Motivated by the recent developments on the complexity of non-commutative determinant and permanent [Chien et al. STOC 2011, Bläser ICALP 2013, Gentry CCC 2014] we attempt at obtaining a tight characterization of hard instances of non-commutative permanent.

We show that computing Cayley permanent and determinant on weighted adjacency matrices of graphs of component size six is \#P complete on algebras that contain $2 \times 2$ matrices and the permutation group $S_3$. Also, we prove a lower bound of $2^{\Omega(n)}$ on the size of branching programs computing the Cayley permanent on adjacency matrices of graphs with component size bounded by two. Further, we observe that the lower bound holds for almost all graphs of component size two.

On the positive side, we show that the Cayley permanent on graphs of component size $c$ can be computed in time $n^{t \text{poly}(t)}$, where $t$ is a parameter depending on the labels of the vertices.

Finally, we exhibit polynomials that are equivalent to the Cayley permanent polynomial but are easy to compute over commutative domains.

1 Introduction

In a seminal work, Valiant [25] showed that computing the permanent of an integer matrix is \#P complete. Further, in [24] Valiant conjectured that computing permanent of an integer matrix would require a super polynomial number of arithmetic operations unlike the determinant. This conjecture is known more popularly as ‘Valiant’s hypothesis’. Since then, there has been several research efforts leading to the development of algebraic complexity theory [8, 9].
Over the last decade, there has been intense research towards settling Valiant’s Hypothesis which have led to several new algebraic and algorithmic techniques. (See e.g., [23, 21] and references therein.)

It should be noted that algebraic complexity theory crucially depends on the ring in which arithmetic operations are performed. Early works on algebraic complexity focussed on commutative rings which led to Nisan studying algebraic computation over non-commutative rings. In his seminal paper [19], Nisan proved that any algebraic branching program computing the non-commutative permanent or determinant requires exponential size. Later on, this was generalized by Chien and Sinclair [10] for different non-commutative algebras, for example algebras given by matrices. More recently, Arvind and Srinivasan [4] established that over algebras that contain \( n \times n \) matrices, non-commutative determinant is equivalent to permanent. This was further extended to \( 2 \times 2 \) matrix algebras in [11]. Finally, the question of non-commutative determinant versus permanent was settled by Bläser [7]. He obtained almost a dichotomy, stating that if a specific quotient ring of the given algebra is non-commutative then computing the determinant is \( \#P \) hard. Otherwise, it is polynomial time computable under a reasonable assumption on the algebra. Gentry [16] obtained much simpler reductions compared to [4, 11, 7] using a completely different approach.

Given the hardness for general instances of non-commutative permanent and determinant, it is natural to ask: are there special cases of matrices for which non-commutative determinant/permanent can be computed efficiently? In the commutative setting, Barvinok [6] was the first to consider the problem of computing the permanent on special classes of matrices. To be precise, Barvinok [6] showed that computing permanent of bounded rank matrices can be done in polynomial time. A square matrix can also be viewed as the weighted adjacency matrix of a directed graph on \( n \) vertices. Flarup, Koiran and Lyaudet [13] considered restrictions on the structure of the weighted graph represented by a matrix. They showed that permanent on weighted adjacency matrices of graphs of bounded tree width characterized polynomial size arithmetic formulas. Further, Flarup and Lyaudet [14] extended the result to include other width measures such as clique-width and path-width. Finally Datta et al., [12] showed that permanent on planar graphs is as hard as the general case.

Along the lines of [6, 13] we study the complexity of non-commutative permanent on restricted classes of matrices. Unlike the commutative setting, we show that non-commutative permanent/determinant remain hard even in the case of very restricted classes of graphs and matrices.

There has also been study of algorithmic questions related to non-commutative circuits. It has been shown that the well known problem of testing if a circuit computes a polynomial identically zero can be done efficiently (see [2] and further references therein). Apart from the polynomial identity testing problems, there are several interesting problems on arithmetic circuits.

Computing the coefficient of a given monomial in the arithmetic circuit (\( \text{CoeffSLP} \)) is one of the well studied problems. It is known that \( \text{CoeffSLP} \) is at least as hard as \( \#P \) [15] However, Arvind et al., [2] showed that \( \text{CoeffSLP} \) can be done in polynomial time for non-commutative circuits. In this paper, we study a more a generalized variant
of CoeffSLP.

Our results

As a special case of graphs of bounded tree-width, we consider directed graphs with every strongly connected component having a constant number of vertices. We show that computing the Cayley permanent on adjacency matrices of graphs with connected components of size at most 6 is \#P hard (Theorem 3.2). Our proof is a careful modification of a recent proof by Gentry [16]. Looking to tighten the result further with respect component size, we show that any Algebraic Branching Program computing non-commutative permanent of graphs with component size bound by two is of size $2^{\Omega(n)}$ (Theorem 3.5). Further, we observe that for almost all graphs of component size two, the lower bound above holds (Theorem 3.7).

In the converse direction, we obtain a $n^{\text{poly}(t)}$ algorithm for computing the permanent on graphs of bounded component size. Here $t$ is a number that depends on the labelling of vertices in each of the component (Theorem 3.1). It should be noted that the hard instances obtained in Theorem 3.5 have the parameter $t \in \Omega(n)$ and hence our results indicate possible existence of a dichotomy for the complexity of non-commutative permanent on graphs of component size two.

In contrast to the commutative case, we show that computing the Cayley permanent on matrices of bounded rank is \#P complete (Corollary 4.2). This essentially follows from the fact that a natural non-commutative variant of elementary symmetric polynomial is computationally equivalent to commutative permanent (Theorem 4.1).

Finally, we show that computing the sum of coefficients of all monomials $m'$ that divide a given monomial $m$ in an arithmetic circuit can be done with a polynomial size circuit (Theorem 5.3) in the non-commutative setting.

2 Preliminaries

For definitions of complexity classes the reader is referred to any of the standard text books, e.g., [1]. Let $K$ be a field and $S = K[x_1, \ldots, x_n]$ be the ring of polynomials over $K$ in $n$ variables. Let $R$ denote a non-commutative ring with identity and associativity property. Unless otherwise stated, we assume that $R$ is an algebra over $K$ and contains the algebra of $n \times n$ matrices with entries from $K$ as a sub algebra. A non-commutative monomial is an ordered sequence of variables. Degree of a monomial $m$ is the length of the sequence and is denoted by $\text{deg}(m)$.

An arithmetic circuit is a directed acyclic graph with labeled vertices which have in-degrees zero or two. Every vertex of zero in-degree is called an input gate and is labeled by an element in $R \cup \{x_1, \ldots, x_n\}$. Vertices of in-degree two are called internal gates and have their labels from $\{\times, +\}$. An arithmetic circuit has at least one vertex of out degree zero called an output gate. In most of our applications, we assume that an arithmetic circuit has exactly one output gate. A polynomial $p_g$ in $R[x_1, \ldots, x_n]$ can be associated with every gate $g$ of an arithmetic circuit defined in an inductive fashion. Input gates compute their label. Let $g$ be an internal gate with left child $f$ and right
child $h$, then $p_g = p_f \circ \text{op} \circ p_h$ where $\text{op}$ is the label of $g$. The polynomial computed by the circuit is the polynomial at one of the output gates and denoted by $p_C$. The size of an arithmetic circuits is the number of gates in it and is denoted by $\text{size}(C)$.

It should be noted that a polynomial computed by an arithmetic circuit $C$ can have coefficients as big as $2^{2\text{size}(C)}$. We restrict ourselves to circuits where the coefficients can be represented in at most $\text{poly}(\text{size}(C))$ bits.

An algebraic branching program (ABP) is a directed acyclic graph with two special nodes $s, t$ and edges labeled by variables or constants in $R$. The weight of a path is the product of the weights of its edges. The polynomial computed by an ABP $P$ is the sum of the weights of all $s \leadsto t$ paths in $P$, and is denoted by $p_P$.

Over a commutative ring, defining the determinant or permanent is straightforward. However, there can be many possibilities for defining them depending on the ordering of the variables (see for example the survey in [5]). We will use the well known definition of the Cayley determinant and Cayley permanent. Let $X = (x_{ij})_{1 \leq i,j \leq n}$ be an $n \times n$ matrix with distinct variables $x_{i,j}$. Then,

$$\text{Cayley - det}(X) = \sum_{\sigma \in S_n} \text{sgn}(\sigma)x_{1,\sigma(1)} \cdots x_{n,\sigma(n)},$$

$$\text{Cayley - perm}(X) = \sum_{\sigma \in S_n} x_{1,\sigma(1)} \cdots x_{n,\sigma(n)}.$$  

In the above, $S_n$ denotes the set of all permutations on $n$ symbols.

**Remark 2.1.** Note that $\text{Cayley - det}$ and $\text{Cayley - perm}$ can also be seen as functions taking $n \times n$ matrices with entries from $R$ as input.

The tensor product of two matrices $A, B \in \mathbb{K}^{n \times n}$ with entries $a_{i,j}, b_{i,j}$ is in this case equal to the Kronecker product and is given by

$$\begin{pmatrix}
  a_{1,1}B & \ldots & a_{1,n}B \\
  \vdots & \ddots & \vdots \\
  a_{n,1}B & \ldots & a_{n,n}B
\end{pmatrix}.$$  

We use the notion of read once certificate for ABPs as in [17]. Let $B$ be an ABP over disjoint sets of variables $X \cup Y$, with $|X| = n$ and $|Y| = m$. Let $p_B(X,Y)$ be the polynomial computed by $B$. We use the following result from [17]:

**Proposition 1.** Let $B$ be an ABP on $X \cup Y$ read-once certified in $Y$. Then the polynomial

$$\sum_{e_1,e_2,\ldots,e_m \in \{0,1\}^m} p_B(X,e_1,\ldots,e_m)$$

can be computed by an ABP of size $\text{poly}(\text{size}(B))$.

It should be noted that the proof of Proposition 1 given in [17] does not require commutativity property for the variables and translates easily to the non-commutative setting.
3 Non-commutative Permanent

Upper Bound

A directed graph $G$ on $n$ vertices is said to be of bounded component size if every strongly connected component of $G$ contains at most $c$ vertices for some fixed constant $c$. In this section we explore the complexity of computing the non-commutative permanent on the adjacency matrix of graphs of bounded component size. The adjacency matrix will have distinct variables denoted by $x_{i,j}$ and the constant zero as entries. We first give an upper bound for the time needed for any algorithm to compute the Cayley permanent on these graphs with respect to the parameter defined below.

**Definition 3.1.** Let $G = (V, E)$ be a graph with component size bounded by $c$, where $V = [n]$. The nearness parameter $\text{near}(C)$ of a strongly connected component of $G$ is defined as $\text{near}(C) = \max_{i,j \in C} |i - j|$. The nearness parameter of $G$ is defined as $\text{near}(G) = \max_{C} \text{near}(C)$, where the maximum is taken over the set of all strongly connected components in $G$.

**Theorem 3.1.** Let $G$ be a directed graph with bounded component size with the edges labeled by elements from $R$. Then the Cayley permanent of the adjacency matrix of $G$ can be computed in time $n^{\text{poly}(\text{near}(G))}$.

**Proof.** For an edge $(i, j) \in E(G)$, let $a_{i,j} \in R$ denote its weight. Let $A_G$ be the weighted adjacency matrix of $G$. Note that, the Cayley permanent of $A_G$ equals the sum of weights of cycle covers in $G$, where weight of a cycle cover $C$ is the product of weights if edges in $C$ multiplied in the Cayley order.

We describe a non-deterministic log-space bounded procedure $P$ that guesses a cycle cover $C$ in $G$ and outputs the product of weights of $C$ with respect to the Cayley ordering. Additionally, we ensure that the algorithm $P$ uses the non-deterministic bits in a read-once fashion, and by the closure property of ABP under read-once exponential sums [17], we obtain a deterministic polynomial time algorithm with running time $n^{O(\text{near}(G))}$. Suppose $C_1, \ldots, C_r$ are the strongly connected components of $G$, sorted in the ascending order of the smallest vertex in the component. We represent a cycle cover in $G$ as a permutation $\gamma$, where $\gamma(i)$ is the successor of vertex $i$ in the cycle cover represented by $\gamma$. We begin with the description of the non-deterministic procedure $P$:

1. Initialize $\text{count} := 1$, $T := \emptyset$, $\gamma :=$ the cycle cover of the empty graph, $f = 1$.
2. For $1 \leq i \leq r$ repeat steps 4 & 5.
3. Non deterministically guess a cycle cover $\gamma'$ in $C_i$, and set $\gamma = \gamma \uplus \gamma'$, $T = T \cup V(C_i)$, where $V(C_i)$ is the vertices in $C_i$.
4. While there is a vertex $k \in T$ with $k = \text{count}$ do the following:
   a) Set $f = f \times a_{k, \gamma(k)}$.
   b) Set $\text{count} := \text{count} + 1$, $T := T \setminus \{k\}$.
If \( \text{count} = n \), then output \( f \) and accept.

Let \( \text{Acc}(G) \) be the sum of the weights output by the algorithm on all accepting paths. Then

**Claim 1.** \( \text{Acc}(G) = \text{perm}(G) \). Moreover, the algorithm \( P \) uses \( O(\text{near}(G) + c^2) \log n \) space, and is read-once on the non-deterministic bits.

**Proof of the Claim.** Firstly, note that a permutation \( \gamma \in S_n \) is a cycle cover of \( G \) if and only if it can be decomposed into vertex disjoint cycle covers \( \gamma_1, \ldots, \gamma_r \) of the strongly connected components \( C_1, \ldots, C_r \) in \( G \). Thus Step 3 enumerates all possible cycle covers in \( G \). Also, the weights outputted at every accepting path are in the Cayley order.

For bounding the space used by the procedure, note that at any stage of the algorithm we have

\[
T = \{ k \mid \text{count} < k \text{ and } k \text{ occurs in the components already explored} \}.
\]

Moreover, it is sufficient to store only part of the partial cover \( \gamma \) restricted to the vertices in \( T \). Thus the number of extra registers used by the algorithm is at most \( |T| \leq \text{near}(G) \), where \( c \) is the maximum size of a strongly connected component. Additionally, while guessing a cycle cover for \( C_i \), \( P \) may need to remember all of the vertices in \( C_i \). Thus the overall space used is \( O((\text{near}(G) + c) \log n) \).

Combining the above algorithm with the closure property of algebraic branching programs over read-once variables given by Proposition 1, we get a non-commutative arithmetic branching program computing \( \text{perm}(G) \). It can be seen that size of the resulting branching program is at most \( 2^{O((c+\text{near}(G)) \log n)} = n^{O(c+\text{near}(G))} \).

**Lower Bound**

The algorithm obtained in Theorem 3.1 requires super polynomial time when \( \text{near}(G) = \omega(1) \). As a natural curiosity, we explore the possibility of the bound in Theorem 3.1 being tight. Though we do not yet have a complete answer, we can give a \#P completeness result for a specific graph of component size at least 6. The completeness result is obtained by a careful analysis of the parameters in the reduction from \#SAT to non-commutative determinant given recently by Gentry [16].

**Theorem 3.2.** Let \( R \) be an algebra over a field \( K \) containing the algebra of \( 2 \times 2 \) matrices over \( K \). Computing the Cayley determinant on graphs with component size 6 with edges labeled from \( R \) is \#P complete.

**Proof.** (Sketch) It is known that counting the number of satisfying assignments in a 2-CNF formula where every variable occurs at most three times is already \#P complete ([20]). Let \( \phi \) be a 2-CNF where every variable occurs at most three times with \( k \) clauses. We complete the proof by a careful analysis of the reduction given in Theorem 6 of [16] applied to \( \phi \).
We recall the definition of a product program. A product program over an algebra \( R \) of length \( n \) takes a \( \ell \)-bit input \( x = (x_1, \ldots, x_\ell) \) and a sequence of instructions \((j_i, a_{i,0}, a_{i,1})_{i \in [n]}\) and computes \( \prod_{i \in [n]} a_{i, x_{j_i}} \) where \( x_{j_i} \) is the \( j \)-th position of \( x \).

Let \( r = \begin{pmatrix} 0 & -1 \\ 1 & -1 \end{pmatrix} \) and \( s = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \).

Lemma 3 in [16] gives a product program of length \( 2^2 + 2^2 - 2 \) for computing a disjunction of two literals. In fact the program for \( x_1 \lor x_2 \) is given by

\[
(1, (s, I)), (2, (r, I)), (1, (s, I)), (2, r^{-1}, I),
\]

where \( I \) is the identity matrix.

Let \( t \) be a \( 2 \times 2 \) matrix as in [16, Theorem 6]. Suppose \( P_c \) is the product program as given above for the clause indexed by \( c \) for \( 1 \leq c \leq k \). Then product program for \( \phi \) is then given by

\[
P = \left( \prod_{1 \leq c \leq k} t \cdot P_c \right) t.
\]

This immediately shows if every variable occurs at most three times in \( \phi \), the product program above reads a bit of the input at most 6 times. Let

\[
C_\ell = \{ i \in [n] \mid \text{the } i \text{-th instruction in } P \text{ uses the } \ell \text{-th bit of the input} \} \quad 1 \leq \ell \leq 4k.
\]

We have \( C_\ell \leq 6 \) for all \( 1 \leq \ell \leq 4k \) by the above argument. Let \( C_\ell \) have the elements \( i_{\ell,1}, \ldots, i_{\ell,|C_\ell|} \). Let \( \pi_0 \) be the identity permutation and \( \pi_1(i_{\ell,\kappa}) = i_{\ell,\kappa+1} \mod |C_\ell| \). Define the following permuted “block barbershop” ([16]) matrix.

\[
M[i, j] = \begin{cases} 
(-1)^{|C_\ell|-1} a_{i,1} & \text{if } i = i_{\ell,1} \text{ and } j = i_{\ell,2} \text{ for some } \ell \\
a_{i,b} & \text{otherwise if } j = \pi_0(i) \\
0 & \text{otherwise}.
\end{cases}
\]

As the rows and columns for \( C_i, C_j \) for \( i \neq j \) are disjunct this matrix corresponds to cycles of length \( |C_\ell| \leq 6 \). This concludes the proof.

As the determinant is a lower bound for the permanent it follows that computing the Cayley permanent on graphs with bounded components of size 6 is \#P hard.

It is not clear if the above arguments can be extended to graphs of component size less than 6. Nevertheless, we will show that any branching program computing non-commutative permanent of directed graphs with component size bounded by 2 must be of exponential size. This shows that the upper bound in Theorem 3.1 is tight up to a \( \log n \) factor in the exponent, however with a different but related parameter.

Our proof crucially depends on Nisan’s [19] partial derivative technique. We begin with some notations following his proof. Let \( f \) be a non-commutative degree \( d \) polynomial in \( n \) variables. Let \( B(f) \) denote the smallest size of a non-commutative ABP computing \( f \). For \( k \in \{0, \ldots, n\} \) let \( M_k(f) \) be the matrix with rows indexed by all possible sequences containing \( k \) variables and columns indexed by all possible sequences containing \( d - k \).
variables (repetitions allowed). The entry of $M_k(f)$ at $(x_{i1} \ldots x_{ik}, x_{j1} \ldots x_{jd-k})$ is the coefficient of the monomial $x_{i1} \cdots x_{ik} \cdot x_{j1} \cdots x_{jd-k}$ in $f$. Nisan established the following result:

**Theorem 3.3.** For any homogeneous polynomial $f$ of degree $d$,

$$B(f) = \sum_{k=0}^{d} \text{rank}(M_k(f)).$$

We prove lower bounds for the Cayley permanent of graphs with every strongly connected component of size exactly 2, i.e., each strongly connected component being a two-cycle. Note that any collection of $n/2$ vertex disjoint two-cycles can be viewed as a permutation $\pi \in S_n$ consisting of disjoint transpositions and that $\pi$ is in fact an involution. The permutation $\pi$ can be seen as an alternate representation of graphs with connected component size 2.

For a permutation $\pi \in S_n$ let the cut at $i$ denoted by $C_i(\pi)$ be the set of pairs $(j, \pi(j))$ that cross $i$, i.e.,

$$C_i(\pi) = \{ (j, \pi(j)) \mid i \in [j, \pi(j)] \cup [\pi(j), j] \}.$$

The cut parameter $t(\pi)$ of $\pi$ is defined as $t(\pi) = \max_{1 \leq k \leq n} |C_k(\pi)|$. Let $G$ be the collection of vertex disjoint 2-cycles denoted by $(a_1, b_1), \ldots, (a_{n/2}, b_{n/2})$, where $n$ is even. The corresponding involution is $\pi = (a_1, b_1) \cdots (a_{n/2}, b_{n/2})$. Without loss of generality, assume that $a_i < b_i$, and $a_1 < a_2 < \ldots < a_{n/2}$. We first observe that the upper bound given in Theorem 3.1 holds true if we consider $\max_k C_k(\pi)$ instead of $\text{near}(G)$.

**Lemma 3.1.** Let $G$ be a collection of disjoint 2-cycles and self loops where every edge is labeled by a distinct variable or a constant from $R$. Then there is an ABP of size $n^{\text{poly}(t)}$ computing the Cayley permanent on $G$, where $t = \max_k C_k(\pi)$ and $\pi$ is the involution corresponding to $G$.

**Proof.** The algorithm is the same as in Theorem 3.1. We only need to argue the space bound as in Claim 1. First note that either $a_i = i$, or $i$ has already occurred in one of the involutions $(a_i, b_1), \ldots, (a_{i-1}, b_{i-1})$. When the algorithm processes the component corresponding to the involution $(a_i, b_i)$, it needs to remember the outgoing edge chosen for $b_i$ (either the self loop or the edge $b_i \rightarrow a_i$). Thus at any stage, the number of edges that needs to be stored is bounded by $t = \max_k C_k(\pi)$. The rest of the arguments are exactly the same as in Theorem 3.1. \hfill \Box

**Lemma 3.2.** Let $G$ be a collection of $\ell$ disjoint two-cycles described by the involution $\pi$ and self loops at every vertex with edge labeled by distinct variables. Then $M_\ell(\text{Cayley-\text{perm}}(G))$ contains the following matrix as a sub-matrix:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^{\otimes t},$$

where $t = \max_k C_k(\pi)$ and $A^{\otimes t}$ is the tensor product of $A$ with itself $t$ times.
Proof. Let \( k \in \{\ell\} \) and \( m = |C_k(\pi)| \leq \ell \). Let \( C_k(\pi) = (a_{i_1}, b_{i_1}), \ldots, (a_{i_m}, b_{i_m}) \) be such that \( a_{i_j} \leq k \leq b_{i_j} \) for a \( j \). Let \( G_k \) be the graph restricted to involutions in \( C_k(\pi) \). By induction on \( m \), we argue that \( M_m(\text{Cayley-perm}(G_k)) \) contains \( I_2^\otimes m \) as a sub-matrix, where \( I_2 \) is the \( 2 \times 2 \) identity matrix. The lemma would then follow since \( M_m(\text{Cayley-perm}(G_k)) \) is itself a sub-matrix of \( M_\ell(\text{Cayley-perm}(G)) \).

We begin with \( m = 1 \) as the base case. Consider the transposition \( (a_{i_j}, b_{i_j}) \), with \( a_{i_j} \leq k \leq b_{i_j} \). The corresponding two cycle has four edges, and let \( f_{i_j} \) be the Cayley permanent of this graph. Then \( M_1(f_{i_j}) \) has the \( 2 \times 2 \) identity matrix as a sub-matrix. Let us dwell on this simple part. For ease of notation let the variables corresponding to the self-loops be given by \( x_a, x_b \) for \( (a_{i_j}, a_{i_j}) \) and \( (b_{i_j}, b_{i_j}) \) respectively and the edge \( (a_{i_j}, b_{i_j}) \) by \( x_{(a,b)} \) and the edge \( (b_{i_j}, a_{i_j}) \) by \( x_{b,a} \). Now our matrix has monomials \( x_a, x_{a,b} \) as rows and \( x_b, x_{b,a} \) as columns. We can ignore the other orderings as these will always be zero. As the valid cycle covers are given by \( x_a x_b \), and \( x_a b x_{b,a} \), the proof is clear.

For the induction step, suppose \( m > 1 \). Suppose \( a_1 < a_2 < \ldots < a_m \). Let \( G' \) be the graph induced by \( C_k(\pi) \setminus \{a_1, b_1\} \). Let \( M' = M_{m-1}(\text{Cayley-perm}(G'_k)) \). The rows of \( M' \) are labeled by monomials consisting of variables with first index \( \leq k \) and the columns of \( M' \) are labeled by monomials consisting only of variables with first index \( > k \). Let \( M = M_m(\text{Cayley-perm}(G_k)) \). \( M \) can be obtained from \( M' \) as follows: Make two copies of the row labels of \( M' \), the first one with monomials pre-multiplied by \( x_{a_1,a_1} \), and the second pre-multiplied by \( x_{a_1,b_1} \). Similarly, make two copies of the columns of \( M' \), the first by inserting \( x_{b_1,a_1} \) to the column labels of \( M' \) at appropriate position, and then inserting \( x_{b_1,b_1} \) similarly. Now, the matrix \( M \) can be viewed as a obtained by two copies of \( M' \) that are placed diagonally (block diagonal). Thus we conclude \( M = M' \otimes I_2 \), the lemma now follows by the induction hypothesis.

This proof above can also be visualized using basic facts from Quantum Computation. Consider a cycle \( (a_{i_j}, b_{i_j}) \) with \( \pi(a_{i_j}) < k \) and \( \pi(b_{i_j}) > k \). We can assign 2 Q-bits for edges outgoing from these vertices. A zero means the edge \( (a_{i_j}, a_{i_j}) \) is taken and a one that \( (a_{i_j}, b_{i_j}) \) is taken. We assign values in the same manner from edges going out of \( b_{i_j} \). It is clear that for valid cycle covers these pair form an entangled quantum state with two Q-bits but only two states (the \((0,0)\) state and the \((1,1)\) state). Now adding Q-bits which have no connection to the previous two cycles gives us the tensor product of the states.

Remark 3.1. It should be noted that, in the induction step above, if \( a_1, b_1 < k \), then \( \text{rank}(M) = \text{rank}(M') \), and hence the ordering of the variables is crucial in the above argument.

Theorem 3.4. Let \( G \) be a collection of disjoint two cycles described by the involution \( \pi \) and self loops at every vertex, with edges labeled by distinct variables. Then any non-commutative ABP computing Cayley permanent on \( G \) has size at least \( \max_k 2^{\Omega(C_k(\pi))} \).

Proof. It is enough to argue that for every \( k \), there is an \( \ell \) with \( \text{rank}(M_\ell(f)) \geq 2^{\Omega(C_k(\pi))} \), then the claim follows from Theorem 3.3. Let \( 2\ell = C_k(\pi) \), and suppose
$(a_{i_1}, b_{i_1}), \ldots, (a_{i_\ell}, b_{i_\ell})$ are the transpositions crossing $k$. Let $G'$ be the sub-graph of $G$ induced by the vertices corresponding to the transpositions above. Let $f = \text{Cayley-perm}(G')$. Applying Lemma 3.2 on $G'$ we conclude that $M_\ell(f)$ has as a sub-matrix

$$
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix} \otimes C_k(\pi)
$$

i.e., the identity matrix of dimension $2^{C_k(\pi)} \times 2^{C_k(\pi)}$. Note that $f$ can be obtained by setting weights of the self loops of vertices not in $G'$ to zero, and setting the remaining variables to 1. Moreover, the matrix $M_\ell(f)$ is a sub matrix of $M_\ell(\text{Cayley-perm}(G))$ obtained by relabelling the rows and columns as per the substitution mentioned above, and removing rows and columns that are labels by zero. From the arguments above, we conclude $\text{rank}(M_\ell(\text{Cayley-perm}(G))) \geq \text{rank}(M_\ell(f)) \geq 2^{C_k(\pi)}$.

The above structural characterization can be used to prove lower bounds for the Cayley permanent of a collection of 2-cycles. Let $\pi = (a_1, b_1) \cdots (a_{n/2}, b_{n/2})$, $a_1 < a_2 < \ldots < a_{n/2}$ be an involution. Then the graph associated with $G$ is the collection of 2-cycles $(a_1, b_1), \ldots, (a_{n/2}, b_{n/2})$ and self loops at every vertex.

**Theorem 3.5.** There exists an involution $\pi$ such that any ABP computing the Cayley permanent of the graph $G$ associated with $\pi$ is of size $2^{\Omega(n)}$. Moreover, $\text{near}(G) \in \Omega(n)$.

**Proof.** Consider the ordering $\pi(1) = 1, \pi(2) = n/2+1, \pi(3) = 3, \pi(4) = n/2+2, \ldots, \pi(n-1) = n/2, \pi(n) = n$. It can be seen that $\max_k C_k(\pi) = n/4$. Thus by Theorem 3.4 the result follows. \qed

**Corollary 3.1.** Any non-commutative ABP computing the Cayley permanent of all matrices represented by outer planar graphs require size $2^{\Omega(n)}$, where $n$ is the number of vertices.

**Proof.** Given an involution $\pi$ with associated graph $G$, we will construct an outer planar graph $G'$ with self loops. Let $\pi = (a_1, b_1) \cdots (a_{n/2}, b_{n/2})$. Arrange $a_1, \ldots, a_{n/2}$ on a horizontal line in that order, and similarly arrange $b_1, \ldots, b_{n/2}$ on a different horizontal line. Add the edges in $G$. Add the edges $a_i \rightarrow a_{i+1}$ and $b_{i+1} \rightarrow b_i$, $1 \leq i < n/2$ each with weight 1. Note that $\text{perm}(G) = \text{perm}(G')$, and the result follows from Theorem 3.5. \qed

We can also prove a small corollary on different orderings.

**Corollary 3.2.** Let $\pi$ be the involution in Theorem 3.5. Let $\pi'$ be an involution which can be constructed from $\pi$ with at most $O(\log n)$ transpositions. Then any non-commutative ABP computing the Cayley permanent on $\pi'$ has exponential size.

**Proof.** Any transposition removed at most a value of 4 from the cut parameter and hence our new complexity is at least $2^{O(n-\log n)}$ which is still exponential in size. \qed
Density of hard instances

The hard instances given in Theorem 3.4 have $\text{near}(G) \in \Omega(n)$, for which the algorithm in Theorem 3.1 is not polynomial time bounded. Thus Theorems 3.1 & 3.5 can be seen as dual to each other. Here we show that in fact almost all involutions $\pi$ have $t(\pi) = \Omega(n)$, and hence implying that almost all graphs with component size two are hard instances as in Theorem 3.4.

As before, let $n = 2m$ be even. Then an involution $\pi$ on $\{1, \ldots, n\}$ with $\pi(i) \neq i$ represents a collection of $m$ intervals

$$I_\pi = \{[i, \pi(i)] \mid 1 \leq i \leq n, \ i < \pi(i)\} \cup \{[\pi(i), i] \mid 1 \leq i \leq n, \ i > \pi(i)\}.$$  
Let $H_\pi$ be the interval graph formed by the intervals in $I_\pi$.

**Lemma 3.3.** Let $\pi$ be an involution and $H_\pi$ be the interval graph as defined above. Then $t(\pi) \geq e/n$, where $e$ is the number of edges in $H_\pi$.

**Proof.** For every edge $(a, b)$ in $H_\pi$, the corresponding intervals $I_a = [i, \pi(i)]$ and $I_b = [j, \pi(j)]$ have non empty intersection. Suppose $i < j$, then $j \in [i, \pi(i)]$. (In the case when $j > \pi(j)$, we have $\pi(j) \in [i, \pi(i)]$. Other cases can be handled analogously.) Thus every edge in $H_\pi$ contributes at least one distinct interval $[i, \pi(i)]$ with $i \leq k \leq \pi(i)$, i.e., it contributes a value to $C_k(\pi)$. Then $e \leq \sum_k C_k(\pi)$. This concludes the proof. 

In [22] Scheinerman showed that, random interval graphs have $\Omega(n^2)$ edges with high probability, i.e.,

**Theorem 3.6.** [22] Let $H_\pi$ be an interval graph where $\pi$ is an involution on $[n]$ chosen uniformly at random. Then $H_\pi$ has at least $m^2/3 - m^{7/4}$ edges with probability at least $1 - 1/\sqrt{n}$.

As an immediate Corollary, we have:

**Corollary 3.3.** For an involution $\pi$ on $[n]$ chosen uniformly at random, we have $t(\pi) = \Omega(n)$ with probability $1 - 1/\sqrt{n}$.

Combining Corollary 3.3 with Theorem 3.4 we get the following:

**Theorem 3.7.** For all but a $1/\sqrt{n}$ fraction of graphs $G$ with connected component size 2, any ABP computing the non-commutative permanent on $G$ requires size $2^{\Omega(n)}$.

**Planar Graphs**

Like in the case of the commutative setting, it can be shown that permanent/determinant on planar graphs is as hard as the general case. We observe that the reduction in [12] extends to the case of non-commutative permanent.

**Theorem 3.8.** 1. noncom–perm $\leq^p_m$ noncom–planar–perm; and
2. noncom–det $\leq^p_m$ noncom–planar–det.
Proof of Theorem 3.8. The proof is essentially the same as in [12]. We give a brief sketch here for the sake of completeness. Let $G$ be a weighted digraph. Consider an arbitrary embedding $E$ of $G$. Obtain a new graph by changing the graph as follows:

- For each pair of edges $(u, v)$ and $(u', v')$ that cross each other in the embedding $E$, do the following:
  - introduce two new vertices $a$ and $b$; and
  - new edges $\{(a, b), (b, a), (u', a), (a, v), (u, b), (b, v')\}$ replacing $(u, v)$ and $(u', v')$.

Note that any of the iterations above do not introduce any new crossings, and hence the process terminates after at most $O(n^2)$ many steps, where $n$ is the number of vertices in $G$. Weight of $(u, v)$ is given to $(v, a)$ and $(u', v')$ is given to $(v', b)$. The remaining edges get the weight 1. By the construction, we can conclude that $\text{perm}(G) = \text{perm}(G')$ and $\det(G) = \det(G')$. \qed

4 Some hard polynomials

We demonstrate some polynomial families whose commutative variants are easy but certain non-commutative variants being \#P complete. The elementary symmetric polynomial $\text{Sym}_{n,d}$ is defined as

$$\text{Sym}_{n,d}(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| = d} \prod_{i \in S} x_i.$$ 

There are several non-commutative variants of the above polynomial. The first one is analogous to the Cayley permanent, i.e.,

$$\text{Cayley-Sym}_{n,d} = \sum_{S = \{i_1 < i_2 < \cdots < i_d\}} \prod_{j=1}^d x_{i_j}.$$ 

It is not hard to see that the above mentioned non-commutative version of $\text{Cayley-Sym}_{n,d}$ can be computed by depth 3 non-commutative circuits for every value of $d \in [n]$. However, the above definition is not satisfactory, since it is not invariant under permutation of variables, which is the inherent property of elementary symmetric polynomials. We define a variant of non-commutative elementary symmetric polynomial which is invariant under the permutation of variables.

$$\text{nc-Sym}_{n,d}(x_1, \ldots, x_n) \triangleq \sum_{\{i_1, \ldots, i_d\} \subseteq [n]} \sum_{\sigma \in S_d} \prod_{j=1}^d x_{i_{\sigma(j)}}.$$ 

We show that with coefficients from the algebra of $n \times n$ matrices allowed, $\text{nc-Sym}_{n,d}$ cannot be computed by polynomial size circuits unless $\text{VP} = \text{VNP}$. 

12
Theorem 4.1. Over any algebra having the algebra of $n \times n$ matrices as a subalgebra, $\text{nc-Sym}_{n,n}$ does not have polynomial size arithmetic circuits unless $\text{perm}_n \in \text{VP}$.

**Proof.** Suppose that $\text{nc-Sym}_{n,n}$ has a circuit $C$ of size polynomial in $n$. We need to show that $\text{perm} \in \text{VP}$. We crucially use the fact that the non-commutative Hadamard product $f \odot g$, where $f$ has polynomial size circuits and $g$ has polynomial size ABPs, can be computed efficiently [3, 4]. The Hadamard product for two polynomials $\sum_i \alpha_i m_i$, $\sum_i \beta_i m_i$ of degree $d$ where the sum is over all possible monomials of degree smaller or equal to $d$ is defined as $\sum_i \alpha_i \beta_i m_i$. Let $X = (x_{i,j})_{1 \leq i,j \leq n}$ be matrix of variables, and $y_1, \ldots, y_n$ be distinct variables different from $x_{i,j}$. In the commutative setting, it was observed in [20] that $\text{perm}(X)$ equals the coefficient of $y_1 \cdots y_n$ in the polynomial

$$P(X, Y) \triangleq \prod_{i=1}^{n} \left( \sum_{j=1}^{n} x_{i,j} y_j \right)$$

over the polynomial ring $\mathbb{K}[x_{1,1}, \ldots, x_{n,n}]$. However, the same cannot be said in the case of non-commuting variables. If $x_{i,j} y_k = y_k x_{i,j}$ for $i, j, k \in [n]$, then in the non-commutative development of [4], the sum of coefficients of all permutations of the monomial $y_1 \cdots y_n$ equals $\text{perm}(X)$. Hence the value $\text{perm}(X)$ can be extracted using a Hadamard product with $\text{nc-Sym}_{n,n}(y_1, \ldots, y_n)$, and then substituting $y_1 = 1, \ldots, y_n = 1$. However, we cannot assume $x_{i,j} y_k = y_k x_{i,j}$, since the Hadamard product may not be computable under this assumption. Let $\ell = \sum_{i,j} x_{i,j}$. Then we have

Claim 2. $\text{perm}(X) = (\text{nc-Sym}_{n,n}(\ell y_1, \ldots, \ell y_n) \odot P)(y_1 = 1, \ldots, y_n = 1)$.

**Proof of the Claim.** Given a permutation $\sigma \in S_n$, there is a unique monomial $m_{\sigma} = x_{1,\sigma(1)} y_{\sigma(1)} \cdots x_{n,\sigma(n)} y_{\sigma(n)}$ in $P$ containing the variables $y_{\sigma(1)}, \ldots, y_{\sigma(n)}$ in that order. Thus taking Hadamard product with $P$ filters out all monomials but $m_{\sigma}$ from the term $\prod_{i=1}^{n} \ell y_{\sigma(i)}$. The monomials where a $y_j$ occurs more than once are eliminated by $\text{nc-Sym}_{n,n}(\ell y_1, \ldots, \ell y_n)$. Thus the only monomials that survive in the Hadamard product are of the form $m_{\sigma}, \sigma \in S_n$. Now substituting $y_i = 1$ for $i \in [n]$ gives the required result.

Note that the polynomial $P(X, Y)$ can be computed by an ABP of size $O(n^2)$. Then, by [3, 4], we obtain an arithmetic circuit $D$ of size $O(n^2 \text{size}(C))$ that computes the polynomial $\text{nc-Sym}_{n,n} \odot P$. Substituting $y_1 = 1, \ldots, y_n = 1$ in $D$ gives the required arithmetic circuit for $\text{perm}(X)$.

Note that by considering the following signed variant of $\text{nc-Sym}_{n,n}$, we can obtain a result analogous to Theorem 4.1 with Cayley-det. Let

$$\text{snc-Sym}_{n,n}(x_1, \ldots, x_n) \triangleq \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} x_{\sigma(i)}.$$

Corollary 4.1. Over a $\mathbb{K}$ algebra containing the algebra of $n \times n$ matrices, $\text{snc-Sym}_{n,n}$ does not have polynomial size circuits unless Cayley-det is computable in polynomial time.
Proof. Proof is exactly the same as Theorem 4.1 by replacing nc-Sym\(_{n,n}\) with snc-Sym\(_{n,n}\).

Bounded Rank Permanent

Barvinok [6] showed that computing permanent of an integer matrix of constant rank can be done in strong polynomial time. In a similar spirit, we explore the complexity of computing the Cayley permanent of bounded rank matrices with entries from \(\mathbb{K} \cup \{x_1, \ldots, x_n\}\). For a matrix \(A\) with entries from \(\mathbb{K} \cup \{x_1, \ldots, x_n\}\), the row rank of the matrix is the rank of \(A\) over \(\mathbb{K}\). Meaning the smallest number \(k\) of rows \(A_{i_1}, \ldots, A_{i_k}\) such that every row \(A_j\) of \(A\) can be written as a \(\mathbb{K}\)-linear combination of \(A_{i_1}, \ldots, A_{i_k}\). The column rank of \(A\) is defined in an analogous manner. As opposed to the case of commutative permanent, For any algebra \(R\) containing the algebra of \(n \times n\) matrices over \(\mathbb{K}\), we have:

**Corollary 4.2.** Cayley-perm and Cayley-det of rank one matrices over \(\mathbb{K} \cup \{x_1, \ldots, x_n\}\) cannot have polynomial size arithmetic circuits unless \(\text{VNP} = \text{VP}\).

*Proof.* We will argue the case of Cayley-perm. Let \(x_1, \ldots, x_n\) be non-commuting variables. Consider the matrix \(A\) with \(A[i,j] = x_j, 1 \leq i, j \leq n\). \(A\) has rank one over \(\mathbb{K}\). We then have \(\text{nc-Sym}_{n,n}(x_1, \ldots, x_n) = \text{Cayley-perm}(A)\). The result now follows by applying Theorem 4.1. For Cayley-det, we can use Corollary 4.1 in place of Theorem 4.1 in the argument above.

5 Computational problems on non-commutative circuits

Computing Coefficients

In this section we consider various computational problems on arithmetic circuits, restricted to the non-commutative setting. We start with the problem of computing the coefficient of a given monomial in the polynomial computed by an arithmetic circuit. In the commutative setting, the problem lies in the second level of the counting hierarchy [15] and is known to be hard for \(\# P\) [18]. It was first seen in [2] that \(\text{mcoeff}\) is easy to compute in the non-commutative case. We provide a different proof of the fact as it is useful in the arguments used later in this section.

**Problem 1** (Monomial Coefficient(mcoeff)). Input: A non-commutative arithmetic circuit \(C\), a non-commutative monomial \(m\) of degree \(d\).
Output: The coefficient of monomial \(m\) in the polynomial computed by \(C\).

**Theorem 5.1.** \(\text{mcoeff}\) is in \(P\).

*Proof.* Suppose that the monomial \(m = x_{j_1} \cdots x_{j_d}\) and is given as an ordered listing of variables. Let \(f\) be a non-commutative polynomial. Then we have the following recursive
formulation for the coefficient function $mc : \mathbb{K}\{x_1, \ldots, x_n\} \times M \rightarrow \mathbb{K}$, where $M$ is the set of all non-commutative monomials in variables $\{x_1, \ldots, x_n\}$.

$$mc(f, m) = \begin{cases} 
\alpha & \text{if } f = \alpha y_j \text{ and } m = y_j, \\
0 & \text{if } f = \alpha y_j \text{ and } m = y_i, \ i \neq j, \\
mc(g, m) + mc(h) & \text{if } f = g + h, \\
\sum_{\ell=1}^{d+1} mc(g, m_{\ell}) \times mc(h, m'_{\ell}) & \text{if } f = g \times h.
\end{cases}$$

(1)

where $m_\ell = x_{i_{\ell}} \cdots x_{i_{\ell-1}}$ and $m'_\ell = x_{i_\ell} \cdots x_{i_d}$. However, if we apply the above recursive definition on the circuit $C$ in a straightforward fashion, the time required to compute $mc(f, m)$ will be $d^{O(\text{depth}(C))}$, since $\text{depth}(C)$ could be as big as $\text{size}(C)$, the running time would be exponential. However, we can have a more careful implementation of the above formulation by allowing a little more space.

For $\ell < k \in [1, d]$, let $m_{\ell,k} = x_{i_{\ell+1}} \cdots x_{i_k}$, and $M = \{m_{\ell,k} : 0 \leq \ell \leq d - 1, 0 \leq k \leq d\}$. Consider a gate $v$ in the circuit $C$. Note that in the process of computing $mc(f, m)$, we require only the values from the set $M_v = \{mc(p_v, m') : m' \in M\}$, where $p_v$ is the polynomial computed at $v$. Thus it is enough to compute and maintain the values $mc(p_v, m')$, $m' \in M$ in a bottom up fashion. For the base case, compute the values for polynomials computed at a leaf gate $v$ as follows, let $m' \in M$ and $\alpha \in R$

$$mc(p_v, m') = \begin{cases} 
\alpha & \text{if } p_v = \alpha, \text{ and } m' = \emptyset, \\
\alpha & \text{if } p_v = \alpha x_j, \alpha \in R, \text{ and } m' = x_j, \\
0 & \text{otherwise.}
\end{cases}$$

For other nodes, we can apply the recursive formula given in (1). If $p_v = p_{v_1} + p_{v_2}$, then the value $mc(p_v, m')$ can be computed using (1) as the values $mc(p_{v_1}, m')$ and $mc(p_{v_2}, m')$ are available by induction. If $p_v = p_{v_1} \times p_{v_2}$, then by induction, the values $mc(p_{v_1}, m''')$ and $mc(p_{v_2}, m''')$ are available for prefix and suffix of the monomial $m'$, as every such monomial occurs as $m_{i,j} \in M$ for some $i < j$. Now, $mc(p_v, m')$ can be computed by (1). For the space bound, the algorithm uses $O(d^2)$ registers for each gate in $C$ and hence the overall space used is $O(d^2 \text{size}(C))$ many registers. For a given monomial $m' \in M$, at most $d$ arithmetic operations are required in the worst case. Thus, the number of arithmetic operations is bounded by $O(d^3 \text{size}(C))$.

\[\square\]

Coefficient function as a polynomial

In the commutative setting, the coefficient function of a given polynomial can be represented as a polynomial [18]. Thus it is desirable to study the arithmetic circuit complexity of coefficient functions. However, over non-commutative rings, we need a carefully chosen representation of monomials to obtain an arithmetic circuit that computes the coefficient function for a given polynomial with small circuits. In the proof of Theorem [5.1], we have used an ordered listing of variables as a representation of the monomial $m$. Here we use a vector representation for non-commutative monomials of a given degree $d$. Let $Y = \{y_{1,1}, \ldots, y_{1,n}, y_{2,1}, \ldots y_{d,n}\}$ be a set of $nd$ distinct variables,
and let \( \tilde{Y}_i = (y_{i,1}, \ldots, y_{i,n}) \). The vector of variables \( \tilde{Y}_\ell \) can be seen as representing the characteristic vector of \( x_{i,\ell} \), i.e., \( y_{i,\ell} = 1 \), and \( y_{i,j} = 0, \forall j \neq i, \ell \). In essence, \( y_{i,j} \) stands for the variable \( x_j \) at the \( i \)-th position in the monomial. Let \( f(x_1, \ldots, x_n) \) be a polynomial of degree \( d \), then we can define the coefficient polynomial \( pc_f(Y) \) as

\[
pc_f(Y) = \sum_{\ell=1}^{d} \sum_{(i_1, \ldots, i_D) \in [n]^D} \prod_{\ell=1}^{D} \left[ mc(f, x_{i_1} \cdots x_{i_D}) y_{\ell,i} \prod_{j \neq k} (1 - y_{\ell,j} y_{\ell,k}) \right].
\]

**Theorem 5.2.** For any non-commutative polynomial \( f \) that can be computed by a polynomial size arithmetic circuit, \( pc_f(Y) \) has a polynomial size arithmetic circuit.

**Proof of Theorem 5.2.** We will apply (1) to obtain an arithmetic circuit computing the polynomial \( pc_f(Y) \). Let \( C \) be an arithmetic circuit of size \( s \), computing \( f \). By induction on the structure of \( C \), we construct a circuit \( C' \) for \( pc_f(Y) \). Note that, it is enough to compute homogeneous degree \( D \) components \( [pc_f(Y)]_D \) of \( pc_f(Y) \), where

\[
[pc_f(Y)]_D = \sum_{(i_1, \ldots, i_D) \in [n]^D} \prod_{\ell=1}^{D} \left[ mc(f, x_{i_1} \cdots x_{i_D}) y_{\ell,i} \prod_{j \neq k} (1 - y_{\ell,j} y_{\ell,k}) \right].
\]

Let \( Y^{i,j} \) denote the set of variables in the vectors \( \tilde{Y}_{i+1}, \ldots, \tilde{Y}_j \). In the base case, we have \( C = \gamma \in \{x_1, \ldots, x_n\} \cup R \). Then the all of the homogeneous components of \( pc_f(Y) \) can be described as follows.

\[
[pc_f(Y)]_0 = \begin{cases} 
\gamma & \text{if } Y = \emptyset \text{ and } \gamma \in R \\
0 & \text{otherwise}.
\end{cases}
\]

\[
[pc_f(Y)]_1 = \begin{cases} 
1 & \text{if } Y = e_j \text{ and } \gamma = x_j \\
0 & \text{if } Y = \emptyset \text{ and } \gamma \in R \\
0 & \text{otherwise}.
\end{cases}
\]

\[
[pc_f(Y)]_{i>1} = 0.
\]

Naturally, the induction step has two cases: \( f = g + h \) and \( f = g \cdot h \).

**Case 1:** \( f = g + h \), then for any \( D \)

\[
[pc_f(Y)]_D = [pc_g(Y)]_D + [pc_h(Y)]_D \quad \forall \ D.
\]

**Case 2:** \( f = g \times h \), then for any \( D \)

\[
[pc_f(Y)]_D = \sum_{i=0}^{d} \sum_{j=0}^{D} [pc_g(Y^{1,i})]_j [pc_h(Y^{i+1,j})]_{D-j}
\]

where \( Y = \tilde{y}_1, \ldots, \tilde{y}_d \). The size of the resulting circuit \( C' \) is \( O(d^3 \text{size}(C)) \), and \( C' \) can in fact be computed in time \( O(d^3 \text{size}(C)) \) given \( C \) as the input. \( \square \)
Partial Coefficient functions

For a given commutative polynomial let \( f(X) = \sum_m c_m m \), the partial coefficient of a given monomial \( m \) is a polynomial defined as \( \text{pcoeff}(f, m) = \sum_{m' \mid m} c_{m'} m' \).

We extend the above definition to the case of non-commutative polynomials as follows. Let \( f \) be non-commutative polynomial, and \( m \) a non-commutative monomial. Then \( \text{pcoeff}(f, m) = \sum_{m' = m \cdot m''} c_{m'} m'' \).

The corresponding computational problem can be defined in the following way.

**Problem 2** (Coefficient Polynomial (\( \text{pcoeff} \))). Input: A non-commutative arithmetic circuit \( C \) computing a polynomial \( f \), and a monomial \( m \).
Output: A non-commutative arithmetic circuit that computes \( \text{pcoeff}(f, m) \).

**Theorem 5.3.** \( \text{pcoeff} \) can be computed in deterministic time \( \text{poly}(\text{size}(C), n, \deg(m)) \).

**Proof.** The algorithm is similar to the proof of Theorem 5.1 except that we need to construct an arithmetic circuit rather than a value. We use the following recursive formulation similar to [1].

If \( f = \alpha \in R \cup \{ x_1, \ldots, x_n \} \) and \( m = \emptyset \) then \( \text{pcoeff}(f, m) = \alpha \). For the summation \( f = g + h \) we compute \( \text{pcoeff}(f, m) = \text{pcoeff}(g, m) + \text{pcoeff}(h, m) \). The final case to handle is a multiplication gate. We define shorthand for sets of variables. Let \( m = x_1 \cdots x_d \), \( m_i = x_1 \cdots x_i \) and \( m'_i = x_{i+1} \cdots x_d \) the rest of the monomial. We define \( m_0 = \emptyset \). Then \( \text{pcoeff}(f, m) = \sum_{i=1}^{d-1} m c_{m_i} \text{pcoeff}(h, m'_i) + \text{pcoeff}(g, m) \cdot \text{pcoeff}(f, \emptyset) \). The rest of the proof is analogous to that of Theorem 5.1 except that, we need to compute and store the values \( m c(p_v, m_{i,j}) \), and \( \text{pcoeff}(p_v, m_{i,j}) \) for every gate \( v \) in the circuit in a bottom up fashion.

---

**6 Conclusion and Open Questions**

Our study originated with the intention of obtaining special cases of matrices on which non-commutative permanent can be computed efficiently. However, our results indicate that non-commutative permanent is hard even for rank one matrices and weighted graphs of bounded component size. Further, existence of a natural non-commutative variant of elementary symmetric polynomial that is \#P hard to evaluate, shows that hardness of non-commutative permanent as not that surprising any more. We conclude with the following open questions

- Prove \#P hardness for non-commutative permanent on graphs of component size two?

- Is there a dichotomy for non-commutative permanent of graphs of component size two?

- Are there non-trivial special classes of matrices for which computing the Cayley permanent can be done efficiently?
Acknowledgement  The authors like to thank V. Arvind and Markus Bläser for helpful discussions and pointing out specific problems to work on. The authors also thank anonymous referees for their comments which helped in improving the presentation. This work was partially done while the first author was visiting IIT Madras sponsored by a grant of the Indo-Max-Planck Center for Computer Science (IMPECS).

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern approach*. Cambridge University Press, 2009.

[2] V. Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on non-commutative and commutative polynomial identity testing. CCC, pages 268–279, 2008.

[3] Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. Arithmetic circuits and the hadamard product of polynomials. In *FSTTCS*, pages 25–36, 2009.

[4] Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. In *STOC*, pages 677–686, 2010.

[5] Helmer Aslaksen. Quaternionic determinants. *The Mathematical Intelligencer*, 18(3):57–65, 1996.

[6] Alexander I. Barvinok. Two algorithmic results for the traveling salesman problem. *Mathematics of Operations Research*, 21(1):65–84, 1996.

[7] Markus Bläser. Noncommutativity makes determinants hard. In *ICALP (1)*, pages 172–183, 2013.

[8] Peter Bürgisser, Michael Clausen, and Amin Shokrollahi. *Algebraic Complexity Theory*. Grundlehren der mathematischen Wissenschaften. Springer Verlag, 1996.

[9] Peter Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7. Springer, 2000.

[10] S. Chien and A. Sinclair. Algebras with polynomial identities and computing the determinant. *SIAM Journal on Computing*, 37(1):252–266, 2007.

[11] Steve Chien, Prahladh Harsha, Alistair Sinclair, and Srikanth Srinivasan. Almost settling the hardness of noncommutative determinant. In *STOC*, pages 499–508, 2011.

[12] Samir Datta, Raghav Kulkarni, Nutan Limaye, and Meena Mahajan. Planarity, determinants, permanents, and (unique) matchings. *ToCT*, 1(3):10, 2010.
[13] Uffe Flarup, Pascal Koiran, and Laurent Lyaudet. On the expressive power of planar perfect matching and permanents of bounded treewidth matrices. In ISAAC, pages 124–136, 2007.

[14] Uffe Flarup and Laurent Lyaudet. On the expressive power of permanents and perfect matchings of matrices of bounded pathwidth/cliquewidth. Theory Comput. Syst., 46(4):761–791, 2010.

[15] Hervé Fournier, Guillaume Malod, and Stefan Mengel. Monomials in arithmetic circuits: Complete problems in the counting hierarchy. In STACS, pages 362–373, 2012.

[16] C. Gentry. Noncommutative determinant is hard: A simple proof using an extension of barrington’s theorem. In CCC, pages 181–187, June 2014.

[17] Meena Mahajan and B. V. Raghavendra Rao. Small space analogues of valiant’s classes and the limitations of skew formulas. Computational Complexity, 22(1):1–38, 2013.

[18] Guillaume Malod. The complexity of polynomials and their coefficient functions. In IEEE Conference on Computational Complexity, pages 193–204, 2007.

[19] Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In STOC, pages 410–418, 1991.

[20] Dan Roth. On the hardness of approximate reasoning. Artif. Intell., 82(1-2):273–302, 1996.

[21] Nitin Saxena. Progress on polynomial identity testing - ii. Electronic Colloquium on Computational Complexity (ECCC), 20:186, 2013.

[22] Edward R. Scheinerman. Random interval graphs. Combinatorica, 8(4):357–371, 1988.

[23] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. Foundations and Trends in Theoretical Computer Science, 5(3-4):207–388, 2010.

[24] L. G. Valiant. Completeness classes in algebra. STOC ’79, pages 249–261, 1979.

[25] L.G. Valiant. The complexity of computing the permanent. Theoretical Computer Science, 8(2):189 – 201, 1979.

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