algorithm answers positively a question of Koutis and Williams (2016). As applications, we give exact counting algorithms, faster than brute-force search, for counting the number of copies of a tree of size k in a graph, and also the problem of exact counting of m-dimensional k-matchings.
- $k-$MMD: is the problem of checking if there is a degree-k multilinear monomial in the polynomial f with non-zero coefficient. We obtain a randomized algorithm of running time $O(4.32^k \cdot n^{O(1)})$. Additionally, our algorithm is polynomial space bounded.

An earlier version was presented at the FSTTCS 2019 conference [3] (arxiv version [2]).

✉ V. Arvind
arvind@imsc.res.in

Abhranil Chatterjee
abhranilc@imsc.res.in

Rajit Datta
rajit@cmi.ac.in

Partha Mukhopadhyay
partham@cmi.ac.in

1 Institute of Mathematical Sciences (HBNI), Chennai, India

2 Chennai Mathematical Institute, Chennai, India

Other results include fast deterministic algorithms for $(k,n)-$MLC and $k-$MMD problems for depth three circuits.

Keywords Parameterized complexity · Multilinear monomial detection · Multilinear monomial counting · Arithmetic circuits

1 Introduction

Let $\mathbb{F}$ be any field and $X = \{x_1, x_2, \ldots, x_n\}$ be a set of commuting variables. Let $\mathbb{F}[X]$ denote the *commutative polynomial ring* of multivariate polynomials over the field $\mathbb{F}$ in the variables X. That is, the elements of $\mathbb{F}[X]$ are $\mathbb{F}$-linear combinations of *monomials* in the variables X, where monomials are variable products of the form

$$m = x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n},$$

where the e_i are nonnegative integers. The *degree* of the monomial m, defined above, is $e_1 + e_2 + \cdots + e_n$. We denote by X_d the set of all degree-d monomials in the variables X. The *degree* of a polynomial f is the maximum degree of a nonzero monomial occurring in f.

We denote the coefficient of a monomial m in f by $[m]f$. Thus, if f is a polynomial of degree d we can write it as a sum

$$f = \sum_{m \in X_d} [m]f \cdot m,$$

A polynomial $f \in \mathbb{F}[X]$ is *homogeneous* if all its nonzero monomials have the same degree. A *linear form* is a homogeneous degree-1 polynomial. For a degree-d polynomial $f \in \mathbb{F}[X]$, its *homogeneous component* of degree $\ell \leq d$, denoted $H_\ell(f)$ is

$$H_\ell(f) = \sum_{m \in X_\ell} [m]f \cdot m.$$

Thus, we can write $f = \sum_{\ell=0}^{d} H_\ell(f)$.

Computationally, a polynomial f in $\mathbb{F}[X]$ can be given as input in different representations. When f is given explicitly as a list of all its nonzero monomials with coefficients, that is called the *sparse representation*. The sparse representation is usually inefficient because a degree d polynomial can have $\binom{n+d}{d}$ nonzero monomials. There are more compact representations of polynomials, defined below, well-studied in algebraic complexity theory (see e.g. the survey by Shpilka and Yehudayoff [33]), that are central to the present paper.

Definition 1.1 (*Arithmetic Circuit*) An *arithmetic circuit C* over $\mathbb{F}$ is a directed acyclic graph such that each indegree 0 node is labeled with an input variable from X or a scalar from $\mathbb{F}$. Each internal node is called a gate. It has indegree (called fan-in) 2,

and is labelled either $(+)$ or $(\times)$ (addition or multiplication gate). Each gate of the circuit computes a polynomial. Each input gate computes the variable/scalar labelling it. The polynomial computed at a $+$ (or $\times$) gate is the sum (respectively, product) of the polynomials computed at its inputs. The polynomial computed by C is the polynomial computed at the output gate of the circuit.

An arithmetic circuit C is said to be *homogeneous* if every gate in C computes a homogeneous polynomial.

Definition 1.2 (*Algebraic Branching Programs (ABP)*) [26,29] An *algebraic branching program* (ABP) is a directed acyclic graph with one in-degree-0 vertex called the *source*, and one out-degree-0 vertex called the *sink*. The vertex set of the graph is partitioned into layers $0, 1, \ldots, \ell$, with directed edges only between adjacent layers (i to $i+1$). The source and the sink are at layers zero and ℓ respectively. Each edge is labeled by a linear form over variables $x_1, x_2, \ldots, x_n$. Let $P = (e_1, e_2, \ldots, e_\ell)$ be a source-to-sink directed path and let L_i be the linear form labeling the edge e_i on the path. Then the polynomial computed by the ABP is defined as the sum of products

$$\sum_{P} \prod_{i=1}^{\ell} L_i, \tag{1}$$

where the sum is over all source-to-sink directed paths P. An ABP is *homogeneous* if all edge labels are homogeneous linear forms.

1.1 The Problems

Koutis and Williams [21,22,40] introduced and studied two natural algorithmic problems on arithmetic circuits:

1. Given as input an arithmetic circuit C computing a polynomial $f \in \mathbb{F}[X]$, the k-multilinear monomial counting problem, denoted $(k,n)-$MLC is to compute the sum of the coefficients of all degree-k multilinear monomials in the polynomial f.[1]
2. The k-multilinear monomial detection problem, denoted $k-$MMD, is to test if there is a degree-k multilinear monomial in the polynomial f with non-zero coefficient.

Both $(k,n)-$MLC and $k-$MMD can be solved in time $O^*(n^k)$ by a brute-force computation:[2] Suppose the input polynomial f is represented by an arithmetic circuit C of size s. We can first easily compute from C a homogeneous circuit C' of size $O(k^2 \cdot s)$ that represents the degree-k homogeneous component $H_k(f)$. This computation is done by keeping track of the different homogeneous components of C at every gate, discarding the homogeneous components of degree more than k [34] (or see the survey for details [33, Theorem 2.2]). Note that $H_k(f)$ has at most $\binom{n+k}{k}$ monomials. We can

[1] In this formulation, of course, the problem is a generalization of counting the degree-k multilinear monomials.

[2] The O^* notation suppresses factors polynomial in the input size.

compute the sparse representation of $H_k(f)$ in time $O^*(\binom{n+k}{k})$, again gate by gate, bottom up, in the circuit C'.

The above two problems have attracted significant attention in recent times. In particular, Koutis [21], Williams [40], and Koutis-Williams [22] have studied $(k,n)-$MLC and $k-$MMD problems from the viewpoint of parameterized and exact algorithms. These problems are natural generalizations of the well-studied k-path detection and counting problems in a given graph [21]. Moreover, some other combinatorial problems like k-Tree, m-Dimensional k-Matching [22], well-studied in the parameterized complexity, reduce to these problems. In fact, the first randomized FPT algorithms for the decision version of these combinatorial problems were obtained from an $O^*(2^k)$ algorithm for $k-$MMD for monotone circuits using an algebraic technique based on group algebras [21,22,40]. Recently, Brand et al. [9] have given the first randomized FPT algorithm for $k-$MMD for general circuits that runs in time $O^*(4.32^k)$. Their method is based on exterior algebra and color coding [1].

In general, the exact counting versions of the k-path problem and many related problems are #W[1]-hard with respect to parameter k. For these counting problems, improvements to the trivial $O^*(n^k)$ time exhaustive search algorithm are known only in some cases (like counting k-paths) [7]. In this connection, Koutis and Williams [22] ask if there is an algorithm for $(k,n)-$MLC that improves upon the naive $O^*(n^k)$ time algorithm. It would yield faster algorithms for several exact counting problems. Indeed, Koutis and Williams in [22] give an algorithm of running time $O^*(n^{k/2})$ to compute the *parity* of the sum of coefficients of degree-k multilinear monomials.

This Paper

In this paper, we make progress on the Koutis and Williams problem, mentioned above, by giving an $O^*(n^{k/2})$ algorithm for the $(k,n)-$MLC problem. Broadly, we develop a new approach to the $k-$MMD, $(k,n)-$MLC problems, and related problems. Our algorithms are based on computing the Hadamard product of polynomials. The Hadamard product (also known as Schur product) generally refers to Hadamard product of matrices and is widely used in matrix analysis. We consider the Hadamard product of polynomials (e.g., see [4]).

The Hadamard product of polynomials has turned out to be a useful tool in *noncommutative* computation [4,6]. A contribution of the present paper is to develop a new method for computing the Hadamard product in the *commutative* setting (as defined above), which turns out to be useful for designing efficient FPT and exact algorithms. An initial application of the Hadamard product of polynomials in arithmetic circuit complexity was in the context of proving hardness, e.g., showing hardness of the noncommutative determinant [6]. In contrast, the main application of Hadamard product in the present paper is in showing upper bound results. Broadly speaking, transferring techniques from circuit complexity to algorithm design is an important recent area of research. We refer the reader to the articles of Williams [39,41].

At this point, before giving an overview of our results, we give some formal basic definitions and set up the notation for the paper.

1.2 Basic Definitions and Notation

We begin with the definition of the Hadamard product.

Definition 1.3 The *Hadamard product* of polynomials f and g in $\mathbb{F}[X]$ is defined as

$$f \circ g = \sum_m ([m]f \cdot [m]g) \cdot m,$$

where m runs over all monomials nonzero in f or g, and $[m]f$ denotes the coefficient of the monomial m in f.

The underlying field $\mathbb{F}$ for the polynomials rings we study in this paper is either the field of rationals $\mathbb{Q}$ or a finite field $\mathbb{F}_{p^m}$, where p^m is a prime power. We briefly describe the complexity of field arithmetic in these fields. Details can be found in the textbook by von zur Gathen and Gerhard [37]. For $\mathbb{F} = \mathbb{Q}$, the field elements are given as a/b for integers $a, b, b \neq 0$, encoded in binary. The complexity of field arithmetic in $\mathbb{Q}$ is governed by the complexity of integer multiplication. By the well-known Schönhage-Strassen algorithm two n bit integers can be multiplied with $O(n \log n \log\log n)$ bit operations. Recently, this has been improved to $O(n \log n)$ by Harvey and van der Hoeven [19]. Consequently, field arithmetic in $\mathbb{Q}$ can be performed in with $O(n \log n)$ bit operations for rationals a/b, where a and $b \neq 0$ are n-bit integers.

For a finite field $\mathbb{F}_{p^m}$, the field is given by the prime p in binary, along with a univariate irreducible polynomial $p(z)$ of degree m which defines F_{p^m} as the quotient ring $\mathbb{F}_p[z]/(p(z))$. The elements of $\mathbb{F}_{p^m}$ are then representable as polynomials $q(z)$ of degree at most $m-1$ with coefficients from the prime field $\mathbb{F}_p$. In this representation each element of $\mathbb{F}_{p^m}$ requires $m \log p$ bits. Addition in $\mathbb{F}_{p^m}$ can be performed with $O(m \log p)$ bit operations. The complexity of multiplication is governed by the complexity of multiplying polynomials in $\mathbb{F}_p[z]$. There is a long line of work on fast polynomial multiplication [37], and the current best bound [18] has bit complexity $O(m \log p \log(m \log p) \cdot 4^{\log^*(m \log p)})$, where $\log^*(x)$ is the number of times that the natural log function must be iteratively applied to x to make the resulting value bounded by 1.

A polynomial $f \in \mathbb{F}[X]$ is said to be *multilinear* if for every nonzero monomial $m = x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$ of f we have $e_i \leq 1$.

An important family of polynomials for this paper are the *elementary symmetric polynomials* which are defined over any field $\mathbb{F}$ as follows:

The *elementary symmetric polynomial* $S_{n,k} \in \mathbb{F}[X]$ of degree k over the n variables $X = \{x_1, x_2, \ldots, x_n\}$ is defined as

$$S_{n,k} = \sum_{S \subset [n] : |S| = k} \prod_{i \in S} x_i.$$

By definition, $S_{n,k}$ is the sum of all the degree-k multilinear monomials.

Two other important families of multilinear polynomials relevant for this paper, again defined over all fields, are the determinant and permanent polynomials:

Let $\mathrm{X} = \{X_{ij}\}_{1\le i,j\le n}$ be an $n \times n$ matrix of commuting indeterminates. The n^{th} *permanent polynomial* is defined as:

$$\mathrm{Per}(\mathrm{X}) = \sum_{\sigma\in S_n}\prod_{i=1}^{n} X_{i,\sigma(i)}.$$

It is a homogeneous degree-n multilinear polynomial. The n^{th} *determinant polynomial* is defined as

$$\mathrm{Det}(\mathrm{X}) = \sum_{\sigma\in S_n}(-1)^{\mathrm{sgn}(\sigma)}\prod_{i=1}^{n} X_{i,\sigma(i)}.$$

1.2.1 Noncommutative Computation

Let $Y = \{y_1, y_2, \ldots, y_n\}$ be n *noncommuting variables*. Monomials over Y are essentially *words* over Y, treated as an alphabet, and the free monoid Y^* is the set of all monomials over Y. The degree of a monomial is just its length. For a field $\mathbb{F}$, we have the *free noncommutative ring* $\mathbb{F}\langle Y\rangle$, whose elements are finite $\mathbb{F}$-linear combinations of monomials, with addition defined coefficient-wise and multiplication inherited from monomial multiplication by distributivity. The degree of a noncommutative polynomial g is the maximum degree of a nonzero monomial occurring in it, and g is called homogeneous if all its monomials have the same degree.

Noncommutative Circuits and ABPs

- A *noncommutative arithmetic circuit* computing a polynomial in $\mathbb{F}\langle Y\rangle$ is defined like commutative arithmetic circuits (Definition 1.1). The only difference is that at each multiplication gate of the circuit the order of multiplication is important. This is taken care of by prescribing an order of the inputs at each multiplication gate.
- A *noncommutative ABP* is also defined like commutative ABPs. The crucial difference again is the order of multiplication: more precisely, in Equation 1 of Definition 1.2 the product $\prod_{i=0}^{\ell} L_i$ is left to right in increasing order of indices.

We note that when an input polynomial f is given by *black-box access*, in the commutative setting it means evaluating f at chosen scalar points $\bar{a} \in \mathbb{F}^n$, and in the noncommutative setting it means evaluating f at any matrix tuple $(M_1, M_2, \ldots, M_n)$ substituted for the variables.

An important ingredient of our main results in the following theorem [6] that allows us to compute in polynomial time the Hadamard product of a noncommutative ABP with a noncommutative polynomial f, even with only black-box access to f.

Theorem 1.4 [6, Corollary 4] *Given a noncommutative ABP of size s_1 computing a degree-d polynomial $g \in \mathbb{F}\langle Y\rangle$ and another degree-d polynomial $f \in \mathbb{F}\langle Y\rangle$ by an arithmetic circuit of size s_2 we can compute an arithmetic circuit of size $O(s_1^3 \cdot s_2)$ for $f \circ g$ in time polynomial in s_1, s_2 and d. Furthermore, if f is given by black-box access then we can evaluate $f \circ g$ at any matrix-valued input $(M_1, M_2, \ldots, M_n)$, where the M_i are $t \times t$ matrices over $\mathbb{F}$ in time polynomial in s_1, d and t.*

Further details on noncommutative computation can be found in Nisan's work [26] and the survey paper [33].

From Commutative to Noncommutative

A crucial element of our results on Hadamard product computation is going from the commutative to the noncommutative setting.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be n commuting variables and $Y = \{y_1, y_2, \ldots, y_n\}$ be n corresponding noncommuting variables.[3] Suppose $f \in \mathbb{F}[X]$ is a homogeneous degree-k polynomial represented by an arithmetic circuit C. We define its noncommutative version $\widehat{C}$ which computes a noncommutative homogeneous degree-k polynomial denoted $\hat{f} \in \mathbb{F}\langle Y \rangle$ as follows.

Definition 1.5 Let C be a commutative arithmetic homogeneous circuit C computing a homogeneous degree-k polynomial $f \in \mathbb{F}[X]$. The *noncommutative version* of C, $\widehat{C}$ is the noncommutative circuit obtained from C by fixing an ordering of the inputs to each product gate in C and replacing x_i by the noncommuting variable y_i, $1 \leq i \leq n$. The polynomial computed by $\widehat{C}$ is denoted $\hat{f} \in \mathbb{F}\langle y_1, y_2, \ldots, y_n \rangle$.

We similarly define the noncommutative version $\widehat{B}$ of an ABP B.

The *arithmetic circuit complexity* of a polynomial f is the size $s(f)$ of the smallest circuit representing f. The *ABP complexity* of f is the size $b(f)$ of the smallest ABP representing f. This notation is used for both commutative and noncommutative polynomials.

Remark 1.6 The definition of the noncommutative $\widehat{C}$ is entirely dependent on the ordering of the inputs to each product gate of C. This could, for instance, be by increasing order of the gate names of the circuit C. Since C is a homogeneous circuit, we note that the circuit $\widehat{C}$ is also a homogeneous circuit.

We introduce the following notation:

$$X_k \triangleq \{\text{all degree } k \text{ monomials over } X\}.$$
$$Y^k \triangleq \{\text{all degree } k \text{ monomials over } Y\}.$$

For mapping noncommutative polynomials back to commutative polynomials, we use the substitution map:

$$\nu : Y \to X \text{ defined as } \nu(x_i) = y_i, 1 \leq i \leq n.$$

This map extends to $\nu : Y^k \to X_k$ and, by linearity, gives a ring homomorphism (it is easily checked that $\nu(f + g) = \nu(f) + \nu(g)$ and $\nu(fg) = \nu(f)\nu(g)$):

$$\nu : \mathbb{F}\langle Y \rangle \to \mathbb{F}[X],$$

and its kernel, $\ker(\nu)$, is precisely all those noncommutative polynomials over Y that vanish if the variables are allowed to commute.

[3] Throughout, we use y_i to denote noncommuting variables associated with the x_i.

Each monomial $m \in X_k$ can appear as a different noncommutative monomial $\hat{m} \in \nu^{-1}(m)$ in $\hat{f} \in \mathbb{F}\langle Y \rangle$. We will use the notation $\hat{m} \to m$ to denote that $\hat{m} \in \nu^{-1}(m)$. Observe that

$$[m]f = \sum_{\hat{m} \in \nu^{-1}(m)} [\hat{m}]\hat{f} = \sum_{\hat{m}:\hat{m} \to m} [\hat{m}]\hat{f}.$$

The noncommutative circuit $\widehat{C}$ is not directly useful for computing Hadamard product. However, the following symmetrization helps. We first explain how permutations $\sigma \in S_k$ act on the set Y^k of degree-k monomials. The action extends, by linearity, to all homogeneous degree-k polynomials.

For each monomial $\hat{m} = y_{i_1} y_{i_2} \cdots y_{i_k}$, the permutation $\sigma \in S_k$ maps $\hat{m}$ to the monomial $\hat{m}^\sigma$ defined as

$$\hat{m}^\sigma = y_{i_{\sigma(1)}} y_{i_{\sigma(2)}} \cdots y_{i_{\sigma(k)}}.$$

By linearity, the polynomial $\hat{f}$ is mapped by σ to the polynomial

$$\hat{f}^\sigma = \sum_{\hat{m} \in Y^k} [\hat{m}]\hat{f} \cdot \hat{m}^\sigma.$$

Definition 1.7 (*Symmetrized polynomial*) The *symmetrized polynomial* $f^* \in \mathbb{F}\langle Y \rangle$ obtained from $f \in \mathbb{F}[X]$ is defined as the degree-k homogeneous polynomial

$$f^* = \sum_{\sigma \in S_k} \hat{f}^\sigma.$$

1.2.2 Polynomial Identity Testing

A fundamental algorithmic problem concerning arithmetic circuits and algebraic branching programs is *polynomial identity testing*: Given a polynomial $f \in \mathbb{F}[X]$ as input, check if f is identically zero (which means that there are no nonzero monomials in f).

The polynomial identity testing (or PIT) problem is a central problem in the field of randomized and algebraic computation and has received a lot of attention over the years (see the survey [31,33] for more details). When the field $\mathbb{F}$ is larger than the degree, there is a simple randomized test using the Demillo-Lipton-Schwartz-Zippel Lemma [15,32,42]. The PIT problem has been studied both in the commutative and noncommutative settings [33].

The input polynomial f can be represented by an arithmetic circuit or an ABP, or simply given black-box access.

1.2.3 Complexity Theory Background

The basic complexity classes of interest in this paper are P and NP, which are the classes of decision problems solvable in deterministic polynomial time and nondeterministic

polynomial time, respectively. Polynomial time as the notion of feasible computation, and the accompanying hardness theory of NP-completeness, is refined in the world of parameterized computation where the input instance is augmented with a fixed parameter k. Feasible parameterized computation means that the running time is of the form $t(k) \cdot n^{O(1)}$ for inputs of size n and fixed parameter k, where $t(\cdot)$ can be an arbitrary function that depends solely on parameter k. The parameterized analogue of P is denoted FPT. It is the class of *fixed parameter time* solvable problems, and algorithms with such running time are called FPT algorithms. The analogue of NP is denoted W[1], but the hardness theory has more technical details that can be found in the textbook by Downey and Fellows [16]. Cygan et al. [12] is a very good source for parameterized algorithms.

The notation $O^*(T(n,k))$ suppresses polynomial factors. Thus, a function in $O^*(T(n,k))$ is of the form $O(T(n,k) \cdot \text{poly}(n,k))$, where the notation $\text{poly}(n,n')$ is used as an alternative to $O(n^{O(1)}n'^{O(1)})$.

The following is a convenient notation for ascending sums of binomial coefficients:

$$\binom{n}{\downarrow i} \triangleq \sum_{j=0}^{i} \binom{n}{j}.$$

1.3 Overview of Results

We apply the Hadamard product of polynomials in the setting of commutative computation. This is achieved by combining earlier ideas [4,6] with a symmetrization trick described in Sect. 2. We then use it to design new algorithms for $(k,n)-$MLC, $k-$MMD, and related problems.

Firstly, computing the Hadamard product of the elementary symmetric polynomial $S_{n,k} \in \mathbb{F}[X]$ with a polynomial $f \in \mathbb{F}[X]$ *sieves out* precisely the degree-k multilinear component of f. This connection with the symmetric polynomial gives the following result.

Theorem 1.8 *For input polynomials $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ to the $(k,n)-$*MLC *problem, where f is represented by an algebraic branching program of size s, there is a deterministic $O^*(\binom{n}{\downarrow k/2})$-time algorithm. For input polynomials f represented by an arithmetic circuit C of size s, there is a deterministic $O^*(\binom{n}{\downarrow k/2} \cdot s^{c\cdot\log k})$-time algorithm, where c is a constant.*

In the above theorem, the underlying field $\mathbb{F}$ could be the rationals or any finite field. We note that the above running time beats the naive $O^*(n^k)$ bound, answering the question asked by Koutis and Williams [22].

An important ingredient of the proof is Theorem 1.4, which a result in [6].

The other ingredient is an algorithm of Björklund et al. [8] for evaluating the rectangular permanent over noncommutative rings, that can be viewed as an algorithm for evaluating $S^*_{n,k}$ (a symmetrized noncommutative version of $S_{n,k}$, defined in Sect. 2) over matrices.

This yields a deterministic $O^*(\binom{n}{\downarrow k/2})$-time algorithm when the input polynomial is given by an algebraic branching program (ABP).

When the input polynomial f is given by an arithmetic circuit, we can first obtain from it a circuit for the degree-k homogeneous component of f. Now, applying a standard transformation via depth-reduction we can transform this degree-k n-variate commutative arithmetic circuit of size, say s to an ABP of size $s^{O(\log k)}$ [33] which yields the claimed algorithm for $(k,n)-$MLC.

Theorem 1.9 *The $k-$MMD problem for input polynomials $f \in \mathbb{F}[X]$, represented as arithmetic circuits, has a randomized $O^*(4.32^k)$-time algorithm that is polynomial space-bounded, and $\mathbb{F}$ is either the rationals or a finite field.*

We briefly sketch the proof idea. Suppose that C is the input arithmetic circuit computing a homogeneous polynomial f of degree k. We essentially show that $k-$MMD is reducible to checking if the Hadamard product $f \circ C'$ is nonzero for some circuit C' from a collection $\mathcal{C}$ of homogeneous degree-k depth two circuits. This collection of depth two circuits is obtained by application of color coding [1].

Furthermore, the commutative Hadamard product $f \circ C'$ turns out to be computable in $O^*(2^k)$ time by a symmetrization trick combined with Ryser's formula for the permanent. The overall running time (because of trying several choices for C') turns out to be $O^*(4.32^k)$. Finally, checking if $f \circ C'$ is nonzero reduces to an instance of polynomial identity testing which can be solved in randomized polynomial time using Demillo-Lipton-Schwartz-Zippel Lemma [15,32,42].

Next, we state the results showing fast *deterministic* algorithms for depth three circuits. We use the notation $\Sigma^{[s']}\Pi^{[k]}\Sigma$ to denote depth-three circuits with the output gate as a $+$ gate of fan-in s', with the next layer of $\times$ gates of fan-in k, each computing the product of k homogeneous linear forms over X. The overall size of the circuit is then $s = O(s' \cdot k \cdot n)$.

Theorem 1.10 *Given any homogeneous depth three $\Sigma^{[s']}\Pi^{[k]}\Sigma$ circuit of degree k, the $(k,n)-$MLC problem can be solved in deterministic $O^*(2^k)$-time. Over $\mathbb{Z}$, the $k-$MMD problem can be solved in deterministic $O^*(4^k)$-time. Over finite fields, $k-$MMD problem can be solved in deterministic $e^k k^{O(\log k)} O^*(2^{ck} + 2^k)$ time, where $c \leq 5$.*

It is well-known that the elementary symmetric polynomial $S_{n,k}$ can be computed using an ABP of size $\text{poly}(n, k)$.

We then compute the Hadamard product of the given depth three circuit with that homogeneous ABP for $S_{n,k}$, and check whether the resulting depth three circuit is identically zero or not. The same idea yields the algorithm to compute the sum of the coefficients of the multilinear terms as well.

Related Work We briefly discuss here some related work (we have a more detailed discussion appears in Sect. 7).

Soon after the first version of our paper [2] appeared in ArXiv, an independent work by Pratt [27, v1] and [28] also considers the $k-$MMD and $(k,n)-$MLC problems. The main ingredient of [27] is the application of Waring decomposition over the rationals of symmetric polynomials [23] which does not have any known analogue over finite fields of small characteristic.

The algorithms obtained [27] for $k-$MMD and $(k,n)-$MLC over the rationals are faster ($O^*(4.08^k)$-time for $k-$MMD and $O^*(n^{k/2})$-time for $(k,n)-$MLC). In comparison, our algorithms work both over the rationals and finite fields. As already

mentioned, the algorithm of Koutis and Williams [22] for $(k,n)-$MLC works over $\mathbb{F}_2$ and the running time is $O^*(n^{k/2})$. In this sense, our algorithm for $(k,n)-$MLC can also be viewed as a generalization that is independent of the field's characteristic. It is to be noted that over fields of small characteristic a Waring decomposition of the input polynomial may not even exist. For example, over $\mathbb{F}_2$ the polynomial xy has no Waring decomposition. The more recent work of Brand and Pratt [10], extending Pratt's results [28] in explained in Sect. 7.

Our algorithm obtained in Theorem 1.9 for $k-$MMD appears quite different from the exterior algebra based algorithm [9], but it has the same running time of $O^*(4.32^k)$ because of the specific application of color coding due to Hüffner et al. [20]. However, we note that our algorithm requires only $\text{poly}(n,k)$ space, whereas the algorithm in [9] takes exponential space.

The field of parameterized approximate counting has seen several interesting directions in recent years. We refer the reader to [13,14,38] for results showing reductions from approximate counting to the decision problem.

Organization. The rest of this paper is organized as follows. In Sect. 2 we explain the Hadamard product framework. The proof of Theorem 1.8 and its consequences are given in Sect. 3. Section 5 contains the proof of Theorem 1.9. The proof of Theorem 1.10 is given in Sect. 6.

2 Hadamard Product Framework

Given two arithmetic circuits C_1 and C_2 computing polynomials f_1 and f_2, it is in general unlikely that $f_1 \circ f_2$ can be computed by an arithmetic circuit C of size $\text{poly}(|C_1|, |C_2|)$. This can be observed from the fact that for the symbolic matrix $\mathrm{X} = (x_{i,j})_{1\le i,j\le n}$, the Hadamard product of the determinant polynomial $\mathrm{Det}(\mathrm{X})$ with itself is the permanent polynomial $\mathrm{Per}(\mathrm{X})$ which does not have polynomial-size circuit assuming Valiant's VP $\neq$ VNP hypothesis [35]. However, it is well known that $\mathrm{Det}(\mathrm{X})$ can be computed by a polynomial-size ABP [24].

Nevertheless, we develop a method for computing the *scaled* Hadamard product of commutative polynomials in some special cases.

Definition 2.1 The *scaled* Hadamard product of polynomials $f, g \in \mathbb{F}[X]$ is defined as

$$f \circ^{\mathrm{S}} g = \sum_m (m! \cdot [m]f \cdot [m]g) \cdot m,$$

where for monomial $m = x_{i_1}^{e_1} x_{i_2}^{e_2} \dots x_{i_r}^{e_r}$ we define $m! = e_1! \cdot e_2! \cdots e_r!$.

Computing the scaled Hadamard product is key to our algorithmic results for $k-$MMD and $(k,n)-$MLC. Broadly, it works as follows: we transform polynomials f and g to suitable *noncommutative* polynomials. We compute their (noncommutative) Hadamard product (Theorem 1.4) [4,6], and then recover the scaled commutative Hadamard product $f \circ^{\mathrm{S}} g$ (or evaluate it at a desired point $\vec{a} \in \mathbb{F}^n$).

Suppose $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ is a homogeneous degree-k polynomial given by a circuit C. We have its noncommutative version $\widehat{C}$ which computes the noncommutative homogeneous degree-k polynomial $\hat{f} \in \mathbb{F}\langle y_1, y_2, \ldots, y_n\rangle$ (see Definition 1.5).

Recall from Sect. 1.2.1 that X_k denotes the set of all degree-k monomials over X, Y^k denote all degree-k noncommutative monomials over Y, and we have $[m]f = \sum_{\hat{m}\to m}[\hat{m}]\hat{f}$, where $m \in X_k$ and $\hat{m} \in Y^k$. We will use the *symmetrized polynomial* (see Definition 1.7), $f^* = \sum_{\sigma\in S_k} \hat{f}^\sigma$, to compute the scaled Hadamard product $f \circ^{\mathrm{S}} g$.

Lemma 2.2 *For a homogeneous degree-k commutative polynomial $f \in \mathbb{F}[X]$ given by circuit C, and its noncommutative version $\widehat{C}$ computing polynomial $\hat{f} \in \mathbb{F}\langle Y\rangle$, consider the* symmetrized *noncommutative polynomial $f^* = \sum_{\sigma\in S_k} \hat{f}^\sigma$. Then for each monomial $m \in X_k$ and each word $m' \in Y^k$ such that $m' \to m$, we have: $[m']f^* = m! \cdot [m]f$.*

Proof Let $f = \sum_m [m]f \cdot m$ and $\hat{f} = \sum_{\hat{m}} [\hat{m}]\hat{f} \cdot \hat{m}$. As already observed, $[m]f = \sum_{\hat{m}\to m}[\hat{m}]\hat{f}$. Now, we write $f^* = \sum_{m'}[m']f^* \cdot m'$. The group S_k acts on Y^k by permuting the positions. Suppose $m = x_{i_1}^{e_1} \cdots x_{i_q}^{e_q}$ is a type $\boldsymbol{e} = (e_1, \ldots, e_q)$ monomial over X_k and $m' \to m$. Then, by the well-known *Orbit-Stabilizer Theorem* [11, Chapter 1] the orbit $O_{m'}$ of m' under the action of S_k has size $\frac{k!}{m!}$. It follows that

$$[m']f^* = \sum_{\hat{m}\in O_{m'}} m! \cdot [\hat{m}]\hat{f} = m! \sum_{\hat{m}\to m} [\hat{m}]\hat{f} = m! \cdot [m]f.$$

It is important to note that for some $\hat{m} \in Y^k$ such that $\hat{m} \to m$, even if $[\hat{m}]\hat{f} = 0$ then also $[\hat{m}]f^* = m! \cdot [m]f$. □

Next, we apply Lemma 2.2 to compute scaled Hadamard product in the commutative setting via noncommutative Hadamard product.

Remark 2.3 We note that given a commutative circuit C computing f, the noncommutative polynomial $\hat{f}$ depends on the circuit structure of C. However, f^* depends only on the polynomial f.

Lemma 2.4 *Let $g \in \mathbb{F}[X]$ be a homogeneous degree-k polynomial and C be some arithmetic circuit for g. For any homogeneous degree-k polynomial $f \in \mathbb{F}[X]$ and any point $\vec{a} \in \mathbb{F}^n$*

$$(f \circ^S g)(\vec{a}) = (f^* \circ \hat{g})(\vec{a}),$$

where the noncommutative polynomial $\hat{g}$ is defined by the given circuit C.

Proof We write $f = \sum_m [m]f \cdot m$ and $g = \sum_{m'}[m']g \cdot m'$. By definition, we have $f \circ^{\mathrm{S}} g = \sum_m m! \cdot [m]f \cdot [m]g \cdot m$.

The polynomial computed by $\widehat{C}$ is $\hat{g}(Y) = \sum_{m\in X_k}\sum_{\hat{m}\to m}[\hat{m}]\hat{g}\cdot\hat{m}$. By Lemma 2.2, the noncommutative polynomial $f^*(Y) = \sum_{m\in X_k}\sum_{\hat{m}\to m} m! \cdot [m]f \cdot \hat{m}$. Therefore,

$$(f^* \circ \hat{g})(Y) = \sum_{m\in X_k}\sum_{\hat{m}\to m} m! \cdot [m]f \cdot [\hat{m}]\hat{g} \cdot \hat{m} = \sum_{m\in X_k} m! \cdot [m]f \cdot \left(\sum_{\hat{m}\to m}[\hat{m}]\hat{g}\right) \cdot \hat{m}.$$

Consequently, for any point $\vec{a} \in \mathbb{F}^n$ we have

$$(f^* \circ \hat{g})(\vec{a}) = \sum_{m \in X_k} m! \cdot [m]f \cdot \left(\sum_{\hat{m} \to m} [\hat{m}]\hat{g} \right) \cdot \hat{m}(\vec{a}).$$

Since $[m]g = \sum_{\hat{m} \to m} [\hat{m}]\hat{g}$, we have

$$(f^* \circ \hat{g})(\vec{a}) = \sum_{m \in X_k} m! \cdot [m]f \cdot m(\vec{a}) \cdot [m]g = (f \circ^{\mathrm{S}} g)(\vec{a}).$$

□

We note an immediate corollary of the above.

Corollary 2.5 *Let f_1, f_2 be homogeneous degree-k polynomials in $\mathbb{F}[X]$. Given a noncommutative circuit C computing the polynomial $\hat{f_1} \circ f_2^* \in \mathbb{F}\langle Y \rangle$, one can obtain a commutative circuit $\tilde{C}$ for $f_1 \circ^S f_2 \in \mathbb{F}[X]$ by replacing the noncommutative variables y_i in C by the commutative variables x_i.*

Proof Let $f_1 = \sum_m [m]f_1 \cdot m_1$. So, $\hat{f_1} = \sum_{\hat{m}} [\hat{m}]\hat{f} \cdot \hat{m}$ and $[m]f = \sum_{\hat{m} \to m} [\hat{m}]\hat{f}$. Then, $\hat{f_1} \circ f_2^*(Y) = \sum_{\hat{m}} [\hat{m}]\hat{f_1} \cdot [\hat{m}]f_2^* \cdot \hat{m} = \sum_{\hat{m}} [\hat{m}]\hat{f_1} \cdot m![m]f_2 \cdot \hat{m}$ where $\hat{m} \to m$. Now replacing the noncommutative variables by commutative variables, we obtain

$$\hat{f_1} \circ f_2^*(X) = \sum_m m! \cdot \left(\sum_{\hat{m} \to m} [\hat{m}]\hat{f_1} \right) \cdot [m]f_2 \cdot m$$

Since, $[m]f = \sum_{\hat{m} \to m} [\hat{m}]\hat{f}$, we further simplify and get $\hat{f_1} \circ f_2^*(X) = f_1 \circ^{\mathrm{S}} f_2(X)$. □

Remark 2.6 A key conceptual tool in [27] is the apolar inner product for homogeneous degree-k polynomials f and g in $\mathbb{F}[X]$, which is defined as

$$\langle f, g \rangle = f(\partial_{x_1}, \ldots, \partial_{x_n}) \circ g(x_1, \ldots, x_n).$$

We note that in the Hadamard product framework, we can express the apolar inner product of f and g as $f \circ^{\mathrm{S}} g$ evaluated at the all-ones vector $\vec{1} \in \mathbb{F}^n$. In Sect. 7 we present more details.

3 The Sum of Coefficients of Multilinear Monomials

In this section we prove Theorem 1.8. As already sketched in Sect. 1, the main idea is to apply the symmetrization trick to reduce the $(k,n)-$MLC problem to evaluating the rectangular permanent over a suitable matrix ring. Then we use a result of [8] to solve the instance of the rectangular permanent evaluation problem.

Before we present the results we recall the definition of ABPs (Definition 1.2) and the following different but equivalent formulation of ABPs:

A homogeneous ABP of width w computing a degree-k polynomial over X can be thought of as the $(1, w)^{th}$ entry of the product of $w \times w$ matrices $M_1 \cdots M_k$ where entries of each M_i are homogeneous linear forms over X. By $[x_j]M_i$, we denote the $w \times w$ matrix over $\mathbb{F}$, such that $(p, q)^{th}$ entry of the matrix,$([x_j]M_i)(p, q) = [x_j](M_i(p, q))$, the coefficient of x_j in the linear form of the $(p, q)^{th}$ entry of M_i.

Permanent of Rectangular Matrices

We now define the permanent of a rectangular matrix. The permanent of a rectangular $k \times n$ matrix $A = (a_{ij})$, $k \leq n$, with entries over a ring R is defined as

$$\mathrm{rPer}(A) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^{k} a_{i,\sigma(i)},$$

where $I_{k,n}$ is the set of all injections from $[k]$ to $[n]$. We define the noncommutative polynomial $S^*_{n,k}$ as

$$S^*_{n,k}(y_1, y_2, \ldots, y_n) = \sum_{T \subseteq [n]:|T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} y_{\sigma(i)},$$

which is the symmetrized version of the elementary symmetric polynomial $S_{n,k}$ as defined in Lemma 2.2. Given a set of $t \times t$ matrices $M_1, \ldots, M_n$ over some field $\mathbb{F}$ define the rectangular (block) matrix $A = (a_{i,j})_{i \in [k], j \in [n]}$ such that $a_{i,j} = M_j$. Thus, A is a $k \times n$ matrix with entries from the ring of $t \times t$ matrices over the field $\mathbb{F}$. The following observation is crucial.

Observation 3.1 $S^*_{n,k}(M_1, \ldots, M_n) = \mathrm{rPer}(A)$.

Proof To see this, observe that ,

$$\mathrm{rPer}(A) = \sum_{T \subseteq [n]:|T|=k} \mathrm{Per}(A_T) = \sum_{T \subseteq [n]:|T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} M_{\sigma(i)}.$$

Here A_T is the minor of A such that the columns are indexed by the set T. □

In the sequel, we will apply a result from [8], showing that over *any* ring R, the permanent of a rectangular $k \times n$ matrix can be evaluated with $O^*(\binom{n}{\downarrow k/2})$ ring operations. In particular, if R is the matrix ring $\mathbb{M}_s(\mathbb{F})$, the algorithm runs in time $O(k\binom{n}{\downarrow k/2}\,\mathrm{poly}(n, s))$. We now present the proof of Theorem 1.8.

Proof We first prove a special case of the theorem when the polynomial f is given by a homogeneous degree-k ABP B of width w. Notice that we can compute the sum of the coefficients of the degree-k multilinear terms in f by evaluating $(f \circ S_{n,k})(\vec{1})$. Now to compute the Hadamard product efficiently, we will transfer the problem to the

noncommutative domain. Let $\widehat{B}$ define the noncommutative version of the commutative ABP B for the polynomial f. By Lemma 2.4, it suffices to compute $(\widehat{B} \circ S^*_{n,k})(\vec{1})$. Now, the following lemma reduces this computation to evaluating $S^*_{n,k}$ over a suitable matrix ring. We recall the following result from [5] (see, also [6]). □

Lemma 3.2 (Theorem 2 of [5]) *Let f be a homogeneous degree-k noncommutative polynomial in $\mathbb{F}\langle Y\rangle$ and B be an ABP of width w computing a homogeneous degree-k polynomial $g = (M_1 \cdots M_k)(1, w)$ in $\mathbb{F}\langle Y\rangle$.*

Then $(f \circ g)(\vec{1}) = (f(A^B_1, \ldots, A^B_n))(1, (k+1)w)$ where for each $i \in [n]$, A^B_i is the following $(k+1)w \times (k+1)w$ block superdiagonal matrix,

$$A^B_i = \begin{bmatrix} 0 & [y_i]M_1 & 0 & \ldots & 0 \\ 0 & 0 & [y_i]M_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & [y_i]M_k \\ 0 & 0 & 0 & \ldots & 0 \end{bmatrix}.$$

Proof We sketch the proof as we will require some details for the proof of Theorem 1.8. For any monomial $\hat{m} = y_{i_1} y_{i_2} \cdots y_{i_k} \in Y^k$,

$$(A^B_{i_1} A^B_{i_2} \cdots A^B_{i_k})(1, (k+1)w) = ([y_{i_1}]M_1 \cdot [y_{i_2}]M_2 \cdots [y_{i_k}]M_k)(1, w) = [\hat{m}]g,$$

from the definition. Hence, we have,

$$\begin{aligned} f(A^B_1, A^B_2, \ldots, A^B_n)(1, (k+1)w) &= \sum_{\hat{m} \in Y^k} [\hat{m}]f \cdot \hat{m}(A^B_{i_1}, A^B_{i_2}, \ldots, A^B_{i_k})(1, (k+1)w) \\ &= \sum_{\hat{m} \in Y^k} [\hat{m}]f \cdot [\hat{m}]g. \end{aligned}$$

□

Now, we construct a $k \times n$ rectangular matrix $A = (a_{i,j})_{i \in [k], j \in [n]}$ from the ABP $\widehat{B}$ (obtained from the given ABP B) by setting $a_{i,j} = A^{\widehat{B}}_j$ as defined. Using Observation 3.1, we now have,

$$\mathrm{rPer}(A)(1, (k+1)w) = S^*_{n,k}(A^{\widehat{B}}_1, \ldots, A^{\widehat{B}}_n)(1, (k+1)w).$$

Now by Lemmas 2.4 and 3.2, we conclude that,

$$S^*_{n,k}(A^{\widehat{B}}_1, \ldots, A^{\widehat{B}}_n)(1, (k+1)w). = (S^*_{n,k} \circ \widehat{B})(\vec{1}) = (S_{n,k} \circ^{\mathrm{S}} B)(\vec{1}).$$

Hence applying the algorithm of Björklund et al. for evaluating the rectangular permanent over noncommutative ring [8], we compute the sum of the coefficients deterministically in time $O(k\binom{n}{\downarrow k/2}\,\mathrm{poly}(n, k, w))$.

Now, we prove the general case. We apply a standard transformation from circuits to ABPs [33,36] and reduce the problem to the ABP case. More precisely, given an

arithmetic circuit of size s computing a polynomial f of degree k, f can also be computed by a homogeneous ABP of size $s^{O(\log k)}$. In particular, if f is given by an arithmetic circuit of size s, we first get a circuit of $\text{poly}(k, s)$ size for degree-k part of f using a standard method of homogenization [33, Theorem 2.2]. Then, we convert the homogeneous circuit to a homogeneous ABP of size $s^{O(\log k)}$. The width w of the new ABP is also bounded by $s^{O(\log k)}$. Next, we apply the first part of the proof on the newly constructed ABP. Notice that the entire computation can be done in deterministic $O(k\binom{n}{\downarrow k/2}\text{poly}(n, k, w))$ time which is $O^*(\binom{n}{\downarrow k/2} \cdot s^{c\cdot\log k})$ for some constant c. □

4 Some Applications

We show some applications of Theorem 1.8 and the technique developed there. The first two are algorithmic applications and the last is a hardness result.

Applying Theorem 1.8, we improve the counting complexity of two combinatorial problems studied in [22]. To the best of our knowledge, nothing better than the brute-force exhaustive search algorithms were known for the counting version of these problems. We start with the k-Tree problem: Let T be a tree with k vertices and G be an n-vertex graph. A *homomorphic embedding* of T in G is defined by an injective map $\varphi : V(T) \to V(G)$ such that for all $u, v \in V(T)$

$$uv \in E(T) \implies \varphi(u)\varphi(v) \in E(G).$$

A *copy* of T in G is $\varphi(T)$ for some homomorphic embedding φ of T in G. Let $e_{T,G}$ denote the number of homomorphic embeddings of T in G and $c_{T,G}$ denote the number of copies of T in G. Let $\text{Aut}(T)$ denote the automorphism group of T. For two such homomorphic embeddings φ_1 and φ_2 we have $\varphi_1(T) = \varphi_2(T)$ if and only if $\varphi_1^{-1}\varphi_2 \in \text{Aut}(T)$. Hence,

$$c_{T,G} = \frac{e_{T,G}}{|\,\text{Aut}(T)|}. \tag{2}$$

The exact counting version of k-Tree is the problem of counting the number of copies of T in G for an input pair (T, G). Clearly, there is a trivial $O^*(n^k)$ exhaustive search algorithm for the problem. We apply Theorem 1.8 to obtain an essentially quadratic speed-up.

Corollary 4.1 *The exact counting* k-Tree *problem of counting the number of copies of a given* k*-vertex tree in a given* n*-vertex graph can be computed in deterministic* $O^*(\binom{n}{\downarrow k/2})$ *time.*

Proof By Eq. 2 it suffices to count the number of homomorphic embeddings of T in G, as counting the number of automorphisms $|\,\text{Aut}(T)|$ of T can be done in $\text{poly}(k)$ time.

To this end, we will use a modification of the construction defined in [22, Theorem 2.2]. Let the nodes of T be $\{1, 2, \ldots, k\}$ and the nodes of G be $\{1, 2, \ldots, n\}$. W.l.o.g.,

we can consider T to be a tree rooted at 1. As a consequence, every node $i \in [k]$ of T uniquely defines a subtree T_i rooted at i. Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of n commuting variables corresponding to the n vertices of G. We inductively define a polynomial C_{ij} in $\mathbb{F}[X]$ as follows:

- If i is a leaf node of the tree T then $C_{ij} = x_j$.
- Otherwise, let $i_1, i_2, \ldots, i_\ell$ be the children of i in T. Inductively, we can assume that the polynomials $C_{i_t,j}$, $1 \le t \le \ell$, $1 \le j \le n$ are already defined. We define

$$C_{ij} = x_j \prod_{t=1}^{\ell} \left(\sum_{j':(j,j')\in E(G)} C_{i_t,j'} \right).$$

Finally, we define the polynomial Q as

$$Q(X) = \sum_{j=1}^{n} C_{1j}.$$

By definition, it follows that C_{ij} is a homogeneous polynomial of degree $|V(T_i)|$ for each $i \in [k]$. Consequently, $Q(X)$ is a homogeneous degree-k polynomial. □

Claim 4.2 *Let $i \in [k]$ and the subtree T_i have r nodes. Then the number of degree-r multilinear monomials in C_{ij} is exactly the number of homomorphic embeddings of T_i in G that maps i to j. Hence, the number of degree-r multilinear monomials in $\sum_{j\in V(G)} C_{ij}$ is the number of homomorphic embeddings of T_i in G.*

The above claim is easily proved by induction on the size of T_i. It is clearly true for $|V(T_i)| = 1$. In general, suppose $i_1, i_2, \ldots, i_\ell$ are the children of i in tree T. Any homomorphic embedding $\varphi : T_i \to G$ that maps i to j is defined uniquely by homomorphic embeddings $\varphi_t : T_{i_t} \to G$ such that the ranges of the φ_t are all disjoint and the i_t map to distinct G-neighbors of j. Clearly, from the definition of C_{ij} and induction, it follows that there is a unique multilinear monomial of C_{ij} that corresponds to φ. Conversely, each multilinear monomial defines a unique homomorphic embedding $\varphi : T_i \to G$ that maps i to j.

Thus, the exact counting k-Tree problem is solved by counting the number of multilinear monomials in Q. By Theorem 1.8, it suffices to construct for Q an ABP of size $\mathrm{poly}(n, k)$.

From the recursively defined structure of the noncommutative formula for C_{ij} we can analyze the size. Let size_ℓ bound the noncommutative formula size for the polynomial Q_ℓ defined as above for trees of size ℓ (note that $Q_k = Q$). We note from the formula structure that $\mathrm{size}_1 = n$ and

$$\mathrm{size}_k = nk + \sum_{t=1}^{\ell} \mathrm{size}_{k_t},$$

where k_t are the subtree sizes and $k_1 + k_2 + \cdots + k_\ell = k - 1$. Clearly, $\text{size}_k \leq nk^2$. Thus, Q has a formulas of size at most nk^2 which can be converted to an ABP of size $O(nk^2)$ by standard techniques [33]. □

We now consider the exact counting version of the m-Dimensional k-Matching problem: Let $U_1, U_2, \ldots, U_m$ be mutually disjoint sets, and let C be a collection of m-tuples from their cartesian product $U_1 \times \cdots \times U_m$. An m-dimensional k-matching in C is a subcollection of k m-tuples such that no two of these m-tuples share a common coordinate. Koutis and Williams [22] obtain a faster parameterized algorithm for the decision version of this problem. We present an exact counting algorithm as an application of Theorem 1.8.

Corollary 4.3 *Given mutually disjoint sets U_i, $i \in [m]$, and a collection C of m-tuples from $U_1 \times \cdots \times U_m$, we can count the number of m-dimensional k-matchings in C in deterministic $O^*(\binom{n}{\downarrow (m-1)k/2})$ time.*

Proof Following [22], encode each element u in $U = \cup_{i=2}^{m} U_i$ by a variable $x_u \in X$. Encode each m-tuple $t = (u_1, \ldots, u_m) \in C \subseteq U_1 \times \cdots \times U_m$ by the monomial $M_t = \prod_{i=2}^{m} x_{u_i}$. Assume $U_1 = \{u_{1,1}, \ldots, u_{1,n}\}$, and let $T_j \subseteq C$ denote the subset of m-tuples whose first coordinate is $u_{1,j}$. Consider the polynomial,

$$P(X, z) = \prod_{j=1}^{n} \left(1 + \sum_{t \in T_j} (z \cdot M_t)\right)$$

Clearly, $P(X, z)$ has an ABP of size $\text{poly}(n, m, |C|)$. Let $Q(X) = [z^k]P(X, z)$. In polynomial time we can obtain an ABP of size $\text{poly}(n, m, |C|)$ for $Q(X)$ by standard method of Vandermonde matrix based interpolation on the variable z. Clearly, $Q(X)$ is a homogeneous degree-km polynomial and the nonzero multilinear monomials of $Q(X)$ are in $1 - 1$ correspondence with the m-dimensional k-matchings in C. Therefore, the number of multilinear terms in $Q(X)$ is the required count. We can now apply the first part of Theorem 1.8 to count the number of multilinear terms in $Q(X)$.

□

Hardness of the Rectangular Permanent Over General Rings

In [8], it is shown that the $k \times n$ rectangular permanent can be evaluated over commutative rings and commutative semirings in $O(h(k) \cdot \text{poly}(n, k))$ time for some computable function h. In other words, the problem is in FPT, parameterized by the number of rows. An interesting question is to ask whether one can get any FPT algorithm when the entries are from noncommutative rings (in particular, matrix rings). We prove that such an algorithm is unlikely to exist. We show that counting the number of k-paths in a graph G, a well-known #W[1]-complete problem, reduces to this problem. So, unless ETH fails we do not have such an algorithm [16].

Theorem 4.4 *Given a $k \times n$ matrix X with entries $x_{ij} \in \mathbb{M}_{t\times t}(\mathbb{Q})$, computing the rectangular permanent of X is* #W[1]*-hard with k as the parameter where $t = (k+1)n$ under polynomial-time many-one reduction.*

Proof If we have an algorithm to compute the permanent of a $k \times n$ matrix over noncommutative rings which is FPT in parameter k, that yields an algorithm which is FPT in k for evaluating the polynomial $S^*_{n,k}$ on matrix inputs. This follows from Observation 3.1. Now, given a graph G we can compute a homogeneous ABP of width n and k layers for the graph polynomial C_G defined as follows.

Let $G(V, E)$ be a directed graph with n vertices where $V(G) = \{v_1, v_2, \ldots, v_n\}$. A k-walk is a sequence of k vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ where $(v_{i_j}, v_{i_{j+1}}) \in E$ for each $1 \leq j \leq k-1$. A k-path is a k-walk where no vertex is repeated. Let A be the adjacency matrix of G, and let $y_1, y_2, \ldots, y_n$ be noncommuting variables. Define an $n \times n$ matrix B

$$B[i, j] = A[i, j] \cdot y_i, \quad 1 \leq i, j \leq n.$$

Let $\vec{1}$ denote the all 1's vector of length n. Let $\vec{y}$ be the length n vector defined by $\vec{y}[i] = y_i$. The *graph polynomial* $C_G \in \mathbb{F}\langle Y\rangle$ is defined as

$$C_G(Y) = \vec{1}^T \cdot B^{k-1} \cdot \vec{y}.$$

Let W be the set of all k-walks in G. The following observation is folklore. □

Observation 4.5

$$C_G(Y) = \sum_{v_{i_1} v_{i_2} \ldots v_{i_k} \in W} y_{i_1} y_{i_2} \cdots y_{i_k}.$$

Hence, G contains a k-path if and only if the graph polynomial C_G contains a multilinear term.

Clearly the number of k-paths in G is equal to $(C_G \circ S_{n,k})(\vec{1})$. By Lemma 2.4, we know that it suffices to compute $(\widehat{C}_G \circ S^*_{n,k})(\vec{1})$. We construct $kn \times kn$ matrices $A_1, \ldots, A_n$ from the ABP of $\widehat{C}_G$ following Lemma 3.2. Then from Lemma 3.2, we know that $(\widehat{C}_G \circ S^*_{n,k})(\vec{1}) = S^*_{n,k}(A_1, \ldots, A_n)(1, t)$ where $t = (k+1)n$. So if we have an algorithm which is FPT in k for evaluating $S^*_{n,k}$ over matrix inputs, we also get an algorithm to count the number of k-paths in G in FPT(k) time. □

5 Multilinear Monomial Detection

In this section, we prove Theorem 1.9. First, we give an algorithm for computing the Hadamard product for a special case in the commutative setting. Any depth two $\Pi^{[k]}\Sigma$ circuit computes the product of k homogeneous linear forms over the input set of variables X.

Lemma 5.1 *Given an arithmetic circuit C of size s computing $g \in \mathbb{F}[X]$, and a homogeneous $\Pi^{[k]}\Sigma$ circuit computing $f \in \mathbb{F}[X]$, and any point $\vec{a} \in \mathbb{F}^n$, we can evaluate $(f \circ^S g)(\vec{a})$ in $O^*(2^k)$ time and in polynomial space.*

Proof By standard homogenization technique [33, Theorem 2.2] we can extract the homogeneous degree-k component of C and thus we can assume that C computes a homogeneous degree-k polynomial. Write $f = \prod_{j=1}^{k} L_j$, for homogeneous linear forms L_j. The corresponding noncommutative polynomial $\hat{f}$ is defined by the natural order of the j indices (and replacing x_i by y_i for each i). □

Claim 5.2 *The noncommutative polynomial f^* has a (noncommutative) $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ circuit, which we can write as $f^* = \sum_{i=1}^{2^k} C_i$, where each C_i is a (noncommutative) $\Pi^{[k]}\Sigma$ circuit.*

Before we prove the claim, we show that it easily yields the desired algorithm. First we notice that

$$\widehat{C} \circ f^* = \sum_{i=1}^{2^k} \widehat{C} \circ C_i.$$

Now, by Theorem 1.4, we can compute in $\text{poly}(n, s, k)$ time a $\text{poly}(n, s, k)$ size circuit for the (noncommutative) Hadamard product $\widehat{C} \circ C_i$. As argued in the proof of Lemma 2.4, for any $\vec{a} \in \mathbb{F}^n$ we have

$$(g \circ^S f)(\vec{a}) = (C \circ^S f)(\vec{a}) = (\widehat{C} \circ f^*)(\vec{a}).$$

Thus, we can evaluate $(g \circ^S f)(\vec{a})$ by incrementally computing $(\widehat{C} \circ C_i)(\vec{a})$ and adding up for $1 \leq i \leq 2^k$. This can be clearly implemented using only polynomial space. □

Proof of Claim 5.2 By definition,

$$f^* = \sum_{\sigma \in S_k} \hat{L}_{\sigma(1)} \hat{L}_{\sigma(2)} \cdots \hat{L}_{\sigma(k)}.$$

Now define the $k \times k$ matrix T such that the elements in each row of T are the linear forms $\hat{L}_1, \hat{L}_2, \ldots, \hat{L}_k$ in this order. Then the (noncommutative) permanent of T is given by

$$\text{Perm}(T) = \sum_{\sigma \in S_k} \prod_{j=1}^{k} \hat{L}_{\sigma(j)}$$

which is just f^*.

We will now apply Ryser's formula [30] to express $\text{Perm}(T)$ as a depth-3 homogeneous noncommutative $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ formula. We recall Ryser's formula [30] for

$\mathrm{Per}(A)$, where A is a $k \times k$ matrix with noncommuting entries A_{ij}:

$$\mathrm{Per}(A) = \sum_{S \subseteq [k]} (-1)^{|S|} \prod_{i=1}^{k} \sum_{j \in S} A_{ij}.$$

It is a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ formula for the $k \times k$ noncommutative permanent. Now, substituting $\hat{L}_j$ for A_{ij} $1 \leq i, j \leq k$, it follows that $f^* = \mathrm{Per}(T)$ has a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ noncommutative formula. Note that Ryser's formula is usually given for the commutative permanent. It is easy to observe that the same proof, based on the principle of inclusion-exclusion, also holds for the noncommutative permanent. □

Remark 5.3 Over the rationals, we can get an alternative proof of Lemma 5.1 by using Fischer's identity [17] to obtain a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ formula for f^*.

Now we are ready to prove Theorem 1.9.

Proof By homogenization, we can assume that C computes a homogeneous degree k polynomial f.

We will refer to maps $\zeta : [n] \to [k]$ as *coloring maps*. The map ζ can be seen as assigning colors[4] to the elements of $[n]$: $\zeta(i)$ is the *color* of i.

We will pick a collection of coloring maps $\{\zeta_i : [n] \to [k]\}$ each picked independently and uniformly at random. For each coloring map ζ_i we define a $\Pi^{[k]}\Sigma$ formula

$$P_i = \prod_{j=1}^{k} \sum_{\ell : \zeta_i(\ell) = j} x_\ell.$$

A monomial is *covered* by a coloring map ζ_i if the monomial is nonzero in P_i. The probability that a random coloring map covers a given degree-k multilinear monomial is

$$k! \cdot k^{-k} \approx e^{-k}.$$

Hence, for a collection $\mathcal{C}$ of $O^*(e^k)$ many coloring maps $\mathcal{C} = \{\zeta_i : [n] \to [k]\}$ picked independently and uniformly at random, it holds with constant probability that every multilinear monomial of degree k is covered at least once by some ζ_i in $\mathcal{C}$. This probability bound is by a simple and standard union bound argument. Now, for each coloring map $\zeta_i \in \mathcal{C}$ we consider the circuit $C_i' = C \circ^{\mathrm{S}} P_i$.

Notice that for each multilinear monomial m, the multiplicative factor $m!$ is 1. Also, the coefficient of each monomial is exactly 1 in each P_i, and if f contains a multilinear term then it is covered by *some* P_i. Now, we perform the randomized polynomial identity test on each circuit C_i' by applying the Demillo-Lipton-Schwartz-Zippel Lemma [15,32,42] in randomized polynomial time to complete the procedure.

[4] This terminology is in keeping with a seminal paper's in the field [1] which introduced color coding. However, it should be clear that this notion of coloring has nothing to do with graph colorings.

More precisely, we pick a random $\vec{a} \in \mathbb{F}^n$ and evaluate C_i' at $\vec{a}$ to check if it is nonzero. By Lemma 5.1, the computation of $C_i'(\vec{a})$ can be done deterministically in time $O^*(2^k)$ time and poly(n, k) space.[5] Hence the total running time of the procedure is $O^*((2e)^k)$.

In order to improve the running time to $O^*(4.32^k)$, we apply the color coding technique of Hüffner et al. [20]. The idea is to use more than k colors to reduce the number of coloring maps required to cover the degree-k monomials. But this would increase the formal degree of each depth two circuit which we need to handle.

We will use $1.3k$ colors[6] and each circuit P_i will now be a $\Pi^{[1.3k]}\Sigma$ circuit. For each coloring map $\zeta_i : [n] \to [1.3k]$ chosen uniformly at random, we define the following $\Pi^{[1.3k]}\Sigma$ circuit

$$P_i(x_1, x_2, \ldots, x_n, z_1, \ldots, z_{1.3k}) = \prod_{j=1}^{1.3k} \left(\sum_{\ell:\zeta_i(\ell)=j} x_\ell + z_j \right).$$

Since each P_i is of degree $1.3k$, we need to modify the circuit C to another circuit C' of degree $1.3k$ in order to apply Hadamard products. To that end, we define the circuit $C' \in \mathbb{F}[X, Z]$ as follows

$$C'(X, Z) = C(X) \cdot S_{1.3k,0.3k}(z_1, \ldots, z_{1.3k}),$$

where $S_{1.3k,0.3k}(z_1, \ldots, z_{1.3k})$ is the elementary symmetric polynomial of degree $0.3k$ over the variables $z_1, \ldots, z_{1.3k}$. By the result of [20], for $O^*(1.752^k)$ random coloring maps, with high probability each multilinear monomial in C is covered by the monomials of some P_i (over the X variables).

Now to compute $\widehat{C'} \circ P_i^*$ for each i, we symmetrize the polynomial P_i. Of course, the symmetrization happens over the X variables as well as over the Z variables. But in $\widehat{C'}$ we are only interested in the monomials (or words) where the rightmost $0.3k$ variables are over Z variables. In the noncommutative circuit $\widehat{C'}$, every subword $z_{i_1} z_{i_2} \ldots z_{i_{0.3k}}$ receives a natural ordering $i_1 < i_2 < \ldots < i_{0.3k}$.

Notice that

$$P_i^*(Y, Z) = \sum_{\sigma \in S_{1.3k}} \prod_{j=1}^{1.3k} \left(\sum_{\ell:\zeta_i(\ell)=\sigma(j)} y_\ell + z_{\sigma(j)} \right).$$

Our goal is to understand the part of $P_i^*(Y, Z)$ where each monomial ends with a subword of the form $z_{i_1} z_{i_2} \ldots z_{i_{0.3k}}$ and the top k symbols are over the X variables. For a fixed set of indices $W = \{i_1 < i_2 < \ldots < i_{0.3k}\}$, define the set $T = [1.3k] \setminus W$. Let $S_{[k],T}$ be the set of permutations $\sigma \in S_{1.3k}$ such that $\sigma : [k] \to T$ and $\sigma(k+j) = i_j$ for

[5] Since the syntactic degree of the circuit is not bounded here, and if we have to account for the bit level complexity (over $\mathbb{Z}$) of the scalars generated in the intermediate stage we may get field elements whose bit level complexity is exponential in the input size. So, a standard technique is to take a random prime of polynomial bit-size and evaluate the circuit modulo that prime.

[6] By $1.3k$ and $0.3k$, we mean the integers $\lceil 1.3k \rceil$ and $\lceil 0.3k \rceil$, respectively.

$1 \leq j \leq 0.3k$. As we have fixed the last $0.3k$ positions, each $\sigma \in S_{[k],T}$ corresponds to some $\sigma' \in S_k$. Let $Z_W = z_{i_1} z_{i_2} \ldots z_{i_{0.3k}}$. Now we notice the following. □

Observation 5.4 *The part of $P_i^*(Y, Z)$ where each monomial ends with the subword Z_W is $P_{i,W}^* \cdot Z_W$, where*

$$P_{i,W}^*(Y) = \sum_{\sigma \in S_{[k],T}} \prod_{j=1}^{k} \left(\sum_{\ell : \zeta_i(\ell) = \sigma(j)} y_\ell \right) = \sum_{\sigma' \in S_k} \prod_{j=1}^{k} \left(\sum_{\ell : \zeta_i(\ell) = \sigma'(j)} y_\ell \right).$$

Now, just like the case above, it suffices to perform polynomial identity testing for

$$(\widehat{C'} \circ P_i^*)(Y, Z) = \sum_{W \subseteq [1.3k] : |W| = 0.3k} \left(\widehat{C}(Y) \circ P_{i,W}^*(Y) \right) \cdot Z_W.$$

for each i. But this is same as testing $C' \circ^{\mathrm{S}} P_i$ for identity. Now we eliminate the Z variables by substituting 1 and evaluate X variables on a random point $\vec{a} \in \mathbb{F}^n$. By Lemma 5.1, $(C' \circ^{\mathrm{S}} P_i)(\vec{a}, \vec{1})$ can be computed in $O^*(2^{1.3k}) = O^*(2.46^k)$ time and $\mathrm{poly}(n, k)$ space. The bound on the success probability follows from Demillo-Lipton-Schwartz-Zippel Lemma [15,32,42].

We repeat the above procedure for each coloring map and obtain a randomized $O^*(4.32^k)$ algorithm. This completes the proof of Theorem 1.9. □

6 Deterministic Algorithms for Depth Three Circuits

In this section we obtain fast *deterministic* algorithms for $(k,n)-$MLC and $k-$MMD for depth-three arithmetic circuits. These are obtained by a simple application of Hadamard product combined with symmetrization. We will require the following.

Theorem 6.1 [4, Theorem 4] *Let A and B be noncommutative ABPs of sizes s_1 and s_2, computing homogeneous degree-k polynomials $f_1, f_2 \in \mathbb{F}\langle Y \rangle$, respectively. Then, we can compute an ABP of size $O(s_1 s_2)$ for the Hadamard product $f_1 \circ f_2$ in deterministic $\mathrm{poly}(s_1, s_2)$ time. Furthermore, if A and B are $\Pi^{[k]}\Sigma$ circuits, then we can compute a $\Pi^{[k]}\Sigma$ circuit for $f_1 \circ f_2$ in $\mathrm{poly}(s_1, s_2)$ time.*

We now prove Theorem 1.10.

Proof Let C be the given $\Sigma^{[s']}\Pi^{[k]}\Sigma$ circuit computing the polynomial $f \in \mathbb{F}[X]$. We first consider the $k-$MMD problem.

Suppose the coefficients are integers (without loss of generality, if we assume the underlying field is rationals). Let $C = \sum_{i=1}^{s'} C_i$ where each C_i is a $\Pi^{[k]}\Sigma$ circuit. We obtain a circuit for $C \circ^{\mathrm{S}} C(X)$ as follows. By Corollary 2.5, it suffices to obtain a circuit for $\widehat{C} \circ C^*(Y)$. Notice that $C^* = \sum_{i=1}^{s'} C_i^*$ and by Claim 5.2 we know that each C_i^* is a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ circuit which can be computed in $O^*(2^k)$ time. By distributivity, the problem of computing $\widehat{C} \circ C^*(Y)$ reduces to computing the noncommutative

Hadamard product of $s' \cdot 2^k$ many pairs of depth-two $\Pi^{[k]}\Sigma$ circuits. By Theorem 6.1, each such Hadamard product can be computed in poly(n, k) time. Hence, we obtain a depth three commutative $\Sigma^{[s'\cdot 2^k]}\Pi^{[k]}\Sigma$ circuit $\tilde{C}$ for $C \circ^S C(X)$ in $O^*(2^k)$ time. Note that m is a nonzero monomial in C if and only if $[m]\tilde{C} > 0$.

Let B be the poly(n, k) size ABP for $S_{n,k}$. Now the idea is to compute $\tilde{C} \circ^S B(\vec{1})$, and if it is nonzero, we know that C contains a degree-k multilinear term. Again this reduces to computing $s' \cdot 2^k$ scaled Hadamard products, each of the form $\Pi^{[k]}\Sigma \circ^S B(\vec{1})$. By Lemma 5.1, each such computation can be done in $O^*(2^k)$ time incurring a overall running time $O^*(4^k)$.

In the case of finite fields, the above proof does not work since $\tilde{C} \circ^S B(\vec{1})$ could be zero modulo the characteristic.The idea is similar to the proof of Theorem 1.9. But instead of random coloring maps we pick $\zeta_i : [n] \to [k]$ from the explicit (n, k)-family perfect hash functions constructed in [25], which is of size $e^k k^{O(\log k)} \log n$, and define a $\Pi^{[k]}\Sigma$ formula

$$P_i = \prod_{j=1}^{k} \sum_{\ell:\zeta_i(\ell)=j} x_\ell$$

for each coloring map ζ_i. Now for each i, we construct the circuit $C'_i = C \circ^S P_i$. As already explained each $C'_i(X)$ is a $\Sigma^{[s'\cdot 2^k]}\Pi^{[k]}\Sigma$ circuit and it can be obtained in deterministic $O^*(2^k)$ time. Clearly, if C contains a multilinear monomial, we can detect it by doing identity testing of each C'_i. Now we apply a result of Saxena [31] where he shows that the identity testing of a $\Sigma\Pi^{[k]}\Sigma$ circuit over finite fields can be done in deterministic $O^*(2^{ck})$ time where the constant $c \leq 5$. The final running time is $e^k k^{O(\log k)} O^*(2^{ck} + 2^k)$.

As a by-product of the above technique, we get a fast deterministic algorithm to compute the sum of the coefficients of degree-k multilinear monomials in a depth there circuit, solving the $(k,n)-$MLC problem. Notice that, we need to compute $C \circ^S B(\vec{1})$. As already explained, it can be obtained in deterministic $O^*(2^k)$ time.

□

Remark 6.2 The main result of [31, Theorem 1] is actually a polynomial identity testing algorithm for a larger class of circuits. It also uses the identity testing algorithm for noncommutative ABPs [29]. Indeed, the bound $c \leq 5$ is from [29, Theorem 4].

7 Discussion

In conclusion, we broadly compare the Hadamard product used in this paper with the apolar inner product used in the work of Pratt and others [10,27,28].

Recall, given two commutative homogeneous degree d polynomials f and g in $\mathbb{F}[X]$, the apolar inner product $\langle f, g \rangle$ is defined as follows.

$$\langle f, g\rangle = f\left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right) g = \sum_m m![m]f \cdot [m]g,$$

where the sum is over all degree-d monomials $m = x_1^{e_1} x_2^{e_2} \ldots x_n^{e_n} \in X_d$.

The *Waring rank* of a homogeneous degree-d polynomial $f \in \mathbb{F}[X]$ is the least r such that $f = \sum_{j=1}^{r} \alpha_j \cdot L_j^d$, where for each $j, \alpha_j \in \mathbb{F}$ and L_j is a homogeneous linear form. If f has Waring rank r and g is given by an arithmetic circuit Pratt [27,28] has shown that the apolar inner product of f and g can be computed in $O^*(r)$ time. Hence, using the Waring decomposition of the elementary symmetric polynomials (over the field of rationals) [23] yields faster algorithms [28] for $k-$MMD and $(k,n)-$MLC: $O^*(4.08^k)$-time for $k-$MMD and $O^*(n^{k/2})$-time for $(k,n)-$MLC) over the field of rationals.

However, the Waring decomposition does not appear to have a finite field analogue. For example, over $\mathbb{F}_2$ the polynomial xy has no Waring decomposition. In comparison, our algorithms are essentially oblivious to the underlying field.

More recently, Brand and Pratt [10] have shown that the apolar inner product of commutative polynomials f and g can be computed in $O^*(r)$ time if the partial derivative space of f has rank r. This strengthens Pratt's result [28] as the Waring rank of f is an upper bound on its partial derivative rank.

It is interesting to compare this with the Hadamard product based approach of our paper. We first note that the apolar inner product of polynomials $f, g \in \mathbb{F}[X]$ can be computed by first computing their scaled Hadamard product (see Sect. 2) and evaluating the resulting polynomial at $x_i = 1$ for each $i \in [n]$. Furthermore, the computation of Hadamard product of f and g can be done efficiently, as shown in Lemma 2.4 (Sect. 2), in time polynomial in the noncommutative algebra branching program complexity $b(f^*)$ (i.e. the minimum size of the ABP computing f^*) of the noncommutative polynomial f^*. The partial derivative space rank of f essentially coincides with $b(f^*)$:

Fact 7.1 *For any commutative polynomial f, the ABP complexity $b(f^*)$ of f^* is the dimension of the space of partial derivatives of f.*

Proof Let $f \in \mathbb{F}[X]$ be a homogeneous degree-k commutative polynomial, where $X = \{x_1, x_2, \ldots, x_n\}$ is a set of commuting variables. Let $Y = \{y_1, y_2, \ldots, y_n\}$ be a corresponding set of free noncommuting variables.

Recall that Nisan [26] has shown that the ABP complexity $b(g)$ of a noncommutative polynomial $g \in \mathbb{F}\langle Y\rangle$ of degree k is exactly $\sum_{\ell=1}^{k} \text{rank}(M_\ell(g))$ where the matrix $M_\ell(g)$ is defined as follows. The rows of $M_\ell(g)$ are indexed by words in Y^ℓ and the columns of $M_\ell(g)$ are indexed by words in $Y^{k-\ell}$. For some $w_1 \in Y^\ell$ and $w_2 \in Y^{k-\ell}$, the $(w_1, w_2)^{th}$ entry of $M_\ell(g)$ is $[w_1 w_2]g$.

We now consider the matrix $M_\ell(f^*)$ for the noncommutative polynomial $f^* \in \mathbb{F}\langle Y\rangle$. Let $w_1 \in Y^\ell$ and $w_2 \in Y^{k-\ell}$. Let $\sigma \in S_\ell$ and $\tau \in S_{k-\ell}$ be any permutations. By definition of f^*, it is clear that

$$M_\ell(f^*)(w_1, w_2) = M_\ell(f^*)(w_1^\sigma, w_2^\tau).$$

In particular, the row of $M_\ell(f^*)$ indexed by w_1 is identical to the row indexed by w_1^σ.

Furthermore, the row of $M_\ell(f^*)$ indexed by w_1 has the following structure for any $w_1 \in Y^\ell$: the w_2-th entry of this row is the same as the w_2^τ-th entry of this row.

Now consider the partial derivative matrix of f, $\tilde{M}_\ell(f)$ defined as follows. The rows of $\tilde{M}_\ell(f)$ are indexed by commuting degree-ℓ monomials in X_ℓ. For some $m \in \mathrm{X}_\ell$, the corresponding row is the coefficient vector of $\frac{\partial f}{\partial m}$.

It follows from the above considerations that a subset of rows of $\tilde{M}_\ell(f)$ labeled by monomials $m_1, m_2, \ldots, m_r$ are linearly independent iff for noncommuting monomials $w_1, w_2, \ldots, w_r$, such that $w_i \to m_i$, the rows of row of $M_\ell(f^*)$ indexed by the w_i are linearly independent.

Therefore, $\mathrm{rank}(M_\ell(f^*)) = \mathrm{rank}(\tilde{M}_\ell(f))$, which completes the proof. □

We conclude with the following arithmetic circuit complexity question, a positive answer to which would have interesting algorithmic implications: Is there a polynomial f over the rationals that is *positively* weakly equivalent[7] to the elementary symmetric polynomial $S_{n,k}$ such that the ABP complexity $b(f^*)$ is $O^*(2^k)$? It would even suffice to show that f^* has arithmetic circuits of size $O^*(2^k)$ to improve on the current best deterministic algorithm for the k-path problem.

Acknowledgements We are grateful to the referees for their comments and suggestions that have helped us improve the presentation. We thank anonymous reviewers of FSTTCS 2019 for their comments on an earlier version of this paper.

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM **42**(4), 844–856 (1995). https://doi.org/10.1145/210332.210337
2. Arvind, V., Chatterjee, A., Datta, R., Mukhopadhyay, P.: Fast exact algorithms using Hadamard product of polynomials. CoRR arXiv:1807.04496 (2018)
3. Arvind, V., Chatterjee, A., Datta, R., Mukhopadhyay, P.: Efficient black-box identity testing over free group algebra (accepted in RANDOM 2019). CoRR arXiv:1904.12337 (2019)
4. Arvind, V., Joglekar, P.S., Srinivasan, S.: Arithmetic circuits and the Hadamard product of polynomials. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15–17, 2009, IIT Kanpur, India, pp. 25–36 (2009)
5. Arvind, V., Srinivasan, S.: On the hardness of the noncommutative determinant. In: Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5–8 June 2010, pp. 677–686 (2010). https://doi.org/10.1145/1806689.1806782
6. Arvind, V., Srinivasan, S.: On the hardness of the noncommutative determinant. Comput. Complex. **27**(1), 1–29 (2018). https://doi.org/10.1007/s00037-016-0148-5
7. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Counting paths and packings in halves. In: Fiat, A., Sanders, P. (eds.) Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7–9, 2009. Proceedings, volume 5757 of Lecture Notes in Computer Science. Springer, pp. 578–586 (2009). https://doi.org/10.1007/978-3-642-04128-0_52
8. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Evaluation of permanents in rings and semirings. Inf. Process. Lett. **110**(20), 867–870 (2010). https://doi.org/10.1016/j.ipl.2010.07.005
9. Brand, C., Dell, H., Husfeldt, T.: Extensor-coding. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25–29, 2018, pp. 151–164 (2018). https://doi.org/10.1145/3188745.3188902

[7] A polynomial g is positively weakly equivalent to f if it has the same set of nonzero monomials, with any positive coefficients.

10. Brand, C., Pratt, K.: Parameterized applications of symbolic differentiation of (totally) multilinear polynomials. In: Bansal, N., Merelli, E., Worrell, J. (eds.) 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12–16, 2021, Glasgow, Scotland (Virtual Conference), volume 198 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 38:1–38:19 (2021)
11. Cameron, P.J.: Permutation Groups. London Mathematical Society Student Texts. Cambridge University Press, Cambridge (1999)
12. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, Berlin (2015)
13. Dell, H., Lapinskas, J.: Fine-grained reductions from approximate counting to decision. ACM Trans. Comput. Theory **13**(2):8:1–8:24 (2021)
14. Dell, H., Lapinskas, J., Meeks, K.: Approximately counting and sampling small witnesses using a colourful decision oracle. In: Chawla, S. (ed.) Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5–8, 2020. SIAM, pp. 2201–2211 (2020)
15. Demillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Inf. Process. Lett. **7**(4), 193–195 (1978). https://doi.org/10.1016/0020-0190(78)90067-4
16. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer (2013). https://doi.org/10.1007/978-1-4471-5559-1
17. Fischer, I.: Sums of like powers of multivariate linear forms. Math. Mag. **67**(1), 59–61 (1994). https://doi.org/10.1080/0025570X.1994.11996185
18. Harvey, D., van der Hoeven, J.: Faster polynomial multiplication over finite fields using cyclotomic coefficient rings. J. Complex. (2019). https://doi.org/10.1016/j.jco.2019.03.004
19. Harvey, D., van der Hoeven, J.: Integer multiplication in time $O(n \log n)$. Ann. Math. **193**, 563–617 (2021). https://doi.org/10.4007/annals.2021.193.2.4
20. Hüffner, F., Wernicke, S., Zichner, T.: Algorithm engineering for color-coding with applications to signaling pathway detection. Algorithmica **52**(2), 114–132 (2008). https://doi.org/10.1007/s00453-007-9008-7
21. Koutis, I.: Faster algebraic algorithms for path and packing problems. In: Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7–11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games, pp. 575–586 (2008). https://doi.org/10.1007/978-3-540-70575-8_47
22. Koutis, I., Williams, R.: Limits and applications of group algebras for parameterized problems. ACM Trans. Algorithms **12**(3):31:1–31:18 (2016). https://doi.org/10.1145/2885499
23. Lee, H.: Power sum decompositions of elementary symmetric polynomials. Linear Algebra Applications **492**(08) (2015)
24. Mahajan, M., Vinay, V.: Determinant: combinatorics, algorithms, and complexity. Chic. J. Theor. Comput. Sci. (1997). http://cjtcs.cs.uchicago.edu/articles/1997/5/contents.html
25. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and near-optimal derandomization. In: 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23–25 October 1995. IEEE Computer Society, pp. 182–191 (1995). https://doi.org/10.1109/SFCS.1995.492475
26. Nisan, N.: Lower bounds for non-commutative computation (extended abstract). In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5–8, 1991, New Orleans, Louisiana, USA, pp. 410–418 (1991). https://doi.org/10.1145/103418.103462
27. Pratt, K.: Faster algorithms via waring decompositions. CoRR (2018). arXiv:1807.06194
28. Pratt, K.: Waring rank, parameterized and exact algorithms. In: Zuckerman, D. (ed.) 60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9–12, 2019. IEEE Computer Society, pp. 806–823 (2019). https://doi.org/10.1109/FOCS.2019.00053
29. Raz, R., Shpilka, A.: Deterministic polynomial identity testing in non-commutative models. Comput. Complex. **14**(1), 1–19 (2005). https://doi.org/10.1007/s00037-005-0188-8
30. Ryser, H.J.: Combinatorial mathematics. Carus mathematical monographs. Mathematical Association of America; distributed by Wiley (New York, 1963). https://books.google.co.in/books?id=wOruAAAAMAAJ
31. Saxena, N.: Diagonal circuit identity testing and lower bounds. In: Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7–11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games, pp. 60–71 (2008). https://doi.org/10.1007/978-3-540-70575-8_6

32. Schwartz, J.T.: Fast probabilistic algorithm for verification of polynomial identities. J. ACM. **27**(4), 701–717 (1980)
33. Shpilka, A., Yehudayoff, A.: Arithmetic circuits: a survey of recent results and open questions. Found. Trends Theor. Comput. Sci. **5**(3–4), 207–388 (2010). https://doi.org/10.1561/0400000039
34. Strassen, V.: Vermeidung von divisionen. Journal für die reine und angewandte Mathematik **264**, 184–202 (1973)
35. Valiant, L.G.: Completeness classes in algebra. In: Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30–May 2, 1979, Atlanta, Georgia, USA, pp. 249–261 (1979)
36. Valiant, L.G., Skyum, S., Berkowitz, S., Rackoff, C.: Fast parallel computation of polynomials using few processors. SIAM J. Comput. **12**(4), 641–644 (1983). https://doi.org/10.1137/0212043
37. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 3rd ed. Cambridge University Press, Cambridge (2013)
38. Williams, R.R.: Counting solutions to polynomial systems via reductions. In: Seidel, R. (ed.) 1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7–10, 2018, New Orleans, LA, USA, volume 61 of OASICS. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 6:1–6:15 (2018)
39. Williams, R.R.: The polynomial method in circuit complexity applied to algorithm design (invited talk). In: 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15–17, 2014, New Delhi, India, pp. 47–60 (2014). https://doi.org/10.4230/LIPIcs.FSTTCS.2014.47
40. Williams, R.: Finding paths of length k in $O^*(2^k)$ time. Inf. Process. Lett. **109**(6), 315–318 (2009). https://doi.org/10.1016/j.ipl.2008.11.004
41. Williams, R.: Algorithms for circuits and circuits for algorithms. In: IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11–13, 2014, pp. 248–261 (2014). https://doi.org/10.1109/CCC.2014.33
42. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Proceedinsg of the International Symposium on Symbolic and Algebraic Computation, pp. 216–226 (1979)