New directions in Arithmetic and Boolean Circuit Complexity

by

Srikanth Srinivasan

The Institute of Mathematical Sciences, Chennai.

A thesis submitted to the

Board of Studies for Mathematical Sciences

In partial fulfilment of requirements

For the degree of

DOCTOR OF PHILOSOPHY

of

Homi Bhabha National Institute

Acknowledgements

Firstly, I wish to thank my advisor V Arvind for his advice, encouragement, and understanding. Arvind taught me Complexity Theory during my second semester at IMSc, and I found the subject captivating. Research with him was only more so, both because of his choice of problems and his way of attacking them. His breadth of knowledge of Mathematics and Computer Science has inspired me to broaden my own horizons. I could not have asked for a better advisor.

Secondly, I would like to thank R Ramanujam for being a constant source of encouragement. I may not have had the opportunity to study at IMSc had it not been for him; and I definitely would not have read (and thoroughly enjoyed) and thought about Logic in Computer Science and Mathematics had I not attended his wonderful classes on the subject.

I would also like to thank Meena Mahajan for her boundless enthusiasm in organizing seminars, for the excellent talks that she gave in these seminars (something I have tried countless times to emulate), and for all her help and advice. Let me also thank the other faculty members at IMSc, all of whom I have learnt a great deal from.

I have also been fortunate enough to visit other academic and research institutes in and outside India. I would like to thank Manindra Agrawal for inviting me to visit IIT Kanpur; Jaikumar Radhakrishnan and Prahladh Harsha for inviting me to visit TIFR; Satya Lokam for sponsoring a visit to Microsoft Research, Bangalore; Peter Bro Miltersen for inviting me to visit the University of Aarhus; and Steve Chien for inviting me to visit Microsoft Research, Silicon Valley. I have thoroughly enjoyed and learned a great deal during all these visits.

Finally, I would like to thank IMSc for providing me with excellent facilities during the time that I have spent here.

Contents

Chapter 1

Introduction

The aim of Theoretical Computer Science in general and Complexity Theory in particular is to estimate the amount of computational resources — time, space, randomness, etc. — required to solve natural computational problems. Broadly speaking, this overarching goal can be divided into two subproblems: the first of these is to obtain more resource-efficient algorithms to solve computational problems, and the second is to understand what makes a problem hard to solve and to prove limits on what efficient computation can accomplish. We will be interested mainly in the latter; a research program that is vaguely referred to as *lower bounds*.

Proving lower bounds has been a notoriously hard problem for Theoretical Computer Scientists. Despite much effort over the past four decades or so, most of the fundamental questions we started out with remain unanswered, and worse, we don't even seem to have the techniques to tackle them. However, during this time, researchers have classified these questions into varying degrees of difficulty, eliminated various techniques that are not going to work, identified various barriers to proving strong lower bounds, and come up with easier questions that must be answered before we can handle the more fundamental ones.

The last of these lines of research has led to many theorems regarding lower bounds in restricted models of computation. The purpose of this thesis is to supplement these efforts: namely, to point out some interesting new directions for lower bounds, and take some steps towards resolving these questions.

Specifically, we study the question of proving lower bounds for constant-depth Boolean circuits with *help functions* and noncommutative Algebraic Branching Programs with *help polynomials*; of proving lower bounds for monotone arithmetic circuits of *bounded width*; and of proving lower bounds on the size of *noncommutative* arithmetic circuits computing the noncommutative determinant. Below, we introduce these problems in greater detail, and state our results.

1.1 Boolean circuits

Boolean circuits provide a mathematical model of the circuitry inside modern computers. They compute Boolean functions of their 0-1 inputs using Boolean operations such as AND, OR, and NOT. The formal definition follows.

Definition 1.1.1. *A* Boolean circuit C *over* n *Boolean variables* $X = \{x_1, x_2, \ldots, x_n\}$ *is a labelled directed acyclic graph such that:*

- *The internal nodes of* C *are labelled by* $\wedge$ *and* $\vee$ *— these are called the AND and OR gates respectively — or constants from* $\{0, 1\}$*, and the leaves are labelled by Boolean literals over* X*, which are elements from the set* $\{x_1, \overline{x_i} \mid i \in [n]\}$*.*
- C *has a designated output node* v*.*

Each node computes a Boolean function of the n *input variables in a natural way. A leaf node labelled by* x_i *or* $\overline{x_i}$ *just computes the projection onto the* i*th variable or its negation, and a leaf node labelled by a constant just computes that constant function. The internal AND and OR gates just compute the AND and OR of the functions computed by their children. The Boolean function computed by* C *is defined to be the function computed by the output node* v*.*

The size *of* C *— denoted* $\text{size}(C)$ *— is defined to be the number of vertices in* C *and the* depth *of* C *— denoted* $\text{depth}(C)$ *— is defined to be the length of the longest directed path in* C*.*

A Boolean circuit over n variables computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$. To compute a Boolean function $F : \{0,1\}^* \to \{0,1\}$ defined on all input lengths, we use a *Circuit Family*, which is defined to be a collection of Boolean circuits $\{C_n \mid n \in \mathbb{N}\}$ such that C_n computes a Boolean function on $\{0,1\}^n$; the circuit family $\{C_n \mid n \in \mathbb{N}\}$ is said to compute F if C_n computes the function F restricted to $\{0,1\}^n$.

We normally measure the efficiency of (a family of) Boolean circuits by their size. Given a function $s : \mathbb{N} \to \mathbb{N}$, we use $\text{Size}(s(n))$ to denote the class of Boolean functions $F : \{0,1\}^* \to \{0,1\}$ that can computed by a family of circuits $\{C_n \mid n \in \mathbb{N}\}$ such that $\text{size}(C_n) \leq s(n)$. We also use $\text{Size}(n^{O(1)})$ to denote $\bigcup_{c>0} \text{Size}(cn^c)$. It is a standard fact that any function $F : \{0,1\}^* \to \{0,1\}$ that lies in P can be computed by a polynomial-sized circuit family. Hence, $\text{P} \subseteq \text{Size}(n^{O(1)})$.

It will also be useful to restrict circuits by depth. Given functions $s, d : \mathbb{N} \to \mathbb{N}$, we use $\text{SizeDepth}(s(n), d(n))$ to denote the class of functions that can be computed by circuits of size $s(n)$ and depth $d(n)$.

1.1.1 Boolean circuit lower bounds

The primary aim of the area of Boolean circuit complexity is to obtain *explicit* functions that cannot be computed by *small* circuit families. Here, "explicit" usually means that the function $F : \{0,1\}^* \to \{0,1\}$ itself can be computed in a somewhat restricted complexity class, such as NP or EXP; and "small" usually refers to circuit families of size polynomial or subexponential in the input length.

Both the above conditions are required to make the problem non-trivial. Explicitness is key since it is known, by a counting argument, that there exist Boolean functions $F : \{0,1\}^* \to \{0,1\}$ that cannot be computed by circuit families of size $\Omega(2^n/n)$ [Sha49]; this also easily implies that there are functions $F : \{0,1\}^* \to \{0,1\}$ computable in EEXP and in ESPACE that do not have polynomial-sized circuits, which is why we choose an explicit lower bound to be one computable in NP or in EXP. It is also known [Sha49] that every Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit of size $O(2^n/n)$, and hence we can only hope to prove lower bounds against circuit families of size smaller than this.

Sufficiently strong explicit lower bounds for Boolean circuits would have many repercussions in Complexity Theory. If there exists an explicit function in NP that cannot be computed by a polynomial-sized circuit family, then NP $\not\subseteq$ Size($n^{O(1)}$) and since P $\subseteq$ Size($n^{O(1)}$), this proves that P $\neq$ NP, thus resolving a preeminent problem in Complexity Theory. It is also known [IW97] that if there are explicit functions in E (the subclass of functions in EXP that can be computed in time $2^{O(n)}$) that do not have circuits of size $2^{\varepsilon n}$ for some fixed $\varepsilon > 0$, then P = BPP, and hence allowing algorithms to use randomness does not result in superpolynomial speedups.

Despite much effort over the past three decades or so, we do not have any non-trivial explicit lower bounds for general Boolean circuits. It is easy to show that for any fixed $c > 0$, EXP contains a function $F \notin$ Size(n^c); however, we do not know if EXP contains a function $F \notin$ Size($n^{O(1)}$). For NP, the situation is even worse: we do not know of a superlinear lower bound for any function in NP; the best known explicit lower bound for any function in NP is $5n - o(n)$ [LR01, IM02] — this function is in fact in P. However, progress has been made for restricted classes of Boolean circuits. We discuss a small sample of such results below.

A Boolean circuit is said to be *monotone* if it does not use any negative literals (i.e literals of the form $\overline{x_i}$ for some i). Such a circuit clearly computes a monotone function, i.e, a Boolean function $f : \{0,1\}^n \to \{0,1\}$ such that for any x, y with $x_i \leq y_i$ for each i, $f(x) \leq f(y)$. In 1985, Razborov [Raz85] showed that any monotone circuit computing the monotone Perfect Matching function (checking if an input graph has a perfect matching) or the Clique function (checking if an input graph has a clique of a certain size) on graphs with n vertices must have size $n^{\Omega(\log n)}$. Following this, Alon and Boppana strengthened the lower bound for Clique to $2^{n^{\Omega(1)}}$. Tardos [Tar88] showed such a lower bound for a monotone function in P.

Researchers have also considered restrictions on the depth of Boolean circuits. In the early

1980s, Furst, Saxe, and Sipser [FSS84] and Ajtai [Ajt83] independently showed that the Parity function cannot be computed by polynomial-sized constant-depth Boolean circuits: that is, Parity $\notin \text{SizeDepth}(n^{O(1)}, O(1))$. This was improved to an exponential lower bound by Yao [Yao85] and further strengthened and simplified by Håstad [Hås89] to show a near-optimal lower bound: Parity $\notin \text{SizeDepth}(2^{\Omega(n^{1/d-1})}, d)$ for any constant $d \in \mathbb{N}$.

These results were followed by investigations into the power of constant-depth Boolean circuits that were allowed to use Parity and other gates in addition to AND and OR gates. Razborov [Raz87] showed that even with Parity gates, constant-depth circuits of subexponential size cannot compute the Majority function. Smolensky [Smo87] considered more general Boolean circuits that use Mod_p gates (a Mod_p gate outputs 1 if and only if the number of 1s in its input is not divisible by p. Thus, a Parity gate is just a Mod_2 gate) for a prime p and showed that for any distinct primes p and q, the function Mod_q cannot be computed by subexponential-sized constant-depth Boolean circuits with Mod_p gates.

However, since the late 80s and early 90s, the flood of lower bound results in Boolean circuit complexity has abated. The lower bounds above have not been significantly strengthened to more general circuits: to give the reader an idea of how far we are from proving lower bounds on general Boolean circuits, it is not yet known if NP contains a language that cannot be computed by an $O(n)$-sized family of constant-depth Boolean circuits augmented with Mod_6 gates; it is also not known if there is a language in EXP that cannot be computed by such a circuit family of polynomial size. It is suspected, however, that to significantly improve the state of lower bounds today, new techniques are needed [RR97]. Currently, much research in Boolean circuit complexity is concentrated on very special classes of circuits where known techniques fail to give lower bounds (see, for example, [KP97, GHR92, BS99, BM99, AB01, CH05, Bou05, CW09]).

1.1.2 Our results

We consider a different way of extending known lower bounds on constant-depth circuits that we call the *Help functions problem*. Fix an input length $n \in \mathbb{N}$ and an arbitrary collection of Boolean functions $H = \{h_i : \{0,1\}^n \to \{0,1\} \mid i \in [m]\}$ that we will refer to as the *help functions*. We wish to prove explicit lower bounds for the sizes of constant-depth circuits that are given the values of $h_1, h_2, \ldots, h_m$ at the input x for "free".

More formally, we consider constant-depth Boolean circuits such that their leaves are labelled with functions from H and constants from the set $\{0,1\}$. The function computed by such a circuit is defined in the natural way: on an input x, a leaf labelled by a function h computes $h(x)$ and a leaf labelled by a constant computes that constant; the inductive definition of the functions computed by the internal nodes remains unchanged.

Given $s, d \in \mathbb{N}$, let us denote by $\text{SizeDepth}_H^n(s, d)$ to be the class of Boolean functions

$g : \{0,1\}^n \to \{0,1\}$ that can be computed by such a constant-depth Boolean circuit of size at most s and depth at most d. Our objective is to find an explicit function $f : \{0,1\}^n \to \{0,1\}$ such that $f \notin \text{SizeDepth}^n_H(s,d)$. Notice that the function f *must* depend on the choice of the help functions H; otherwise, H may contain the function f, in which case $f \in \text{SizeDepth}^n_H(1,0)$ trivially. Hence, we phrase the help functions problem as an algorithmic question, as follows:

For functions $m, s, d : \mathbb{N} \to \mathbb{N}$, the $(m(n), s(n), d(n))$-Help function problem is defined as follows:

- INPUT: A collection of truth-tables of m functions $h_1, h_2, \ldots, h_m : \{0,1\}^n \to \{0,1\}$.
- DESIRED OUTPUT: A truth-table of a function $f : \{0,1\}^n \to \{0,1\}$ such that $f \notin \text{SizeDepth}^n_H(s(n), d(n))$.

Note that the size of the input in the above problem is $m \cdot 2^n$. Hence, we would want an algorithm for the above problem that runs in time polynomial in this quantity. We are interested mainly in the case when $s(n)$ and $m(n)$ are slightly superpolynomial in n, and $d(n)$ is a constant.

The interest in this problem is that solving it efficiently would have several implications for standard complexity classes. A polynomial-time algorithm for the above problem would imply that EXP does not polynomial-time reduce to the very small complexity class $\text{SizeDepth}(n^{O(1)}, O(1))$. A uniform NC solution for this problem would imply that PSPACE does not logspace reduce to $\text{SizeDepth}(n^{O(1)}, O(1))$. The above are special cases of the conjectured separations $\text{EXP} \not\subseteq \text{Size}(n^{O(1)})$ and $\text{PSPACE} \not\subseteq \text{Size}(n^{O(1)})$ respectively.

We are not the first to study this problem. Jin-yi Cai [Cai90] studied the power of constant-depth circuits augmented with a few help functions. However, his results do not apply in our setting where we allow superpolynomially many help functions. He also does not make the above connection to outstanding problems in Complexity Theory when the regime of superpolynomially many help functions is considered.

We connect the study of the Help functions problem to a very interesting problem in the literature called the *Remote Point Problem* (RPP). This problem was introduced by Alon, Panigrahy, and Yekhanin [APY09], who studied it as a restriction of the question of constructing *Rigid Matrices*.

An $m \times n$ matrix M over a field $\mathbb{F}$ is said to be (r,k)-rigid if one cannot reduce the rank of M to k or less by modifying at most r entries in each row. It is easy to see that this is equivalent to the following: given any subspace V of $\mathbb{F}^n$ of dimension at most k, one of the rows of M is at Hamming distance at least $r+1$ from V. Explicit constructions of rigid matrices would imply lower bounds for circuits of certain kinds [Val77]. However, constructing such matrices has been an open problem for a long time. Alon et al. [APY09] suggest the RPP, stated below, as a problem of intermediate difficulty.

Fix $N \in \mathbb{N}$ and functions $k, r : \mathbb{N} \to \mathbb{N}$. The $(k(N), r(N))$-RPP is defined as follows:

- INPUT: A subspace V of $\mathbb{F}_2^N$ of dimension at most $k(N)$ given by its generators.
- DESIRED OUTPUT: A point $x \in \mathbb{F}_2^N$ such that the Hamming distance of x from V is at least $r(N)$.

Note that the RPP does indeed seem to be an easier problem than constructing rigid matrices, since one now has access to the subspace V beforehand. More formally, the problem of constructing a rigid matrix may be phrased in terms of constructing efficient *hitting sets* for the RPP.

We show that an efficient solution to the $(k(N), r(N))$-RPP, for suitable $k(\cdot)$ and $r(\cdot)$, implies an efficient solution to the help functions problem in the parameters of interest. It can be show that for this choice of $k(\cdot)$ and $r(\cdot)$, a solution to the RPP always exists: in fact, a random vector in $\mathbb{F}_2^N$ is a solution with high probability. However, we need a deterministic solution.

Result 1. *For any constant depth d, there are universal constants c_0, c_1 such that if there is a polynomial-time (respectively uniform* NC*) solution to the $(k(N), r(N))$-RPP with $k(N) = 2^{(\log\log N)^{c_0}}$ and $r(N) = \frac{N}{2^{(\log\log N)^{c_1}}}$, then there is a polynomial-time (respectively uniform* NC*) solution to the $(m(n), s(n), d)$-help functions problem for some $m(n) = n^{\omega(1)}$ and $s(n) = n^{\omega(1)}$.*

The above shows that a solution to the RPP already implies lower bounds. Moreover, it makes sense also to look at efficient parallel algorithms for this problem, since they imply different lower bounds.

We now turn to algorithms for the Remote Point Problem. The RPP is a relatively new problem, and the best known parameters come from the paper of Alon et al. [APY09] that defined this problem. They show that, for any constant $c > 0$, the $(k(N), r(N))$-RPP has a polynomial-time algorithm for any $k(N) \leq N/2$ and $r(N) \leq cN\frac{\log k(N)}{k(N)}$. Note that these parameters are not sufficient to guarantee a solution to the help functions problem. We are unable to improve on these parameters. However, we show that these parameters can also be achieved by an efficient parallel algorithm.

Result 2. *For any constant c and any $k(\cdot), r(\cdot)$ such that $k(N) \leq N/2$ and $r(N) \leq cN\frac{\log k(N)}{k(N)}$, the $(k(N), r(N))$-RPP can be solved in uniform* NC*.*

The high level idea of our algorithm is similar to the solution of [APY09]. The main point of difference is that our solution goes via a hitting set for the $(N/2, c \log N)$-RPP using small-bias spaces [NN93, AGHP92]. Hitting sets with these parameters are themselves easy to construct (an alternative, easier construction is given in a manuscript version of [APY09]);

however, our hitting sets have enough additional structure to guarantee that we are able to extract from our hitting sets a point that is far from the given subspace. Moreover, the above can be done by an NC algorithm.

These results have appeared in [AS10a] and [AS10b].

1.2 Arithmetic circuits

Many computational problems of interest to us can be written down naturally as multivariate polynomials in the input variables: such problems include the determinant, the permanent, matrix multiplication, the Fast Fourier Transform, etc. The arithmetic circuit is a mathematical model that captures a natural class of algorithms for such problems: algorithms that only use the algebraic operations of the underlying field — addition and multiplication — to compute the function at hand.

Let us fix some notation. Throughout this section, unless otherwise mentioned, we work over a fixed, arbitrary field $\mathbb{F}$. Given a polynomial f from the ring $\mathbb{F}[X]$ for some set of variables X, we use $\deg(f)$ to denote the degree of the polynomial f, that is the highest degree of a monomial with a non-zero coefficient in f.

Definition 1.2.1. *An arithmetic circuit C over the variables $X = \{x_1, x_2, \ldots, x_n\}$ is a labelled directed acyclic graph with its leaves labelled by elements of $X \cup \mathbb{F}$ and internal nodes labelled by $+$ and $\times$ — these internal nodes are referred to as addition and multiplication gates respectively. The circuit has a designated output node v.*

An arithmetic circuit is said to be a formula *if the underlying undirected graph is a tree.*

We distinguish between bounded fanin *and* unbounded fanin *arithmetic circuits. The circuit C is said to have bounded fanin iff each of its internal nodes has at most two children. Otherwise, C has unbounded fanin.*

The size *of the circuit — denoted* $\mathrm{size}(C)$ *— is the number of vertices in the underlying graph. The* depth *of the circuit — denoted* $\mathrm{depth}(C)$ *— is the length of the longest directed path in the circuit.*

The polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ computed by the above arithmetic circuit C is defined in the following inductive manner. Each leaf simply computes the polynomial it is labelled with. A $+$ gate computes the sum of the polynomials computed by its children; and each $\times$ gate computes the product of the polynomials computed by its children. The polynomial f is just the polynomial computed by the output node v of the circuit.

1.2.1 Arithmetic circuit lower bounds

An important aim of research in arithmetic circuit complexity is to come up with an explicit family of polynomials $F = \{f_n \mid n \in \mathbb{N}\}$ such that F cannot be computed by a family of circuits of polynomial size. It is easy to see that this can be done if the degree of the polynomial is allowed to be arbitrary: one can easily verify that any bounded fanin circuit computing the polynomial x^d must have at least $\log d$ many vertices. To exclude trivial solutions of this form, we use the notion of a p-family [Bür00].

Definition 1.2.2. *Given a family of polynomials* $F = \left\{f_n \in \mathbb{F}[x_1, x_2, \ldots, x_{t(n)}] \;\middle|\; n \in \mathbb{N}\right\}$, *we say that* F *is a* p-family *if* $t(n)$ *and* $\deg(f_n)$ *are polynomially bounded functions in* n. *We say that the polynomial family* F *is* p-computable *if there is a family of arithmetic circuits* $\left\{C_n(x_1, \ldots, x_{t(n)}) \;\middle|\; n \in \mathbb{N}\right\}$ *such that, for each* n, C_n *computes* f_n *and* $\mathrm{size}(C_n)$ *is a polynomially bounded function of* n.

Example 1.2.3. *Two important examples of p-families are the determinant and permanent families of polynomials, which we now define. Fix any* $n \in \mathbb{N}$. *Let* $X = \{x_{i,j} \mid i, j \in [n]\}$. *We denote by* S_n *the group of permutations on* $[n]$. *For* $\sigma \in S_n$, *we denote by* $\mathrm{sgn}(\sigma)$ *the* sign of the permutation σ, *which is* -1 *if* σ *has an odd number of inversions and* 1 *otherwise.*

The polynomials $\det_n(X)$ *and* $\mathrm{per}_n(X)$ *are defined as follows.*

$$\det_n(X) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) x_{1,\sigma(1)} x_{2,\sigma(2)} \cdots x_{n,\sigma(n)}$$

$$\mathrm{per}_n(X) = \sum_{\sigma \in S_n} x_{1,\sigma(1)} x_{2,\sigma(2)} \cdots x_{n,\sigma(n)}$$

We denote by det *the family* $\{\det_n \mid n \in \mathbb{N}\}$ *and* per *the family* $\{\mathrm{per}_n \mid n \in \mathbb{N}\}$. *Despite the apparent similarities in their definitions, the complexities of these families of polynomials are believed to be very different. On the one hand, the family* det *is known [Ber84, Chi85, MV97] to be p-computable. On the other hand, it is believed [Val79, Bür00], but as yet unproven, that* per *cannot be computed by any polynomial-sized circuit family.*

We can now be more precise: we would like to come up with an explicit p-family F such that F is not p-computable. There are many notions of "explicit" that are natural here [Bür00]: for our purposes, it is sufficient to define a p-family $F = \left\{f_n(x_1, \ldots, x_{t(n)}) \;\middle|\; n \in \mathbb{N}\right\}$ to be explicit if there is a deterministic Turing Machine which, when given as input n and a monic monomial $x_1^{e_1} x_2^{e_2} \ldots x_{t(n)}^{e(n)}$, computes the coefficient of this monomial in the polynomial f_n in time $n^{O(1)}$.

The objective of coming up with a lower bound of the above kind is far from being met: the best explicit lower bound for general arithmetic circuits is $\Omega(n \log n)$, and follows from results due to Strassen [Str73] and Baur and Strassen [BS83].

Even in the case of constant-depth arithmetic formulas, explicit lower bounds remain elusive. When the field is $\mathbb{F}_2$, explicit exponential lower bounds for computing certain elementary symmetric polynomials follow from the lower bounds of Razborov [Raz87] and Smolensky [Smo87] for constant-depth Boolean circuits with Parity gates. In other fields, however, the situation is much less clear. Grigoriev and Karpinski [GK98] showed that, when $\mathbb{F}$ is of *constant* size, any depth-3 circuit computing the determinant over $\mathbb{F}$ must be of exponential size. Grigoriev and Razborov [GR00] showed such a lower bound for some symmetric functions (again when $\mathbb{F}$ is of constant size). When the field is large enough, however, these proof techniques break down, and the best known lower bound for depth-3 circuits over fields of characteristic zero is $\Omega(n^2)$, due to Shpilka and Wigderson [SW01]. Indeed, by a result credited to Ben-Or (see [SW01]), it is known that depth-3 circuits over large fields can be surprisingly powerful: every elementary symmetric polynomial in n variables over a field of size at least n can be computed by a depth-3 formula of size $O(n^2)$.

Strong lower bounds are known under other restrictions on the arithmetic circuits under consideration. One such restriction is *monotonicity*. Let the underlying field be the real numbers $\mathbb{R}$. An arithmetic circuit C computing a polynomial $p \in \mathbb{R}[x_1, x_2, \ldots, x_n]$ is said to be *monotone* if it uses no non-negative constants from $\mathbb{R}$. Monotonicity is a fairly severe restriction on the arithmetic circuit, since it ensures that no cancellations can occur in the circuit. Such a circuit can only compute polynomials with non-negative coefficients, and it is an interesting problem to demonstrate the existence of explicit polynomials of this kind that cannot be computed by monotone circuits of polynomial size. Jerrum and Snir [JS82] showed that per_n cannot be computed by monotone arithmetic circuits of size $o(n2^n)$. Valiant [Val80] demonstrated the weaknesses of monotone computation by showing that general arithmetic circuits can be exponentially more powerful than monotone arithmetic circuits.

Another restricted setting in which explicit lower bounds are known is that of *noncommutative arithmetic circuits*. Noncommutative arithmetic circuits compute multivariate polynomials in the noncommutative polynomial ring $\mathbb{F}\langle x_1, x_2, \ldots, x_n\rangle$. These circuits are defined similarly to the bounded fanin arithmetic circuits defined above, with the following additional feature: given any $\times$-gate of fanin 2, one of its two children is labelled the *left child* and the other the *right child*. The polynomial $f \in \mathbb{F}\langle x_1, x_2, \ldots, x_n\rangle$ is defined exactly the same way, except at each $\times$-gate, where the ordering among the children is taken into account in defining the polynomial computed at the gate. Noncommutative arithmetic circuits are easier to analyze since each monomial can appear in only a restricted number of ways. For example, if p and q are homogeneous polynomials from the ring $\mathbb{F}\langle X\rangle$, then each monomial in the product $p \cdot q$ appears as a result of the product of exactly one monomial in p and one monomial in q.

Indeed, Nisan [Nis91] proved that any noncommutative *formula* computing the noncommutative versions of $\det_n$ or per_n must be of size $2^{\Omega(n)}$.[1] (Note the contrast from the

[1]Nisan's lower bound actually holds for Algebraic Branching Programs, a slightly stronger model of computation we will see later.

commutative case, where $\det_n$ has polynomial-sized circuits and quasipolynomial-sized formulas.) This might give one hope that it is possible to prove that per_n can be shown to have superpolynomial complexity in the general noncommutative circuit model also, which would be a special case of the general commutative circuit lower bound. However, this hope remains unfulfilled: no explicit lower bounds are known for general noncommutative arithmetic circuits. Nisan's lower bound remains the strongest lower bound we known for noncommutative computation.

Finally, we mention some results regarding *multilinear circuits*. A polynomial $f \in \mathbb{F}[X]$ is said to be *multilinear* if the degree of each variable in the polynomial is at most one. An arithmetic circuit is said to be multilinear if the polynomial computed at each of its gates is multilinear; such a circuit of course computes a multilinear polynomial. Multilinear circuits were first considered by Nisan and Wigderson [NW97], who conjectured that multilinear bounded fan-in circuits for the determinant must have depth $\Omega(n)$; by a recent result of Raz and Yehudayoff [RY08], this is equivalent to proving that any multilinear formula for the determinant must have exponential size. The first breakthrough in this direction was made by Raz [Raz09], who showed that any multilinear formula computing $\det_n$ or per_n must be of superpolynomial size; using these techniques, Raz [Raz06] also showed a superpolynomial separation between the power of multilinear circuits of depth $O(\log n)$ and $O(\log^2 n)$. Raz and Yehudayoff [RY09] showed stronger lower bounds and separations for constant-depth multilinear circuits. In the slightly more general setting of *syntactic* multilinear circuits of arbitrary depth, Raz, Shpilka, and Yehudayoff [RSY08] demonstrate an explicit multilinear polynomial that has no circuits of size $n^{4/3-\varepsilon}$, for any $\varepsilon > 0$.

1.2.2 Our results

We prove lower bounds and hardness results for restricted kinds of arithmetic circuits.

Monotone constant-width circuits

We first consider monotone arithmetic circuits of *bounded width*. An arithmetic circuit is said to be of width w if its gates can be arranged in layers, with edges going from one layer to the next, with the property that the width of each layer is bounded by w; a family of bounded width circuits is a family of arithmetic circuits of width $O(1)$. Such circuits capture the power of bounded memory algorithms for computing polynomials. They have been studied extensively in the literature (e.g. [BOC92, LMR10, MR09, JR09]), but not with a view towards proving lower bounds. One reason for this is the result of Ben-Or and Cleve [BOC92], which shows that bounded-width circuits can be fairly powerful: any arithmetic

formula of size s can be simulated by a width-4 circuit of size $O(s^2)$.[2]

The proof of Ben-Or and Cleve's result proceeds by explicitly transforming a given arithmetic formula into a width-4 circuit. This process, however, uses negation crucially and hence, it is not clear if an analogous result holds in the monotone world: can one prove that any monotone arithmetic formula can be simulated by a constant-width arithmetic circuit of comparable size? We show that the answer is no in a strong sense, and hence that the use of negation gates in the construction of Ben-Or and Cleve cannot be avoided.

Result 3. *For any constant d, there is an explicit family of polynomials $\{p_n \in \mathbb{F}[x_1, x_2, \ldots, x_n]\}$ such that p_n can be computed by a monotone arithmetic formula of size $O(n)$ and depth $2d$, but not by any monotone arithmetic circuit of width d and size $o(2^{n^{1/d}})$.*

The result has the nice consequence that the hierarchies of polynomial families that can be computed by constant-width and constant-depth monotone circuits of polynomial size are infinite. This follows since any polynomial computed by a monotone arithmetic formula of constant depth d and size s can be computed by a monotone arithmetic circuit of width d and size $O(s)$.

Noncommutative arithmetic circuits

We now explain our results regarding noncommutative arithmetic circuits. We start with the definition of a slightly different model of computation: the noncommutative *Algebraic Branching Programs* (ABPs). In this section, all polynomials and circuits will be noncommutative.

Definition 1.2.4. *An ABP is a directed acyclic graph with one vertex of in-degree zero, which is called the* source*, and one vertex of out-degree zero, which is called the* sink*. The vertices of the graph are partitioned into levels numbered $0, 1, \ldots, d$. Edges may only go from level i to level $i+1$ for $i = 0, 1, \ldots, d-1$. The source is the only vertex at level 0 and the sink is the only vertex at level d. Each edge is labeled with a homogeneous linear form in the variables X. The size of the ABP is the number of vertices.*

The ABP computes a degree d homogeneous polynomial $f \in \mathbb{F}\langle X \rangle$ as follows. Fix any path γ from source to sink with edges $e_1, e_2, \ldots, e_d$, where e_i is the edge from level $i-1$ to level i, and let ℓ_i denote the linear form labelling edge e_i. We denote by f_γ the homogeneous degree d polynomial $\ell_1 \cdot \ell_2 \cdots \ell_d$ (note that the order of multiplication is important). The polynomial f computed by the ABP is simply

$$f = \sum_{\gamma \in \mathcal{P}} f_\gamma$$

[2]Ben-Or and Cleve state their results in terms of *register machines*. They show that any polynomial computed by an arithmetic formula of size s can be computed by a 3-register machine of size s^2. In our slightly different formalism of bounded-width circuits, this construction seems to require width 4.

where $\mathcal{P}$ is the set of all paths from the source to the sink.

In the noncommutative setting, this model was first studied by Nisan [Nis91]. It is easy to see that any polynomial computed by an ABP can be computed by a noncommutative arithmetic circuit of comparable size. Nisan [Nis91] observed (see [RS05] for a complete proof) that any homogeneous polynomial computed by an arithmetic formula of size s can be computed by an ABP of size $O(s^2)$; he also showed that any ABP computing the polynomials $\det_n$, per_n, and a host of other polynomials must be of size $2^{\Omega(n)}$.

In analogy with our work on Boolean circuits outlined in Section 1.1, we study ABPs with *help polynomials*. Given a small collection $H = \{h_1, h_2, \ldots, h_m\}$ of *arbitrary* polynomials over the noncommutative ring $\mathbb{F}\langle x_1, x_2, \ldots, x_n\rangle$, we define an ABP using the help polynomials H to be an ABP that is allowed to use linear combinations of its input variables as well as the polynomials in H as labels for its edges. The polynomial computed by the ABP is defined in exactly the same way. The lower bound question for this model is defined below.

Let $X = \{x_1, x_2, \ldots, x_n\}$. For functions $m, s, d : \mathbb{N} \to \mathbb{N}$, the $(m(n), s(n), d(n))$-Help polynomial problem is defined as follows:

- INPUT: A collection of m polynomials $H = \{h_1, h_2, \ldots, h_m\} \subseteq \mathbb{F}\langle X\rangle$, given by their coefficients.
- DESIRED OUTPUT: A polynomial $F \in \mathbb{F}\langle X\rangle$ of degree $d(n)$ such that F cannot be computed by a size $s(n)$ ABP using the help polynomials H.

We would like an algorithm for the above problem that runs in time polynomial in its input size and output size n^d. We note that the polynomial $F \in \mathbb{F}\langle X\rangle$ is explicit in a weaker sense than the definition given above. However, we feel that this is justified here, since the polynomials in H are allowed to be arbitrary. Moreover, this still means that the polynomial $F \in \mathbb{F}\langle X\rangle$ can be computed in time $m^{O(1)}n^{O(d)}$, which for $m = n^{O(d)}$ and $d = \mathrm{poly}(n)$ is $\exp(n)$, and this is also a reasonable notion of "explicit" that is often used in the literature.

To state our result in simpler form here, we assume that H contains only homogeneous polynomials. The more general form of the result is similar. Let $\Delta(H)$ denote $\max_{h\in H} \deg(h)$ and $\delta(H) = \min_{h\in H} \deg(h)$.

Result 4. *For $m(n), s(n) = n^{o(d(n))}$ and any constant $\varepsilon > 0$, the $(m(n), s(n), d(n))$-Help polynomial problem can be solved efficiently if either $\Delta(H) \leq d(n)(1-\varepsilon)$ or $\delta(H) \geq \frac{d(n)}{2}(1+\varepsilon)$.*

Our solution is inspired by the connection between the Help functions problem and the Remote Point Problem outlined in Section 1.1.2. We show that to solve the Help polynomial problem, it suffices to solve a problem similar to the Remote Point Problem, that we call

the *Remote Point Problem in the rank metric* or the *Remote Matrix Problem* (RMP). This problem is defined below.

For $N \in \mathbb{N}$, let $\mathbb{F}_2^{N \times N}$ denote the vector space of all $N \times N$ matrices with entries from $\mathbb{F}_2$. Given functions $k, r : \mathbb{N} \to \mathbb{N}$, we define the $(k(N), r(N))$-RMP as follows:

- INPUT: A subspace V of $\mathbb{F}_2^{N \times N}$ of dimension at most $k(N)$ given by its generators.
- DESIRED OUTPUT: A matrix $M \in \mathbb{F}_2^{N \times N}$ such that for each $M' \in V$, $\text{rank}(M - M') \geq r(N)$.

Our lower bounds for the Help polynomials problem defined above follows from an easy solution to the $(k, N/k)$-RMP. It is easily seen that a solution with improved parameters will result in a better solution to the Help polynomials problem. We take a first step in this direction by giving an algorithm with slightly improved parameters for the RMP – unfortunately, the improvement in parameters is too small to translate into an appreciably better solution to the Help polynomials problem. At a high level, this improved algorithm is similar to the algorithm of Alon, Panigrahy, and Yekhanin [APY09] for the Remote Point Problem. The result is as follows.

Result 5. *For any fixed constant $c > 0$, there is a polynomial-time algorithm for the $(\ell N, r)$-RMP, for ℓ, r such that $\ell \cdot r < c \log N$.*

Continuing our study of noncommutative arithmetic circuits, we study the problem of proving lower bounds for general noncommutative arithmetic circuits computing the noncommutative version of the determinant polynomial det. There are many valid ways of defining the noncommutative determinant with different orderings of the variables in each monomial, and we study some of these. For concreteness, we concentrate in the introduction on the *Cayley determinant* which is defined below. Let $X = \{x_{ij} \mid i, j \in [n]\}$.

$$\text{Cdet}_n(X) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \cdot x_{1,\sigma(1)} x_{2,\sigma(2)} \cdots x_{n,\sigma(n)}$$

The Cayley permanent $\text{Cper}_n(X)$ is defined similarly.

Nisan [Nis91] showed that any noncommutative ABP computing Cdet_n must be of size $2^{\Omega(n)}$. However, this fails to imply anything for general circuits, since in the same paper, Nisan also proved an exponential separation between the power of ABPs and general circuits. Hence, it may very well be that Cdet_n has polynomial-sized arithmetic circuits.

However, the general feeling is that the noncommutative determinant cannot be computed efficiently even by general arithmetic circuits. The reason for this belief is that we do not know of any algorithm to compute the noncommutative determinant over any sufficiently rich noncommutative domain.[3]

[3]There are efficient algorithms to compute the determinant over some restricted noncommutative domains, such as the ring of constant-dimensional upper triangular matrices with entries from $\mathbb{F}$ [CH10].

We put the above intuition on firm ground, by showing that if $\{\mathrm{Cdet}_n \mid n \in \mathbb{N}\}$ can be computed by a circuit family of polynomial size, then so can $\{\mathrm{Cper}_n \mid n \in \mathbb{N}\}$. In particular, this implies that the *commutative* permanent has small commutative arithmetic circuits, something that is unlikely to happen.

Result 6. *Suppose that* $\{\mathrm{Cdet}_n \mid n \in \mathbb{N}\}$ *can be computed by a circuit family of size* $s(n)$*. Then,* $\{\mathrm{Cper}_n \mid n \in \mathbb{N}\}$ *can be computed by a circuit family of size* $\mathrm{poly}(n, s(n))$*.*

Another motivation for studying the complexity of the Cayley determinant is for its role in a long-standing approach [GG81, KKL$^+$93, CRS03] to the design of randomized approximation algorithms for the 0-1 permanent through good unbiased estimators based on the determinant. Godsil and Gutman [GG81] demonstrated a simple exponential time Monte Carlo algorithm for approximating the 0-1 permanent based on computing the determinants of suitably picked random matrices with real entries; this approach was fine-tuned by Karmarkar et al. [KKL$^+$93] to give better (though still exponential-time) Monte Carlo algorithms using determinants of matrices with complex values. Building on these, Chien, Rasmussen, and Sinclair [CRS03] showed that polynomial-time algorithms to compute the Cayley determinant over Clifford algebras of polynomial dimension would yield polynomial-time approximation algorithms for the permanent. The question of whether there is such an algorithm is open.

We show that such an algorithm is unlikely to exist. Specifically, we show that our techniques yield a reduction from computing the 0-1 permanent (and indeed, the permanent over non-negative rational entries) to computing the Cayley determinant over sufficiently rich noncommutative domains, such as matrix algebras and Clifford algebras of polynomial dimension. Since the 0-1 permanent is #P-complete [Val79], this implies that there is no polynomial time algorithm to compute the Cayley determinant over these algebras unless the polynomial hierarchy collapses.

For any $n \in \mathbb{N}$, let $\mathcal{A}_n$ denote the algebra of $n \times n$ matrices over $\mathbb{Q}$, and let $\mathcal{B}_n$ denote the $(\log n)$th Clifford algebra (this is an algebra of dimension n over $\mathbb{R}$).

Result 7. *If either of the following are true,*

- *There is a* $\mathrm{poly}(n)$*-time algorithm to evaluate* Cdet_n *over the algebra* $\mathcal{A}_n$*.*
- *There is a* $\mathrm{poly}(n)$*-time algorithm to evaluate* Cdet_n *over the algebra* $\mathcal{B}_n$*.*

then the 0-1 *permanent can be computed in polynomial time.*

The above results on the complexity of the noncommutative determinant rely on a binary operation on polynomials that we call the *Hadamard product*. This product was used implicitly in [AMS08, AM08] to devise identity-testing algorithms for noncommutative polynomials.

The Hadamard product was formally defined in [AJ09] and further used in [AJS09a] to devise a complexity-theoretically optimal deterministic identity test for noncommutative ABPs computing polynomials in $\mathbb{Q}\langle X\rangle$.

The results described above have appeared or are to appear in [AS10a], [AJS09b], and [AS].

1.3 Organization of this thesis

We present our results in the same order as they are described above. In Chapter 2, we describe the Boolean Help functions problem and its connection to the Remote Point Problem. We also show that the best-known parameters for this algorithmic question can be achieved by an efficient parallel algorithm. Following this, we consider the Help polynomials problem – an arithmetic analogue of the help functions problem – and describe our results regarding this problem in Chapter 3. In Chapter 4, we consider constant-width computation and show a separation between the power of constant-depth formulas and constant-width circuits under the restriction of monotonicity. This is followed by results on the complexity of the noncommutative determinant, which we present in Chapter 5.

Chapter 2

The Help functions problem

2.1 Introduction

In this chapter, we present our results on lower bounds for *Constant-depth Boolean circuits with help functions*. We start by defining constant-depth Boolean circuits and constant-depth Boolean circuit families, followed by a formal description of the Help functions problem, that was first studied by Jin-yi Cai [Cai90]. We then provide our own motivation for studying this problem, and our results regarding this problem. The main results are the following.

- We present an approach to solving the Help functions problem: we show that if the Remote Point Problem (RPP) introduced by Alon, Panigrahy, and Yekhanin [APY09] with certain parameters can be solved efficiently, then so can the help functions problem for the parameters that we are interested in.
- We demonstrate that the best-known parameters for the RPP can be achieved by a procedure that can be implemented by an efficient parallel algorithm.

2.2 Preliminaries

We first recall the definition of Boolean circuits from Section 1.1.

Definition 2.2.1. *A* Boolean circuit C *over* n *Boolean variables* $X = \{x_1, x_2, \ldots, x_n\}$ *is a labelled directed acyclic graph such that:*

- *The internal nodes of* C *are labelled by* $\wedge$ *and* $\vee$ *– these are called the AND and OR gates respectively – or constants from* $\{0, 1\}$*, and the leaves are labelled by Boolean literals over* X*, which are elements from the set* $\{x_1, \overline{x_i} \mid i \in [n]\}$*.*

- *C has a set of designated output nodes $v_1, v_2, \ldots, v_m$.*

Each node computes a Boolean function of the n input variables in a natural way. A leaf node labelled by x_i or $\overline{x_i}$ just computes the projection to the ith variable or its negation respectively and a leaf node labelled by a constant just computes that constant function. The internal AND and OR gates just compute the AND and OR of the functions computed by their children. The function computed by C is defined to be $x \mapsto (f_1(x), f_2(x), \ldots, f_m(x))$, where f_i is the Boolean function computed by v_i, for each $i \in [m]$.

The size *of C — denoted* $\mathrm{size}(C)$ *— is defined to be the number of vertices in C and the* depth *of C — denoted* $\mathrm{depth}(C)$ *— is defined to be the length of the longest directed path in C.*

Mostly, we will be interested in the case that $m = 1$, that is, when C computes a Boolean function. Unless explicitly stated otherwise, we assume that any circuit family under consideration has this property.

A Boolean circuit over n variables computes a function $f : \{0,1\}^n \to \{0,1\}^*$. Since we want to consider functions $F : \{0,1\}^* \to \{0,1\}^*$ defined on all input lengths, we define the notion of a *Circuit Family*, which is an ensemble of Boolean circuits $\{C_n \mid n \in \mathbb{N}\}$ such that C_n computes a function on $\{0,1\}^n$; the circuit family $\{C_n \mid n \in \mathbb{N}\}$ is said to compute F if C_n computes the function F restricted to $\{0,1\}^n$.

Given a function $s : \mathbb{N} \to \mathbb{N}$, we use $\mathrm{size}(s(n))$ to denote the class of Boolean functions $F : \{0,1\}^* \to \{0,1\}$ that can computed by a family of circuits $\{C_n \mid n \in \mathbb{N}\}$ such that $\mathrm{size}(C_n) \leq s(n)$. Similarly, given functions $s, d : \mathbb{N} \to \mathbb{N}$, we use $\mathrm{SizeDepth}(s(n), d(n))$ to denote the class of functions that can be computed by circuits of size $s(n)$ and depth $d(n)$. Some of these complexity classes are standard in the literature, in which case we use the standard terminology. We use the standard notation P/poly to denote $\bigcup_{c>0} \mathrm{Size}(cn^c)$, and the standard notation AC^0 to denote $\bigcup_{c>0} \mathrm{SizeDepth}(cn^c, c)$.

Finally, we use $\mathrm{FSize}(s(n))$, $\mathrm{FSizeDepth}(s(n), d(n))$ to denote the analogues of the above classes for arbitrary (not necessarily Boolean) functions $F : \{0,1\}^* \to \{0,1\}^*$.

2.2.1 Uniformity conditions

Note that the cardinality of the collection of all circuit families $\{C_n\}$ is uncountable, and hence they can compute Boolean functions not computable by any Turing machine. To overcome this undesirable aspect of circuit families, especially when giving algorithms that can be realized by circuit families, we impose certain *uniformity* conditions on the circuits.

We say that a circuit family $\mathcal{C} = \{C_n \mid n \in \mathbb{N}\}$ is *uniform* if the map $1^n \mapsto C_n$ is computable in space $O((\log n)^{O(1)})$. This notion of uniformity is a relaxation of the notions that are

popular in Complexity theory literature. The case where the above map is computable in space $O(\log n)$ is called *Logspace uniformity* and was defined by Borodin [Bor77]. A more restrictive version – called DLOGTIME-*uniformity* [BIS90] – is the most standard notion of uniformity for circuit families. However, the more relaxed version defined above seems better suited to our purposes.

We denote by Unif–SizeDepth$(s(n), d(n))$ the class of Boolean functions that can be computed by a uniform circuit family of size $s(n)$ and depth $d(n)$ and by FUnif–SizeDepth$(s(n), d(n))$ the class of all functions $F : \{0,1\}^* \to \{0,1\}^*$ that can be computed by a uniform circuit family of size $s(n)$ and depth $d(n)$.

2.2.2 The Help functions problem

We now define the Help functions problem. Our aim is to consider constant-depth circuits augmented with a few arbitrary "help functions" in the bottom layer. More precisely, given a small collection $H = \{h_i : \{0,1\}^n \to \{0,1\} \mid i \in \{1, 2, \ldots, m\}\}$ of *arbitrary* Boolean functions, what functions cannot be computed by a small constant-depth circuit that gets as input $h_1(x), h_2(x), \ldots, h_m(x)$? Clearly, such a function must depend on $h_1, h_2, \ldots, h_m$, since they are allowed to be arbitrary. We are interested in if we can come up with such a function $f : \{0,1\}^n \to \{0,1\}$ *efficiently*. We now define the model and the problem formally.

Fix a collection of Boolean functions H defined on $\{0,1\}^n$. A Boolean circuit C using the help functions H is defined in exactly the same way as a standard Boolean circuit, except that leaves may only be labelled by constants or a function $h \in H$. The function computed by such a circuit is defined in the natural way: on an input x, a leaf labelled by a function h computes $h(x)$ and a leaf labelled by a constant computes that constant; the inductive definition of the functions computed by the internal nodes remains the same. Given a collection H of Boolean functions defined on $\{0,1\}^n$ and $s, d \in \mathbb{N}$, we will use $\text{SizeDepth}^n_H(s, d)$ to denote the collection of functions $f : \{0,1\}^n \to \{0,1\}$ that can be computed by Boolean circuits of size at most s and depth at most d using H.

For functions $m, s, d : \mathbb{N} \to \mathbb{N}$, the $(m(n), s(n), d(n))$-Help function problem is defined as follows:

- INPUT: A collection of truth-tables of m functions $h_1, h_2, \ldots, h_m : \{0,1\}^n \to \{0,1\}$.
- DESIRED OUTPUT: A truth-table of a function $f : \{0,1\}^n \to \{0,1\}$ such that $f \notin \text{SizeDepth}^n_H(s(n), d(n))$.

Note that the input length above is $m2^n$. The parameters that we are primarily interested in is when $m(n) = n^{\omega(1)}$, $s(n) = m(n)^{\omega(1)}$, and $d(n) = O(1)$.

A similar problem was studied by Jin-yi Cai in [Cai90]. In this paper, Cai showed an exponential lower bound for the size of constant-depth circuits that computes m specific parities of subsets of the input bits, in the presence of (any) $m-1$ help functions, but only when $m \ll n^{1/5}$. His proof is essentially based on Smolensky's dimension argument [Smo87]. Unfortunately, his result seems inapplicable in our context, for two reasons. The first is that we would like a single Boolean function that cannot be computed using the help functions, whereas Cai shows a lower bound for computing a short list. The second, and more vital, reason is that in our setting is that when we allow for polynomially or superpolynomially many help functions Smolensky's argument doesn't seem to work.

At the same time, we note that we are unable to recover a lower bound using our methods in the setting considered by Cai.

2.3 Connection to other lower bounds

In this section, we give some more motivation to study the Help functions problem. The problems we study are special cases of the following two outstanding problems: proving that EXP $\not\subseteq$ P/poly and that PSPACE $\not\subseteq$ P/poly. Given any class $\mathcal{C}$ of languages, we let $\mathcal{R}^p_m(\mathcal{C})$ the polynomial-time many-one closure of $\mathcal{C}$, i.e., the class of languages that can be polynomial-time many-one reduced to a language in $\mathcal{C}$; we similarly use $\mathcal{R}^l_m(\mathcal{C})$ to denote the logspace many-one closure of $\mathcal{C}$.

We consider the problem of proving lower bounds against the closures of AC^0 with respect to polynomial-time and logspace many-one reductions. More precisely, we consider the problem of showing that EXP $\not\subseteq \mathcal{R}^p_m\mathrm{AC}^0$ and PSPACE $\not\subseteq \mathcal{R}^l_m\mathrm{AC}^0$.

Let us first provide some motivation for proving lower bounds against the complexity classes $\mathcal{R}^p_m(\mathrm{AC}^0)$ and $\mathcal{R}^l_m(\mathrm{AC}^0)$. Consider, for example, the problem of showing that EXP $\not\subseteq \mathcal{R}^p_m(\mathrm{AC}^0)$. We know, from the Time-Hierarchy theorem, that EXP $\not\subseteq$ P; also, from the lower bounds of Furst, Saxe, and Sipser [FSS84] and Ajtai [Ajt83], it follows easily that EXP $\not\subseteq \mathrm{AC}^0$. The lower bound techniques used in proving the above two results are very different: the first is a relativizing, non-naturalizing technique, whereas the second in a non-relativizing, but naturalizable lower bound. However, neither technique by itself will be sufficient to prove a lower bound for P/poly, and it is possible that a synthesis of the two will be necessary if we are to tackle this harder question. It seems like proving a lower bound against $\mathcal{R}^p_m(\mathrm{AC}^0)$ will require exactly that, and hopefully, lessons gained in tackling this question will be useful in proving that EXP $\not\subseteq$ P/poly. A similar case can be made for the problem of proving that PSPACE $\not\subseteq \mathcal{R}^l_m(\mathrm{AC}^0)$.

We now show the connection between the above lower bound questions and the Help functions problem. We show that a suitably efficient solution to the $(m(n), s(n), d(n))$-Help functions problem for certain m, s, and d would imply that EXP $\not\subseteq \mathcal{R}^p_m(\mathrm{AC}^0)$ and

PSPACE $\not\subseteq \mathcal{R}^l_m(\mathrm{AC}')$. The proof proceeds by a standard diagonalization argument. The idea is simple: we capture the output of a reduction on inputs of a certain length (that is the input to the constant-depth circuit) by means of a small set of help functions H. Then, assuming that we can solve the help functions problem efficiently, we just plug in a function that cannot be solved by a constant-depth circuit using H. The formal proof follows.

Theorem 2.3.1. *Let $m(n), s(n)$ be such that $n^{\omega(1)} \leq m(n) \leq 2^{o(n)}$ and $n^{\omega(1)} \leq s(n) \leq 2^{o(n)}$. Let $N = m(n)2^n$ be the length of the input to the $(m(n), s(n), d(n))$-Help functions problem. Then, the following hold.*

- *If, for every constant d, the $(m(n), s(n), d)$-help functions problem can be solved by a deterministic algorithm running in time $2^{(\log N)^{O(1)}}$, then* $\mathrm{EXP} \not\subseteq \mathcal{R}^p_m(\mathrm{AC}^0)$.
- *If, for every constant d, the $(m(n), s(n), d)$-help functions problem lies in* $\mathrm{FUnif}\text{–}\mathrm{SizeDepth}(2^{(\log N)^{O(1)}}, (\log N)^{O(1)})$[1]*, then* $\mathrm{PSPACE} \not\subseteq \mathcal{R}^l_m(\mathrm{AC}^0)$.

Proof. For any $d \in \mathbb{N}$, let AC^0_d denote the class of languages that are accepted by circuit families of polynomial size and depth d.

We start with the first claim of the theorem. Assume that the (m, s, d)-help functions problem can be solved in deterministic time $2^{(\log N)^{O(1)}} = 2^{n^{O(1)}}$ for any constant d. Note that to prove that $\mathrm{EXP} \not\subseteq \mathcal{R}^p_m(\mathrm{AC}^0)$, it suffices to prove that $\mathrm{EXP} \not\subseteq \mathcal{R}^p_m(\mathrm{AC}^0_d)$ for each fixed $d \in \mathbb{N}$, since EXP contains languages that are complete for it under polynomial-time many-one reductions and hence, if any complete language L reduces in polynomial time to $\mathrm{AC}^0_{d_0}$, then all of EXP will reduce to $\mathrm{AC}^0_{d_0}$. We will now describe, for any fixed $d \in \mathbb{N}$, an EXP machine that accepts a language $L_d \notin \mathcal{R}^p_m(\mathrm{AC}^0_d)$.

We proceed by diagonalization. Let $R_1, R_2, R_3, \ldots$ be any standard enumeration of all polynomial-time many-one reductions such that each reduction appears infinitely often in the list. Fix $n \in \mathbb{N}$ and let $m = m(n)$. On an input $x \in \{0,1\}^n$, the EXP machine does the following: for each $y \in \{0,1\}^n$, it runs R_n for m steps and computes $R_n(y)$ (if R_n does not halt in m steps, the machine outputs 0 and halts). It can thus produce the truth tables of functions $h_i : \{0,1\}^n \to \{0,1\}$ $(i \in [m])$ such that for each $y \in \{0,1\}^n$, $h_i(y)$ is the ith bit of $R_n(y)$ if $|R_n(y)| \geq i$ and 0 otherwise. Now, by assumption, in time $2^{n^{O(1)}}$, the EXP machine can compute the truth table of a function $g_n : \{0,1\}^n \to \{0,1\}$ such that $g_n \notin \mathrm{SizeDepth}_{\{h_1,\ldots,h_m\}}(s(n), d)$. Having computed g_n, the EXP machine just outputs $g_n(x)$. That is, we define the language L_d so that the characteristic function of L_d restricted to inputs in $\{0,1\}^n$ is the function g_n. By a standard diagonalization argument, it follows that L_d cannot be polynomial-time many-one reduced to any language in AC^0_d.

Now we prove the second part of the theorem. The proof is almost the same as that of the first part. Under the assumption that the $(m(n), s(n), d)$-Help functions problem is in

[1]We note that the help functions problem does not define a unique function. What we mean is that the circuit family computes some function that solves the help functions problem.

FUnif–SizeDepth$(2^{(\log N)^{O(1)}}, (\log N)^{O(1)})$ for any constant d, we show that there is a language L'_d accepted by a machine M' using space $n^{O(1)}$ such that L'_d cannot be reduced in logspace to AC^0_d. Similarly to the case of EXP, along with the existence of many-one complete languages for PSPACE, this implies that PSPACE $\not\subseteq \mathcal{R}^p_m(\mathrm{AC}^0)$. This time we enumerate the logspace reductions $R'_1, R'_2, \ldots$ so that each reduction appears infinitely often in the enumeration. Note that $\lfloor \log m \rfloor$ is asymptotically an upper bound on the space used by these reductions.

Fix an input x of length n. We define the help functions $h'_1, h'_2, \ldots, h'_m$ in a manner similar to the way we defined $h_1, h_2, \ldots, h_m$. Given $y \in \{0,1\}^n$, if R'_n uses space more than $\lfloor \log m \rfloor$ on input y, then we define $h'_i(y)$ to be 0 for each i. Otherwise, we define $h'_i(y)$ to be the ith bit of the output of R'_n (if the length of the output of R'_n is less than i, then $h'_i(y)$ is defined to be 0). Note that if R'_n uses space at most $\lfloor \log m \rfloor$, then it runs for at most m steps and hence the length of its output is at most m. Hence, the help functions capture the entire output of R'_n. Also note that each $h'_i(y)$ can be computed in space $O(\log m)$, which is $o(n)$.

As in the first part of the theorem, we would like to define M' so that the characteristic function of L'_d restricted to $\{0,1\}^n$ is a function g'_n such that $g'_n \notin \mathrm{SizeDepth}_{\{h'_1,\ldots,h'_m\}}(s(n), d)$. This would imply that L'_d cannot be reduced in logspace to AC^0. The function $g'_n : \{0,1\}^n \to \{0,1\}$ is of course the output of the uniform circuit family $\mathcal{C} = \{C_k \mid k \in \mathbb{N}\}$ solving the help functions problem on the input $(h'_1, h'_2, \ldots, h'_m)$. We only need to see that $g'_n(x)$ can be computed in space $n^{O(1)}$. Let C be the circuit from the family $\mathcal{C}$ that solves the help functions problem at this input length.

The machine M' needs to compute the xth bit of the output of the circuit C on input $(h'_1, h'_2, \ldots, h'_m)$. Observe that computing this value is equivalent to evaluating a game tree of depth $(\log N)^{O(1)} = n^{O(1)}$, whose leaves are labelled by expressions of the form $h'_i(y)$, for $i \in [m]$ and $y \in \{0,1\}^n$, and can be evaluated in space $o(n)$ by the argument above. Moreover, the game tree itself is given by the circuit C and can hence be evaluated in space $n^{O(1)}$ (this follows from the uniformity of the circuit family.)[2] Hence, it is easy to see that the entire game tree can be evaluated in space $n^{O(1)}$, which proves that L'_d is indeed in PSPACE. □

2.4 The connection to the Remote Point Problem

In this section, we address the problem of proving lower bounds for constant depth circuits using help functions. Our goal is to show how the problem is related to the Remote Point Problem defined by Alon et al. [APY09].

Our main tool will be the well known fact that constant depth circuits can be well approx-

[2]This is exactly why we why defined our notion of uniformity to be so that the circuit is computable in polylogarithmic space. Since the notions of uniformity that are more popular in the literature are more restrictive than our own, our result also holds for those notions of uniformity.

imated by polylogarithmic degree polynomials, for different notions of approximation. We state the results of Tarui [Tar93] (see also [BRS95]) in the form that we require. In what follows, the field we work in will be $\mathbb{F}_2$, but our results can be stated over any constant sized field, and over the rationals.

A polynomial $p(x_1, x_2, \cdots, x_n, r_1, \cdots, r_k)$ is called a *probabilistic polynomial* if it has as input the standard input bits $x_1, x_2, \ldots, x_n$ and, in addition, random input bits $r_1, r_2, \ldots, r_k$. We say that the polynomial p *represents* a Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ with error ϵ if

$$\text{Prob}[p(x_1, \cdots, x_n, r_1, \cdots, r_k) = f(x_1, \cdots, x_n)] \geq 1 - \epsilon,$$

where the probability is over random choices of bits r_j.

Theorem 2.4.1. [Tar93, BRS95] *Every function f computed by a Boolean circuit of depth d and size s is represented by a probabilistic polynomial $p(x_1, x_2, \cdots, x_n, r_1, \cdots, r_k)$ of degree $O(\log(1/\epsilon)\log^2 n)^d$ that represents $f(x_1, \cdots, x_n)$ with error $s\epsilon$.* [3]

We now show that the help functions lower bound problem is connected to the Remote Point Problem (RPP) introduced by Alon et al. [APY09]. An interesting deterministic algorithm for the RPP is presented in [APY09]. A deterministic algorithm with considerably stronger parameters would solve our lower bound question. This connection is elaborated upon below. We first define the RPP formally.

Fix $N \in \mathbb{N}$ and functions $k, r : \mathbb{N} \to \mathbb{N}$. Given vectors $x, y \in \mathbb{F}_2^N$, we denote by $\Delta(x, y)$ their Hamming distance, that is, $\Delta(x, y) = |\{i \mid x_i \neq y_i\}|$. The $(k(N), r(N))$-RPP is defined as follows:

- INPUT: A subspace V of $\mathbb{F}_2^N$ of dimension at most $k(N)$ given by its generators.
- DESIRED OUTPUT: A point $x \in \mathbb{F}_2^N$ such that $\Delta(x, V) \geq r(N)$, where $\Delta(x, V) = \min_{v \in V} \Delta(x, v)$.

A randomized algorithm that simply picks v at random would be a good solution with high probability (for most parameters $k(N)$ and $r(N)$ of interest). The challenge is to give an efficient deterministic algorithm for the RPP. Alon et al. in [APY09] give, for any fixed $c > 0$, a polynomial-time deterministic algorithm to solve the $(k, N\frac{c \log k}{k})$-RPP. We now state and prove the connection between the RPP and our lower bound question.

To keep notation simple, we consider the (m, s, d)-Help functions problem, for the case when $m(n), s(n) = 2^{(\log n)^c}$, for some constant $c > 1$.

[3] Tarui's construction yields a probabilistic polynomial q with *integer* coefficients. We can obtain the desired polynomial p over $\mathbb{F}_2$ from q by reducing the coefficients modulo 2.

Theorem 2.4.2. *Let $N = 2^n$. For any constant depth $d \in \mathbb{N}$, and any constants $c, c_1, c_2 > 0$ such that $c_1 > (3d+1)\max\{c, c_2\}$, the following holds. Fix functions $k(N)$ and $r(N)$ such that $k(N) \geq 2^{(\log\log N)^{c_1}}$ and $r(N) \geq \frac{N}{2^{(\log\log N)^{c_2}}}$. We have the following.*

- *If the $(k(N), r(N))$-Remote Point Problem can be solved in deterministic time $2^{(\log N)^{O(1)}}$, then the $(m(n), s(n), d)$-help functions problem can be solved in deterministic time $2^{(\log N)^{O(1)}}$.*
- *If the $(k(N), r(N))$-Remote Point Problem lies in* FUnif–SizeDepth$(2^{(\log N)^{O(1)}}, (\log N)^{O(1)})$*, then the $(m(n), s(n), d)$-help functions problem lies in* FUnif–SizeDepth$(2^{(\log N)^{O(1)}}, (\log N)^{O(1)})$.

Proof. The proof is an easy application of Theorem 2.4.1. We use m and s to denote $m(n)$ and $s(n)$ respectively. Let $H = \{h_1, h_2, \ldots, h_m\}$. Consider a circuit C corresponding to the class $\text{SizeDepth}^n_H(s, d)$. To wit, the function it computes is $C(h_1(\overline{x}), h_2(\overline{x}), \cdots, h_m(\overline{x}))$, where C is depth-d, unbounded fanin and of size s. Now, for $\overline{x}$ picked uniformly at random from $\{0,1\}^n$ suppose the probability distribution of $(h_1(\overline{x}), h_2(\overline{x}), \cdots, h_m(\overline{x}))$ on the set $\{0,1\}^m$ is μ. By Theorem 2.4.1 there is a probabilistic polynomial $p(y_1, y_2, \cdots, y_m, r_1, r_2, \cdots, r_t)$ of degree $O(\log(1/\epsilon)\log^2 m)^d$ that represents $C(y_1, y_2, \cdots, y_m)$ with error $s\epsilon$. By a standard averaging argument it follows that we can fix the random bits $r_1, r_2, \cdots, r_t$ to get

$$\Pr_{\mu}[p(y_1, y_2, \cdots, y_m, r_1, r_2, \cdots, r_t) = C(y_1, y_2, \cdots, y_m)] \geq 1 - s\epsilon,$$

where $(y_1, y_2, \cdots, y_m)$ is picked according to distribution μ. But that is equivalent to

$$\text{Prob}[p(h_1(\overline{x}), \cdots, h_m(\overline{x}), r_1, r_2, \cdots, r_t) = C(h_1(\overline{x}), h_2(\overline{x}), \cdots, h_m(\overline{x}))] \geq 1 - s\epsilon, \qquad (2.1)$$

where $\overline{x}$ is picked uniformly at random from $\{0,1\}^n$.

Choose c_1' such that $\max\{c, c_2\} < c_1' < c_1/(3d+1)$. Let $\epsilon = \frac{1}{2^{(\log n)^{c_1'}}}$. Then the degree of p above is at most $(\log n)^{3c_1'd}$. We will consider Boolean functions on n bits as vectors in $\mathbb{F}_2^N$. Let H' be vectors that can be written as the pointwise product of at most $(\log n)^{3c_1'd}$ functions in H and let V be the subspace of $\mathbb{F}_2^N$ spanned by the vectors in H'. Note that the dimension of V is $m^{(\log n)^{3c_1'd}} < 2^{(\log n)^{c_1}} = k(N)$. More importantly, the subspace V contains all polynomials in the help functions of degree at most $O(\log(1/\epsilon)\log^2 m)^d$. Thus, by Inequality 2.1, every function $f \in \text{SizeDepth}^n_H(s, d)$ is at Hamming distance at most $Ns\epsilon$ from some element of V. Therefore, it follows that finding a vector $v \in \mathbb{F}_2^N$ that is r-far from V for $r = \frac{N}{2^{(\log n)^{c_1}}} > Ns\epsilon$ would give us an explicit Boolean function that is not in $\text{SizeDepth}^n_H(s, d)$.

Note that such a function is exactly a solution to the (k, r)-RPP on the input that consists of the vectors in H'. Under the assumption that the (k, r)-RPP can be solved

in deterministic time $2^{(\log N)^{O(1)}} = 2^{n^{O(1)}}$, this explicit function can be computed in deterministic time $2^{n^{O(1)}}$ and under the stronger assumption that the (k, r)-RPP lies in FUnif–SizeDepth$(2^{(\log N)^{O(1)}}, (\log N)^{O(1)})$, the explicit function can also be computed in this class. □

2.5 Parallel algorithms for the RPP

In the previous sections, we have proved that efficient solutions to the Remote Point Problem, with suitable parameters, would give us a solution to the Help functions problem, which in turn could separate EXP from the polynomial-time many-one closure of AC^0 or PSPACE from the logspace many-one closure of AC^0. While an efficient deterministic solution is sufficient for the former application, we needed an efficient *parallel* solution for the latter, i.e., an algorithm that could be implemented by a small polylog-depth Boolean circuit.

Before describing the results of this section, we recall some standard complexity theoretic notation. For any fixed $i \in \mathbb{N}$, the class of Boolean functions Unif–SizeDepth$(N^{O(1)}, (\log N)^i)$ (where N is the size of the input) is and polylogarithmic depth is denoted NC^i. Moreover, NC denotes $\bigcup_{i\in\mathbb{N}} NC^i$. Similarly, we use FNC^i to denote FUnif–SizeDepth$(N^{O(1)}, (\log N)^i)$ and FNC to denote $\bigcup_{i\in\mathbb{N}} FNC^i$. While P is normally taken to be the class of problems that can be solved efficiently, NC is considered to be the class of problems that can be nontrivially *parallelized*, i.e., the class of problems that can be solved significantly faster on a parallel computer than on a sequential computer. Similarly, FNC is the class of functions that have far better parallel algorithms than sequential ones.

We will abuse terminology and call an algorithm an FNC^i-algorithm (resp. FNC-algorithm) if it can be implemented by a polynomial-sized family of circuits of depth $(\log N)^i$ (resp. $(\log N)^{O(1)}$). For Boolean functions, we use the terms NC^i-algorithm and NC-algorithm respectively.

As mentioned above, Alon et al. [APY09] give a nice polynomial time-bounded algorithm for computing a $v \in \mathbb{F}_2^N$ that is $c \log N$-far from a given subspace L of dimension $N/2$ and c is a fixed constant. For L such that $\dim(L) = k < N/2$ they give a polynomial-time algorithm for computing a point $v \in \mathbb{F}_2^N$ that is $\frac{cN \log k}{k}$-far from L. In this section, we show how to achieve these parameters with an efficient parallel algorithm.

Another motivation for the results in this section is a more detailed study of the Remote Point Problem as an algorithmic question. We consider the RPP in the setting of arbitrary *groups* in place of the field $\mathbb{F}_2$. We note that the solution of Alon et al. works in the setting of arbitrary groups and our own works in the context of *Abelian* groups. We summarize the results of this section below.

1. The first question we address is whether we can give a deterministic parallel (i.e. FNC)

algorithm for the problem — Alon et al.'s algorithm is inherently sequential as it is based on the method of conditional probabilities and pessimistic estimators.

It turns out an element of an ε-bias space ([NN93, AGHP92]) for suitably chosen ε is a solution to the Remote Point Problem which gives us an FNC algorithm quite easily.

2. Since the RPP for $\mathbb{F}_2^N$ can be solved using small bias spaces which are also defined in the setting of arbitrary abelian groups [AMN98], it naturally leads us to address the problem in a more general group-theoretic setting.

In the generalization we study, we will replace $\mathbb{F}_2$ with an arbitrary fixed finite group $\mathcal{G}$ such that $|\mathcal{G}| \geq 2$. Hence we will have the N-fold product group $\mathcal{G}^N$ instead of the vector space $\mathbb{F}_2^N$.

Given elements $x = (x_1, x_2, \ldots, x_N), y = (y_1, y_2, \ldots, y_N)$ of $\mathcal{G}^N$, let $\Delta(x, y) = |\{i \mid x_i \neq y_i\}|$. I.e. $\Delta(x, y)$ is the *Hamming distance* between x and y. Furthermore, for $S \subseteq \mathcal{G}^N$, let $\Delta(x, S)$ denote $\min_{y \in S} \Delta(x, y)$.

We now define the *Remote Point Problem (RPP) over a finite group* $\mathcal{G}$. The input is a subgroup $\mathcal{H}$ of $\mathcal{G}^N$, where $\mathcal{H}$ is given by a generating set, and a number $r \in [N]$. The problem is to compute efficiently an element $x \in \mathcal{G}^n$ such that $\Delta(x, H) > r$. The results we show in this general setting are the following.

(a) The Remote Point Problem over any *Abelian group* $\mathcal{G}$ has an FNC^2 algorithm for $r = O(\frac{N \log k}{k})$ and $k \leq N/2$, where $k = \log_{|\mathcal{G}|} |\mathcal{H}|$.

(b) Over an arbitrary group $\mathcal{G}$ the Remote point problem has a polynomial-time algorithm for $r = O(\frac{N \log k}{k})$ and $k \leq N/2$, where $k = \log_{|\mathcal{G}|} |\mathcal{H}|$.

The parallel algorithm stated in part(a) above is based on ε-bias space constructions for finite Abelian groups due to Azar, Motwani, and Naor [AMN98]. The sequential algorithm stated in part(b) above is a group-theoretic generalization of the Alon et al algorithm for $\mathbb{F}_2^N$ [APY09].

2.5.1 Preliminaries

Fix a finite group $\mathcal{G}$ such that $|\mathcal{G}| \geq 2$. Given any $x \in \mathcal{G}^n$, let $wt(x)$ denote the number of coordinates i such that $x_i \neq 1$, where 1 is the identity of the group $\mathcal{G}$. By $B(r)$, we will refer to the set of $x \in \mathcal{G}^n$ such that $wt(x) \leq r$. Given a subset S of $\mathcal{G}^n$, $B(S, r)$ will denote the set $S \cdot B(r) = \{sx \mid s \in S, x \in B(r)\}$. Clearly, for any $S \subseteq \mathcal{G}^n$ and any $x \in \mathcal{G}^n$, $x \in B(S, r)$ if and only if $\Delta(x, S) \leq r$. We say that x is *r-close* to S if $x \in B(S, r)$ and *r-far* from S if $x \notin B(S, r)$.

Given a subgroup $\mathcal{H}$ of $\mathcal{G}^n$, denote by $\delta(\mathcal{H})$ the quantity $\log_{|\mathcal{G}|} |\mathcal{H}|$. We will call $\delta(\mathcal{H})$ the *dimension of* $\mathcal{H}$ *in* $\mathcal{G}^n$. This notion is analogous to the dimension of a subspace V of the vector space $\mathbb{F}_2^N$.

Fix functions $k, r : \mathbb{N} \to \mathbb{N}$. The $(k(N), r(N))$-*Remote Point Problem (RPP) over* $\mathcal{G}$ is defined to be the following algorithmic problem:

INPUT: A subgroup $\mathcal{H}$ of $\mathcal{G}^N$ (given by its generators) of dimension at most $k(N)$.

OUTPUT: An $x \in \mathcal{G}^N$ such that $x \notin B(\mathcal{H}, r(N))$.

Clearly, there are inputs to the above problem where no solution can be found. But the input instances of the kind that we will study will clearly have a solution (in fact, a random point of $\mathcal{G}^n$ will be a solution with high probability).

A simple counting argument shows that there is a valid solution to the (k, r)-RPP over $\mathcal{G}$ on input subgroups $\mathcal{H}$ where $\delta(\mathcal{H}) + r \leq N(1 - \frac{H(r/N)}{\log |\mathcal{G}|} - \varepsilon)$, for any fixed $\varepsilon > 0$ (where $H(\cdot)$ denotes the binary entropy function). However, the best known deterministic solution to the RPP – from [APY09] – is a polynomial time $(k, \frac{cN \log k}{k})$-algorithm which works over $\mathbb{F}_2^N$ (i.e, the group $\mathcal{G}$ involved is the additive group of the field $\mathbb{F}_2$).

Some Group-Theoretic Algorithms

We introduce basic definitions and review some group-theoretic algorithms. Let $\mathrm{Sym}(\Omega)$ denote the group of all permutations on a finite set Ω of size m. In this section we use G, H etc. to denote *permutation groups on* Ω, which are simply subgroups of $\mathrm{Sym}(\Omega)$.

Let G be a subgroup of $\mathrm{Sym}(\Omega)$. For a subset $\Delta \subseteq \Omega$ denote by $G_{\{\Delta\}}$ the *point-wise stabilizer* of Δ. I.e $G_{\{\Delta\}}$ is the subgroup consisting of exactly those elements of G that fix each element of Δ.

Theorem 2.5.1 (Schreier-Sims)**.** [Luk93]

1. *If a subgroup G of* $\mathrm{Sym}(\Omega)$ *is given by a generating set as input along with the subset Δ there is a polynomial-time (sequential) algorithm for computing a generator set for $G_{\{\Delta\}}$.*
2. *If a subgroup G of* $\mathrm{Sym}(\Omega)$ *is given by a generating set as input, then there is a polynomial time algorithm for computing $|G|$.*
3. *Given as input a permutation $\sigma \in \mathrm{Sym}(\Omega)$ and a generator set for a subgroup G of* $\mathrm{Sym}(\Omega)$*, we can test in deterministic polynomial time if σ is an element of G.*

We are also interested in a special case of this problem which we now define. A subset $\Gamma \subseteq \Omega$ is an *orbit* of G if $\Gamma = \{\sigma(i) \mid \sigma \in G\}$ for some $i \in \Omega$. Any subgroup G of $\mathrm{Sym}(\Omega)$ partitions Ω into orbits (called G-orbits).

For a constant $b > 0$, a subgroup G of $\mathrm{Sym}(\Omega)$ is defined to be a *b-bounded permutation group* if every G-orbit is of size at most b.

In [MC87], McKenzie and Cook studied the parallel complexity of *Abelian* permutation group problems. Specifically, they gave an NC^3 algorithm for testing membership in an Abelian permutation group given by a generator set and for computing the order of an Abelian permutation group. When restricted to b-bounded Abelian permutation groups, the algorithms of [MC87] for these problems are actually NC^2 algorithms. We formally state their result and derive a consequence.

Theorem 2.5.2 ([MC87])**.** *There is an* NC^2 *algorithm for membership testing in a b-bounded Abelian permutation group G given by a generator set.*

We now consider problems over $\mathcal{G}^N$, for a fixed finite group $\mathcal{G}$. We know from basic group theory that every group $\mathcal{G}$ is a permutation group acting on itself. I.e. every $\mathcal{G}$ can be seen as a subgroup of $\mathrm{Sym}(\mathcal{G})$, where $\mathcal{G}$ acts on itself by left (or right) multiplication. Therefore, $\mathcal{G}^N$ can be easily seen as a permutation group on the set $\Omega = \mathcal{G} \times [N]$ and hence, $\mathcal{G}^N$ can be considered a subgroup of $\mathrm{Sym}(\Omega)$. Furthermore, notice that each subset $\mathcal{G} \times \{i\}$ is an orbit of this group $\mathcal{G}^N$. Hence, $\mathcal{G}^N$ is a b-bounded permutation group contained in $\mathrm{Sym}(\Omega)$, where $b = |\mathcal{G}|$. Finally, if $\mathcal{G}$ is an Abelian group, then so is this subgroup of $\mathrm{Sym}(\Omega)$. We have the following lemma as an easy consequence of Theorem 2.5.2.

Lemma 2.5.3. *Let $\mathcal{G}$ be Abelian. There is an* NC^2 *algorithm that takes as input a generator set for some subgroup $\mathcal{H}$ of $\mathcal{G}^N$ and an $x \in \mathcal{G}^N$, and accepts iff $x \in \mathcal{H}$.*

Given any $y = (y_1, y_2, \ldots, y_i) \in \mathcal{G}^i$ with $1 \leq i \leq N$ and any $S \subseteq \mathcal{G}^N$, let S_y denote the set $\{x \in S \mid x_j = y_j \text{ for } 1 \leq j \leq i\}$.

Lemma 2.5.4. *Let $\mathcal{G}$ be any fixed finite group. There is a polynomial time algorithm that takes as input a subgroup $\mathcal{H}$ of $\mathcal{G}^N$, where $\mathcal{H}$ is given by generators, and a $y \in \mathcal{G}^i$ with $1 \leq i \leq N$, and computes $|\mathcal{H}_y|$.*

Proof. Let $\mathcal{K} = \{(x_1, x_2, \ldots, x_N) \in \mathcal{H} \mid x_1 = x_2 = \cdots = x_i = 1\}$, where 1 denotes the identity element of $\mathcal{G}$. Clearly, $\mathcal{K}$ is a subgroup of $\mathcal{H}$. The set $\mathcal{H}_y$, if nonempty, is simply a coset of $\mathcal{K}$ and thus, we have $|\mathcal{H}_y| = |\mathcal{K}|$. To check if $\mathcal{H}_y$ is nonempty, we consider the map $\pi_i : \mathcal{G}^N \to \mathcal{G}^i$ that projects its input onto its first i coordinates; note that $\mathcal{H}_y$ is nonempty iff the subgroup $\pi_i(\mathcal{H})$ contains y, which can be checked in polynomial time by point (3) of Theorem 2.5.1 (here, we are identifying $\mathcal{G}^N$ with a subgroup of $\mathrm{Sym}(\mathcal{G} \times [N])$ as above). If $y \notin \pi_i(\mathcal{H})$, the algorithm outputs 0. Otherwise, we have $|\mathcal{H}_y| = |\mathcal{K}|$ and it suffices to compute $|\mathcal{K}|$. But $\mathcal{K}$ is simply the point-wise stabilizer of the set $\mathcal{G} \times [i]$ in $\mathcal{H}$, and hence $|\mathcal{K}|$ can be computed in polynomial time by points (1) and (2) of Theorem 2.5.1. □

2.5.2 Expanding Cayley Graphs and the Remote Point Problem

Fix a group $\mathcal{G}$ such that $|\mathcal{G}| \geq 2$, and consider an instance of the RPP over $\mathcal{G}$. The main idea that we develop in this section is that if we have a (symmetric) expanding generator set S for the group $\mathcal{G}^N$ with appropriate expansion parameters then for a subgroup $\mathcal{H}$ of $\mathcal{G}^N$ such that $\delta(\mathcal{H}) \leq k$ some element of S will be r-far from H, for suitable k and r.

We review some definitions related to expander graphs (e.g. see the survey of Hoory, Linial, and Wigderson [HLW06]). An undirected multigraph $G = (V, E)$ is an (n, d, α)-graph for $n, d \in \mathbb{N}$ and $\alpha > 0$ if $|V| = n$, the degree of each vertex is d, and the second largest value $\lambda(G)$ from among the absolute values of eigenvalues of $A(G)$ – the adjacency matrix of the graph G – is bounded by αd.

A *random walk* of length $t \in \mathbb{N}$ on an (n, d, α)-graph $G = (V, E)$ is the output of the following random process: a vertex $v_0 \in V$ of picked uniformly at random, and for $0 \leq i < t$, if v_i has been picked, then v_{i+1} is obtained by selecting a neighbour v_{i+1} uniformly at random (i.e a random edge out of v_i is picked, and v_{i+1} is chosen to be the other endpoint of the edge); the output of the process is $(v_0, v_1, \ldots, v_t)$. We now state an important result regarding random walks on expanders (see [HLW06, Theorem 3.6] for details).

Lemma 2.5.5. *Let $G = (V, E)$ be an (n, d, α)-graph and $B \subseteq V$ with $|B| \leq \beta n$. Then, the probability that a random walk $(v_0, v_1, \ldots, v_t)$ is entirely contained inside B (i.e, $v_i \in B$ for each i) is bounded by $(\beta + \alpha)^t$.*

Let $\mathcal{H}$ be a group and S a *symmetric* multiset of elements from $\mathcal{H}$. I.e. there is a bijection of multisets $\varphi : S \to S$ such that $\varphi(s) = s^{-1}$ for each $s \in S$. We define the Cayley graph $C(\mathcal{H}, S)$ to be the (multi)graph G with vertex set $\mathcal{H}$ and edges of the form (x, xs) for each $x \in \mathcal{H}$ and each $s \in S$; since S is symmetric, we consider $C(\mathcal{H}, S)$ to be an undirected graph by identifying the edges (x, xs) and $(xs, (xs)\varphi(s))$, for each x and s.

We now show a lemma that will help relate generators of expanding Cayley graphs on $\mathcal{G}^n$ and the RPP over $\mathcal{G}$. In what follows, let S be a symmetric multiset of elements from $\mathcal{G}^N$; let G denote the Cayley graph $C(\mathcal{G}^N, S)$.

Lemma 2.5.6. *Assume S as above is such that G is an $(|\mathcal{G}|^N, |S|, \alpha)$-graph, where $\alpha \leq \frac{1}{N^d}$, for some fixed $d > 0$. Then, given any subgroup $\mathcal{H}$ of $\mathcal{G}^N$ such that $\delta(\mathcal{H}) \leq 2N/3$, we have $\frac{|S \cap \mathcal{H}|}{|S|} \leq \frac{1}{N^{d/2}}$ for large enough N (where the elements of $S \cap \mathcal{H}$ are counted with repetitions).*

Proof. Let $S' = S \cap \mathcal{H}$ and let $\eta = |S'|/|S|$. We want an upper bound on η. Consider a random walk $(x_0, x_1, \ldots, x_t)$ of length t on the graph G (the exact value of t will be fixed later). Let $\mathcal{B}$ denote the following event: there is a $y \in \mathcal{G}^N$ such that all the vertices $x_0, x_1, \ldots, x_t$ are all contained in the coset $y\mathcal{H}$ of $\mathcal{H}$. Let p denote the probability that $\mathcal{B}$ occurs.

We will first lower bound p. At each step of the random walk, a random $s_i \in S$ is chosen and x_{i+1} is set to $x_i s_i$. If these s_i all happen to belong to S', then the cosets $x_i\mathcal{H}$ and $x_{i+1}\mathcal{H}$ are the same for all i and hence, the event $\mathcal{B}$ does occur. Hence, $p \geq \eta^t$.

We now upper bound p. Fix any coset $y\mathcal{H}$ of the subgroup $\mathcal{H}$. Since the dimension of $\mathcal{H}$ in $\mathcal{G}^N$ is bounded by $2N/3$, we have $|y\mathcal{H}| = |\mathcal{H}| \leq |\mathcal{G}|^{2N/3} \leq 2^{-N/3}|\mathcal{G}^N|$. That is, the coset $y\mathcal{H}$ is a very small subset of $\mathcal{G}^N$. Applying Lemma 2.5.5, we see that the probability that the random walk $(x_0, x_1, \ldots, x_t)$ is completely contained inside this coset is bounded by $(2^{-N/3} + N^{-d})^t \leq \frac{2^t}{N^{dt}}$, for large enough N. As the total number of cosets of $\mathcal{H}$ is bounded by $|\mathcal{G}|^N$, an application of the union bound tells us that p is upper bounded by $|\mathcal{G}|^N \frac{2^t}{N^{dt}} \leq \frac{|\mathcal{G}|^{N+t}}{N^{dt}}$. Setting $t = \frac{2N}{d\log_{|G|} N - 2}$ we see that p is at most $\frac{1}{N^{dt/2}}$.

Putting the upper and lower bounds together, we see that $\eta^t \leq \frac{1}{N^{dt/2}}$ and hence, $\eta \leq \frac{1}{N^{d/2}}$. This completes the proof. □

We follow the structure of the algorithm for the RPP over $\mathbb{F}_2$ in [APY09]. We first describe their algorithm for the $(N/2, c\log N)$-RPP, followed by our own algorithm for this problem. We then describe how they extend this algorithm to one for the $(k, \frac{cN\log k}{k})$-RPP for any $k \leq N/2$; the same procedure works for our algorithm also.

The algorithm for the $(N/2, c\log N)$-RPP proceeds as follows. On an input instance consisting of a subgroup V (which is a subspace of $\mathbb{F}_2^N$) of dimension at most $N/2$,

1. The algorithm first computes a collection of $m = N^{O(c)}$ subspaces $V_1, V_2, \ldots, V_m$, each of dimension at most $2N/3$ such that $B(V, c\log N) \subseteq \bigcup_{i=1}^m V_i$.

2. The algorithm then finds an $x \in \mathbb{F}_2^N$ such that $x \notin \bigcup_i V_i$. (This is done using a method similar to the method of pessimistic estimators introduced by Raghavan [Rag88].)

Our algorithm will proceed exactly as the above algorithm in the first step. The second step of our algorithm will be different (assuming that the group $\mathcal{G}$ is Abelian). We first state Step 1 of the algorithm of [APY09] in greater generality:

Lemma 2.5.7. *Let $\mathcal{G}$ be any fixed finite group with $|\mathcal{G}| \geq 2$. For any constant $c > 0$ and large enough N, the following holds. Given any subgroup $\mathcal{H}$ of $\mathcal{G}^N$ such that $\delta(\mathcal{H}) \leq \frac{N}{2}$, there is a collection of $m \leq N^{10c}$ subgroups $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ such that $B(\mathcal{H}, c\log N) \subseteq \bigcup_{i=1}^m \mathcal{H}_i$, and $\delta(\mathcal{H}_i) \leq 2N/3$ for each i. Moreover, there is a logspace algorithm that, when given as input $\mathcal{H}$ as a set of generators, produces generators for the subgroups $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$.*

Proof. The proof follows exactly as in [APY09]. We reproduce it here for completeness and to analyze the complexity of the procedure.

Let 1 denote the identity element of $\mathcal{G}$. For each $S \subseteq [N]$, let $\mathcal{G}^N(S)$ denote the subgroup of $\mathcal{G}^N$ consisting of those x such that $x_i = 1$ for each $i \notin S$. Note that $\delta(\mathcal{G}^N(S)) = |S|$. Also note that for each $S \subseteq [N]$, the group $\mathcal{G}^N(S)$ is a normal subgroup; in particular, this implies that the set $\mathcal{K} \cdot \mathcal{G}^N(S)$ is a subgroup of $\mathcal{G}^N$ whenever $\mathcal{K}$ is a subgroup of $\mathcal{G}^N$.

Partition the set $[N]$ into $\ell \leq 10c \log N$ sets of size at most $\lceil \frac{N}{10c \log N} \rceil$ each – we will call these sets $S_1, S_2, \ldots, S_\ell$. For each $A \subseteq [\ell]$ of size $\lceil c \log N \rceil$, let $\mathcal{K}_A$ denote the subgroup $\mathcal{G}^N(\bigcup_{i \in A} S_i)$. Note that the number of such subgroups is at most $2^\ell \leq N^{10c}$. Also, for each A as above, $\delta(\mathcal{K}_A) = |\bigcup_{i \in A} S_i| \leq \left(\frac{N}{10c \log N} + 1 \right) (c \log N + 1) < \frac{N}{9}$, for large enough N.

Consider any $x \in B(c \log N)$ (i.e, an element x of $\mathcal{G}^N$ s.t $wt(x) \leq c \log N$). We know that $x \in \mathcal{G}^N(S)$ for some S of size at most $c \log N$. Hence, it can be seen that $x \in \mathcal{G}^N(\bigcup_{i \in A} S_i)$ for some A of size $\lceil c \log N \rceil$; this shows that $B(c \log N) \subseteq \bigcup_A \mathcal{K}_A$. Therefore, we see that $B(\mathcal{H}, c \log N) = \mathcal{H} B(c \log N) \subseteq \bigcup_A \mathcal{H}\mathcal{K}_A$.

For each $A \subseteq [\ell]$ of size $\lceil c \log n \rceil$, let $\mathcal{H}_A$ denote the subgroup $\mathcal{H}\mathcal{K}_A$ (note that this is indeed a subgroup, since $\mathcal{K}_A$ is a normal subgroup). Moreover, the cardinality of this subgroup is bounded by $|\mathcal{H}| \cdot |\mathcal{K}_A| \leq |\mathcal{G}|^{N/2} |\mathcal{G}|^{N/9} < |\mathcal{G}|^{2N/3}$; hence, $\delta(\mathcal{H}_A) \leq 2N/3$. Thus, the collection of subgroups $\{\mathcal{H}_A\}_A$ satisfies all the properties mentioned in the statement of the lemma. That a set of generators for this subgroup can be computed in deterministic logspace – for some suitable choice of $S_1, S_2, \ldots, S_\ell$ – is a routine check from the definition of the subgroups $\{\mathcal{K}_A\}_A$. This completes the proof of the lemma. □

Using Lemma 2.5.7, we are able to efficiently "cover" $B(\mathcal{H}, c \log N)$ for any small subgroup $\mathcal{H}$ of $\mathcal{G}^N$ by a union of small subgroups. Therefore, to find a point that is $c \log N$-far from $\mathcal{H}$, it suffices to find a point $x \in \mathcal{G}^N$ not contained in any of the covering subgroups. To do this, we note that if S is a multiset containing elements from $\mathcal{G}^N$ such that $C(\mathcal{G}^N, S)$ is a Cayley graph with good expansion, then S must contain such an element. This is formally stated below.

Lemma 2.5.8. *For any constant $c > 0$ and large enough $N \in \mathbb{N}$, the following holds. Let S be any multiset of elements of $\mathcal{G}^N$ such that $\lambda(C(\mathcal{G}^N, S)) < \frac{1}{N^{20c}}$. Then, for $m \leq N^{10c}$ and any collection $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ of subgroups such that $\delta(\mathcal{H}_i) \leq 2N/3$ for each i, there is some $s \in S$ such that $s \notin \bigcup_i \mathcal{H}_i$.*

Proof. The proof follows easily from Lemma 2.5.6. Given any $i \in [m]$, we know, from Lemma 2.5.6, that $|S \cap \mathcal{H}_i| < \frac{|S|}{N^{-10c}}$ (where the elements of the multisets are counted with repetitions). Hence, $|S \cap \bigcup_i \mathcal{H}_i| \leq \sum_i |S \cap \mathcal{H}_i| < \frac{m|S|}{N^{-10c}} \leq |S|$. Therefore, there must be some $s \in S$ such that $s \notin \bigcup_i \mathcal{H}_i$. □

Therefore, to find a point x that is $c \log N$-far from the subspace $\mathcal{H}$, it suffices to construct an S such that $C(\mathcal{G}^N, S)$ is a sufficiently good expander, find the covering subgroups $\mathcal{H}_i$

$(i \in [m])$, and then to find an $s \in S$ that does not lie in any of the $\mathcal{H}_i$. We follow the above approach to give an efficient parallel algorithm for the RPP in the case that $\mathcal{G}$ is an Abelian group. For arbitrary groups, we show that the method of [APY09] yields a polynomial time algorithm.

2.5.3 Remote Point Problem for Abelian Groups

Fix an Abelian group $\mathcal{G}$. Recall that a *character* χ of $\mathcal{G}^N$ is a homomorphism from $\mathcal{G}^N$ to $\mathbb{C}_1^*$, the multiplicative subgroup of the complex numbers of absolute value 1. For $\varepsilon > 0$, a distribution μ over $\mathcal{G}^N$ is said to be ε-biased if, given any non-trivial character χ of $\mathcal{G}^N$,

$$\left| \mathop{\mathbf{E}}_{x \sim \mu} [\chi(x)] \right| \leq \varepsilon$$

A multiset S consisting of elements from $\mathcal{G}^N$ is said to be an *ε-biased space in $\mathcal{G}^N$* if the uniform distribution over S is an ε-biased distribution.

It can be checked that a multiset consisting of $(\frac{n}{\varepsilon})^{O(1)}$ independent, uniformly random elements from $\mathcal{G}^N$ form an ε-biased space with high probability. Explicit ε-biased spaces were constructed for the group $\mathbb{F}_2^N$ by Naor and Naor in [NN93]; further constructions were given by Alon et al. in [AGHP92]. Explicit constructions of ε-biased spaces in $\mathbb{Z}_d^N$ were given by Azar et al. in [AMN98]. We observe that this last construction yields a construction for all Abelian groups $\mathcal{G}^N$, when $\mathcal{G}$ is of constant size. We first state the result of [AMN98] in a form that we will find suitable.

Theorem 2.5.9. *For any fixed d, there is an* FNC^2 *algorithm that does the following. On input N and $\varepsilon > 0$ (both in unary), the algorithm produces a symmetric multiset $S \subseteq \mathbb{Z}_d^n$ of size $O((\frac{N}{\varepsilon})^2)$ such that S is an ε-biased space in $\mathbb{Z}_d^N$.*

Proof. It is easy to see that the ε-biased space construction in [AMN98] can be implemented in deterministic logspace (and hence in FNC^2). If the space S obtained is not symmetric, we can consider the multiset that is the disjoint union of S and S^{-1}, which is also easily seen to be ε-biased. □

Remark 2.5.10. *We note that the definition of small bias spaces in [AMN98] differs somewhat from our own definition above. But it is easy to see that an ε-bias space in $\mathbb{Z}_d^N$ in the sense of [AMN98] is a $(d\varepsilon)$-bias space according to our definition above.*

Remark 2.5.11. *In a recent paper, Meka and Zuckerman [MZ09] observe, as we do below, that the construction of [AMN98] gives small bias spaces for any arbitrary Abelian group $\mathcal{G}$. Nevertheless, we present our own proof of this fact, since the small bias spaces that follow from our proof are of* smaller *size. Specifically, our proof shows how to explicitly construct*

sample spaces of size $O\left(\frac{N^2}{\varepsilon^2}\right)$*, whereas the relevant result in [MZ09] only produces small bias spaces of size* $O\left((\frac{N}{\varepsilon})^b\right)$*, where* b *is some constant that depends on* $\mathcal{G}$ *(and can be as large as* $\Omega(\log|\mathcal{G}|)$*).*

Lemma 2.5.12. *For any fixed group* $\mathcal{G}$*, there is an* FNC^2 *algorithm which, on input* n *and* $\varepsilon > 0$ *in unary, produces a symmetric multiset* $S \subseteq \mathcal{G}^N$ *of size* $O((\frac{N}{\varepsilon})^2)$ *such that* S *is an* ε*-biased space in* $\mathcal{G}^N$*.*

Proof. By the Fundamental Theorem of finite Abelian groups, $\mathcal{G} \cong \mathbb{Z}_{d_1} \oplus \mathbb{Z}_{d_2} \oplus \cdots \oplus \mathbb{Z}_{d_k}$, for positive integers $d_1, d_2, \ldots, d_k$ such that $d_1 \mid d_2 \mid \cdots \mid d_k$. Let $\mathcal{G}_0$ denote $\mathbb{Z}_{d_k}^k$. Note that for any $s, t \in \mathbb{N}$, $\mathbb{Z}_s \cong \mathbb{Z}_{st}/\mathbb{Z}_t$. Hence, we see that that $\mathcal{G} \cong \mathcal{G}_0/\mathcal{H}$, where $\mathcal{H}$ is the subgroup $\mathbb{Z}_{e_1} \oplus \mathbb{Z}_{e_2} \oplus \cdots \oplus \mathbb{Z}_{e_k}$, and $e_i = d_k/d_i$ for each $i \in [k]$. Therefore, $\mathcal{G}^N \cong \mathcal{G}_0^N/\mathcal{H}^N$. Let $\pi : \mathcal{G}_0^N \to \mathcal{G}^N$ be the natural onto homomorphism with kernel $\mathcal{H}^N$. Note that π is just the projection map and can easily be computed in FNC^2.

Since $\mathcal{G}_0^N \cong \mathbb{Z}_{d_k}^{Nk}$, by Theorem 2.5.9, there is an FNC^2 algorithm that constructs a symmetric multiset $S_0 \subseteq \mathcal{G}_0^N$ of size $O(\left(\frac{kN}{\varepsilon}\right)^2)$ such that S_0 is an ε-biased space in $\mathcal{G}_0^n$. We claim that the multiset $S = \pi(S_0)$ is a symmetric ε-biased space in $\mathcal{G}^N$. To see this, consider any non-trivial character χ of $\mathcal{G}^N$; note that $\chi_0 = \chi \circ \pi$ is a non-trivial character of $\mathcal{G}_0^N$. We have

$$\left|\mathop{\mathbf{E}}_{x\sim S}[\chi(x)]\right| = \left|\mathop{\mathbf{E}}_{x_0\sim S_0}[\chi(\pi(x_0))]\right| = \left|\mathop{\mathbf{E}}_{x_0\sim S_0}[\chi_0(x)]\right| \leq \varepsilon$$

where the first equality follows from the definition of S, and the last inequality follows from the fact that S_0 is an ε-biased space in $\mathcal{G}_0^N$. Since χ was an arbitrary non-trivial character of $\mathcal{G}^N$, we have proved that S is indeed an ε-biased space in $\mathcal{G}^N$. It is easy to see that S is symmetric. Finally, note that S can be computed in FNC^2. This completes the proof. □

Finally, we mention a well-known connection between small bias spaces in $\mathcal{G}^N$ and Cayley graphs over $\mathcal{G}^N$ (e.g. see Alon and Roichman [AR94]).

Lemma 2.5.13. *Given any symmetric multiset* $S \subseteq \mathcal{G}^N$*, the Cayley graph* $C(\mathcal{G}^N, S)$ *is an* $(|\mathcal{G}|^N, |S|, \alpha)$*-graph iff* S *is an* α*-biased space.*

Lemmas 2.5.13 and 2.5.12 have the following easy consequence:

Lemma 2.5.14. *For any Abelian group* $\mathcal{G}$*, there is an* FNC^2 *algorithm which, on unary inputs* N *and* $\alpha > 0$*, produces a symmetric multiset* $S \subseteq \mathcal{G}^N$ *of size* $O((\frac{N}{\alpha})^2)$ *such that* $C(\mathcal{G}^N, S)$ *is a* $(|\mathcal{G}|^N, |S|, \alpha)$*-graph.*

Putting the above statement together with the results of Section 2.5.2, we have the following.

Theorem 2.5.15. *For any constant* $c > 0$*, the* $(N/2, c\log N)$*-RPP over* $\mathcal{G}$ *has an* FNC^2 *algorithm.*

Proof. Let $\mathcal{H}$ denote the input subgroup. By Lemma 2.5.7, there is a logspace (and hence NC^2) algorithm that computes a collection of $m = N^{O(c)}$ many subgroups $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ such that $B(\mathcal{H}, c \log N) \subseteq \bigcup_{i=1}^{m} \mathcal{H}_i$ and $\delta(\mathcal{H}_i) \leq 2N/3$ for each $i \in [m]$. Now, fix any multiset $S \subseteq \mathcal{G}^N$ such that the Cayley graph $C(\mathcal{G}^N, S)$ is a $(|\mathcal{G}|^N, |S|, \alpha)$-graph, where $\alpha = \frac{1}{2N^{20c}}$; by Lemma 2.5.14, such an S can be constructed in FNC^2. It follows from Lemma 2.5.8 that there is some $s \in S$ such that $s \notin \bigcup_{i=1}^{m} \mathcal{H}_i$. Finally, by Lemma 2.5.3, there is an NC^2 algorithm to test if each $s \in S$ belongs to $\mathcal{H}_i$, for any $i \in [m]$. Hence, we can find out (in parallel) exactly which $s \in S$ do not belong to any of the $\mathcal{H}_i$ and output one of them. The output element s is surely $c \log N$-far from $\mathcal{H}$. □

Let $\mathcal{G}$ be Abelian. We observe that a method of [APY09], coupled with Theorem 2.5.15, yields an efficient $(k, \frac{cN \log k}{k})$-algorithm for any constant $c > 0$, and $k \leq N/2$.

Theorem 2.5.16. *Let $c > 0$ be any constant. If $\mathcal{G}$ is an Abelian group, then the $(k, \frac{cN \log k}{k})$-RPP over $\mathcal{G}$ has an FNC^2 algorithm for any $k \leq N/2$.*

Proof. Given as input a subgroup $\mathcal{H}$ such that $\delta(\mathcal{H}) = k \leq N/2$, the algorithm partitions $[N]$ as $[N] = \bigcup_{i=1}^{m} T_i$, where $2k \leq |T_i| < 4k$ for each i; note that $m \geq N/4k$. Let $\mathcal{H}_i$ denote the subgroup obtained when $\mathcal{H}$ is projected onto the coordinates in T_i. Since $\delta(\mathcal{H}_i) \leq k \leq |T_i|/2$, we can, by Theorem 2.5.15, efficiently find a point $x_i \in \mathcal{G}^{|T_i|}$ that is at least $4c \log k$-far from $\mathcal{H}_i$. Putting these x_i together in the natural way, we obtain an $x \in \mathcal{G}^N$ that is $\frac{cN \log k}{k}$-far from the subgroup $\mathcal{H}$.

Since $\mathcal{G}$ is Abelian, using the algorithm of Theorem 2.5.15, the x_i can all be computed in parallel in FNC^2. Hence, the above is an FNC^2-procedure. □

2.5.4 RPP over General Groups

Let $\mathcal{G}$ denote some fixed finite group. We now generalize the polynomial-time algorithm of [APY09], described for $\mathbb{F}_2$, to compute a point $x \in \mathcal{G}^N$ that is $c \log N$-far from a given input subgroup $\mathcal{H}$ such that $\delta(\mathcal{H}) \leq N/2$.

Theorem 2.5.17. *For any constant $c > 0$, the $(N/2, c \log N)$-RPP over $\mathcal{G}$ has a polynomial time algorithm.*

Proof. The algorithm we describe will work for N larger than a suitable constant. For smaller N, we can solve the problem using a brute-force search algorithm. Given as input a subgroup $\mathcal{H}$ of $\mathcal{G}^n$ such that $\delta(\mathcal{H}) \leq N/2$ and $r \leq c \log N$, we first compute a collection of $m = N^{O(c)}$ subgroups $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m$ such that $B(\mathcal{H}, r) \subseteq \bigcup_{i=1}^{m} \mathcal{H}_i$ and $\delta(\mathcal{H}_i) \leq 2N/3$ for each i. By Theorem 2.5.7, such a collection of subgroups can be computed even in deterministic logspace, and hence in polynomial time. Our aim is to pick a point $x \in \mathcal{G}^N$ such that $x \notin \bigcup_{i=1}^{m} \mathcal{H}_i$.

We will pick the components $x_1, x_2, \ldots, x_N$ of x from $\mathcal{G}$, successively in the order of the indices, maintaining the invariant that after $x_1, x_2, \ldots, x_j$ have been picked, we have $\sum_{i=1}^{m} |\mathcal{H}_{ij}| < |\mathcal{G}|^{N-j}$, where $\mathcal{H}_{ij} = \{y \in \mathcal{H}_i \mid y_k = x_k \ \forall k \leq j\}$. For N larger than a constant depending on $\mathcal{G}$ and c, the invariant clearly holds before x_1 is picked. Furthermore, if the invariant holds after all of $x_1, x_2, \ldots, x_N$ have been fixed, then the resulting point $x \in \mathcal{G}^N$ does not belong to $\mathcal{H}_i$ for any i.

Suppose $x_1, x_2, \ldots, x_j$ have been picked for some $j \in \{0, 1, \ldots, N-1\}$. We pick x_{j+1} as follows. For each $g \in \mathcal{G}$, let $\mathcal{H}_{ijg} = \{y \in \mathcal{H}_{ij} \mid y_{j+1} = g\}$. By Lemma 2.5.4, we can compute $|\mathcal{H}_{ijg}|$ for each $g \in G$ in polynomial time. Moreover, since $|\mathcal{G}|^{N-j} > \sum_i |\mathcal{H}_{ij}| = \sum_{i,g} |\mathcal{H}_{ijg}|$, there must be some $g_0 \in G$ such that $\sum_i |\mathcal{H}_{ijg_0}| < |\mathcal{G}|^{N-j-1}$. Setting x_{j+1} to be g_0, we are done.

The correctness of the algorithm is clear from the invariant maintained across iterations. That the algorithm runs in polynomial time is obvious. □

Analogous to Theorem 2.5.16, we have the following solution to RPP for general groups.

Theorem 2.5.18. *Let $c > 0$ be any constant. For any $\mathcal{G}$, the RPP over $\mathcal{G}$ has a polynomial time $(k, \frac{cN \log k}{k})$-algorithm for any $k \leq N/2$.*

Proof. The construction is exactly the same as in the proof of Theorem 2.5.16. The only difference is that we will apply the algorithm of Theorem 2.5.17. In this case, the x_i can all be found in deterministic polynomial time. Hence, the entire procedure gives us a polynomial-time algorithm. □

2.5.5 Limitations of expanding sets

In the previous sections, we have shown how generators for expanding Cayley graphs on $\mathcal{G}^N$, where $\mathcal{G}$ is a fixed finite group, can help solve the RPP over $\mathcal{G}$. In particular, we have the following easy consequence of Lemmas 2.5.7 and 2.5.8.

Corollary 2.5.19. *For any constant $c > 0$, large enough N, and any symmetric multiset $S \subseteq \mathcal{G}^N$ such that $\lambda(C(\mathcal{G}^N, S)) < \frac{1}{N^{20c}}$, the following holds. If $\mathcal{H}$ is any subgroup of $\mathcal{G}^N$ such that $\delta(\mathcal{H}) \leq N/2$, there is some $s \in S$ such that $s \notin B(\mathcal{H}, c \log n)$.*

In this section, we explore the possibility that the parameters in Corollary 2.5.19 are far from optimal. Is it true that any polynomial-sized symmetric multiset $S \subseteq \mathcal{G}^N$ with good enough expansion properties is $\omega(\log N)$-far from every subgroup of dimension at most $N/2$? We show that this is not true. Formally, we prove:

Theorem 2.5.20. *For any constant $c > 0$ and large enough N, there is a symmetric multiset $S \subseteq \mathbb{F}_2^N$ such that $\lambda(C(\mathbb{F}_2^N, S)) \leq \frac{1}{N^c}$ but there is a subspace L of dimension $N/2$ such that $S \subseteq B(L, 20c \log N)$.*

It is known (see [HLW06]) that for any fixed $\mathcal{G}$ and any multiset $S \subseteq \mathcal{G}^n$, $\lambda(C(\mathcal{G}, S)) = \Omega(1/\sqrt{|S|})$. Hence, the above theorem tells us that just the expansion properties of $C(\mathbb{F}_2^N, S)$ for any poly N-sized S are not sufficient to guarantee $\omega(\log N)$-distance from every subspace of dimension $N/2$.

Proof. Given any subspace L of $\mathbb{F}_2^N$ and any $p \in [0, 1]$, define the probability distribution $\tilde{\mu}(L, p)$ over $\mathbb{F}_2^N$ to be the distribution of the output of the following sampling algorithm: Pick $x \in L$ uniformly at random; pick $y \in \mathbb{F}_2^N$ by setting each y_i to 1 independently with probability p; output $x + y$. Let $\mu(L, p)$ be the distribution $\tilde{\mu}(L, p)$ conditioned on the event that the output element of the above sampling algorithm lies in $B(L, 2pN)$. We will show that for suitable L and $p_0 = \frac{10c \log N}{N}$, $\mu(L, p_0)$ is a $\frac{1}{N^c}$-biased distribution. By Lemma 2.5.13, this will clearly imply the theorem with S being the support of $\mu(L, p_0)$ (with each $x \in B(L, 2p_0N)$ being repeated sufficiently many times in S); it is obvious that S is symmetric since each $x \in \mathbb{F}_2^N$ is its own inverse. (Note that the space S produced above is possibly of exponential size. However, it is easy to show by sampling from S that there is a space S' of size $O(N^{2c+1})$ such that $\lambda(C(\mathbb{F}_2^N, S')) \leq \frac{2}{N^c}$ and $S' \subseteq B(L, 20c \log N)$. We omit the details.)

We choose L of dimension at most $N/2$ so that it has the following property: the subspace $L^\perp = \left\{x \in \mathbb{F}_2^N \;\middle|\; \forall y \in L \;\; \oplus_i x_i y_i = 0\right\}$ contains no non-zero x such that $wt(x) < N/10$ (we note that this is the same as stipulating that $L^\perp$ is a binary linear code in $\mathbb{F}_2^N$ with rate at least $N/2$ and distance at least $N/10$; however, this point is not essential to our proof). It is easy to check that a random subspace of dimension $N/2$ satisfies this property with good probability. Fix any such subspace L. Having fixed the subspace L, the distribution $\mu(L, p_0)$ has also been fixed. We now show that $\mu(L, p_0)$ is a $\frac{1}{N^c}$-biased distribution.

We will analyze the simpler distribution $\tilde{\mu}(L, p_0)$. An application of the Chernoff bound tells us that the statistical distance between $\tilde{\mu}(L, p_0)$ and $\mu(L, p_0)$ is at most $1/N^{2c}$ and hence it suffices to show that $\tilde{\mu}(L, p_0)$ is $\frac{1}{2N^c}$-biased, if N is large enough. Let us denote $\tilde{\mu}(L, p_0)$ by $\tilde{\mu}$.

It is well known that the characters of $\mathbb{F}_2^N$ are of the form χ_z for $z \in \mathbb{F}_2^N$, where $\chi_z(x) = (-1)^{\oplus_i x_i z_i}$; note that χ_z is a non-trivial character iff $z \neq 0$. Fix a non-trivial character χ_z. Let $\chi_{z,i}$ denote the function that maps a bit b to $(-1)^{bz_i}$; we have $\chi_z(x) = \prod_{i=1}^n \chi_{z,i}(x_i)$. We will analyze $|\mathbf{E}_{x' \sim \tilde{\mu}}[\chi_z(x')]|$ in one of two different ways depending on whether $z \in L^\perp$ or not.

Case 1, $z \notin L^\perp$: In this case, we have

$$\begin{aligned}\left|\mathop{\mathbf{E}}_{x' \sim \tilde{\mu}}[\chi_z(x')]\right| &= \left|\mathop{\mathbf{E}}_{x,y}[\chi_z(x+y)]\right| \\ &= \left|\mathop{\mathbf{E}}_{x \in L}[\chi_z(x)]\right| \cdot \left|\mathop{\mathbf{E}}_{y}[\chi_z(y)]\right| = 0 \cdot \left|\mathop{\mathbf{E}}_{y}[\chi_z(y)]\right| = 0\end{aligned}$$

Case 2, $z \in L^\perp$: We know, by the choice of L, that in this case, $wt(z) \geq N/10$. Also, by definition, each $y_i \in \mathbb{F}_2$ is picked such that $y_i = 1$ with probability p_0. Hence, for any $i \in [n]$, $|\mathbf{E}_{y_i}[\chi_{z,i}(y_i)]|$ is $1 - 2p_0$ if $z_i = 1$ and 1 otherwise. Therefore, we have

$$\begin{aligned}\left|\mathop{\mathbf{E}}_{x' \sim \tilde{\mu}}[\chi_z(x')]\right| &= \left|\mathop{\mathbf{E}}_{x,y}[\chi_z(x+y)]\right| = \left|\mathop{\mathbf{E}}_{x \in L}[\chi_z(x)]\right| \cdot \left|\mathop{\mathbf{E}}_{y}[\chi_z(y)]\right| \\ &= \left|\mathop{\mathbf{E}}_{x \in L}[\chi_z(x)]\right| \cdot \prod_i \left|\mathop{\mathbf{E}}_{y_i}[\chi_{z,i}(y_i)]\right| \\ &\leq (1-2p_0)^{N/10} \leq \frac{1}{N^{2c}} < \frac{1}{2N^c}\end{aligned}$$

Hence, the absolute value of the expectation of χ_z over the distribution $\tilde{\mu}$ is bounded by $\frac{1}{2N^c}$. Since χ_z was an arbitrary non-trivial character of $\mathbb{F}_2^N$, this shows that $\tilde{\mu}$ is $\frac{1}{2N^c}$-biased and completes the proof. □

2.6 Discussion

In this chapter, we introduced the Help functions problem and presented an approach to solving it, namely by solving the Remote Point Problem. Despite the fact that the parameters achieved by hitherto known algorithms for the RPP fail to be useful in this regard, we feel that this seems to be the most promising way to solve the general help functions problem.

The following obvious open questions present themselves:

- Is there a more promising approach to the Help functions problem?
- Can one solve non-trivial special cases, such as the $(m, s, 2)$-help functions problem for m, s being $n^{\omega(1)}$?
- Can one solve the $(N/2, c(N)\log N)$-RPP for any $c(N) = \omega(1)$? This would yield a solution to the $(k, N\frac{c(N)\log k}{k})$-RPP by the method of Alon et al. described above.

Another very appealing problem suggested by the Help functions problem is the following. Since the Parity function is known to be hard for small constant-depth circuits [Ajt83, FSS84,

Hås89], a natural place to look for a function that does not lie in $\text{SizeDepth}_H^n(n^{O(1)}, O(1))$ is the set of parities: that is, the set $\mathcal{S} = \{\chi_S \mid S \subseteq [n]\}$, where $\chi_S(x) = \bigoplus_{j \in S} x_j$. Is this approach feasible? More formally,

- Does there exist any small set of Help functions H such that $\text{SizeDepth}_H^n(s(n), d)$ for some constant d and $s(n)$ slightly superpolynomial?

We believe that the answer to the above question is no. The reason for this is that a positive answer to the above would have surprising consequences in the communication complexity realm: more precisely, the Inner Product function would lie in the Polynomial Hierarchy in the communication complexity world (see [Lok01] for details).

Even in very special cases, the Help functions problem turns out to be very interesting. For example, consider the case when the set H of help functions is made up on parities of subsets of the input bits. That is, $H = \{\chi_{S_1}, \chi_{S_2}, \ldots, \chi_{S_m}\}$, where $S_i \subseteq [n]$ for each i. The following are subcases of the above question.

- Can one prove that there exists $S \subseteq [n]$ such that such that $\chi_S \notin \text{SizeDepth}(s, d)$ for s being slightly superpolynomial in n and d being constant?
- Can one prove that the inner product function cannot be computed by a small constant-depth circuit using the above help functions?

Chapter 3

The Help polynomials problem

3.1 Introduction

In the last chapter, we defined the *Help functions* problem, which is the problem of coming up with lower bounds for the class of functions computed by small constant-depth boolean circuits with few *arbitrary* functions at the leaf level. We explore a similar problem in this chapter, but in the realm of arithmetic circuits: the problem we consider is one of proving lower bounds on *noncommutative Algebraic Branching Programs* (ABPs) augmented with a few *Help polynomials*.

Our motivation in studying this problem is twofold. The first reason is that we wish to understand the Help functions problem better. By studying a very similar problem in the setting of noncommutative ABPs (where there is a precise characterization of complexity of polynomials – see [Nis91]), we hope to tease out more about the structure of the problem and hopefully unearth techniques that are useful in the boolean case.

A second motivation is proving lower bounds on noncommutative computation itself. Nisan [Nis91] has considered the problem of proving lower bounds in the noncommutative setting. In his seminal paper, he proved that any noncommutative ABP computing (the noncommutative versions of) the determinant or permanent polynomials must be of size $2^{\Omega(n)}$. However, this work remains the best lower bound we have for noncommutative computation in general. A nice way of making progress beyond the work of Nisan seems to be to see what can be done in the setting where a few hard polynomials are given to the ABP for "free". This is the scenario we examine.

Our results are the following.

1. Similarly in spirit to the boolean case, we show that the solving the Help polynomials problem is also linked to a problem similar to the *Remote Point Problem* studied by

Alon et al. [APY09]. More precisely, we show that the problem is connected to the *Remote Point Problem* in the *rank metric* which is defined as the rank distance between matrices. Here, even a very simple solution to the RPP in the rank metric gives us non-trivial lower bounds on ABPs using help polynomials.

2. We also study the Remote Point Problem in the Rank metric, and we build on ideas from Alon et al.'s work (for the Hamming metric version) in [APY09] to give a deterministic polynomial-time algorithm for certain parameters. However, these parameters are not sufficient to prove very much stronger lower bounds for ABPs augmented with help polynomials.

3.2 Noncommutative Algebraic Branching Programs

Let $X = \{x_1, x_2, \cdots, x_n\}$ be a set of n noncommuting variables, and $\mathbb{F}\langle X\rangle$ denote the noncommutative ring of polynomials over X with coefficients from the field $\mathbb{F}$. For $f \in \mathbb{F}\langle X\rangle$, let $d(f)$ denote the degree of f. Let $\mathrm{Mon}_d(X)$ be the set of degree d monomials over X. For a polynomial f and a monomial m over X, let $f(m)$ denote the coefficient of m in f. A nonempty subset $H \subseteq \mathbb{F}\langle X\rangle$ is *homogeneous* if there is a $d \in \mathbb{N}$ such that all the polynomials in H are homogeneous of degree d.

Let $G = (V, E)$ be a directed acyclic graph. For $u, v \in V$, let $\mathcal{P}_{u,v}$ be the set of paths from u to v, where a path in $\mathcal{P}_{u,v}$ is a tuple of the form $((u_0, u_1), (u_1, u_2), \ldots, (u_{l-1}, u_l))$ where $u_0 = u$ and $u_l = v$.

Definition 3.2.1. *Let $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_m\}$ be disjoint variable sets. Let $H = \{h_1, h_2, \ldots, h_m\} \subseteq \mathbb{F}\langle X\rangle$. An* Algebraic Branching Program (ABP) with help polynomials *H is a layered directed acyclic graph A with a source s and a sink t. Every edge e of A is labeled by a linear form $L(e)$ in variables $X \cup Y$. If $L(e) = \sum_i \alpha_i x_i + \sum_j \beta_j y_j$, the* polynomial $L'(e)$ associated with edge e *is obtained by substituting h_j for y_j, $1 \le j \le m$, in $L(e)$. I.e. $L'(e) = \sum_i \alpha_i x_i + \sum_j \beta_j h_j$. The* size *of A is the number of vertices in A.*

Given a path $\gamma = (e_1, e_2, \ldots, e_l)$ in A, define the polynomial $f_\gamma = L'(e_1) \cdot L'(e_2) \cdot \ldots \cdot L'(e_t)$ (note that the order of multiplication is important). For vertices u and v of A, we define the polynomial $f_{u,v} = \sum_{\gamma \in \mathcal{P}_{u,v}} f_\gamma$. The ABP A computes the polynomial $f_{s,t}$.

Suppose $L(e) = \sum_i \alpha_i x_i + \sum_j \beta_j y_j$. We say that the edge e is *homogeneously labeled* if all the polynomials in the set $\{x_i \mid \alpha_i \neq 0\} \cup \{h_j \mid \beta_j \neq 0\}$ are homogeneous and of the same degree $d(e)$. If the above set is empty, we let $d(e) = 0$. Now, suppose all edges of an ABP A are homogeneously labeled; then, for a path $\gamma = (e_1, e_2, \ldots, e_t)$ in A let $d(\gamma) = \sum_{i=1}^{t} d(e_i)$. The ABP A with help polynomials H is *homogeneous* if:

- all the edges in A are homogeneously labeled,

- For all u, v in A and $\gamma_1, \gamma_2 \in \mathcal{P}_{u,v}$, $d(\gamma_1) = d(\gamma_2)$.

For a homogeneous ABP A with help polynomials and any pair of vertices u, v in A, the polynomial computed from u to v is homogeneous.

In the absence of help polynomials, this gives the standard Algebraic Branching Programs as defined in, e.g. Nisan [Nis91]. Nisan [Nis91] has shown explicit lower bounds, e.g. for the Permanent and Determinant, for this model of computation. Our aim is to prove lower bounds for ABPs with help polynomials.

Let $X = \{x_1, x_2, \ldots, x_n\}$. For functions $m, s, d : \mathbb{N} \to \mathbb{N}$, the $(m(n), s(n), d(n))$-Help polynomial problem is defined as follows:

- INPUT: A collection of m polynomials $H = \{h_1, h_2, \ldots, h_m\} \subseteq \mathbb{F}\langle X \rangle$, given by their coefficients.
- DESIRED OUTPUT: A polynomial $F \in \mathbb{F}\langle X \rangle$ of degree $d(n)$ such that F cannot be computed by a size $s(n)$ ABP using H.

We would like an algorithm for the above problem that runs in time polynomial in the input and output sizes. Note that the polynomial $F \in \mathbb{F}\langle X \rangle$ is explicit in a weaker sense than defined in the introduction. However, we feel that this is justified here, since the polynomials in H are allowed to be arbitrary. And also note that the polynomial $F \in \mathbb{F}\langle X \rangle$ can be computed in time $m(n)^{O(1)} n^{O(d(n))}$, which for $m(n) = n^{O(d(n))}$ and $d(n) = \text{poly}(n)$ is $\exp(n)$, and this is also a reasonable notion of "explicit" that is often used in the literature.

3.3 Homogenization

In this section, we show that any ABP with arbitrary help polynomials computing a homogeneous polynomial can be transformed into an equivalent *homogeneous* ABP with homogeneous help polynomials with only a small increase in size. Thus, it suffices to prove lower bounds against homogeneous ABPs with help polynomials. Fix the help polynomial set $H \subseteq \mathbb{F}\langle X \rangle$. Let $m = |H|$ and $d(H) = \max_{h \in H} d(h)$. Also, fix some $d \in \mathbb{N}$.

Given $f \in \mathbb{F}\langle X \rangle$ and $i \in \mathbb{N}$, let $f^{(i)}$ denote the ith homogeneous part of h. For $2 \leq i \leq d$, let $\tilde{H}_i = \left\{ h^{(i)} \in \mathbb{F}\langle X \rangle \;\middle|\; h \in H \right\}$; let $\tilde{H} = \bigcup_{2 \leq i} \tilde{H}_i$. Let $\tilde{m}_i$ denote $|\tilde{H}_i|$ for each i, and let $\tilde{m}$ denote $|\tilde{H}| = \sum_i \tilde{m}_i$. We show the following homogenization theorem.

Theorem 3.3.1. *Given any ABP A using the help polynomials H computing a homogeneous polynomial f of degree $d \geq 1$, there is a homogeneous ABP $\tilde{A}$ using the help polynomials $\tilde{H}$ that computes the same polynomial as A, where the size of $\tilde{A}$ is at most $S(d+1)$, where S denotes the size of A.*

Proof. The following construction is fairly standard. Let s and t be the designated source and sink, respectively, of the ABP. We will use the notation of Section 3.2.

We now define $\tilde{A}$. $\tilde{A}$ will use the variables $X \cup \tilde{Y}$, where $\tilde{Y} = \left\{ y_i^{(j)} \;\middle|\; 1 \leq i \leq m, 2 \leq j \leq d(h_i) \right\}$. The vertices of $\tilde{A}$ are tuples (u, i), where u is a vertex of A and $i \in \mathbb{N}$ is a number between 0 and d. The source of $\tilde{A}$ will be $(s, 0)$ and the sink (t, d). We will define the set of edges of $\tilde{A}$ in two stages. We will first construct an ABP on the set of vertices of $\tilde{A}$ which will include edges with weights from $\mathbb{F}$ (i.e, edges e such that $L(e)$ is a non-zero degree 0 polynomial), and we will then show how to remove these edges from the ABP. Consider any edge e in the ABP A; let the label $L(e)$ of e be $\sum_{i=1}^{n} \alpha_i x_i + \sum_{j=1}^{m} \beta_j y_j$ and $0 \leq k \leq d$, define the linear form $L(e)_k$ – which captures the kth homogeneous part of $L'(e)$, the polynomial computed by edge e – as follows:

- If $k = 0$, define $L(e)_k$ to be the field element $\sum_{j=1}^{m} \beta_j h_j^{(0)}$.
- If $k = 1$, define $L(e)_k$ to be $\sum_{i=1}^{n} \alpha_i x_i + \sum_{j=1}^{m} \beta_j h_j^{(1)}$.
- If $k > 1$, define $L(e)_k$ to be $\sum_{j=1}^{m} \beta_j y_j^{(k)}$

Fix any vertex (v, k) of $\tilde{A}$. Let $\{u_1, u_2, \ldots, u_l\}$ be the predecessors of v in A and let e_i denote the edge (u_i, v). Then, it is easy to see that

$$f_{s,v}^{(k)} = \sum_{i=1}^{l} \sum_{j=0}^{k} f_{s,u_i}^{(j)} L'(e_i)_{(k-j)}$$

Hence, we define edges $e_{i,j}$ in $\tilde{A}$ from vertices (u_i, j) to (v, k) with label $L(e_{i,j}) = L(e_i)_{k-j}$. (Note that the label $L(e_{i,k})$ is just a field element. We will change this presently.) This concludes the first stage. Note that, since we only add edges from (u, i) to (v, j) when (u, v) is an edge in A, the graph of $\tilde{A}$ is acyclic. Also note that an edge e is labeled by a field element if and only if it connects vertices of the form (u, k) and (v, k), for some u, v, and k. Finally, it is easily seen from the definition of $\tilde{A}$ that the polynomial computed from $(s, 0)$ to (u, i) is the polynomial $f_{s,u}^{(i)}$ for any s, u, and i.

In the second stage, we will get rid of those edges in $\tilde{A}$ such that $L(e) \in \mathbb{F}$. We do this in two passes. Fix some topological ordering of the vertices of $\tilde{A}$, and order the edges $(\tilde{u}, \tilde{v})$ of $\tilde{A}$ lexicographically. As long as there is an edge $e = (\tilde{u}, \tilde{v})$ of $\tilde{A}$ such that $\tilde{v}$ is *not* the designated sink (t, d) and $L(e) \in \mathbb{F}$, we let e be the least such edge and do the following: we remove the edge e, and for each edge $e' = (\tilde{v}, \tilde{w})$ of $\tilde{A}$ going out of v, we change the label of the edge $e'' = (\tilde{u}, \tilde{w})$ to $L(e'') + L(e) \cdot L(e')$ (if no such edge e'' exists, we add this edge to the ABP and give it the label $L(e) \cdot L(e')$). It should be clear that the homogeneity of the ABP is preserved. After at most $O((sd)^2)$ many such modifications, all edges in $\tilde{A}$ that are

labeled by field elements are of the form $(\tilde{u}, (t, d))$. Moreover, by the above construction, it is clear that $\tilde{u} = (u, d)$ for some vertex $u \neq t$ of A. Since $d \geq 1$, we know that $\tilde{u} \neq (s, 0)$, the designated source node. We also know that there are no edges into $\tilde{u}$ which are labeled by a field element. We now do the following: for each edge $e = (\tilde{u}, (t, d))$ labeled by a field element, we remove the vertex $\tilde{u}$ and for each edge $e' = (\tilde{v}, \tilde{u})$, we remove e' and change the label of $e'' = (\tilde{v}, (t, d))$ to $L(e'') + L(e') \cdot L(e)$ (if no such e'' exists, we add such an edge e'' and set its label to $L(e') \cdot L(e)$). This concludes the construction.

It is easy to prove inductively that after every modification of $\tilde{A}$, the polynomial computed from $(s, 0)$ to (t, d) remains $f_{s,t}^{(d)}$. Hence, the ABP $\tilde{A}$ computes exactly the polynomial f computed by A. Also, by construction, the edges of $\tilde{A}$ are all homogeneously labeled; finally, it can also be seen that given a path γ from vertex (u, i) to vertex (v, j) in $\tilde{A}$, $d(\gamma) = j - i$: hence, the ABP is indeed homogeneous, and we are done. □

3.4 Decomposition of Communication Matrices

We now generalize the key lemma of Nisan [Nis91] that connects the size of noncommutative ABPs for an $f \in \mathbb{F}\langle X \rangle$ to the ranks of certain communication matrices $M_k(f)$. The generalization is for noncommutative ABPs with help polynomials, and it gives a more complicated connection between the size of ABPs to the ranks of certain matrices. For usual noncommutative ABPs considered in [Nis91], Nisan's lemma directly yields the lower bounds. In our case, this generalization allows us to formulate the lower bound problem as a Remote Point Problem for the rank metric.

We will assume that the explicit polynomial for which we will be proving lower bounds is homogeneous. Thus, by Theorem 3.3.1 we can assume that each help polynomial in $H = \{h_1, h_2, \ldots, h_m\}$ is homogeneous and of degree at least 2.

We first fix some notation. Let $d \in \mathbb{N}$ be an even number. Let $d(H) = \max_{h \in H} d(h)$. Also, for $2 \leq i \leq d(H)$, let $H_i = \{h \in H \mid d(h) = i\}$.

Suppose $f \in \mathbb{F}\langle X \rangle$ is homogeneous of even degree $d \geq 2$, and $k \in \mathbb{N}$ such that $0 \leq k \leq d$. We define the $n^k \times n^{d-k}$ matrix $M_k(f)$ (as in [Nis91]): Each row is labeled by a distinct monomial in $\mathrm{Mon}_k(X)$ and each column by a distinct monomial in $\mathrm{Mon}_{d-k}(X)$. Given monomials $m_1 \in \mathrm{Mon}_k(X)$ and $m_2 \in \mathrm{Mon}_{d-k}(X)$, the (m_1, m_2)th entry of $M_k(f)$ is the coefficient of the monomial $m_1 m_2$ in f and is denoted by $M_k(f)(m_1, m_2)$.

Call M an (l, m)-*matrix* if M is an $n^l \times n^m$ matrix with entries from $\mathbb{F}$, where the rows of M are labeled by monomials in $\mathrm{Mon}_l(X)$ and columns by monomials in $\mathrm{Mon}_m(X)$. Suppose $0 \leq l \leq k$ and $0 \leq m \leq d - k$. Let M_1 be an (l, m)-matrix and M_2 a $(k - l, (d - k) - m)$-matrix. We define the $(k, d-k)$-matrix $M = M_1 \otimes_{l,m}^{k} M_2$ as follows: Suppose $m_1 \in \mathrm{Mon}_k(X)$ and $m_2 \in \mathrm{Mon}_{d-k}(X)$ are monomials such that $m_1 = m_{11} m_{12}$ with $m_{11} \in \mathrm{Mon}_{k-l}(X)$ and

$m_{12} \in \mathrm{Mon}_l(X)$ and $m_2 = m_{21}m_{22}$ with $m_{21} \in \mathrm{Mon}_m(X)$ and $m_{22} \in \mathrm{Mon}_{(d-k)-m}(X)$. Then the $(m_1, m_2)^{th}$ entry of M is defined as

$$M(m_1, m_2) = M_1(m_{12}, m_{21}) \cdot M_2(m_{11}, m_{22}).$$

Let A be a homogeneous ABP with help polynomials H computing a polynomial f of degree d. Let u, v and w be vertices in the ABP A, and $\gamma_1 \in \mathcal{P}_{u,v}$ and $\gamma_2 \in \mathcal{P}_{v,w}$ be paths. We denote by $\gamma_1 \circ \gamma_2 \in \mathcal{P}_{u,w}$ the concatenation of γ_1 and γ_2.

Since A is homogeneous, each of the polynomials $f_{u,v}$ for vertices u, v of A is homogeneous. For $1 \leq k \leq d/2$, define the *k-cut* of A, $C_k \subseteq V(A) \cup E(A)$, as follows: A vertex $v \in V(A)$ is in C_k iff $d(f_{s,v}) = k$, and an edge $e = (u, v) \in E(A)$ is in C_k iff $d(f_{s,u}) < k$ and $d(f_{s,v}) > k$. For each $x \in C_k$, let $\mathcal{P}_x$ denote the set of s-t paths passing through x. Clearly, the sets $\{\mathcal{P}_x \mid x \in C_k\}$ partition $\mathcal{P}_{s,t}$, the set of all paths from s to t. Thus, we have

$$\begin{aligned} f &= \sum_{x \in C_k} \sum_{\gamma \in \mathcal{P}_x} f_\gamma \\ &= \sum_{v \in C_k \cap V(A)} \sum_{\gamma \in \mathcal{P}_v} f_\gamma \; + \sum_{e \in C_k \cap E(A)} \sum_{\gamma \in \mathcal{P}_e} f_\gamma. \end{aligned} \tag{3.1}$$

We now analyze Equation 3.1. For $v \in C_k \cap V(A)$, $\mathcal{P}_v = \{\gamma_1 \circ \gamma_2 \mid \gamma_1 \in \mathcal{P}_{s,v}, \gamma_2 \in \mathcal{P}_{v,t}\}$. Hence, for any $v \in C_k \cap V(A)$:

$$\begin{aligned} \sum_{\gamma \in \mathcal{P}_v} f_\gamma = \sum_{\substack{\gamma_1 \in \mathcal{P}_{s,v} \\ \gamma_2 \in \mathcal{P}_{v,t}}} f_{\gamma_1 \circ \gamma_2} &= \sum_{\substack{\gamma_1 \in \mathcal{P}_{s,v} \\ \gamma_2 \in \mathcal{P}_{v,t}}} f_{\gamma_1} \cdot f_{\gamma_2} \\ &= f_{s,v} f_{v,t}. \end{aligned} \tag{3.2}$$

Similarly, for any edge $e = (u, v) \in C_k \cap E(A)$, $\mathcal{P}_e = \{\gamma_1 \circ (e) \circ \gamma_2 \mid \gamma_1 \in \mathcal{P}_{s,u}, \gamma_2 \in \mathcal{P}_{v,t}\}$, where (e) denotes the path containing just the edge e. Thus,

$$\begin{aligned} \sum_{\gamma \in \mathcal{P}_e} f_\gamma = \sum_{\substack{\gamma_1 \in \mathcal{P}_{s,u} \\ \gamma_2 \in \mathcal{P}_{v,t}}} f_{\gamma_1 \circ (e) \circ \gamma_2} &= \sum_{\substack{\gamma_1 \in \mathcal{P}_{s,v} \\ \gamma_2 \in \mathcal{P}_{v,t}}} f_{\gamma_1} \cdot L'(e) \cdot f_{\gamma_2} \\ &= f_{s,u} L'(e) f_{v,t}. \end{aligned} \tag{3.3}$$

From Equations 3.1, 3.2, and 3.3, we get

$$f = \sum_{v \in C_k \cap V(A)} f_{s,v} f_{v,t} + \sum_{e=(u,v) \in C_k \cap E(A)} f_{s,u} L'(e) f_{v,t}.$$

As A is homogeneous of degree d, each polynomial in the sums above is homogeneous of degree d. Hence

$$M_k(f) = \sum_{v \in C_k \cap V(A)} M_k(f_{s,v} f_{v,t}) \; + \sum_{e=(u,v) \in C_k \cap E(A)} M_k(f_{s,u} L'(e) f_{v,t}). \tag{3.4}$$

For any $v \in C_k \cap V(A)$, $f_{s,v}$ and $f_{v,t}$ are homogeneous degree k and $d-k$ polynomials respectively. We denote by M_v the matrix $M_k(f_{s,v} f_{v,t})$. Notice that for $m_1 \in \mathrm{Mon}_k(X)$ and $m_2 \in \mathrm{Mon}_{d-k}(X)$, the $(m_1, m_2)^{th}$ entry of the matrix $M_v = M_k(f_{s,v} f_{v,t})$ is $f_{s,v}(m_1) f_{v,t}(m_2)$. Thus, M_v is an outer product of two column vectors and is hence a matrix of rank at most 1. Therefore, the first summation in Equation 3.4 is a matrix of rank at most $|C_k \cap V(A)|$.

For $e = (u, v) \in C_k \cap E(A)$, we know that $d(f_{s,u}) < k$ and $d(f_{s,v}) > k$ and thus, $d(e) \geq 2$. Hence, $L'(e) = \sum_{h \in H_{d(e)}} \beta_{e,h} h$, for $\beta_{e,h} \in \mathbb{F}$. Therefore, expanding the second summation in Equation 3.4, we get

$$\sum_{\substack{e=(u,v) \in \\ C_k \cap E(A)}} M_k(f_{s,u} L'(e) f_{v,t}) = \sum_{\substack{e=(u,v) \in \\ C_k \cap E(A)}} \sum_{h \in H_{d(e)}} \beta_{e,h} M_k(f_{s,u} \cdot h \cdot f_{v,t}) \tag{3.5}$$

Consider a term of the form $M_k(f_{s,u} h f_{v,t})$. For the rest of the proof let $d(w)$ denote $d(f_{s,w})$, for any vertex w of A. Given monomials $m_{11} \in \mathrm{Mon}_{d(u)}(X)$, $m_{12} \in \mathrm{Mon}_{k-d(u)}(X)$, $m_{21} \in \mathrm{Mon}_{d(h)-(k-d(u))}(X)$, and $m_{22} \in \mathrm{Mon}_{d-d(v)}(X)$, the entry $M_k(f_{s,u} h f_{v,t})(m_{11}m_{12}, m_{21}m_{22}) = h(m_{12}m_{21}) f_{s,u}(m_{11}) f_{v,t}(m_{22})$, since all polynomials involved are homogeneous. Hence, the matrix $M_k(f_{s,u} h f_{v,t})$ is precisely $M_{k-d(u)}(h) \otimes^k_{k-d(u),d(h)-(k-d(u))} M_e$, where $M_e(m_{11}, m_{22}) = f_{s,u}(m_{11}) f_{v,t}(m_{22})$, for $m_{11} \in \mathrm{Mon}_{d(u)}(X), m_{22} \in \mathrm{Mon}_{d-d(v)}(X)$. Clearly, M_e is a matrix of rank at most 1, for any $e \in C_k \cap E(A)$ and $h \in H_{d(e)}$. Continuing with the above calculation, we get

$$\begin{aligned}
\sum_{\substack{e=(u,v) \in \\ C_k \cap E(A)}} M_k(f_{s,u} L'(e) f_{v,t}) &= \sum_{\substack{e=(u,v) \in \\ C_k \cap E(A)}} \sum_{h \in H_{d(e)}} \beta_{e,h} M_{l_e}(h) \otimes^k_{l_e, m_e} M_e \\
&= \sum_{h \in H} \sum_{i=d_1(h)}^{d_2(h)} M_i(h) \otimes^k_{i,d(h)-i} \cdot \sum_{\substack{e=(u,v) \in C_k: \\ d(e)=d(h) \\ d(u)=k-i}} \beta_{e,h} M_e,
\end{aligned}$$

where $d_1(h) = \max\{1, d(h) - (d-k)\}$, $d_2(h) = \min\{d(h) - 1, k\}$, $l_e = k - d(u)$, and $m_e = d(h) - (k - d(u))$.

Plugging the above observations into Equation 3.4, we have

$$M_k(f) = \underbrace{\left(\sum_{v \in C_k \cap V(A)} M_v\right)}_{M'} + \sum_{h \in H} \sum_{i=d_1(h)}^{d_2(h)} M_i(h) \otimes^k_{i,d(h)-i} \underbrace{\left(\sum_{\substack{e=(u,v) \in C_k: \\ d(e)=d(h) \\ d(u)=k-i}} \beta_{e,h} M_e\right)}_{M'_{i,h}}$$

Notice that M' above has rank at most $|V(A)|$, and $M'_{i,h}$ has rank at most $|E(A)| \leq |V(A)|^2$ for any $h \in H$ and $d_1(h) \leq i \leq d_2(h)$. Hence, we have proved the following result:

Theorem 3.4.1. *Let A be a homogeneous ABP of size S computing a (homogeneous) polynomial f of degree d using the help polynomials H. Then, for any $k \in \{0, 1, \ldots, d\}$, we can write $M_k(f)$ as:*

$$M_k(f) = M' + \sum_{h \in H} \sum_{i=d_1(h)}^{d_2(h)} M_i(h) \otimes^k_{i,d(h)-i} M'_{i,h},$$

where $d_1(h) = \max\{1, d(h) - (d-k)\}$ and $d_2(h) = \min\{d(h) - 1, k\}$ such that $\text{rank}M' \leq S$ and $\text{rank}M'_{i,h} \leq S^2$ for each $h \in H$, and $i \in \{\max\{1, d(h) - (d-k)\} \ldots, \min\{d(h) - 1, k\}\}$.

3.5 Remote Point Problem for the rank metric

We now introduce an algorithmic problem that will help us prove lower bounds on the sizes of ABPs computing explicit polynomials using a (given) set of help polynomials H. This problem is actually the *Remote Point Problem for matrices in the rank metric* or the *Remote Matrix Problem* (RMP). This problem is analogous to the Remote Point Problem (RPP), which we discussed in Section 2.2.

Given two matrices $P, Q \in \mathbb{F}^{a \times b}$, the *Rank distance* between P and Q is defined to be $\text{rank}P - Q$. It is known that this defines a metric, known as the *rank metric* on the set of all $a \times b$ matrices over $\mathbb{F}$.

The RMP problem. Given two functions $k, r : \mathbb{N} \to \mathbb{N}$, we define $(k(N), r(N))$-RMP as follows.

- INPUT: A subspace V of $\mathbb{F}_2^{N \times N}$ of dimension at most $k(N)$ given by its generators $P_1, P_2, \ldots, P_k$.

- DESIRED OUTPUT: A matrix $P \in \mathbb{F}_2^{N \times N}$ such that for any $P' \in V$, $\text{rank}(P - P') \geq r(N)$.

We say that the $(k(N), r(N))$-RMP problem has an efficient solution over $\mathbb{F}$ if there is a *deterministic* algorithm that runs in time polynomial in N and computes a matrix P that is at rank distance at least r from the subspace generated by the $P_1, P_2, \ldots, P_k$.

Remark 3.5.1. *How does a solution to RMP give us an explicit noncommutative polynomial f for which we can show lower bounds for the sizes of noncommutative ABPs with help polynomials? We now explain the connection.*

Let A be a homogeneous ABP of size S computing a polynomial f of degree d. Let $d_1(h)$ denote $\max\{1, d(h) - d/2\}$ and $d_2(h)$ denote $\min\{d/2, d(h) - 1\}$. For $a, b, p, q \in \mathbb{N}$ such that $p \in [n^a]$ and $q \in [n^b]$, let $E_{a,b}^{p,q}$ be the $n^a \times n^b$ elementary matrix with 1 as (p,q)th entry, and 0 elsewhere. The matrices $\left\{E_{a,b}^{p,q} \;\middle|\; p \in [n^a], q \in [n^b]\right\}$ span all matrices in $\mathbb{F}^{n^a \times n^b}$. By Theorem 3.4.1

$$M_{d/2}(f) = M' + \sum_{h \in H} \sum_{i=d_1(h)}^{d_2(h)} M_i(h) \otimes_{i,d(h)-i}^{d/2} M'_{i,h},$$

where $\text{rank} M' \leq S$. *For $h \in H$ and $i \in \{d_1(h), \ldots, d_2(h)\}$, the matrix $M'_{i,h}$ is an $n^{d/2-i} \times n^{d/2-d(h)+i}$ dimension matrix. We can write $M'_{i,h}$ as a linear combination of the elementary matrices in $\{E_{d/2-i,d/2-d(h)+i}^{p,q} \mid p \in [n^{d/2-i}], q \in [n^{d/2-d(h)+i}]\}$.*

Let $\mathcal{A}$ be the set of matrices of the form $M_i(h) \otimes_{i,d(h)-i}^{d/2} E_{d/2-i,d/2-d(h)+i}^{p,q}$, where $h \in H$, $i \in \{d_1(h), \ldots, d_2(h)\}$, and $p \in [n^{d/2-i}]$, $q \in [n^{d/2-d(h)+i}]$. Each matrix in $\mathcal{A}$ is an $n^{d/2} \times n^{d/2}$ matrix, with its rows and columns labeled by monomials in $\text{Mon}_{d/2}(X)$. Every matrix of the form $M_i(h) \otimes_{i,d(h)-i}^{d/2} M'_{i,h}$ is a linear combination of matrices in $\mathcal{A}$. Crucially, note that $\mathcal{A}$ depends only on the set of help polynomials and the parameter d, and it does not depend *on the ABP A.*

By substitution for $M'_{i,h}$ we obtain the following expression for $M_{d/2}(f)$ in terms of linear combination of matrices in $\mathcal{A}$.

$$M_{d/2}(f) = M' + \sum_{M \in \mathcal{A}} \alpha_M M,$$

where $\alpha_M \in \mathbb{F}$. Since, M' has rank at most S, it implies that $M_{d/2}(f)$ is at rank distance at most S from the subspace generated by the matrices in $\mathcal{A}$. Thus, if we can compute a matrix $\hat{M}$ in deterministic time polynomial in n^d that has rank distance $S = 2^{O(n)}$ from the subspace generated by $\mathcal{A}$ we would obtain an explicit homogeneous degree d polynomial f with lower bound $2^{\Omega(n)}$ by setting $\hat{M} = M_{d/2}(f)$. This is the approach that we will take for proving lower bounds.

We present the following simple algorithm, which suffices for our lower bound application.

Theorem 3.5.2. *For any $k(\cdot)$, the $(k(N), \lfloor N/(k(N)+1) \rfloor)$-RMP has an efficient solution over any field $\mathbb{F}$ such that field operations in $\mathbb{F}$ and Gaussian elimination over $\mathbb{F}$ can be performed in polynomial time.*

Proof. We assume that $k < N$; otherwise the problem is trivial. Let r denote $\lfloor N/k + 1 \rfloor$. Choose the first r column vectors in each of the matrices $P_1, P_2, \ldots, P_k$. Let $v_1, v_2, \ldots, v_{rk} \in \mathbb{F}^N$ be these vectors in some order. As $rk \leq N - r$, using Gaussian elimination, we can efficiently choose $v_{rk+1}, v_{rk+2}, \ldots, v_{r(k+1)} \in \mathbb{F}^N$ with the following property: for every $i \in [k+1]$, v_{rk+i} is linearly independent of $v_1, v_2, \ldots, v_{rk+(i-1)}$. Let P be any matrix that has $v_{rk+1}, v_{rk+2}, \ldots, v_{r(k+1)}$ as its first r columns. It is not too difficult to see that given any matrix P' in the subspace generated by $P_1, P_2, \ldots, P_k$, the first r columns of $P - P'$ remain independent, i.e $\text{rank}(P - P') \geq r$. □

We find this version of the Remote Point Problem also to be an intriguing algorithmic question. In [APY09] Alon et al. provide a nontrivial algorithm for RPP in the Hamming metric (over $\mathbb{F}_2$). We use similar methods to provide an improved solution to RMP for small fields. The result is proved in Section 3.7. Unfortunately, the improvement in parameters over the trivial solution above is not enough to translate into an appreciably better lower bound.

3.6 Lower bounds for ABPs with Help Polynomials

In this section, we prove some lower bounds for ABPs computing some explicit polynomials using a set of given help polynomials H. Here, 'explicit' means that the coefficients of the polynomial can be written down in time polynomial in the number of coefficients of the input (the help polynomials H) and the output (the hard to compute polynomial).

Throughout this section, $\mathbb{F}$ will be a field over which field operations and Gaussian elimination can be performed efficiently. Let the set of help polynomials be $H = \{h_1, h_2, \ldots, h_m\}$; let $d(H) = \max_{h \in H} d(h)$.

We will first consider the case of homogeneous ABPs using the help polynomials H; H is, in this case, assumed to be a set of homogeneous polynomials. We will then derive a lower bound for general ABPs and a general set of help polynomials using Theorem 3.3.1.

3.6.1 The homogeneous case

Let H be a set of homogeneous polynomials in this section. Our aim is to produce, for any degree $d \in \mathbb{N}$, an explicit homogeneous polynomial F_d of degree d that cannot be computed by homogeneous ABPs. To avoid some trivialities, we will assume that d is even.

We first observe that, to compute homogeneous polynomials of degree d, a homogeneous ABP cannot meaningfully use help polynomials of degree greater than d:

Lemma 3.6.1. *Let A be a homogeneous ABP using the help polynomials H to compute a polynomial f of degree d. Then, there is a homogeneous ABP A', of size at most the size of A, such that A' computes f and furthermore, for every edge $e \in E(A')$, $d(e) \leq d$.*

Proof. Simply take A and throw away all edges $e \in E(A)$ such that $d(e) > d$; call the resulting homogeneous ABP A'. Since A is homogeneous, no path from source to sink in A can contain an edge e that was removed above. Hence, the polynomial computed remains the same. □

Hence, to prove a lower bound for an explicit homogeneous polynomial of degree d, it suffices to prove a lower bound on the sizes of ABPs computing this polynomial using the help polynomials $H_{\leq d} = \{h \in H \mid d(h) \leq d\}$. As above, let $d(H_{\leq d}) = \max_{h \in H_{\leq d}} d(h)$.

We begin with a simple explicit lower bound. Call a homogeneous polynomial $F \in \mathbb{F}\langle X \rangle$ of degree d *d-full-rank* if $\mathrm{rank} M_{d/2}(F) = n^{d/2}$. Full-rank polynomials are easily constructed; here is a simple example of one: $F(X) = \sum_{m \in \mathrm{Mon}_{d/2}(X)} m \cdot m$. It follows easily from Nisan's result [Nis91] that, without any help polynomials, homogeneous ABPs computing any d-full-rank polynomial are of size at least $n^{d/2}$. We show below that a similar lower bound continues to hold as long as $d(H) \leq d(1-\epsilon)$, for any fixed $\epsilon > 0$.

Theorem 3.6.2. *Assume that $d(H_{\leq d}) \leq d(1-\epsilon)$, for a fixed constant $\epsilon > 0$ and let $F \in \mathbb{F}\langle X \rangle$ be a d-full-rank polynomial. Then, any homogeneous ABP A computing F has size at least $\left(n^{\frac{\epsilon d}{4}}/\sqrt{2md}\right)$.*

Proof. Consider a homogeneous ABP A computing F using the help polynomials H. By the above lemma, we may assume that A uses only the polynomials $H_{\leq d}$. Let S denote the size of A. For any $h \in H_{\leq d}$, let $d_1(h)$ denote $\max\{1, d(h) - d/2\}$ and $d_2(h)$ denote $\min\{d/2, d(h) - 1\}$. By Theorem 3.4.1, we know that

$$M_{d/2}(F) = M' + \sum_{h \in H_{\leq d}} \sum_{i=d_1(h)}^{d_2(h)} M_i(h) \otimes_{i,d(h)-i}^{d/2} M'_{i,h}$$

where $\mathrm{rank} M' \leq S$ and $\mathrm{rank} M'_{i,h} \leq S^2$, for each $h \in H_{\leq d}$ and $i \in \{d_1(h), \ldots, d_2(h)\}$. For any h and any i such that $0 \leq i \leq d(h)$, $\mathrm{rank} M_i(h) \leq \min\{n^i, n^{d(h)-i}\}$, which is at most $n^{d(h)/2} \leq n^{d(H_{\leq d})/2}$. By our assumption on $d(H_{\leq d})$, we see that $\mathrm{rank} M_i(h) \leq n^{(1-\epsilon)d/2}$. By the definition of $\otimes_{i,d(h)-i}^{d/2}$, this implies that $\mathrm{rank} M_i(h) \otimes_{i,d(h)-i}^{d/2} M'_{i,h} \leq \mathrm{rank} M_i(h) \cdot \mathrm{rank} M'_{i,h}$,

which is at most $n^{(1-\epsilon)d/2}S^2$. Thus, we see that

$$\begin{aligned}\text{rank}M_{d/2}(F) &\leq S + \sum_{h\in H_{\leq d}} \sum_{i=d_1(h)}^{d_2(h)} n^{(1-\epsilon)d/2}S^2 \\ &\leq S + |H_{\leq d}|dn^{(1-\epsilon)d/2}S^2 \\ &\leq 2mdS^2n^{(1-\epsilon)d/2}\end{aligned}$$

As F is d-full-rank, this implies that

$$\begin{aligned}2mdS^2n^{(1-\epsilon)d/2} &\geq n^{d/2} \\ \therefore \quad S &\geq \frac{n^{\frac{\epsilon d}{4}}}{\sqrt{2md}}\end{aligned}$$

□

The above theorem tells us that as long as the help polynomials are not too many in number ($m = n^{o(d)}$ will do), and of degree at most $(1-\epsilon)d$, then any full rank polynomial remains hard to compute for ABPs with these help polynomials.

We now consider the case when $d(H_{\leq d})$ can be as large as d. In this case, we are unable to come up with an unconditional explicit lower bound. A strong solution to the RMP introduced in Section 3.5 would give us such a bound. However, with the suboptimal solution of Theorem 3.5.2, we are able to come up with explicit lower bounds in a special case. Let $\delta(H)$ denote $\min_{h\in H} d(h)$. By assuming some lower bounds on $\delta(H)$, we are able to compute an explicit hard function.

Theorem 3.6.3. *Assume $\delta(H) \geq (\frac{1}{2}+\epsilon)d$, for a fixed constant $\epsilon > 0$. Then, there exists an explicit homogeneous polynomial $F \in \mathbb{F}\langle X\rangle$ of degree d such that any homogeneous ABP A computing F using the help polynomials H has size at least $\lfloor n^{\frac{\epsilon d}{2}}/2md\rfloor$.*

Proof. Let A be a homogeneous ABP A of size S computing a polynomial f of degree d. Let $d_1(h)$ denote $\max\{1, d(h) - d/2\}$ and $d_2(h)$ denote $\min\{d/2, d(h) - 1\}$. As explained in Remark 3.5.1, let $E^{p,q}_{a,b}$ denote the $n^a \times n^b$-sized elementary matrix with 1 in the (p,q)th entry and 0s elsewhere. The matrices $\left\{E^{p,q}_{a,b} \;\middle|\; p \in [n^a], q \in [n^b]\right\}$ span all $n^a \times n^b$ matrices.

By Theorem 3.4.1

$$M_{d/2}(f) = M' + \sum_{h\in H_{\leq d}} \sum_{i=d_1(h)}^{d_2(h)} M_i(h) \otimes^{d/2}_{i,d(h)-i} M'_{i,h}$$

where $\text{rank}M' \leq S$. As explained in Remark 3.5.1, $M'_{i,h}$ is an $n^{d/2-i} \times n^{d/2-d(h)+i}$ dimension matrix and is in the span of $\{E^{p,q}_{d/2-i,d/2-d(h)+i}\}$, where $p \in [n^{d/2-i}], q \in [n^{d/2-d(h)+i}]$.

Let $\mathcal{A}$ denote the set of $n^{d/2} \times n^{d/2}$ matrices of the form $M_i(h) \otimes_{i,d(h)-i}^{d/2} E^{p,q}_{d/2-i,d/2-d(h)+i}$, where $h \in H_{\leq d}$, $i \in \{d_1(h), \ldots, d_2(h)\}$, and $p \in [n^{d/2-i}]$, $q \in [n^{d/2-d(h)+i}]$. Then we obtain

$$M_{d/2}(f) = M' + \sum_{M \in \mathcal{A}} \alpha_M M, \tag{3.6}$$

where $\alpha_M \in \mathbb{F}$. Since M' is a matrix of rank at most S, this implies that M is at rank distance at most S from the subspace generated by the matrices in $\mathcal{A}$.

Let $k = |\mathcal{A}|$. For each $h \in H$ and $i \in \{d_1(h), \ldots, d_2(h)\}$, we have added precisely $n^{d-d(h)}$ many matrices of the form $M_i(h) \otimes_{i,d(h)-i}^{d/2} E$, where E is an elementary matrix of dimension $n^{d/2-i} \times n^{d/2-d(h)+i}$. Since $d(h) \geq d(\frac{1}{2} + \epsilon)$ for each $h \in H_{\leq d} \subseteq H$, this implies that $k \leq mdn^{\frac{d}{2}(1-\epsilon)}$. Let N denote $n^{d/2}$; $\mathcal{A}$ consists of $k \leq mdN^{1-\epsilon}$ $N \times N$ matrices. By Theorem 3.5.2, we can, in time poly(N), come up with an $N \times N$ matrix M_0 that is at rank distance at least $\lfloor \frac{N}{k+1} \rfloor$ from the subspace generated by the matrices in $\mathcal{A}$. We label the rows and columns of M_0 by monomials from $\mathrm{Mon}_{d/2}(X)$, in the same way as the matrices in $\mathcal{A}$ are labeled. Using M_0, we define the homogeneous degree d polynomial $F \in \mathbb{F}\langle X \rangle$ to be the unique polynomial such that $M_{d/2}(F) = M_0$; that is, given any monomial $m \in \mathrm{Mon}_d(X)$ such that $m = m_1 \cdot m_2$ for $m_1, m_2 \in \mathrm{Mon}_{d/2}(X)$, $F(m)$ is defined to be $M_0(m_1, m_2)$.

Let A be a homogeneous ABP of size S computing F using the help polynomials H. Then, by Equation 3.6 we have

$$M_{d/2}(F) = M' + \sum_{M \in \mathcal{A}} \alpha_M M$$

where $\alpha_M \in \mathbb{F}$, and $\mathrm{rank} M' \leq S$. Since $M_{d/2}(F)$ is M_0, which is at rank distance at least $\lfloor N/(k+1) \rfloor$ from the subspace generate by $\mathcal{A}$, we see that $S \geq \mathrm{rank} M' \geq \lfloor N/(k+1) \rfloor$. This implies that,

$$S \geq \left\lfloor \frac{N}{mdN^{1-\epsilon}+1} \right\rfloor \geq \left\lfloor \frac{N^\epsilon}{2md} \right\rfloor = \left\lfloor \frac{n^{\frac{\epsilon d}{2}}}{2md} \right\rfloor$$

□

Remark 3.6.4. *The rather unnatural condition on $\delta(H)$ above can be removed with better solutions to the RMP problem. Specifically, one can show along the above lines that if the $(k, N/k^{\frac{1}{2}-\varepsilon})$-RMP has an efficient solution for $k = N^{2\delta}$, then for any H, there is an explicit polynomial that cannot be computed by any ABP A using H of size at most $n^{\Omega(\epsilon d)}/(md)^{O(1)}$. Here, ϵ and δ are arbitrary constants in $(0, 1)$.*

3.6.2 The inhomogeneous case

Let $\tilde{H}$ denote the set of all homogeneous parts of degree at least 2 obtained from polynomials in H, i.e $\tilde{H} = \left\{ h_j^{(i)} \;\middle|\; j \in [m], 2 \leq i \leq d(h_j) \right\}$. For $2 \leq i \leq d(H)$, let $\tilde{H}_i = \left\{ h \in \tilde{H} \;\middle|\; d(h) = i \right\}$. Note that $\tilde{H} = \bigcup_{2 \leq i \leq d(H)} \tilde{H}_i$.

As in the previous subsection, we construct explicit hard polynomials for even $d \in \mathbb{N}$. Let $\tilde{H}_{\leq d}$ denote $\bigcup_{2 \leq i \leq d} \tilde{H}_i$ if $d \leq d(H)$, and $\tilde{H}$ otherwise.

Corollary 3.6.5. *Assume $d(\tilde{H}_{\leq d}) \leq d(1-\epsilon)$, for a fixed constant $\epsilon > 0$. Then, there is an explicit homogeneous polynomial F of degree d such that any ABP that computes F using the help polynomials H has size at least $\frac{n^{\frac{\epsilon d}{4}}}{\sqrt{2md(d+1)}}$.*

Proof. Let F be a d-full-rank polynomial, as defined in Section 3.6.1. Consider any ABP A computing F using H. By Theorem 3.3.1, there exists a homogeneous ABP $\tilde{A}$ computing F using $\tilde{H}$, where the size of $\tilde{A}$ is at most $S(d+1)$. By Lemma 3.6.1, we may assume that $\tilde{A}$ uses only the help polynomials in $\tilde{H}_{\leq d}$. Since $|\tilde{H}_{\leq d}| \leq md$, Theorem 3.6.2 tells us that $S(d+1) \geq n^{\frac{\epsilon d}{4}}/\sqrt{2md^2}$, which implies the result. □

Corollary 3.6.6. *Let $\delta(\tilde{H}) = \min_{h \in \tilde{H}} d(h)$, and assume $\delta(\tilde{H}) \geq (\frac{1}{2} + \epsilon)d$ for a fixed constant $\epsilon > 0$. Then, there exists an explicit homogeneous polynomial $F \in \mathbb{F}\langle X \rangle$ of degree d such that any ABP A computing F using the help polynomials H has size at least $\frac{1}{d+1}\left\lfloor \frac{n^{\frac{\epsilon d}{2}}}{2md^2} \right\rfloor$.*

Proof. By Theorem 3.3.1, given any ABP A of size S computing a homogeneous polynomial of degree d, there is a homogeneous ABP $\tilde{A}$ of size at most $S(d+1)$ that computes the same polynomial as A using the help polynomials $\tilde{H}$. By Lemma 3.6.1, we may assume that $\tilde{A}$ only uses the help polynomials $\tilde{H}_{\leq d}$. Now, let F be the explicit polynomial from Theorem 3.6.3, with $\tilde{H}_{\leq d}$ taking on the role of H in the statement of the theorem; since $|\tilde{H}_{\leq d}| \leq md$, Theorem 3.6.3 tells us that $S(d+1) \geq \lfloor n^{\frac{\epsilon d}{2}}/2md^2 \rfloor$, which implies the result. □

3.7 A better solution to the RMP

Following the approach of Alon et al. [APY09], who provide a nontrivial algorithm for RPP in the Hamming metric (over $\mathbb{F}_2$), we improve on the parameters of Theorem 3.5.2 for the RMP over small prime fields. It is interesting to note that in our solution we get similar parameters as [APY09]. As mentioned earlier, the improvement in parameters over the simple solution of Theorem 3.5.2 is too little to give us a much better lower bound.

Throughout this section, $\mathbb{F}$ will denote a constant-sized field. The main result is stated below.

Theorem 3.7.1. *For any fixed constant $c > 0$, the $(\ell N, r)$-RMP has an efficient solution over any constant-sized field $\mathbb{F}$ and for any $\ell, r > 0$ such that $\ell \cdot r < c \log N$.*

In proving the above theorem, we will follow the algorithm of [APY09]. We need the following lemma, implicit in [APY09]:

Lemma 3.7.2. *Fix any field $\mathbb{F}$ such that Gaussian elimination over $\mathbb{F}$ can be performed in polynomial time. There is a $\mathrm{poly}(M, m, |\mathbb{F}|)$ time algorithm for the following problem: Given subspaces $V_1, V_2, \ldots, V_m$ of $\mathbb{F}^M$ such that $\sum_{i=1}^m |V_i| < |\mathbb{F}|^M$, find a point $u \in \mathbb{F}^M$ such that $u \notin \bigcup_i V_i$.*

Proof. The algorithm will fix the coordinates of u one by one. Assuming that the values $u_1, u_2, \ldots, u_i$ have been fixed for $0 \leq i \leq n$, let $U_i = \left\{ w \in \mathbb{F}^M \;\middle|\; w_j = u_j \text{ for } 1 \leq j \leq i \right\}$. The algorithm will fix the coordinates of u, ensuring that the following is true: For each i such that $1 \leq i \leq M$, $\sum_{j=1}^m |V_j \cap U_i| < |U_i| = |\mathbb{F}|^{M-i}$. Note that, since U_0 is just $\mathbb{F}^M$, the inequality is satisfied at $i = 0$ by the assumption on the size of the subspaces $V_1, V_2, \ldots, V_m$; also note that the inequality is satisfied at $i = M$ if and only if $u \notin \bigcup_i V_i$.

Assuming $u_1, u_2, \ldots, u_i$ have been fixed for $i < M$, we define, for every $\alpha \in \mathbb{F}$, the set $U_{i,\alpha} = \{w \in U_i \mid w_{i+1} = \alpha\}$. Clearly, the sets $\{U_{i,\alpha}\}_\alpha$ partition U_i. Hence, we see that $\sum_{j=1}^m |V_j \cap U_i| = \sum_{\alpha \in \mathbb{F}} \sum_{j=1}^m |V_j \cap U_{i,\alpha}|$ and thus, there is some $\alpha \in \mathbb{F}$ such that $\sum_{j=1}^m |V_j \cap U_{i,\alpha}| < \frac{|U_i|}{|\mathbb{F}|} = |\mathbb{F}^{M-i-1}|$.

Here is the algorithm:

- While $u_1, u_2, \ldots, u_i$ have been determined for $i < M$, do the following:
  - As mentioned above, the following invariant is maintained: $\sum_{j=1}^k |V_j \cap U_i| < |U_i| = |\mathbb{F}|^{M-i}$.
  - Find $\alpha \in \mathbb{F}$ such that $\sum_{j=1}^k |V_j \cap U_{i,\alpha}| < \frac{|U_i|}{|\mathbb{F}|} = |\mathbb{F}^{M-i-1}|$. By the reasoning in the paragraph above, such an α exists and surely, it can be found in $\mathrm{poly}(M, k, |\mathbb{F}|)$ time using Gaussian elimination.
  - Set u_{i+1} to α.

The correctness of the algorithm is clear from the reasoning above. □

We now briefly describe the improved algorithm for the RMP. Let $P_1, P_2, \ldots, P_k$ be the input matrices. We denote by L the subspace of $\mathbb{F}^{N \times N}$ spanned by these matrices. Also, let B_r denote the matrices of rank at most r. The idea of the algorithm is to "cover" the set $L + B_r$ by a union of subspaces $V_1, V_2, \ldots, V_m$ such that $\sum_i |V_i| < |\mathbb{F}|^{N^2}$. We then use the algorithm from Lemma 3.7.2 to find a matrix P that is not in $\bigcup_i V_i$; by the way we have picked the subspaces, it is clear that M will then be at rank distance at least r from the subspace L.

What follows is an important definition.

Definition 3.7.3. *Fix positive integers (d_1, d_2). Given $\mathcal{T}$, a collection of subspaces of $\mathbb{F}^N$, we say that $\mathcal{T}$ is (d_1, d_2)-good if:*

- $\dim(U) \leq N - d_1$ *for each* $U \in \mathcal{T}$.
- *Each* $A \subseteq \mathbb{F}^N$ *of size* d_2 *is contained in some* $U \in \mathcal{T}$.

The following claim illustrates the importance of (d_1, d_2)-good subspaces of $\mathbb{F}^N$.

Claim 3.7.4. *There is an algorithm that, when given as input* $\mathcal{T}$*, a* (d_1, d_2)*-good collection of subspaces of* $\mathbb{F}^N$*, produces a collection* $\mathcal{S}$ *of subspaces of* $\mathbb{F}^{N \times N}$ *of cardinality at most* $|\mathcal{T}|$*, with the following properties:*

- $\dim(V) \leq N^2 - d_1 N$ *for each* $V \in \mathcal{S}$.
- $B_{d_2} \subseteq \bigcup_{V \in \mathcal{S}} V$

Moreover, the algorithm runs in time $\mathrm{poly}(|\mathcal{T}|, N)$.

Proof. For each $U \in \mathcal{T}$, let $V(U)$ denote the subspace of $\mathbb{F}^{N \times N}$ generated by all vectors of the form uv^{T}, where $u \in U$ and $v \in \mathbb{F}^N$. The collection $\mathcal{S}$ is the collection of all such vector spaces $V(U)$, for $U \in \mathcal{T}$. Clearly, the cardinality of $\mathcal{S}$ is bounded by $|\mathcal{T}|$.

Note that a basis for $V(U)$ can be constructed by picking only uv^{T} where u and v range over bases for U and $\mathbb{F}^N$ respectively. This shows that $\dim(V(U)) \leq N^2 - d_1 N$ and that $V(U)$ can be constructed efficiently.

Finally, given any matrix Q of rank at most d_2, it can be written as a sum of matrices $Q_1 + Q_2 + \ldots + Q_{d_2}$, where each Q_i is a matrix of rank at most 1 and hence can be written as $u_i v_i^{\mathrm{T}}$, where $u_i, v_i \in \mathbb{F}^N$. Let $A = \{u_1, u_2, \ldots, u_{d_2}\}$. Since $\mathcal{T}$ is (d_1, d_2)-good, there is some $U \in \mathcal{T}$ such that $A \subseteq U$. This implies that $u_i v_i^{\mathrm{T}} \in V(U)$ for each $i \in [d_2]$. As $V(U)$ is a subspace, it must contain their sum Q. This concludes the proof. □

It is easily seen that a random collection of subspaces of $\mathbb{F}^N$ of appropriate dimension is (d_1, d_2)-good for the values of d_1 and d_2 that are of interest to us. We now assert the existence of an explicit collection of subspaces with this property.

Claim 3.7.5. *Fix any constant* $c \geq 1$*. For any* $\ell, r \in \mathbb{N}$ *such that* $\ell \cdot r < c \log N$*, there is an algorithm that runs in time* $N^{O(c)}$ *and produces an* (ℓ, r)*-good collection of subspaces of* $\mathbb{F}^N$.

We prove the above claim in the next section. Assuming the claim, we can prove Theorem 3.7.1.

Proof of Theorem 3.7.1. We will describe an algorithm for the problem. Without loss of generality, assume that $c \geq 1$. Let L be the input subspace of dimension at most ℓN. We would like to find a matrix P that is at rank distance at least r from L.

We first use the algorithm referred to in Claim 3.7.5 to construct an $(\ell+1, r)$-good collection of subspaces $\mathcal{T}$ of $\mathbb{F}^N$ in time $N^{O(c)}$. Clearly, $|\mathcal{T}| = N^{O(c)}$. Then, we use the algorithm of Claim 3.7.4 to construct a collection of subspaces $\mathcal{S}$ of $\mathbb{F}^{N\times N}$ of size $N^{O(c)}$ with the following properties:

- $\dim(V) \leq N^2 - (\ell+1)N$ for each $V \in \mathcal{S}$.
- $B_r \subseteq \bigcup_{V\in\mathcal{S}} V$

Consider the collection of subspaces $\mathcal{S}' = \{L + V \mid V \in \mathcal{S}\}$. Clearly, $L + B_r \subseteq \bigcup_{V\in\mathcal{S}'} V$. Moreover, the dimension of each subspace in $\mathcal{S}'$ is at most $\ell N + N^2 - (\ell+1)N \leq N^2 - N$. Hence, each subspace in $\mathcal{S}'$ is of cardinality at most $|\mathbb{F}|^{N^2-N}$. Since $|\mathcal{S}'| = N^{O(c)}$, for large enough N, we have $\sum_{V\in\mathcal{S}'} |V| < |\mathbb{F}|^{N^2}$. Hence, using the algorithm described in Lemma 3.7.2, we can, in time $N^{O(c)}$, find a matrix $P \notin \bigcup_{V\in\mathcal{S}'} V$. By construction, this matrix P is at rank distance greater than r from the subspace L. The entire algorithm runs in time $N^{O(c)}$. □

3.7.1 Proof of Claim 3.7.5

We give two different constructions: one for the case that $\ell \geq r$ and the other for the case that $\ell \leq r$.

The following notation will be useful. For each $i \in [N]$, let $e_i \in \mathbb{F}^N$ denote the vector that has a 1 in coordinate i and is 0 elsewhere. For any vector $x \in \mathbb{F}^N$ and $S \subseteq [N]$, we denote by $x|_S$ the vector in $\mathbb{F}^{|S|}$ that is the projection of x to the coordinates indexed by S.

Case 1: $\ell \geq r$

For each $A \subseteq \mathbb{F}^{2\ell}$ of cardinality r, let V_A be the subspace generated by $\left\{x \in \mathbb{F}^N \;\middle|\; x|_{[2\ell]} \in A\right\}$. It is easily seen that $\dim(V_A) \leq N - 2\ell + r \leq N - \ell$. Moreover, given any $A_1 \subseteq \mathbb{F}^N$ of size r, $A_1 \subseteq V_A$ where A is any subset of $\mathbb{F}^{2\ell}$ of size r containing $\left\{x|_{[2\ell]} \;\middle|\; x \in A_1\right\}$. Hence, the collection $\mathcal{T} = \left\{V_A \;\middle|\; A \subseteq \mathbb{F}^{2\ell}, |A| = r\right\}$ is an (ℓ, r)-good collection of subspaces.

The cardinality of $\mathcal{T}$ is $\binom{|\mathbb{F}|^{2\ell}}{r} \leq |\mathbb{F}|^{2\ell r} = N^{O(c)}$. Surely, $\mathcal{T}$ can be constructed in time $N^{O(c)}$.

Case 2: $\ell \leq r$

Given a set $A \subseteq \mathbb{F}^m$ for some $m \in \mathbb{N}$, we denote by rank$A$ the size of any maximal set of linearly independent vectors from A; we denote by corank(A) the value $(|A| - \text{rank}A)$.

Fix a set $A \subseteq \mathbb{F}^m$ for some $m \in \mathbb{N}$. Given $d, d' \in \mathbb{N}$, we say that A is *d-wise corank d'* if each $B \subseteq A$ such that $|B| = d$ satisfies $\text{corank}(B) \leq d'$; A is said to be *d-wise linearly independent* if it is d-wise corank 0. Sets that are d-wise linearly independent have been studied before: see [ABI86, Proposition 6.5], where matrices whose columns form a d-wise linearly independent set of vectors are used to construct d-wise independent sample spaces. The following claim follows from this result and from the lower bound on the size of any d-wise independent sample space proved in [ABI86, Proposition 6.4].

Claim 3.7.6 (implicit in [ABI86])**.** *Consider a set $A \subseteq \mathbb{F}^m$ of cardinality t. If A is d-wise linearly independent with $d \leq 2\sqrt{t}$, then $m \geq \frac{d \log_{|\mathbb{F}|} t}{5}$, for large enough d, t.*

Using the above claim, we prove the following lower bound on the size of sets that are d-wise corank d' for suitable d, d'.

Claim 3.7.7. *Consider a set $A \subseteq \mathbb{F}^r$ of cardinality t. There is an absolute constant c_0 such that the following holds. Let A be d-wise corank d' for positive integers d, d' with $c_0 d' \leq d \leq 2\sqrt{t}$. Then, $r \geq \frac{d \log_{|\mathbb{F}|} t}{12d'}$ if t, d, d' are large enough.*

Proof. Denote by d'' the value $\lfloor d/2d' \rfloor$. We construct a sequence of sets $A_0, A_1, \ldots$ as follows: A_0 is the set A; for any $i \geq 0$, if A_i has been constructed and is d''-wise linearly independent, we stop; otherwise, there is a $B \subseteq A_i$ of cardinality d'' that is not linearly independent – in this case, we set $A_{i+1} = A_i \setminus B$; we stop at $i = d'$. It is easy to see that the cardinality t_i of A_i is $t - id''$. It can also be checked that if A_i is d_i-wise corank d_i', then A_{i+1}, if constructed, is $(d_i - d'')$-wise corank $d_i' - 1$; it therefore follows that the set S_i, if constructed, is $(d - id'')$-wise corank $d' - i$, for any $i \geq 0$ – in particular, $S_{d'}$ is $d/2$-wise linearly independent.

We base our analysis on when the above process stops. Let i_0 be the largest i so that A_i is constructed. Its size t_{i_0} is at least $t - d'd'' \geq t - d/2 \geq t/2$ for large enough t. If $i_0 = d'$, then A_{i_0} is a set of size at least $t/2$ that is $d/2$-wise linearly independent – by Claim 3.7.6, we get $r \geq \frac{d \log_{|\mathbb{F}|} t}{12}$ for large enough d, t. Otherwise, $i_0 < d'$ and we must have A_{i_0} is d''-wise linearly independent – in this case, by Claim 3.7.6, we get $r \geq \frac{d'' \log_{|\mathbb{F}|} t}{5} \geq \frac{d \log_{|\mathbb{F}|} t}{12d'}$ if c_0 is large enough. Thus, in either case, our claim holds. □

Now, we apply the above lemma with $t = |\mathbb{F}|^{\lceil \frac{20}{c_0} \sqrt{c \log N} \rceil}$ and $d = c_0 \lceil \sqrt{c \log N} \rceil$. We obtain the following corollary:

Corollary 3.7.8. *Let t, d be as defined above. For large enough N, given any $A \subseteq \mathbb{F}^r$ of size t, there is a subset B of A of cardinality d such that* $\text{corank}(B) \geq \ell$.

Proof. Assume that A is d-wise corank d' for some d'. We will show that $d' \geq \ell$. For large enough N, by Claim 3.7.7, we have $d' \geq \min\{\frac{d}{c_0}, \frac{d \log_{|\mathbb{F}|} t}{12r}\}$. It remains to be shown that this quantity is at least ℓ.

Note that, since $\ell \leq r$, $\ell^2 \leq \ell r \leq c \log N$. Hence, $\ell \leq \sqrt{c \log N}$. Thus, by the choice of d, we see that $d/c_0 \geq \ell$. Moreover,

$$\frac{d \log_{|\mathbb{F}|} t}{12r} \geq \frac{20c \log N}{12r} > \ell$$

Hence, we see that $d' \geq \ell$. □

We now define the (ℓ, r)-good collection of subspaces. For each $S \subseteq [t]$ of cardinality d, and each $A \subseteq \mathbb{F}^d$ of size $d - \ell$, let $V_{S,A}$ be the subspace generated by $\left\{x \in \mathbb{F}^N \;\middle|\; x|_S = u \text{ for some } u \in A\right\}$. It can be seen that $\dim(V_{S,A}) \leq N - d + d - \ell = N - \ell$ for each S, A.

Given any $A_1 \subseteq \mathbb{F}^N$ of cardinality r, let $P \in \mathbb{F}^{r \times N}$ be the matrix the rows of which are the elements of A_1. Let A_2 denote the set of the first t columns of P. By Corollary 3.7.8, there is a $B \subseteq A_2$ of size d such that $\operatorname{corank}(B) \geq \ell$. Let $S \subseteq [t]$ index the columns of B in P. It can be seen that $A_1 \subseteq V_{S,A'}$ for any A' of size $d - \ell$ containing a set that spans $\{v|_S \mid v \in A_1\}$ (such an A' exists since $\operatorname{corank}(B) \geq \ell$).

Thus, we can take for our collection $\mathcal{T}$ of (ℓ, r)-good subspaces the collection of all $V_{S,A}$, where $S \subseteq [t]$ with $|S| = d$, and $A \subseteq \mathbb{F}^d$ of size $d - \ell$. The size of $\mathcal{T}$ is bounded by $\binom{t}{d}\binom{|\mathbb{F}|^d}{d-\ell} \leq t^d |\mathbb{F}|^{d^2} = N^{O(c)}$, by our choice of d and t. Clearly, $\mathcal{T}$ can be constructed in time $N^{O(c)}$.

3.8 Discussion

The main problem in the area of noncommutative computation is to prove lower bounds for stronger models of computation than Algebraic Branching Programs. It would be nice if the model of ABPs with help functions could be used to take a first step in this direction: is there a reasonably strong, natural model of noncommutative computation (stronger or incomparable in power to the ABPs) for which lower bounds can be reduced to proving lower bounds for ABPs with certain help functions?

A more general question is inspired by the connections of certain lower bound questions to the Remote Point Problem and its variant the Remote Matrix Problem. Are there other lower bound questions the solutions to which can be reduced to these algorithmic questions?

Finally, can the upper bounds we give on the Remote Matrix Problem be improved? Besides giving improved lower bounds for the model of ABPs with help functions, such bounds will perhaps shed light on the Remote Point Problem also.

Chapter 4

Lower bounds for Monotone constant-width circuits

4.1 Introduction

In this chapter, we consider the problem of proving lower bounds for *constant-width arithmetic circuits*. Such circuits model bounded memory algorithms for computing functions defined by multivariate polynomials. Our motivation for studying such circuits is twofold.

The first motivation comes from the world of noncommutative computation (though our main focus in this chapter will be on the commutative model). Using a rank argument, Nisan, in a seminal paper [Nis91], showed exponential size lower bounds for noncommutative formulas (and noncommutative algebraic branching programs) that compute the noncommutative permanent or determinant polynomials in the ring $\mathbb{F}\langle X\rangle$, where $X = \{x_1, \cdots, x_n\}$ are noncommuting variables.

In what directions can we extend Nisan's result? A result of Ben-Or and Cleve [BOC92] shows that bounded-width arithmetic circuits (both commutative and noncommutative) are at least as powerful as formulas (indeed width four is sufficient). Can we extend Nisan's lower bound arguments to prove size lower bounds for *noncommutative* bounded-width circuits? A study of bounded-width computation should help us in this aspect.

A second motivation comes from the proof of the result of Ben-Or and Cleve [BOC92] itself and its impact on *monotone* formulas, i.e. formulas that compute polynomials with real coefficients *without* using any negative constants. The proof proceeds by constructing, from any given formula computing a polynomial p, a bounded-width circuit computing the same polynomial of roughly the same size. However, this procedure has the property that it destroys monotonicity, since it uses subtraction in a crucial way. Hence, the following question makes sense: does the result of Ben-Or and Cleve hold in the monotone world?

That is, is it the case that every polynomial that can be computed by a monotone formula of small size can also be computed by a monotone bounded-width circuit of small size?

Our results are the following.

- We show that even proving lower bounds for width-2 noncommutative arithmetic circuits is an interesting and non-trivial problem. More precisely, we show — using an example of Nisan's [Nis91] — that such circuits can compute polynomials that cannot be computed by subexponential-sized noncommutative formulas (and indeed Algebraic Branching Programs, a slightly stronger model).

- An important open question in noncommutative computation is whether noncommutative Algebraic Branching Programs are superpolynomially more powerful than formulas. We show that this is true in the monotone setting. We also show, using a technique of Ben-Or, that width-2 noncommutative circuits can compute some polynomials that witness this separation.

- Finally, we give a strong negative answer to the question of whether monotone formulas can be simulated by monotone constant-width circuits of roughly the same size. More precisely, we show that for any constant $d \in \mathbb{N}$, there is a family of polynomials $\{p_n \in \mathbb{R}[x_1, x_2, \ldots, x_n] \mid n \in \mathbb{N}\}$ such with non-negative coefficients such that p_n can be computed by a monotone arithmetic formula of depth $2d$ and size $O(n)$, but any width-d monotone arithmetic circuit computing p_n must have size $\Omega(2^{n^{1/d}})$. Note that these results hold in the *commutative* model where such separations are harder to prove: in particular, they imply the corresponding separations in the noncommutative model.

- This last result also shows that the hierarchies of polynomial families computed by constant-depth monotone circuits and constant-width monotone circuits are infinite. A superpolynomial separation in the case of constant-depth circuits already follows from a result of Raz and Yehudayoff [RY09] (indeed, this result works in the stronger *multilinear* model), but our lower bounds are stronger in some parameters.

We first recall some basic definitions.

Definition 4.1.1. *[Nis91, RS05] An* Algebraic Branching Program *(ABP) over a field $\mathbb{F}$ and variables $x_1, x_2, \cdots, x_n$ is a* layered *directed acyclic graph with one* source *vertex of indegree zero and one* sink *vertex of outdegree zero. Let the layers be numbered $0, 1, \cdots, d$. Edges only go from layer i to $i+1$ for each i. The source and sink are the unique layer 0 and layer d vertices, respectively. Each edge in the ABP is labeled with a linear form over $\mathbb{F}$ in the input variables. The size of the ABP is the number of vertices. Each source to sink path in the ABP computes the product of the linear forms labeling the edges on the path, and the sum of these polynomials over all source to sink paths is the polynomial computed by the ABP.*

The scalars in an ABP can come from any field $\mathbb{F}$. If the input variables $X = \{x_1, x_2, \cdots, x_n\}$ are noncommuting then the ABP (or circuit) computes a polynomial in the free noncommutative ring $\mathbb{F}\langle X\rangle$. If the variables are commuting then the polynomial computed is in the ring $\mathbb{F}[X]$.

Definition 4.1.2. *A commutative arithmetic circuit over $\mathbb{F}$ and variables $x_1, x_2, \cdots, x_n$ is a directed acyclic graph with each node of indegree zero labeled by a variable or a scalar constant. Each internal node g of the DAG is labeled by $+$ or $\times$ (i.e. it is a plus or multiply gate) and is of indegree two. A node of the DAG is designated as the output gate. Each internal gate of the arithmetic circuit computes a polynomial (by adding or multiplying its input polynomials). The polynomial computed at the output gate is the polynomial computed by the circuit.*

The circuit is said to be a formula *if the underlying undirected graph is a tree. The circuit is said to be* layered *if its vertices are partitioned into vertex sets $V_1 \cup V_2 \cup \ldots \cup V_t$ such that V_1 consists only of leaves, and given any internal node g in V_i for $i > 1$, the children of g are either nodes from V_1 (consisting of constants or variables) or nodes from the set V_{i-1}. The size of a circuit is the number of nodes in it, and the width of a layered circuit is $\max_{i>1} |V_i|$. An arithmetic circuit over the field $\mathbb{R}$ is* monotone *if all the scalars used are nonnegative. Finally, a layered arithmetic circuit is* staggered *if, in each layer i with $i > 1$, every node except possibly one is a product gate of the form $g = u \times 1$, for some gate u from the previous layer.*

Noncommutative arithmetic circuits are defined in exactly the same way, except that for each internal node that is a multiply gate, one of its two children is labelled the left child *and the other the* right child. *This defines the order of multiplication at the gate.*

The notion of bounded (i.e, constant) width staggered circuits of width w is identical to the notion of a *straight-line program* with w registers that has been studied in the literature [BOC92]. It is easy to see that a polynomial that can be computed by a width-w arithmetic circuit of size s can be computed by a width $2w$ staggered arithmetic circuit of size ws. But if one wants to optimize on the width of the resulting staggered circuit, then one can do better than this. The following lemma shows that staggered circuits of width w can efficiently simulate width $w - 1$ (not necessarily staggered) arithmetic circuits. It holds in all settings of interest: commutative and noncommutative; and monotone as well as non-monotone.

Lemma 4.1.3. *Given any layered arithmetic circuit C of width w and size s computing a polynomial p, there is a staggered arithmetic circuit C' of width at most $w+1$ and size $O(ws)$ computing the same polynomial.*

Proof. The circuit C' is constructed by showing how to compute, for $i \geq 1$, the polynomials computed in layer $i + 1$ of C from the polynomials computed in the ith layer in C in a staggered fashion, using at most w layers of width at most $w + 1$. Equivalently, it amounts to designing a straight-line program with $w+1$ registers such that: initially, w of the registers

contain the polynomials computed in the w nodes of the i^{th} layer. In the end, w of the $w+1$ registers will contain the polynomials computed at the $i+1^{st}$ layer of C. Note that this is trivial for $i = 2$ since all nodes in layer 2 have only leaves as children. For some $i > 1$, let the U denote the nodes of C in layer i and V the nodes of C in layer $i+1$.

We define an undirected multigraph G corresponding to layers i and $i+1$ as follows: its vertex set $V(G)$ is U. For each gate $v \in V$ in circuit C that takes inputs $u_1, u_2 \in U$ we include the edge $\{u_1, u_2\}$ in $E(G)$. Notice that if $u_1 = u_2$ we add a self-loop to $E(G)$. Furthermore, if $v \in V$ takes one input as a $u \in U$ and the other inputs is a constant or a variable, then too we add a self-loop at vertex u. Finally, if both inputs to v are constants and/or variables, there is no edge in G corresponding to v. We note some properties of this graph G.

1. We have $|V(G)| \leq w$ and $|E(G)| + |V'| \leq w$, where V' is the set of those nodes in V that take only constants and/or variables as input.
2. Each vertex $u \in V(G)$ corresponds to a polynomial p_u computed at u in the i^{th} layer. Each edge $e \in E(G)$ is defined by some $v \in V$ and it corresponds to the polynomial q_e computed at v. In order to compute the polynomial corresponding to e we need the polynomials corresponding to its end points.

We have $w+1$ registers, w of which contain the polynomials $p_u, u \in U$. Our goal is to compute the polynomials $q_e, e \in E(G)$ using these registers. Using the graph structure of G, we will give an ordering of the edges $E(G)$. If we compute the polynomials q_e in that order then for every q_e computed we will have a free register to store q_e (when we do not need a polynomial p_u for further computation, we can free the register containing p_u).

Thus, what we want to do is specify such an ordering of the edges in $E(G)$[1].

We pick edges from $E(G)$ one by one. When $e \in E(G)$ is picked, we delete e from the graph and store q_e in a free register. Crucially, note that when a vertex $u \in V(G)$ becomes isolated in this process the polynomial p_u is not required for further computation and the register containing p_u is freed. Thus, at any point of time in this edge-deletion procedure, the number of registers required is equal to the sum of the number of edges removed from G and the number of *non-isolated* vertices left in G.

The edge picking procedure works as follows. We break G into its connected components $G_1 \cup G_2 \cup \ldots \cup G_s \cup G_{s+1} \cup \ldots \cup G_{s+t}$, where $G_1, G_2, \ldots, G_s$ are the *acyclic* components and $G_{s+1}, \ldots, G_{s+t}$ have cycles. We first compute the edges of G_1, and then those of G_2, and so on. At the end, we compute the polynomials corresponding to the nodes in V'.

Each connected component G_i is processed as follows: if there is an edge e in G_i that is not a cut edge, we pick the edge e and delete it from the graph; otherwise, since every edge of G_i

[1]We can blur the distinction between vertices and edges and the polynomials they represent.

is a cut edge, G_i must be a tree, and in this case, we remove any edge e that is incident to a degree-1 vertex. Proceeding thus, we maintain the invariant that at all points, all but one of the components of G_i are isolated vertices. Fix any intermediate point in the computation of $E(G_i)$. Let the number of registers used to store the non-isolated vertices of G_i and the edges removed from G_i be n_v and n_e respectively. Since G_i has at most one component with edges, it is immediate that $n_v \leq n'_e + 1$, where n'_e is the number of surviving edges in G_i. Hence, the number of registers used for the vertices and edges in G_i is at most $n'_e + n_e + 1 = |E(G_i)| + 1$. In particular, if G_i is acyclic this is at most $|V(G_i)|$.

Now, consider an arbitrary point in the computation of G. Assume that the edges of G_i are being computed. Note that the number of registers being used is bounded by the quantity

$$\sum_{j<i} |E(G_j)| + (|E(G_i)| + 1) + \sum j > i |V(G_j)|$$

Let us show that, for any i, the above is at most $w + 1$. Consider the case when $i \leq s$ and G_i is acyclic. Then, we have $|E(G_j)| < |V(G_j)|$ for each $j \leq i$ and hence, the above sum is bounded by $|V(G)|$, which is at most w. Similarly, when $i > s$ and G_i is cyclic, we have $|V(G_j)| \leq |E(G_j)|$ for each $j > i$ and hence the above sum is bounded by $|E(G)| + 1$, which is at most $w + 1$. Thus, the number of registers needed at any point is at most $w + 1$.

Moreover, since at each step the polynomial of some node $v \in V$ is computed, the total number of steps in the straight-line program is at most w. This proves the lemma. □

A seminal result in the area of bounded width circuits is due to Ben-Or and Cleve [BOC92] where they show that size s arithmetic formulas computing a polynomial in $\mathbb{F}[X]$ (or in $\mathbb{F}\langle X\rangle$ in the noncommutative case) can be evaluated by staggered arithmetic circuits of width three and size $O(s^2 n)$. Bounded width circuits have also been studied under various restrictions in [LMR10, MR09, JR09]. However, they have not considered the question of proving explicit lower bounds.

What is the power of arithmetic circuits of width 2? It is easy to see that the width-two circuit model is universal. We state this (folklore) observation.

Proposition 4.1.4. *Any polynomial of degree d with s monomials in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ (or in $\mathbb{F}\langle x_1, \cdots, x_n\rangle$) can be computed by a width two arithmetic circuit of size $O(d{\cdot}s)$. Furthermore, any monotone polynomial (i.e, with non-negative real coefficients) can be computed by a width two monotone circuit over $\mathbb{R}$ of size $O(d \cdot s)$.*

4.2 Some Observations

To motivate the study of constant-width circuits, we point out that, for the problem of proving lower bounds for noncommutative bounded width circuits, Nisan's rank argument

is not useful.

Fix a field $\mathbb{F}$ and a set of variables $X = \{x_1, x_2, \ldots, x_n\}$. Following Nisan [Nis91], we define for any homogeneous polynomial $f \in \mathbb{F}\langle X\rangle$ of degree d and any k such that $0 \leq k \leq d$, an $n^k \times n^{d-k}$ matrix $M_k(f)$ with entries from $\mathbb{F}$ as follows: the rows and columns of $M_k(f)$ are labelled by monomials of degree k and $d-k$ respectively and given monomials m_1 and m_2 of degree k and $d-k$, the (m_1, m_2)th entry of $M_k(f)$ is the coefficient of the monomial $m_1 \cdot m_2$ in f. Nisan showed the following (what he proves is actually slightly stronger).

Proposition 4.2.1. *For any homogeneous f of degree d and $k \in \{0, 1, \ldots, d\}$, the size of any ABP computing f must be at least* $\mathrm{rank}(M_k(f))$.

This is the lower bound technique he uses to show explicit lower bounds for ABPs and formulas. For example, consider the noncommutative "palindromes" polynomial $P(x_0, x_1) \in \mathbb{F}\langle x_0, x_1\rangle$, which is defined as follows:

$$P(x_0, x_1) = \sum_{w \in \{x_0, x_1\}^n} ww^R$$

where w^R denotes the reverse of w.

It is easy to see, and has already been observed by Nisan [Nis91], that $\mathrm{rank}(M_n(f)) = 2^n$ and hence any noncommutative ABP or formula computing P must have size at least $2^{\Omega(n)}$. However, we can give an easy width-2 noncommutative arithmetic circuit for $P(x_0, x_1)$ of size $O(n)$. Indeed, we can even ensure that each gate in this circuit is monotone and *homogeneous*, that is, each gate computes a homogeneous polynomial.

Proposition 4.2.2. *The palindromes polynomial $P(x_0, x_1)$ has a width-2 noncommutative arithmetic circuit of size $O(n)$.*

Proof. The circuit is simply a slight variant of the one described by Nisan for this polynomial, designed to make sure that the width is at most 2. For any $0 \leq i \leq n$, let $P_i(x_0, x_1)$ denote $\sum_{w \in \{x_0, x_1\}^i} ww^R$. Clearly, $P = P_n$. P_0 is simply the constant polynomial 1 and hence can be computed in width 2. We compute P_{i+1} from P_i using the rule $P_{i+1} = x_0 P_i x_0 + x_1 P_i x_1$. To do this in width 2, we store two copies of P_i in two registers and compute $x_0 P_i x_0$ in one of them and $x_1 P_0 x_1$ in the other (neither of these computations needs the other register); finally, we add the contents of the two registers to obtain P_{i+1}.

Continuing in this way, we get an $O(n)$-sized width-2 homogeneous circuit for the polynomial P. □

What then is a good candidate explicit polynomial that is not computable by width-2 circuits of polynomial size? We believe that the polynomial P_k^ℓ (of Section 4.3) for suitable k is the right candidate. A lower bound argument still eludes us. However, if we consider *monotone*

constant-width circuits then even in the commutative case we can show exponential size lower bounds for monotone width-k circuits computing P_k^ℓ. Since P_k^ℓ is computable by depth $2k$ arithmetic circuits (of unbounded fanin), it follows that the constant-width and the constant-depth hierarchies of monotone arithmetic circuits are infinite. We present these results in Section 4.3.

Remark 4.2.3. *Regarding the separation of the constant-depth hierarchy of monotone circuits, we note that a separation has also been proved by Raz and Yehudayoff in [RY09]; their lower bounds show a superpolynomial separation between the power of depth k* multilinear *circuits and depth $k+1$ monotone circuits for any k (see [RY09] for the definition and results regarding multilinear circuits). In contrast, our separation works only for monotone circuits, and only for infinitely many k. Nonetheless, we think that our separation is interesting because the separation we achieve is stronger. More precisely, the results of [RY09] show a separation of the order of $2^{(\log s)^{1+\Omega(1/k)}}$ (that is, there is a polynomial that can be computed by circuits of depth $k+1$ and size s but not by depth k circuits of size $2^{(\log s)^{1+\Omega(1/k)}}$). On the other hand, our separation is at least as large as $2^{(\log s)^c}$ for any $c > 0$ (see Section 4.3 for the precise separation).*

A related question is the comparative power of noncommutative ABPs and noncommutative formulas. Noncommutative formulas have polynomial size noncommutative ABPs. However, $s^{O(\log s)}$ is the best known formula size upper bound for noncommutative ABPs of size s. An interesting question is whether we can prove a separation result between noncommutative ABPs and formulas. We note that such a separation in the *monotone* case follows from an old result of Snir [Sni80].

Proposition 4.2.4. *Let $X = \{x_0, x_1\}$ be a set of two noncommuting variables. Let L denote the set of all monomials of degree $2n$ with an equal number of x_0 and x_1, and consider the polynomial $E \in \mathbb{Q}\langle x_0, x_1\rangle$, where $E = \sum_{w \in L} w$.*

1. *There is a monotone homogeneous ABP for E of size $O(n^2)$.*
2. *Any monotone formula computing E is of size $n^{\Omega(\lg n)}$.*

Proof. The first part is directly from a standard $O(n^2)$ size DFA that accepts precisely the set $L = \{w \in \{x_0, x_1\}^{2n} \mid w$ has an equal number of x_0's and x_1's$\}$. The second part follows from the fact that such a monotone formula would yield a commutative monotone formula for the symmetric polynomial of degree n over the variables $y_1, y_2, \cdots, y_{2n}$: this is obtained by first observing that the formula must compute homogeneous polynomials at each gate. Furthermore, we can label each gate (and each leaf) by a triple (i, j, d) where $j - i + 1 = d$ is the degree of the homogeneous polynomial computed at this gate such that each monomial generated at this gate will occupy the positions from i to j in the output monomials containing it. Hence we have x_0's at the leaf nodes labeled by triples $(i, i, 1)$ for all $2n$ values of i. We replace the x_0's labeled $(i, i, 1)$ by y_i and each x_1 by 1. The resulting

formula computes the symmetric polynomial as claimed. Snir in [Sni80] has shown a tight $n^{\Omega(\log n)}$ lower bound for monotone formulas computing the symmetric polynomial of degree n over the variables $y_1, y_2, \cdots, y_{2n}$. □

To illustrate again the power of constant width circuits, we note that there is, surprisingly, a width-2 circuit for computing the polynomial E.

Proposition 4.2.5. *There is a width-2 circuit of size $n^{O(1)}$ for computing E if the field $\mathbb{F}$ has at least cn^2 distinct elements for some constant c.*

Proof Sketch. This is based on the well-known trick of Ben-Or (see [SW01]) for computing the symmetric polynomials in depth 3. We consider the polynomial $g(x_0, x_1, z) = (x_0 z^{2^{k+1}+1} + x_1 z + 1)^{2^{k+1}}$, where $2^{k-1} < n \leq 2^k$. (z will eventually be a scalar from $\mathbb{F}$.) The coefficient of $z^{(2^{k+1}+1)n+n}$ in g is precisely the polynomial E. Following Ben-Or's argument, the problem of recovering the polynomial E can be reduced to solving a system of linear equations with an invertible coefficient matrix. Hence E can be expressed as a sum $E = \sum_{i=1}^{2n} \beta_i g(x_0, x_1, z_i)$, where the z_is are all distinct field elements. The terms $\beta_i g(x_0, x_1, z_i)$ can be evaluated with *one* register using repeated squaring of $x_0 z_i^{2^{k+1}+1} + x_1 z_i + 1$. The second register is used as an accumulator to compute the sum of these terms. ■

These observations are additional motivation for the study of constant-width arithmetic circuits. In Section 4.3 we prove lower bound results for monotone constant-width circuits.

4.3 Monotone constant width circuits

In this section we study *monotone* constant-width arithmetic circuits. We prove that there exist explicit polynomials computed by linear-sized constant-depth formulas that cannot be computed by subexponential-sized monotone constant-width circuits. This shows that there is no analogue of the result of Ben-Or and Cleve in the monotone world. Hence, any conversion from a formula to a constant-width circuit *must* destroy monotonicity.

The explicit polynomials that we construct yield the additional consequence that constant-width and constant-depth monotone arithmetic circuits form an infinite hierarchy.

All our polynomials in this section will be commutative, unless we explicitly state otherwise. We now define the explicit polynomials. For positive integers k and ℓ we define a polynomial P_k^ℓ on ℓ^{2k} variables as follows:

$$P_1^\ell(x_1, x_2, \ldots, x_{\ell^2}) = \sum_{i=1}^{\ell} \prod_{j=1}^{\ell} x_{(i-1)\ell+j}$$

$$P_{k+1}^\ell(x_1, x_2, \ldots, x_{\ell^{2k+2}}) = \sum_{i=1}^{\ell} \prod_{j=1}^{\ell} P_k^\ell(x_{(i-1)\ell^{2k+1}+(j-1)\ell^{2k}+1}, \ldots, x_{(i-1)\ell^{2k+1}+j\ell^{2k}})$$

An easy inductive argument from the definition gives the following.

Lemma 4.3.1. *The polynomial P_k^ℓ is homogeneous of degree ℓ^k on ℓ^{2k} variables and has $\ell^{\frac{\ell^k-1}{\ell-1}}$ distinct monomials.*

By definition, P_k^ℓ can be computed by a depth $2k$ monotone formula of size $O(\ell^k)$. Furthermore, the reason for the choice of P_k^ℓ is clear: we can argue that the polynomials P_k^ℓ are the "hardest" polynomials for constant-depth circuits. We make this more precise in the following observation.

Proposition 4.3.2. *Given a depth k arithmetic circuit C of size s, there is a projection reduction from C to the polynomial P_k^ℓ where $\ell = O(s^{2k})$.*

Proof Sketch. We sketch the easy argument. We can transform C into a formula. Furthermore, we can make it a layered formula with $2k$ alternating $+$ and $\times$ layers such that the output gate is a plus gate. This formula is of size at most s^{2k}. Clearly, a projection reduction (mapping variables to variables or constants) will transform P_k^ℓ to this formula, for $\ell = O(s^{2k})$. ∎

It is easy to see the following from the fact that a monotone depth $2k$ arithmetic circuit of size s can be simulated by a monotone width $2k$ circuit of size $O(s)$.

Proposition 4.3.3. *For any positive integers ℓ and k there is a monotone circuit of width $2k$ and size $O(\ell^{2k})$ that computes P_{2k}^ℓ.*

We now state the main lower bound result. For each $k > 0$ there is $\ell_0 \in \mathbb{Z}^+$ such that for all $\ell > \ell_0$ any width k monotone circuit for P_k^ℓ is of size $\Omega(2^\ell)$. We will prove this result by induction on k. For the induction argument it is convenient to make a stronger induction hypothesis.

For a polynomial $f \in \mathbb{F}[X]$, where $X = \{x_1, x_2, \cdots, x_n\}$ let $\text{mon}(f) = \{m \mid m \text{ is a nonzero monomial in } f\}$. I.e. $\text{mon}(f)$ denotes the set of nonzero monomials in the polynomial f. Also, let $\text{var}(f)$ denote the set of variables occurring in the monomials in $\text{mon}(f)$. Similarly, for an arithmetic circuit C we denote by $\text{mon}(C)$ and $\text{var}(C)$ respectively the set of nonzero monomials and variables occurring in the polynomial computed by C.

We call a layered circuit C *minimal* if there is no smaller circuit C' of the same width s.t $\text{mon}(C) = \text{mon}(C')$. It can be seen that for any monotone circuit C, there is a minimal circuit C' of the same width s.t $\text{mon}(C') = \text{mon}(C)$ and has the following properties.

- The only constants used in C' are 0 and 1. Furthermore, no gate is ever multiplied by a constant.

- By the minimality of C' every node g in C' has a path to the output node of C'. Hence, given any node g in C' computing a polynomial p, there is a monomial m such that $\text{mon}(m \cdot p) \subseteq \text{mon}(C')$. In particular, this implies that if C' computes a homogeneous multilinear polynomial, then p must be a homogeneous multilinear polynomial.
- If C' computes a homogeneous multilinear polynomial of degree d, and if a node g in layer i also computes a polynomial p of degree d, then in layer $i+1$, there is a sum gate g' such that g is one of its children. Thus, the gate g' computes a homogeneous multilinear polynomial p' of degree d such that $\text{mon}(p) \subseteq \text{mon}(p')$. In particular, $\text{mon}(p) \subseteq \text{mon}(C')$.

We call a minimal circuit satisfying the above a *good* minimal circuit. We now show a useful property of minimal circuits C, which applies to circuits satisfying $\text{mon}(C) \subseteq P_k^\ell$, for all $\ell, k \geq 1$.

Lemma 4.3.4. *Let $f = \sum_{i=1}^{\ell} P_i$ be a homogeneous monotone polynomial of degree $d \geq 1$ with $var(P_i) \cap var(P_j) = \emptyset$ for all $i \neq j$. Given any good minimal circuit such that $mon(C) \subseteq mon(f)$, we have the following: if a gate g in C computes a polynomial p of degree less than d, or a product of two such polynomials, then $var(p) \subseteq var(P_i)$ for a* unique *i.*

Proof. For any polynomial $q \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ we can define a bipartite graph $G(q)$ as follows: one partition of the vertex set is $\text{mon}(q)$ and the other partition $\text{var}(q)$. A pair $\{x, m\}$ is an undirected edge if the variable x occurs in monomial m. It is clear that the graph $G(f)$ is just the disjoint union of all the $G(P_i)$.

If the polynomial p computed by gate g is of degree $d' < d$, then, since C is good, there is a monomial m of degree $d' - d$ such that $\text{mon}(m \cdot p) \subseteq \text{mon}(C) \subseteq \text{mon}(f)$. This implies that $G(m \cdot p)$ is a subgraph of $G(f)$. On the other hand, $G(m \cdot p)$ is clearly seen to be a connected graph. This implies that, in fact, $G(m \cdot p)$ is a subgraph of $G(P_i)$ for some i and hence, $\text{var}(p) \subseteq \text{var}(P_i)$ for a *unique* i. This proves the lemma in this case.

Similarly, if p is a product of two polynomials of degree less than d, then $G(p)$ is a connected graph, and by the above reasoning, it must be the subgraph of some $G(P_i)$. Hence, the lemma follows. □

We now state and prove a stronger lower bound statement. It shows that P_k^ℓ is even hard to "approximate" by polynomial size width-k monotone circuits.

Theorem 4.3.5. *For each $k > 0$ there is $\ell_0 \in \mathbb{Z}^+$ such that for all $\ell > \ell_0$ and any width-k monotone circuit C such that*

$$mon(C) \subseteq mon(P_k^\ell) \text{ and } |mon(C)| \geq \frac{|mon(P_k^\ell)|}{2},$$

the circuit C is of size at least $\frac{2^\ell}{10}$.

Proof. Let us fix some notation: given $i \in \mathbb{Z}^+$ and $j \in [w]$, we denote by $g_{i,j}$ the jth node in layer i of C and by $f_{i,j}$ the polynomial computed by $g_{i,j}$. Also, given a set of monomials M, we say that a circuit C_1 *computes* M if $\mathrm{mon}(C_1) \supseteq M$.

Without loss of generality, we assume throughout that C is a good minimal circuit. The proof is by induction on k. The case $k = 1$ is distinct and easy to handle. Thus, we consider as the induction base case the case $k = 2$. Consider a width two monotone circuit C such that $\mathrm{mon}(C) \subseteq \mathrm{mon}(P_2^\ell)$ and $|\mathrm{mon}(C)| \geq |\mathrm{mon}(P_2^\ell)|/2 = \ell^{\ell+1}/2$. Let f denote the polynomial computed by C. By Lemma 4.3.1 both f and P_2^ℓ are homogeneous polynomials of degree $d = \ell^2$.

We write the polynomial P_2^ℓ as $\sum_{i=1}^{\ell} P_i$, where $\mathrm{var}(P_i) = \{x_{(i-1)\ell^3+1}, \ldots, x_{i\ell^3}\}$. Note that $\mathrm{var}(P_i) \cap \mathrm{var}(P_j) = \emptyset$ for $i \neq j$. Let $f = \sum_{i=1}^{\ell} P_i'$ where $\mathrm{mon}(P_i') \subseteq \mathrm{mon}(P_i)$ for each i.

Since C is good and f is homogeneous, each gate of C computes only homogeneous polynomials. Moreover, since $\mathrm{mon}(C) \subseteq \mathrm{mon}(P_2^\ell)$ and $\mathrm{var}(P_i) \cap \mathrm{var}(P_j) = \emptyset$ for $i \neq j$, Lemma 4.3.4 implies that given any node g in C that computes a polynomial p of degree less than d or a product of such polynomials satisfies $\mathrm{var}(p) \subseteq \mathrm{var}(P_i)$ for *one* i. Consider the lowest layer (i_0 say) when the circuit C computes a degree d monotone polynomial. W.l.o.g assume that $f_{i_0,1}$ is such a polynomial. We list some crucial properties satisfied by $g_{i_0,1}$ and C.

1. By the minimality of i_0, the node $g_{i_0,1}$ is a product gate computing the product of polynomials of degree less than d. Hence, $\mathrm{var}(f_{i_0,1}) \subseteq \mathrm{var}(P_i)$ for exactly one i. W.l.o.g , we assume $i = 1$. Since $\deg(f_{i_0}, 1) = d$ and C is good, we in fact have $\mathrm{mon}(f_{i_0,1}) \subseteq \mathrm{mon}(P_1)$.

2. Since $\deg(f_{i_0,1}) = d$ and C is good, we know that there is a node $g_{i_0+1,j_{i_0+1}}$ that is a sum gate with $g_{i_0,1}$ as child; $g_{i_0+1,j_{i_0+1}}$ computes a homogeneous polynomial of degree d and $\mathrm{mon}(f_{i_0+1,j_{i_0+1}}) \supseteq \mathrm{mon}(f_{i_0,1})$. Iterating this argument, we see that there must be a sequence of nodes g_{i,j_i}, for $i > i_0$ such that for each i, g_{i,j_i} is a sum gate with $g_{i-1,j_{i-1}}$ as child, such that $\mathrm{mon}(f_{i_0,1}) \subseteq \mathrm{mon}(f_{i_0+1,j_{i_0+1}}) \subseteq \mathrm{mon}(f_{i_0+2,j_{i_0+2}}) \ldots$, and each f_{i,j_i} is a homogeneous polynomial of degree d. We assume, w.l.o.g, that $j_i = 1$ for each $i > i_0$.

By the choice of i_0, note that the node $g_{i_0,2}$ either computes a polynomial of degree less than d or computes a product of polynomials of degree less than d. Hence, $\mathrm{var}(f_{i_0,2}) \subseteq \mathrm{var}(P_i)$ for some i. If $i > 1$, we assume w.l.o.g. that $\mathrm{var}(p) \subseteq \mathrm{var}(P_2)$. Let us consider the circuit C with the variables in $\mathrm{var}(P_1) \cup \mathrm{var}(P_2)$ set to 0. The polynomial computed by the new circuit C' is now $f' = f - P_1' - P_2' = \sum_{i=3}^{\ell} P_i'$. Let $q_{i,j}$ denote the new polynomial computed by the node $g_{i,j}$. Note that each $q_{i_0,j}$ is now a constant polynomial.

Consider the monotone circuit C'' obtained from C' as follows: we remove all the gates below layer i_0; the gate $g_{i_0,2}$ in layer i_0 is replaced by a product gate $c \times 1$, where c is the constant it computes in C'; from layer i_0 onwards, all nodes of the form $g_{i,1}$ are removed; in any edge connecting nodes $g_{i,1}$ and $g_{i+1,2}$, the node $g_{i,1}$ is replaced by the constant 0. Clearly, C'' is a

width 1 circuit. For ease of notation, we will refer to the nodes of C'' with the same names as the corresponding nodes in C'. For any node $g_{i,2}$ in C'' ($i \geq i_0$), let $q'_{i,2}$ be the polynomial it now computes. Crucially, we observe the following from the above construction.

Claim 4.3.6. *For each $i \geq i_0$, $mon(q'_{i,2}) \supseteq mon(q_{i,2}) \setminus mon(q_{i,1})$.*

We now finish the proof of the base case. Define a sequence $i_1 < i_2 < \ldots < i_t$ of layers as follows: for each $j \in [t]$, i_j is the least $i > i_{j-1}$ such that $\mathrm{mon}(q_{i,1}) \supsetneq \mathrm{mon}(q_{i_{j-1},1})$, and $\mathrm{mon}(q_{i_t,1}) = \mathrm{mon}(f')$. Clearly, t is at most the size of C. Note that it must be the case that $q_{i_j,1} = q_{i_j-1,1} + q_{i_j-1,2}$. Hence, we have $\mathrm{mon}(q_{i_j,1}) = \mathrm{mon}(q_{i_j-1,1}) \cup \mathrm{mon}(q_{i_j-1,2}) = \mathrm{mon}(q_{i_j-1,1}) \cup (\mathrm{mon}(q_{i_j-1,2}) \setminus \mathrm{mon}(q_{i_j-1,1}))$. By the above claim, the set $\mathrm{mon}(q_{i_j-1,2}) \setminus \mathrm{mon}(q_{i_j-1,1})$, which we will denote by S_j, can be computed by a width-1 circuit. Thus, $\mathrm{mon}(f') = \mathrm{mon}(q_{i_t,1}) = \mathrm{mon}(q_{i_0,1}) \cup \bigcup_{j=1}^{t} S_j$, where each S_j can be computed by a width-1 circuit. Since $q_{i_0,1}$ is the zero polynomial, we have $\mathrm{mon}(f') = \bigcup_{j=1}^{t} S_j$.

Now, consider any width-1 monotone circuit computing a set $S \subseteq P_2^\ell$. It is easy to see that the set S computed must have a very restricted form.

Claim 4.3.7. *The set S is of the form $mon(p)$ where $p = (\sum_{i \in X_1} x_i) \prod_{j \in X_2} x_j$, and $X_1 \cap X_2 = \emptyset$.*

Clearly, as each set S_j satisfies $S_j \subseteq \mathrm{var}(P'_i)$ for some i, it can have at most ℓ^3 monomials. Therefore, if the monotone circuit C is of overall size less than 2^ℓ then it can compute a polynomial of the form $P'_1 + P'_2 + f'$, where f' has at most $2^\ell \ell^3$ monomials. Since $|\mathrm{mon}(P'_i)| \leq |\mathrm{mon}(P_i)| = \ell^\ell$ for each i, we have for suitably large ℓ

$$|\mathrm{mon}(C)| \leq 2\ell^\ell + 2^\ell \ell^3 < 3\ell^\ell < \frac{\ell^{\ell+1}}{2} = \frac{|\mathrm{mon}(P_2^\ell)|}{2}$$

and the base case follows.

The induction step.

Consider any monotone circuit $\hat{C}$ of width $k-1$ such that $\mathrm{mon}(\hat{C}) \subseteq \mathrm{mon}(P_{k-1}^\ell)$ and $|\mathrm{mon}(\hat{C})| \geq |\mathrm{mon}(P_{k-1}^\ell)|/2$. As induction hypothesis we assume that $\hat{C}$ must be of size at least $2^\ell/10$.

Let $P_k^\ell = \sum_{i=1}^{\ell} P_i$, with $\mathrm{var}(P_i) = \{x_{(i-1)\ell^{2k+1}+1}, \ldots, x_{i\ell^{2k+1}}\}$ as in the base case. By definition, the ℓ variable sets $\mathrm{var}(P_i)$ are mutually disjoint and each P_i has degree $d = \ell^k$. It is convenient to also write $P_i = \prod_{j=1}^{\ell} Q_{ij}$, where each Q_{ij} is of type P_{k-1}^ℓ. We have $\mathrm{var}(Q_{ij}) = \{x_{(i-1)\ell^{2k+1}+(j-1)\ell^{2k}+1}, \ldots, x_{(i-1)\ell^{2k+1}+j\ell^{2k}}\}$.

We start by considering any width $k-1$ circuit $\hat{C}$ of size less than $2^\ell/10$ such that $\mathrm{mon}(\hat{C}) \subseteq \mathrm{mon}(P_k^\ell)$. For any $i \in [\ell]$, by fixing all the variables outside $\mathrm{var}(P_i)$ to 0, we obtain a width

$k-1$ circuit $\hat{C}_i$ of the same size s.t $\text{mon}(\hat{C}_i) \subseteq \text{mon}(P_i)$. Further, by setting all the variables outside $\text{var}(Q_{ij})$ to 1 for some $j \in [\ell]$, we obtain a circuit $\hat{C}_{ij}$ s.t $\text{mon}(\hat{C}_{ij}) \subseteq \text{mon}(Q_{ij})$. By the induction hypothesis, we see that $|\text{mon}(\hat{C}_{ij})| \leq |\text{mon}(Q_{ij})|/2$. Clearly $\text{mon}(\hat{C}_i) \subseteq \text{mon}(\hat{C}_{i1}) \times \text{mon}(\hat{C}_{i2}) \times \ldots \times \text{mon}(\hat{C}_{i\ell})$. Therefore, $|\text{mon}(\hat{C}_i)| \leq \prod_j |\text{mon}(\hat{C}_{ij})| \leq |\text{mon}(P_i)|/2^\ell$. Finally, as $\text{mon}(\hat{C}) = \bigcup_i \text{mon}(\hat{C}_i)$, $|\text{mon}(\hat{C})| \leq \sum_i |\text{mon}(\hat{C}_i)| \leq |\text{mon}(P_k^\ell)|/2^\ell$. We have established the following claim.

Claim 4.3.8. *For any width $k-1$ circuit $\hat{C}$ of size less than $2^\ell/10$ such that $mon(\hat{C}) \subseteq mon(P_k^\ell)$, we have $|mon(\hat{C})| \leq \frac{|mon(P_k^\ell)|}{2^\ell}$.*

For the induction step, consider any monotone width-k circuit C such that $\text{mon}(C) \subseteq \text{mon}(P_k^\ell)$ and of size at most $2^\ell/10$. We will show that $|\text{mon}(C)| < |\text{mon}(P_k^\ell)|/2$. W.l.o.g, we can assume that C is a good minimal circuit. Let f denote the polynomial computed by C; we write $f = \sum_{i=1}^{\ell} P_i'$, where $\text{mon}(P_i') \subseteq \text{mon}(P_i)$ for each i.

As in the base case, let i_0 be the first layer where a polynomial of degree d is computed. W.l.o.g. we can assume that $f_{i_0,1}$ is such a polynomial. By the minimality of i_0, the node $g_{i_0,1}$ must be a product node with children computing polynomials of degree less than d. This implies, as in the base case, that $\text{var}(f_{i_0,1}) \subseteq \text{var}(P_i)$ for a unique i. W.l.o.g. we assume that $i = 1$. As before, we can fix a sequence of nodes g_{i,j_i} for each $i > i_0$ such that g_{i,j_i} is a sum gate with $g_{i-1,j_{i-1}}$ as a child. It is easily seen that $\text{mon}(f_{i_0,1}) \subseteq \text{mon}(f_{i_0+1,j_{i_0+1}}) \subseteq \text{mon}(f_{i_0+2,j_{i_0+2}}) \ldots$, and each f_{i,j_i} computes a homogeneous polynomial of degree d. Renaming nodes if necessary, we assume $j_i = 1$ for all i.

Now consider $f_{i_0,j}$ for $j > 1$. By the minimality of i_0, we see that each $f_{i_0,j}$ is either a polynomial of degree less than d or a product of two such polynomials. Hence, $\text{var}(f_{i_0,j}) \subseteq \text{var}(P_s)$ for some $s \in [\ell]$. Thus, there is a set $S \subseteq [\ell]$ s.t $|S| = k' < k$ such that $\bigcup_{j>1} \text{var}(f_{i_0,j}) \subseteq \bigcup_{s \in S} \text{var}(P_s)$. Without loss of generality, we assume that those $s \in S$ that are greater than 1 are among $\{2, 3, \ldots, k\}$.

Consider the circuit C' obtained when each of the variables in $\bigcup_{s \in [k]} \text{var}(P_s)$ is set to 0. Let $q_{i,j}$ be the polynomial computed by $g_{i,j}$ in C'. The polynomial computed by C' is just $f' = f - \sum_{s \in [k]} P_s'$. Note that $q_{i_0,j}$ is now simply a constant for each j, and that the size of C' is at most the size of C which by assumption is bounded by $2^\ell/10$. Using this size bound we will argue that C' cannot compute too many monomials.

We now modify C' as follows: we remove all the gates below layer i_0; each gate $g_{i_0,j}$ with $j > 1$ is replaced by a product gate of the form $c \times 1$ where c is the constant $g_{i_0,1}$ computes in C'; from layer i_0 onwards, all nodes of the form $g_{i,1}$ are removed; in any edge connecting nodes $g_{i,1}$ and $g_{i+1,j}$ for $j > 1$, the node $g_{i,1}$ is replaced by the constant 0. Call this new circuit C''. Clearly, C'' has size at most the size of C and width at most $k-1$. For ease of notation, we will refer to the nodes of C'' with the same names as the corresponding nodes in C'. For any node $g_{i,j}$ in C'' ($i \geq i_0$ and $j > 1$), let $q'_{i,j}$ be the polynomial it now computes.

As in the base case, we observe the following from the above construction.

Claim 4.3.9. *For each $i \geq i_0$ and each $j > 1$, $mon(q'_{i,j}) \supseteq mon(q_{i,j}) \setminus mon(q_{i,1})$.*

Using this, we show that the circuit C' was essentially just using the gates $g_{i,1}$ to store the sum of polynomials computed using width $k-1$ circuits.

Construct a sequence of layers $i_1 < i_2 < \ldots < i_t$ in C' as follows: for each $j \in [t]$, i_j is the least $i > i_{j-1}$ such that $\text{mon}(q_{i,1}) \supsetneq \text{mon}(q_{i_{j-1},1})$, and $\text{mon}(q_{i_t,1}) = \text{mon}(f')$. Surely, t is at most the size of C'. Now, fix any i_j for $j \geq 1$. Clearly, it must be the case that $q_{i_j,1} = q_{i_j-1,1} + q_{i_j-1,s}$ for some $s > 1$; therefore, we have $\text{mon}(q_{i_j,1}) \subseteq \text{mon}(q_{i_j-1,1}) \cup (\text{mon}(q_{i_j-1,s}) \setminus \text{mon}(q_{i_j-1,1}))$. Denote the set $\text{mon}(q_{i_j-1,s}) \setminus \text{mon}(q_{i_j-1,1})$ by S_j. Since the above holds for all j, and $\text{mon}(q_{i_j-1,1}) = \text{mon}(q_{i_{j-1},1})$, we see that $\text{mon}(f') = \text{mon}(q_{i_t,1}) \subseteq \text{mon}(q_{i_0,1}) \cup \bigcup_j S_j = \bigcup_j S_j$, since $q_{i_0,1}$ is the zero polynomial.

We will now analyze $|S_j|$ for each j. By the above claim, there is a width $k-1$ circuit C'' of size at most the size of C such that $S_j \subseteq \text{mon}(C'') \subseteq P_k^\ell$. If the size of C (and hence that of C' and C'') is at most $2^\ell/10$, it follows from Claim 4.3.8 that $|S_j| \leq |\text{mon}(P_k^\ell)|/2^\ell$. Hence, we see that $|\text{mon}(f')| \leq t|\text{mon}(P_k^\ell)|/2^\ell$, which is at most $|\text{mon}(P_k^\ell)|/10$. But we know that the polynomial f computed by the circuit C is of the form $f' + \sum_{i\in[k]} P'_i$, where $|\text{mon}(P'_i)| \leq |\text{mon}(P_i)| = |\text{mon}(P_k^\ell)|/\ell$. Therefore,

$$|\text{mon}(f)| \leq \frac{k}{\ell}|\text{mon}(P_k^\ell)| + |\text{mon}(f')| \leq |\text{mon}(P_k^\ell)|\left(\frac{k}{\ell} + \frac{1}{10}\right) < \frac{|\text{mon}(P_k^\ell)|}{2}$$

for large enough ℓ. This proves the induction step. □

For $k \in \mathbb{Z}^+$ and $c > 0$ let $\text{Depth}_{k,c}$ and $\text{Width}_{k,c}$ denote the set of families $\{f_n\}_{n>0}$ of monotone polynomials $f_n \in \mathbb{R}[x_1, x_2, \ldots, x_n]$ computed by $c \cdot n^c$-sized monotone circuits of depth k and width k respectively. For $k \in \mathbb{Z}^+$, let $\text{Depth}_k = \bigcup_{c>0} \text{Depth}_{k,c}$ and $\text{Width}_k = \bigcup_{c>0} \text{Width}_{k,c}$. Thus, Depth_k and Width_k denote the set of families of monotone polynomials computed by $\text{poly}(n)$-sized monotone circuits of depth k and width k respectively. Note that, for each $k \in \mathbb{Z}^+$ we have $\text{Depth}_k \subseteq \text{Width}_k$. Moreover, from the definition of P_k^ℓ, we see that the family $\{P_k^{\lfloor n^{1/2k} \rfloor}\}_n \in \text{Depth}_{2k}$. Finally, in Theorem 4.3.5 we have shown that the family $\{P_k^{\lfloor n^{1/2k} \rfloor}\}_n \notin \text{Width}_k$, for constant k. Hence, we have the following corollary of Theorem 4.3.5.

Corollary 4.3.10. *For any fixed $k \in \mathbb{Z}^+$, $\text{Width}_k \subsetneq \text{Width}_{2k}$ and $\text{Depth}_k \subsetneq \text{Depth}_{2k}$.*

Theorem 4.3.5 can also be used to give a separation between the power of circuits of width (respectively, depth) k and $k+1$ for infinitely many k. We now state this separation. For any $k \in \mathbb{N}$ and any function $f : \mathbb{N} \to \mathbb{N}$, let us denote by f^k the *k-th iterate* of f, i.e the function $f \circ f \circ \ldots \circ f$ (k times). Given non-decreasing functions $f, g : \mathbb{N} \to \mathbb{N}$, call f a *sub*

$1/k$-*th iterate of* g if $f^k(n) < g(n)$, for large enough n (closely related notions have been defined in [Sze62] and [RR97]). It can be verified that sub $1/k$-th iterates of exponential functions can grow fairly quickly: for example, for any $\varepsilon > 0$ and any $k, c \in \mathbb{N}$, the function $2^{(\log n)^c}$ is a sub $1/k$-th iterate of 2^{n^ε}.

We now state the precise separation that can be inferred from the above theorem. For any $k, n \in \mathbb{N}$ with $k \geq 2$ and any polynomial $p \in \mathbb{R}[x_1, x_2, \ldots, x_n]$, let $w_k(p)$ (respectively $d_k(p)$) denote the size of the smallest monotone width k (respectively depth k) circuit that computes p.

Corollary 4.3.11. *There is an absolute constant $\alpha > 0$ such that the following holds. Fix any $k \in \mathbb{N}$ where $k \geq 2$. Also, fix any non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ that is a sub $1/k$-th iterate of $2^{\alpha n^{1/2k}}$. Then, for large enough n, there is a monotone polynomial $p \in \mathbb{R}[x_1, x_2, \ldots, x_n]$ such that for some $k', k'' \in \{k, k+1, \ldots, 2k-1\}$, $w_{k'}(p) \geq f(w_{k'+1}(p))$ and $d_{k''}(p) \geq f(d_{k''+1}(p))$.*

Proof. Let p denote the monotone polynomial $P_k^{\lfloor n^{1/2k} \rfloor} \in \mathbb{R}[x_1, x_2, \ldots, x_n]$. Theorem 4.3.5 tells us that $w_k(p) = \Omega(2^{\lfloor n^{1/2k} \rfloor})$. To obtain a lower bound on $d_k(p)$, note that any polynomial computed by a circuit of size s and depth k can be computed by a width k circuit of size $O(s^k)$; this tells us that $d_k(p) = 2^{\Omega(n^{1/2k})}$. Hence, there is some constant $\beta > 0$ such that $\min\{w_k(p), d_k(p)\} \geq 2^{\beta n^{1/2k}}$, for large enough n.

By definition, $p = P_k^{\lfloor n^{1/2k} \rfloor}$ has a depth $2k$ circuit of size $O(n)$, i.e $d_{2k}(p) = O(n)$. Proposition 4.3.3 tells us that $w_{2k}(p) = O(n)$ also. Hence, for some constant $\gamma > 0$ and large enough n, we have $\max\{w_{2k}(p), d_{2k}(p)\} \leq \gamma n$.

The above statements imply that $w_k(p) \geq g(w_{2k}(p))$ and $d_k(p) \geq g(d_{2k}(p))$, where $g(n) = 2^{\alpha n^{1/2k}}$ for some constant $\alpha > 0$ and n is large enough. Now, fix any non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ that is a sub $1/k$-th iterate of g. We see that $w_k(p) \geq g(w_{2k}(p)) > f^k(w_{2k}(p))$ for large enough n; clearly, this implies that for some $k' \in \{k, k+1, \ldots, 2k-1\}$, we must have $w_{k'}(p) \geq f(w_{k'+1}(p))$. Similarly, there is also a $k'' \in \{k, k+1, \ldots, 2k-1\}$ such that $d_{k''}(p) \geq f(d_{k''+1}(p))$. □

Similar corollaries hold for noncommutative circuits too. We define the polynomial P_k^ℓ in exactly the same way in the noncommutative setting. Note that any monotone bounded width noncommutative circuit computing P_k^ℓ automatically gives us a monotone commutative circuit of the same size and width computing the commutative version of P_k^ℓ. Hence, the lower bound of Theorem 4.3.5 also holds for noncommutative width-k circuits. For $k \in \mathbb{Z}^+$, let ncDepth_k and ncWidth_k denote the set of families of monotone polynomials $\{f_n \in \mathbb{R}\langle x_1, x_2, \ldots, x_n\rangle \mid n \in \mathbb{Z}^+\}$ computed by poly(n)-sized monotone (noncommutative) circuits of depth k and width k respectively. Analogous to the commutative case, we obtain the following.

Corollary 4.3.12. *For any fixed $k \in \mathbb{Z}^+$, $ncWidth_k \subsetneq ncWidth_{2k}$ and $ncDepth_k \subsetneq ncDepth_{2k}$.*

And finally, we observe that the separations between width and depth k and $k+1$ that hold in the commutative monotone case also hold in the noncommutative monotone case. Define, for any $k, n \in \mathbb{N}$ with $k \geq 2$ and any polynomial $p \in \mathbb{R}\langle x_1, x_2, \ldots, x_n \rangle$, let $ncw_k(p)$ (respectively $ncd_k(p)$) denote the size of the smallest monotone width k (respectively depth k) circuit that computes p. We have the following.

Corollary 4.3.13. *There is an absolute constant $\alpha > 0$ such that the following holds. Fix any $k \in \mathbb{N}$ where $k \geq 2$. Also, fix any non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ that is a sub $1/k$-th iterate of $2^{\alpha n^{1/2k}}$. Then, for large enough n, there is a monotone polynomial $p \in \mathbb{R}\langle x_1, x_2, \ldots, x_n \rangle$ such that for some $k', k'' \in \{k, k+1, \ldots, 2k-1\}$, $ncw_{k'}(p) \geq f(ncw_{k'+1}(p))$ and $ncd_{k''}(p) \geq f(ncd_{k''+1}(p))$.*

4.4 Discussion

In this chapter, we considered the power of constant-width computation. In the monotone case, we were able to prove explicit lower bounds (for very simple polynomials — polynomials that can be computed by small constant-depth formulas). The main open question is to extend this to the non-monotone case, even for noncommutative width-2 circuits.

- Can one prove an explicit superpolynomial lower bound for noncommutative width-2 circuits?

As state above, we believe that the polynomial P_k^ℓ remains hard for this class of circuits for $k \geq 2$. However, we do not know how to prove this. Proposition 4.2.2 shows that a simple application of Nisan's technique [Nis91] will not work to prove this.

Chapter 5

The Hardness of the Noncommutative Determinant

5.1 Introduction

We now turn to the study of the computational complexity of the noncommutative determinant. In his seminal paper [Nis91] Nisan first systematically studied the problem of proving lower bounds for noncommutative computation. The focus of his study was noncommutative arithmetic circuits, noncommutative arithmetic formulas and noncommutative algebraic branching programs. In his central result based on a rank argument, Nisan shows that the noncommutative permanent or determinant polynomials in the ring $\mathbb{F}\langle x_{11}, \cdots, x_{nn}\rangle$ require exponential size noncommutative algebraic branching programs.

Nisan's results are over the *free* noncommutative ring $\mathbb{F}\langle X\rangle$. Chien and Sinclair, in [CS07], explore the same question over other noncommutative algebras. They refine Nisan's rank argument to show exponential size lower bounds for formulas computing the permanent or determinant over specific noncommutative algebras, like the algebra of 2×2 matrices over $\mathbb{F}$, the quaternion algebra, and a host of other examples.

However, the question of whether there is a small noncommutative *circuit* for the determinant or permanent remains unanswered. (Indeed, no explicit lower bounds are known for the general noncommutative circuit model.) Since the existence of small noncommutative arithmetic circuits for the permanent would imply the existence of small *commutative* arithmetic circuits for the permanent, we have a good reason to believe that the permanent does not have small noncommutative arithmetic circuits. However, as far as we know, no such argument has been given for the case of the noncommutative *determinant*. Indeed, since Nisan [Nis91] has also shown an exponential separation between the power of noncommutative formulas and circuits, it may very well be that the noncommutative determinant has

polynomial-sized arithmetic circuits.

Another motivation for studying the computational difficulty of computing the noncommutative determinant (as a function) is an approach to designing randomized approximation algorithms for the $0-1$ permanent by designing good unbiased estimators based on the determinant. This approach has a long history starting with [GG81, KKL$^+$93]. Of specific interest are the works of Barvinok [Bar]; Chien, Rasmussen, and Sinclair [CRS03]; and more recently that of Moore and Russell [MR]. Barvinok [Bar] defines a variant of the noncommutative determinant called the *symmetrized determinant* and shows that given inputs from a *constant dimensional* matrix algebra, the symmetrized determinant over these inputs can be evaluated in polynomial time. He uses these to define a series of algorithms that he conjectures might yield progressively better randomized approximation algorithms for the (commutative) permanent. Chien, Rasmussen, and Sinclair [CRS03] show that efficient algorithms to compute the determinant over Clifford algebras of polynomial dimension would yield efficient approximation algorithms for the permanent. Moore and Russell [MR] provide evidence that Barvinok's approach might not work, but their results also imply that computing the symmetrized or standard noncommutative determinant over polynomial dimensional matrix algebras would give a good estimator for the permanent.

The results of this chapter are the following.

1. We provide evidence that the noncommutative determinant is hard. We show that if the noncommutative determinant[1] can be computed by a small noncommutative arithmetic circuit, then so can the noncommutative permanent and therefore, the commutative permanent has small commutative arithmetic circuits. This is in marked contrast to the commutative case, where the determinant is known to be computable by polynomial sized circuits, but the permanent is not known (or expected) to have subexponential sized arithmetic circuits.

2. We show that computing the noncommutative determinant over matrix algebras of polynomial dimension is as hard as computing the commutative permanent. We also derive as a consequence that computing the $n \times n$ permanent over nonnegative rationals is polynomial-time reducible to computing the noncommutative determinant over Clifford algebras of poly(n) dimension.

   This points to the intractability of carrying over Barvinok's approach for large dimension, and also to the possibility that the approach of Chien, Rasmussen, and Sinclair might be computationally infeasible.

   We stress that our result here is potentially more useful than a noncommutative circuit lower bound for the determinant, from an algorithmic point of view. For, an arithmetic circuit lower bound result would not rule out the possibility of a polynomial-time

[1] We haven't defined this polynomial formally yet and there are, in fact, many ways of doing it. See Section 5.2.

algorithm for the noncommutative determinant over even polynomial dimension matrix algebras. For example, Barvinok's algorithm [Bar] computes the symmetrized determinant over constant dimensional matrix algebras, whereas any algebraic branching program that computes the symmetrized determinant over constant dimensional matrix algebras must be of exponential size [CS07].

5.2 Preliminaries

For any set of variables X, let $\mathbb{F}\langle X\rangle$ denote the ring of *noncommuting* polynomials over X. Let $\mathrm{Mon}(X)$ denote the set of noncommutative monomials over X; given $d \in \mathbb{N}$, let $\mathrm{Mon}_d(X)$ denote the monomials over X of degree exactly d. For $f \in \mathbb{F}\langle X\rangle$ and $m \in \mathrm{Mon}(X)$, we will denote by $f(m)$ the coefficient of the monomial m in f.

For any ring R, we use $M_n(R)$ to denote the ring of $n \times n$ matrices with entries from R.

Fix $X = \{x_1, x_2, \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, two disjoint sets of variables. Given $f \in \mathbb{F}\langle X\rangle$, matrices $A_i \in M_k(\mathbb{F}\langle Y\rangle)$ for $1 \leq i \leq m$, and $i_0, j_0 \in [k]$, we use $f(A_1, A_2, \ldots, A_m)(i_0, j_0)$ to denote the (i_0, j_0)th entry of the matrix $f(A_1, A_2, \ldots, A_m) \in M_k(\mathbb{F}\langle Y\rangle)$. We will use the notion of *noncommutative circuits* and *noncommutative ABPs*, defined in Section 4.1.

5.2.1 Noncommutative determinants and permanents

Given $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$ for $n \in \mathbb{N}$, we define the $n \times n$ noncommutative determinant and permanent polynomials over the set of variables X. By fixing the order of multiplication in each monomial of the *commutative* determinant/permanent polynomials in different ways, one can obtain many different reasonable ways of defining the $n \times n$ noncommutative determinant and permanent, and indeed many of these definitions have been studied (see [Asl96], which surveys various flavours of the noncommutative determinant). The most straightforward definitions are those of the *Cayley determinant* and *Cayley permanent* – we will denote these by $\mathrm{Cdet}_n(X)$ and $\mathrm{Cperm}_n(X)$ respectively – which use the row order of multiplication. That is,

$$\mathrm{Cdet}_n(X) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma)\ x_{1,\sigma(1)} \cdot x_{2,\sigma(2)} \cdots x_{n,\sigma(n)},$$
$$\mathrm{Cperm}_n(X) = \sum_{\sigma \in S_n} x_{1,\sigma(1)} \cdot x_{2,\sigma(2)} \cdots x_{n,\sigma(n)}.$$

We also define the *Moore determinant* and *Moore permanent* – denoted $\mathrm{Mdet}_n(X)$ and $\mathrm{Mperm}_n(X)$ respectively – by ordering the variables in each monomial using the cyclic order

of the corresponding permutation. Given $\sigma \in S_n$, we write it as a product of disjoint cycles $(n^\sigma_{11} \cdots n^\sigma_{1l_1})(n^\sigma_{21} \cdots n^\sigma_{2l_2}) \cdots (n^\sigma_{r1} \cdots n^\sigma_{rl_r})$ such that $\forall i \in [r]$ and $j \in [l_r]\backslash\{1\}$, we have $n^\sigma_{i1} < n^\sigma_{ij}$ and $n^\sigma_{11} > n^\sigma_{21} > \cdots > n^\sigma_{r1}$. The Moore determinant and permanent are defined as

$$\mathrm{Mdet}_n(X) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma)\ x_{n^\sigma_{11},n^\sigma_{12}} \cdots x_{n^\sigma_{1l_r},n^\sigma_{11}} \cdots x_{n^\sigma_{r1},n^\sigma_{r2}} \cdots x_{n^\sigma_{rl_r},n^\sigma_{r1}},$$

$$\mathrm{Mperm}_n(X) = \sum_{\sigma \in S_n} x_{n^\sigma_{11},n^\sigma_{12}} \cdots x_{n^\sigma_{1l_r},n^\sigma_{11}} \cdots x_{n^\sigma_{r1},n^\sigma_{r2}} \cdots x_{n^\sigma_{rl_r},n^\sigma_{r1}}.$$

In the setting of a field $\mathbb{F}$ of characteristic 0, Alexander Barvinok, in [Bar], has studied another variant of the noncommutative determinant called the *symmetrized determinant*, which is denoted $\mathrm{sdet}_n(X)$. It is defined as follows:

$$\mathrm{sdet}_n(X) = \frac{1}{n!} \sum_{\sigma,\tau \in S_n} \mathrm{sgn}(\sigma)\mathrm{sgn}(\tau)\ x_{\tau(1),\sigma(1)} x_{\tau(2),\sigma(2)} \cdots x_{\tau(n),\sigma(n)}.$$

Barvinok shows that, for any fixed dimensional associative algebra $\mathcal{A}$ over $\mathbb{F}$ of characteristic zero, there is a polynomial-time algorithm which, on input an $n \times n$ matrix A with entries from $\mathcal{A}$, computes $\mathrm{sdet}_n(A)$. It is not known whether such algorithms exist for the Cayley or Moore determinants.

5.3 The Hadamard Product

A key notion we require for all our reductions is the Hadamard product of polynomials that was introduced in [AJS09a].

Definition 5.3.1. *Given polynomials $f, g \in \mathbb{F}\langle X\rangle$, their* Hadamard product $h = f \circ g$ *is defined as follows: h is the unique polynomial in $\mathbb{F}\langle X\rangle$ such that for any monomial $m \in \mathrm{Mon}(X)$, the coefficient $h(m) = f(m) \cdot g(m)$.*

In [AJS09a, Theorem 5] we show that given a noncommutative circuit for polynomial f and an ABP for polynomial g we can efficiently compute a noncommutative circuit for their Hadamard product $f \circ g$. However, the construction we present in [AJS09a] modifies the noncommutative circuit for the polynomial f. Hence, it will not work if we are allowed only black-box access to f, which we require for certain applications in this paper.

Suppose we have an efficient *black-box* algorithm for evaluating the polynomial $f \in \mathbb{F}\langle X\rangle$, where the variables in X take values in some matrix algebra (say, $n \times n$ matrices over a field $\mathbb{F}$). Furthermore, suppose we have an *explicit* ABP for the polynomial g. *Ideally,* we would like to obtain an efficient algorithm for computing their Hadamard product $f \circ g$ over the same matrix algebra.

However, what we can show is that we can put together the ABP and the black-box algorithm for f to obtain an efficient algorithm that computes $f \circ g$ over $\mathbb{F}$. This turns out to be sufficient to prove all our hardness results for the different noncommutative determinants.

Theorem 5.3.2. *Fix $d \in \mathbb{N}$. Let $Z = \{z_1, z_2, \ldots, z_m\}$ be a set of noncommuting variables and $g \in \mathbb{F}\langle Z \rangle$ be a homogeneous polynomial of degree d such that g is computed by an ABP P of size S. Then, there exist matrices $A_1, A_2, \ldots, A_n \in M_S(\mathbb{F})$ such that for any homogeneous polynomial $f \in \mathbb{F}\langle Z \rangle$ of degree d, $f \circ g = f(A_1 z_1, A_2 z_2, \ldots, A_n z_n)(1, S)$. Moreover, given the ABP P, the matrices $A_1, A_2, \ldots, A_n$ can be computed in time polynomial in the size of the description of P.*

Proof. Let the vertices of P be named $1, 2, \ldots, S$ where 1 is the source of the ABP and S is the sink. Define the matrices $A_1, A_2, \ldots, A_n \in M_S(\mathbb{F})$ as follows: $A_i(k, l)$ is the coefficient of the variable z_i in the linear form labelling the edge that goes from vertex k to vertex l; if there is no such edge, the entry $A_i(k, l) = 0$. For any monomial $m = z_{i_1} z_{i_2} \cdots z_{i_d} \in \mathrm{Mon}_d(Z)$, let A_m denote the matrix $A_{i_1} A_{i_2} \cdots A_{i_d}$. We see that

$$\begin{aligned} f(A_1 z_1, A_2 z_2, \ldots, A_n z_n) &= \sum_{i_1, i_2, \ldots, i_d \in [n]} f(z_{i_1} z_{i_2} \cdots z_{i_d})(A_{i_1} z_{i_1})(A_{i_2} z_{i_2}) \cdots (A_{i_d} z_{i_d}) \\ &= \sum_{i_1, i_2, \ldots, i_d \in [n]} f(z_{i_1} z_{i_2} \cdots z_{i_d})(A_{i_1} A_{i_2} \cdots A_{i_d})(z_{i_1} z_{i_2} \cdots z_{i_d}) \\ &= \sum_{m \in \mathrm{Mon}_d(Z)} f(m) A_m m \end{aligned}$$

Note that the coefficient $g(m)$ of a monomial $m = z_{i_1} z_{i_2} \cdots z_{i_d}$ in g is just $A_m(1, S) = \sum_{k_1, k_2, \ldots, k_{d-1} \in [S]} \prod_{j=1}^{d} A_{i_j}(k_{j-1}, k_j)$, where $k_0 = 1$ and $k_d = S$. Putting the above observations together, we see that $f(A_1 z_1, A_2 z_2, \ldots, A_n z_n)(1, S) = \sum_{m \in \mathrm{Mon}_d(Z)} f(m) A_m(1, S) m = \sum_{m \in \mathrm{Mon}_d(Z)} f(m) g(m) m = f \circ g$. Since the entries of the matrices $A_1, A_2, \ldots, A_n$ can be read off from the labels of P, it can be seen that $A_1, A_2, \ldots, A_n$ can be computed in polynomial time given the ABP P. This completes the proof. □

Remark 5.3.3. *We note that the matrices A_i in the statement of Theorem 5.3.2 can actually be computed from the ABP even more efficiently, say, in uniform* AC^0.

The following corollary is immediate.

Corollary 5.3.4. [AJS09a] *Given a noncommutative circuit of size S' for $f \in \mathbb{F}\langle Z \rangle$ and an ABP of size S for $g \in \mathbb{F}\langle Z \rangle$, we can efficiently compute a noncommutative circuit of size $O(S' S^3)$ for $f \circ g$.*

The next corollary is the more useful version for this paper.

Corollary 5.3.5. *Let $Z = \{z_1, z_2, \ldots, z_n\}$. Suppose $\mathcal{A}$ is a polynomial-time algorithm for computing a homogeneous degree d polynomial $f \in \mathbb{F}\langle Z\rangle$ for matrix inputs from $M_S(\mathbb{F})$.[2] Given as input an ABP P, with S nodes, computing a homogeneous degree d polynomial $g \in \mathbb{F}\langle Z\rangle$, and scalars $a_1, a_2, \ldots, a_n \in \mathbb{F}$, we can compute $f \circ g(a_1, a_2, \ldots, a_n)$ in polynomial time.*

Proof. We first compute matrices $A_1, A_2, \ldots, A_n$, described in the Theorem 5.3.2, in time polynomial in the description of the ABP P. Then we invoke the given algorithm $\mathcal{A}$ on input $(A_1a_1, A_2a_2, \ldots, A_na_n)$ to obtain as output an $S \times S$ matrix whose $(1, S)^{th}$ entry contains $f \circ g(a_1, a_2, \ldots, a_n)$. Clearly, the simulation runs in polynomial time. □

5.4 The hardness of the Cayley determinant

We consider polynomials over an arbitrary field $\mathbb{F}$ (for the algorithmic results $\mathbb{F}$ is either rational numbers or a finite field). The main result of this section is that if there is a polynomial-time algorithm to compute the $2n \times 2n$ Cayley determinant over inputs from $M_S(\mathbb{F})$ for $S = c \cdot n^2$ (for a suitable constant c) then there is a polynomial-time algorithm to compute the $n \times n$ permanent over $\mathbb{F}$.

Throughout this section let X denote $\{x_{ij} \mid 1 \leq i, j \leq 2n\}$, and Y denote $\{y_{ij} \mid 1 \leq i, j \leq n\}$. Our aim is to show that if there is a polynomial-time algorithm for computing $\mathrm{Cdet}_{2n}(X)$ where x_{ij} takes values in $M_S(\mathbb{F})$ then there is a polynomial-time algorithm that computes $\mathrm{Cperm}_n(Y)$ where y_{ij} takes values in $\mathbb{F}$.

The $2n \times 2n$ determinant has $2n!$ many signed monomials of degree $2n$ of the form $x_{1,\sigma(1)}x_{2,\sigma(2)} \cdots x_{2n,\sigma(2n)}$ for $\sigma \in S_{2n}$. We will identify $n!$ of these monomials, all of which have the same sign. More precisely, we will design a small ABP with which we will be able to pick out these $n!$ monomials of the same sign.

We now define these $n!$ many permutations from S_{2n} which have the same sign and the corresponding monomials of Cdet_{2n} that can be picked out by a small ABP.

Definition 5.4.1. *Let $n \in \mathbb{N}$. For each permutation $\pi \in S_n$, we define a permutation $\rho(\pi)$ in S_{2n}, called the* interleaving *of π, as follows:*

$$\rho(\pi)(i) = \begin{cases} \pi(\frac{i+1}{2}), & \text{if } i \text{ is odd,} \\ n + \pi(\frac{i}{2}), & \text{if } i \text{ is even.} \end{cases}$$

That is, the elements $\rho(\pi)(1), \rho(\pi)(2), \cdots, \rho(\pi)(2n)$ are simply $\pi(1), (n + \pi(1)), \pi(2), (n + \pi(2)), \cdots, \pi(n), (n + \pi(n))$.

[2] The statement can be generalized to any unital algebra $\mathcal{A}$ in place of the field $\mathbb{F}$.

The following lemma states a crucial property of the permutation $\rho(\pi)$.

Lemma 5.4.2. *The sign of the permutation $\rho(\pi)$ is independent of π. More precisely, for every $\pi \in S_n$, we have $\mathrm{sgn}(\rho(\pi)) = \mathrm{sgn}(\rho(1_n))$, where 1_n denotes the identity permutation in S_n.*

Proof. For each $\pi \in S_n$ we can define the permutation $\pi_2 \in S_{2n}$ as $\pi_2(i) = \pi(i)$ for $1 \leq i \leq n$ and $\pi_2(n+j) = n + \pi(j)$ for $1 \leq j \leq n$. It is easy to verify that $\mathrm{sgn}(\pi_2) = \mathrm{sgn}(\pi)^2 = 1$ for every $\pi \in S_n$. To see this we write π_2 as a product of disjoint cycles and notice that every cycle occurs an even number of times. Furthermore, we can check that $\rho(\pi) = \rho(1_n)\pi_2$, where we evaluate products of permutations from left to right. Hence it follows that $\mathrm{sgn}(\rho(\pi)) = \mathrm{sgn}(\rho(1_n))\mathrm{sgn}(\pi_2) = \mathrm{sgn}(\rho(1_n))$. □

We will denote by ρ_0 the permutation $\rho(1_n)$, where 1_n denotes the identity permutation in S_n.

For $\sigma \in S_{2n}$, we will denote by m_σ the monomial $x_{1,\sigma(1)}x_{2,\sigma(2)}\cdots x_{2n,\sigma(2n)} \in \mathrm{Mon}(X)$. For $\sigma, \tau \in S_{2n}$, we will denote the monomial $x_{\sigma(1),\tau(1)}x_{\sigma(2),\tau(2)}\cdots x_{\sigma(2n),\tau(2n)}$ by $m_{\sigma,\tau}$.

In the next lemma we show that there is an ABP that will filter out monomials that are not of the form $m_{\rho(\pi)}$ from among the m_σ.

Lemma 5.4.3. *There is an ABP P of size $O(n^2)$ and width n that computes a homogeneous polynomial $F \in \mathbb{F}\langle X\rangle$ of degree $2n$ such that for any $\sigma, \tau \in S_{2n}$,*

- *$F(m_\sigma) = 1$ if $\sigma = \rho(\pi)$ for some $\pi \in S_n$, and 0 otherwise.*
- *$F(m_{\sigma,\tau}) = 0$ unless $\sigma = 1_{2n}$, where 1_{2n} denotes the identity permutation in S_{2n}.*

Moreover, the above ABP P can be computed in time $\mathrm{poly}(n)$.

Proof. The ABP is essentially just a finite automaton over the alphabet X with the following properties: for input monomials of the form m_σ it accepts only those monomials that are of the form $m_{\rho(\pi)}$. Further, for input monomials of the form $m_{\sigma,\tau}$ it accepts only those monomials of the form $m_{1_{2n},\tau}$. We give the formal description of this ABP P below.

The ABP P contains $2n+1$ layers, labelled $\{0, 1, \ldots, 2n\}$. For each even $i \in \{0, 1, \ldots, 2n\}$, there is exactly one node q_i at level i; for each odd $i \in \{0, 1, \ldots, 2n\}$, there are n nodes $p_{i,1}, p_{i,2}, \ldots, p_{i,n}$ at level i. We now describe the edges of P: for each even $i \in \{0, 1, \ldots, 2n-2\}$ and $j \in [n]$, there is an edge from q_i to $p_{i+1,j}$ labelled $x_{i+1,j}$; for each odd $i \in \{0, 1, \ldots, 2n\}$ and $j \in [n]$, there is an edge from $p_{i,j}$ to q_{i+1} labelled $x_{i+1,n+j}$.

It is easy to see that P as defined above satisfies the requirements of the statement of the lemma. It is also clear that the ABP P can be computed in polynomial time. □

Note that the ABP P of Lemma 5.4.3 can in fact be constructed in uniform AC^0.

Remark 5.4.4. *For this section we require only the first part of Lemma 5.4.3. The second part of Lemma 5.4.3 is used in Section 5.6.*

We are now ready to prove that if there is a small noncommutative arithmetic circuit that computes the Cayley determinant polynomial, then there is a small noncommutative arithmetic circuit that computes the Cayley permanent polynomial.

Theorem 5.4.5. *For any $n \in \mathbb{N}$, if there is a circuit C of size s computing $\mathrm{Cdet}_{2n}(X)$, then there is a circuit C' of size polynomial in s and n that computes $\mathrm{Cperm}_n(Y)$.*

Proof. Assuming the existence of the circuit C as stated above, by Corollary 5.3.4, there is a noncommutative arithmetic circuit C'' of size $\mathrm{poly}(s, n)$ that computes the polynomial $F'' = \mathrm{Cdet}_{2n} \circ F$, where F is the polynomial referred to in Lemma 5.4.3. For any monomial m, if $m \neq m_\sigma$ for any $\sigma \in S_{2n}$, then $\mathrm{Cdet}_{2n}(m) = 0$ and hence, in this case, $F''(m) = 0$; moreover, for $m = m_\sigma$, we have $F(m) = 1$ if $\sigma = \rho(\pi)$ for some $\pi \in S_n$, and 0 otherwise. Hence, we see that

$$F''(X) = \sum_{\pi \in S_n} \mathrm{sgn}(\rho(\pi)) m_{\rho(\pi)} = \mathrm{sgn}(\rho_0) \left(\sum_{\pi \in S_n} m_{\rho(\pi)} \right)$$

where the last equality follows from Lemma 5.4.2.

Let C' be the circuit obtained from C'' by substituting x_{ij} with $y_{\frac{1+i}{2},j}$ if i is odd and $j \in [n]$, and by 1 if i is even or $j \notin [n]$, and by multiplying the output of the resulting circuit by $\mathrm{sgn}(\rho_0)$. Let F' denote the polynomial computed by C'. Then, we have

$$F'(X) = \sum_{\pi \in S_n} m'_{\rho(\pi)}$$

where $m'_{\rho(\pi)}$ denotes the monomial obtained from $m_{\rho(\pi)}$ after the substitution. It can be checked that for any $\pi \in S_n$, the monomial $m'_{\rho(\pi)} = y_{1,\pi(1)} y_{2,\pi(2)} \cdots y_{n,\pi(n)}$. Hence, the polynomial F' computed by C' in indeed $\mathrm{Cperm}_n(Y)$. It is easily seen that the size of C' is $\mathrm{poly}(s, n)$. □

We now show that evaluating the polynomial Cdet_{2n} over $M_S(\mathbb{F})$, for $S = c \cdot n^2$ for suitable $c > 0$, is at least as hard as evaluating the permanent over $\mathbb{F}$.

Theorem 5.4.6. *If there is a polynomial-time algorithm $\mathcal{A}$ that computes the $2n \times 2n$ Cayley determinant of matrices with entries in $M_S(\mathbb{F})$, for $S = c \cdot n^2$ for suitable $c > 0$, then there is a polynomial-time algorithm that computes the $n \times n$ permanent over $\mathbb{F}$.*

Proof. This is an easy consequence of Corollary 5.3.5. Consider the algorithm given by Corollary 5.3.5 for computing $\mathrm{Cdet}_{2n} \circ F$ over the field $\mathbb{F}$, where the ABP in Corollary 5.3.5 is the ABP of Lemma 5.4.3 computing F.

In order to evaluate the permanent over inputs $a_{ij}, 1 \leq i, j \leq n$ we will substitute $x_{2i-1,j} = a_{ij}$ for $1 \leq i, j \leq n$ and we put $x_{i,j} = 1$ when i is even or $j > n$. As in the proof of Theorem 5.4.5 it follows that for this substitution the algorithm computing $\mathrm{Cdet}_{2n} \circ F$ will output $\mathrm{sgn}(\rho_0)\mathrm{Cperm}_n(a_{11}, \ldots, a_{nn})$. Since $\mathrm{sgn}(\rho_0)$ can be easily computed, we have a polynomial-time algorithm for computing the $n \times n$ permanent over $\mathbb{F}$. □

Remark 5.4.7. *The above result has a stronger consequence: for any fixed $\varepsilon > 0$, if there is a polynomial-time algorithm that computes the $m \times m$ Cayley determinant over $M_{m^\varepsilon}(\mathbb{F})$, then there is a polynomial-time algorithm that computes $\Omega(m^{\varepsilon/2}) \times \Omega(m^{\varepsilon/2})$ permanents over $\mathbb{F}$, hence implying that permanent over $\mathbb{F}$ is polynomial-time computable.*

5.5 The Cayley determinant over Clifford algebras

We now consider the complexity of computing the determinant over real Clifford algebras of polynomially large dimension. We show via a polynomial-time reduction that computing the permanent over rationals is reducible to this problem. Indeed, by inspecting our result we can observe that even *approximating* the determinant over such Clifford algebras would yield similar approximation algorithms for the permanent over the reals.

We first define the basic notions in the theory of Clifford algebras. A thorough treatment can be found in [LS09]. Fix $m \in \mathbb{N}$. The (real) *Clifford algebra* CL'_m is a 2^m-dimensional vector space over $\mathbb{R}$ with basis elements of the form $e_{i_1} e_{i_2} e_{i_3} \cdots e_{i_k}$ where $i_1 < i_2 < i_3 \cdots < i_k$ are elements from $[m]$. Multiplication between elements of the basis is defined by the following rules: $e_i^2 = 1$ and $e_i e_j = -e_j e_i$ for distinct $i, j \in [m]$; this is extended linearly to all pairs of elements from the Clifford algebra. Given $i_1 < i_2 < \cdots < i_k$ from $[m]$, we denote by e_S the basis element $e_{i_1} e_{i_2} \cdots e_{i_k}$, where $S = \{i_1, i_2, \ldots, i_k\}$. Each element of the Clifford algebra is uniquely expressible as $\sum_{S \subseteq [m]} c_S e_S$, where $c_S \in \mathbb{R}$ for each S. (Note that $e_\emptyset$ and 1 both refer to the multiplicative identity of the algebra.) An *idempotent* of the Clifford algebra is an element e such that $e^2 = e$. Given $h = \sum_{S \subseteq [m]} c_S e_S$ in CL'_m, we define its *norm* $|h|$ to be $\sqrt{\sum_{S \subseteq [m]} c_S^2}$.

The subset of basis elements $\{e_S \mid S \text{ has even cardinality}\}$ generates a strict subalgebra of CL'_m. We will denote this subalgebra by CL_m. This is the algebra of interest to us. The term 'Clifford algebra' will henceforth refer to CL_m for some $m \in \mathbb{N}$.

Chien, Rasmussen, and Sinclair [CRS03] have shown that a polynomial-time algorithm that, when given as input an $n \times n$ matrix B with entries from CL_m for $m = 2 \log n + 2$, computes $|\mathrm{Cdet}_n(B)|^2$ can be used to design a randomized polynomial time algorithm to approximate

the 0-1 permanent (over $\mathbb{Q}$).

In this section, we prove that if there is a polynomial-time algorithm to compute either $|\mathrm{Cdet}_n(B)|^2$ or $\mathrm{Cdet}_n(B)$, then the permanent (over inputs from $\mathbb{R}$) can actually be computed in polynomial time. For an $n \times n$ real matrix A, let $\mathrm{perm}_n(A)$ denote the permanent of A.

Remark 5.5.1. *In a sense, our result in this section should not be surprising. We have already proved (in Theorem 5.4.6) that computing the determinant over matrix algebras is at least as hard as computing the permanent. Also, it is known that Clifford algebras of polynomial dimension are isomorphic to matrix algebras of polynomial dimension (see, for example, [LS09, Chapter 5]). However, in this section we actually give an explicit polynomial-time reduction showing that computing the permanent over the reals is reducible to computing either $|\mathrm{Cdet}_n(B)|^2$ or $\mathrm{Cdet}_n(B)$ where the entries of B are from the Clifford algebra CL_m.*

Suppose we wish to compute the permanent of an $n \times n$ matrix with entries from $\mathbb{R}$. W.l.o.g., we assume that $n = 2^\ell$ for some $\ell \in \mathbb{N}$. Let m denote 5ℓ. The next lemma is about the existence of certain elements in the algebra CL_m useful for the reduction.

Lemma 5.5.2. *Let n, ℓ, m be as above. Then, there exist $h_1, h_2, \ldots, h_n, h'_1 h'_2, \ldots, h'_n \in CL_m$ and an idempotent $e \in CL_m$ such that:*

- *For all j, $h_j h'_j = e$.*
- *For all $j \neq k$, $h_j h'_k = 0$.*
- $|e|^2 = \frac{1}{2^\ell}$.

Moreover, the elements $h_1, h_2, \ldots, h_n, h'_1, h'_2, \ldots, h'_n$ and e can be constructed in time $\mathrm{poly}(n)$.

We defer the proof of the above lemma and first prove the main result of this section.

Theorem 5.5.3. *Let n, ℓ, m be as above. There is a polynomial-time algorithm which, when given any matrix $A \in M_n(\mathbb{R})$, computes a $B \in M_{2n}(CL_m)$ such that $|\mathrm{Cdet}_{2n}(B)|^2 = \frac{\mathrm{perm}_n(A)^2}{2^\ell}$.*

Proof. The matrix B will be the following: for any odd $i \in [2n]$ and any $j \in [2n]$, set $B(i,j)$ – the (i,j)th entry of B – to be $A(\frac{i+1}{2}, j)h_j$ if $j \leq n$ and 0 if $j > n$; for any even $i \in [2n]$ and $j \in [2n]$, set $B(i,j)$ to be h'_{j-n} if $j > n$ and 0 otherwise. Clearly, B can be computed in polynomial time given A. Note the following property of B: for any odd $i \in [2n]$ and $j, k \in [2n]$

$$B(i,j)B(i+1,k) = \begin{cases} A(\frac{i+1}{2}, j)e & \text{if } j \leq n \text{ and } k = n + j, \\ 0 & \text{otherwise.} \end{cases}$$

Here e denotes the idempotent from Lemma 5.5.2. The following claim is easy to see.

Claim 5.5.4. *For any permutation $\sigma \in S_{2n}$, the product $\prod_{i=1}^{2n} B(i,\sigma(i)) = (\prod_{i=1}^{n} A(i,\pi(i)))e$ if $\sigma = \rho(\pi)$ for some $\pi \in S_n$ and it is 0 otherwise.*

Let us consider $\mathrm{Cdet}_{2n}(B)$. We have:

$$\begin{aligned}\mathrm{Cdet}_{2n}(B) &= \sum_{\sigma \in S_{2n}} \mathrm{sgn}(\sigma) B(1,\sigma(1)) \cdot B(2,\sigma(2)) \cdots B(2n,\sigma(2n)) \\ &= \sum_{\pi \in S_n} \mathrm{sgn}(\rho(\pi))(\prod_{i=1}^{n} A(i,\pi(i)))e \\ &= \mathrm{sgn}(\rho_0)\mathrm{perm}_n(A)e\end{aligned}$$

Thus, we see that $|\mathrm{Cdet}_{2n}(B)|^2 = \mathrm{perm}_n(A)^2 |e|^2 = \frac{\mathrm{perm}_n(A)^2}{2^\ell}$. □

We have the following easy consequence of the above theorem.

Corollary 5.5.5. *Fix any $\varepsilon > 0$, and suppose there is a polynomial-time algorithm that computes $|\mathrm{Cdet}_n(B)|^2$ on input an $n \times n$ matrix B with entries from CL_m for $m = \varepsilon \log n$. Then there is a polynomial-time algorithm that computes the $n \times n$ permanent of matrices with nonnegative rational entries.*

Proof. The statement directly follows from Theorem 5.5.3 for $m = \lceil 5 \log n \rceil$. To prove hardness for $m = \varepsilon \log n$, we note that a polynomial-time algorithm to compute $|\mathrm{Cdet}_n(B)|^2$ over $CL_{\varepsilon \log n}$ can be used to compute $|\mathrm{Cdet}_{n^{\varepsilon/5}}(B)|^2$ over $CL_{5 \log n^{\varepsilon/5}}$ in polynomial time. □

A δ-approximation algorithm $\mathcal{A}$ for a function $f : \Sigma^* \longrightarrow \mathbb{Q}$ is an algorithm such that for each $x \in \Sigma^*$

$$(1-\delta)f(x) \leq \mathcal{A}(x) \leq (1+\delta)f(x).$$

Our reduction from computing the permanent for nonnegative entries to computing $|\mathrm{Cdet}_n(B)|^2$ actually yields an approximation preserving reduction. We formalize this in the next corollary.

Corollary 5.5.6. *Fix any $\delta > 0$ and $\varepsilon > 0$. Suppose there is a polynomial-time δ-approximation algorithm for the function that on input an $n \times n$ matrix B with entries from CL_m for $m = \varepsilon \log n$ takes the value $|\mathrm{Cdet}_n(B)|^2$. Then there is a polynomial-time δ-approximation algorithm for the $n \times n$ permanent with nonnegative rational entries.*

We now prove Lemma 5.5.2.

Proof of Lemma 5.5.2. Let $e_1, e_2, \ldots, e_m$ denote the generators of CL'_m. Partition the set $[m]$ into ℓ subsets of size 5 as follows: set $S_i = \{5(i-1)+j \mid j \in [5]\}$ for each $i \in [\ell]$. For each $i \in [\ell]$, let $S_{i,0} = \{5(i-1)+1, 5(i-1)+2, 5(i-1)+3, 5(i-1)+5\}$ and $S_{i,1} = \{5(i-1)+2, 5(i-1)+3, 5(i-1)+4, 5(i-1)+5\}$.

Using the fact that $e_i^2 = 1$ and $e_i e_j = -e_j e_i$ for $i \neq j$ it easily follows that for any two *disjoint* sets $S, T \subseteq [m]$ such that $|S|, |T|$ are even, we have $e_S e_T = e_T e_S$. Hence, the elements $e_{S_{i,b_1}}$ and $e_{S_{j,b_2}}$ commute for $i \neq j$ and any $b_1, b_2 \in \{0,1\}$. Furthermore, for all $i \in [\ell]$ and $b \in \{0,1\}$ we have $e^2_{S_{i,b}} = 1$. Also, we have $e_{S_{i,0}} e_{S_{i,1}} = -e_{S_{i,1}} e_{S_{i,0}}$. Finally, notice that $e_{S_{i,b}}$ for $1 \leq i \leq \ell$ and $b \in \{0,1\}$ are all elements of CL_m.

For $i \in [\ell]$ and $b \in \{0,1\}$, set $g_{i,0} = \frac{1+e_{S_{i,1}}}{2}$ and $g_{i,1} = \frac{e_{S_{i,0}}(1-e_{S_{i,1}})}{2}$. Also, set $g'_{i,0} = g_{i,0}$ and $g'_{i,1} = \frac{e_{S_{i,0}}(1+e_{S_{i,1}})}{2}$. Notice that $g^2_{i,0} = g_{i,0}$. We also note an additional relation $e_{S_{i,0}}(1-e_{S_{i,1}}) = (1+e_{S_{i,1}})e_{S_{i,0}}$. Using these we can easily derive the following crucial properties of these elements of CL_m.

- For each $i \in [\ell]$ and $b \in \{0,1\}$, $g_{i,b} g'_{i,b} = g_{i,0}$.
- For each $i \in [\ell]$ and $b \in \{0,1\}$, $g_{i,b} g'_{i,1-b} = 0$.
- For $i_1 \neq i_2$ and any $b_1, b_2 \in \{0,1\}$, the elements g_{i_1,b_1} and g'_{i_2,b_2} commute.

Finally, we define h_j, h'_j for a fixed $j \in [n]$. Let $b_1 b_2 \ldots b_\ell$ be the binary representation of the integer $j-1$ (recall that $n = 2^\ell$). We define $h_j = g_{1,b_1} g_{2,b_2} \cdots g_{\ell,b_\ell}$ and $h'_j = g'_{1,b_1} g'_{2,b_2} \cdots g'_{\ell,b_\ell}$. Also, we define e to be $g_{1,0} g_{2,0} \cdots g_{\ell,0}$, which is the same as h_1 and h'_1.

We now prove that the h_j, h'_j ($j \in [n]$) and e satisfy the properties claimed in the statement of the lemma. Fix any $j \in [n]$ and let $b_1 b_2 \ldots b_\ell$ be the binary representation of $j-1$. We have

$$\begin{aligned} h_j h'_j &= g_{1,b_1} g_{2,b_2} \cdots g_{\ell,b_\ell} g'_{1,b_1} g'_{2,b_2} \cdots g'_{\ell,b_\ell} \\ &= (g_{1,b_1} g'_{1,b_1}) \cdot (g_{2,b_1} g'_{2,b_2}) \cdots (g_{\ell,b_\ell} g'_{\ell,b_\ell}) \\ &= g_{1,0} g_{2,0} \cdots g_{\ell,0} = e \end{aligned}$$

The second equality follows from the fact that $g_{i_1,b}$ and $g'_{i_2,b}$ commute for any distinct i_1 and i_2. The third equality follows from the fact that for any i and b, $g_{i,b} g'_{i,b} = g_{i,0}$. This proves the first property claimed in the statement of the lemma. Similarly, we can see that e is an idempotent: $e^2 = h_1^2 = e$.

Fix any distinct $j, k \in [n]$. Let $b_1 b_2 \ldots b_\ell$ and $b'_1 b'_2 \ldots b'_\ell$ be the binary representations of j

and k. Since $j \neq k$, we can fix some i such that $b_i \neq b'_i$. We have

$$\begin{aligned} h_j h'_k &= g_{1,b_1} g_{2,b_2} \cdots g_{\ell,b_\ell} g'_{1,b'_1} g'_{2,b'_2} \cdots g'_{\ell,b'_\ell} \\ &= (g_{1,b_1} g'_{1,b_1}) \cdot (g_{2,b_1} g'_{2,b_2}) \cdots (g_{i,b_i} g'_{i,b'_i}) \cdots (g_{\ell,b_\ell} g'_{\ell,b_\ell}) \\ &= (g_{1,b_1} g'_{1,b_1}) \cdot (g_{2,b_1} g'_{2,b_2}) \cdots 0 \cdots (g_{\ell,b_\ell} g'_{\ell,b_\ell}) = 0 \end{aligned}$$

where the third equality follows from the fact that we have $g_{i,b} g'_{i,1-b} = 0$. This proves the second claim made in the lemma.

Finally, we note that

$$\begin{aligned} |e|^2 = |g_{1,0} g_{2,0} \cdots g_{\ell,0}|^2 &= \left| \frac{1}{2^\ell} \sum_{T \subseteq \ell} \prod_{i \in T} e_{S_{i,1}} \right|^2 \\ = \frac{1}{4^\ell} \left| \sum_{T \subseteq \ell} \prod_{i \in T} e_{S_{i,1}} \right|^2 &= \frac{2^\ell}{4^\ell} = \frac{1}{2^\ell} \end{aligned}$$

It is easily seen from their definitions that the h_j, h'_j and e can be computed in time $\mathrm{poly}(n)$. This completes the proof of the lemma. □

5.6 The Symmetrized Determinant

In this section, we observe that the $2n \times 2n$ symmetrized determinant over $O(n^2)$-dimensional matrix algebras is at least as hard to compute as the permanent. This stands in marked contrast to the result of Barvinok [Bar], who shows that over *constant-dimensional* matrix algebras, the symmetrized determinant is polynomial-time computable.

In this section, let $\mathbb{F}$ denote a field of characteristic 0. Let $X = \{x_{ij} \mid 1 \leq i, j \leq 2n\}$ and $Y = \{y_{ij} \mid 1 \leq i, j \leq n\}$. Recall that for $\sigma, \tau \in S_{2n}$, the monomial $m_{\sigma,\tau}$ is $x_{\sigma(1),\tau(1)} x_{\sigma(2),\tau(2)} \cdots x_{\sigma(2n),\tau(2n)}$, and the monomial m_σ is $x_{1,\sigma(1)} x_{2,\sigma(2)} \cdots x_{2n,\sigma(2n)}$.

Theorem 5.6.1. *If the* $\mathrm{sdet}_{2n}(X)$ *polynomial over* $\mathbb{F}$ *can be computed by a polynomial-sized noncommutative arithmetic circuit, then the polynomial* $\mathrm{Cperm}_n(Y)$ *can also be computed by a polynomial-sized noncommutative arithmetic circuit.*

Proof. Assume that $\mathrm{sdet}_{2n}(X)$ is computed by a circuit C of size s. As in Theorem 5.4.5, we will proceed by taking Hadamard product. Let P be the ABP defined in Lemma 5.4.3 and $F(X)$ the polynomial it computes. Let F'' denote the polynomial $\mathrm{sdet}_{2n}(X) \circ F$. Note that by Corollary 5.3.4, F'' can be computed by a circuit C'' of size $\mathrm{poly}(s, n)$. From Lemma 5.4.3, we have $F(m_{\sigma,\tau}) = 0$ unless $\sigma = 1_{2n}$, the identity permutation in S_{2n}; moreover, we

also have $F(m_{1_{2n},\tau}) = F(m_\tau)$ which is 1 if $\tau = \rho(\pi)$ for some $\pi \in S_n$ and 0 otherwise. By the above reasoning,

$$F''(X) = \frac{1}{(2n)!}\sum_{\pi\in S_n}\mathrm{sgn}(\rho(\pi))m_{\rho(\pi)} = \frac{\mathrm{sgn}(\rho_0)}{(2n)!}\sum_{\pi\in S_n} m_{\rho(\pi)}$$

Now, we substitute each x_{ij} by $y_{\frac{1+i}{2},j}$ if i is odd and $j \in [n]$ and by 1 if i is even or $j \notin [n]$ in the circuit C''. The effect of this substitution is to transform $m_{\rho(\pi)}$ into $y_{1,\pi(1)}y_{2,\pi(2)}\cdots y_{n,\pi(n)}$ for each $\pi \in S_n$. Hence, the resulting polynomial is simply $\frac{\mathrm{sgn}(\rho_0)\mathrm{Cperm}_n(Y)}{(2n)!}$. Thus, by multiplying by $\mathrm{sgn}(\rho_0)(2n)!$, we obtain a circuit C' of size $\mathrm{poly}(s,n)$ that computes $\mathrm{Cperm}_n(Y)$. □

Theorem 5.6.2. *If there is a polynomial-time algorithm $\mathcal{A}$ that computes the $2n \times 2n$ symmetrized determinant of matrices with entries in $M_S(\mathbb{F})$, for $S = c \cdot n^2$ for suitable $c > 0$, then there is a polynomial-time algorithm that computes the $n \times n$ permanent over $\mathbb{F}$.*

Proof. The proof is almost exactly identical to that of Theorem 5.4.6. Consider the algorithm given by Corollary 5.3.5 for computing $\mathrm{sdet}_{2n} \circ F$ over the field $\mathbb{F}$, where the ABP in Corollary 5.3.5 is the ABP of Lemma 5.4.3 computing F.

In order to evaluate the permanent over inputs $a_{ij}, 1 \le i,j \le n$ we will substitute $x_{2i-1,j} = a_{ij}$ for $1 \le i,j \le n$ and we put $x_{i,j} = 1$ when i is even or $j > n$. As in the proof of Theorem 5.6.1, it follows that for this substitution the algorithm computing $\mathrm{sdet}_{2n} \circ F$ will output $\frac{\mathrm{sgn}(\rho_0)}{(2n)!}\mathrm{Cperm}_n(a_{11},\ldots,a_{nn})$. Since $\mathrm{sgn}(\rho_0)$ and $(2n)!$ are easily computable, we have a polynomial-time algorithm for computing the $n \times n$ permanent over $\mathbb{F}$. □

5.7 The Moore determinant

We demonstrate by a simple reduction that the Moore determinant and permanent are interreducible. We also show that the computing the Moore determinant over a field of characteristic zero is at least as hard as counting the number of directed Hamilton Cycles of a directed graph, which is a well-known #P-complete problem. If the field is of characteristic k, then computing the Moore determinant over the field is at least as hard as counting the number of Hamilton cycles of a directed graph modulo the prime k, which is hard for $\mathrm{Mod}_k\mathrm{P}$.

Assume $X = \{x_{ij} \mid 1 \le i,j \le n\}$. Given a permutation $\sigma \in S_n$, we write σ as a product of disjoint cycles as follows: $(n^\sigma_{11}\cdots n^\sigma_{1l_1})(n^\sigma_{21}\cdots n^\sigma_{2l_2})\cdots(n^\sigma_{r1}\cdots n^\sigma_{rl_r})$ with $n^\sigma_{i1} < n^\sigma_{ij}$ for all $i \in [r]$ and $j \in [l_r] \setminus \{1\}$ and satisfying $n^\sigma_{11} > n^\sigma_{21} > \cdots > n^\sigma_{r1}$. Let w_σ denote the monomial $x_{n^\sigma_{11},n^\sigma_{12}}\cdots x_{n^\sigma_{1l_r},n^\sigma_{11}}\cdots x_{n^\sigma_{r1},n^\sigma_{r2}}\cdots x_{n^\sigma_{rl_r},n^\sigma_{r1}}$.

Let C_n denote the set of all 1-cycles in S_n, i.e permutations whose cycle decomposition consists of a single cycle of length n. Define the polynomial $\mathrm{HC}_n(x_{11},\ldots,x_{nn}) \in \mathbb{F}\langle X\rangle$ to be $\sum_{\sigma\in C_n} w_\sigma$. Fix any directed graph G on n vertices with adjacency matrix A. Let $H(G)$

denote $\mathrm{HC}_n(A(1,1),\ldots,A(n,n))$. The quantity $H(G)$ has a simple description: if $\mathbb{F}$ is of characteristic 0, then $H(G)$ is the number of directed Hamiltonian cycles in G; and if $\mathbb{F}$ is of characteristic k, then $H(G)$ is the number of directed Hamiltonian cycles of G modulo k.

We have the following easy lemma:

Lemma 5.7.1. *There are ABPs P_1' and P_2' of size $O(n^2)$ and width n that compute homogeneous polynomials $F_1', F_2' \in \mathbb{F}\langle X\rangle$ of degree n such that for any $\sigma \in S_n$, we have*

- $F_1'(w_\sigma) = \mathrm{sgn}(\sigma)$.
- $F_2'(w_\sigma) = \mathrm{sgn}(\sigma)$ *if* $\sigma \in C_n$ *and* 0 *otherwise.*

Moreover, the above ABPs can be computed in time $\mathrm{poly}(n)$.

Proof. Recall that given a permutation $\sigma \in S_n$, the quantity $\mathrm{sgn}(\sigma)$ is $(-1)^{n+c_\sigma}$, where c_σ is the number of disjoint cycles in σ. Moreover, note that if σ as a product of disjoint cycles is $(n^\sigma_{11}\cdots n^\sigma_{1l_1})(n^\sigma_{21}\cdots n^\sigma_{2l_2})\cdots(n^\sigma_{r1}\cdots n^\sigma_{rl_r})$ as above, the value c_σ is simply the number of *left-to-right minima* in this representation, i.e the number of n^σ_{ij} such that $n^\sigma_{ij} < n^\sigma_{kl}$ for all n^σ_{kl} to the *left* of n^σ_{ij}. Using this observation, it is easy to design an ABP P_1' that keeps track of the sign of the permutation and computes a polynomial F_1' as above. The ABP P_2' can be constructed similarly; the main difference from the case of P_1' is that the ABP must produce the coefficient 0 unless $n^\sigma_{11} = 1$. We omit the formal description of P_1' and P_2'. □

The analogue of Theorem 5.4.5 for the Moore determinant follows below. The statement here is stronger: we show that the arithmetic circuit complexity of $\mathrm{Mdet}_n(X)$ is polynomial *if and only if* the arithmetic circuit complexity of $\mathrm{Mperm}_n(X)$ is polynomial.

Theorem 5.7.2. *The Moore determinant polynomial* $\mathrm{Mdet}_n(X)$ *can be computed by a polynomial-sized noncommutative arithmetic circuit if and only if the Moore permanent polynomial* $\mathrm{Mperm}_n(X)$ *can be computed by a polynomial-sized noncommutative arithmetic circuit.*

Proof. As in the proof of Theorem 5.4.5, we will use the Hadamard product; this time, it can be used to erase or introduce the signs of the permutations corresponding to each monomial w_σ. Formally, we have $\mathrm{Mperm}_n(X) = \mathrm{Mdet}_n(X) \circ F_1'(X)$ and $\mathrm{Mdet}_n(X) = \mathrm{Mperm}_n(X) \circ F_1'(X)$, where $F_1'(X)$ is the polynomial defined in the statement of Lemma 5.7.1. Hence, if $\mathrm{Mdet}_n(X)$ (resp. $\mathrm{Mperm}_n(X)$) is computed by a noncommutative arithmetic circuit of size s, then by applying Corollary 5.3.4, we see that $\mathrm{Mperm}_n(X)$ (resp. $\mathrm{Mdet}_n(X)$) is computed by a noncommutative arithmetic circuit of size $\mathrm{poly}(s,n)$. □

Remark 5.7.3. *Note that Theorem 5.7.2 proves an equivalence (up to polynomial factors) between the arithmetic circuit complexities of the Moore determinant and permanent. This is a stronger statement than we obtained in the case of the Cayley determinant and permanent, where we only showed (roughly) that the Cayley determinant is at least as hard to compute as the Cayley permanent. The reason for this is that we are unable to obtain a small ABP that performs the function of P_1' for the monomials m_σ (defined in Section 5.4): that is, a small ABP computing a polynomial F_1 such that $F_1(m_\sigma) = \mathrm{sgn}(m_\sigma)$ for every $\sigma \in S_n$. However, we are unable to rule out the possibility that such an ABP exists. If it does, then as above, we can obtain a simple equivalence between the complexities of the Cayley determinant and permanent.*

We now consider the complexity of computing the Moore determinant over matrix algebras of polynomial dimension. We can, as in the previous sections, show that this is at least as hard as computing the permanent over matrices with entries from $\mathbb{F}$, but we take a different route this time. We show that if the Moore determinant over a field of characteristic k can be computed in polynomial time, then there is a polynomial-time algorithm to compute the number of directed Hamilton cycles $H(G)$ modulo k for an input directed graph G. This allows us to draw stronger consequences, namely that the Moore determinant is hard to compute even when the field $\mathbb{F}$ is of characteristic 2, something that would not follow if we reduced the permanent to this problem (since the permanent is polynomial-time computable over fields of characteristic 2).

Theorem 5.7.4. *If there is a polynomial-time algorithm $\mathcal{A}$ that computes the $n \times n$ Moore determinant of matrices with entries in $M_S(\mathbb{F})$, for $S = c \cdot n^2$ for suitable $c > 0$, then there is a polynomial-time algorithm that, on input a directed graph G, computes $H(G)$.*

Proof. Note that $\mathrm{HC}_n(X) = \mathrm{Mdet}_n(X) \circ F_2'$, where F_2' is the polynomial computed by ABP P_2' constructed in Lemma 5.7.1. Moreover, $H(G) = \mathrm{HC}_n(A(1,1), \ldots, A(n,n))$, where A is the adjacency matrix of the graph G. Hence, to compute $H(G)$, we need to compute $\mathrm{HC}_n(A(1,1), \ldots, A(n,n))$, which can be done in polynomial time by Corollary 5.3.5. □

5.8 Completeness Results

In this section we observe that the noncommutative Cayley determinant over integer matrices is complete for GapP w.r.t. polynomial-time Turing reductions. Likewise, the noncommutative Cayley determinant over a finite field of characteristic $k \neq 2$ is hard for the modular counting complexity class $\mathrm{Mod}_k\mathrm{P}$ w.r.t. polynomial-time Turing reductions. These observations also hold for the symmetrized determinant. For the Moore determinant, we prove the above results without any restriction on the characteristic of the underlying field. We formally describe these observations.

Definition 5.8.1. [FFK94],[BG92] *A function* $f : \Sigma^* \longrightarrow \mathbb{Z}$ *is in* GapP *if there is a polynomial time NDTM* M *such that for each* $x \in \Sigma^*$ *the value* $f(x)$ *is* $\mathrm{acc}_M(x) - \mathrm{rej}_M(x)$.

For a prime k, *the class* $\mathrm{Mod}_k\mathrm{P}$ *consist of languages* $L \subseteq \Sigma^*$ *such that for some function* $f \in \mathrm{GapP}$ *we have* $x \in L$ *if and only if* $f(x) \equiv 0 (mod\ k)$.

By Valiant's result [Val79] it is known that the integer permanent is GapP-complete with respect to polynomial-time Turing reductions. Furthermore, the permanent over $\mathbb{F}_k$ is $\mathrm{Mod}_k\mathrm{P}$-hard for prime $k \neq 2$.

Now, for $n \in \mathbb{N}$, consider the Cayley determinant for $2n \times 2n$ matrices with entries from $M_S(\mathbb{Z})$, where $S = cn^2$ for some constant c. By Theorem 5.4.6, there is a fixed $c > 0$ such that computing the integer permanent for $n \times n$ matrices is polynomial-time reducible to computing the $(1, S)^{th}$ entry of such a Cayley determinant. The same observation holds modulo k for a prime k.

Furthermore, the problem of computing the $(1, S)^{th}$ entry of such a Cayley determinant over $\mathbb{Z}$ is easily seen to be in GapP: we can design a polynomial-time NDTM which takes as input a $2n \times 2n$ matrix with entries from $M_S(\mathbb{Z})$ and the difference in the number of accepting and rejecting paths is the $(1, S)^{th}$ entry of its Cayley determinant. Hence we have the following.

Corollary 5.8.2. *There exists a constant* c *such that the following holds. For* $S = cn^2$, *computing the* $(1, S)^{th}$ *entry of the Cayley determinant for* $2n \times 2n$ *matrices with entries from* $M_S(\mathbb{Z})$ *is* GapP*-complete w.r.t. polynomial-time Turing reductions. Given a finite field* $\mathbb{F}$ *of characteristic* $k \neq 2$, *computing the* $(1, S)^{th}$ *of the Cayley determinant for* $2n \times 2n$ *matrices over* $M_S(\mathbb{F})$ *is hard w.r.t. polynomial-time Turing reductions for* $\mathrm{Mod}_k\mathrm{P}$.

We have similar GapP-completeness and Mod_kP-hardness consequences for the symmetrized determinant from the results in Sections 5.6. For the Moore determinant, by Theorem 5.7.4, we additionally obtain hardness for $\oplus$P over fields of characteristic 2.

Corollary 5.8.3. *There exists a constant* c *such that the following holds. For* $S = cn^2$, *computing the* $(1, S)^{th}$ *entry of the Moore determinant for* $2n \times 2n$ *matrices with entries from* $M_S(\mathbb{Z})$ *is* GapP*-complete w.r.t. polynomial-time Turing reductions. Given a finite field* $\mathbb{F}$ *of any characteristic* $k > 1$, *computing the* $(1, S)^{th}$ *of the Moore determinant for* $2n \times 2n$ *matrices over* $M_S(\mathbb{F})$ *is hard w.r.t. polynomial-time Turing reductions for* $\mathrm{Mod}_k\mathrm{P}$.

Proof. The result follows from Theorem 5.7.4 and the following observations: computing $H(G)$ over the rationals on an input graph G is GapP-complete w.r.t. polynomial-time Turing reductions; similarly, computing $H(G)$ over a field $\mathbb{F}$ of characteristic k (including $k = 2$) is hard for $\mathrm{Mod}_k\mathrm{P}$ w.r.t. polynomial-time Turing reductions. □

5.9 Discussion

Our work raises further interesting questions regarding the complexity of the noncommutative determinant.

An important open question is the complexity of computing the noncommutative determinant over constant dimensional matrix algebras. Theorem 5.4.6 can be easily used to show that assuming that the permanent of an $n \times n$ matrix over $\mathbb{F}$ cannot be computed in subexponential time, the $n \times n$ noncommutative Cayley, symmetrized, and Moore determinants with entries from $M_{(\log n)^{\omega(1)}}(\mathbb{F})$ cannot be computed in polynomial time. Can one strengthen this result to one that says something about computing the Cayley or Moore determinant over matrices with entries from $M_c(\mathbb{F})$ for some absolute constant c? (Recall that the symmetrized determinant, on the other hand, *is* efficiently computable over constant dimensional matrix algebras.) It is interesting to note that [CS07] have shown an exponential lower bound for the ABP complexity of the Cayley determinant over even 2×2 matrices.

Also, note that our results do not imply that the Cayley determinant is hard to compute over $M_k(\mathbb{F})$ when $\mathbb{F}$ is a field of characteristic 2, since the permanent is known to be polynomial-time computable over such fields. On the other hand, we have proved that the Moore determinant over such domains (where k is polynomial) is hard for $\oplus$P. Can we prove an analogous result for the Cayley determinant?

Chapter 6

Conclusions

As we mentioned in the introduction, our aim in this thesis has been to try and add to the body of work that proves lower bounds on restricted models of computation, in the hope that this sheds light on how to tackle more general lower bound questions.

In Chapter 2, we considered the *Help functions problem*, where we study the power of a constant-depth circuit that had access to the value of a few possibly hard to compute functions on the input. Proving explicit lower bounds for this model implies lower bounds against the closure of AC^0 under certain reductions. More precisely, a polynomial-time solution to the help functions problem implies that EXP does not reduce to AC^0 in polynomial time, and NC solution implies that PSPACE does not reduce to AC^0 in logspace.

We provided an interesting connection between the Help functions problem and the Remote Point Problem, introduced by Alon, Panigrahy, and Yekhanin [APY09]. Additionally, we showed that the parameters achieved by Alon, Panigrahy, and Yekhanin for the Remote Point Problem could also be achieved by an NC^2 algorithm. However, these parameters are not good enough to give us any meaningful solution to the Help functions problem.

Motivated by the Help functions problem, we studied a variant in the algebraic domain, that we called the *Help Polynomials problem*. We considered the model of Algebraic Branching Programs (ABPs), a model for which Nisan [Nis91] proved lower bounds quite some time ago. We studied the power of these models when augmented with a small collection of arbitrary polynomials. Our results were similar to those obtained in the case of the Help functions problem mentioned above: we provided an approach to solving this question via a connection to a variant of the Remote Point Problem that we called the *Remote Matrix Problem*. In this case, even trivial solutions to this problem gave us good parameters for the Help polynomials problem. However, a good solution to the Remote Matrix Problem continues to evade us: the best parameters we could achieve were only slightly better than that of the trivial solution.

Many open questions remain. We note some of the more important ones below (they are also presented at the end of Chapters 2 and 3).

- Is there a more promising approach to the Help functions problem and Help polynomials problems? For the Help functions problem, we have not been able to get standard techniques like those of Håstad [Hås89] and Smolensky [Smo87] to work, but maybe a solution along these lines is possible.
- Can one solve non-trivial special cases, such as the help functions problem for DNFs?
- Can one improve the state of the art for the Remote Point and Remote Matrix Problems? A concrete question here is the following: is there a polynomial-time deterministic algorithm which, when given a subspace of $\mathbb{F}_2^N$ of dimension $N/2$, computes a vector at distance $\omega(\log N)$ from it?
- Are there other lower bound questions that can be reduced to a Remote Point Problem or a similar algorithmic question?

In Chapter 4, we turned to *Bounded-width arithmetic circuits*. There, we were able to prove explicit lower bounds for such circuits under the fairly strong restriction of *monotonicity*. The main open question is to extend this to the non-monotone case, even for noncommutative width-2 circuits.

- Can one prove an explicit superpolynomial lower bound for noncommutative width-2 circuits? A more concrete question is if the family of polynomials P_2^ℓ defined in Chapter 4 (or indeed P_k^ℓ for any constant $k \geq 2$) can be shown to be hard for width-2 circuits. We believe that this is true.

Note that, by the result of Ben-Or and Cleve [BOC92], width-4 arithmetic circuits are at least as powerful as formulas. However, this is not known to be true for width-2 circuits. In fact, it is unclear if these circuits are even as powerful as depth-3 $\Sigma\Pi\Sigma$-formulas. Thus, the above is a nice problem that may serve as a bridge between current techniques and major unsolved problems in arithmetic circuit complexity.

Finally, in Chapter 5, we considered the question of whether the noncommutative determinant has polynomial-sized (noncommutative) arithmetic circuits. We were able to prove a conditional result, namely, that this cannot happen unless the noncommutative permanent has polynomial-sized arithmetic circuits, and hence the commutative permanent has polynomial-sized (commutative) arithmetic circuits. The major open question in the area of noncommutative arithmetic circuit complexity is to prove an unconditional lower bound. That is,

- Is there an explicit family of noncommutative polynomials that cannot be computed by noncommutative arithmetic circuits of polynomial size?

Some progress has been made towards resolving this question in recent work of Hrubeš, Wigderson, and Yehudayoff [HWY], who provide an approach towards proving such a lower bound by studying the sum-of-squares problem.

Questions also remain regarding the complexity of computing the noncommutative determinant. Our work shows that computing the noncommutative determinant (say the Cayley determinant for concreteness) over matrix algebras of polynomial dimension is as hard as computing the Permanent. Can we prove such a hardness result for computing the determinant over matrix algebras of smaller dimension? As of now, the author is unaware of any algorithmic result for computing the determinant even over 2×2 matrices with rational entries.[1] So, it is possible that computing the determinant over such algebras remains as hard as the permanent. We state this question formally.

- Does the problem of computing the Cayley determinant over 2×2 Matrix algebras over $\mathbb{Q}$ have a polynomial-time deterministic algorithm?

[1] We note that such a result is known for the *Symmetrized Determinant*, considered in Chapter 5.

Bibliography

[AB01] Noga Alon and Richard Beigel. Lower bounds for approximations by low degree polynomials over $\mathbb{Z}_m$. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity*, pages 184–187, 2001.

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992.

[AJ09] Vikraman Arvind and Pushkar S. Joglekar. Arithmetic circuits, monomial algebras and finite automata. In *Proceedings of the 34th International Symposium on the Mathematical Foundations of Computer Science*, pages 78–89, 2009.

[AJS09a] Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. Arithmetic circuits and the Hadamard product of polynomials. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 25–36, 2009.

[AJS09b] Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. On Lower Bounds for Constant Width Arithmetic circuits. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, pages 637–646, 2009.

[Ajt83] Miklòs Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[AM08] Vikraman Arvind and Partha Mukhopadhyay. Derandomizing the Isolation lemma and lower bounds for circuit size. In *Proceedings of 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 12th International Workshop on Randomization and Computation*, pages 276–289, 2008.

[AMN98] Yossi Azar, Rajeev Motwani, and Joseph Naor. Approximating probability distributions using small sample spaces. *Combinatorica*, 18(2):151–171, 1998.

[AMS08] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on noncommutative and commutative polynomial identity testing. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, pages 268–279, 2008.

[APY09] Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the Nearest Codeword Problem. In *Proceedings of 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 13th International Workshop on Randomization and Computation*, pages 339–351, 2009.

[AR94] Noga Alon and Yuval Roichman. Random cayley graphs and expanders. *Random Structures and Algorithms*, 5(2):271–285, 1994.

[AS] Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. In *STOC 2010, To Appear*.

[AS10a] Vikraman Arvind and Srikanth Srinivasan. Circuit Lower Bounds, Help Functions, and the Remote Point Problem. In *Proceedings of the First Symposium on Innovations in Computer Science*, pages 383–397, 2010.

[AS10b] Vikraman Arvind and Srikanth Srinivasan. The Remote Point Problem, Small Bias Spaces, and Expanding Generator Sets. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, pages 59–70, 2010.

[Asl96] Helmer Aslaksen. Quaternionic determinants. *Mathematical Intelligencer*, 18(3):57–65, 1996.

[Bar] Alexander Barvinok. New permanent estimators via non-commutative determinants. preprint available from `http://www.math.lsa.umich.edu/~barvinok/papers.html`.

[Ber84] Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147–150, 1984.

[BG92] Richard Beigel and John Gill. Counting classes: Thresholds, parity, mods, and fewness. *Theoretical Computer Science*, 103(1):3–23, 1992.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

[BM99] Richard Beigel and Alexis Maciel. Circuit lower bounds collapse relativized complexity classes. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, pages 222–226, 1999.

[BOC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992.

[Bor77] Allan Borodin. On relating time and space to size and depth. *SIAM Journal on Computing*, 6(4):733–744, 1977.

[Bou05] Jean Bourgain. Estimation of certain exponential sums arising in complexity theory. *Comptes Rendus Mathematique*, 340(9):627 – 631, 2005.

[BRS95] Richard Beigel, Nick Reingold, and Daniel A. Spielman. PP is closed under intersection. *J. Comput. Syst. Sci.*, 50(2):191–202, 1995.

[BS83] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, 1983.

[BS99] David A. Mix Barrington and Howard Straubing. Lower bounds for modular counting by circuits with modular gates. *Computational Complexity*, 8(3):258–272, 1999.

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

[Cai90] Jin-Yi Cai. Lower bounds for constant-depth circuits in the presence of help bits. *Information Processing Letters*, 36(2):79–83, 1990.

[CH05] Arkadev Chattopadhyay and Kristoffer Arnsfelt Hansen. Lower bounds for circuits with few modular and symmetric gates. In *Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming*, pages 994–1005, 2005.

[CH10] Steve Chien and Prahladh Harsha. personal communication, 2010.

[Chi85] Alexander L. Chistov. Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic. In *Proceedings of Fundamentals of Computation Theory*, pages 63–69, London, UK, 1985. Springer-Verlag.

[CRS03] Steve Chien, Lars Eilstrup Rasmussen, and Alistair Sinclair. Clifford algebras and approximating the permanent. *Journal of Computer and System Sciences*, 67(2):263–290, 2003.

[CS07] Steve Chien and Alistair Sinclair. Algebras with polynomial identities and computing the determinant. *SIAM Journal on Computing*, 37(1):252–266, 2007.

[CW09] Arkadev Chattopadhyay and Avi Wigderson. Linear systems over composite moduli. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 43–52, 2009.

[FFK94] Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[GG81] Chris D. Godsil and Ivan Gutman. On the matching polynomial of a graph. *Algebraic Methods in Graph Theory, Vol. I, II (Szeged, 1978)*, pages 241–249, 1981.

[GHR92] Mikael Goldmann, Johan Håstad, and Alexander A. Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992.

[GK98] Dima Grigoriev and Marek Karpinski. An exponential lower bound for depth 3 arithmetic circuits. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 577–582, 1998.

[GR00] Dima Grigoriev and Alexander A. Razborov. Exponential lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields. *Applied Algebra in Engineering, Communication, and Computing*, 10(6):465–487, 2000.

[Hås89] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Randomness and Computation*, pages 6–20. JAI Press, 1989.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

[HWY] Pavel Hrubeš, Avi Wigderson, and Amir Yehudayoff. Non-commutative circuits and the sum-of-squares problem. In *STOC 2010, To Appear*.

[IM02] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of $5n - o(n)$ for boolean circuits. In *Proceedings of the 27th International Symposium on the Mathematical Foundations of Computer Science*, pages 353–364, 2002.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 220–229, 1997.

[JR09] Maurice J. Jansen and B. V. Raghavendra Rao. Simulation of arithmetical circuits by branching programs with preservation of constant width and syntactic multilinearity. In *Proceedings of the Fourth International Symposium on Computer Science in Russia*, pages 179–190, 2009.

[JS82] Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *Journal of the ACM*, 29(3):874–897, 1982.

[KKL+93] Narendra Karmarkar, Richard M. Karp, Richard J. Lipton, László Lovász, and Michael Luby. A monte-carlo algorithm for estimating the permanent. *SIAM Journal on Computing*, 22(2):284–293, 1993.

[KP97] Matthias Krause and Pavel Pudlák. On the computational power of depth-2 circuits with threshold and modulo gates. *Theoretical Computer Science*, 174(1-2):137–156, 1997.

[LMR10] Nutan Limaye, Meena Mahajan, and B. V. Raghavendra Rao. Arithmetizing classes around NC^1 and L. *Theory of Computing Systems*, 46(3):499–522, 2010.

[Lok01] Satyanarayana V. Lokam. Spectral methods for matrix rigidity with applications to size-depth trade-offs and communication complexity. *Journal of Computer and System Sciences*, 63(3):449–473, 2001.

[LR01] Oded Lachish and Ran Raz. Explicit lower bound of $4.5n - o(n)$ for boolena circuits. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 399–408, 2001.

[LS09] Douglas Lundholm and Lars Svensson. Clifford algebra, geometric algebra, and applications. Lecture notes available at `http://arxiv.org/abs/0907.5356`, 2009.

[Luk93] Eugene M. Luks. Permutation groups and polynomial-time computation. In Larry Finkelstein and William M. Kantor, editors, *Groups and Computation*, volume 11 of *American Mathematical Society DIMACS Series*, pages 139–175. (DIMACS, 1991), 1993.

[MC87] Pierre McKenzie and Stephen A. Cook. The parallel complexity of abelian permutation group problems. *SIAM Journal of Computing*, 16(5):880–909, 1987.

[MR] Cristopher Moore and Alexander Russell. Approximating the permanent via non-abelian determinants. manuscript, available at `http://arxiv.org/abs/0906.1702`.

[MR09] Meena Mahajan and B. V. Raghavendra Rao. Small-space analogues of valiant's classes. In *Proceedings of Fundamentals of Computation Theory*, pages 250–261, 2009.

[MV97] Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, 1997, 1997.

[MZ09] Raghu Meka and David Zuckerman. Small-bias spaces for group products. In *Proceedings of 12th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 13th International Workshop on Randomization and Computation*, pages 658–672, 2009.

[Nis91] Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 410–418, 1991.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

[NW97] Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997.

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130 – 143, 1988.

[Raz85] Alexander A. Razborov. Lower bounds on the monotone complexity of some boolean functions. *Doklady Academii Nauk SSSR*, 281(4):798–801, 1985.

[Raz87] Alexander A. Razborov. Lower bounds on the size of constant-depth networks over a complete basis with logical addition. *Mathematicheskie Zametki*, 41(4):598–607, 1987.

[Raz06] Ran Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(1):121–135, 2006.

[Raz09] Ran Raz. Multilinear formulas for permanent and determinant are of super-polynomial size. *Journal of the ACM*, 56(2), 2009.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

[RS05] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005.

[RSY08] Ran Raz, Amir Shpilka, and Amir Yehudayoff. A lower bound for the size of syntactically multilinear arithmetic circuits. *SIAM Journal on Computing*, 38(4):1624–1647, 2008.

[RY08] Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008.

[RY09] Ran Raz and Amir Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009.

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 77–82, 1987.

[Sni80] Marc Snir. On the size complexity of monotone formulas. In *Proceedings of the 7th International Colloquium on Automata, Languages, and Programming*, pages 621–631, 1980.

[Str73] V. Strassen. Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. (German) [The computational complexity of elementary symmetric functions and interpolation coefficients]. *Numerische Mathematik*, 20(3):238–251, June 1973.

[SW01] Amir Shpilka and Avi Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001.

[Sze62] G. Szekeres. Fractional iteration of exponentially growing functions. *Journal of the Australian Mathematical Society*, 2(03):301–320, 1962.

[Tar88] Éva Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988.

[Tar93] Jun Tarui. Probablistic polynomials, ac0 functions, and the polynomial-time hierarchy. *Theoretical Computer Science*, 113(1):167–183, 1993.

[Val77] Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *Proceedings of the 6th International Symposium on the Mathematical Foundations of Computer Science*, pages 162–176, 1977.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val80] Leslie G. Valiant. Negation can be exponentially powerful. *Theoretical Computer Science*, 12:303–314, 1980.

[Yao85] Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.