Electronic Colloquium on Computational Complexity, Report No. 105 (2014)

Noncommutative Determinant is Hard: A Simple Proof Using an Extension of Barrington's Theorem

Craig Gentry
IBM T. J. Watson Research Center
Email: cbgentry@us.ibm.com

***Abstract*—We show that, for many noncommutative algebras A, the hardness of computing the determinant of matrices over A follows almost immediately from Barrington's Theorem. Barrington showed how to express a $\mathbf{NC}^1$ circuit as a product program over any non-solvable group. We construct a simple matrix directly from Barrington's product program whose determinant counts the number of solutions to the product program. This gives a simple proof that computing the determinant over algebras containing a non-solvable group is #P-hard or $\mathbf{Mod}_p$P-hard, depending on the characteristic of the algebra.**

To show that computing the determinant is hard over noncommutative matrix algebras whose group of units is solvable, we construct new product programs (in the spirit of Barrington) that can evaluate 3SAT formulas even though the algebra's group of units is solvable.

The hardness of noncommutative determinant is already known; it was recently proven by retooling Valiant's (rather complex) reduction of #3SAT to computing the permanent. Our emphasis here is on obtaining a conceptually simpler proof.

I. Introduction

The determinant and permanent have tantalizingly similar formulas:

$$\det(M) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i \in [n]} m_{i,\sigma(i)}, \ \text{per}(M) = \sum_{\sigma \in S_n} \prod_{i \in [n]} m_{i,\sigma(i)}$$

But the permanent seems much harder to compute than the determinant. We have efficient algorithms to compute the determinant over commutative algebras (such as finite fields), but computing the permanent even of 0/1 matrices is #P-complete [Val79]. Still, the tantalizing similarity of their formulas begs the question: to what extent can computing the permanent be reduced to a determinant computation?

One way to reduce permanent to determinant is to try to change the signs of M's entries so as to cancel the signs of the permutations. This approach works efficiently sometimes – most notably, when M is the adjacency matrix of a planar graph G [TF61], [Kas61], [Kas67], in which case $\text{per}(M)$ counts the number of perfect matchings in G, a counting problem #P-complete for general graphs.

A different way to reduce permanent to determinant is to consider the determinant of matrices whose entries come from a noncommutative (but associative) algebra, such as 2×2 matrices over a field, or the quaternion algebra over a field. In the noncommutative setting, the determinant is defined precisely as above, with the proviso that we must compute the products $\prod_{i \in [n]} m_{i,\sigma(i)}$ in order from $i = 1$ to n, since order of multiplication matters. The standard methods of computing the determinant break down when the underlying algebra is noncommutative. Elementary row operations no longer preserve M's determinant up to sign. (Interestingly, permuting M's columns *does* preserve the determinant up to sign, but elementary column operations in general do not.) Commutativity seems so essential to efficient computation of the determinant that it raises the question: can we show that computing the determinant over noncommutative commutative algebras is as hard as computing the permanent?

A. Recent Work on Noncommutative Determinant

Nisan [Nis91] gave the first result on noncommutative determinant, showing (among other things) that any arithmetic branching program (ABP) that computes the determinant of a matrix over the noncommutative free algebra $\mathbb{F}[x_{1,1}, \ldots, x_{n,n}]$ must have exponential size. Only recently have there been results showing the hardness of noncommutative determinant against models of computation stronger than ABPs.

In 2010, Arvind and Srinivasan [AS10] reduced computing the permanent to computing the determinant over certain noncommutative algebras – specifically, where the $m_{i,j}$'s are themselves linear-sized matrices. Chien, Harsha, Sinclair and Srinivasan [CHSS11] extended this result all the way down to 2×2 matrices. Specifically, they showed that computing the determinant when the entries themselves come from $M(2, \mathbb{F})$ – the matrix algebra of 2×2 matrices over field $\mathbb{F}$ – is as hard as computing the permanent over $\mathbb{F}$. This "almost settled" the hardness of noncommutative determinant, but left some cases open, such as the quaternion algebra. Bläser [Blä13] finally completed the picture, proving that computing the determinant is hard over (essentially) all noncommutative algebras.

The proof strategy followed by Chien et al. and Bläser is rather complex; it "works by retooling Valiant's original reduction from #3SAT to permanent" [CHSS11]. Valiant's reduction has complicated gadgets for variables, clauses, and XOR. Chien et al. describe their approach as follows: "when working with $M(2, \mathbb{F})$, what we show is that there is just enough noncommutative behavior in $M(2, \mathbb{F})$ to make Valiant's reduction (or a slight modification of it) go through"

ISSN 1433-8092

[CHSS11]. Their use of noncommutativity is subtle, buried in the modification of Valiant's reduction. While Chien et al. and Bläser settle the complexity of noncommutative determinant, we think that it is useful to have a conceptually simpler proof that better highlights how noncommutativity makes computing the determinant hard.

B. Our Results

We give a simple proof that noncommutative determinant is hard. that covers most noncommutative algebras of interest. For these algebras, we show that the hardness of noncommutative determinant follows almost immediately from Barrington's Theorem [Bar86].

Barrington [Bar86] is often cited for the proposition that width-5 branching programs (BPs) can efficiently compute NC^1 circuits. But he actually proved a much more general statement about the computational power of "non-solvable" non-abelian groups (we will define "non-solvable" later), of which the alternating group A_5 used in his width-5 BP is merely the smallest example. We build on Barrington's Theorem to show that noncommutative determinant is hard over algebras whose group of units is non-solvable. In our proof, Barrington does all of the heavy-lifting regarding noncommutativity, and we use his results as "black box".

Let us recall briefly what Barrington showed. Let G be a group. A *product program* P over G of length n that takes ℓ-bit inputs consists of two distinct "special" elements $a_0, a_1 \in G$ and a sequence of instructions $\langle \mathsf{inp}(i), a_{i,0}, a_{i,1} \rangle_{i\in[n]}$, where $\mathsf{inp} : [n] \to [\ell]$ indicates which input bit is read during a step, and each $a_{i,b} \in G$. For ℓ-bit input x, the value of $P(x)$ is simply $\prod_{i\in[n]} a_{i,x_{\mathsf{inp}(i)}}$. We say P "computes" a predicate F if $P(x) = a_{F(x)}$ for all x. Barrington showed that, if G is a fixed finite non-solvable group, then any circuit of depth d can be expressed as a *product program* over G of length $n = c_G^d$, where c_G is a constant that depends on G. If F can be computed by a circuit of logarithmic depth, then there is a product program that computes it efficiently. Barrington's result extends naturally to any algebra (such as a matrix ring) whose group of units is finite and non-solvable.

For any product program P over an algebra, we describe a simple *product program matrix* M_P whose determinant is $\sum_{x\in\{0,1\}^\ell} P(x)$, the sum of the evaluations of the product program. This determinant equals $\#\{F(x) = 0\} \cdot a_0 + \#\{F(x) = 1\} \cdot a_1$, and thus counts the number of satisfying assignments for F (up to the characteristic of the algebra). The product program matrix is about as simple as one could imagine. We put (up to sign) the the elements $a_{i,0}$ and $a_{i,1}$ of the product program into the i-th row. We need to be a little clever how we allocate these terms to the columns, to ensure that the products that are counted in the determinant are precisely those that have the form $\prod_{i\in[n]} a_{i,x_{\mathsf{inp}(i)}}$ for some x. This simple product program matrix is our main contribution.

We also consider the case where A's group of units is solvable. As a second contribution, we construct new product programs that are capable of evaluating d-CNFs over noncommutative algebras whose group of units is solvable (and conclude that computing the determinant of product program matrices over noncommutative algebras is hard).

Overall, our results cover all noncommutative algebras covered by Chien et al. [CHSS11] – in particular, algebras that contain a matrix subalgebra – but also algebras they did not cover, like the quaternion algebra. Our results have a gap: they do not necessarily cover all infinite noncommutative division algebras. (Noncommutative division algebras always have a non-solvable group of units, but Barrington's Theorem requires the group to be finite.) So, unlike [Blä13], our proof strategy does not provide the "complete picture".

But, again, we stress that this result is not new. Our contribution is a comparatively simple proof that may help us better understand how noncommutativity amplifies computational hardness.

II. Preliminaries

A. Algebras

Here, we present some essential facts about algebras, following the presentation in [Blä13].

When we refer to an *algebra* A, we mean an associative *algebra over a field* k, also known as a k-algebra. An algebra is a vector space over k, equipped with a bilinear mapping $\cdot : A \times A \to A$, called multiplication. Multiplication is associative and distributes over addition. The field k is in the center of A; it commutes with all elements of A, though A itself maybe noncommutative. We assume A is finite dimensional as a vector space, and contains an identity element '1'.

Some examples of noncommutative algebras over a field include 2×2 matrices and the quaternion algebra $\mathbb{H}$ over the reals. $\mathbb{H}$ contains elements of the form $q = a+b\mathbf{i}+c\mathbf{j}+d\mathbf{k}$ where $a, b, c, d \in \mathbb{R}$, and multiplication uses field multiplication and the relations $\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$, $\mathbf{jk} = \mathbf{i} = -\mathbf{kj}$, $\mathbf{ki} = \mathbf{j} = -\mathbf{ik}$.

A left *ideal* of A is a vector space that is closed under left multiplication with elements of A. (Right and two-sided ideals are defined analogously.) An ideal I is called *nilpotent* if some finite power of I is 0. The *radical* of A, denoted $\mathrm{Rad}(A)$, is the sum of all nilpotent left ideals; it is a maximal nilpotent two-sided ideal that also contains all of the nilpotent right ideals. A is called *semisimple* if $\mathrm{Rad}(A) = \{0\}$. The algebra $A/\mathrm{Rad}(A)$ is always semisimple.

A is called *simple* if 0 and A are its only two-sided ideals. An algebra D is called a *division algebra* if all of its nonzero elements are invertible. The quaternions $\mathbb{H}$ are an example of a division algebra. An algebra is called *local* if $A/\mathrm{Rad}(A)$ is a division algebra.

The Wedderburn-Artin Theorem gives a precise characterization of semisimple algebras.

Theorem 1 (Wedderburn-Artin)**.** Any finite dimensional semisimple algebra A is isomorphic to a finite direct sum $\oplus_i A_i$ of simple algebras. Any finite dimensional simple algebra A_i is isomorphic to $D_i^{n_i\times n_i}$, $n_i \geq 1$, where D_i is a division algebra. The pairs (D_i, n_i) are uniquely determined by A up to isomorphism.

We may refer to $D^{n\times n}$ as a *matrix algebra* when $n \geq 2$.

B. Groups

For an algebra A, we denote its group of units (invertible elements) by $A^\times$. Here, we present some essential facts about groups.

Let G be a group. G is *abelian* (or *commutative*) if $ab = ba$ for all $a, b \in G$; otherwise, G is nonabelian, or noncommutative. The commutator subgroup of G, denoted by G' or $[G, G]$, is the group generated by the set $\{aba^{-1}b^{-1} : a, b \in G\}$ of commutators in G. When G is abelian, G' is trivial. The commutator of two groups G_1, G_2, denoted by $[G_1, G_2]$, is the group generated by $\{aba^{-1}b^{-1} : a \in G_1, b \in G_2\}$.

The *lower central series* of G is given by $G, [G, G], [[G, G], G], [[[G, G], G], G], \ldots$. The group G is called *nilpotent* if this series reaches the trivial group after a finite number of steps. Every abelian group is nilpotent, but the converse is not true.

The *derived series* $G_0 = G, G_1, G_2, \ldots$ of a group is given by $G_{i+1} = [G_i, G_i]$. G is called *solvable* if this series reaches the trivial group after a finite number of steps. Otherwise, if the series stabilizes at a nontrivial group, G is called *non-solvable*. Every nilpotent group is solvable, but the converse is not true. An example of a group that is solvable but not nilpotent is S_3, the symmetric group on three elements. Its lower central series stabilizes at $[S_3, S_3] = A_3$, the alternating group on three elements, which is abelian (in fact, it is the cyclic group of order 3).

G is called a *perfect* group if $G = G'$. Every non-solvable group has a perfect subgroup – namely, the subgroup on which the derived series stabilizes. In some sense, perfect groups – and especially *simple* groups, a special case of perfect groups – are the "most non-abelian" groups.

Many natural groups are non-solvable, or even perfect. For example, the special linear group $\mathrm{SL}(n, k)$ of $n \times n$ matrices over k with determinant 1 is almost always a perfect group when $n \geq 2$; the exceptions are when $n = 2$ and k equals $\mathbb{F}_2$ or $\mathbb{F}_3$. When $\mathrm{SL}(n, k)$ is perfect, the general linear group $\mathrm{GL}(n, k)$ is of course non-solvable. The smallest non-solvable group is A_5, the alternating group on five elements.

Another class of non-solvable groups comes from division algebras.

Theorem 2 ([Hua49], [Hua50])**.** Let D be a noncommutative division algebra. Then the group of units of D is non-solvable.

For example, the group of units of the quaternion algebra $\mathbb{H}$ is non-solvable.

C. Branching Programs and Barrington's Theorem

Branching programs are a space-bounded model of computation, where at each step the program looks at just one bit of the input. More formally, a branching program is defined as follows.

Definition 1 (Branching Program)**.** A branching program (BP) P of width $w \geq 2$ and length n that takes ℓ-bit inputs is a sequence of instructions $\langle \mathsf{inp}(i), f_{i,0}, f_{i,1} \rangle_{i \in [n]}$ where $\mathsf{inp} : [n] \to [\ell]$ indicates which bit of the input is read during a step, each $f_{i,b}$ is a transition function from $\{0, \ldots, w-1\}$ to $\{0, \ldots, w-1\}$, and each $f_{n,b}$ maps into $\{0, 1\}$. For $x \in \{0,1\}^\ell$, the value of $P(x)$ is $f_{n,x_{\mathsf{inp}(n)}} \circ \cdots \circ f_{1,x_{\mathsf{inp}(1)}}(0) \in \{0, 1\}$. We say P computes a function $F : \{0,1\}^\ell \to \{0,1\}$ if $P(x) = F(x)$ for all x.

A *product program*, defined below, is similar to a branching program, but without the width constraint and where the transition function is defined by multiplication over some algebra or group.

Definition 2 (Product Program)**.** Let A be some algebra or group with associative multiplication. A product program (PP) P over A of length n that takes ℓ-bit inputs consists of two distinct elements $a_0, a_1 \in A$ and a sequence of instructions $\langle \mathsf{inp}(i), a_{i,0}, a_{i,1} \rangle_{i \in [n]}$, where $\mathsf{inp} : [n] \to [\ell]$ indicates which bit of the input is read during a step, and each $a_{i,b}$ is an element of A. For $x \in \{0,1\}^\ell$, the value of $P(x)$ is $\prod_{i \in [n]} a_{i,x_{\mathsf{inp}(i)}}$. We say P computes a function $F : \{0,1\}^\ell \to \{0,1\}$ if $P(x) = a_{F(x)}$ for all x.

One connection between BPs and PPs is the following. Let S_w be the permutation group over w elements. If there is a PP over S_w of length n that computes function $F : \{0,1\}^\ell \to \{0,1\}$, then there is a BP of width w and length n that computes F. (The transition function $f_{i,b}$ is just the permutation described by $a_{i,b}$.)

Barrington proved the following.

Theorem 3 (Barrington [Bar86])**.** Let G be a fixed finite non-solvable group and let F be a function that can be computed by a boolean circuit of depth d. Then, there is a PP over G of length at most c_G^d that computes F, where c_G is a constant that depends on G. The PP is computable in time linear in its length.

Corollary 1. For any fixed finite non-solvable group G, non-uniform NC^1 is inside the class of languages efficiently computable by PPs over G. In particular, since S_5 is non-solvable, non-uniform NC^1 is inside the class of languages efficiently computable by width-5 BPs.

The constant c_G in Theorem 3 can be taken to be $4 \cdot w_G$, where w_G is the the *commutator width* of a perfect subgroup of G. A perfect group is generated by its commutators, and the commutator width is the maximum number of (commutator) generators needed to express an element of the group. For simple groups, $w_G = 1$: every element can be expressed as a commutator [LOST10].

When G is an infinite group – e.g., the group of units of an algebra over an infinite field (such as $\mathbb{C}$) – there are complications. Most significantly, it is unclear how to efficiently construct a PP in general (in time polynomial in t_G^d for some constant t_G). So, our results for algebras with non-solvable unit groups G are limited to the case that G has *efficiently implementable PPs*, by which we mean G's to which Barrington's Theorem can be straightforwardly extended.

An example of an infinite group with efficiently implementable PPs is the group SO(3) of rotations in $\mathbb{R}^3$. This

group contains the finite polyhedral groups as subgroups – in particular, the icosahedral group of rotational symmetries of the regular dodecahedron and regular icosahedron, which is isomorphic to A_5. Once we find a A_5 PP for a function, we can easily generate a corresponding PP over SO(3). Precision issues cause some headaches, but the PP can still be evaluated to high precision in time polynomial in its length.

Another example of an infinite group with efficiently implementable PPs is the quaternion algebra $\mathbb{H}$ over the reals. We recall some more facts about quaternions. The conjugate of a quaternion $q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ is $\bar{q} = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$. The norm of q is $a^2+b^2+c^2+d^2 = q\cdot\bar{q}$. Norms are multiplicative, and the quaterions of norm 1 form a subgroup of invertible quaternions. For quaternions of norm 1, $q^{-1} = \bar{q}$. Pure quaternions have $a = 0$ and can be interpreted as points in $\mathbb{R}^3$. The map $\text{Rot}_q : \mathbb{R}^3 \to \mathbb{R}^3$ given by $p \to q\cdot p\cdot\bar{q}$ maps each pure quaternion p to another pure quaternion, and in fact induces a rotation of three-dimensional space. This map gives an onto homomorphism $\phi : \mathbb{H} \to$ SO(3), where two quaternions induce the same rotation if and only if they are $\mathbb{R}^\times$-multiples of each other. Given a PP $P = (a_0, a_1, \langle \mathsf{inp}(i), a_{i,0}, a_{i,1}\rangle_{i\in[n]})$ over SO(3), we can easily generate one for $\mathbb{H}$ by selecting some $p \in \mathbb{R}^3$ such that a_0 and a_1 send p to distinct points, finding norm-1 quaternions $q_0, q_1, \{q_{i,b}\}$ such that $\phi(q_0) = a_0$, $\phi(q_1) = a_1$, $\phi(q_{i,b}) = a_{i,b}$, and then setting the PP on input $x \in \{0,1\}^\ell$ to evaluate $(\prod_{i\in[n]} q_{i,x_{\mathsf{inp}(i)}})\cdot p\cdot(\prod_{i\in[n]} q_{i,x_{\mathsf{inp}(i)}})^{-1}$.

D. Counting Problems

#3SAT is the canonical #P-complete problem: given a formula in 3CNF, count the number of satisfying assignments. Counting the number of 3SAT solutions modulo p is the canonical Mod$_p$P-complete problem.

Let us relate these counting classes to product programs (from Section II-C). Suppose that we have a product program $P = (a_0, a_1, \langle \mathsf{inp}(i), a_{i,0}, a_{i,1}\rangle_{i\in[n]})$ over algebra A for some function $F : \{0,1\}^\ell \to \{0,1\}$. Here is a useful fact.

Fact 1. From the value $\sum_{x\in\{0,1\}^\ell} P(x)$, we can compute the number of satisfying assignments for F, up to the characteristic of the algebra A.

The reason is that the sum equals $N\cdot a_1 + (2^\ell - N)\cdot a_0 = N\cdot(a_1 - a_0) + 2^\ell\cdot a_0$, where N is the number of satisfying assignments for F. Since a_0 and a_1 are distinct, at least one coefficient of $a_1 - a_0$ (viewed as a vector over the field k) is nonzero, hence invertible, allowing us to recover N up to the characteristic of the field. Therefore, for product programs capable of evaluating a 3SAT formula, the value $\sum_{x\in\{0,1\}^\ell} P(x)$ must be hard to compute.

Theorem 4. Suppose that A is algebra for which there exist polynomial-length efficiently implementable product programs for computing 3SAT formulas. Then the problem of computing

$$\sum_{x\in\{0,1\}^\ell} P(x) = \sum_{x\in\{0,1\}^\ell} \prod_{i\in[n]} a_{i,x_{\mathsf{inp}(i)}} \tag{1}$$

for polynomial-length product programs over A is #P-hard or Mod$_p$P-hard, depending on the characteristic of the algebra A.

From Barrington's Theorem, we can immediately conclude the following.

Corollary 2. If A is an algebra whose group of units $A^\times$ is non-solvable (and has efficiently implementable PPs), then the problem of computing the value in Equation 1 is #P-hard or Mod$_p$P-hard, depending on the characteristic of the algebra A.

In Section III, we present a simple matrix whose determinant is precisely $\sum_{x\in\{0,1\}^\ell} P(x)$.

E. Noncommutative Determinant

Definition 3 (Determinant)**.** Let $M = (m_{i,j})$ be a $n\times n$ matrix over a (possibly noncommutative) algebra A. The determinant of M is defined by:

$$\det M = \sum_{\sigma\in S_n} \text{sgn}(\sigma)\cdot\prod_{i\in[n]} m_{i,\sigma(i)}.$$

The above definition, where multiplication is ordered by rows, is sometimes called the Cayley determinant. If A is noncommutative, order of multiplication matters, and the quantity may be different if multiplication were ordered by columns.

III. A Simple Matrix whose Determinant Counts Solutions

A. Intuition

As a warm-up, consider the following matrix, which has non-zero entries only on two diagonals (with wrap-around).

$$\begin{matrix} \star & \star & 0 & 0 & 0 \\ 0 & \star & \star & 0 & 0 \\ 0 & 0 & \star & \star & 0 \\ 0 & 0 & 0 & \star & \star \\ \star & 0 & 0 & 0 & \star \end{matrix}$$

We could not find a name for such matrices (a "bidiagonal" matrix has no wraparound), so let us invent a name:

Definition 4 (Barber pole matrix)**.** A matrix $M \in A^{n\times n}$ over algebra A is a barber pole matrix if $n \geq 2$ and $M[i,j]$ is nonzero only if $j = i$ or $j = i + 1 \bmod n$. M is a block barber pole matrix if it consists of barber pole matrices $M^{(1)}, \ldots, M^{(\ell)}$ of dimensions $n_1 + \cdots + n_\ell = n$ positioned in order (not interleaved) across the diagonal.

(In the real world, the stripes on a barber pole typically go instead from top-right to bottom-left, but barbers are accustomed to looking at the world through a mirror...)

Over commutative rings, the determinant of a barber pole matrix has a very simple form:

$$\det(M) \quad = \quad \prod_i m_{i,i} + (-1)^{n-1}\prod_i m_{i,i+1} \tag{2}$$

where $i+1$ is taken modulo n. The entries from the different "stripes" of the barber pole do not mix; rather, each product in the determinant summation is "consistent", taking entries from only one stripe. Equation 2 holds for noncommutative algebras as well, for noncommutative determinant as in Definition 3.

The (noncommutative) determinant of a block barber pole matrix is also fairly simple. For simplicity, suppose all block sizes n_k are odd, so that we don't need to deal with $(-1)^{n_k-1}$. Let $\mathsf{inp}(i) \in [\ell]$ denote the index of the block that the i-th row belongs to, and conversely let $C_k \subset [n]$ denote the subset of rows belonging to the k-th block. Then,

$$\det(M) = \prod_{k\in[\ell]} \left(\prod_{i\in C_k} m_{i,i} + \prod_{i\in C_k} m_{i,i+1} \right) \quad (3)$$

$$= \sum_{x\in\{0,1\}^\ell} \prod_{i\in[n]} m_{i,i+x_{\mathsf{inp}(i)}} \quad (4)$$

where $i+1$ wraps in the $\mathsf{inp}(i)$-th block. Note that the determinant of a block barber pole matrix is efficiently computable even in the noncommutative setting.

Consider what happens when we replace $m_{i,i+x_{\mathsf{inp}(i)}}$ notationally with $a_{i,x_{\mathsf{inp}(i)}}$. Then, the expression in Equation 4 looks exactly like the expression in Equation 1 that we want to compute! The only difference is that, in the products in Equation 4, the bits of x are used in order because the rows of the blocks are not interleaved, while in general a product program will not read the bits of x in order.

But now, suppose that we just permute the rows to conform to the order of our product program, and redefine $\mathsf{inp}(i)$ to be the function used by our product program. Then, we obtain a matrix whose determinant (up to sign) corresponds precisely to the expression in Equation 1, which allows us to derive the number of solutions to a product program over A. To correct the sign, we can apply the same permutation to the columns.

Interestingly, the transition from an easy determinant (of the block barber pole matrix) to a hard one (for the product program matrix) is easy to identity. It occurs precisely when we permute the rows of the matrix, an operation that fails to preserve determinant (up to sign) in the noncommutative setting.

B. Formal Details

Let $P = (a_0, a_1, \langle \mathsf{inp}(i), a_{i,0}, a_{i,1}\rangle)$ be a product program over A of length n that takes ℓ-bit inputs. For $k \in [\ell]$, let $C_k = \{i \in [n] : \mathsf{inp}(i) = k\}$, with elements denoted $i_{k,1}, \ldots, i_{k,|C_k|}$. Assume $|C_k| \geq 2$ for all k. (If necessary, we can easily modify P to meet this condition by inserting "dummy" elements.) Let π_0 be the identity permutation: $\pi_0(i) = i$ for all $i \in [n]$. Let π_1 be the "shift right" permutation, given by $\pi_1(i_{k,t}) = i_{k,t+1}$ where $t+1$ is computed modulo $|C_k|$, that "cycles" elements within each C_k. Let M_P be the following *product program matrix* for P:

$$M_P[i,j] = \begin{cases} (-1)^{|C_k|-1} \cdot a_{i,1} & \text{if } i = i_{k,1}, j = i_{k,2} \text{ for some } k \\ a_{i,b} & \text{otherwise if } j = \pi_b(i) \\ 0 & \text{otherwise} \end{cases}$$

Clearly, the product program matrix is well-defined and efficiently computable from the product program. Now, we prove that M_P has a determinant equal to the expression in Equation 1.

Theorem 5. The (noncommutative) determinant of the product program matrix M_P for P as described above is $\sum_{x\in\{0,1\}^\ell} \prod_{i\in[n]} a_{i,x_{\mathsf{inp}(i)}}$.

Proof: The (noncommutative) determinant of M_P is $\sum_{\sigma\in S_n} \mathrm{sgn}(\sigma) \cdot \prod_{i\in[n]} M_P[i,\sigma(i)]$. We call σ a "consistent" permutation if there exists $x \in \{0,1\}^\ell$ such that $\sigma(i) = \pi_{x_{\mathsf{inp}(i)}}(i)$ for all i. We denote by σ_x the consistent permutation associated to x.

Suppose $\prod_{i\in[n]} M_P[i,\sigma(i)]$ is nonzero, but σ is not consistent. Since the product is nonzero, $\sigma(i) \in \{\pi_0(i), \pi_1(i)\}$ for all i. But since it is inconsistent, there exists $k \in [\ell]$ such that σ is inconsistent over C_k – i.e., σ equals neither π_0 nor π_1 when restricted to C_k. For this k, there must exist some t such that $\sigma(i_{k,t}) = \pi_1(i_{k,t}) = i_{k,t+1}$ and $\sigma(i_{k,t+1}) = \pi_0(i_{k,t+1}) = i_{k,t+1}$, contradicting the fact that σ is a permutation. We can therefore restrict the summation to consistent permutations: $\det(M_P) = \sum_{x\in\{0,1\}^\ell} \mathrm{sgn}(\sigma_x) \cdot \prod_{i\in[n]} M_P[i,\sigma_x(i)]$.

The sign of σ_{0^ℓ} is 1, and the sign of σ_x is $(-1)^{\sum x_k(|C_k|-1)}$, since it takes $|C_k|-1$ transpositions to "shift right" $|C_k|$ elements. On the other hand, we have $\prod_{i\in[n]} M_P[i,\sigma_x(i)] = (-1)^{\sum x_k(|C_k|-1)} \cdot \prod_{i\in[n]} a_{i,x_{\mathsf{inp}}(i)}$. So, $\mathrm{sgn}(\sigma_x) \cdot \prod_{i\in[n]} M_P[i,\sigma_x(i)]$ equals $\prod_{i\in[n]} a_{i,x_{\mathsf{inp}}(i)}$, completing the proof. ∎

Corollary 3. If A is an algebra whose group of units $A^\times$ is non-solvable (and has efficiently implementable PPs), then the problem of computing the determinant of the product program matrix M_P over A is is #P-hard or Mod_pP-hard, depending on the characteristic of the algebra A.

Proof: By Corollary 2 and Theorem 5. ∎

IV. Noncommutative Algebras with Solvable Group of Units

Here, we use product programs matrices to show that computing the determinant is hard over all noncommutative algebras A that have a *solvable* group of units. First, we show that all such algebras contain a matrix algebra. Then, we build new product programs that can evaluate d-CNF formulas over any matrix algebra A. All of the PPs here are efficiently implementable: we show precisely how to construct them using 2×2 matrices with entries in $\{-1, 0, 1\}$. Our PPs make essential use of non-invertible elements in A. Finally, we invoke Theorems 4 and 5 to prove that computing the product program matrix M_P over A is hard.

To begin, note that given an algebra A, we can restrict our attention to the algebra $A/\mathrm{Rad}(A)$, the semisimple part of A. The Wedderburn-Artin Theorem (Theorem 1) says that $A/\mathrm{Rad}(A)$ decomposes as $\oplus A_i$ where A_i is isomorphic to $D_i^{n_i\times n_i}$ and the D_i's are division algebras. If $A/\mathrm{Rad}(A)$ is noncommutative and has a solvable group of units, then for some i it must hold that $D_i^{n_i\times n_i}$ is noncommutative and has a solvable group of units. The following lemma says this can only happen if $n_i \geq 2$, when when $D_i^{n_i\times n_i}$ is a matrix algebra.

Lemma 1. Let D be a division algebra. Suppose $D^{n\times n}$ is noncommutative and its unit group is solvable. Then $n \geq 2$.

Proof: Assume D is a *noncommutative* division algebra. Then, D has a non-solvable group of units [Hua49], [Hua50]. This implies that the unit group of $D^{n\times n}$ is also non-solvable, since it contains D as a subalgebra.

So, D must be commutative. But since $D^{n\times n}$ is noncommutative, we must have $n \geq 2$. ∎

Now we state our main results for this Section.

Theorem 6. For any division algebra D and any constant d, one can construct a product program of length $k(2^d+2^{d-1}-2)$ for d-CNF with k clauses over the algebra $D^{2\times 2}$.

From Theorem 1, Theorem 4, Theorem 5, Lemma 1 and Theorem 6, we obtain the following corollary.

Corollary 4. Let A be a noncommutative algebra with a solvable group of units. Then, for any constant d, one can construct a product program over A of length poly(k) for a d-CNF with k clauses. If p is the characteristic of A, then computing the determinant over A is #P-hard if $p = 0$ and Mod$_p$P-hard otherwise.

Before we prove Theorem 6, we need a couple of lemmas.

Lemma 2. For any division algebra D, the group of units of $D^{2\times 2}$ contains a subgroup isomorphic to S_3. In particular, $D^{2\times 2}$ contains the matrices

$$r = \begin{matrix} 0 & -1 \\ 1 & -1 \end{matrix}, \qquad s = \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix},$$

that generate a group isomorphic to S_3.

Proof: (Lemma 2) Clearly $D^{2\times 2}$ contains the matrices r and s (though -1 will equal 1 if the algebra has characteristic 2). The element r has order 3 and s has order 2, and they do not commute, regardless of what D is. The group they generate has order at least 6 by Lagrange's Theorem. Using the relation $rs = sr^2$ and $r^2s = sr$ (together with the relations $r^3 = 1 = s^2$), we can re-express any word over r and s as r^is^j with $i \in \{0,1,2\}, j \in \{0,1\}$, and so the group has order exactly 6. S_3 is the only non-abelian group of order 6. ∎

Lemma 3. One can construct a product program of length $2^d + 2^{d-1} - 2$ over the group S_3 that computes a disjunction of d literals. Furthermore, in the product program, we can have $a_0 = r$ (the element of order 3 from Lemma 2) and $a_1 = 1$.

Remark 1. Lemma 3 can be generalized beyond S_3 to obtain product programs for a disjunction of d literals over any group that is not nilpotent. Since we do not need this fact here, we omit the proof.

Proof: (Lemma 3) For $d = 1$, the lemma holds for a product program of length 1 with $a_{1,0} = a_0$ and $a_{1,1} = a_1$. Assume the lemma is true for $d-1$, and let P_{d-1} be the associated product program. Let $b_0 = s$ (the element of order 2 from Lemma 2) and $b_1 = 1$. For $x \in \{0,1\}^d$, let x' denote the first $d-1$ bits of x. Our product program P_d for d literals computes

$$P_d(x) = b_{x_d} \cdot P_{d-1}(x') \cdot b_{x_d} \cdot P_{d-1}(x')^{-1}.$$

(We avoid writing out P_d formally.) If any bit in x is a 1, then either $P_{d-1}(x') = 1$ or $b_{x_d} = 1$, in which case $P_d(x) = 1$. If all bits in x are 0, then $P_{d-1}(x') = r$ and $b_{x_d} = s$, and $P_d(x) = srsr^{-1} = r$. The length of P_d is twice the length of P_{d-1} plus 2, putting the length at $2^d + 2^{d-1} - 2$. ∎

At this point, we have constructed a product program over any matrix algebra for a disjunction of d literals, where the product program outputs the identity matrix I if the disjunction is satisfied, and the matrix r otherwise. Now, we need to compute a conjunction over the disjunctions. Roughly speaking, we accomplish this by picking a singular 2×2 matrix t such that $t \cdot I$ has the vector $e_1 = (1,0)^T$ as an eigenvector with eigenvalue 1, while $t \cdot r$ has e_1 as an eigenvector with eigenvalue 0. Our product program for d-CNF computes a product that still has e_1 as an eigenvector, where the eigenvalues "multiply through" – in particular, the eigenvalue is 0 precisely when a $(t \cdot r)$ term is included in the product, which happens precisely when one of the disjunctions is not satisfied.

Proof: (Theorem 6) By Lemma 2, for any division algebra D, the algebra $D^{2\times 2}$ contains a subgroup isomorphic to S_3, generated by matrices r and s as defined above. By Lemma 3, there is a product program P_{clause} over S_3 that correctly evaluates a disjunction of d literals, in the sense that for all $x \in \{0,1\}^d$, $P_{clause}(x) = a_{\text{OR}(x)}$, where $a_0 = r$ and $a_1 = I$. We now construct a product program P over $D^{2\times 2}$ for d-CNF with k clauses, using the product program P_{clause} over $D^{2\times 2}$ for a disjunction of d literals. For input x to P, let x_c be the d literals of x that are used in the c-th clause. Let $t \in D^{2\times 2}$ be the matrix with 1 in the upper-left corner and zeros elsewhere. Our product program P computes:

$$P(x) = \left(\prod_{c\in[k]} t \cdot P_{clause}(x_c) \right) \cdot t$$

If all disjunctions are satisfied, then $P(x) = (t \cdot I)^k \cdot t = t$.

Suppose one of the disjunctions is unsatisfied. Then the expression above contains a term $t \cdot a_0$, which is a matrix with zeros everywhere but the upper-right corner. Regardless of whether or not the other disjunctions are satisfied, the expression above before the final t is always a matrix with zeros everywhere but the upper-right corner. Multiplication with the final t results in an all-zero matrix.

To recap, $P(x) = 0$ when x does not satisfy the d-CNF, and $P(x) = t$ otherwise. The length of P is k times the length of P_{clause}. (The t's are folded into neighboring terms.) ∎

References

[AS10] Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. In Leonard J. Schulman, editor, *STOC*, pages 677–686. ACM, 2010.

[Bar86] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. In Juris Hartmanis, editor, *STOC*, pages 1–5. ACM, 1986.

[Blä13] Markus Bläser. Noncommutativity makes determinants hard. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2013.

[CHSS11] Steve Chien, Prahladh Harsha, Alistair Sinclair, and Srikanth Srinivasan. Almost settling the hardness of noncommutative determinant. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 499–508. ACM, 2011.

[Hua49] Loo-Keng Hua. Some properties of a sfield. *Proc. Nat. Acad. Sci. U.S.A.*, 35:533–537, 1949.

[Hua50] Loo-Keng Hua. On the multiplicative group of a sfield. *Acad. Sinica Sci. Record*, 3:1–6, 1950.

[Kas61] Pieter Kasteleyn. The statistics of dimers on a lattice. *Physica*, 27:1209–1225, 1961.

[Kas67] Pieter Kasteleyn. Graph theory and crystal physics. *Graph Theory and Theoretical Physics*, pages 43–110, 1967.

[LOST10] Martin W Liebeck, Eamonn A OBrien, Aner Shalev, and Pham Huu Tiep. The ore conjecture. *J. Eur. Math. Soc.(JEMS)*, 12(4):939–1008, 2010.

[Nis91] Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *STOC*, pages 410–418. ACM, 1991.

[TF61] Harold Neville Vazeille Temperley and Michael Ellis Fisher. Dimer problem in statistical mechanics - an exact result. *Philosophical Magazine*, 6:1061–1063, 1961.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de